Trochę Historii…

Zanim przejdziemy do C ++, możesz dowiedzieć się trochę o ewolucji języka programowania C++. C++ jest tak głęboko zakorzenione w C, że powinieneś najpierw zobaczyć, gdzie C się rozpoczął. Bell Labs najpierw opracował język programowania C na początku lat 70., głównie dlatego, że programiści Bell musieli napisać swój system operacyjny UNIX dla nowego komputera DEC (Digital Equipment Corporation). Do tego czasu systemy operacyjne były pisane w języku asemblerowym, co jest uciążliwe, czasochłonne i trudne do utrzymania. Ludzie z Bell Labs wiedzieli, że potrzebują języka programowania wyższego poziomu, aby szybciej wdrożyć projekt i stworzyć kod, który był łatwiejszy w utrzymaniu. Ponieważ inne języki wysokiego poziomu w tym czasie (COBOL, FORTRAN, PL / I i Algol) były zbyt wolne dla kodu systemu operacyjnego, programiści Bell Labs zdecydowali się napisać własny język. Oparli swój nowy język na Algolu i BCPL. BCPL miał duży wpływ na C, chociaż nie oferował różnych typów danych wymaganych przez twórców języka C. Po kilku wersjach programiści Bell opracowali język, który dobrze spełniał swoje cele. C jest efektywny (jest czasem nazywany językiem wysokim, niskopoziomowym ze względu na szybkość realizacji), elastyczny i zawiera odpowiednie elementy językowe, które umożliwiają jego utrzymanie w czasie. W latach 80. Bjourn Stroustrup, pracujący dla AT & T, przejął język C do następnego etapu. Pan Stroustrup dodał funkcje, aby zrekompensować niektóre pułapki C i zmienił sposób, w jaki programiści przeglądają programy, dodając do języka orientację obiektową. Rozpoczęto aspekt programowania obiektowego języka Pan Stroustrup zdał sobie sprawę, że programiści C++ potrzebują elastyczności i modułowości oferowanej przez prawdziwy język programowania OOP.

C ++ w porównaniu z innymi językami

Jeśli programowałeś wcześniej, powinieneś trochę wiedzieć o tym, jak C++ różni się od innych języków programowania dostępnych na rynku. C++ jest wydajny i ma znacznie silniejsze typowanie niż jego poprzednik C. C jest znane jako słabo napisany język; zmienne typy danych nie muszą koniecznie zawierać tego samego typu danych. (Funkcjonowanie prototypowania i rzutowanie typów pomagają złagodzić ten problem.). Na przykład, jeśli zadeklarujesz zmienną całkowitą i zdecydujesz się umieścić w niej wartość znaku, C to umożliwi to. Dane mogą nie być w oczekiwanym formacie, ale C robi to, co najlepsze. Różni się to znacznie od języków o silniejszym typie, takich jak COBOL i Pascal. C++ to mały, blokowy język programowania. Ma mniej niż 46 słów kluczowych. Aby zrekompensować swoje małe słownictwo, C++ ma jeden z największych asortymentów operatorów, takich jak +, - i && (drugi tylko dla APL). Duża liczba operatorów w C++ może skusić programistów do pisania tajemniczych programów, które mają tylko niewielką ilość kodu. Jak się jednak przekonasz się, że uczynienie programu bardziej czytelnym jest ważniejsze niż zapisanie niektórych bajtów. Nauczysz się jak korzystać z operatorów C++ w jak największym stopniu, zachowując czytelne programy. Duża liczba operatorów C ++ (prawie równa liczbie słów kluczowych) wymaga bardziej rozważnego korzystania z tabeli priorytetów operatorów. W przeciwieństwie do większości innych języków, które mają tylko cztery lub pięć poziomów pierwszeństwa, C++ ma 15. Gdy uczysz się C +, musisz opanować każdy z tych 15 poziomów. Nie jest to tak trudne, jak się wydaje, ale jego znaczenie nie może być przecenione. C++ również nie ma żadnych instrukcji wejścia ani wyjścia. Możesz przeczytać to zdanie jeszcze raz! C++ nie ma poleceń, które wykonują dane wejściowe lub wyjściowe. Jest to jeden z najważniejszych powodów, dla których C++ jest dostępny na tak wielu różnych komputerach. Instrukcje We / Wy (wejścia / wyjścia) większości języków wiążą te języki z określonym sprzętem. Na przykład BASIC ma prawie dwadzieścia poleceń I / O - niektóre z nich piszą na ekranie, do drukarki, do modemu i tak dalej. Jeśli napiszesz program BASIC dla mikrokomputera, jest duża szansa, że nie może działać na komputerze mainframe bez znacznych modyfikacji. Wprowadzanie i wyprowadzanie C++ odbywa się poprzez obfite korzystanie z operatorów i wywoływanie funkcji. Z każdym kompilatorem C++

przychodzi biblioteka standardowych funkcji wejścia / wyjścia. Funkcje I / O są niezależne od sprzętu, ponieważ działają na dowolnym urządzeniu i na każdym komputerze zgodnym ze standardem AT & T C++. Aby całkowicie opanować C++, musisz być bardziej świadomy swojego sprzętu komputerowego, niż wymagałoby to większości innych języków. Z pewnością nie musisz być specjalistą od sprzętu, ale zrozumienie wewnętrznej reprezentacji danych sprawia, że C++ jest znacznie bardziej użyteczny i znaczący. Pomaga także, jeśli możesz zapoznać się z liczbami binarnymi i szesnastkowymi.

C++ i mikrokomputery

C był stosunkowo nieznanym językiem, dopóki nie został umieszczony na mikrokomputerze. Wraz z wynalezieniem i rozwojem mikrokomputera C stał się światowym językiem komputerowym. C++ rozszerza to zastosowanie na mniejsze komputery. Większość czytelników C++ według przykładu prawdopodobnie pracuje nad opartym na mikrokomputerze systemem C++. Jeśli nie znasz się na komputerach, ta sekcja pomoże ci dowiedzieć się, w jaki sposób zostały opracowane mikrokomputery. W latach siedemdziesiątych NASA stworzyła mikroczip, maleńki opłatek krzemu, który zajmuje przestrzeń mniejszą niż znaczek pocztowy. Komponenty komputerowe zostały umieszczone na tych mikroprocesorach, dlatego komputery wymagały znacznie mniej miejsca niż wcześniej. NASA produkowała te mniejsze komputery w odpowiedzi na ich potrzebę wysyłania rakiet na Księżyc za pomocą komputerów pokładowych. Komputery na Ziemi nie mogły zapewnić dokładności rakiet w ułamku sekundy, ponieważ fale radiowe wędrowały kilka sekund, aby podróżować między Ziemią a Księżycem. Dzięki rozwojowi mikroczipy stały się wystarczająco małe, aby komputery mogły podróżować z rakietą i bezpiecznie obliczyć trajektorię rakiety. Program kosmiczny nie był jedynym beneficjentem miniaturyzacji komputerowej. Ponieważ mikroczipy stały się sercem mikrokomputera, komputery mogły teraz zmieścić się na komputerach. Te mikrokomputery kosztują znacznie mniej niż ich większe odpowiedniki, więc wiele osób zaczęło je kupować. Tak narodził się rynek komputerów domowych i małych firm. Obecnie mikrokomputery są zwykle nazywane komputerami. Wczesne komputery osobiste nie miały pamięci dużych komputerów używanych przez rząd i wielki biznes. Mimo to właściciele komputerów wciąż potrzebowali sposobu na zaprogramowanie tych maszyn. BASIC był pierwszym językiem programowania używanym na komputerach PC. Z biegiem lat wiele innych języków zostało przeniesionych z większych komputerów na PC. Jednak żaden język nie był tak skuteczny, jak C, stając się światowym standardowym językiem programowania. C ++ wydaje się być następnym standardem.

Co to jest program?

W niniejszej sekcji przedstawiono podstawowe pojęcia dotyczące programowania. Zadanie programowania komputerów zostało opisane jako satysfakcjonujące, wymagające, łatwe, trudne, szybkie i wolne. W rzeczywistości jest to połączenie wszystkich tych opisów. Pisanie złożonych programów do rozwiązywania zaawansowanych problemów może być frustrujące i czasochłonne, ale możesz się dobrze bawić po drodze, szczególnie przy bogatym asortymencie funkcji, które oferuje C++. Opisano także koncepcję programowania, od uruchomienia programu do jego uruchomienia na komputerze. Najtrudniejszą częścią programowania jest przełamanie problemu w logiczne kroki, które komputer może wykonać. Zanim skończysz tą część, napiszesz i wykonasz swój pierwszy program w C++. Tu zapoznasz się z

♦ Koncepcją programowania

♦ Danymi wynikowymi programu

♦ Projektowaniem programu

♦ Korzystaniem z edytora

♦ Korzystaniem z kompilatora

♦ Wpisywaniem i uruchamianiem programu C ++

♦ Obsługi błędów

Programy Komputerowe

Zanim będziesz mógł pracować z C++, musisz napisać program w C++. Widziałeś, że słowo program jest używane tu wielokrotnie. Poniższa notatka określa program bardziej formalnie.

NOTKA: Program to lista instrukcji, które nakazują komputerowi wykonanie pewnych czynności.

Pamiętaj, że komputery to tylko maszyny. Nie są sprytne ,w rzeczywistości jest zupełnie odwrotnie! Nic nie robią, dopóki nie otrzymają szczegółowych instrukcji. Procesor tekstu, na przykład, jest programem napisanym przez kogoś - w języku takim jak C ++ - który mówi komputerowi, jak zachować się podczas wpisywania słów. Znasz pojęcie programowania, jeśli kiedykolwiek stosowałeś się do przepisu, który jest "programem" lub listą instrukcji, informujących o tym, jak przygotować danie. Dobry przepis podaje te instrukcje w prawidłowej kolejności i zawiera wystarczający opis, aby można było skutecznie przeprowadzać instrukcje bez zakładania byle czego. Jeśli chcesz, aby Twój komputer pomógł Ci z budżetem, śledził nazwiska i adresy lub obliczał zużycie paliwa, potrzebujesz programu, który powie, jak to zrobić. Możesz dostarczyć ten program na dwa sposoby: kup program napisany przez kogoś innego lub sam napisz program. Samo pisanie programu ma dużą zaletę dla wielu zastosowań : Program robi dokładnie to, co chcesz. Jeśli kupisz już napisany, musisz dostosować do swoich potrzeb. Tu właśnie wkracza C++. Dzięki językowi programowania C++ możesz sprawić, że komputer precyzyjnie wykonuje twoje zadania. Aby przekazać instrukcje programowania w języku C++ do komputera, potrzebny jest edytor i kompilator C++. Edytor jest podobny do edytora tekstu; jest to program, który umożliwia wpisanie programu C++ do pamięci, wprowadzanie zmian (takich jak przenoszenie, kopiowanie, wstawianie i usuwanie tekstu) i trwalsze zapisywanie programu w pliku na dysku. Po użyciu edytora do wpisania programu należy go skompilować przed uruchomieniem. Język programowania C++ nazywany jest językiem kompilowanym. Nie możesz napisać programu w języku C++ i uruchamiać go na swoim komputerze, chyba że masz kompilator C++. Ten kompilator pobiera instrukcje w języku C++ i tłumaczy je na formę, które komputer może odczytać. Kompilator C++ jest

narzędziem używanym przez komputer do zrozumienia instrukcji języka C ++ w programach. Wiele kompilatorów ma wbudowany edytor. Jeśli tak jest, prawdopodobnie twoje programowanie w C ++ jest bardziej zintegrowane. Dla niektórych początkujących programistów proces kompilowania programu przed jego uruchomieniem może wydawać się dodatkowym i bezsensownym krokiem. Jeśli znasz język programowania BASIC, być może nie słyszałeś o kompilatorze lub nie rozumiesz potrzeby takiego. To dlatego, że BASIC (również APL i niektóre wersje innych języków komputerowych) nie jest językiem kompilowanym, ale językiem interpretowanym. Zamiast tłumaczyć cały program na formę czytelną maszynowo (jak kompilator robi w jednym kroku), interpreter tłumaczy każdą instrukcję programu - a następnie wykonuje ją - przed tłumaczeniem następnej. Różnica między nimi jest subtelna, ale wynik nie jest taki: kompilatory generują o wiele wydajniejsze i szybciej działające programy niż robią to interpretery. Ten pozornie dodatkowy krok kompilacji wart jest wysiłku (a przy dzisiejszych kompilatorach nie jest potrzebny dodatkowy wysiłek). Ponieważ komputery są maszynami, które nie myślą, instrukcje napisane w C++ muszą być szczegółowe. Nie można założyć, że Twój komputer rozumie, co zrobić, jeśli niektóre instrukcje nie znajdują się w twoim programie lub jeśli napiszesz instrukcję niezgodną z wymaganiami języka C++. Po napisaniu i skompilowaniu programu w C++ musisz go uruchomić lub wykonać. W przeciwnym razie komputer nie będzie wiedział, że chcesz postępować zgodnie z instrukcjami w programie. Tak jak kucharz musi przestrzegać instrukcji receptury przed przygotowaniem potrawy, tak i twój komputer musi wykonać instrukcje programu, zanim będzie mógł wykonać to, co chcesz. Kiedy uruchamiasz program, mówisz komputerowi, aby wykonał twoje instrukcje.

Program I Jego Dane Wyjściowe

Podczas programowania pamiętaj o różnicy między programem a jego danymi wyjściowymi. Twój program zawiera tylko instrukcje napisane w C++, ale komputer postępuje zgodnie z instrukcjami tylko po uruchomieniu programu. Często widzisz listę programów (czyli instrukcje C++ w programie), a następnie wyniki, które pojawiają się podczas uruchamiania programu. Wyniki są danymi wyjściowymi programu i przechodzą do urządzenia wyjściowego, takiego jak ekran, drukarka lub plik dyskowy.

Projektowanie Programu

Musisz zaplanować swoje programy przed wpisaniem ich do edytora C++. Kiedy budowniczowie budują domy, nie od razu chwytają drewno i narzędzia i zaczynają budować! Najpierw dowiadują się, czego chce właściciel domu, następnie sporządzają plany, zamawiają materiały, zbierają robotników i wreszcie zaczynają budować dom. Najtrudniejszą częścią pisania programu jest podzielenie go w logiczne kroki, które komputer może wykonać. Nauka języka C++ jest wymogiem, ale nie jest to jedyna rzecz do rozważenia. Jest metoda pisania programów, formalna procedura, której powinieneś się nauczyć, co ułatwia pracę programistyczną. Aby napisać program powinieneś:

1. Określ problem, który należy rozwiązać za pomocą komputera.

2. Zaprojektuj dane wyjściowe programu (co powinien zobaczyć użytkownik).

3. Podziel problem na logiczne kroki, aby osiągnąć ten wynik.

4. Napisz program (używając edytora).

5. Skompiluj program.

6. Sprawdź program, aby upewnić się, że działa zgodnie z oczekiwaniami.

Jak widać z tej procedury, pisanie twojego programu następuje pod koniec twojego programowania. Jest to ważne, ponieważ najpierw musisz zaplanować, powiedzieć komputerowi, jak wykonać każde

zadanie. Twój komputer może wykonywać instrukcje tylko krok po kroku. Musisz założyć, że twój komputer nie ma wcześniejszej wiedzy o problemie, więc to od ciebie zależy dostarczenie tej wiedzy, która przecież jest tym, co robi dobry przepis. Byłaby to bezużyteczna recepta na ciasto, gdyby wszystko, co napisano, brzmiało: "Upiec ciasto". Dlaczego? Ponieważ zakłada to zbyt wiele wiedzy ze strony piekarza. Nawet jeśli przepisujesz krok po kroku, należy zachować ostrożność (poprzez planowanie), aby upewnić się, że kroki idą po kolei. Czy nie byłoby głupio kazać piekarzowi wkładać składniki do piekarnika, zanim je zmiesza? Zanim ujrzysz aktualny program, pojawi się proces myślowy wymagany do napisania programu. Cele programu są przedstawione w pierwszej kolejności, a następnie cele są podzielone na logiczne kroki, a na końcu program zostaje napisany. Projektowanie programu z wyprzedzeniem gwarantuje, że cała struktura programu jest dokładniejsza i nie wymaga wprowadzania zmian później. Budowniczy, na przykład, wie, że pokój jest o wiele trudniejszy do dodania po wybudowaniu domu. Jeśli nie zaplanujesz właściwie każdego kroku, stworzenie dłuższego, działającego programu zajmie ci więcej czasu. Zawsze trudniej jest wprowadzać istotne zmiany po napisaniu programu. Planowanie i rozwijanie zgodnie z tymi sześcioma krokami staje się o wiele ważniejsze, gdy piszesz dłuższe i bardziej skomplikowane programy. Teraz czas na uruchomienie C ++, abyś mógł doświadczyć satysfakcji z pisania własnego programu i oglądania go.

Korzystanie Z Edytorów Programu

Instrukcje w twoim programie C ++ są nazywane kodem źródłowym. Wpisujesz kod źródłowy do pamięci komputera, używając edytora programów. Po wpisaniu kodu źródłowego C++ (twój program), powinieneś zapisać go na dysku, zanim skompilujesz i uruchomisz program. Większość kompilatorów C++ oczekuje programów źródłowych C++ przechowywane w plikach o nazwach kończących się na .CPP. Na przykład następujące poprawne nazwy plików dla większości kompilatorów C++:

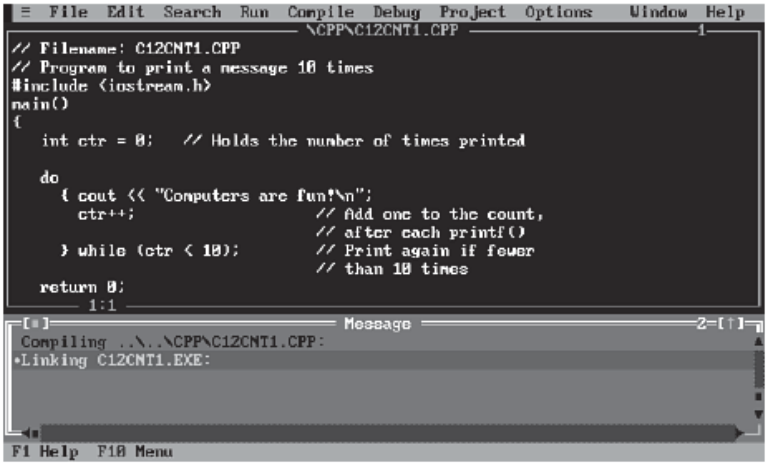
MYPROG.CPP

SALESACT.CPP

EMPLYEE.CPP

ACCREC.CPP

Wiele kompilatorów C ++ zawiera wbudowany edytor. Dwa z najpopularniejszych kompilatorów C ++ to kompilatory Borland C++ i Microsoft C / C++ Te dwa programy działają w całkowicie zintegrowanych środowiskach, które odciążają programistę od martwienia się o znalezienie oddzielnego edytora programów lub uczenie się wielu poleceń specyficznych dla kompilatora. Poniżej mamy ekran Borland C++



W górnej części ekranu znajduje się menu, w którym można edytować, kompilować i uruchamiać opcje rozwijane. Środek ekranu zawiera treść edytora programów i jest to obszar, w którym znajduje się program. Na tym ekranie wpisujesz, edytujesz, kompilujesz i uruchamiasz programy źródłowe C++. Bez zintegrowanego środowiska należałoby uruchomić edytor, napisać program, zapisać program na dysku, wyjść z edytora, uruchomić kompilator, a dopiero potem uruchomić skompilowany program z systemu operacyjnego. Z Borland C++ i Microsoft C/C++ wystarczy wpisać program w edytorze, a następnie - w jednym kroku - wybrać odpowiednią opcję menu, która kompiluje i uruchamia program. Jeśli nie posiadasz zintegrowanego środowiska, takiego jak Borland C++ lub Microsoft C / C++, musisz znaleźć edytor programów. Procesory tekstu mogą działać jako edytory, ale musisz nauczyć się, jak zapisywać i ładować pliki w prawdziwym formacie tekstowym ASCII. Często łatwiej jest korzystać z edytora niż z edytora tekstu. Jeśli używasz komputera innego niż komputer PC, takiego jak minikomputer UNIX lub komputer mainframe, musisz określić, które edytory są dostępne. Większość systemów UNIX zawiera edytor vi. Jeśli programujesz w systemie operacyjnym UNIX, warto poświęcić czas na naukę vi. To jest UNIX, co EDLIN jest na systemy operacyjne dla komputerów PC i jest dostępny na prawie każdym komputerze UNIX na świecie. Użytkownicy mainframe mają dostępne inne edytory, takie jak edytor ISPF. Być może będziesz musiał skontaktować się z działem systemów, aby znaleźć edytor dostępny z twojego konta.

Korzystanie Z Kompilatora C++

Po wpisaniu i edycji kodu źródłowego programu C++ musisz skompilować program. Proces kompilacji programu zależy od wersji C++ i używanego komputera. Użytkownicy Borland C++ i Microsoft C / C++ potrzebują tylko nacisnąć Alt-R, aby skompilować i uruchomić swoje programy. Podczas kompilacji programów na większości komputerów kompilator ostatecznie tworzy plik wykonywalny o nazwie zaczynającej się od tej samej nazwy co kod źródłowy, ale kończy się rozszerzeniem pliku .EXE. Na przykład, jeśli twój program źródłowy ma nazwę GRADEAVG.CPP, komputer wygeneruje skompilowany plik o nazwie GRADEAVG.EXE, który możesz wykonać w monicie DOS, wpisując nazwę gradeavg.

UWAGA. Każdy program w naszym tekście będą zawierać komentarz, który określa zalecaną nazwę pliku dla programu źródłowego. Nie musisz przestrzegać konwencji nazewnictwa plików używanych w tej książce; nazwy plików to tylko sugestie. Jeśli używasz komputera typu mainframe, musisz postępować zgodnie z konwencjami nazewnictwa zestawów danych skonfigurowanymi przez administratora systemu. Każda nazwa programu na przykładowym dysku jest zgodna z nazwami plików wykazów programów. Użytkownicy UNIX mogą musieli użyć kompilatora cfront. Większość kompilatorów cfront faktycznie konwertuje kod C++ do zwykłego kodu C. Kod C jest następnie kompilowany przez kompilator C. Spowoduje to utworzenie pliku wykonywalnego, którego nazwa (domyślnie) to A.OUT. Następnie można uruchomić plik A.OUT z monitu systemu UNIX. Użytkownicy komputerów mainframe mają na ogół standardowe procedury do kompilowania programów źródłowych w języku C++ i przechowywania ich wyników na koncie testowym. W przeciwieństwie do wielu innych języków programowania, twój program C++ musi być poprowadzony przez preprocesor zanim zostanie skompilowany. Preprocesor odczytuje dyrektywy preprocesorów, które wprowadzasz do programu, aby kontrolować kompilację programu. Twój kompilator C ++ automatycznie wykonuje krok preprocesora, więc nie wymaga dodatkowego wysiłku ani poleceń do nauki z twojej strony. Konieczne może być odwołanie się do podręczników referencyjnych kompilatora lub do personelu systemowego firmy, aby dowiedzieć się, jak skompilować programy dla swojego środowiska programistycznego. Ponownie, uczenie się środowiska programistycznego nie jest tak istotne, jak uczenie się języka C ++. Kompilator jest po prostu sposobem na przekształcenie twojego programu z pliku kodu źródłowego do pliku wykonywalnego. Twój program musi przejść jeszcze jeden etap po

kompilacji i przed uruchomieniem. Nazywa się to łączeniem lub etapem edycji linków. Kiedy twój program jest połączony, program zwany linkerem dostarcza potrzebne informacje o czasie pracy do skompilowanego programu. Można również połączyć kilka skompilowanych programów w jeden program wykonywalny, łącząc je. Jednak w większości przypadków twój kompilator inicjuje etap edycji łącza (jest to szczególnie ważne w przypadku zintegrowanych kompilatorów, takich jak Borland C++ i Microsoft C/C++) i nie musisz się martwić o ten proces.

Uruchomienie Programu Przykładowego

Przed zagłębieniem się w specyfikę języka C++ powinieneś poświęcić chwilę na zapoznanie się z edytorem i kompilatorem C++. Począwszy od następnej części , powinieneś skupić całą swoją koncentrację na języku programowania C ++ i nie martwić się o użycie konkretnego edytora lub środowiska kompilującego. W związku z tym uruchom swój edytor wyboru i wpisz Listing 2.1, do twojego komputera. Bądź tak dokładny, jak to tylko możliwe - pojedynczy błąd wpisywania może spowodować, że kompilator C ++ wygeneruje serię błędów. W tym momencie nie musisz rozumieć treści programu; celem jest dać ci praktykę w używaniu twojego edytora i kompilatora.

Listing 2.1 Ćwiczenia Z Edytorem

Skomentuj program z nazwą programu.

Dołącz plik nagłówkowy iostream.h, aby dane wyjściowe działały poprawnie.

Początek funkcji main ().

Zdefiniuj stałą BELL, czyli sygnał dźwiękowy komputera.

Zainicjuj zmienną całkowitą ctr na 0.

Zdefiniuj tablicę znaków fname, aby pomieścić 20 elementów.

Drukuj na ekranie ***Jak masz na imię?***

Zaakceptuj ciąg znaków na klawiaturze.

Przetwórz pętlę, gdy zmienna ctr jest mniejsza niż pięć.

Wydrukuj ciąg zaakceptowany z klawiatury.

Zwiększ wartość zmiennej ctr o 1.

Wydrukuj na ekranie kod znaków, który wywołuje sygnał dźwiękowy.

Wróć do systemu operacyjnego.

```
// Nazwa pliku: C2FIRST.CPP

// Żąda imienia, wypisuje go nazwę pięć razy i wydaje sygnał dźwiękowy.

#include <iostream.h>

main()

{

const char BELL = '\ a'; // Stała, która wywołuje dźwięk

int ctr = 0; // Zmienna całkowita do zliczania pętli
```

```
char fname [20]; // Zdefiniuj tablicę znaków do przechowywania nazwy

cout << "Jak masz na imię? "; // Monituj użytkownika

cin >> fname; // Pobierz nazwę z klawiatury

while (ctr <5) // Pętla, aby wydrukować nazwę

{// dokładnie pięć razy.

cout << fname << "\ n";

ctr ++;

}

cout << BELL; // Zadzwoń

return 0;

}
```

Bądź tak dokładny, jak to możliwe. W większości języków programowania - a zwłaszcza w C++ - znaki wpisywane do programu muszą być bardzo dokładne. W tym przykładowym programie C++ widzimy na przykład nawiasy (), nawiasy, [] i nawiasy klamrowe {}, ale nie można ich używać zamiennie. Komentarze (słowa następujące po dwóch ukośnikach, //) po prawej stronie niektórych wierszy nie muszą kończyć się w tym samym miejscu, które widzisz , wymienianie kolejno. Mogą być tak długie lub krótkie, jak ich potrzebujesz. Powinieneś jednak zapoznać się ze swoim edytorem i dokładnie nauczyć się znaków spacji, abyś mógł wpisać ten program dokładnie tak, jak pokazano. Skompiluj program i uruchom go. Oczywiście po raz pierwszy możesz to zrobić, sprawdzając podręczniki referencyjne lub kontaktując się z kimś, kto zna Twój kompilator C++. Nie martw się o uszkodzenie komputera: Nic, co robisz z klawiatury, nie może zaszkodzić fizycznemu komputerowi. Najgorszą rzeczą, jaką możesz zrobić w tym momencie, jest wymazanie części oprogramowania kompilatora lub zmiana opcji kompilatora - z których wszystkie można łatwo naprawić, ponownie ładując kompilator z oryginalnego źródła.

Obsługa Błędów

Ponieważ piszesz instrukcje dla maszyny, musisz być bardzo dokładny. Jeśli źle przeliterujesz słowo, pominiesz cudzysłów lub popełnisz inny błąd, twój kompilator C++ poinformuje cię o błędzie. W Borland C++ i Microsoft C/C++ błąd prawdopodobnie pojawia się w osobnym oknie. Najczęstszym błędem jest błąd składni, co zwykle oznacza błędnie napisane słowo. Kiedy pojawi się komunikat o błędzie (lub więcej niż jeden), musisz powrócić do edytora programów i poprawić błąd. Jeśli nie rozumiesz błędu, może zajść potrzeba sprawdzenia podręcznika lub wyszukania kodu źródłowego twojego programu, dopóki nie znajdziesz obraźliwego kodu. Po prawidłowym wpisaniu programu za pomocą edytora (i nie otrzymasz żadnych błędów kompilacji), program powinien działać poprawnie, prosząc o imię, a następnie drukując je na ekranie pięć razy. Po tym, jak drukuje twoje imię po raz piąty, usłyszysz dzwonek komputera. Ten przykład pomaga zilustrować różnicę między programem a jego wydajnością. Musisz wpisać program (lub załadować go z dysku), a następnie uruchomić program, aby zobaczyć jego wynik.

Twój Pierwszy Program

W niniejszej części przedstawiono ważne polecenia języka C++ i inne elementy. Przed przyjrzeniem się konkretnemu językowi wielu ludzi lubi "przechodzić" przez kilka prostych programów, aby uzyskać ogólny obraz tego, co obejmuje program w języku C++. Dokonuje się tego tutaj. Reszta obejmuje te polecenia i elementy bardziej formalnie. Ta część wprowadza następujące tematy:

♦ Przegląd programów C++ i ich struktura

♦ Zmienne i literały

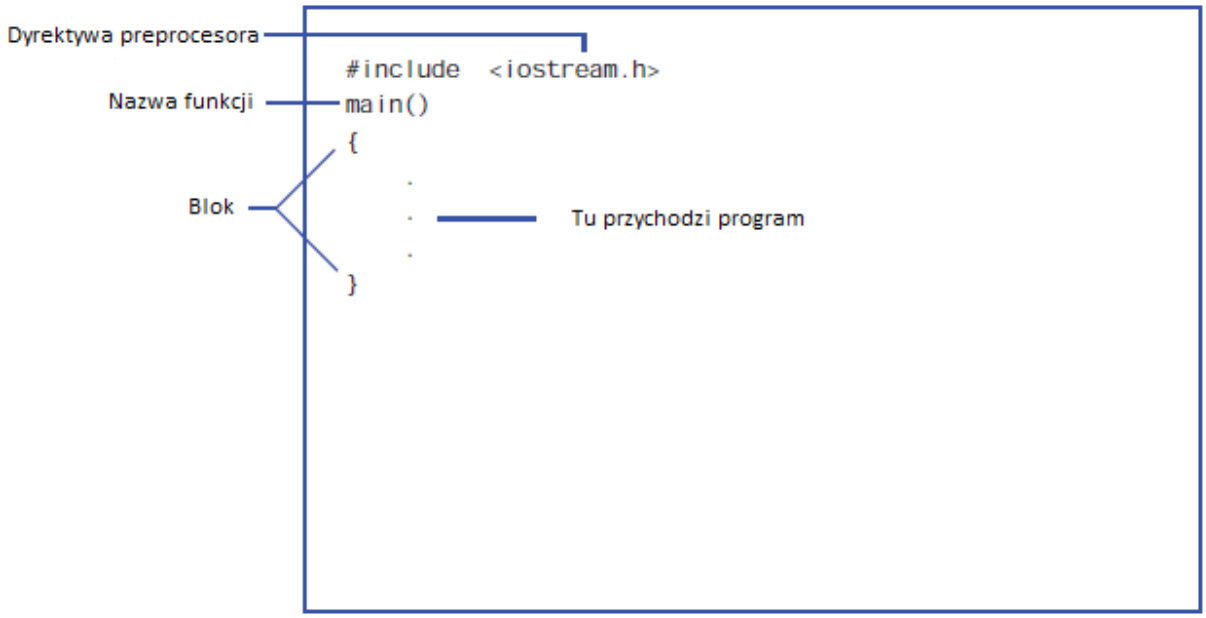
♦ Proste operatory matematyczne

♦ Format wyjściowy ekranu

W niniejszej części przedstawiono kilka ogólnych narzędzi potrzebnych do zapoznania się z językiem programowania C++.

Spójrzmy Na Program C++

Rysunek pokazuje zarys typowego małego programu C++. Na rysunku nie pokazano żadnych poleceń C++. Chociaż program zawiera o wiele więcej, niż sugeruje ten zarys, jest to ogólny format pierwszych przykładów w tym tekście.



Aby jak najszybciej zapoznać się z programami w C++, powinieneś zacząć przeglądać program w całości. Poniżej znajduje się lista prostego przykładowego programu C++. Nie robi wiele, ale pozwala zobaczyć ogólny format programowania w C++. W następnych kilku sekcjach omówiono elementy z tego i innych programów. Możesz nie rozumieć wszystkiego w tym programie, ale jest to dobre miejsce na rozpoczęcie.

// Nazwa pliku: FIRST.CPP

// Początkowy program w C++ demonstrujący komentarze w C++

// i pokazuje kilka zmiennych i ich deklaracje.

```
#include <iostream.h>

main()

{

int i, j; // Te trzy linie deklarują cztery zmienne.

char c;

float x;

i = 4; // i i j są przypisanymi literałami całkowitymi.

j = i + 7;

c = "A"; // Wszystkie literały znaków są

// ujęte w pojedyncze cudzysłowy.

x = 9,087; // x wymaga wartości zmiennoprzecinkowej, ponieważ

// został zadeklarowany jako zmienna zmiennoprzecinkowa.

x = x * 4,5; // Zmień, co było w x z formułą.

// Wysyłanie wartości zmiennych do ekranu.

cout << i << "," << j << "," << c << "," << x << "\ n";

return 0; // ZAWSZE kończ programy i funkcje z return.

// 0  powoduje, że wraca do systemu operacyjnego i

// zwykle wskazuje, że nie wystąpiły błędy.

}
```

Na razie zapoznaj się z tym ogólnym programem. Sprawdź, czy potrafisz zrozumieć jakąkolwiek część lub całość. Jeśli jesteś początkującym programistą, powinieneś wiedzieć, że komputer czyta każdą linię programu, zaczynając od pierwszej linii i kończąc pracę, aż zakończy wszystkie instrukcje w programie. Wyjście tego programu jest minimalne: Po prostu wyświetla cztery wartości na ekranie po wykonaniu niektórych przypisań i obliczeń dowolnych wartości. Po prostu skoncentruj się na ogólnym formacie w tym momencie.

Format Programu C++

W przeciwieństwie do niektórych innych języków programowania, takich jak COBOL, C++ jest językiem swobodnym, co oznacza, że instrukcje programistyczne mogą zaczynać się w dowolnej kolumnie dowolnej linii. Możesz wstawić puste linie w programie, jeśli chcesz. Ten przykładowy program nazywa się FIRST.CPP (możesz znaleźć nazwę każdego programu w tym tekście w pierwszym wierszu każdej listy programów). Zawiera kilka pustych wierszy, aby pomóc w oddzieleniu części programu. W prostym programie takim jak ten, rozdział nie jest tak krytyczny, jak może być w dłuższym, bardziej złożonym programie. Ogólnie rzecz biorąc, spacje w programach C++ są również w formie swobodnej. Twoim celem nie powinno być, aby twoje programy były tak kompaktowe, jak to tylko możliwe. Twoim celem powinno być uczynienie twoich programów możliwie czytelnymi. Na przykład program FIRST.CPP pokazany w poprzedniej sekcji może zostać przepisany w następujący sposób:

```
// Nazwa pliku: FIRST.CPP Początkowy program C++, który demonstruje
// komentarze C++ i pokazuje kilka zmiennych i ich
// deklaracje.
#include <iostream.h>
main() {int i, j; // Te trzy linie deklarują cztery zmienne.
char c; float x; i = 4; // i i są przypisane do liczb całkowitych.
j = i + 7; c = "A"; // Wszystkie literały znaków są zawarte w
// pojedyncze cytaty.
x = 9.087; // x wymaga wartości zmiennoprzecinkowej, ponieważ była
// zadeklarowana jako zmienna zmiennoprzecinkowa.
x = x * 4.5; // Zmień, co było w x z formułą.
// Wysyła wartości zmiennych do ekranu.
cout << i << "," << j << "," << c << "," << x << "\ n"; return 0; // ZAWSZE
// kończ programy i funkcje z return. 0 wraca do
// systemu operacyjnego i zwykle wskazuje, że nie wystąpiły żadne błędy.
}
```

Dla kompilatora C++ dwa programy są dokładnie takie same i dają dokładnie taki sam wynik. Jednak dla osób, które muszą przeczytać program, pierwszy styl jest znacznie bardziej czytelny.

Czytelność Jest Kluczowa

Dopóki programy wykonują swoją pracę i generują poprawne wyniki, kogo to obchodzi, jak dobrze są napisane? Nawet w dzisiejszym świecie szybkich komputerów i obfitej pamięci i miejsca na dysku, powinieneś się tym przejmować. Nawet jeśli nikt nigdy nie patrzy na twój program w C++, być może będziesz musiał go zmienić w późniejszym terminie. Im bardziej czytelny jest twój program, tym szybciej możesz znaleźć to, co wymaga zmian i odpowiednio go zmienić. Jeśli pracujesz jako programista dla korporacji, prawie na pewno możesz spodziewać się modyfikacji kodu źródłowego innej osoby, a inni prawdopodobnie będą modyfikować twoje. W działach programistycznych mówi się, że długoletni pracownicy piszą czytelne programy. Biorąc pod uwagę tę nową globalną gospodarkę i wszystkie zmiany, które napotykają firmy w nadchodzących latach, firmy poszukują programistów, którzy piszą na przyszłość. Programy, które są proste, czytelne, pełne białej przestrzeni (oddzielające linie i przestrzenie) i pozbawione trudnych do odczytania "sztuczek", które tworzą niechlujne programy, są najbardziej pożądane. Użyj wystarczającej ilości białego odstępu, aby mieć oddzielne linie i spacje w twoich programach. Zauważ, że pierwsze kilka wierszy początku FIRST.CPP w pierwszej kolumnie, ale treść programu jest wcięta o kilka spacji. Pomaga to programistom "zerwać" w ważnym kodzie. Kiedy piszesz programy, które zawierają kilka sekcji (nazywanych blokami), twoje użycie białej przestrzeni pomaga obserwatorowi podążać za okiem i rozpoznać następny wcięty blok.

Wielkie Litery A Małe Litery

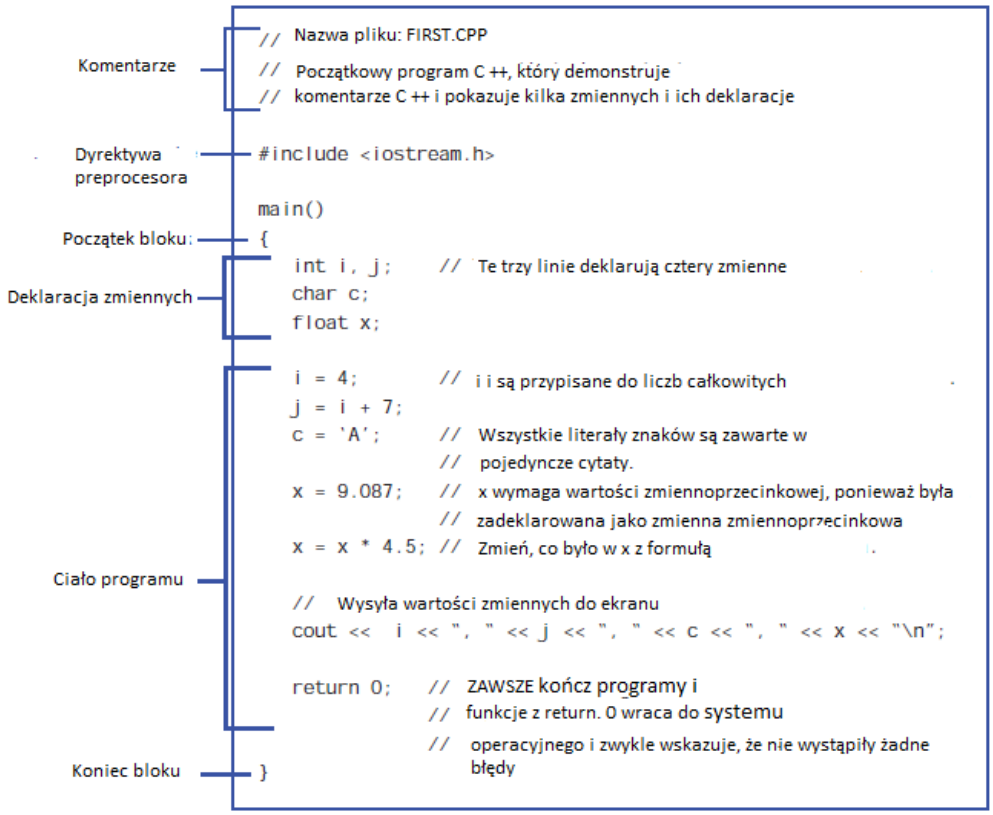
Twoje wielkie i małe litery są znacznie bardziej znaczące w C++ niż w większości innych języków programowania. Możesz zobaczyć, że większość FIRST.CPP jest pisana małymi literami. Cały język C++ jest pisany małymi literami. Na przykład musisz wpisać słowa kluczowe int, char i return w programach zawierając małe litery. Jeśli użyjesz wielkich liter, Twój kompilator C++ będzie generował wiele błędów i nie będzie chciał skompilować programu, dopóki nie poprawisz błędów. Wielu programistów C++ rezerwuje wielkie litery dla niektórych słów i wiadomości wysyłanych na ekran, drukarkę lub plik na dysku; używają małych liter w prawie wszystkim innym.

Nawias Klamrowy I main()

Wszystkie programy w C++ wymagają następującej linii

main()
{

Instrukcje następujące po main() są wykonywane jako pierwsze. Instrukcje następujące po main() są wykonywane jako pierwsze. Sekcja programu C++, która zaczyna się od main(), po której następuje nawias otwierający, {, nazywana jest główną funkcją. Program C++ jest w rzeczywistości zbiorem funkcji (małe fragmenty kodu). Funkcja o nazwie main() jest zawsze wymagana i zawsze wykonywana jest jako pierwsza funkcja. W pokazanym tu programie przykładowym prawie cały program jest main(), ponieważ pasujący nawias zamykający, który następuje po nawiasie otwierającym main(), znajduje się na końcu programu. Wszystko pomiędzy dwoma pasującymi nawiasami nazywa się blokiem. Na razie wystarczy zdać sobie sprawę, że ten przykładowy program zawiera tylko jedną funkcję, main(), a cała funkcja jest pojedynczym blokiem, ponieważ istnieje tylko jedna para nawiasów klamrowych. Wszystkie wykonywalne instrukcje C++ muszą zawierać znak średnika (;), więc C++ ma świadomość, że instrukcja się kończy. Ponieważ komputer ignoruje wszystkie komentarze, nie umieszczaj średników po komentarzach. Zauważ, że linie zawierające main() i nawiasy klamrowe również nie kończą się średnikami, ponieważ te linie po prostu definiują początek i koniec funkcji i nie są wykonywane. Kiedy lepiej poznasz C++, dowiesz się, kiedy uwzględnić średnik i kiedy go opuścić. Wielu początkujących programistów C++ uczy się szybko, gdy potrzebne są średniki; Twój kompilator z pewnością daje ci znać, jeśli zapomnisz wstawić średnik, w którym jest potrzebny. Rysunek 2 powtarza przykładowy program przedstawiony na Rysunku 1. Zawiera dodatkowe oznaczenia, które ułatwią zapoznanie się z nowymi terminami oraz innymi elementami opisanymi w dalszej części tej części . Wszystkie wykonywalne instrukcje C++ muszą kończyć się średnikiem (;).
Rysunek 2



Komentarz w C++

Nauczyłeś się różnicy między programem a jego danymi wyjściowymi. Większość użytkowników programu nie widzi rzeczywistego programu; widzą wyniki z wykonania instrukcji programu. Z drugiej strony, programiści przyglądają się listingom programów, dodają nowe procedury, zmieniają stare i aktualizują pod kątem postępów w sprzęcie komputerowym. Jak wyjaśniono wcześniej, czytelność programu jest ważna, więc ty i inni programiści możecie to łatwo przejrzeć. Niemniej jednak, niezależnie od tego, jak jasno piszesz programy w C++, zawsze możesz zwiększyć ich czytelność, dodając komentarze w całym tekście. Komentarze to wiadomości, które wstawiasz do swoich programów w C++, wyjaśniając, co dzieje się w tym momencie w programie. Na przykład, jeśli napiszesz program listy płac, możesz umieścić komentarz przed procedurą drukowania próbnego, która opisuje, co ma się wydarzyć. Nigdy nie umieszczasz w komentarzu instrukcji języka C++, ponieważ komentarz jest komunikatem dla ludzi - nie komputerów. Twój kompilator C++ ignoruje wszystkie komentarze w każdym programie.

UWAGA: Komentarze C++ zawsze zaczynają się od znaku // i kończą się na końcu linii.

Niektórzy programiści decydują się na komentarz kilku linii. Zauważ w przykładowym programie, FIRST.CPP, że pierwsze trzy linie są liniami komentarza. Komentarze wyjaśniają nazwę pliku i trochę o programie. Komentarze mogą również współdzielić linie z innymi poleceniami C++. Możesz zobaczyć kilka komentarzy udostępniających wiersze poleceniami w programie FIRST.CPP. Wyjaśniają, co robią poszczególne linie. Używaj obfitych komentarzy, ale pamiętaj, komu służą: ludziom, nie komputerom. Użyj komentarzy, aby pomóc objaśnić swój kod, ale nie przesadzaj. Na przykład, nawet jeśli nie znasz się na C++, poniższa instrukcja jest łatwa: wyświetla "C++ jest łatwy" na ekranie.

cout << “C++ jest łatwy”; // Wyświetla C++ jest łatwy na ekranie

Ten komentarz jest zbędny i nie dodaje nic do zrozumienia linii kodu. Byłoby znacznie lepiej, w tym przypadku, pominąć komentarz. Jeśli prawie powtarzasz kod C++, pomiń ten konkretny komentarz. Nie każda linia programu C++ powinna być komentowana. Komentuj tylko wtedy, gdy linie kodu potrzebują wyjaśnienia ludziom, którzy patrzą na Twój program. Nie ma znaczenia, czy w swoich komentarzach używasz wielkich, małych liter, czy kombinacji obu, ponieważ C++ je ignoruje. Większość programistów C++ używa wielkich liter w komentarzach, tak jak w codziennym pisaniu. Użyj dowolnego przypadku odpowiednie dla liter w wiadomości. C++ może również używać komentarzy w stylu C. Są to komentarze rozpoczynające się od / * i kończące się znakiem * /.

Komentuj w trakcie

Wstawiaj swoje komentarze podczas pisania programów. Jesteś najbardziej zaznajomiony z logiką programu w momencie pisania programu w edytorze. Niektórzy odkładają dodawanie komentarzy, dopóki program nie zostanie napisany. Najczęściej jednak te komentarze nigdy nie są dodawane, lub są napisane z pół-serca. Jeśli skomentujesz podczas pisania kodu, możesz spojrzeć na swoje komentarze podczas pracy nad kolejnymi sekcjami programu - zamiast odszyfrowywania poprzedniego kodu. Pomaga ci to, gdy chcesz wyszukać coś wcześniej w programie.

PRZYKŁADY

1. Załóżmy, że chcesz napisać program w języku C++, który tworzy fantazyjny tytuł w polu zawierający twoje imię z migającymi kropkami wokół niego (jak marquee). Kod C++ do tego może być trudny do zrozumienia. Przed takim kodem możesz chcieć wstaw następujący komentarz, aby inni mogli zrozumieć kod później:

```
// Następne kilka linii narysuje fantazyjne pole
// wokół nazwy , następnie wyświetla migające kropki wokół
// nazwy jak hollywoodzki marquee
```

Nie oznacza to, że C++ robi cokolwiek, ponieważ komentarz nie jest poleceniem, ale sprawiłoby, że następne kilka linii kodu byłoby bardziej zrozumiałe dla ciebie i innych. Komentarz wyjaśnia w języku angielskim, dla osób czytających program, dokładnie to, co program przygotowuje.

2. Powinieneś również umieścić nazwę pliku dysku programu w jednym z pierwszych komentarzy. Na przykład w przedstawionym wcześniej programie FIRST.CPP pierwszy wiersz jest początkiem komentarza:

```
// Nazwa pliku: FIRST.CPP
```

Komentarz jest pierwszym z trzech wierszy, ale ten wiersz informuje, w którym pliku dysku znajduje się program. W całym tekście, programy mają komentarze, które zawierają możliwą nazwę pliku, pod którą program może być przechowywany.

WSKAZÓWKA: Dobrym pomysłem może być umieszczenie twojego nazwiska na górze programu w komentarzu. Jeśli ludzie będą musieli zmodyfikować program w późniejszym terminie, najpierw będą chcieli skonsultować się z Tobą, jako oryginalnym programistą, zanim go zmienią.

Wyjaśnienie Przykładowego Programu

Po zapoznaniu się z programem C++, jego strukturą i komentarzami, reszta sekcji przechodzi przez cały przykładowy program. Nie oczekuj, że staniesz się ekspertem C++, wypełniając tę sekcję! Na razie wystarczy usiąść i postępować zgodnie z opisem krok po kroku kodu programu. Jak opisano wcześniej, ten przykładowy program zawiera kilka komentarzy. Pierwsze trzy linie programu to komentarze:

// Nazwa pliku: FIRST.CPP Początkowy program C++, który demonstruje
// komentarze C++ i pokazuje kilka zmiennych i ich
// deklaracje.

Ten komentarz wyświetla nazwę pliku i wyjaśnia cel programu. To nie jedyny komentarz w programie; inne pojawiają się w całym kodzie. Następny wiersz zaczynający się od #include jest nazywany dyrektywą preprocesora i jest pokazany tutaj:
#include <iostream.h>
Ta dziwnie wyglądająca instrukcja nie jest w rzeczywistości poleceniem C++, ale jest instrukcją, która nakazuje kompilatorowi C++ załadowanie pliku z dysku na środek bieżącego programu. Jedynym celem tej dyskusji jest zapewnienie prawidłowego działania wyjścia generowanego przy pomocy cout. Następne dwa wiersze (następujące po pustej linii oddzielającej) to pokazano tutaj:

main( )
{
Rozpoczyna się funkcja main(). Zasadniczo, nawiasy otwierające i zamykające funkcji main() zamykają treść tego programu i instrukcji main(), które wykonują. Programy w C++ często zawierają więcej niż jedną funkcję, ale zawsze zawierają funkcję o nazwie main(). Funkcja main() nie musi być pierwszą, ale zwykle jest. Klamra otwierająca rozpoczyna pierwszy i jedyny blok tego programu. Kiedy programista kompiluje i uruchamia ten program, komputer szuka main() i zaczyna wykonywanie jakiejkolwiek instrukcji zgodnie z nawiasem otwierającym main(). Oto trzy następujące linie:

int i, j; // Te trzy linie deklarują cztery zmienne.

char c;

float x;

Te trzy linie deklarują zmienne. Deklaracja zmiennych opisuje zmienne używane w bloku kodu. Deklaracje zmiennych opisują przechowywanie danych programu. Program C++ przetwarza dane w znaczące wyniki. Wszystkie programy w C++ zawierają :
♦ Polecenia
♦ Dane
Dane zawierają zmienne i literały (czasami nazywane stałymi). Jak sama nazwa wskazuje, zmienną są dane, które mogą się zmieniać (stają się zmiennymi) w trakcie działania programu. Literał pozostaje taki sam. W życiu zmienną może być twoja pensja. Z czasem rośnie (jeśli masz szczęście). Dosłowne byłoby twoje imię lub numer ubezpieczenia społecznego, ponieważ każdy z nich pozostaje z tobą przez całe życie i nie zmienia się (naturalnie). Jednakże, aby uzyskać przegląd elementów przykładowego programu, poniższa dyskusja wyjaśnia zmienne i literały w tym programie. C++ umożliwia korzystanie z kilku rodzajów literałów. Na razie musisz po prostu zrozumieć, że literałem C++ jest dowolna liczba, znak, słowo lub fraza. Oto wszystkie poprawne literały C++:
5.6
-45
‘Q’
“Mary”
18.67643
0.0
Jak widać, niektóre literały mają charakter numeryczny, a niektóre są oparte na znakach. Pojedyncze i podwójne cudzysłowy wokół dwóch literałów nie są jednak częścią rzeczywistych literałów. Literał

jednoliterowy wymaga wokół niego pojedynczych cudzysłowów; zaś ciąg znaków, taki jak "Mary", wymaga podwójnego cudzysłowu. Poszukaj literałów w przykładowym programie. Znajdziesz te:
4
7
‘A’
9.087
4.5
Zmienna jest jak pudełko wewnątrz twojego komputera, które coś posiada. To "coś" może być liczbą lub postacią. Możesz mieć tyle zmiennych, ile potrzeba do przechowywania zmieniających się danych. Po zdefiniowaniu zmiennej zachowuje ona swoją wartość, dopóki jej nie zmienisz lub nie zdefiniujesz ponownie. Zmienne mają nazwy, dzięki czemu można je odróżnić. Korzystasz z operator przypisania, znak równości (=), aby przypisać wartości do zmiennych. Poniższe zdanie,
sales = 25000;
umieszcza literalną wartość 25000 w zmiennej o nazwie sales. W przykładowym programie znajdują się następujące zmienne:
i
j
c
x
Trzy linie kodu, które następują po nawiasie otwierającym programu przykładowego, deklarują te zmienne. Ta deklaracja zmiennej informuje resztę programu, że dwie liczby całkowite o nazwach i i j, a także zmienna znakowa o nazwie c oraz zmienna zmiennoprzecinkowa o nazwie x pojawią się w całym programie. Terminy całkowite i zmiennoprzecinkowe odnoszą się zasadniczo do dwóch różnych typów liczb: liczby całkowite są liczbami całkowitymi, a liczby zmiennoprzecinkowe zawierają liczby dziesiętne. Następne kilka instrukcji przykładowego programu przypisuje wartości do tych zmiennych.

```
i = 4; // i i j są przypisanymi literałami całkowitymi.

j = i + 7;

c = "A"; // Wszystkie literały znaków są

// ujęte w pojedyncze cudzysłowy.

x = 9,087; // x wymaga wartości zmiennoprzecinkowej, ponieważ

// został zadeklarowany jako zmienna zmiennoprzecinkowa.

x = x * 4,5; // Zmień, co było w x z formułą.
```

Pierwsza linia umieszcza 4 w zmiennej całkowitej i. Druga linia dodaje 7 do wartości zmiennej i, aby uzyskać 11, która następnie jest przypisywana (lub wprowadzana) do zmiennej o nazwie j. Znak plus (+) w C++ działa tak, jak w matematyce. Pozostałe podstawowe operatory matematyczne przedstawiono w tabeli

Operator	Znaczenie	Przykład
+	Dodawanie	4 + 5
–	Odejmowanie	7 – 2
*	Mnożenie	12 * 6
/	Dzielenie	48 / 12

Literał literowy A jest przypisany do zmiennej c. Numer 9.087 jest przypisany do zmiennej o nazwie x, a następnie x jest natychmiast zastępowany nową wartością: sama (9.087) pomnożona przez 4,5. Pomaga to zilustrować, dlaczego projektanci komputerów używają gwiazdki (*) do mnożenie, a nie małych liter jak x, jak zwykle ludzie, aby pokazać mnożenie; komputer pomyliłby zmienną x z symbolem mnożenia x, gdyby oba były dozwolone.
WSKAZÓWKA: Jeśli operatory matematyczne znajdują się po prawej stronie znaku równości, program wykonuje matematykę przed przypisaniem wyniku do zmiennej. Następna linia (po komentarzu) zawiera następujące specjalne - i początkowo mylące-oświadczenie:

cout << i << "," << j << "," << c << "," << x << "\ n";

kiedy program osiągnie ten wiersz, drukuje zawartość czterech zmiennych na ekranie. Ważną częścią tej linii jest to, że cztery wartości dla i, j, c i x są drukowane na ekranie. Wyjście z tej linii to :

4, 11, A, 40,891499

Ponieważ jest to jedyny sygnał w programie, jest to jedyne wyjście, jakie generuje program przykładowy. Można by pomyśleć, że program jest dość długi na tak małe wydruki. Gdy dowiesz się więcej o C++, powinieneś być w stanie napisać więcej przydatnych programów. cout nie jest poleceniem C++. cout jest operatorem opisanym kompilatorowi w pliku #include o nazwie iostream.h i wysyła dane wyjściowe na ekran. C++ obsługuje również funkcję printf () dla sformatowanych danych wyjściowych. Widziałeś już jedną funkcję, main(), która jest tą, dla której piszesz kod. Projektanci programowania C++ napisali już kod funkcji printf. W tym momencie możesz myśleć o printf jako komendzie, która wyprowadza wartości na ekran, ale w rzeczywistości jest wbudowaną funkcją.
UWAGA: Aby odróżnić printf od zwykłych poleceń C++, nawiasy są używane po nazwie, tak jak w printf (). W C++ wszystkie nazwy funkcji mają nawiasy po sobie. Czasami te nawiasy coś mają między sobą, a czasami są puste. Ostatnie dwie linie w programie są takie:

return 0; // ZAWSZE kończ programy i funkcje z return.

}

Polecenie return po prostu mówi C++, że ta funkcja została zakończona. C++ zwraca kontrolę do tego, co kontrolowało program, zanim zaczęło działać. W tym przypadku, ponieważ była tylko jedna funkcja, kontrola jest zwracana albo do DOS albo do środowiska edycji C++. C++ wymaga wartości zwracanej. Większość programistów C++ zwraca 0 (tak jak ten program) do systemu operacyjnego. Dopóki nie użyjesz zmiennej return przez system operacyjny, nie masz większego znaczenia dla wartości zwracanej. Dopóki nie musisz być bardziej szczegółowy, zawsze zwracaj 0 z main(). W rzeczywistości wiele instrukcji return jest opcjonalnych. C++ będzie wiedzieć, kiedy dotrze do końca programu bez tej

instrukcji. Dobrą praktyką programowania jest jednak umieszczanie instrukcji return na końcu każdej funkcji, w tym main(). Ponieważ niektóre funkcje wymagają instrukcji return (jeśli zwracasz wartości), lepiej jest wpaść w nawyk ich używania, zamiast ryzykować pozostawienie go, kiedy naprawdę go potrzebujesz. Czasami zobaczysz nawiasy wokół wartości zwracanej, jak w:

return 0; // ZAWSZE kończ programy i funkcje z return.

Nawiasy są niepotrzebne i czasami prowadzą do tego, że uczniowie w C++ myślą, że powrót jest funkcją wbudowaną. Jednak nawiasy są zalecane, gdy chcesz zwrócić wyrażenie. Klamra zamykająca po powrocie wykonuje dwie rzeczy w tym programie. Sygnalizuje koniec bloku (rozpoczęty wcześniej wraz z nawiasem otwierającym), który jest końcem funkcji main() i sygnalizuje koniec programu.

Pytania Kontrolne

1. Co musi znaleźć się przed każdym komentarzem w programie C++?
2. Co to jest zmienna?
3. Co to jest literał?
4. Jakie są cztery operatory matematyczne C++?
5. Jaki operator przypisuje zmiennej jego wartość? (Podpowiedź: to się nazywa
operator przypisania.)
6. Prawda czy fałsz: zmienna może składać się tylko z dwóch typów:
liczby całkowitej i znaków.
7. Co to jest operator, który zapisuje dane wyjściowe na ekranie?
8. Czy jest następująca nazwa zmiennej lub literał ciągu?
miasto
9. Co, jeśli coś jest nie tak, z następującą instrukcją C++
RETURN;

Zmienne i literały

Aby zrozumieć przetwarzanie danych w C++, musisz zrozumieć jak C++ tworzy, przechowuje i manipuluje danymi. Ta część uczy w jaki sposób C++ obsługuje dane poprzez wprowadzenie następujących tematów:

♦ Pojęcia zmiennych i literałów

♦ Rodzaje zmiennych i literałów C++

♦ Specjalne literały

♦ Zmienne stałe

♦ Nazewnictwo i używanie zmiennych

♦ Deklarowanie zmiennych

♦ Przypisywanie wartości do zmiennych

Teraz, gdy zobaczysz przegląd języka programowania C++, możesz zacząć pisać programy w C++. W tej części możesz np. napisać własne programy od zera. Dowiedziałeś się ,że programy w C++ składają się z poleceń i danych. Dana jest sercem wszystkich programów C++; jeśli nie wypiszesz prawidłowo lub nie użyjesz zmiennych i literałów, twoje dane są niedokładne, a twoje wyniki również będą niedokładne. Powiedzenie komputerowe mówi, że jeśli włożysz śmieci, dostaniesz śmieci. To jest bardzo prawdziwe. Ludzie zwykle obwiniają komputery za błędy, ale komputery nie zawsze są winne. Zamiast tego ich dane często nie są prawidłowo wprowadzane do ich programów. Ta część zajmuje dużo czasu, skupiając się na zmiennych numerycznych i literałach numerycznych. Jeśli nie jesteś "liczbową" osobą, nie martw się. Praca z liczbami to zadanie komputera. Musisz zrozumieć tylko jak powiedzieć komputerowi, co chcesz zrobić.

Zmienne

Zmienne mają cechy. Kiedy zdecydujesz, że Twój program potrzebuje innej zmiennej, po prostu deklarujesz nową zmienną, a C++ zapewnia, że ją otrzymasz. W C++ deklaracje zmiennych można umieszczać w dowolnym miejscu w programie, o ile nie są one przywoływane, dopóki nie zostaną zadeklarowane. Aby zadeklarować zmienną, musisz zrozumieć możliwe cechy, jakie występują.

♦ Każda zmienna ma nazwę.

♦ Każda zmienna ma swój typ.

♦ Każda zmienna zawiera wartość, którą tam umieścisz, przypisując ją do tej zmiennej.

W poniższych sekcjach wyjaśniono szczegółowo każdą z tych cech.

Nazwy Zmiennych

Ponieważ w jednym programie możesz mieć wiele zmiennych, musisz im nadać nazwy, aby je śledzić. Nazwy zmiennych są unikalne, tak jak adresy domów są unikalne. Jeśli dwie zmienne mają taką samą nazwę, C++ nie wie, do którego odsyłasz, gdy zażądasz jednego z nich. Nazwy zmiennych mogą składać się z jednej litery lub nawet z 32 znaków. Ich nazwy muszą zaczynać się od litery alfabetu, ale po pierwszej literze mogą zawierać litery, cyfry i znaku podkreślenia (_). Następująca lista nazw zmiennych jest poprawna:

pensja aug91_sales i index_age kwota

Tradycyjne jest używanie małych liter dla nazw zmiennych C++. Nie musisz przestrzegać tej tradycji, ale powinieneś wiedzieć, że wielkie litery w nazwach zmiennych różnią się od małych. Na przykład każda z czterech poniższych zmiennych jest inaczej postrzegana przez kompilator C++ :

sale Sale SALE sALE

Bądź ostrożny z klawiszem Shift, gdy piszesz nazwę zmiennej. Nie należy przypadkowo zmieniać wielkości liter nazwy zmiennej w programie. Jeśli tak, C++ interpretuje je jako odrębne i oddzielne zmienne. Zmienne nie mogą mieć tej samej nazwy co polecenie lub funkcja C++. Poniżej podano niepoprawne nazwy zmiennych:

81_sales Aug91 + Sale MY AGE print

Typy zmiennych

Zmienne mogą zawierać różne typy danych. Tabela zawiera listę różnych typów zmiennych C++. Na przykład, jeśli zmienna zawiera liczbę całkowitą, C++ nie przyjmuje punktu dziesiętnego ani części ułamkowej (część po prawej stronie przecinka dziesiętnego) dla wartości zmiennej. Duża liczba typów jest możliwa w C++. Na razie najważniejsze typy, na które powinieneś się skoncentrować to char, int i float. Możesz dodać przedrostek long, aby niektóre z nich zawierały większe wartości, niż w innym przypadku. Użycie niepodpisanego prefiksu pozwala im trzymać tylko liczby dodatnie

Nazwa	Typ
`char`	Character
`unsigned char`	Unsigned character
`signed char`	Signed character (same as `char`)
`int`	Integer
`unsigned int`	Unsigned integer
`signed int`	Signed integer (same as `int`)
`short int`	Short integer
`unsigned short int`	Unsigned short integer
`signed short int`	Signed short integer (same as `short int`)
`long`	Long integer
`long int`	Long integer (same as `long`)
`signed long int`	Signed long integer (same as `long int`)
`unsigned long int`	Unsigned long integer
`float`	Floating-point
`double`	Double floating-point
`long double`	Long double floating-point

W następnej sekcji bardziej szczegółowo opisano każdy z tych typów. Na razie musisz skoncentrować się na znaczeniu zadeklarowania ich przed użyciem.

Deklarowanie zmiennych

Istnieją dwa miejsca, w których można zadeklarować zmienną:

♦ Przed kodem korzystającym ze zmiennej

♦ Przed nazwą funkcji (np. Przed main () w programie)

Pierwszy z nich jest najbardziej powszechny. Aby zadeklarować zmienną, musisz podać jej typ, a następnie jej nazwę. W poprzedniej części zobaczyłeś program, który zadeklarował cztery zmienne w następujący sposób.

* Początek funkcji main ().

* Deklaruj zmienne i i j jako liczby całkowite.

* Deklaruj zmienną c jako znak.

* Deklaracja zmiennej x jako zmienną ą

```
main()

{

int i, j; // Te trzy linie deklarują cztery zmienne.

char c;

float x;

// Reszta programu następuje tu .
```

Deklarujemy dwie zmienne całkowite o nazwach i i j. Nie masz jednak pojęcia, co znajduje się wewnątrz tych zmiennych. Zazwyczaj nie można założyć, że zmienna zawiera zero lub dowolną inną liczbę, dopóki nie zostanie przypisana jej wartość. Pierwsza linia zasadniczo mówi C++: "Zamierzam użyć dwóch zmiennych liczbowych gdzieś w tym programie. Oczekuj ich. Chcę, żeby były nazwani i i j. Kiedy wstawiam wartość do i lub j, upewniam się, że wartość jest liczbą całkowitą. " Bez takiej deklaracji nie można później przypisać wartości i lub j. Wszystkie zmienne muszą zostać zadeklarowane przed ich użyciem. Nie musi to być prawdą w innych językach programowania, takich jak BASIC, ale ma to zastosowanie w C++. Można zadeklarować każdą z tych dwóch zmiennych we własnym wierszu, tak jak w poniższym kodzie:

```
main()

{

int i;

int j;

// Reszta programu
```

W ten sposób nie zyskujesz jednak żadnej czytelności. Większość programistów C++ woli deklarować zmienne tego samego typu w tej samej linii. Druga linia w tym przykładzie deklaruje zmienną znakową o nazwie c. W tym miejscu powinny znajdować się tylko pojedyncze znaki. Następnie deklarowana jest zmienna zmiennoprzecinkowa o nazwie x.

Przykłady

1. Załóżmy, że musisz śledzić pierwsze, środkowe i ostatnie inicjały danej osoby. Ponieważ początkowy jest oczywiście znakiem, rozsądnie byłoby zadeklarować trzy zmienne znakowe, aby utrzymać trzy inicjały. W C++ możesz to zrobić za pomocą następującej instrukcji:

```
main()

{

char first, middle, last;

// Reszta programu następuje.
```

Ta instrukcja może pójść za nawiasem otwierającym main(). Informuje on resztę programu o wymaganych zmiennych trzech znaków.

2. Możesz zadeklarować te trzy zmienne również w trzech osobnych liniach, ale niekoniecznie poprawi to czytelność. Można to osiągnąć za pomocą:

```
main()

{

char first;

char middle;

char last;

// Reszta programu następuje.
```

3. Załóżmy, że chcesz śledzić wiek i wagę danej osoby. Jeśli chcesz przechowywać te wartości w liczbach całkowitych, prawdopodobnie będą to zmienne całkowite. Poniższe stwierdzenie zadeklaruje te zmienne:

```
main()

{

int wiek, waga;

// Reszta programu następuje.
```

Patrząc na typy danych

Można się zastanawiać, dlaczego tak ważne jest posiadanie tylu różnych typów zmiennych. W końcu liczba to tylko liczba. C++ ma jednak więcej typów danych niż prawie wszystkie inne języki programowania. Typ zmiennej jest krytyczny, ale wybór rodzaju spośród wielu ofert nie jest tak trudny, jak mogłoby się wydawać. Zmienna znakowa jest łatwa do zrozumienia. Zmienna znakowa może pomieścić tylko jeden znak. Nie możesz umieścić więcej niż jednego znaku w zmiennej znakowej.

Liczby całkowite zawierają liczby całkowite. Chociaż matematycy mogą się śmiać z tej definicji, liczba całkowita jest w rzeczywistości dowolną liczbą, która nie zawiera przecinka dziesiętnego. Wszystkie następujące wyrażenia są liczbami całkowitymi:

45 -932 0 12 5421

Liczby zmiennoprzecinkowe zawierają liczby dziesiętne. Są one znane matematykom jako liczby rzeczywiste. Za każdym razem, gdy musisz przechowywać wynagrodzenie, temperaturę lub dowolną inną liczbę, która może mieć część ułamkową (część dziesiętną), musisz zapisać ją w zmiennej zmiennoprzecinkowej. Wszystkie poniższe wyrażenia są liczbami zmiennoprzecinkowymi, a każda zmienna zmiennoprzecinkowa może je przechowywać:

45,12 -2344,54 0,00 0,04594

Czasami trzeba śledzić duże liczby, a czasami trzeba śledzić mniejsze liczby. Tabela pokazuje listę zakresów, które może pomieścić każdy typ zmiennej C++.

Typ	Zakres
char	–128 to 127
unsigned char	0 to 255
signed char	–128 to 127
int	–32768 to 32767
unsigned int	0 to 65535
signed int	–32768 to 32767
short int	–32768 to 32767
unsigned short int	0 to 65535
signed short int	–32768 to 32767
long int	–2147483648 to 2147483647
signed long int	–2147483648 to 2147483647
float	–3.4E–38 to 3.4E+38
double	–1.7E–308 to 1.7E+308
long double	–3.4E–4932 to 1.1E+4932

Zauważ, że long integer i long double zwykle zawierają większe liczby (i dlatego mają wyższą dokładność) niż zwykłe liczby całkowite i zwykłe zmienne podwójne zmiennoprzecinkowe. Wynika to z większej liczby lokalizacji pamięci używanych przez wiele kompilatorów C++ dla tych typów danych. Znowu jest to zazwyczaj - ale nie zawsze - przypadek. Ogólnie, wszystkie zmienne numeryczne powinny być ze znakiem (domyślne), chyba że wiesz na pewno, że twoje dane zawierają tylko liczby dodatnie.

(Niektóre wartości, takie jak wiek i odległości, są zawsze dodatnie.) Dokonując zmiennej bez znaku, uzyskuje się niewielki dodatkowy zakres pamięci Zakres wartości musi być jednak zawsze dodatni. Oczywiście musisz zdawać sobie sprawę z rodzaju danych przechowywanych przez zmienne. Na pewno nie zawsze wiesz dokładnie, co każda zmienna trzyma, ale możesz mieć ogólny pomysł. Na przykład, przechowując wiek osoby, powinieneś zdawać sobie sprawę, że długa zmienna całkowita byłaby stratą przestrzeni, ponieważ nikt nie może dożyć wieku, który nie może być zapisany przez zwykłą liczbę całkowitą. Na początku może wydawać się dziwne, że w tabeli stwierdza się, że zmienne znaków mogą zawierać wartości liczbowe. W C++, całkowite i zmienne znaków często mogą być używane zamiennie. Każdy znak tabeli ASCII ma unikalny numer, który odpowiada jego lokalizacji w tabeli. Jeśli przechowujesz liczbę w zmiennej znakowej, C++ traktuje dane tak, jakby były ASCII znak, który pasował do tego numeru w tabeli. I odwrotnie, możesz przechowywać dane znakowe w zmiennej liczbowej. C++ znajduje numer ASCII postaci i zapisuje ten numer zamiast znaku

Wyznaczanie długich, bez znaku i zmiennoprzecinkowych literałów

Kiedy wpiszesz liczbę, C++ interpretuje jej typ jako najmniejszy typ, który może pomieścić tę liczbę. Na przykład, jeśli wpiszesz 63, C++ wie, że ten numer pasuje do lokalizacji ze znakiem liczby całkowitej. Nie traktuje liczby jako długiej liczby całkowitej, ponieważ 63 nie jest wystarczająco duże, aby zagwarantować długą literalną wielkość całkowitą. Jednak można dodać znak sufiksu do literałów numerycznych, aby zastąpić domyślny typ. Jeśli wstawisz L na końcu liczby całkowitej, C++ interpretuje tę liczbę całkowitą jako długą liczbę całkowitą. Liczba 63 jest liczbą całkowitą dosłowną, ale liczba 63L jest długością całkowitą dosłowną. Przypisz sufiks U, aby wyznaczyć liczbę całkowitą bez znaku. Liczba 63 to domyślnie liczba całkowita ze znakiem. Jeśli wpiszesz 63U, C++ traktuje je jako liczbę całkowitą bez znaku. Sufiks UL oznacza niepodpisany długi literał. C++ interpretuje wszystkie literały zmiennoprzecinkowe (liczby zawierające przecinki dziesiętne) jako podwójne litery zmiennoprzecinkowe (podwójne liczby zmiennoprzecinkowe trzymają większe liczby niż literały zmiennoprzecinkowe). Ten proces zapewnia maksymalną dokładność w takich liczbach. Jeśli użyjesz literału 6.82, C++ traktuje to jako typ danych zmiennoprzecinkowych double, nawet jeśli pasowałoby to do zwykłego float. Sufiks liczby zmiennoprzecinkowej (F) lub długi podwójny sufiks zmiennoprzecinkowy (L) można dołączyć do literałów, które zawierają kropki dziesiętne reprezentujące literał zmiennoprzecinkowy lub długi literał podwójny zmiennoprzecinkowy. Rzadko możesz używać tych sufiksów, ale jeśli musisz przypisać wartość dosłowną do zmiennej rozszerzonej lub bez znaku, twoje literały mogą być nieco dokładniejsze, jeśli dodasz U, L, UL lub F (ich odpowiedniki z małą literą też działają), .

Przypisywanie wartości do zmiennych

Teraz, gdy wiesz o typach zmiennych C++, jesteś gotowy, aby poznać specyfikę przypisywania wartości do tych zmiennych. Robisz to za pomocą instrukcji przypisania. Znak równości (=) służy do przypisywania wartości do zmiennych. Format instrukcji przypisania jest

zmienna = wyrażenie;

Zmienna jest dowolną zmienną, którą zadeklarowałeś wcześniej. Wyrażenie jest dowolną zmienną, literałem, wyrażeniem lub kombinacją, która tworzy wynikowy typ danych, który jest taki sam, jak typ danych zmiennej

Przykłady

1. Jeśli chcesz śledzić aktualny wiek, zarobki i osoby na utrzymaniu, możesz zapisać te wartości w trzech zmiennych C++. Najpierw zadeklaruj zmienne, decydując o poprawnych typach i nazwach dobrych dla

nich. Następnie przypisujesz im wartości. Później w programie wartości te mogą ulec zmianie (na przykład, jeśli program oblicza dla ciebie nowy wzrost płac). Dobre nazwy zmiennych obejmują age, salary, dependents, pensję i osoby na utrzymaniu. Aby zadeklarować te trzy zmienne, pierwsza część funkcji main () będzie wyglądać następująco:

```
// Zadeklaruj i przechowuj trzy wartości.

main()

{

int age;

float salary;

int dependents;
```

Zauważ, że nie musisz zadeklarować razem wszystkich zmiennych całkowitych. Następne trzy instrukcje przypisują wartości do zmiennych.

```
age=32;
salary=25000.00;
dependents=2;
// Reszta programu
```

Ten przykład nie jest zbyt długi i nie robi wiele, ale ilustruje użycie i przypisanie wartości do zmiennych.

2. Nie umieszczaj przecinków w wartościach, które przypisujesz zmiennym. Literały numeryczne nigdy nie powinny zawierać przecinków. Następujące stwierdzenie jest nieprawidłowe:

```
wynagrodzenie = 25 000,00;
```

3. Możesz przypisać zmienne lub wyrażenia matematyczne do innych zmiennych. Załóżmy, że wcześniej w programie zapisałeś swoją stawkę podatku w zmiennej o nazwie tax_rate, a następnie zdecydowałeś się użyć swojej stawki podatkowej również dla stawki współmałżonka. W odpowiednim momencie programu, będziesz kodować następujące elementy:

```
spouse_tax_rate = tax_rate;
```

(Dodawanie spacji wokół znaku równości jest dopuszczalne dla kompilatora C++, ale nie musisz tego robić.) W tym momencie w programie wartość w tax_rate jest kopiowana do nowej zmiennej o nazwie spouse_tax_rate. Wartość w tax_rate jest nadal dostępna po zakończeniu tej linii. Zmienne zostały zadeklarowane wcześniej w programie. Jeśli stawka podatkowa twojego współmałżonka wynosi 40 procent twojego, możesz przypisać wyrażenie do zmiennej małżonka, jak w:

```
spouse_tax_rate = tax_rate * .40;
```

Każdy z czterech symboli matematycznych, których nauczyłeś się w poprzedniej części, a także dodatkowe, których nauczyłeś się później w tekście, może być częścią wyrażenia przypisanego do zmiennej.

4. Jeśli chcesz przypisać dane znakowe do zmiennej znakowej, musisz ją ująć w znaki pojedynczego cudzysłowu. Wszystkie literały znaków C++ muszą być ujęte w znaki pojedynczego cudzysłowu. Poniższa sekcja programu deklaruje trzy zmienne, następnie przypisuje im trzy inicjały. Inicjały są literałami znaków, ponieważ są ujęte w pojedyncze cudzysłowy.

```
main()

{

char first, middle, last;

first = ‘G’;

middle = ‘M’;

last = ‘P’;

// Reszta programu
```

Ponieważ są to zmienne, możesz ponownie przypisać ich wartości później, jeśli program to gwarantuje.

UWAGA: Nie mieszaj typów. C pozwala programistom to zrobić, ale C++ tego nie robi. Na przykład w zmiennej middle przedstawionej w poprzednim przykładzie nie można było zapisać literału zmiennoprzecinkowego:

```
middle  = 345,43244; // Nie możesz tego zrobić!
```

Gdyby tak było, środek miałby dziwną wartość, która wydawałaby się bez znaczenia. Upewnij się, że wartości przypisane do zmiennych odpowiadają typowi zmiennej. Jedynym istotnym wyjątkiem jest sytuacja, w której w krótkim czasie zostanie przypisana liczba całkowita do zmiennej znakowej lub znak do zmiennej liczbowej.

Literały

Podobnie jak w przypadku zmiennych, istnieje kilka typów literałów C++. Pamiętaj, że literał się nie zmienia. Liczby całkowite to liczby całkowite, które nie zawierają kropek dziesiętnych. Literały zmiennoprzecinkowe są liczbami, które zawierają ułamkową część (kropkę dziesiętną z opcjonalną wartością na prawo od kropki dziesiętnej).

Przypisywanie literałów całkowitych

Wiesz już, że liczba całkowita to dowolna liczba całkowita bez kropki dziesiętnej. C++ umożliwia przypisywanie zmiennych liczb całkowitych do zmiennych, używanie liczb całkowitych do obliczeń i drukowanie liczb całkowitych za pomocą operatora cout. Zwykła liczba całkowita nie może rozpoczynać się od wiodącego 0. Dla C++, liczba 012 nie jest liczbą dwamnaście. Jeśli poprzedzisz liczbę całkowitą literałem 0, C++ interpretuje ją jako ósemkową literał. Literał ósemkowy jest liczbą podstawową-8. Ósemkowy system numerowania nie jest już używany w dzisiejszych systemach komputerowych. Nowsze wersje C++ zachowują ósemkowe możliwości zgodności z poprzednimi wersjami. Szczególną liczbą całkowitą w C++, która jest nadal bardzo często używana dzisiaj, jest 16-literowy lub szesnastkowy, literał. Jeśli chcesz reprezentować szesnastkową liczbę całkowitą, dodaj przedrostek 0x do niej. Następujące liczby są liczbami szesnastkowymi:

```
0x10  0x2C4  0xFFFF  0X9
```

Zauważ, że nie ma znaczenia, czy używasz małej lub dużej litery x po początkowym zera, czy wielkiej lub małej litery w systemie szesnastkowym (dla liczb szesnastkowych od A do F). Jeśli piszesz aplikacje biznesowe w C++, możesz pomyśleć, że nigdy nie będziesz potrzebował używać szesnastkowych i możesz mieć rację. Aby jednak w pełni zrozumieć C++ i swój komputer, powinieneś zapoznać się z podstawami liczb szesnastkowych.

UWAGA: Literały zmiennoprzecinkowe mogą zaczynać się od początkowego zera, na przykład 0.7. Są poprawnie interpretowane przez C++. Tylko liczby całkowite mogą być literałami szesnastkowymi lub ósemkowymi.

Rozmiar słowa twojego komputera jest ważny

Jeśli piszesz wiele programów systemowych, które używają liczb szesnastkowych, prawdopodobnie chcesz zapisać te liczby w zmiennych bez znaku. Dzięki temu C++ nieprawidłowo interpretuje liczby dodatnie jako liczby ujemne. Na przykład, jeśli twój komputer przechowuje liczby całkowite w 2-bajtowych słowach (jak większość komputerów), szesnastkowa literalna 0xFFFF reprezentuje albo -1 albo 65535, w zależności od tego, jak interpretowany jest bit znaku. Jeśli zadeklarowano liczbę całkowitą bez znaku, taką jak

unsigned_int i_num = 0xFFFF;

C++ wie, że chcesz użyć bitu znaku jako danych, a nie znaku. Jeśli zadeklarowałeś tę samą wartość jako liczbę całkowitą ze znakiem, jak w

int i_num = 0xFFFF; / * Słowo "ze znakiem" jest opcjonalne. * /

C++ uważa, że jest to liczba ujemna (-1), ponieważ bit znaku jest włączony. (Jeśli miałbyś konwertować 0xFFFF na binarny, dostałbyś szesnaście jedynek.)

Przypisywanie literałów ciągu znaków

Jeden typ literału C++, nazywany literałem ciągu, nie ma dopasowanej zmiennej. Literał ciągu jest zawsze ujęty w podwójny cudzysłów. Oto przykłady literałów łańcuchowych:

"Programowanie w C++" "123" "" "4323 E. Dąb" "x"

Dowolny ciąg znaków między podwójnym cudzysłowem - nawet pojedynczym znakiem - jest uważany za literał ciągu. Pojedyncza spacja, słowo lub grupa słów między podwójnymi cudzysłowami są literałami napisanymi w C++. Jeśli literał łańcuchowy zawiera tylko cyfry, to nie jest liczbą; jest to ciąg cyfr, których nie można użyć do wykonywania matematyki. Możesz wykonywać matematykę tylko na liczbach, a nie na literałach ciągów.

UWAGA: Literał łańcuchowy to dowolny znak, cyfra lub grupa znaków ujęta w podwójny cudzysłów. Literałem znakowym jest dowolny znak zawarty w pojedynczym cudzysłowie.

Podwójny cudzysłów nigdy nie jest uważany za część literału literowego. Podwójne cudzysłowy otaczają ciąg znaków i po prostu informują kompilator C++, że kod jest dosłownym ciągiem znaków, a nie innym typem literału. Łatwo jest drukować literały ciągów. Po prostu umieść literał ciągu w instrukcji cout. Poniższy kod wypisuje literał ciągu na ekranie:

Poniższy kod wypisuje literał łańcuchowy, C++ w przykładach.

cout << " C++ w przykładach";

Przykłady

1.Poniższy program wyświetla prostą wiadomość na ekranie. Nie są potrzebne żadne zmienne, ponieważ nie zapisano ani nie obliczono odniesienia.

// Nazwa pliku: ST1.CPP

// Wyświetl ciąg znaków na ekranie.

```
#include <iostream.h>

main()

{

cout << “Programowanie w C++ jest fajne!”;

return 0;

}
```

Pamiętaj, aby ostatnia linia w twoim programie C++ (przed nawiasem zamykającym) to było return

2. Prawdopodobnie chcesz oznaczyć dane wyjściowe ze swoich programów. Nie wyświetlaj wartości zmiennej, chyba że wyświetlasz również literał łańcuchowy, który opisuje tę zmienną. Następujący program oblicza podatek od sprzedaży za sprzedaż i wyświetla podatek. Zauważ, że drukowana jest wiadomość, która mówi użytkownikowi, co oznacza następny numer.

```
// Nazwa pliku: ST2.CPP

// Oblicz podatek od sprzedaży i wyświetl odpowiednią wiadomość.

#include <iostream.h>

main()

{

float sale, tax;

float tax_rate = .08; // Procent podatku od sprzedaży

// Określ kwotę sprzedaży.

sale = 22.54;

// Oblicz podatek od sprzedaży..

tax = sale * tax_rate;

// Wyświetl wyniki.

cout << “ Podatek obrotowy to “ << tax << “\n”;

return 0;

}
```

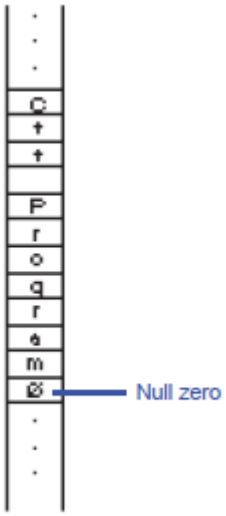
Oto wyniki z programu:

Podatek obrotowy wynosi 1,8032

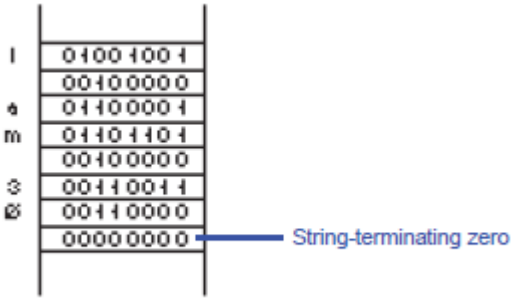
Później nauczysz się wyświetlać z dokładnością do dwóch miejsc po przecinku aby grosze pojawiły się poprawnie.

Końcówki literałowo-łańcuchowe

Dodatkowy aspekt literałów łańcuchowych czasem dezorientuje początkujących programistów C++. Wszystkie napisy łańcuchowe kończą się zerem. Nie widać zera, ale C++ przechowuje zero na końcu łańcucha w pamięci. Rysunek pokazuje, jak wygląda ciąg "C++ Program" w pamięci.



Nie musisz się martwić umieszczaniem zera na końcu literału ciągłego; C++ robi to za każdym razem, gdy przechowuje ciąg znaków. Jeśli twój program zawierał na przykład napis "Program w C++", kompilator rozpoznałby go jako literał łańcuchowy (z podwójnego cudzysłowu) i zapisał zero na końcu. Zero jest ważne dla C++. Nazywa się to ogranicznikiem łańcucha. Bez tego C++ nie wiedziałby, gdzie literał łańcucha zakończył się w pamięci. (Pamiętaj, że podwójne cudzysłowy nie są zapisywane jako część ciągu znaków, więc C++ nie może ich użyć do określenia, gdzie kończy się ciąg znaków). Zero ograniczające łańcuch znaków nie jest tym samym co znak zera. Jeśli spojrzysz na tabelę ASCII , zobaczysz, że pierwszy wpis, numer ASCII 0, jest znakiem pustym. To zero z ograniczeniem ciągów różni się od znaku "0", który ma wartość ASCII równą 48. Wszystkie lokalizacje pamięci w komputerze faktycznie trzymają wzory bitowe dla znaków. Jeśli litera A jest zapisana w pamięci, A tam nie ma; binarny wzór bitowy dla ASCII A (01000001) jest tam przechowywany. Ponieważ binarny wzorzec bitowy dla zerowego zera to 00000000, zera ograniczające ciąg jest również nazywane binarnym zerem. Aby to zilustrować, rysunek pokazuje wzory bitów dla następujących literałów łańcuchowych, gdy są przechowywane w pamięci: "I am 30"



Rysunek pokazuje, jak łańcuch jest przechowywany w pamięci twojego komputera na poziomie binarnym. Ważne jest, abyś rozpoznał, że znak 0, wewnątrz liczby 30, nie jest równy zeru (na poziomie bitów) jako zakończenie zerowe łańcucha znaków. Jeśli tak, C++ pomyślałby, że ten ciąg zakończył się po 3, co byłoby niepoprawne. Jest to dość zaawansowana koncepcja, ale naprawdę musisz ją zrozumieć, zanim przejdziesz dalej.

Długości napisów

Wiele razy twój program musi znać długość łańcucha. Staje się to krytyczne, gdy nauczysz się akceptować wprowadzanie ciągów z klawiatury. Długość łańcucha to liczba znaków do, ale nie zawierająca, zero ograniczające łańcuch. Nie uwzględniaj znaku pustego w tej liczbie, nawet jeśli wiesz, że C++ dodaje go na końcu łańcucha.

Przykłady

1.Oo literały ciągów

„0", „C", „Trochę dłuższy łańcuch"

Przypisywanie literałów znakowych

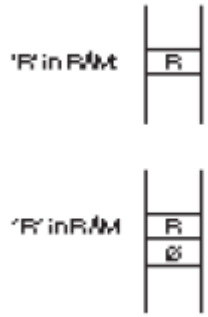
Wszystkie literały znakowe C powinny być ujęte w pojedyncze cudzysłowy. Pojedyncze cudzysłowy nie są częścią znaku , ale służą do rozgraniczenia postaci. Poniżej przedstawiono poprawne znaki literowe C++:

'w' 'W' 'C' '7' '*' '=' '.' 'K'

C++ nie dodaje zerowego zera do końca literałów znakowych. Powinieneś wiedzieć, że poniższe są różne dla C++.

'R' i "R"

'R' jest literałem pojedynczego znaku. Ma długość jednego znaku, ponieważ wszystkie literały znakowe (i zmienne) mają długość jednego znaku. "R" jest ciągiem znaków, ponieważ jest on ograniczony podwójnymi cudzysłowami. Jego długość jest również jedna, ale zawiera zero null w pamięci, więc C++ wie, gdzie kończy się ciąg. Z powodu tej różnicy nie można mieszać literałów znakowych i literałów ciągów znakowych. Rysunek poniższy pokazuje, jak te dwa literały są przechowywane w pamięci.



Wszystkie znaki alfabetyczne, numeryczne i specjalne na klawiaturze mogą być literałami znaków. Niektórych postaci nie można jednak przedstawić za pomocą klawiatury. Zawierają one niektóre z wyższych znaków ASCII (takich jak hiszpański Ñ). Ponieważ nie masz kluczy dla każdego znaku w tabeli ASCII, C++ pozwala ci reprezentować te znaki, wpisując ich szesnastkową liczbę ASCII w pojedynczym cudzysłowie. Na przykład, aby zapisać hiszpański Ñ w zmiennej, wyszukaj jego szesnastkowy numer ASCII. Jest to A5. Dodaj przedrostek \ x do niego i ułóż go w pojedynczy cudzysłów, aby C++ wiedział, jak używać znaku specjalnego. Możesz to zrobić za pomocą następującego kodu:

char sn = '\ xA5'; // Umieszcza hiszpański Ñ w zmiennej o nazwie sn.

W ten sposób można zapisać (lub wydrukować) dowolny znak z tabeli ASCII, nawet jeśli ta postać nie ma klucza na klawiaturze. Pojedynczy cudzysłów nadal informuje C++, że pojedynczy znak znajduje się

wewnątrz cudzysłowów. Mimo że "\ xA5" zawiera cztery znaki wewnątrz znaków cudzysłowu, te cztery znaki reprezentują pojedynczy znak, a nie ciąg znaków. Jeśli dołączysz te cztery znaki do literału, C++ potraktuje \ xA5 jako pojedynczy znak w ciągu znaków. Poniższy tekst literowy "An accented a is \ xA0" jest ciągiem C++, który ma 18 znaków, a nie 21 znaków. C++ interpretuje znak \ xA0 jako á, tak jak powinien.

UWAGA: Jeśli znasz znaki ASCII, wpisując ich numery ASCII za pomocą kombinacji Alt-klawiatura, nie rób tego w programach w C++. Mogą działać na twoim komputerze (nie wszystkie kompilatory C++ obsługują to), ale twój program może nie być przenośny dla kompilatora C++ innego komputera.

Dowolny znak poprzedzony ukośnikiem odwrotnym, \, (takie jak te zostały) nazywa się sekwencją specjalną lub znakiem ucieczki. Tabela poniższa pokazuje dodatkowe sekwencje specjalne, które są przydatne, gdy chcesz wydrukować znaki specjalne.

WSKAZÓWKA: Włącz "\ n" w cout, jeśli chcesz przejść do następnego wiersza podczas drukowania dokumentu.

Znak Ucieczki	Znaczenie
\a	Alarm (dzwonek terminala)
\b	Backspace
\f	Znak kończący wiersz (dla drukarek)
\n	Nowa linia (powrót karetki i wysuw linii)
\r	Powrót karetki
\t	Tabulator
\v	Pionowa tabulacja
\\	Backslash
\?	Znak zapytania
\’	Pojedynczy cudzysłów
\”	Podwójny cudzysłów
\000	Liczba ósemkowa
\xhh	Liczba szesnastkowa
\0	Zero null (lub zero binarne)

Matematyka ze znakami C++

Ponieważ C++ tak blisko łączy znaki z ich numerami ASCII, możesz wykonywać arytmetyczne dane na znakach. Poniższa sekcja kodu,

char c;

c = "T" + 5; // Dodaj pięć do znaku ASCII.

faktycznie przechowuje Y w c. Wartość ASCII litery T wynosi 84. Dodanie 5 do 84 daje 89. Ponieważ zmienna c nie jest zmienną całkowitą, ale jest zmienną znakową, C++ dodaje znak ASCII dla 89, a nie faktyczną liczbę. I odwrotnie, możesz przechowywać literały znaków w zmiennych całkowitych. Jeśli tak, C++ przechowuje odpowiedni numer ASCII dla tej postaci. Poniższa sekcja kodu

int i = "P";

nie wstawia litery P do i, ponieważ i nie jest zmienną znakową. C++ przypisuje liczbę 80 w zmiennej, ponieważ 80 to numer ASCII dla litery P.

Przykłady

1. Aby wyświetlić dwie nazwy w dwóch różnych liniach, dodając \ n między nimi. Wyświetl imię Heniek , przesuń kursor w dół do nowej linii i wyświetl Jurek

cout << “Heniek\nJurek”;.

Gdy program osiągnie ten wiersz, wyświetli

Heniek

Jurek

Można również oddzielić te dwie nazwy przez dołączenie większej liczby operatorów cout, tak jak:

cout << "Harry" << "\ n" << "Jerry";

Ponieważ \n zajmuje tylko jeden bajt pamięci, możesz wypisać go jako literał znaków, wpisując ‘\ n’ zamiast poprzedniego "\n".

2. Poniższy krótki program zadzwoni dzwonkiem twojego komputera, przypisując sekwencję ucieczki \a do zmiennej, a następnie wyświetlając tę zmienną.

```
// Nazwa pliku: BELL.CPP

// Dzwoni dzwonkiem

#include <iostream.h>

main()

{

char bell=’\a’;

cout << bell; // Brak nowej linii w tym miejscu..

return 0;

}
```

Stałe Zmienne

Określenie zmienna stała może wydawać się sprzecznością. W końcu stała nigdy się nie zmienia, a zmienna zawiera wartości, które się zmieniają. W terminologii C++ można zadeklarować zmienne jako stałe ze słowem kluczowym const. W całym programie stałe zachowują się jak zmienne; możesz użyć zmiennej stałej wszędzie, gdzie możesz użyć zmiennej, ale nie możesz zmienić stałych zmiennych. Aby zadeklarować stałą, umieść słowo kluczowe const przed deklaracją zmiennej, na przykład:

```
const int days_of_week = 7;
```

C++ oferuje słowo kluczowe const jako ulepszenie dyrektywy preprocessor #define używanej przez C. Chociaż C++ obsługuje również #define, const umożliwia określenie stałych wartości dla określonych typów danych. Słowo kluczowe const jest odpowiednie, gdy masz dane, które się nie zmieniają. Na przykład matematyczny π jest dobrym kandydatem na stałą. Jeśli przypadkowo spróbujesz zapisać wartość w stałej, C++ cię poinformuje. Większość programistów C++ decyduje się na wpisanie ich stałych nazw wielkimi literami, aby odróżnić je od zwykłych zmiennych. Jest to jedyny przypadek, gdy wielkie litery królują w C++.

UWAGA: Ten tekst zastrzega stałą nazwy stałych programu C++ zadeklarowanych za pomocą słowa kluczowego const. Termin literał jest używany dla wartości danych liczbowych, znakowych i łańcuchowych. Niektóre książki decydują się na używanie terminów stały i literał zamiennie, ale w C++ różnica może być krytyczna.

Przykład

Załóżmy, że nauczyciel chciał obliczyć obszar koła dla klasy. Aby to zrobić, nauczyciel potrzebuje wartości π (matematycznie, π wynosi około 3,14159). Ponieważ π pozostaje stałe, jest dobrym kandydatem na słowo kluczowe const, jak pokazuje poniższy program:

*Skomentuj nazwę pliku i opis programu.

*Zadeklaruj stałą wartość dla π.

*Zadeklaruj zmienne dla promienia i obszaru.

*Oblicz i wyświetl pole dla obu wartości promienia.

```
// Nazwa pliku: AREAC.CPP

// Oblicza okrąg o promieniu 5 i 20

#include <iostream.h>

main()

{

const float PI=3.14159;

float radius = 5;

float area;

area = radius * radius * PI; // Obliczanie powierzchni okręgu

cout << “Pole to  “ << area <pole obszar z nowym promieniem

area = radius * radius * PI;

cout << “Pole to  “ << area << “ o promieniu  20.\n”;

return 0;

}
```

Tablice znaków i ciągi znaków

Mimo że C++ nie ma zmiennych łańcuchowych, możesz działać tak, jakby C++ je zawierał za pomocą tablic znaków. Pojęcie tablic może być dla ciebie nowością, ale ta część wyjaśnia, jak łatwo jest je deklarować i używać. Po zadeklarowaniu tych tablic mogą one przechowywać ciągi znaków - tak jakby były prawdziwymi zmiennymi łańcuchowymi. Ta część obejmuje

♦ Tablice znaków

♦ Porównanie tablic znaków i łańcuchów znaków

♦ Przykłady tablic znaków i łańcuchów znaków

Po opanowaniu tej części, jesteś na najlepszej drodze, aby móc manipulować prawie wszystkimi typami zmiennych C++. Manipulowanie znakami i słowami to jedna cecha, która oddziela komputer od potężnego kalkulatora; ta możliwość daje komputerowi prawdziwe możliwości przetwarzania danych.

Tablice znaków

Prawie każdy typ danych w C++ ma zmienną, ale nie ma zmiennej do przechowywania ciągów znaków. Autorzy C++ zdali sobie sprawę, że potrzebujesz jakiegoś sposobu przechowywania łańcuchów w zmiennych, ale zamiast zapisywania ich w zmiennej łańcuchowej (jak niektóre języki, takie jak BASIC lub Pascal), musisz przechowywać je w tablicy znaków. Jeśli nigdy wcześniej nie programowałeś, tablica może być dla ciebie nowa. Tablica jest listą zmiennych (nazywanych czasem tabelą), a większość języków programowania pozwala na korzystanie z takich list. Załóżmy, że musisz śledzić zapisy sprzedaży 100 sprzedawców. Możesz stworzyć 100 nazw zmiennych i przypisać każdemu z nich rekord sprzedaży innego sprzedawcy. Wszystkie te różne nazwy zmiennych są jednak trudne do śledzenia. Gdybyśmy umieścili je w tablicy zmiennych zmiennoprzecinkowych, musielibyśmy śledzić tylko jedną nazwę (nazwę tablicy) i odnosić się do każdej ze 100 wartości za pomocą indeksu liczbowego. Ostatnie kilka części kursu obejmuje bardziej szczegółowo przetwarzanie macierzy. Jednak aby pracować z danymi ciągu znaków we wczesnych programach, musisz zapoznać się z pojęciem tablic znaków. Ponieważ ciąg jest po prostu listą jednego lub więcej znaków, a tablica znaków jest idealnym miejscem do przechowywania ciągów informacji. Załóżmy, że chcesz śledzić pełne imię i nazwisko osoby, jej wiek i wynagrodzenie w zmiennych. Wiek i płaca są łatwe, ponieważ istnieją typy zmienne, które mogą przechowywać takie dane. Poniższy kod deklaruje te dwie zmienne:

```
int age;

float salary;
```

Nie posiadasz żadnej zmiennej łańcuchowej do przechowywania nazwy, ale możesz utworzyć odpowiednią tablicę znaków (która w rzeczywistości jest jedną lub więcej zmiennych znakowych z rzędu w pamięci) z następującą deklaracją:

```
char name [15];
```

Zastrzega to tablicę znaków. Deklaracja tablicowa zawsze zawiera nawiasy ([]), które deklarują spację dla tablicy. Ta tablica ma 15 znaków. Nazwa tablicy to name. Możesz także przypisać wartość do tablicy znaków w chwili deklarowania tablicy. Następująca deklaracja deklaracji nie tylko deklaruje tablicę znaków, ale również przypisuje nazwę "Michael Jones" w tym samym czasie:

Zadeklaruj tablicę znaków o nazwie nazwa o długości 15 znaków i przypisz Michaela Jonesa do tablicy.

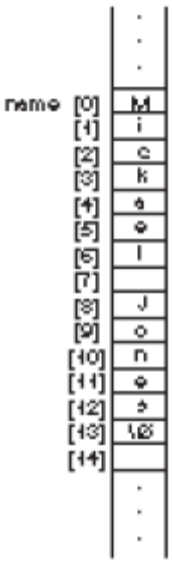
```
char name [15] = "Michael Jones";
```

Rysunek 1 pokazuje wygląd tej tablicy w pamięci. Każde z 15 pól tablicy nazywa się elementem. Zwróć uwagę na zerowe zero (znak kończący łańcuch) na końcu łańcucha. Zauważ także, że ostatni znak tablicy nie zawiera żadnych danych. Wypełniłeś tylko pierwsze 14 elementów tablicy danymi i zerami danych. 15 element rzeczywiście ma wartość - ale cokolwiek następuje po ciągu zakończonym null, nie jest problemem.

Możesz uzyskać dostęp do poszczególnych elementów w tablicy lub możesz uzyskać dostęp do tablicy jako całości. Jest to podstawowa zaleta tablicy w porównaniu z użyciem wielu zmiennych o różnych nazwach. Możesz przypisać wartości do poszczególnych elementów tablicy, umieszczając w nawiasach położenie elementów, zwane indeksem dolnym, w następujący sposób:

name [3] = 'k';

To powoduje nadpisanie h w nazwisku Michael k. Ciąg jest teraz podobny do tego na rysunku 2.



Wszystkie indeksy tablicowe zaczynają się od zera. Dlatego, aby nadpisać pierwszy element, musisz użyć 0 jako dolnego indeksu. Przypisanie nazwy [3] (jak powyżej) zmienia wartość czwartego elementu

w tablicy. Możesz wydrukować cały ciąg - lub dokładniej całą tablicę - za pomocą pojedynczej instrukcji cout, jak następuje:

cout << nazwa;

Zauważ, że gdy wyświetlasz tablicę, nie wstawiasz nawiasów za nazwą tablicy. Musisz zachować wystarczającą liczbę znaków w tablicy, aby pomieścić cały ciąg znaków. Następująca linia,

char name [5] = "Michael Jones";

jest niepoprawna, ponieważ rezerwuje tylko pięć znaków dla tablicy, podczas gdy nazwa i jej zero null wymagają 14 znaków. Jednak C++ daje komunikat o błędzie dla tego błędu (illegal initialization).

Jeśli twój ciąg zawiera 13 znaków, to musi mieć również 14-ty dla zerowego zerowego lub nigdy nie będzie traktowany jak ciąg. Aby pomóc w wyeliminowaniu tego błędu, C++ daje ci skrót. Poniższe instrukcje dwóch tablic znaków są takie same:

char char [9] = "Ogier";

i

char horse [] = "Ogier";

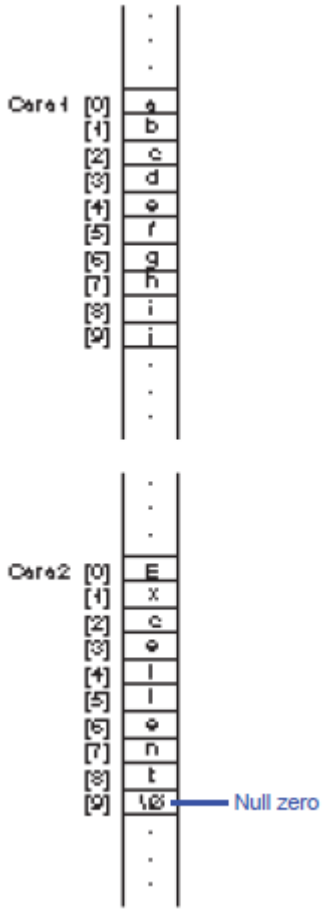
Jeśli przypiszesz wartość do tablicy znaków w tym samym czasie, gdy zadeklarujesz tablicę, C++ zlicza długość ciągu, dodaje jedną dla zerowego zera i rezerwuje dla ciebie obszar tablicy. Jeśli nie przypisujesz wartości do tablicy w momencie jej deklaracji, nie możesz jej zadeklarować za pomocą pustych nawiasów. Poniższa instrukcja ,

char ludzie [];

nie rezerwuje żadnej przestrzeni dla tablicy o nazwie ludzie. Ponieważ nie przypisałeś wartości do tablicy, gdy ją zadeklarowałeś, C++ przyjmuje, że ta tablica zawiera zero elementów. Dlatego nie ma miejsca na późniejsze wstawianie wartości do tej tablicy. Większość kompilatorów generuje błąd, jeśli spróbujesz tego dokonać.

Tablice znaków a łańcuchy

W poprzedniej sekcji zobaczyłeś, jak umieścić ciąg w tablicy znaków. Łańcuchy mogą istnieć w C++ tylko jako literały łańcuchowe lub jako informacje przechowywane w tablicach znaków. W tym momencie musisz tylko zrozumieć, że ciągi muszą być przechowywane w tablicach znaków. Podczas czytania tego tekstu i zaznajamiania się z tablicami i ciągami, powinieneś stać się bardziej wygodny w ich użyciu. Spójrz na dwie tablice pokazane na rysunku 3. Pierwsza, zwana cara1, jest tablicą znaków, ale nie zawiera ciągu. Zamiast ciągu znaków zawiera listę kilku znaków. Druga tablica, zwana cara2, zawiera ciąg znaków, ponieważ na końcu ma zero zerowe.



Można zainicjować te tablice następującymi instrukcjami przypisania.

Zadeklaruj tablicę cara1 z 10 pojedynczymi znakami.

Deklaracja tablicy cara2 za pomocą łańcucha znaków "Excellent".

char cara1[10]={‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’,‘j’};
char cara2[10]=”Excellent”;

Jeśli chcesz umieścić tylko pojedyncze znaki w tablicy, musisz zamknąć listę znaków w nawiasach, jak pokazano. Możesz zainicjować cara1 później w programie, używając instrukcji przypisania, jak robi to poniższy kod.

char cara1 [10];

cara1 [0] = "a";

cara1 [1] = 'b';

cara1 [2] = "c";

cara1 [3] = 'd';

cara1 [4] = 'e';

cara1 [5] = 'f';

cara1 [6] = "g";

```
cara1 [7] = 'h';

cara1 [8] = "i";

cara1 [9] = 'j'; // Ostatni element możliwy z indeksem dolnym dziewięć.
```

Ponieważ tablica znaków cara1 nie zawiera zerowego zera, nie zawiera ciągu znaków. Zawiera znaki, które mogą być przechowywane w tablicy - i używane osobno - ale nie mogą być traktowane w programie tak, jakby były ciągiem znaków. Ponieważ tablica znaków nie jest zmienną łańcuchową (może być używana tylko do przechowywania ciągu znaków), nie może przejść po lewej stronie znaku równości (=). Poniższy program jest nieprawidłowy:

```
#include <iostream.h>

main()

{

char petname[20]; // Rezerwuj przestrzeń dla nazwy zwierzęcia.

petname = “Alfalfa”; // NIEPOPRAWNE!

cout << petname; // Program nigdy tu nie dojdzie

return;

}
```

Ponieważ nazwa zwierzaka nie została przypisana w momencie zadeklarowania tablicy znaków, nie można jej później przypisać wartości. Następujące jest dozwolone, ponieważ możesz przypisać wartości pojedynczo do tablicy znaków:

```
#include <iostream.h>

main()

{

char petname[20]; // Rezerwuj przestrzeń dla nazwy zwierzęcia.

petname[0]=’A’; // Przypisz wartości po jednym elemencie na raz

petname[1]=’l’;

petname[2]=’f’;

petname[3]=’a’;

petname[4]=’l’;

petname[5]=’f’;

petname[6]=’a’;

petname[7]=’\0'; // Potrzebny do zapewnienia, że jest to ciąg znaków!

cout <<petname; // Teraz imię zwierzaka drukuje się poprawnie

return;
```

}

Tablica znaków petname zawiera teraz ciąg znaków, ponieważ ostatni znak jest zerowy. Jak długi jest ciąg w petname? Ma siedem znaków, ponieważ długość łańcucha nigdy nie zawiera zerowego zeru. Nie możesz przypisać więcej niż 20 znaków do tej tablicy, ponieważ zarezerwowane miejsce ma tylko 20 znaków. Jednak możesz przechowywać dowolny ciąg 19 znaków (pozostawiając jeden dla zerowego zera) lub mniej znaków w tablicy. Jeśli przypiszesz ciąg znaków "Alfalfa" w tablicy, jak pokazano, a następnie przypisz zero do pseudonimu [3], jak w:

petname [3] = '\ 0';

ciąg w petname ma teraz tylko trzy znaki. W efekcie skrócono ciąg znaków. Pozostało jeszcze 20 znaków zarezerwowanych dla nazwy, ale dane w nim zawarte to ciąg "Alf" zakończony zerowym zerem. Istnieje wiele innych sposobów przypisania wartości do łańcucha. Możesz na przykład użyć funkcji strcpy(). Jest to wbudowana funkcja, która umożliwia kopiowanie literału ciągu znaków w ciągu znaków. Aby skopiować nazwę zwierzaka "Alfalfa" do tablicy pseudonimów, wpisz:

strcpy (petname, "Lucerna"); // Kopiuje alfalfę do tablicy.

Funkcja strcpy () ("kopia smyczkowa") zakłada, że pierwsza wartość w nawiasach jest nazwą tablicy znaków, a druga wartość jest poprawną literałem ciągu lub inną tablicą znaków, która przechowuje ciąg znaków. Musisz mieć pewność, że pierwsza tablica znaków w nawiasach jest wystarczająco długa (w liczbie zarezerwowanych elementów), aby pomieścić dowolny ciąg do niej skopiowany.

Przykłady

1. Załóżmy, że chcesz śledzić imię swojej ciotki w programie, abyś mógł ją wyświetlić. Jeśli twoja ciocia ma na imię Ruth Ann Cooper, musisz zarezerwować co najmniej 16 elementów -15, aby zachować imię i nazwisko, a drugie trzymać znak null. Następująca instrukcja odpowiednio rezerwuje tablicę znaków, aby zachować jej imię i nazwisko:

char aunt_name [16];

2. Jeśli chcesz umieścić nazwę swojej cioci w tablicy w tym samym czasie, w którym zarezerwujesz przestrzeń tablicową, możesz to zrobić tak:

char aunt_name [16] = "Ruth Ann Cooper";

Możesz także pominąć rozmiar tablicy i pozwolić C++ na policzenie potrzebnej liczby:

char aunt_name [] = "Ruth Ann Cooper";

3. Załóżmy, że chcesz śledzić nazwiska trzech przyjaciół. Najdłuższa nazwa to 20 znaków (w tym zerowe zero). Musisz po prostu zarezerwować wystarczającą ilość przestrzeni tablicy znaków, aby pomieścić imię każdego znajomego. Poniższy kod wykonuje sztuczkę:

char friend1 [20];

char friend2 [20];

char friend3 [20];

Deklaracje macierzy powinny pojawić się w górnej części bloku, wraz z dowolnymi zmiennymi liczb całkowitymi, zmiennoprzecinkowymi lub znakowymi, które należy zadeklarować.

4. Następny przykład prosi użytkownika o imię i nazwisko. Użyj operatora cin (przeciwieństwo cout), aby pobrać dane z klawiatury. Następnie program wypisuje inicjały użytkownika na ekranie, drukując pierwszy znak każdej nazwy w tablicy. Program musi wydrukować indeks dolny każdej tablicy, ponieważ pierwszy indeks każdej tablicy zaczyna się od 0, a nie 1.

```
// Nazwa pliku: C5INIT.CPP

// Wydrukuj inicjały użytkownika.

#include <iostream.h>

main()

{

char first [20]; // Zatrzymuje imię

char last [20]; // Zatrzymuje nazwisko

cout << "Jak masz na imię? \ n ";

cin >>first;

cout << "Jak masz na nazwisko? \ n ";

cin >> last;

// Wydrukuj inicjały

cout << "Twoje inicjały to" << pierwszy [0] << ""

<< ostatnia [0];

return 0;

}
```

5. Poniższy program pobiera tablice znaków trzech znajomych i przypisuje je wartościom łańcuchów za pomocą trzech metod przedstawionych w tej części. Zwróć uwagę na dodatkowy plik #include użyty w funkcji string strcpy ().

```
// Nazwa pliku: C5STR.CPP

// Przechowuj i zainicjuj trzy tablice znaków dla trzech przyjaciół

#include <iostream.h>

#include <string.h>

main()

{

// Deklaruj wszystkie tablice i zainicjuj pierwszy.

char friend1 [20] = "Jackie Paul Johnson";

char friend2 [20];

char friend3 [20];
```

```
// Użyj funkcji, aby zainicjować drugą tablicę.
strcpy (friend2, "Julie L. Roberts");
friend3 [0] = "A"; // Inicjalizuj ostatni,
friend3 [1] = 'd'; // element na raz.
friend3 [2] = 'a';
friend3 [3] = 'm';
friend3 [4] = '';
friend3 [5] = 'G';
friend3 [6] = '.';
friend3 [7] = '';
friend3 [8] = 'S';
friend3 [9] = 'm';
friend3 [10] = 'i';
friend3 [11] = 't';
friend3 [12] = 'h';
friend3 [13] = '\ 0';
// Wyświetlaj wszystkie trzy nazwiska.
cout << przyjaciel1 << "\ n";
cout << friend2 << "\ n";
cout << przyjaciel3 << "\ n";
return 0;
}
```

Ostatnia metoda inicjowania tablicy znaków za pomocą string - jeden element na raz - nie jest używany tak często, jak inne metody.

Dyrektywy preprocesora

Jak zapewne pamiętacie, kompilator C ++ kieruje programy przez preprocesor, zanim je skompiluje. Preprocesor można nazwać „prekompilatorem”, ponieważ wstępnie przetwarza i przygotowuje kod źródłowy do kompilacji, zanim kompilator go odbierze. Ponieważ ten proces wstępny jest tak ważny dla C ++, powinieneś się z nim zapoznać przed nauczeniem się bardziej specjalistycznych poleceń w języku. Zwykłe polecenia C ++ nie wpływają na preprocesor. Musisz dostarczyć specjalne polecenia inne niż C ++, zwane dyrektywami preprocesora, aby kontrolować preprocesor. Dyrektywy te pozwalają na przykład modyfikować kod źródłowy, zanim kod dotrze do kompilatora. Prawie każdy właściwy program C ++ zawiera dyrektywy preprocesora. Ta część uczy dwóch najczęściej występujących: #include i #define.

Zrozumienie dyrektyw preprocesora

Dyrektywy preprocesora to polecenia dostarczane do preprocesora. Wszystkie dyrektywy preprocesora zaczynają się od znaku funta (#). Nigdy nie umieszczaj średnika na końcu dyrektyw preprocesora, ponieważ są to komendy preprocesora, a nie komendy C ++. Dyrektywy preprocesora zazwyczaj zaczynają się w pierwszej kolumnie twojego programu źródłowego. Mogą oczywiście zaczynać się w dowolnej kolumnie, ale powinieneś starać się zachować zgodność ze standardową praktyką i zacząć je w pierwszej kolumnie, gdziekolwiek się pojawią. Dyrektywy preprocesora powodują, że preprocesor C ++ zmienia kod źródłowy, ale zmiany te trwają tylko tak długo, jak kompilacja. Gdy ponownie spojrzysz na kod źródłowy, preprocesor kończy pracę z plikiem, a jego zmiany nie są już w nim zawarte. Twój preprocesor w żaden sposób nie kompiluje twojego programu ani nie zmienia twoich aktualnych poleceń C ++. Ta koncepcja dezorientuje niektórych początkujących studentów C ++, ale pamiętaj tylko, że twój program jeszcze się nie skompilował, kiedy wykonasz instrukcje preprocesora. Powiedziano, że preprocesor jest niczym więcej niż edytorem tekstu w twoim programie.

Dyrektywa #include

Dyrektywa #include preprocesora łączy plik dyskowy z programem źródłowym. Pamiętaj, że dyrektywa preprocesora nie robi nic więcej niż polecenie edytora tekstu dla twojego programu; edytory tekstu mogą także łączyć pliki. Format dyrektywy preprocesora #include jest następujący:

#include <nazwa pliku>

lub

#include „nazwa pliku”

W dyrektywie #include nazwa pliku musi być plikiem tekstowym ASCII (tak jak plik źródłowy) i znajdować się gdzieś na dysku. Aby lepiej zilustrować tę regułę, może pomóc na chwilę opuścić C ++. Poniższy przykład pokazuje zawartość dwóch plików na dysku. Jeden nazywa się OUTSIDE, a drugi INSIDE. Oto zawartość pliku OUTSIDE:

Teraz jest czas dla wszystkich dobrych ludzi

#include <INSIDE>

przyjść z pomocą ich krajowi

Plik INSIDE zawiera następujące elementy:

Szybki brązowy lis podskoczył

nad leniwym psem.

Załóżmy, że możesz uruchomić plik OUTSIDE przez preprocesor C++, który znajduje dyrektywę #include i zastępuje ją całym plikiem o nazwie INSIDE. Innymi słowy, dyrektywa preprocesora C ++ łączy plik INSIDE z plikiem OUTSIDE - w lokalizacji #include - a OUTSIDE rozwija się, aby dołączyć scalony tekst. Po zakończeniu wstępnego przetwarzania OUTSIDE wygląda następująco:

Teraz jest czas dla wszystkich dobrych ludzi

Szybki brązowy lis podskoczył

nad leniwym psem.

przyjść z pomocą ich krajowi.

Plik INSIDE pozostaje na dysku w oryginalnej formie. Zmienia się tylko plik zawierający dyrektywę #include. Ta zmiana jest tylko tymczasowa; to znaczy, OUTSIDE jest rozwijany tylko o dołączony plik tak długo, jak trzeba skompilować program. Pomocnych może być kilka przykładów z życia, ponieważ pliki OUTSIDE i INSIDE nie są programami w C ++. Możesz dołączyć plik zawierający wspólny kod, którego często używasz. Załóżmy, że często drukujesz swoje nazwisko i adres. Możesz wpisać kilka wierszy kodu w każdym programie, który wypisuje twoje imię i adres:

cout << „Kelly Jane Peterson \ n”;

cout << „Mieszkanie nr 217 \ n”;

cout << „4323 East Skelly Drive \ n”;

cout << „Nowy Jork, Nowy Jork \ n”;

cout << „10012 \ n”;

Zamiast ponownie wpisywać te same pięć wierszy, wpisujesz je raz i zapisujesz w pliku o nazwie MYADD.C. Od tego momentu wystarczy wpisać tylko jedną linię:

#include <myadd.c>

Pozwala to nie tylko oszczędzać na pisaniu, ale także zapewnia spójność i dokładność. (Czasami ten rodzaj powtarzanego tekstu jest znany jako). Zazwyczaj można użyć nawiasów kątowych, < > lub podwójnych cudzysłowów, „”, wokół dołączonej nazwy pliku z tymi samymi wynikami. Nawiasy kątowe informują preprocesora, aby szukał pliku dołączania w domyślnym katalogu dołączania, skonfigurowanym przez kompilator. Podwójne znaki cudzysłowu informują preprocesora, aby najpierw szukał pliku włączeń w katalogu, w którym przechowywany jest kod źródłowy, a następnie szukał pliku włączania w systemie. Przez większość czasu widać dołączone nawiasy kwadratowe wokół dołączonej nazwy pliku. Jeśli chcesz uwzględnić sekcje kodu w innych programach, pamiętaj o zapisaniu tego kodu w katalogu dołączeń systemu (jeśli używasz nawiasów kątowych). Mimo że #include działa dobrze dla wstawionego kodu źródłowego, istnieją inne sposoby włączenia wspólnego kodu źródłowego, które są bardziej wydajne. Ten przykładowy kod źródłowy #include dobrze służy do wyjaśnienia, co robi dyrektywa preprocesora #include. Pomimo tego, #include rzadko jest używany do dołączania tekstu kodu źródłowego, ale częściej jest używany do dołączania specjalnych plików systemowych zwanych plikami nagłówkowymi. Te pliki systemowe pomagają C ++ interpretować wiele wbudowanych funkcji, których używasz. Twój kompilator C ++ ma własne pliki nagłówkowe. Gdy ty (lub administrator systemu) zainstalowałeś kompilator C ++, te pliki nagłówkowe były automatycznie zapisywane na dysku twardym w katalogu włączenia systemu. Ich nazwy plików zawsze kończą się na

.h, aby odróżnić je od zwykłego kodu źródłowego C++. Najpopularniejszy plik nagłówka nosi nazwę iostream.h. Ten plik dostarcza kompilatorowi C++ potrzebnych informacji na temat wbudowanych operatorów cout i cin, a także innych przydatnych wbudowanych procedur, które wykonują wejście i wyjście. Nazwa „iostream.h" oznacza input / nagłówek strumienia wyjściowego. W tym momencie nie musisz rozumieć pliku iostream.h. Musisz tylko umieścić ten plik przed main() w każdym pisanym programie. Rzadko zdarza się, aby program C++ nie potrzebował pliku iostream.h. Nawet gdy plik nie jest potrzebny, w tym nie szkodzi. Twoje programy mogą działać bez iostream.h, o ile nie używają zdefiniowanego tam operatora wejścia lub wyjścia. Niemniej jednak twoje programy są dokładniejsze, a ukryte błędy wychodzą na powierzchnię znacznie szybciej, jeśli dołączysz ten plik. Za każdym razem, gdy opisywana jest nowa funkcja wbudowana, dołączany jest pasujący plik nagłówka funkcji. Ponieważ prawie każdy program w C ++, który piszesz, zawiera skrypt do wydrukowania na ekranie, prawie każdy program zawiera następujący wiersz:

Dołącz wbudowany plik nagłówka C ++ o nazwie iostream.h.

#include <iostream.h>

Widziałeś funkcję strcpy (). Plik nagłówka nosi nazwę string.h. Dlatego jeśli piszesz program zawierający strcpy (), dołącz odpowiedni plik nagłówka w tym samym czasie, co <iostream.h>. Są one wyświetlane w osobnych wierszach, takich jak:

#include <iostream.h>

#include <string.h>

Kolejność plików dołączanych nie ma znaczenia, o ile dołączasz pliki przed funkcjami, które ich potrzebują. Większość programistów C ++ dołącza wszystkie potrzebne pliki nagłówkowe przed main (). Te pliki nagłówkowe są po prostu plikami tekstowymi. Jeśli chcesz, znajdź plik nagłówkowy, taki jak stdio.h na dysku twardym i spójrz na niego. W tym momencie plik może wydawać się skomplikowany, ale nie ma w nim nic „ukrytego". Nie zmieniaj pliku nagłówka w żaden sposób, patrząc na niego. W takim przypadku konieczne może być ponowne załadowanie kompilatora w celu przywrócenia pliku.

Przykłady

1. Poniższy program jest krótki. Obejmuje to procedurę drukowania nazwy i adresu opisaną wcześniej. Po wydrukowaniu nazwy i adresu kończy się.

// Nazwa pliku: C6INC1.CPP

// Ilustruje dyrektywy preprocesora #include.

#include <iostream.h>

main()

{

#include „myadd.c"

return 0;

}

Podwójne cudzysłowy są używane, ponieważ plik o nazwie MYADD.C jest przechowywany w tym samym katalogu, co plik źródłowy. Pamiętaj, że jeśli wpiszesz ten program na komputerze (po wpisaniu i zapisaniu pliku MYADD.C), a następnie skompilujesz program, plik MYADD.C zostanie dołączony tylko

tak długo, jak długo będzie trwało skompilowanie programu. Twój kompilator nie widzi tego pliku. Twój kompilator działa tak, jakby wpisano następujące informacje:

```
// Nazwa pliku: C6INCL1.CPP

// Ilustruje dyrektywę preprocesora #include.

#include <iostream.h>

main()

{

cout („Kelly Jane Peterson \ n”;

cout („Mieszkanie nr 217 \ n”;

cout („4323 East Skelly Drive \ n”;

cout („Nowy Jork, Nowy Jork \ n”;

cout („10012 \ n”;

return 0;

}
```

To wyjaśnia, co należy rozumieć przez preprocesor: zmiany są wprowadzane w kodzie źródłowym przed jego skompilowaniem. Oryginalny kod źródłowy zostanie przywrócony, gdy tylko kompilacja zostanie zakończona. Kiedy spojrzysz ponownie na swój program, wygląda on tak, jakby został pierwotnie napisany, z instrukcją #include.

2. Poniższy program kopiuje komunikat do tablicy znaków i drukuje go na ekranie. Ponieważ używane są funkcje wbudowane cout i strcpy (), oba ich pliki nagłówkowe są dołączone.

```
// Nazwa pliku: C6INCL3.CPP

// Używa dwóch plików nagłówkowych.

#include <iostream.h>

#include <string.h>

main()

{

wiadomość char [20];

strcpy (komunikat „To jest fajne!”);

cout << wiadomość;

return 0;

}
```

Dyrektywa #define

Dyrektywa #define preprocesora jest używana w programowaniu w C++, chociaż nie tak często jak w C. Ze względu na słowo kluczowe const (w C++), które pozwala definiować zmienne jako stałe, #define nie jest używane tak często w C++. Niemniej jednak #define jest przydatne dla kompatybilności z programami C, które konwertujesz na C++. Dyrektywa #define może początkowo wydawać się dziwna, ale jest podobna do komendy wyszukiwania i zamiany w edytorze tekstu. Format #define jest następujący:

#define ARGUMENT1 argument2

gdzie ARGUMENT1 to pojedyncze słowo bez spacji. Zastosuj te same reguły nazewnictwa dla pierwszego argumentu instrukcji #define jak dla zmiennych . W przypadku pierwszego argumentu tradycyjnie stosuje się wielkie litery - jedno z niewielu zastosowań wielkich liter w całym języku C++. Co najmniej jedna spacja oddziela ARGUMENT1 od argumentu 2. Argument2 może być dowolnym znakiem, słowem lub frazą; może także zawierać spacje lub cokolwiek innego, co możesz wpisać na klawiaturze. Ponieważ #define jest dyrektywą preprocesora, a nie komendą C ++, nie umieszczaj średnika na końcu jego wyrażenia. Dyrektywa #define preprocesora zastępuje występowanie ARGUMENT1 wszędzie w twoim programie zawartością argumentu2. W większości przypadków dyrektywa #define powinna poprzedzać main () (wraz z wszelkimi dyrektywami #include). Spójrz na następującą dyrektywę #define:

Zdefiniuj literał AGELIMIT na 21.

#define AGELIMIT 21

Jeśli twój program zawiera jedno lub więcej wystąpień terminu AGELIMIT, preprocesor zastępuje każde z nich numerem 21. Kompilator reaguje tak, jakbyś rzeczywiście wpisał 2 zamiast AGELIMIT, ponieważ preprocesor zmienia wszystkie wystąpienia AGELIMIT na 21, zanim kompilator odczyta kod źródłowy. Ale znowu zmiana jest tylko tymczasowa. Po skompilowaniu programu zobaczysz go tak, jak go napisałeś, z #define i AGELIMIT nadal nienaruszone. AGELIMIT nie jest zmienną, ponieważ zmienne są deklarowane i przypisywane wartościom tylko w momencie kompilacji i uruchomienia programu. Preprocesor zmienia plik źródłowy przed jego skompilowaniem. Być może zastanawiasz się, dlaczego miałbyś kiedykolwiek zadawać sobie tyle trudu. Jeśli chcesz mieć 21 wszędzie, gdzie występuje AGELIMIT, możesz wpisać 21 na początek! Ale zaletą używania #define zamiast literałów jest to, że jeśli granica wieku kiedykolwiek się zmieni (być może do 18), musisz zmienić tylko jedną linię w programie, a nie każde wystąpienie literału 21. Ponieważ #define pozwala ci łatwo w celu zdefiniowania i zmiany literałów zastąpione argumenty dyrektywy #define są czasami nazywane zdefiniowanymi literałami. (Programiści C twierdzą, że #define „definiuje stałe”, ale programiści C ++ rzadko używają słowa „stała”, chyba że omawiają użycie stałej.) Możesz zdefiniować dowolny rodzaj literału, w tym literały łańcuchowe. Poniższy program zawiera zdefiniowany literał ciągu, który zastępuje ciąg w dwóch miejscach.

```
// Filename: C6DEF1.CPP

// Definiuje literał ciąg i używa go dwa razy.

#include <iostream.h>

#define MYNAME “Phil Ward”

main()

{
```

```
char name[]=MYNAME;

cout << “My name is “ << name << “\n”; // Prints the array.

cout << “My name is “ << MYNAME << “\n”; // Prints the

// defined literal.

return 0;

}
```

Pierwszy argument #define jest pisany wielkimi literami, aby odróżnić go od nazw zmiennych w programie. Zmienne są zwykle pisane małymi literami. Chociaż twój preprocesor i kompilator nie pomieszają tych dwóch, inni użytkownicy, którzy patrzą na twój program, mogą szybciej skanować i stwierdzić, które elementy są zdefiniowanymi literałami, a które nie. Będą wiedzieć, kiedy zobaczą wielkie litery (jeśli przestrzegasz zalecanego standardu dla tego pierwszego argumentu #define), aby spojrzeć na na górę programu pod kątem jego rzeczywistej zdefiniowanej wartości. Fakt, że zdefiniowane literały nie są zmiennymi, jest jeszcze bardziej wyraźne w następującym programie. Ten program wypisuje pięć wartości. Spróbuj odgadnąć, jakie są te pięć wartości, zanim spojrzysz na odpowiedź po programie.

```
// Filename: C6DEF2.CPP

// Ilustruje, że #define literały nie są zmiennymi.

#include <iostream.h>

#define X1 b+c

#define X2 X1 + X1

#define X3 X2 * c + X1 - d

#define X4 2 * X1 + 3 * X2 + 4 * X3

main()

{

int b = 2; // Deklaruje i inicjuje cztery zmienne

int c = 3;

int d = 4;

int e = X4;

// Prints the values.

cout << e << “, “ << X1 << “, “ << X2;

cout << “, “ << X3 << “, “ << X4 << “\n”;

return 0;

}
```

Dane wyjściowe z tego programu to

44 5 10 17 44

Gdybyś traktował X1, X2, X3 i X4 jako zmienne, nie uzyskałbyś poprawnych odpowiedzi. X1 do X4 nie są zmiennymi; są zdefiniowanymi literałami. Przed skompilowaniem programu preprocesor odczytuje pierwszy wiersz i zmienia każde wystąpienie X1 na b + c. Dzieje się to przed przetworzeniem następnego # zdefiniowania. Dlatego po pierwszym #define kod źródłowy wygląda następująco:

```
// Filename: C6DEF2.CPP

// Ilustruje, że #define literały nie są zmiennymi.

#include <iostream.h>

#define X2 b+c + b+c

#define X3 X2 * c + b+c - d

#define X4 2 * b+c + 3 * X2 + 4 * X3

main()

{

int b=2; // Deklaruje i inicjuje cztery zmienne.

int c=3;

int d=4;

int e=X4;

// Prints the values.

cout << e << “, “ << b+c << “, “ << X2;

cout << “, “ << X3 << “, “ << X4 << “\n”;

return 0;

}
```

Po zakończeniu pierwszego #define, drugi przejmuje i zmienia każde wystąpienie X2 na b + c + b + c. Twój kod źródłowy w tym momencie staje się:

```
// Filename: C6DEF2.CPP

// Ilustruje, że #define literały nie są zmiennymi.

#include <iostream.h>

#define X3 b+c + b+c * c + b+c - d

#define X4 2 * b+c + 3 * b+c + b+c + 4 * X3

main()

{

int b=2; // Deklaruje i inicjuje cztery zmienne.
```

```
int c=3;

int d=4;

int e=X4;

// Prints the values.

cout << e << “, “ << b+c << “, “ << b+c + b+c;

cout << “, “ << X3 << “, “ << X4 << “\n”;

return 0;

}
```

Po zakończeniu drugiego #define, trzeci przejmuje i zmienia każde wystąpienie X3 na b + c + b + c * c + b + c - d. Twój kod źródłowy staje się wtedy:

```
// Filename: C6DEF2.CPP

// Ilustruje, że #define literały nie są zmiennymi.

#include <iostream.h>

#define X4 2 * b+c + 3 * b+c + b+c + 4 * b+c + b+c * c + b+c - d

main()

{

int b=2; // Deklaruje i inicjuje cztery zmienne.

int c=3;

int d=4;

int e=X4;

// Prints the values.

cout << e << “, “ << b+c << “, “ << b+c + b+c;

cout << “, “ << b+c + b+c * c + b+c - d

<< “, “ << X4 << “\n”;

return 0;

}
```

Kod źródłowy szybko rośnie! Po trzecim #define kończy się czwarty i ostatni i zmienia każde wystąpienie X4 na 2 * b + c + 3 * b + c + b + c + 4 * b + c + b + c * c + b + c - d. Twój kod źródłowy w tym ostatnim punkcie staje się

```
// Filename: C6DEF2.CPP

// Ilustruje, że #define literały nie są zmiennymi.

#include <iostream.h>
```

```
main()

{

int b=2; // Deklaruje i inicjuje cztery zmienne.

int c=3;

int d=4;

int e=2 * b+c + 3 * b+c + b+c + 4 * b+c + b+c * c + b+c - d;

// Prints the values.

cout << e << “, “ << b+c << “, “ << b+c + b+c;

cout << “, “ << b+c + b+c * c + b+c - d

<< “, “ << 2 * b+c + 3 * b+c + b+c + 4 * b+c +

b+c * c + b+c - d << “\n”;

return 0;

}
```

To właśnie czyta twój kompilator. Nie wpisałeś tego kompletnego wpisu; wpisałeś oryginalny wpis (pokazany jako pierwszy). Preprocesor rozszerzył kod źródłowy do tej dłuższej formy, tak jakbyś wpisał go w ten sposób. Jest to skrajny przykład, ale służy zilustrowaniu działania #define w kodzie źródłowym i nie definiuje żadnych zmiennych. #Define zachowuje się jak polecenie wyszukiwania i zamiany edytora tekstu. Ze względu na zachowanie #-Define, możesz nawet przepisać język C ++! Jeśli jesteś przyzwyczajony do języka BASIC, możesz wygodniej pisać PRINT zamiast w C ++, gdy chcesz drukować na ekranie. Jeśli tak, to następująca instrukcja #define,

#define PRINT cout umożliwia drukowanie w C ++ za pomocą następujących instrukcji:

PRINT << “This is a new printing technique\n”;

PRINT << “I could have used cout instead.”\n;

Działa to, ponieważ zanim kompilator odczyta program, odczyta tylko następujące elementy:

cout << „Jest to nowa technika drukowania \ n”;

cout << „Zamiast tego mogłem użyć cout.” \ n;

W kolejnej części, dowiesz się o dwóch funkcjach czasami używanych do wprowadzania i wysyłania danych, zwanych printf() i scanf(). Możesz równie łatwo przedefiniować nazwy funkcji za pomocą #define, jak w przypadku cout. Pamiętaj również, że nie można zastąpić zdefiniowanego literału, jeśli znajduje się on w innym literałowym łańcuchu znaków. Na przykład nie można użyć następującej instrukcji #define:

#define AGE

aby zamienić informacje w tym cout:

cout << „WIEK”;

ponieważ AGE jest literałem ciągowym i drukuje dosłownie tak, jak pojawia się w podwójnym cudzysłowie. Preprocesor może zastępować tylko zdefiniowane literały, które nie pojawiają się w cudzysłowie.

Nie przesadzaj z #define

Wielu wczesnych programistów C lubiło redefiniować części języka, aby pasowały do tego, do czego były przyzwyczajone w innym języku. Przykład cout to PRINT był tylko jednym przykładem tego. Możesz przedefiniować praktycznie każdą instrukcję lub funkcję C ++, aby „wyglądała" w dowolny sposób. Istnieje jednak niebezpieczeństwo, dlatego należy zachować ostrożność przy użyciu #define w tym celu. Przedefiniowanie języka staje się mylące dla innych, którzy zmodyfikują Twój program później. Ponadto, gdy zapoznasz się z C ++, będziesz naturalnie coraz częściej używać prawdziwego języka C ++. Kiedy czujesz się dobrze w C ++, starsze programy, które przedefiniowałeś, będą mylące - nawet dla ciebie! Jeśli programujesz w C ++, użyj konwencji językowych, które zapewnia C ++. Nie należy próbować redefiniować poleceń w języku. Pomyśl o dyrektywie #define jako sposobie definiowania literałów liczbowych i łańcuchowych. Jeśli te literały kiedykolwiek się zmienią, musisz zmienić tylko jeden wiersz w swoim programie.

„Po prostu powiedz nie" pokusie przedefiniowania poleceń i wbudowanych funkcji. Jeszcze lepiej, zmodyfikuj dowolny starszy kod C, który używa #define, i zastąp dyrektywę #define preprocesora bardziej użyteczną komendą const.

Przykłady

1. Załóżmy, że chcesz śledzić docelową kwotę sprzedaży swojej firmy w wysokości 55 000,00 USD. Ta kwota docelowa nie zmieniła się w ciągu ostatnich dwóch lat. Ponieważ prawdopodobnie wkrótce się to nie zmieni (sprzedaż jest płaska), decydujesz się na użycie zdefiniowanego literału do przedstawienia tej kwoty docelowej. Następnie, jeśli sprzedaż docelowa ulegnie zmianie, wystarczy zmienić kwotę w wierszu #define na:

#define TARGETSALES 55000.00

która definiuje literał zmiennoprzecinkowy. Następnie możesz przypisać TARGETSALES do zmiennych zmiennoprzecinkowych i wydrukować ich wartość, tak jakbyś wpisał 55000.00 w całym programie, ponieważ te linie pokazują:

amt = TARGETSALES

cout << TARGETSALES;

2. Jeśli definiujesz te same literały w wielu programach, zapisz literały na dysku i dołącz je. Nie musisz więc wpisywać zdefiniowanych literałów na początku każdego programu. Jeśli przechowujesz te literały w pliku MYDEFS.C w katalogu swojego programu, możesz dołączyć ten plik za pomocą następującej instrukcji #include:

#include „mydefs.c"

(Aby użyć nawiasów kątowych, musisz zapisać plik w katalogu dołączonym systemu).

3. Zdefiniowane literały są odpowiednie dla rozmiarów tablic. Załóżmy na przykład, że deklarujesz tablicę dla nazwy klienta. Pisząc program, wiesz, że nie masz klienta, którego nazwa jest dłuższa niż 22 znaki (w tym null). Dlatego możesz to zrobić:

#define CNMLENGTH 22

Po zdefiniowaniu tablicy możesz użyć tego:

char cust_name [CNMLENGTH]

Można również użyć innych instrukcji wymagających rozmiaru tablicy

CNMLENGTH.

4. Wielu programistów C ++ definiuje listę komunikatów o błędach. Po zdefiniowaniu wiadomości za pomocą łatwej do zapamiętania nazwy mogą wydrukować te literały, jeśli wystąpi błąd, i nadal utrzymywać spójność swoich programów. Następujące komunikaty o błędach (lub podobnej formie) często pojawiają się na początku programów w C ++.

#define DISKERR “Wygląda na to, że twój napęd dyskowy nie działa”

#define PRNTERR “Twoja drukarka nie odpowiada”

#define AGEERR “„Nie możesz wejść w tak mały wiek”

#define NAMEERR “Musisz podać pełne imię i nazwisko”

Proste wejście / wyjście

Widziałeś już operator cout. Drukuje wartości na ekranie. cout jest czymś więcej niż się nauczyłeś. Używając cout i ekranu (najpopularniejsze urządzenie wyjściowe), możesz drukować informacje w dowolny sposób. Twoje programy stają się również znacznie wydajniejsze, jeśli nauczysz się otrzymywać dane z klawiatury. cin jest operatorem, który odzwierciedla cout. Zamiast wysyłać wartości wyjściowe na ekran, cin akceptuje wartości, które użytkownik wpisuje na klawiaturze. Operatory cout i cin oferują nowe operatory wejścia i wyjścia programatora C ++, których mogą używać z względną łatwością. Oba te operatory mają ograniczony zakres, ale dają ci możliwość wysyłania danych wyjściowych i odbierania danych wejściowych do programów. Istnieją odpowiednie funkcje dostarczane ze wszystkimi wywoływanymi kompilatorami C ++, printf () i scanf (). Te funkcje są nadal używane przez programistów C ++ ze względu na ich powszechne stosowanie w zwykłych programach C.

Będziesz zaskoczony, o ile bardziej zaawansowane mogą być twoje programy po nauczeniu się tych operatorów wejścia / wyjścia.

Operator cout

Operator cout wysyła dane do standardowego urządzenia wyjściowego. Standardowym urządzeniem wyjściowym jest zwykle ekran; możesz jednak przekierować standardowe wyjście na inne urządzenie. Jeśli nie znasz przekierowań urządzeń na poziomie systemu operacyjnego, nie martw się, dowiesz się więcej o tym w tej książce. W tym momencie cout wysyła wszystkie dane wyjściowe na ekran. Format cout jest inny niż w innych poleceniach C++ . Format cout to

cout << dane [<< dane];

Symbolem zastępczym danych mogą być zmienne, literały, wyrażenia lub kombinacja wszystkich trzech.

Wyświetlanie ciągów

Aby wyświetlić stałą ciągu, po prostu wpisz stałą ciągu za operatorem cout. Na przykład, aby wydrukować ciąg „Deszcz w Hiszpanii”, wystarczy wpisać:

Wyświetl na ekranie zdanie „Deszcz w Hiszpanii”.

cout << „Deszcz w Hiszpanii”;

Musisz jednak pamiętać, że cout nie wykonuje automatycznego powrotu karetki. Oznacza to, że pojawi się kursor na ekranie

bezpośrednio po ostatnim drukowanym znaku i kolejnych pseudonimach. Aby lepiej zrozumieć tę koncepcję, spróbuj przewidzieć wyniki następujących trzech operatorów cout:

count << “Line 1”;

cout << “Line 2”;

cout << “Line 3”;

Operatorzy ci wytwarzają następujące dane wyjściowe:

Linia 1 Linia 2 Linia 3

co prawdopodobnie nie jest tym, co zamierzałeś. Dlatego musisz dołączyć znak nowej linii, \ n, ilekroć chcesz przenieść kursor do następnego wiersza. Następujący trzej operatorzy cout wytwarzają trzywierszowe dane wyjściowe:

cout << „Linia 1 \ n”;

cout << „Linia 2 \ n”;

cout << „Linia 3 \ n”;

Wyjście z tych cout to

Linia 1

Linia 2

Linia 3

Znak \n wysyła kursor do następnego wiersza bez względu na to, gdzie go wstawisz. Następujące trzy operatory cout również wytwarzają poprawne wyjście trzywierszowe:

cout << „Linia 1”;

cout << „\ nLine 2 \ n”;

cout „Line 3”;

Drugi kod drukuje nowy wiersz, zanim drukuje cokolwiek innego. Następnie drukuje ciąg znaków, a następnie kolejną nową linię. Trzeci ciąg jest drukowany w trzecim wierszu. Możesz także wydrukować ciągi znaków zapisane w tablicach znaków, wpisując nazwę tablicy wewnątrz cout. Jeśli miałbyś przechowywać swoje imię w tablicy zdefiniowanej jako:

char my_name [] = „Lyndon Harris”;

możesz wydrukować nazwę z następującą kreacją:

cout << moja_nazwa;

Następująca sekcja kodu drukuje trzy literały łańcuchowe w trzech różnych wierszach:

cout << „Nancy Carson \ n”;

cout << „1213 Oak Street \ n”;

cout << „Fairbanks, Alaska \ n”;

cout jest często używany do oznaczania wyników. Przed wydrukowaniem wieku, kwoty, wynagrodzenia lub jakichkolwiek innych danych liczbowych powinieneś wydrukować stałą ciągu, która mówi użytkownikowi, co oznacza liczba. Poniższy napis informuje użytkownika, że następny wydrukowany numer to wiek. Bez tego użytkownika użytkownik nie wiedziałby, co reprezentuje ten numer.

cout << „Oto wiek znaleziony w naszych plikach:”;

Możesz wydrukować pusty wiersz, drukując dwa znaki nowego wiersza, \ n, obok siebie po ciągu znaków, jak w:

cout << „Przygotuj faktury ... \ n \ n”;

Przykłady

1. Poniższy program przechowuje kilka wartości w trzech zmiennych, a następnie drukuje wyniki:

```
// Nazwa pliku: C7PRNT1.CPP
// Drukuje wartości w zmiennych.
#include <iostream.h>
main()
{
char first = 'E'; // Store some character, integer,
char middle = 'W'; // and floating-point variable.
char last = 'C';
int age = 32;
int dependents = 2;
float salary = 25000.00;
float bonus = 575.25;
// Prints the results.
cout << first << middle << last;
cout << age << dependents;
cout << salary << bonus;
return 0;
}
```

2. Ostatni program nie pomaga użytkownikowi. Dane wyjściowe nie są oznaczone i są drukowane w jednym wierszu. Oto ten sam program z kilkoma wiadomościami i kilkoma znakami nowej linii umieszczonymi tam, gdzie jest to potrzebne:

```
// Nazwa pliku: C7PRNT2.CPP
// Drukuje wartości w zmiennych z odpowiednimi etykietami.
#include <iostream.h>
main()
{
char first = 'E'; // Store some character, integer,
char middle = 'W'; // and floating-point variable.
char last = 'C';
int age = 32;
```

```
int dependents = 2;

float salary = 25000.00;

float bonus = 575.25;

// Prints the results.

cout << “Here are the initials:\n”;

cout << first << middle << last <<“\n”;

cout << “The age and number of dependents are\n”;

cout << age << “ “ << dependents << “\n\n”;

cout << “The salary and bonus are\n”;

cout << salary << ‘ ‘ << bonus;

return 0;

}
```

Dane wyjściowe tego programu są wyświetlane poniżej:

```
Oto inicjały:

ERZ

Wiek i liczba osób na utrzymaniu to

32 2

Wynagrodzenie i premia są

25000 575,25
```

Pierwsze wartości zmiennoprzecinkowe nie są drukowane z zerami, ale liczba jest poprawna. Następna sekcja pokazuje, jak ustawić liczbę zer początkowych i końcowych.

3. Jeśli musisz wydrukować tabelę liczb, możesz użyć do tego znaku tabulatora \ t. Umieść znak tabulacji między każdą z drukowanych liczb. Poniższy program wypisuje listę nazw zespołów i liczbę trafień w pierwszych trzech tygodniach sezonu:

```
// Nazwa pliku: C7TEAM.CPP

// Drukuje tabelę nazw drużyn i trafień przez trzy tygodnie.

#include <iostream.h>

main()

{

cout << “Parrots\tRams\tKings\tTitans\tChargers\n”;

cout << “3\t5\t3\t1\t0\n”;

cout << “2\t5\t1\t0\t1\n”;
```

```
cout << “2\t6\t4\t3\t0\n”;

return 0;

}
```

Ten program tworzy tabelę pokazaną poniżej. Widać, że mimo że nazwy mają różne szerokości, liczby są drukowane poprawnie pod nimi. Znak \ t wymusza następną nazwę lub wartość do następnej pozycji tabulacji (co osiem znaków).

Parrots Rams Kings Titans Chargers

3 5 3 1 0

2 5 1 0 1

2 6 4 3 0

Operatory sterujące

Widziałeś już potrzebę dodatkowej kontroli wyników programu. Wszystkie liczby zmiennoprzecinkowe są drukowane ze zbyt dużą liczbą miejsc dziesiętnych dla większości aplikacji. Co jeśli chcesz wydrukować tylko dolary i centy (dwa miejsca po przecinku) lub wydrukować średnią z jednym miejscem po przecinku?

Możesz określić, ile pozycji drukowania chcesz użyć przy drukowaniu liczby. Na przykład następujące cout wypisuje liczbę 456, używając trzech pozycji (długości danych):

```
cout << 456;
```

Gdyby 456 były przechowywane w zmiennej całkowitej, nadal używałby trzech pozycji do drukowania, ponieważ liczba wydrukowanych cyfr wynosi trzy. Możesz jednak określić liczbę pozycji do wydrukowania. Następujące cout wyświetla liczbę 456 w pięciu pozycjach (z dwoma wiodącymi spacjami):

```
cout << setw (5) << setfill („„) << 456;
```

Zazwyczaj używasz manipulatora setw, gdy chcesz wydrukować dane w jednolitych kolumnach. Pamiętaj, aby dołączyć plik nagłówka iomanip.h do wszystkich programów korzystających z manipulatorów, ponieważ iomanip.h opisuje, jak działa setw w kompilatorze. Poniższy program pokazuje znaczenie liczby szerokości. Każde wyjście cout jest opisane w komentarzu po jego lewej stronie.

```
// Nazwa pliku: C7MOD1.CPP

// Ilustruje różne modyfikatory szerokości liczb całkowitych.

#include <iostream.h>

#include <iomanip.h>

main()

{// Wynik pojawia się poniżej.

cout << 456 << 456 << 456 << „\ n”; // Drukuje 456456456

cout << setw (5) << 456 << setw (5) << 456 << setw (5) <<
```

456 << „\ n”; // Drukuje 456 456 456

cout << setw (7) << 456 << setw (7) << 456 << setw (7) <<

456 << „\ n”; // Drukuje 456 456 456

return 0;

}

Gdy używasz manipulatora setw wewnątrz znaku konwersji, C ++ wyrównuje liczbę do prawej o określonej szerokości. Gdy określisz ośmiocyfrową szerokość, C ++ drukuje wartość wewnątrz tych ośmiu cyfr, wypełniając liczbę wiodącymi odstępami, jeśli liczba nie wypełnia całej szerokości.

UWAGA: Jeśli nie określisz szerokości wystarczająco dużej, aby pomieścić liczbę, C ++ ignoruje twoje żądanie szerokości i drukuje liczbę w całości.

Możesz kontrolować szerokość ciągów w ten sam sposób za pomocą manipulatora setw. Jeśli nie określisz wystarczającej szerokości, aby wypisać pełny ciąg, C ++ ignoruje szerokość. Aplikacja listy mailingowej z tyłu tej książki wykorzystuje tę technikę do drukowania nazw na etykietach adresowych.

UWAGA: setw () staje się ważniejszy podczas drukowania liczb zmiennoprzecinkowych.

setprecision (2) wyświetla liczbę zmiennoprzecinkową z dwoma miejscami dziesiętnymi. Jeśli C ++ musi zaokrąglić część ułamkową, robi to. Następujące cout:

cout << setw (6) << setprecision (2) << 134.568767;

produkuje następujące dane wyjściowe:

134,57

Bez setw lub manipulatorów setprecision C ++ wydrukowałby:

134,568767

WSKAZÓWKA: Podczas drukowania liczb zmiennoprzecinkowych C ++ zawsze drukuje całą część po lewej stronie dziesiętnej (aby zachować jak największą dokładność) bez względu na to, ile pozycji określisz. Dlatego wielu programistów C ++ ignoruje manipulator setw dla liczb zmiennoprzecinkowych i podaje tylko precyzję, jak w setprecision (2).

Przykłady

1. Jeśli chcesz kontrolować szerokość swoich danych, użyj manipulatora setw. Poniższy program jest wersją C7TEAM.CPP pokazaną wcześniej. Zamiast używać znaku tab, \t, który jest ograniczony do ośmiu spacji, ten program używa specyfikatora szerokości do ustawiania tabulatorów. Zapewnia, że każda kolumna ma szerokość 10 znaków.

// Nazwa pliku: C7TEAMMD.CPP

// Drukuje tabelę nazw drużyn i trafień przez trzy tygodnie

// użycie znaków konwersji modyfikujących szerokość.

#include <iostream.h>

#include <iomanip.h>

```
main()
{
cout << setw(10) << “Parrots” << setw(10) <<
“Rams” << setw(10) << “Kings” << setw(10) <<
“Titans” << setw(10) << “Chargers” << “\n”;
cout << setw(10) << 3 << setw(10) << 5 <<
setw(10) << 2 << setw(10) << 1 <<
setw(10) << 0 << “\n”;
cout << setw(10) << 2 << setw(10) << 5 <<
setw(10) << 1 << setw(10) << 0 <<
setw(10) << 1 << “\n”;
cout << setw(10) << 2 << setw(10) << 6 <<
setw(10) << 4 << setw(10) << 3 <<
setw(10) << 0 << “\n”;
return 0;
}
```

2. Następujący program to program płacowy. Dane wyjściowe są wyrażone w „dolarach i centach”, ponieważ kwoty w dolarach są drukowane poprawnie z dokładnością do dwóch miejsc po przecinku.

```
// Nazwa pliku: C7PAY1.CPP
// Właściwie oblicza i drukuje dane listy płac w dolarach
// i centach.
#include <iostream.h>
#include <iomanip.h>
main()
{
char emp_name[ ] = “Larry Payton”;
char pay_date[ ] = “03/09/92”;
int hours_worked = 43;
float rate = 7.75; // Pay per hour
float tax_rate = .32; // Tax percentage rate
float gross_pay, taxes, net_pay;
```

```
// Computes the pay amount.

gross_pay = hours_worked * rate;

taxes = tax_rate * gross_pay;

net_pay = gross_pay - taxes;

// Prints the results.

cout << “As of: “ << pay_date << “\n”;

cout << emp_name << “ worked “ << hours_worked <<

“ hours\n”;

cout << “and got paid “ << setw(2) << setprecision(2)

<< gross_pay << “\n”;

cout << “After taxes of: “ << setw(6) << setprecision(2)

<< taxes << “\n”;

cout << “his take-home pay was $” << setw(8) <<

setprecision(2) << net_pay << “\n”;

return 0;

}
```

Wyjście z tego programu jest następuje. Pamiętaj, że zmienne zmiennoprzecinkowe nadal zachowują pełną precyzję (do sześciu miejsc po przecinku), tak jak w poprzednim programie. Modyfikatory manipulatorów setw wpływają tylko na sposób wyprowadzania zmiennych, a nie na to, co jest w nich przechowywane.

Stan: 03/09/92

Larry Payton pracował 43 godziny

i dostał 333,25

Po opodatkowaniu: 106,64

jego wynagrodzenie na wynos wyniosło 226,61 USD

3. Większość programistów C ++ nie używa manipulatora setw podczas drukowania dolarów i centów. Oto program płacowy, który używa metody szerokości zmiennoprzecinkowej skrótu. Zauważ, że poprzednie trzy instrukcje cout nie zawierają manipulatora setw. C ++ automatycznie drukuje pełną liczbę po lewej stronie dziesiętnej i drukuje tylko dwa miejsca po prawej stronie.

```
// Nazwa pliku: C7PAY2.CPP

// Właściwie oblicza i drukuje dane listy płac

// za pomocą modyfikatora skrótu.

#include <iostream.h>
```

```
#include <iomanip.h>
main()
{
char emp_name[ ] = “Larry Payton”;
char pay_date[ ] = “03/09/92”;
int hours_worked = 43;
float rate = 7.75; // Pay per hour
float tax_rate = .32; // Tax percentage rate
float gross_pay, taxes, net_pay;
// Computes the pay amount.
gross_pay = hours_worked * rate;
taxes = tax_rate * gross_pay;
net_pay = gross_pay - taxes;
// Prints the results.
cout << “As of: “ << pay_date << “\n”;
cout << emp_name << “ worked “ << hours_worked <<
“ hours\n”;
cout << “and got paid “ << setprecision(2) << gross_pay
<< “\n”;
cout << “After taxes of: “ << setprecision(2) << taxes
<< “\n”;
cout << “his take-home pay was “ << setprecision(2) <<
net_pay << “\n”;
return 0;
}
```

Wyniki tego programu są takie same jak w przypadku poprzedniego programu.

Operator cin

Teraz rozumiesz, w jaki sposób C ++ reprezentuje dane i zmienne, i wiesz, jak je wydrukować. Jest jeszcze jedna część programowania, której nie widziałeś: wprowadzanie danych do programów.

Do tego momentu nie wprowadzałeś danych do programu. Wszystkie dane, z którymi pracowałeś, zostały przypisane do zmiennych w programie. Jednak nie zawsze jest to najlepszy sposób przesyłania danych do programów; kiedy piszesz programy, rzadko wiesz, jakie są twoje dane. Dane są znane tylko

wtedy, gdy uruchamiasz programy (lub uruchamia je inny użytkownik). Operator cin jest jednym ze sposobów wprowadzania danych z klawiatury. Kiedy twoje programy osiągają linię z cin, użytkownik może wprowadzać wartości bezpośrednio do zmiennych. Twój program może następnie przetworzyć te zmienne i produkować produkcję.

Funkcja cin wypełnia zmienne wartościami

Istnieje duża różnica między cin a instrukcjami przypisania (np. I = 17;). Obie wartości wypełniają zmienne wartościami. Jednak instrukcja przypisania przypisała określone wartości do zmiennych w czasie programowania. Kiedy uruchamiasz program z instrukcjami przypisania, z listy programu dokładnie wiesz, jakie wartości trafiają do zmiennych, ponieważ napisałeś program specjalnie do przechowywania tych wartości. Za każdym razem, gdy uruchamiasz program, wyniki są dokładnie takie same, ponieważ te same wartości są przypisywane do tych samych zmiennych. Nie masz pojęcia, kiedy piszesz programy korzystające z cin, jakie wartości zostaną przypisane do zmiennych cin, ponieważ ich wartości nie są znane, dopóki program nie uruchomi się, a użytkownik nie wprowadzi tych wartości. Oznacza to, że masz bardziej elastyczny program, z którego może korzystać wiele osób. Za każdym razem, gdy program jest uruchamiany, tworzone są różne wyniki, w zależności od wartości wpisanych na każdym cin w programie.

cin ma swoje wady. Dlatego w kolejnych kilku częściach będziesz używać cin, dopóki nie nauczysz się bardziej wydajnych (i elastycznych) metod wprowadzania. Operator cin wygląda bardzo podobnie do cout. Zawiera jedną lub więcej zmiennych, które pojawiają się po prawej stronie nazwy operatora. Format cin to

cin >> wartość [>> wartości];

Plik nagłówkowy iostream.h zawiera informacje, których C ++ potrzebuje do użycia cin, więc dołącz go, gdy używasz cin.

UWAGA: Operator cin używa tych samych manipulatorów (setw i setprecision) jak operator cout.

Jak wspomniano wcześniej, cin stwarza kilka problemów. Operator cin wymaga, aby użytkownik wpisał dane dokładnie tak, jak oczekuje tego cin. Ponieważ nie możesz kontrolować pisania przez użytkownika, nie można tego zapewnić. Możesz chcieć, aby użytkownik wprowadził wartość całkowitą, po której następuje wartość zmiennoprzecinkowa, a wywołanie operatora cin również może się tego spodziewać, ale użytkownik może zdecydować się na wprowadzenie czegoś innego! Jeśli tak się stanie, niewiele można zrobić, ponieważ wynikowe dane wejściowe są niepoprawne, a program C ++ nie ma niezawodnej metody testowania dokładności użytkownika. Przed każdym cinem wydrukuj monit dokładnie wyjaśniający, czego oczekuje użytkownik. W następnych kilku rozdziałach możesz założyć, że użytkownik wie, jak wprowadzić prawidłowe wartości, ale w przypadku „prawdziwych" programów czytaj dalej, aby uzyskać lepsze metody otrzymywania danych wejściowych.

Przykłady

1. Jeśli potrzebujesz programu, który oblicza siedmioprocentowy podatek od sprzedaży, możesz użyć wyciągu cin do obliczenia sprzedaży, obliczenia podatku i wydrukowania wyników, jak pokazuje następujący program:

```
// Nazwa pliku: C7SLTX1.CPP

// Pytaj o kwotę sprzedaży i wydrukuj podatek od sprzedaży.

#include <iostream.h>
```

```
#include <iomanip.h>

main()

{

float total_sale; // User's sale amount goes here.

float stax;

// Display a message for the user.

cout << "What is the total amount of the sale? ";

// Receive the sales amount from user.

cin >> total_sale;

// Calculate sales tax.

stax = total_sale * .07;

cout << "The sales tax for " << setprecision(2) <<

total_sale << " is " << setprecision (2) << stax;

return 0;

}
```

Ponieważ pierwszy krzyk nie zawiera znaku nowej linii, \ n, odpowiedź użytkownika na monit pojawia się po prawej stronie znaku zapytania.

2. Wprowadzając ciągi klawiatury do tablic znaków za pomocą cin, jesteś ograniczony do otrzymywania jednego słowa na raz. Cin nie pozwala ci na wpisanie więcej niż jednego słowa w jednej tablicy znaków na raz. Poniższy program pyta użytkownika o swoje imię i nazwisko. Program musi przechowywać te dwie nazwy w dwóch różnych tablicach znaków, ponieważ cin nie może wprowadzić obu nazw jednocześnie. Następnie program drukuje nazwy w odwrotnej kolejności.

```
// Nazwa pliku: C7PHON1.CPP

// Program, który prosi o nazwę użytkownika i drukuje ją

// na ekranie, tak jak by to wyglądało w książce telefonicznej.

#include <iostream.h>

#include <iomanip.h>

main()

{

char first[20], last[20];

cout << "What is your first name? ";

cin >> first;

cout << "What is your last name? ";
```

```
cin >> last;

cout << “\n\n”; // Prints two blank lines.

cout << “In a phone book, your name would look like this:\n”;

cout << last << “, “ << first;

return 0;

}
```

3. Załóżmy, że chcesz napisać program, który zawiera prosty dodatek dla twojej siedmioletniej córki. Poniższy program monituje ją o podanie dwóch liczb. Program czeka następnie na wpisanie odpowiedzi. Po udzieleniu odpowiedzi program wyświetla poprawny wynik, aby mogła zobaczyć, jak dobrze sobie poradziła.

```
// Nazwa pliku: C7MATH.CPP

// Program pomocy dzieciom z prostym dodatkiem.

// Pytaj dziecko o dwie wartości po wydrukowaniu

// wiadomość tytułowa.

#include <iostream.h>

#include <iomanip.h>

main()

{

int num1, num2, ans;

int her_ans;

cout << “*** Math Practice ***\n\n\n”;

cout << “What is the first number? “;

cin >> num1;

cout << “What is the second number? “;

cin >> num2;

// Compute answer and give her a chance to wait for it.

ans = num1 + num2;

cout << “\nWhat do you think is the answer? “;

cin >> her_ans; // Nothing is done with this.

// Prints answer after a blank line.

cout << “\n” << num1 << “ plus “ << num2 << “ is “

<< ans << “\n\nHope you got it right!”;
```

return 0;

}

printf() i scanf()

Przed C ++ programiści C musieli polegać na wywołaniach funkcji w celu wykonania wejścia i wyjścia. Dwie z tych funkcji, printf() i scanf(), są nadal często używane w programach C ++, chociaż cout i cin mają nad nimi przewagę. printf() (jak cout) drukuje wartości na ekranie, a scanf() (jak cin) wprowadza wartości z klawiatury. printf () wymaga ciągu formatu kontrolującego, który opisuje dane, które chcesz wydrukować. Podobnie, scanf () wymaga ciągu formatu kontrolującego, który opisuje dane, które program chce otrzymywać z klawiatury.

UWAGA: cout jest zamiennikiem C ++ printf(), a cin to wymiana C++ na scanf().

Ponieważ koncentrujesz się na C ++, ta część krótko opisuje printf() i scanf(). W tej książce kilka programów korzysta z tych funkcji, aby zapoznać się z ich formatem.

printf() i scanf() nie są przestarzałe w C ++, ale ich użycie znacznie się zmniejszy, gdy programiści odejdą od C i do C ++. cout i cin nie wymagają kontrolowania ciągów opisujących ich dane; cout i cin są wystarczająco inteligentni, aby wiedzieć, jak traktować dane. Zarówno printf(), jak i scanf() są ograniczone - szczególnie scanf () - ale umożliwiają one twoim programom wysyłanie danych wyjściowych i odbieranie danych wejściowych.

Funkcja printf()

printf() wysyła dane do standardowego urządzenia wyjściowego, którym zazwyczaj jest ekran. Format printf() różni się od formatu zwykłych poleceń C ++. Wartości znajdujące się w nawiasach różnią się w zależności od drukowanych danych. Jednak z reguły obowiązuje następujący format printf():

printf (control_string [, jedna lub więcej wartości]);

Zauważ, że printf() zawsze wymaga control_string. Jest to ciąg znaków lub tablica znaków zawierająca ciąg znaków, który określa sposób drukowania pozostałych wartości (jeśli są wymienione). Wartości te mogą być zmiennymi, literałami, wyrażeniami lub kombinacją wszystkich trzech.

WSKAZÓWKA: Pomimo swojej nazwy printf () wysyła dane wyjściowe na ekran, a nie do drukarki.

Najłatwiejszymi danymi do wyświetlenia za pomocą printf () są łańcuchy. Aby wydrukować stałą ciągu, wystarczy wpisać tę stałą ciągu w funkcji printf (). Na przykład, aby wydrukować ciąg Deszcz w Hiszpanii, wystarczy wpisać następujące:

Wyświetl na ekranie wyrażenie „Deszcz w Hiszpanii”.

printf („Deszcz w Hiszpanii”);

printf(), podobnie jak cout, nie wykonuje automatycznego powrotu karetki. Kolejne printf() zaczynają się obok ostatniego drukowanego znaku. Jeśli chcesz powrotu karetki, musisz podać znak nowej linii, ponieważ:

printf („Deszcz w Hiszpanii \ n”);

Możesz wydrukować ciągi zapisane w tablicach znaków, również wpisując nazwę tablicy wewnątrz printf (). Na przykład, jeśli chcesz przechowywać swoje imię w tablicy zdefiniowanej jako:

char my_name [] = „Lyndon Harris”;

możesz wydrukować nazwę za pomocą tego printf ():

printf (moja_nazwa);

Podczas korzystania z printf () i scanf () należy dołączyć plik nagłówkowy stdio.h, ponieważ stdio.h określa sposób działania funkcji wejścia i wyjścia w kompilatorze. Poniższy program przypisuje a wiadomość w tablicy znaków, a następnie wyświetla tę wiadomość.

```
// Nazwa pliku: C7PS2.CPP

// Drukuje ciąg przechowywany w tablicy znaków.

#include <stdio.h>

main()

{

char message[] = “Please turn on your printer”;

printf(message);

return 0;

}
```

Znaki konwersji

Wewnątrz większości ciągów kontrolnych printf() znajdują się znaki konwersji. Te znaki specjalne mówią printf() dokładnie, w jaki sposób dane (po znakach) mają być interpretowane. Poniżej pokazano listę typowych znaków konwersji. Ponieważ w nawiasach printf() mogą znajdować się dane dowolnego typu, te znaki konwersji są wymagane za każdym razem, gdy drukujesz więcej niż jedną stałą ciągu. Jeśli nie chcesz drukować łańcucha, stała łańcucha musi zawierać co najmniej jeden ze znaków konwersji

%s : Ciąg znaków (do osiągnięcia zerowego zera)

%c : Znak

%d : Dziesiętna liczba całkowita

%f : Liczby zmiennoprzecinkowe

%u : Liczba całkowita bez znaku

%x: liczba szesnastkowa

%% : Drukuje znak procentu (%)

% s Ciąg znaków (do osiągnięcia zerowego zera)

% c Znak

% d Dziesiętna liczba całkowita

% f Liczby zmiennoprzecinkowe

% u Liczba całkowita bez znaku

% x liczba szesnastkowa

%% Drukuje znak procentu (%)

UWAGA: Znaki inne niż przedstawione w tabeli są drukowane dokładnie tak, jak pojawiają się w ciągu kontrolnym.

Aby wydrukować stałą numeryczną lub zmienną, należy dołączyć odpowiedni znak konwersji w ciągu kontrolnym printf (). Jeśli i, j i k są zmiennymi liczbami całkowitymi, nie można wydrukować ich za pomocą następującego printf ().

printf (i, j, k);

Ponieważ printf () jest funkcją, a nie poleceniem, ta funkcja printf () nie ma możliwości dowiedzenia się, jakiego typu są zmienne. Wyniki są nieprzewidywalne, a na ekranie możesz zobaczyć śmieci - jeśli w ogóle coś się pojawi. Podczas drukowania liczb należy najpierw wydrukować ciąg sterujący, który zawiera format tych liczb. Poniższy printf () wypisuje ciąg. Na wyjściu z tego wiersza pojawia się ciąg z liczbą całkowitą (% d) i liczbą zmiennoprzecinkową (% f) wydrukowaną wewnątrz tego ciągu.

printf („Jestem Betty, mam% d lat i zarabiam% f \ n”, 35, 34050.25);

Daje to następujące dane wyjściowe:

Jestem Betty, mam 35 lat i zarabiam 34050.25

Możesz także wyświetlić zmienne całkowite i zmiennoprzecinkowe w ten sam sposób.

Przykłady

1. Poniższy program przechowuje kilka wartości w trzech zmiennych, następnie wyświeltić wyniki.

```
// Nazwa pliku: C7PRNTF.CPP

// Drukuje wartości w zmiennych z odpowiednimi etykietami.

#include <stdio.h>

main()

{

char first='E'; // Store some character, integer,

char middle='W'; // and floating-point variable.

char last='C';

int age=32;

int dependents=2;

float salary=25000.00;

float bonus=575.25;

/* Prints the results. */

printf("Here are the initials\n");

printf("%c%c%c\n\n", first, middle, last);

printf("The age and number of dependents are\n");
```

```
printf(“%d %d\n\n”, age, dependents);

printf(“The salary and bonus are\n”);

printf(“%f %f”, salary, bonus);

return 0;

}
```

Dane wyjściowe z tego programu to

```
Oto inicjały

ERZ

Wiek i liczba osób na utrzymaniu to

32 2

Wynagrodzenie i premia są

25000,000000 575,250000
```

2. Wartości zmiennoprzecinkowe są drukowane ze zbyt wieloma zerami, ale liczby są poprawne. Można ograniczyć liczbę drukowanych zer wiodących i końcowych, dodając specyfikator szerokości w ciągu kontrolnym. Na przykład następujący printf () drukuje wynagrodzenie i premię z dwoma miejscami po przecinku:

```
printf („%. 2f% .2f”, wynagrodzenie, premia);
```

Upewnij się, że wydrukowane wartości są zgodne z dostarczonym z nimi łańcuchem kontrolnym. Funkcja printf () nie może naprawić problemów wynikających z niedopasowania wartości i ciągów kontrolnych. Nie próbuj drukować wartości zmiennoprzecinkowych za pomocą kodów sterujących ciągów znaków. Jeśli podasz pięć zmiennych całkowitych w printf (), pamiętaj o dołączeniu pięciu znaków konwersji d także w printf ().

Wyświetlanie wartości ASCII

Istnieje jeden wyjątek od reguły drukowania z pasującymi znakami konwersji. Jeśli chcesz wydrukować wartość ASCII znaku, możesz wydrukować ten znak (bez względu na to, czy jest to stała, czy zmienna) za pomocą znaku konwersji liczby całkowitej% d. Zamiast drukowania znaku printf () wypisuje pasujący numer ASCII dla tego znaku. I odwrotnie, jeśli wydrukujesz liczbę całkowitą ze znakiem konwersji% c, zobaczysz znak, który pasuje do wartości tej liczby całkowitej z tabeli ASCII. Poniższe printf () ilustrują ten fakt:

```
printf („% c”, 65); // Drukuje literę A.

printf („% d”, „A”); // Drukuje liczbę 65.
```

Funkcja scanf()

Funkcja scanf () odczytuje dane z klawiatury. Kiedy twoje programy osiągną linię za pomocą scanf (), użytkownik może wprowadzić wartości bezpośrednio do zmiennych. Twój program może następnie przetworzyć zmienne i produkować produkcję. Funkcja scanf () wygląda bardzo podobnie do printf (). Zawiera łańcuch kontrolny i jedną lub więcej zmiennych po prawej stronie łańcucha kontrolnego. Ciąg

kontrolny informuje C ++ dokładnie, jak wyglądają przychodzące wartości klawiatury i jakie są ich typy. Format scanf() to

scanf (control_string, jedna lub więcej wartości);

Łańcuch kontrolny scanf() używa prawie takich samych znaków konwersji jak łańcuch kontrolny printf(), z dwiema niewielkimi różnicami. Nigdy nie należy umieszczać znaku nowej linii \ n w ciągu kontrolnym scanf(). Funkcja scanf() „wie", kiedy wprowadzanie danych zostanie zakończone, gdy użytkownik naciśnie klawisz Enter. Jeśli podasz dodatkowy kod nowej linii, scanf() może nie zostać poprawnie zakończony. Ponadto zawsze umieszczaj początkową spację w każdym ciągu sterującym scanf(). Nie wpływa to na dane wejściowe użytkownika, ale scanf() czasami wymaga, aby działał poprawnie. Późniejsze przykłady w tym rozdziale wyjaśniają ten fakt. Jak wspomniano wcześniej, scanf() stwarza kilka problemów. Funkcja scanf() wymaga, aby użytkownik wpisał dane wejściowe dokładnie tak, jak określa łańcuch kontrolny. Ponieważ nie możesz kontrolować pisania przez użytkownika, nie zawsze można to zapewnić. Na przykład możesz chcieć, aby użytkownik wprowadził wartość całkowitą, po której następuje wartość zmiennoprzecinkowa (łańcuch kontrolny scanf() może się tego spodziewać), ale użytkownik może zdecydować się na wprowadzenie czegoś innego! Jeśli tak się stanie, niewiele możesz zrobić. Wynikowe dane wejściowe są niepoprawne, ale Twój program C nie ma niezawodnej metody testowania dokładności użytkownika przed uruchomieniem programu.

PRZESTROGA: Wartości wejściowe klawiatury użytkownika muszą być zgodne, pod względem liczby i typu, z ciągiem kontrolnym zawartym w każdym skanie (). Kolejny problem z scanf () nie jest tak łatwy do zrozumienia dla początkujących, jak ostatni. Funkcja scanf () wymaga użycia zmiennych wskaźnikowych, a nie zmiennych regularnych, w nawiasach. Choć brzmi to skomplikowanie, nie musi tak być. Nie powinieneś mieć problemu z wymaganiami dotyczącymi wskaźnika scanf (), jeśli pamiętasz te dwie proste reguły:

1. Zawsze umieszczaj znak ampersand (&) przed nazwami zmiennych wewnątrz scanf ().

2. Nigdy nie umieszczaj znaku ampersand (&) przed nazwą tablicy wewnątrz scanf ().

Pomimo tych dziwnych reguł scanf () możesz szybko nauczyć się tej funkcji, patrząc na kilka przykładów.

Korzystanie z operatorów matematycznych C++ i pierwszeństwa

Jeśli boisz się tej części, ponieważ nie lubisz matematyki - zrelaksuj się, C++ wykona całą Twoją matematykę! To błędne przekonanie, że musisz być dobry z matematyki, aby zrozumieć, jak programować komputery. W rzeczywistości praktyka programowania zakłada, że jest odwrotnie! Twój komputer jest twoim „niewolnikiem", który postępuje zgodnie z instrukcjami i wykonuje wszystkie obliczenia. Ta sekcja wyjaśnia, w jaki sposób oblicza się C++, wprowadzając cię do

♦ Głównych operatorów matematycznych

♦ Kolejność pierwszeństwa operatorów

♦ Instrukcje przypisania

♦ Obliczenia mieszanych typów danych

♦ Rzutowanie typów

Wiele osób, które nie lubi matematyki, naprawdę lubi uczyć się, jak radzi sobie z nią komputer. Po zapoznaniu się z operatorami matematyki i kilkoma prostymi sposobami ich używania przez C++ powinieneś czuć się swobodnie korzystając z obliczeń w swoich programach. Komputery są szybkie i mogą wykonywać operacje matematyczne znacznie szybciej niż ty!

Podstawowe operatory matematyczne w C++

Operator matematyczny C++ to symbol używany do dodawania, odejmowania, mnożenia, dzielenia i innych operacji. Operatory C++ nie zawsze mają charakter matematyczny, ale wiele z nich jest.

Symbol : Znaczenie

* : Mnożenie

/ : Dzielenie i dzielenie całkowite

% : Moduł lub reszta

+ : Dodawanie

- : Odejmowanie

Większość z tych operatorów działa w znany sposób. Mnożenie, dodawanie i odejmowanie dają takie same wyniki (i zwykle robi to operator dzielenia), jak te uzyskiwane za pomocą kalkulatora.

Nie myl operacji binarnych z liczbami binarnymi. Kiedy operator jest używany między dwoma literałami, zmiennymi lub ich kombinacją, jest nazywany operatorem binarnym, ponieważ działa przy użyciu dwóch wartości. Gdy używasz tych operatorów (na przykład przypisując ich wyniki do zmiennych), w C++ nie ma znaczenia, czy dodasz spacje do operatorów, czy nie.

PRZESTROGA: W celu pomnożenia użyj gwiazdki (*), a nie x, jak zwykle. x nie może być użyte jako znak mnożenia, ponieważ C++ używa x jako nazwy zmiennej. C++ interpretuje x jako wartość zmiennej o nazwie x.

Operatory Jednoargumentowe

Jednoargumentowy operator działa na jedną wartość lub wpływa na nią. Na przykład możesz przypisać zmiennej liczbę dodatnią lub ujemną, używając jednoargumentowego znaku + lub -.

Przykłady

1. Poniższa sekcja kodu przypisuje czterem zmiennym liczbę dodatnią lub ujemną. Znaki plus i minus są jednoznaczne, ponieważ nie są używane między dwiema wartościami.

Zmienna a ma przypisaną wartość ujemną 25.

Zmienna b ma przypisaną wartość dodatnią 25.

Zmienna c ma przypisaną wartość ujemną a.

Zmienna d ma przypisaną dodatnią wartość b.

a = -25; // Przypisz „a” minus 25.

b = +25; // Przypisz „b” plus 25 (+ nie jest potrzebne).

c = -a; // Przypisz „c” ujemną wartość „a” (-25).

d = + b; // Przypisz „d” dodatnią wartość „b” (25, + niepotrzebne).

2. Zasadniczo nie musisz używać jednoznacznego znaku plus. C++ zakłada, że liczba lub zmienna jest dodatnia, nawet jeśli nie ma znaku plus. Poniższe cztery stwierdzenia są równoważne z poprzednimi czterema, z wyjątkiem tego, że nie zawierają znaków plus.

a = -25; // Przypisz „a” minus 25.

b = 25; // Przypisz „b” plus 25.

c = -a; // Przypisz „c” ujemną wartość „a” (-25).

d = b; // Przypisz „d” dodatnią wartość „b” (25).

3. Jednoargumentowy operator ujemny przydaje się, gdy chcesz zanegować pojedynczą liczbę lub zmienną. Negatyw negatywu jest dodatni. Dlatego poniższy krótki program przypisuje liczbę ujemną (za pomocą jednoargumentowego -) zmiennej, a następnie wypisuje ujemną tę samą zmienną. Ponieważ miał na początku liczbę ujemną, cout daje wynik dodatni.

```
// Nazwa pliku: C8NEG.CPP

// Ujemna zmienna  zawierająca wartość ujemną.

#include <iostream.h>

main()

{

signed int temp = -12; // „ssigned” nie jest potrzebne, ponieważ

// to jest ustawienie domyślne.

cout << -temp << „\ n”; // Tworzy 12 na ekranie ekranów.

Return 0;

}
```

Deklaracja zmiennej nie potrzebuje prefiksu signed, ponieważ wszystkie zmienne całkowite są domyślnie signed.

4. Jeśli chcesz odjąć ujemne wartości zmiennej, upewnij się, że wstawiłeś spację przed jednoargumentowym znakiem minus. Na przykład następujący wiersz:

new_temp + new_temp- -inversion_factor;

tymczasowo neguje inversion_factor i odejmuje tę negowaną wartość od new_temp.

Dzielenie i moduł

Znak dzielnia / i operator modułu,%, mogą zachowywać się w nieznany ci sposób. Są jednak równie łatwe w użyciu, jak inni operatorzy, których właśnie widzieliście. Ukośnik do przodu (/) jest zawsze używany do podziału. Jednak generuje liczbę całkowitą o nazwie dziel, jeśli wartości całkowite (literały, zmienne lub kombinacja obu) pojawiają się po obu stronach ukośnika. Jeśli pozostała część, C++ ją odrzuca. Znak procentu (%) tworzy moduł lub resztę dzielenia liczb całkowitych. Wymaga, aby liczby całkowite znajdowały się po obu stronach symbolu, w przeciwnym razie nie działa.

Przykłady

1. Załóżmy, że chcesz obliczyć tygodniowe wynagrodzenie. Poniższy program prosi o roczną wypłatę, dzieli ją przez 52 i wyświetla wyniki z dokładnością do dwóch miejsc po przecinku.

```
// Nazwa pliku: C8DIV.CPP

// Wyświetla tygodniowe wynagrodzenie użytkownika.

#include <stdio.h>

main()

{

float weekly, yearly;

printf(“What is your annual pay? “); // Prompt user.

scanf(“%f”, &yearly);

weekly = yearly/52; // Computes the weekly pay.

printf(“\n\nYour weekly pay is $%.2f”, weekly);

return 0;

}
```

Ponieważ liczba używana to liczba zmiennoprzecinkowa, C++ daje wynik w postaci liczb zmiennoprzecinkowych. Oto przykładowe wyjście z takiego programu:

```
Jaka jest twoja roczna pensja? 38000,00

Tygodniowa wypłata wynosi 730,77 USD
```

Ponieważ ten program używał funkcji scanf() i printf() (aby zapoznać się z oboma sposobami wprowadzania i wyprowadzania danych), plik nagłówkowy stdio.h jest dołączony zamiast iostream.h.

2. Dzielenie liczb całkowitych nie zaokrągla wyników. Jeśli podzielisz dwie liczby całkowite, a odpowiedź nie będzie liczbą całkowitą, C++ ignoruje część ułamkową. Poniższa pomoc printf () pokazuje to. Dane

wyjściowe wynikające z każdego printf() pojawiają się w komentarzu po prawej stronie każdego wiersza.

printf („% d \ n”, 10/2); // 5 (bez reszty)

printf („% d \ n”, 300/100); // 3 (bez reszty)

printf („% d \ n”, 10/3); // 3 (odrzucone resztki)

printf („% d \ n”, 300/165); // 1 (odrzucone resztki)

Kolejność pierwszeństwa

Zrozumienie operatorów matematyki jest pierwszym z dwóch kroków w kierunku rozumienie obliczeń C++. Musisz także zrozumieć kolejność pierwszeństwa. Kolejność pierwszeństwa (czasami nazywana hierarchią matematyczną lub kolejnością operatorów) określa dokładnie, w jaki sposób C++ oblicza formuły. Pierwszeństwo operatorów to dokładnie ta sama koncepcja, której nauczyłeś się na kursach algebry w szkole średniej. (Nie martw się, to prosta część algebry!) Aby zobaczyć, jak działa kolejność pierwszeństwa, spróbuj ustalić wynik następującego prostego obliczenia:

2 + 3 * 2

Jeśli powiedziałeś 10, nie jesteś sam; wiele osób odpowiada za pomocą 10. Jednak 10 jest poprawne tylko wtedy, gdy interpretujesz formułę od lewej. Co zrobić, jeśli najpierw obliczyłeś mnożenie? Jeśli wziąłeś wartość 3 * 2 i uzyskałeś odpowiedź 6, a następnie dodałeś 2, otrzymasz odpowiedź 8 - dokładnie taką samą odpowiedź, jaką oblicza C++ (i okazuje się być prawidłowy)! C++ zawsze wykonuje najpierw mnożenie, dzielenie i moduł, a następnie dodawanie i odejmowanie. Poniżej mamy kolejność operatorów, które dotychczas widziałeś. Oczywiście istnieje wiele więcej poziomów w tabeli priorytetów operatorów C++ niż te pokazane. W przeciwieństwie do większości języków komputerowych, C++ ma 20 poziomów pierwszeństwa.

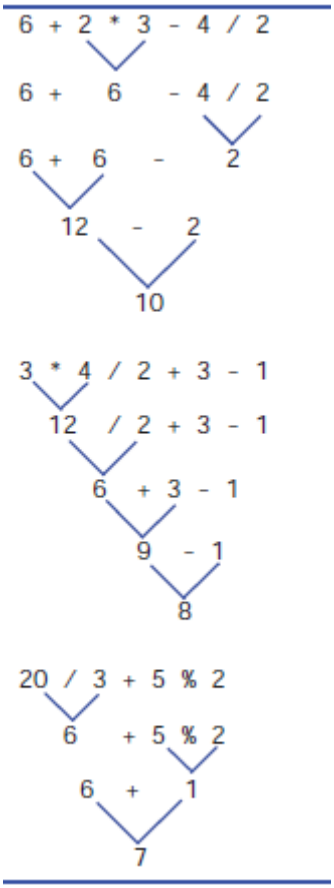
Porządek : Operator

Po pierwsze: mnożenie, dzielenie, reszta, moduł (*, /,%)

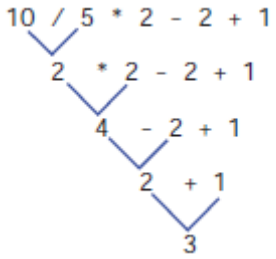
Po drugie: dodawanie, odejmowanie (+, -)

Przykłady

1. Łatwo jest przestrzegać kolejności operatorów C++, jeśli postępujesz zgodnie z wynikiami pośrednio i pojedynczo. Trzy obliczenia na rysunku poniżej pokazują, jak to zrobić.



2. Patrząc wstecz na kolejność tabeli pierwszeństwa, można zauważyć, że mnożenie, dzielenie i moduł są na tym samym poziomie. Oznacza to, że na tym poziomie nie ma hierarchii. Jeśli więcej niż jeden z tych operatorów pojawia się w obliczeniach, C++ wykonuje matematykę od lewej. To samo dotyczy dodawania i odejmowania - C++ najpierw wykonuje operację na skrajnie lewej stronie. Rysunek ilustruje przykład pokazujący ten proces.



Ponieważ dzielenie pojawia się po lewej stronie mnożenia, jest obliczany jako pierwszy. Teraz powinieneś być w stanie śledzić kolejność tych operatorów C++. Nie musisz się martwić o matematykę, ponieważ C++ faktycznie działa. Należy jednak zrozumieć tę kolejność operatorów, aby wiedzieć, jak ustrukturyzować swoje obliczenia. Po opanowaniu tego porządku nadszedł czas, aby dowiedzieć się, jak można go zastąpić nawiasami!

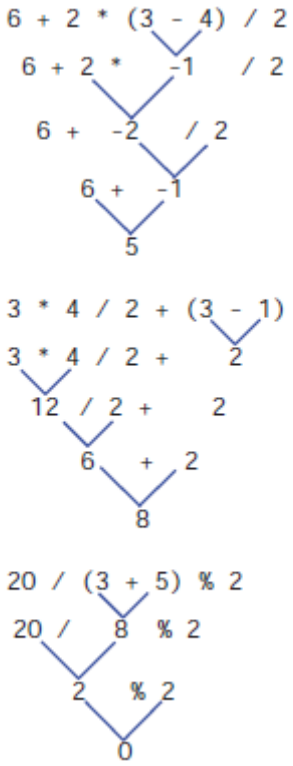
Korzystanie z nawiasów

Jeśli chcesz zastąpić kolejność pierwszeństwa, możesz dodać nawiasy do obliczeń. Nawiasy faktycznie znajdują się na poziomie powyżej mnożenia, dzielenia i modułu w tabeli pierwszeństwa. Innymi słowy,

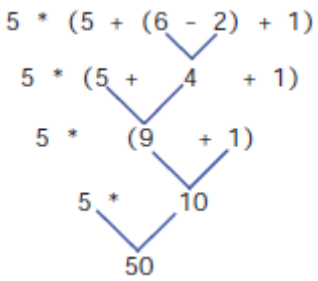
wszelkie obliczenia w nawiasach - niezależnie od tego, czy są to dodawanie, odejmowanie, dzielenie itp. - są zawsze obliczane przed resztą wiersza. Inne obliczenia są następnie wykonywane w ich normalnej kolejności operatora. Pierwsza formuła w tym rozdziale, 2 + 3 * 2, dała 8, ponieważ mnożenie przeprowadzono przed dodaniem. Jednak dodając nawiasy wokół dodawania, jak w (2 + 3) * 2, odpowiedź staje się 10. W tabeli pierwszeństwa pokazanej w dodatku D, „Tabela pierwszeństwa w C++", nawiasy znajdują się na poziomie 3. Ponieważ są wyższe niż na innych poziomach, nawiasy mają pierwszeństwo przed mnożeniem, dzieleniem i wszystkimi innymi operatorami.

Przykłady

1. Obliczenia pokazane na rysunku poniżej ilustrują, w jaki sposób nawiasy zastępują regularną kolejność operatorów. Są to te same trzy formuły pokazane w poprzedniej sekcji, ale ich wyniki są obliczane inaczej, ponieważ nawiasy zastępują normalną kolejność operatorów.



2. Jeśli wyrażenie zawiera nawiasy w nawiasach, C++ najpierw ocenia najgłębsze nawiasy. Ilustrują to wyrażenia poniżej



3. Poniższy program generuje niepoprawny wynik, nawet jeśli wygląda na to, że zadziała. Sprawdź, czy możesz zauważyć błąd!

Komentarze do identyfikacji twojego programu.

Dołącz plik nagłówkowy iostream.h, aby cout działał.

Zadeklaruj zmienne avg, grade1, grade2 i grade3 jako zmienne zmiennoprzecinkowe.

Zmienna średnia staje się równa klasie 3 podzielonej przez 3,0 plus ocena 2 plus ocena 1.

Wyświetl na ekranie Średnia to i średnia z trzech zmiennych ocen.

Wróć do systemu operacyjnego.

```
// Nazwa pliku: C8AVG1.CPP

// Oblicz średnią z trzech ocen.

#include <iostream.h>

main()

{

float avg, grade1, grade2, grade3;

grade1 = 87.5;

grade2 = 92.4;

grade3 = 79.6;

avg = grade1 + grade2 + grade3 / 3.0;

cout << “The average is “ << avg << “\n”;

return 0;

}
```

Problem polega na tym, że dzielenie jest wykonywane jako pierwszy. Dlatego trzecia klasa jest najpierw dzielona przez 3,0, a następnie do tego wyniku są dodawane pozostałe dwie oceny. Aby rozwiązać ten problem, wystarczy dodać jeden zestaw nawiasów, jak pokazano poniżej:

```
// Filename: C8AVG1.CPP

// Oblicz średnią z trzech ocen.

#include <iostream.h>

main()

{

float avg, grade1, grade2, grade3;

grade1 = 87.5;

grade2 = 92.4;

grade3 = 79.6;

avg = grade1 + grade2 + grade3 / 3.0;

cout << “The average is “ << avg << “\n”;
```

```
return 0;

}
```

WSKAZÓWKA: Używaj dużej liczby nawiasów w swoich programach C++, aby wyjaśnić kolejność operatorów, nawet jeśli nie musisz zastępować ich domyślnej kolejności. Korzystanie z nawiasów ułatwia późniejsze obliczenia, gdy konieczne może być zmodyfikowanie programu.

Krótszy nie zawsze jest lepszy

Kiedy programujesz komputery do życia, znacznie ważniejsze jest pisanie programów, które są łatwe do zrozumienia niż programy, które są krótkie lub zawierają skomplikowane obliczenia. Konserwowalność to słowo branży komputerowej dotyczące zmiany i aktualizacji programów napisanych wcześniej w prostym stylu. Świat biznesu szybko się zmienia, a programy, z których firmy korzystają od lat, często muszą być aktualizowane, aby odzwierciedlić to zmieniające się otoczenie. Firmy nie zawsze mają zasoby do pisania programów od zera, więc zwykle modyfikują te, które mają. Wiele lat temu, kiedy sprzęt komputerowy był znacznie droższy, a gdy pamięć komputera była znacznie mniejsza, ważne było pisanie małych programów, co często oznaczało poleganie na sprytnych, zindywidualizowanych sztuczkach i skrótach. Niestety, takie programy są często trudne do poprawienia, szczególnie jeśli oryginalni programiści odchodzą, a ktoś inny (ty!) musi zmodyfikować oryginalny kod. Firmy zdają sobie sprawę z tego, jak ważne jest spędzanie czasu na pisaniu programów, które można łatwo modyfikować i które nie polegają na sztuczkach lub „szybkich i brudnych" procedurach, których trudno jest przestrzegać. Możesz być znacznie bardziej wartościowym programistą, pisząc czyste programy z dużą ilością białych znaków, częstymi uwagami i prostym kodem. Używaj nawiasów w formułach, jeśli formuły są jaśniejsze, i używaj zmiennych do przechowywania wyników na wypadek, gdybyś potrzebował tej samej odpowiedzi później w programie. Podziel długie obliczenia na kilka mniejszych. W pozostałej części tej książki możesz przeczytać wskazówki dotyczące pisania programów, które można utrzymać. Ty i twoi koledzy docenicie te wskazówki, kiedy zastosujecie je we własnych programach C++.

Instrukcje przypisania

W C++ operator przypisania, =, zachowuje się inaczej niż w innych językach. Do tej pory używałeś go do przypisywania wartości do zmiennych, co jest spójne z jego użyciem w większości innych języków programowania. Jednak operator przypisania może być również użyty na inne sposoby, takie jak wiele instrukcji przypisania i przypisań złożonych, co ilustrują poniższe sekcje.

Wiele zadań

Jeśli w wyrażeniu występują dwa lub więcej znaków równości, każdy wykonuje przypisanie. Ten fakt wprowadza nowy aspekt porządku pierwszeństwa, który powinieneś zrozumieć. Rozważ następujące wyrażenie:

```
a = b = c = d = e = 100;
```

Z początku może się to wydawać mylące, zwłaszcza jeśli znasz inne języki komputerowe. Dla C++ znak równości zawsze oznacza: Przypisz wartość po prawej stronie do zmiennej po lewej stronie. Ta kolejność od prawej do lewej jest opisana w tabeli pierwszeństwa. Trzecia kolumna w tabeli jest oznaczona Asocjatywność, która opisuje kierunek operacji. Operator przypisania kojarzy od prawej, podczas gdy niektórzy inni operatorzy C++ kojarzą od lewej.

Ponieważ przypisanie jest powiązane z prawej strony, poprzednie wyrażenie przypisuje 100 do zmiennej o nazwie e. To zadanie tworzy wartość 100 dla wyrażenia. W C++ wszystkie wyrażenia

generować wartości, zazwyczaj wynik przypisań. Dlatego 100 to przypisanie do zmiennej d. Wartość 100 jest przypisywana do c, następnie do b, a na końcu do a. Stare wartości tych zmiennych są zastępowane przez 100 po zakończeniu instrukcji. Ponieważ C++ nie ustawia automatycznie zmiennych na zero przed ich użyciem, możesz to zrobić przed użyciem zmiennych z pojedynczą instrukcją przypisania. Następująca sekcja deklaracji zmiennych i inicjalizacji jest wykonywana przy użyciu wielu instrukcji przypisania

```
main()

{

int ctr, num_emp, num_dep;

float sales, salary, amount;

ctr=num_emp=num_dep=0;

sales=salary=amount=0;

// Reszta programu następuje.
```

W C++ możesz prawie dołączyć instrukcję przypisania w dowolnym miejscu programu, nawet w innych obliczeniach. Rozważmy na przykład tą instrukcję:.

```
value = 5 + (r = 9 - c);
```

co jest całkowicie legalnym oświadczeniem w C++. Operator przypisania znajduje się na pierwszym poziomie tabeli pierwszeństwa i zawsze generuje wartość. Ponieważ jego asocjatywność pochodzi z prawej strony, r jest przypisywane 9 - c, ponieważ znak równości po skrajnej prawej stronie jest oceniany jako pierwszy. Podwyrażenie (r = 9 - c) tworzy wartość (i umieszcza tę wartość w r), która jest następnie dodawana do 5 przed zapisaniem wartości w odpowiedzi.

Przykład

Ponieważ C++ nie inicjuje zmiennych do zera przed ich użyciem, możesz chcieć dołączyć operatora wielu przypisań, aby to zrobić przed użyciem zmiennych. Poniższa sekcja kodu zapewnia, że wszystkie zmienne są inicjowane, zanim pozostała część programu ich użyje.

```
main()

{

int num_emp, dependents, age;

float salary, hr_rate, taxrate;

// Initialize all variables to zero.

num_emp=dependents=age=hours=0;

salary=hr_rate=taxrate=0.0;

// Reszta programu następuje.
```

Przypisania złożone

Wiele razy w programowaniu możesz chcieć zaktualizować wartość zmiennej. Innymi słowy, musisz wziąć bieżącą wartość zmiennej, dodać lub pomnożyć tę wartość przez wyrażenie, a następnie

ponownie przypisać ją do oryginalnej zmiennej. Poniższa instrukcja przypisania demonstruje ten proces:

salary=salary*1.2;

Wyrażenie to zwielokrotnia starą wartość salary przez 1,2 (w efekcie podnosząc wartość wynagrodzenia o 20 procent), a następnie przypisuje ją do wynagrodzenia. C++ udostępnia kilka operatorów, zwanych operatorami złożonymi, z których można korzystać za każdym razem, gdy ta sama zmienna pojawia się po obu stronach znaku równości. Operatory złożone pokazano tu.

Operator: Przykład: Odpowiednik

+ =: bonus + = 500; : bonus = bonus + 500;

- =: budżet- = 50; : budżet = budżet-50;

* =: wynagrodzenie * = 1,2; : wynagrodzenie = wynagrodzenie * 1,2;

/ =: współczynnik / =. 50; : współczynnik = współczynnik / .50;

% =: daynum% = 7; : daynum = daynum% 7;

Operatory złożone są niskie w tabeli pierwszeństwa. Zazwyczaj są one oceniane na końcu lub blisko końca.

Przykłady

1. Przechowujesz ilość produkcji w fabryce w zmiennej o nazwie prod_amt, a twój przełożony właśnie poinformował cię, że do wartości produkcji należy zastosować nowy dodatek. Możesz zakodować tę aktualizację w instrukcji w następujący sposób:

prod_amt = prod_amt + 2.6; // Dodaj 2.6 do bieżącej produkcji.

Zamiast używać tej formuły, użyj operatora dodawania złożonego C++, kodując go w następujący sposób:

prod_amt + = 2,6; // Dodaj 2.6 do bieżącej produkcji.

2. Załóżmy, że jesteś nauczycielem w szkole średniej, który chce podnosić oceny swoich uczniów. Dałeś test, który był zbyt trudny, a oceny nie były zgodne z oczekiwaniami. Jeśli zapisałeś każdą z ocen ucznia w zmiennych o nazwie ocena1, ocena2, ocena3 itd., Możesz zaktualizować oceny w programie za pomocą poniższej sekcji przypisań złożonych.

grade1*=1.1; // Zwiększ ocenę każdego ucznia o 10. procent.

grade2*=1.1;

grade3*=1.1;

3. Pierwszeństwo operatorów złożonych wymaga ważnego rozważenia przy podejmowaniu decyzji o kodowaniu przypisań złożonych. Oznacza to, że musisz uważać na ich interpretację. Załóżmy na przykład, że chcesz zaktualizować wartość zmiennej sprzedaży za pomocą tej formuły:

4-czynnikowy + bonus

Możesz zaktualizować zmienną sprzedaży za pomocą następującego oświadczenia:

sales = *4 - factor + bonus;

To oświadczenie dodaje 4-czynnikową ilość + premię do sprzedaży. Ze względu na pierwszeństwo operatora ta instrukcja nie jest taka sama jak następująca:

sales = sales *4 - factor + bonus;

Ponieważ operator *= jest znacznie niższy w tabeli pierwszeństwa niż * lub -, jest wykonywany jako ostatni i z skojarzeniem od prawej do lewej. Dlatego poniższe są równoważne z punktu widzenia pierwszeństwa:

sales *= 4 - factor + bonus;

i

sales = sales * (4 - factor + bonus);

Mieszanie typów danych w obliczeniach

Możesz mieszać typy danych w C++. Dodanie liczby całkowitej i wartości zmiennoprzecinkowej jest mieszaniem typów danych. C++ generalnie konwertuje mniejszy z tych dwóch typów na drugi. Na przykład, jeśli dodasz podwójną liczbę całkowitą, C++ najpierw konwertuje liczbę całkowitą na podwójną wartość, a następnie wykonuje obliczenia. Ta metoda daje najdokładniejszy możliwy wynik. Automatyczna konwersja typów danych jest tylko tymczasowa; skonwertowana wartość wraca do pierwotnego typu danych, gdy tylko wyrażenie zostanie zakończone. Jeśli C++ przekonwertuje dwa różne typy danych na typ o mniejszej wartości, wartość o wyższej precyzji zostanie obcięta lub skrócona, a dokładność zostanie utracona. Na przykład w poniższym krótkim programie zmiennoprzecinkowa wartość sprzedaży jest dodawana do liczby całkowitej zwanej premią. Zanim C++ oblicza odpowiedź, konwertuje bonus na zmiennoprzecinkowy, co daje odpowiedź zmiennoprzecinkową.

```
// Filename: C8DATA.CPP

// Zademonstruj mieszany typ danych w wyrażeniu.

#include <stdio.h>

main()

{

int bonus=50;

float salary=1400.50;

float total;

total=salary+bonus; / bonus staje się zmiennoprzecinkowy

// ale tylko tymczasowo

printf(“The total is %.2f”, total);

return 0;

}
```

Rzutowanie typów

Przez większość czasu nie musisz się martwić automatyczną konwersją typów danych w C++. Problemy mogą jednak wystąpić, jeśli zmieszane zostaną zmienne niepodpisane ze zmiennymi innych typów danych. Ze względu na różnice w architekturze komputera zmienne unsigned nie zawsze są konwertowane na większy typ danych. Może to spowodować utratę dokładności, a nawet nieprawidłowe wyniki. Możesz zastąpić domyślne konwersje C++, określając własną tymczasową zmianę typu. Ten proces nazywa się rzutowaniem typu. Podczas pisania rzutowania tymczasowo zmieniasz typ danych zmiennej z zadeklarowanego typu danych na nowy. Istnieją dwa formaty rzutowania typu. Otoi są

(typ danych) wyrażenie

i

typ danych (wyrażenie)

gdzie typ danych może być dowolnym poprawnym typem danych C++, takim jak int lub float, a wyrażenie może być zmienną, literalną lub wyrażeniem, które łączy oba te elementy. Poniższy typ kodu tymczasowo rzutuje zmienną całkowitą wiek na zmienną zmiennoprzecinkową podwójną, dzięki czemu można ją pomnożyć przez współczynnik podwójnej zmiennoprzecinkowej. Oba formaty typu pliku obsady są zilustrowane. Zmienna age_factor ma przypisaną wartość zmiennej age (obecnie traktowanej jak podwójna zmienna zmiennoprzecinkowa) pomnożonej przez współczynnik zmiennej.

age_factor = (double)age * factor;; // Tymczasowo zmień wiek podwój

Drugi sposób rzutowania typu dodaje nawiasy wokół zmiennej zamiast typu danych, ponieważ:

age_factor = double(age) * factor;) ; // Tymczasowo zmień wiek aby podwoić.

UWAGA: Rzutowanie typu przez dodanie nawiasów wokół wyrażenia, a nie typ danych, jest nowe w C++. Programiści C nie mają takiej opcji - muszą wstawić typ danych w nawiasach. Druga metoda „wydaje się” jak wywołanie funkcji i wydaje się bardziej naturalna dla tego języka. Dlatego zapoznanie się z drugą metodą wyjaśni kod.

Przykłady

1. Załóżmy, że chcesz zweryfikować naliczone odsetki używane przez bank od pożyczki. Stopa procentowa wynosi 15,5 procent, przechowywana jako .155 w zmiennej zmiennoprzecinkowej. Kwotę należnych odsetek oblicza się, mnożąc stopę procentową przez kwotę salda pożyczki, a następnie mnożąc ją przez liczbę dni w roku, w którym pożyczka została udzielona. Poniższy program znajduje dzienną stopę procentową, dzieląc roczną stopę procentową przez 365, czyli liczbę dni w roku.

C++ musi automatycznie przekonwertować liczbę całkowitą 365 na literę zmiennoprzecinkową, ponieważ jest ona używana w połączeniu ze zmienną zmiennoprzecinkową.

```
// Filename: C8INT1.CPP

// Oblicz odsetki od pożyczki.

#include <stdio.h>

main()

{

int days=45; // Dni od udzielenia pożyczki.
```

```
float principle = 3500.00; // Oryginalna kwota pożyczki

float interest_rate=0.155; // Roczna stopa procentowa

float daily_interest; // Dzienna stopa procentowa

daily_interest=interest_rate/365; // Oblicz wartość zmiennoprzecinkową

// Ponieważ dni są liczbą całkowitą, również jest konwertowane na liczbę zmiennoprzecinkową.

daily_interest = principle * daily_interest * days;

principle+=daily_interest;// Aktualizacja zasady o odsetki.

printf("Saldo, które jesteś winien%.2f\n", principle);

return 0;

}
```

Dane wyjściowe tego programu są następujące:

Saldo, które jesteś winien, wynosi 3566,88

2. Zamiast C++ ma wykonać konwersję, możesz rzutować typy wszystkich mieszanych wyrażeń, aby upewnić się, że są one konwertowane według twoich upodobań. Oto ten sam program, co w pierwszym przykładzie, z wyjątkiem tego, że rzutowania typu są używane do konwersji literałów całkowitych na zmiennoprzecinkowe przed ich użyciem.

```
// Filename: C8INT2.CPP

// Oblicz odsetki od pożyczki za pomocą castingu typu.

#include <stdio.h>

main()

{

int days=45; // Dni od udzielenia pożyczki..

float principle = 3500.00; // Oryginalna kwota pożyczki

float interest_rate=0.155; // Roczna stopa procentowa

float daily_interest; //  Dzienna stopa procentowa

daily_interest=interest_rate/float(365); // Rzutowanie typu dni

// na float.

//  Ponieważ liczba dni jest liczbą całkowitą, przekonwertuj ją również na liczbę zmiennoprzecinkową.

daily_interest = principle * daily_interest * float(days);

principle+=daily_interest;// ktualizacja zasady o odsetki.

printf("Saldo, które jesteś winien%.2f", principle);

return 0;
```

}

Dane wyjściowe z tego programu są dokładnie takie same jak poprzednie.

Ćwiczenia

1. Napisz program, który wypisze każdą z pierwszych ośmiu potęg 2 (2^1, 2^2, 2^3, ... 2^8). Napisz komentarze i umieść swoje imię i nazwisko na górze programu. Wydrukuj literały łańcuchowe opisujące każdą wydrukowaną odpowiedź. Pierwsze dwa wiersze wyniku powinny wyglądać tak:

2 podniesione do pierwszej potęgi to 2

2 podniesione do drugiej potęgi to 4

2. Zmień C8PAY.CPP, aby obliczał i drukował premię w wysokości 15 procent wynagrodzenia brutto. Z premii nie należy pobierać podatków. Po wydrukowaniu czterech zmiennych: Gross_pay, Tax_rate, Bonus i Gross_pay, wydrukuj czek na ekranie, który wygląda jak wydrukowany czek. Dodaj literały ciągów, aby wypisał czek i umieścił twoje imię jako płatnika na dole czeku.

3. Przechowuj wagę i wiek trzech osób w zmiennych. Wydrukuj tabelę z tytułami, wagami i wiekiem. Na dole tabeli wydrukuj średnie.

4. Załóżmy, że pracownik sklepu wideo pracuje 50 godzin. Otrzymuje 4,50 USD za pierwsze 40 godzin, półtorej godziny (1,5 raza regularna stawka płacy) przez pierwsze pięć godzin po 40 i podwójną wypłatę za wszystkie godziny powyżej 45 lat. Przy założeniu 28-procentowej stawki podatkowej , napisz program, który wypisuje na ekranie wynagrodzenie brutto, podatki i wynagrodzenie netto. Oznacz każdą kwotę odpowiednimi tytułami (używając literałów łańcuchowych) i dodaj odpowiednie komentarze w programie.

Podsumowanie

Teraz rozumiesz główne operatory matematyczne w C++ i znaczenie tabeli pierwszeństwa. Operacje grupowe w nawiasach, aby mogły zastąpić domyślne poziomy pierwszeństwa. W przeciwieństwie do niektórych innych języków programowania, każdy operator w C++ ma znaczenie, bez względu na to, gdzie pojawia się w wyrażeniu. Ten fakt umożliwia użycie operatora przypisania (znak równości) w środku innych wyrażeń. Kiedy wykonujesz matematykę w C++, musisz także wiedzieć, w jaki sposób C++ interpretuje typy danych, zwłaszcza gdy łączysz je w tym samym wyrażeniu. Oczywiście możesz tymczasowo wpisać rzut zmienną lub literał, aby zastąpić domyślny typ danych. Ten rozdział wprowadził cię do części książki dotyczącej operatorów C++. Kolejne dwa rozdziały (Rozdział 9, „Operatory relacyjne” i Rozdział 10, „Operatory logiczne”) rozszerzają to wprowadzenie, aby objąć operatory relacyjne i logiczne. Umożliwiają porównywanie danych i odpowiednie obliczanie

Operatory relacyjne

Czasami nie chcesz, aby każda instrukcja w programie C++ była wykonywana przy każdym uruchomieniu programu. Jak dotąd, każdy program był wykonywany od góry i kontynuował, linia po linii, aż do zakończenia ostatniej instrukcji. W zależności od aplikacji może nie zawsze tak być. Programy, które nie zawsze uruchamiają się na pamięć, są znane jako programy z dostępem do danych. W programach opartych na danych dane decydują o tym, co program robi. Na przykład nie chcesz, aby komputer drukował wypłaty dla każdego pracownika za każdy okres wypłaty, na przykład, ponieważ niektórzy pracownicy mogą być na urlopie lub mogą być opłacani prowizją i nie dokonali sprzedaży w tym okresie. Drukowanie wypłat za zero dolarów jest śmieszne. Chcesz, aby komputer drukował czeki tylko dla pracowników, którzy przepracowali. Ta część pokazuje, jak tworzyć programy oparte na danych. Programy te nie działają za każdym razem w ten sam sposób. Jest to możliwe dzięki zastosowaniu operatorów relacyjnych, które warunkowo kontrolują inne instrukcje. Operatory relacyjne najpierw „patrzą" na literały i zmienne w programie, a następnie działają zgodnie z tym, co „znajdują". Może to zabrzmieć jak trudne programowanie, ale w rzeczywistości jest proste i intuicyjne.

Tu zapoznasz się z

♦ Operatorami relacyjnymi

♦ Instrukcją if

♦ Instrukcją else

Nie tylko wprowadza te polecenia porównania, ale przygotowuje cię na znacznie potężniejsze programy, możliwe po nauczeniu się operatorów relacyjnych.

Definiowanie operatorów relacyjnych

Oprócz operatorów matematycznych, których nauczyłeś się wcześniej , istnieją również operatory, których używasz do porównywania danych. Nazywa się je operatorami relacyjnymi, a ich zadaniem jest porównywanie danych. Umożliwiają ustalenie, czy dwie zmienne są równe, a nie równe, a która z nich jest mniejsza od drugiej. Tabela poniżej zawiera listę każdego operatora relacyjnego i jego znaczenie.

Operator: Opis

== : Równa się

> : Większy niż

< : Mniej niż

> = : Większy lub równy

<= : Mniejszy lub równy

!= : Nie równy

Sześć operatorów relacyjnych stanowi podstawę porównania danych w programowaniu w C++. Zawsze pojawiają się z dwoma literałami, zmiennymi, wyrażeniami (lub pewną ich kombinacją), po jednym z każdej strony operatora. Te operatory relacyjne są przydatne i powinieneś je znać tak dobrze, jak znasz operatory matematyczne +, -, *, / i %.

UWAGA: W przeciwieństwie do wielu języków programowania, C++ używa podwójnego znaku równości (==) jako testu na równość. Pojedynczy znak równości (=) jest zarezerwowany do przypisania wartości

Przykłady

1. Załóżmy, że program inicjuje cztery zmienne w następujący sposób:

int a = 5;

int b = 10;

int c = 15;

int d = 5;

Następujące stwierdzenia są wtedy Prawdą:

a jest równe d, więc a == d

b jest mniejsze niż c, więc b < c

c jest większe niż a, więc c > a

b jest większe lub równe a, więc b > = a

d jest mniejsze lub równe b, więc d <= b

b nie jest równe c, więc b != c

To nie są instrukcje C++; są to zestawienia porównawcze (logika relacyjna) między wartościami w zmiennych.

Logika relacyjna jest łatwa.

Logika relacyjna zawsze daje wynik Prawda lub Fałsz. W C++, w przeciwieństwie do niektórych innych języków programowania, można bezpośrednio użyć wyniku True lub False operatorów relacyjnych wewnątrz innych wyrażeń. Wkrótce nauczysz się, jak to zrobić; ale na razie musisz zrozumieć tylko, że następujące prawdziwe i fałszywe oceny są poprawne:

♦ Prawdziwy wynik relacyjny wynosi 1.

♦ Fałszywy wynik relacyjny jest oceniany na 0.

Każde stwierdzenie przedstawione wcześniej w tym przykładzie daje wynik 1 lub True.

2. Jeśli przyjmiesz te same wartości, jakie podano dla czterech zmiennych z poprzedniego przykładu, każda z instrukcji wartości ma wartość False (0):

a == b

b> c

d <a

d> a

a! = d

b> = c

c <= b

Przeanalizuj te stwierdzenia, aby zobaczyć, dlaczego każda z nich jest False i oceniana na 0. Na przykład zmienne a i d są dokładnie równe tej samej wartości (5), więc żadna z nich nie jest większa ani mniejsza od drugiej. Stosujesz logikę relacyjną w życiu codziennym. Pomyśl o następujących stwierdzeniach:

„Masło bez nazwy kosztuje mniej niż markowe".

„Moje dziecko jest młodsze niż Janeky".

„Nasze pensje są równe".

„Psy nie są w tym samym wieku."

Każde z tych stwierdzeń może być prawdziwe lub fałszywe. Nie ma innej możliwej odpowiedzi.

Oglądaj znaki!

Wiele osób twierdzi, że „nie są skłonni do matematyki" lub „nie są logiczni", a ty możesz być jednym z nich. Ale, jak wspomniano , nie musisz być dobry z matematyki, aby być dobrym programistą komputerowym. Nie powinieneś się też bać terminu „logika relacyjna", ponieważ właśnie widziałeś, jak go używasz w życiu codziennym. Niemniej jednak niektóre osoby wprowadzają w błąd. Dwa podstawowe operatory relacyjne, mniejsze niż (<) i większe niż (>), są łatwe do zapamiętania. Prawdopodobnie nauczyłeś się tej koncepcji w szkole, ale mogłeś ją zapomnieć. W rzeczywistości ich znaki mówią ci, co mają na myśli. Strzałka wskazuje mniejszą z dwóch wartości. Zauważ, że w poprzednim przykładzie 1 strzałka (punkt < lub >) zawsze wskazuje na mniejszą liczbę. Większa, otwarta część strzałki wskazuje na większą liczbę. Relacja jest fałszywa, jeśli strzałka wskazuje niewłaściwy kierunek. Innymi słowy, 4 > 9 jest fałszem, ponieważ symbol operatora wskazuje na 9, która nie jest liczbą mniejszą. W języku angielskim stwierdzenie to brzmi „4 jest większe niż 9", co jest oczywiście fałszywe.

Instrukcja if

Włączasz operatory relacyjne do programów C++ z instrukcją if. Takie wyrażenie nazywa się instrukcją decyzyjną, ponieważ testuje relację - przy użyciu operatorów relacyjnych - i, w oparciu o wynik testu, podejmuje decyzję o tym, która instrukcja ma

wykonaj następnie. Instrukcja if wygląda następująco:

if (warunek)

{blok jednej lub więcej instrukcji C++}

Warunek obejmuje wszelkie relacyjne porównanie i musi być ujęty w nawiasy. Widziałeś wcześniej kilka porównań relacyjnych, takich jak a == d, c < d itd. Blokiem jednej lub więcej instrukcji C++ jest dowolna instrukcja C++, taka jak przypisanie lub printf(), ujęte w nawiasy klamrowe. Blok if, czasami nazywany treścią instrukcji if, jest zwykle wcięty w kilku miejscach dla czytelności. To pozwala zobaczyć na pierwszy rzut oka dokładnie, co się dzieje, jeśli warunek jest spełniony. Jeśli po instrukcji if występuje tylko jedna instrukcja, nawiasy klamrowe nie są wymagane (ale zawsze warto je uwzględnić). Blok jest wykonywany tylko wtedy, gdy warunek jest spełniony. Jeśli warunek jest False, C++ ignoruje blok i po prostu wykonuje następną odpowiednią instrukcję w programie, która następuje po instrukcji if. Zasadniczo można odczytać instrukcję if w następujący sposób: „Jeśli warunek jest spełniony, wykonaj blok instrukcji w nawiasach klamrowych. W przeciwnym razie warunek musi być fałszywy; więc nie

wykonuj tego bloku, ale kontynuuj wykonywanie pozostałej części programu, jakby to było, gdyby instrukcja nie istniała.” Do podjęcia decyzji służy instrukcja if. Blok instrukcji następujących po if jest wykonywany, jeśli decyzja (wynik relacji) ma wartość True, ale blok nie wykonuje się inaczej. Podobnie jak w przypadku logiki relacyjnej, używasz również logiki if w życiu codziennym. Rozważ następujące stwierdzenia.

„Jeśli dzień będzie ciepły, pójdę popływać”.

„Jeśli zarobię wystarczająco dużo pieniędzy, zbudujemy nowy dom”.

„Jeśli światło świeci na zielono, idź”.

„Jeśli światło jest czerwone, zatrzymaj się”.

Każde z tych stwierdzeń jest warunkowe. Oznacza to, że wykonujesz działanie tylko wtedy, gdy warunek jest spełniony.

UWAGA: Nie wpisuj średnika po nawiasach testu relacyjnego. Średniki pojawiają się po każdej instrukcji wewnątrz bloku.

Wyrażenia jako warunek

C++ interpretuje każdą niezerową wartość jako True, a zero zawsze jako False. Umożliwia to wstawianie wyrażeń regularnych bezwarunkowych do logiki if. Aby zrozumieć tę koncepcję, rozważ następującą sekcję kodu:

```
main()

{

int age=21; // Deklaruje i przypisuje wiek jako 21.

if (age=85)

{cout << „Wiele przeżyłeś!”; }

// Pozostały kod programu idzie tutaj.
```

Na początku może się wydawać, że printf () nie wykonuje się, ale tak jest! Ponieważ linia kodu używała zwykłego przypisania operator (=) (nie operator relacyjny, ==), C++ wykonuje przypisanie 85 do wieku. Tak jak w przypadku wszystkich zadań, które widziałeś wcześniej , tworzy wartość dla wyrażenia 85. Ponieważ 85 jest niezerowe, C++ interpretuje warunek if jako True, a następnie wykonuje treść instrukcji if.

Mylenie testu równości relacyjnej (==) z regularnym operatorem przypisania (=) jest częstym błędem w programach C++, a niezerowy test True sprawia, że ten błąd jest jeszcze trudniejszy do odnalezienia

Projektanci C++ nie zamierzali tego wprowadzać w błąd. Chcą, abyś skorzystał z tej funkcji w dowolnym momencie mogą. Zamiast stawiać zadanie przed if i testować w wyniku tego zadania możesz połączyć zadanie a jeśli w jednym stwierdzeniu. Sprawdź swoje rozumienie tego, biorąc pod uwagę: Zrobiłby C++ interpretuje następujący warunek jako prawda czy fałsz?

```
if (10 == 10 == 10) ...
```

Bądź ostrożny! Na pierwszy rzut oka wydaje się to Prawdą; ale C++ to interpretuje jak False! Ponieważ operator == kojarzy się z lewej strony, program porównuje pierwsze 10 z drugim. Ponieważ one są równe, wynik wynosi 1 (dla True), a następnie 1 jest porównywany z trzecia 10 - co daje 0 (dla False)!

Przykłady

1. Poniżej podano przykłady prawidłowych instrukcji C++ if. Jeśli (zmienna sales jest większa niż 5000), to premia zmienna staje się równa 500.

```
if (sales > 5000)

{ bonus = 500; }
```

Jeśli jest to część programu C++, wartość w zmiennej sprzedaż określa, co będzie dalej. Jeśli sprzedaż zawiera więcej niż 5000, następną instrukcją, która się wykonuje, jest ta wewnątrz bloku, która inicjuje premię. Jeśli jednak sprzedaż zawiera 5000 lub mniej, blok nie jest wykonywany, a wiersz następujący po bloku if jest wykonywany. Jeśli (wiek zmienny jest mniejszy lub równy 21), to wyświetl Jesteś nieletni. na ekranie i przejdź do nowej linii, wydrukuj Jaka jest twoja ocena? na ekranie i zaakceptuj liczbę całkowitą z klawiatury.

```
if (age <= 21)

{ cout << “Jeste nieletni!.\n”;

cout << “Jaka jest Twoja ocena? “;

cin >> grade; }
```

Jeśli wartość wieku jest mniejsza lub równa 21, wiersze kodu w bloku są wykonywane w następnej kolejności. W przeciwnym razie C++ pominie cały blok i kontynuuje pracę z pozostałym programem. Jeśli (saldo zmiennej jest większe niż zmienna low_balance), to wydrukuj Przeterminowane! do ekranu i przesuń kursor do nowej linii.

```
if (balance > low_balance)

{cout << “Pzekroczymy termin !\n”; }
```

Jeśli wartość w bilansie jest większa niż w low_balance, wykonywanie programu jest kontynuowane w bloku i pojawia się komunikat „Przekroczony termin!” wyświetla na ekranie. Możesz porównać dwie zmienne ze sobą (jak w tym przykładzie) lub zmienną z literałem (jak w poprzednich przykładach) lub literałem z literałem (chociaż rzadko się to robi) lub literałem z dowolnym wyrażeniem w miejsce dowolnej zmiennej lub literału. Poniższa instrukcja if pokazuje wyrażenie zawarte w if. If (zmienna płaca pomnożona przez zmienną tax_rate równa się zmiennej minimum), to zmiennej low_salary przypisuje się 1400.60.

```
If (pay * tax_rate == minimum)

{ low_salary = 1400.60; }
```

Priorytety operatorów są na poziomach 11 i 12, niższych niż inni główne operatory matematyki. Kiedy używasz wyrażeń takich jak pokazane w tym przykładzie, możesz uczynić te wyrażenia o wiele bardziej czytelnymi, umieszczając je w nawiasach (nawet jeśli C++ tego nie wymaga). Oto przepis poprzedniej instrukcji if z dużymi nawiasami:

Jeśli (zmienna płaca (pomnożona przez zmienną stawka_opodatkowa) jest równa zmiennej minimalnej), to zmienna niska_płacenie ma przypisany 1400.60.

```
If ((pay * tax_rate) == minimum)

{ low_salary = 1400.60; }
```

2. Poniżej przedstawiono prosty program, który oblicza wynagrodzenie sprzedawcy. Sprzedawca otrzymuje zryczałtowaną stawkę 4,10 USD za godzinę. Ponadto, jeśli sprzedaż przekracza 8500 USD, sprzedawca otrzymuje również dodatkowe 500 USD jako bonus. Jest to wstępny przykład logiki warunkowej, która zależy od relacji między dwiema wartościami, sprzedażą i 8500 USD.

```
// Filename: C9PAY1.CPP

// Oblicza wynagrodzenie sprzedawcy na podstawie jego sprzedaży.

#include <iostream.h>

#include <stdio.h>

main()

{

char sal_name[20];

int hours;

float total_sales, bonus, pay;

cout << “\n\n”; // Print two blank lines.

cout << “Payroll Calculation\n”;

cout << “--------------------\n”;

// Ask the user for needed values.

cout << “What is salesperson’s last name? “;

cin >> sal_name;

cout << “How many hours did the salesperson work? “;

cin >> hours;

cout << “What were the total sales? “;

cin >> total_sales;

bonus = 0; // Initially, there is no bonus.

// Compute the base pay.

pay = 4.10 * (float)hours; // Type casts the hours.

// Add bonus only if sales were high.

if (total_sales > 8500.00)

{ bonus = 500.00; }

printf(“%s made $%.2f \n”, sal_name, pay);
```

printf(“and got a bonus of $%.2f”, bonus);

return 0;

}

Ten program używa cout, cin i printf () jako wejścia i wyjścia. Możesz je mieszać. Jeśli to zrobisz, dołącz odpowiednie pliki nagłówkowe (stdio.h i iostream.h). Poniższe dane wyjściowe pokazują wynik uruchomienia tego programu dwa razy, za każdym razem z różnymi wartościami wejściowymi. Zauważ, że program robi dwie różne rzeczy: oblicza premię dla jednego pracownika, ale nie dla drugiego. Bonus 500 $ jest bezpośrednim wynikiem wyciągu if. Przydział 500 $ do premii jest wykonywany tylko wtedy, gdy wartość w total_sales jest większa niż 8500 $.

Obliczanie listy płac

Jakie jest nazwisko sprzedawcy? Harrison

Ile godzin pracował sprzedawca? 40

Jaka była łączna sprzedaż? 6050,64

Harrison zarobił 164,00 $

i dostałem bonus 0,00 $

Obliczanie listy płac

Jakie jest nazwisko sprzedawcy? Robertson

Ile godzin pracował sprzedawca? 40

Jaka była łączna sprzedaż? 9800

Robertson zarobił 164,00 $

i dostałem bonus w wysokości 500,00 $

3. Podczas programowania sposobu wprowadzania danych przez użytkowników mądrze jest zaprogramować sprawdzanie poprawności danych na wpisanych wartościach. Jeśli wprowadzą złą wartość (na przykład liczbę ujemną, gdy dane wejściowe nie mogą być ujemne), możesz poinformować ich o problemie i poprosić o ponowne wprowadzenie danych. Oczywiście nie wszystkie dane można zweryfikować, ale większość z nich można sprawdzić pod kątem racjonalności. Na przykład, jeśli napiszesz program do prowadzenia dokumentacji ucznia, aby śledzić imię i nazwisko każdego ucznia, jego adres, wiek i inne istotne dane, możesz sprawdzić, czy wiek mieści się w rozsądnym zakresie. Jeśli użytkownik wprowadzi wiek 213, wiadomo, że wartość jest niepoprawna. Jeśli użytkownik wpisze -4 dla wieku, wiesz, że ta wartość jest również niepoprawna. Jednak nie wszystkie błędne dane dotyczące wieku można sprawdzić. Gdyby użytkownik miał na przykład 21 lat i wpisał 22, twój program nie ma możliwości dowiedzenia się, czy jest to poprawne, ponieważ 22 mieści się w rozsądnym przedziale wiekowym dla studentów. Poniższy program to procedura, która prosi o wiek i upewnia się, że ma więcej niż 10 lat. Z pewnością nie jest to niezawodny test (ponieważ użytkownik wciąż może wprowadzić niepoprawny wiek), ale dba o bardzo niskie wartości. Jeśli użytkownik wejdzie w zły wiek, program poprosi o to ponownie w instrukcji if.

```
// Filename: C9AGE.CPP

// Program zapewniający wartości wiekowe są rozsądne.

#include <stdio.h>

main()

{

int age;

printf("\nWhat is the student's age? ");

scanf(" %d", &age); // With scanf(), remember the &

if (age < 10)

{ printf("%c", '\x07'); // BEEP

printf("*** The age cannot be less than 10 ***\n");

printf("Try again...\n\n");

printf("What is the student's age? ");

scanf(" %d", &age);

}

printf("Thank you. You entered a valid age.");

return 0;

}
```

Ta procedura może być również częścią dłuższego programu. Dowiedz się później, jak wielokrotnie monitować o wartość, dopóki nie zostanie podany prawidłowy sygnał wejściowy. Ten program wykorzystuje dzwonek (ASCII 7), aby ostrzec użytkownika, że został wprowadzony zły wiek. Ponieważ znak \a jest sekwencją ucieczki dla alarmu, dlatego \a może zastąpić \x07 w tym programie. Jeśli wprowadzony wiek jest mniejszy niż 10, użytkownik otrzyma komunikat o błędzie. Program emituje sygnał dźwiękowy i ostrzega użytkownika o złym wieku, zanim poprosi o to ponownie. Poniżej przedstawiono wynik uruchomienia tego programu. Zauważ, że program „wie", ze względu na instrukcję if, czy wiek jest większy niż 10 lat.

```
What is the student's age? 3

*** The age cannot be less than 10 ***

Try again...

What is the student's age? 21

Thank you. You entered a valid age.
```

4. W przeciwieństwie do wielu języków, C++ nie zawiera operatora matematyki kwadratowej. Pamiętaj, że „podniesiesz do kwadratu" liczbę, mnożąc ją przez nią samą (na przykład 3 * 3). Ponieważ wiele komputerów nie pozwala, aby liczby całkowite zawierały więcej niż kwadrat 180, poniższy program używa instrukcji if, aby upewnić się, że liczba pasuje jako liczba całkowita. Program pobiera

wartość od użytkownika i drukuje jej kwadrat - chyba że jest większy niż 180. Komunikat * Kwadrat nie jest dozwolony dla liczb powyżej 180 * pojawia się na ekranie, jeśli użytkownik wpisze dużą liczbę.

```
// Filename: C9SQR1.CPP

// Wydrukuj kwadrat wartości wejściowej

// jeśli wartość wejściowa jest mniejsza niż 180

#include <iostream.h>

main()

{

int num, square;

cout << "\n\n"; // Print two blank lines.

cout << "What number do you want to see the square of? ";

cin >> num;

if (num <= 180)

{ square = num * num;

cout << "The square of " << num << " is " <<

square << "\n";

}

if (num > 180)

{ cout << '\x07'; // BEEP

cout << "\n* Square is not allowed for numbers over 180 *";

cout << "\nRun this program again trying a smaller value.";

}

cout << "\nThank you for requesting square roots.\n";

return 0;

}
```

Poniższe dane wyjściowe pokazują kilka przykładowych przebiegów tego programu. Zauważ, że oba warunki działają: Jeśli użytkownik wprowadzi liczbę mniejszą niż 180, pojawi się obliczony kwadrat, ale jeśli użytkownik wprowadzi większą liczbę, pojawi się komunikat o błędzie.

```
Jaką liczbę chcesz zobaczyć kwadrat? 45

Kwadrat 45 to 2025

Dziękujemy za podanie pierwiastków kwadratowych.

Jaką liczbę chcesz zobaczyć kwadrat? 212
```

* Kwadrat nie jest dozwolony dla liczb powyżej 180 * Uruchom ten program ponownie, próbując mniejszej wartości. Dziękujemy za podanie pierwiastków kwadratowych.

Możesz ulepszyć ten program za pomocą instrukcji else, której dowiesz się w dalszej części. Ten kod obejmuje nadmiarową kontrolę danych wejściowych użytkownika. Zmienna num musi zostać raz sprawdzona, aby wydrukować kwadrat, jeśli liczba wejściowa jest mniejsza lub równa 180, i ponownie sprawdzona pod kątem komunikatu o błędzie, jeśli jest większa niż 180.

5. Wartości 1 i 0 odpowiednio dla Prawda i Fałsz mogą pomóc Ci zaoszczędzić dodatkowy krok programowania, którego niekoniecznie jesteś w stanie zapisać w innych językach. Aby to zrozumieć, sprawdź następującą sekcję kodu:

commission = 0; // Initialize commission

if (sales > 10000)

{ commission = 500.00; }

pay = net_pay + commission; // Commission is 0 unless

// high sales.

Możesz zwiększyć wydajność tego programu, łącząc test relacyjny if, ponieważ wiesz, że jeśli zwróci 1 lub 0:

pay = net_pay + (commission = (sales > 10000) * 500.00);

Ta pojedyncza linia robi to, co zajęły poprzednie cztery linie. Ponieważ przypisanie po skrajnej prawej stronie ma pierwszeństwo, jest obliczane jako pierwsze. Program porównuje sprzedaż zmienną do 10000. Jeśli jest ona większa niż 10000, zwracany jest wynik True, 1. Następnie program mnoży 1 przez 500,00 i zapisuje wynik w prowizji. Jeśli jednak sprzedaż nie była większa niż 10000, wynik 0, a program otrzymuje 0 od pomnożenia 0 przez 500,00. Jakakolwiek wartość (500,00 lub 0) program przypisuje do prowizji, jest następnie dodawana do net_pay i zapisywana jako wypłata.

Instrukcja else

Instrukcja else nigdy nie pojawia się w programie bez instrukcji if. W tej sekcji przedstawiono instrukcję else, pokazując popularną instrukcję kombinacji if-else. Jego format to

if (warunek)

{Blok 1 lub więcej instrukcji C ++}

else

{Blok 1 lub więcej instrukcji C ++}

Pierwsza część instrukcji if-else jest identyczna z instrukcją if. Jeśli warunek jest spełniony, blok instrukcji C++ następujący po if jest wykonywany. Jeśli jednak warunek ma wartość False, zamiast tego wykonywany jest blok instrukcji C++ następujący po else. Podczas gdy prosta instrukcja if określa, co dzieje się tylko wtedy, gdy warunek jest spełniony, if-else określa również, co dzieje się, gdy warunek jest fałszywy. Bez względu na wynik, instrukcja następująca po if-else wykonuje się dalej. Poniżej opisano charakter if-else:

♦ Jeśli test warunków jest prawdziwy, wykonuje się cały blok instrukcji następujących po if.

♦ Jeśli testem warunku jest Fałsz, wykonywany jest cały blok instrukcji następujących po else.

UWAGA: Oprócz cyfr można także porównywać znaki. Podczas porównywania znaków C++ używa tabeli ASCII aby ustalić, która postać jest „mniejsza niż" druga (niższa w tabela ASCII). Ale nie można porównywać ciągów znaków lub tablic ciągów znaków bezpośrednio z operatorami relacyjnymi.

Przykłady

1. Poniższy program pyta użytkownika o liczbę. Następnie drukuje, czy liczba jest większa od zera, za pomocą instrukcji if-else.

```
// Filename: C9IFEL1.CPP

// Demonstruje if-else, wyświetlając, czy

// wartość wejściowa jest większa od zera lub nie.

#include <iostream.h>

main()

{

int num;

cout << "What is your number? ";

cin >> num; // Get the user's number.

if (num > 0)

{ cout << "More than 0\n"; }

else

{ cout << "Less or equal to 0\n"; }

// No matter what the number was, the following executes.

cout << "\n\nThanks for your time!\n";

return 0;

}
```

Nie ma potrzeby testowania obu możliwości, gdy używasz innej. If sprawdza, czy liczba jest większa od zera, a else automatycznie obsługuje wszystkie inne możliwości.

2. Poniższy program pyta użytkownika o jego imię, a następnie zapisuje je w tablicy znaków. Program sprawdza pierwszy znak tablicy, aby zobaczyć, czy należy do pierwszej połowy alfabetu. Jeśli tak, wyświetlony zostanie odpowiedni komunikat.

```
// Filename: C9IFEL2.CPP

// Testuje pierwszą inicjał użytkownika i wyświetla wiadomość.

#include <iostream.h>

main()
```

```
{

char last[20]; // Holds the last name.

cout << “What is your last name? “;

cin >> last;

// Test the initial

if (last[0] <= ‘P’)

{ cout << “Your name is early in the alphabet.\n”;}

else

{ cout << “You have to wait a while for “

<< “YOUR name to be called!\n”;}

return 0;

}
```

Zauważ, że ponieważ program porównuje element tablicy znaków z literałem znaku, literał znaku należy ująć w pojedyncze znaki cudzysłowu. Typ danych po każdej stronie każdego operatora relacyjnego musi być zgodny.

3. Poniższy program jest bardziej kompletną procedurą płacową niż drugi. Wykorzystuje instrukcję if, aby zilustrować sposób obliczania wynagrodzenia za nadgodziny. Logika wygląda następująco:

Jeśli pracownicy przepracują 40 godzin lub mniej, otrzymują regularne wynagrodzenie (stawka godzinowa pomnożona przez liczbę przepracowanych godzin). Jeśli pracownicy pracują od 40 do 50 godzin, otrzymują dodatkowo półtorej stawki godzinowej za te godziny powyżej 40, oprócz swojej regularnej płacy za pierwsze 40 godzin. Wszystkie godziny powyżej 50 są wypłacane według podwójnej stawki.

```
// Filename: C9PAY2.CPP

// Oblicz pełne możliwości wynagrodzenia za nadgodziny.

#include <iostream.h>

#include <stdio.h>

main()

{

int hours;

float dt, ht, rp, rate, pay;

cout << “\n\nHow many hours were worked? “;

cin >> hours;

cout << “\nWhat is the regular hourly pay? “;

cin >> rate;
```

```
// Oblicz wynagrodzenie tutaj

// Możliwość podwójnego czasu

if (hours > 50)

{ dt = 2.0 * rate * (float)(hours - 50);

ht = 1.5 * rate * 10.0;} // Time + 1/2 for 10 hours.

else

{ dt = 0.0; }// Either none or double for hours over 50.

// Time and a half.

if (hours > 40)

{ ht = 1.5 * rate * (float)(hours - 40); }

// Regular Pay

if (hours >= 40)

{ rp = 40 * rate; }

else

{ rp = (float)hours * rate; }

pay = dt + ht + rp; // Add three components of payroll.

printf("\nThe pay is %.2f", pay);

return 0;

}
```

4. Blok instrukcji następujący po if może zawierać dowolną prawidłową instrukcję C++ - nawet inną instrukcję if! Czasami jest to przydatne, jak pokazuje poniższy przykład. Możesz nawet użyć tego programu, aby nagradzać pracowników za ich wieloletnią służbę w Twojej firmie. W tym przykładzie dajesz złoty zegarek osobom z ponad 20-letnim stażem pracy, przycisk do papieru dla osób z ponad 10-letnim stukiem i poklepanie wszystkich po plecach!

```
// Filename: C9SERV.CPP

// Drukuje wiadomość w zależności od lat pracy.

#include <iostream.h>

main()

{

int yrs;

cout << "How many years of service? ";

cin >> yrs; // Determine the years they have worked.
```

```
if (yrs > 20)
{ cout << “Give a gold watch\n”; }
else
{ if (yrs > 10)
{ cout << “Give a paper weight\n”; }
else
{ cout << “Give a pat on the back\n”; }
}
return 0;
}
```

Nie polegaj na if w if, aby obsłużyć zbyt wiele warunków, ponieważ więcej niż trzy lub cztery warunki mogą powodować zamieszanie. Możesz popsuć logikę, na przykład: „Jeśli to prawda, a jeśli to prawda, to zrób coś; ale jeśli nie to, ale coś innego jest Prawdą, to ... ”(i tak dalej). Instrukcja switch, o której dowiesz się później, obsługuje te typy wielokrotności, jeśli selekcje są znacznie lepsze niż długie, jeśli zawiera się w instrukcji if.

Ćwiczenia

1. Napisz program kalkulatora pogody, który poprosi o listę temperatur z ostatnich pięciu dni, a następnie wydrukuje Brrrr! za każdym razem, gdy temperatura spada poniżej zera.

2. Napisz program, który prosi o liczbę, a następnie wypisuje kwadrat i sześcian (liczbę pomnożoną przez siebie trzy razy) liczby, którą wpisujesz, jeśli liczba ta jest większa niż 1. W przeciwnym razie program nic nie wydrukuje.

3. W programie poproś użytkownika o dwie cyfry. Wydrukuj komunikat informujący, jak pierwszy odnosi się do drugiego. Innymi słowy, jeśli użytkownik wprowadzi 5 i 7, program wyświetli „5 jest mniejsze niż 7.”

4. Napisz program, który monituje użytkownika o wynagrodzenie przed opodatkowaniem i drukuje odpowiednie podatki. Podatki wynoszą 10 procent, jeśli pracownik zarabia mniej niż 10 000 USD; 15 procent, jeśli pracownik zarabia 10 000 USD do 20 000 USD, ale nie obejmuje; i 20 procent, jeśli pracownik zarabia 20 000 USD lub więcej.

Podsumowanie

Masz teraz narzędzia do pisania potężnych programów do sprawdzania danych. Pokazano, jak porównać literały, zmienne i kombinacje obu przy użyciu operatorów relacyjnych. Instrukcje if i if-else opierają się na takich porównaniach danych w celu ustalenia, który kod wykonać następnie. Teraz możesz warunkowo wykonywać instrukcje w swoich programach.

Operatory logiczne

Operatory logiczne C++ pozwalają łączyć operatory relacyjne w bardziej rozbudowane instrukcje do testowania danych. Operatory logiczne są czasem nazywane złożonymi operatorami relacyjnymi. Jak pokazuje tabela pierwszeństwa w C++, operatory relacyjne mają pierwszeństwo przed operatorami logicznymi podczas ich łączenia. Tabela pierwszeństwa odgrywa ważną rolę w tego typu operatorach, jak podkreśla ten rozdział. Tu zapoznasz się z

♦ Operatorami logicznymi

♦ Jak wykorzystywane są operatory logiczne

♦ W jaki sposób operatorzy logiczni mają pierwszeństwo

Zakończymy badanie testowania warunkowego, które umożliwia C++, i ilustruje wiele przykładów instrukcji if w programach, które działają na warunkach złożonych testów.

Definiowanie operatorów logicznych

Może się zdarzyć, że będziesz musiał przetestować więcej niż jeden zestaw zmiennych. Możesz połączyć więcej niż jeden test relacyjny w złożony test relacyjny, używając operatorów logicznych C++

Operator: Znaczenie

&& : AND

|| : OR

! : NIE

Pierwsze dwa operatory logiczne, && i ||, nigdy nie pojawiają się same. Zazwyczaj przechodzą między co najmniej dwoma testami relacyjnymi. Poniżej pokazano, jak działa każdy operator logiczny. Tabele te nazywane są tabelami prawdy, ponieważ pokazują, jak osiągnąć wyniki True z instrukcji if korzystającej z tych operatorów. Poświęć trochę czasu na przestudiowanie tych tabel.

Tabele prawdy.

Tabela prawdy AND (&&)

(Obie strony muszą być prawdziwe)

True AND True = True

True AND False = False

False AND True = False

False AND False = False

Tabela prawdy OR (||)

(Jedna lub druga strona musi mieć wartość True)

True OR True = True

True OR False = True

False OR True = True

False OR False = False

Tabela prawdy NOT (!)

(Powoduje odwrotną relację)

NOT True = False

NOT False = True

Operatory logiczne i ich zastosowania

Prawda i fałsz po każdej stronie operatorów reprezentują relacyjny test if. Na przykład następujące instrukcje są poprawne, jeśli testy wykorzystujące operatory logiczne (czasami nazywane złożonymi operatorami relacyjnymi). Jeśli zmienna a jest mniejsza niż zmienna b, a zmienna c jest większa niż zmienna d, wówczas wyniki drukowania są nieprawidłowe. do ekranu.

```
if ((a < b) && (c > d))

{ cout << “Results are invalid.”; }
```

Zmienna a musi być mniejsza niż b, a jednocześnie c musi być większa niż d, aby można było wykonać printf(). Instrukcja if nadal wymaga nawiasów wokół pełnego testu warunkowego. Rozważ tę część programu:’=

```
if ((sales > 5000) || (hrs_worked > 81))

{ bonus=500; }
```

Sprzedaż musi być większa niż 5000 lub hrs_worked musi być większa niż 81, zanim zadanie zostanie wykonane.

```
if (!(sales < 2500))

{ bonus = 500; }
```

Jeśli sprzedaż jest większa lub równa 2500, premia jest inicjowana. To ilustruje ważną wskazówkę programistyczną: Użyj! oszczędnie. Lub, jak to mądrze ujmują niektórzy profesjonaliści: „Nie używaj! albo twoje programy nie będą! (niejasne). ” O wiele łatwiej jest przepisać poprzedni przykład, przekształcając go w pozytywny test relacyjny:

```
if (sales >= 2500)

{ bonus 500; }
```

Ale operator ! jest czasem pomocny, szczególnie podczas testowania warunków końca pliku dla plików dyskowych, o czym dowiesz się w rozdziale 30, „Pliki sekwencyjne”. Jednak w większości przypadków można tego uniknąć! za pomocą odwrotnej logiki przedstawionej poniżej:

!(var1 == var2) is the same as (var1 != var2)

!(var1 <= var2) is the same as (var1 > var2)

!(var1 >= var2) is the same as (var1 < var2)

!(var1 != var2) is the same as (var1 == var2)

!(var1 > var2) is the same as (var1 <= var2)

!(var1 < var2) is the same as (var1 >= var2)

Zauważ, że ogólny format instrukcji if jest zachowywany, gdy używasz operatorów logicznych, ale test relacyjny rozszerza się o więcej niż jedną relację. Możesz nawet mieć trzy lub więcej, jak w poniższym stwierdzeniu:

if ((a == B) && (d == f) || (l = m) || !(k <> 2)) ...

Jest to jednak trochę za dużo, a dobra praktyka programowania nakazuje stosowanie co najwyżej dwóch testów relacyjnych w jednej instrukcji if. Jeśli musisz połączyć więcej niż dwa, użyj do tego więcej niż jednej instrukcji if. Podobnie jak w przypadku innych operatorów relacyjnych, w codziennej rozmowie używasz również następujących operatorów logicznych.

„Jeśli moja płaca jest wysoka, a czas wakacji długi, możemy lecieć do Włoch tego lata".

„Jeśli wyniesiesz śmieci lub posprzątasz swój pokój, możesz dziś wieczorem oglądać telewizję."

„Jeśli nie jesteś dobry, zostaniesz ukarany".

Prawdy wewnętrzne

Prawdziwe lub fałszywe wyniki testów relacyjnych występują wewnętrznie na poziomie bitów. Na przykład weź test if:

if (a == 6) ...

aby ustalić prawdziwość relacji, (a == 6). Komputer pobiera wartość binarną 6 lub 00000110 i porównuje ją krok po kroku ze zmienną a. Jeśli a zawiera 7, binarny 00000111, wynikiem tego równego testu jest Fałsz, ponieważ prawy bit (zwany bitem najmniej znaczącym) jest inny.

Wydajność logiczna C++

C++ stara się być bardziej wydajny niż inne języki. Jeśli połączysz wiele testów relacyjnych z jednym z operatorów logicznych,

C++ nie zawsze interpretuje pełne wyrażenie. To ostatecznie sprawia, że twoje programy działają szybciej, ale istnieją zagrożenia! Na przykład, jeśli twój program przejdzie test warunkowy:

if ((5 > 4) || (sales < 15) && (15 != 15))...

C++ ocenia tylko pierwszy warunek (5 > 4) i zdaje sobie sprawę, że nie musi szukać dalej. Ponieważ (5 > 4) jest Prawdą, a ponieważ || (LUB) wszystko, co następuje po nim, jest nadal Prawdą, C++ nie przejmuje się resztą wyrażenia. To samo dotyczy następującej wypowiedzi:

if ((7 < 3) && (age > 15) && (initial == 'D'))...

Tutaj C++ ocenia tylko pierwszy warunek, czyli Fałsz. Ponieważ && (AND) wszystko, co następuje po nim, jest również False, C++ nie interpretuje wyrażenia po prawej stronie (7 < 3). W większości przypadków nie stanowi to problemu, ale pamiętaj, że poniższe wyrażenie może nie spełniać Twoich oczekiwań:

if ((5 > 4) || (num = 0))...

Przypisanie (num = 0) nigdy się nie wykonuje, ponieważ C++ musi interpretować tylko (5> 4), aby ustalić, czy całe wyrażenie jest Trur czy False. Z powodu tego niebezpieczeństwa nie należy dołączać wyrażeń przypisania w tym samym stanie, co test logiczny. Następujący singiel, jeśli warunek:

if ((sales > old_sales) || (inventory_flag = ‘Y’))...

należy podzielić na dwie wypowiedzi, takie jak:

if ((sales > old_sales) || (inventory_flag))...

więc tagowi zapasowego zawsze przypisywana jest wartość „Y”, bez względu na to, w jaki sposób testy wyrażenia (sales > old_sales)

Przykłady

1. Letnie Igrzyska Olimpijskie odbywają się co cztery lata w ciągu każdego roku, który jest podzielny równomiernie przez 4. Spis powszechny Stanów Zjednoczonych jest przeprowadzany co 10 lat, w każdym roku, który jest równomiernie podzielny przez 10. Poniższy krótki program wymaga jednego roku, a następnie informuje użytkownika, czy jest to rok letnich igrzysk olimpijskich, rok spisu powszechnego, czy jedno i drugie. Wykorzystuje operatory relacyjne, operatory logiczne i operator modułu do określenia tego wyniku.

```
// Filename: C10YEAR.CPP

// Określa, czy to rok letnich igrzysk olimpijskich,

// Rok spisu powszechnego w USA lub oba.

#include <iostream.h>

main()

{

int year;

// Ask for a year

cout << “What is a year for the test? “;

cin >> year;

// Test the year

if (((year % 4)==0) && ((year % 10)==0))

{ cout << “Both Olympics and U.S. Census!”;

return 0; } // Quit program, return to operating

// system.

if ((year % 4)==0)

{ cout << “Summer Olympics only”; }

else

{ if ((year % 10)==0)

{ cout << “U.S. Census only”; }

}
```

```
return 0;

}
```

2. Teraz, gdy wiesz o relacjach złożonych, możesz napisać program sprawdzający wiek, taki jak ten o nazwie C9AGE.CPP. Ten program zapewnił, że wiek będzie powyżej 10 lat. Jest to kolejny sposób na sprawdzenie poprawności danych wejściowych pod kątem racjonalności. Poniższy program zawiera operator logiczny w celu ustalenia, czy wiek jest większy niż 10, czy mniejszy niż 100. Jeśli którykolwiek z nich jest w takim przypadku program stwierdza, że użytkownik nie podał prawidłowego wieku.

```
// Filename: C10AGE.CPP

// Program, który pomaga zapewnić rozsądne wartości wiekowe.

#include <iostream.h>

main()

{

int age;

cout << “What is your age? “;

cin >> age;

if ((age < 10) || (age > 100))

{ cout << “ \x07 \x07 \n”; // Beep twice

cout << “*** The age must be between 10 and”

“100 ***\n”; }

else

{ cout << “You entered a valid age.”; }

return 0;

}
```

3. Poniższy program może zostać wykorzystany przez sklep wideo do obliczenia rabatu na podstawie liczby transakcji wynajmowanych przez osoby oraz ich statusu klienta. Klienci są klasyfikowani albo jako R dla zwykłych, albo S dla specjalnych. Klienci specjalni są członkami klubu wynajmu od ponad roku. Automatycznie otrzymują 50-procentową zniżkę na wszystkie czynsze. Sklep organizuje również „dni wartości” kilka razy w roku. W dni wartościowe wszyscy klienci otrzymują 50-procentową zniżkę. Klienci specjalni nie otrzymują dodatkowych 50 centów zniżki w dni wartościowe, ponieważ każdy dzień jest dla nich zniżką. Program pyta o status każdego klienta i czy jest to dzień wartości. Następnie używa || związek z testem na zniżkę. Nawet zanim zacząłeś uczyć się C++, prawdopodobnie spojrzałbyś na ten problem z myślą o następującym pomyśle.

„Jeśli klient jest Specjalny lub jeśli jest to dzień wartości, odejmij 50 centów od czynszu.”

To w zasadzie pomysł decyzji if w następnym programie. Mimo że klienci specjalni nie otrzymują dodatkowej zniżki na dni wartościowe, istnieje jeden finał, jeśli przetestuje się je, aby wydrukować dodatkową wiadomość na dole wskazanego rozliczenia na ekranie.

```
// Filename: C10VIDEO.CPP
// Program that computes video rental amounts and gives
// appropriate discounts based on the day or customer status.
#include <iostream.h>
#include <stdio.h>
main()
{
float tape_charge, discount, rental_amt;
char first_name[15];
char last_name[15];
int num_tapes;
char val_day, sp_stat;
cout << "\n\n *** Video Rental Computation ***\n";
cout << " ------------------------\n";
// Underline title
tape_charge = 2.00;
// Before-discount tape fee-per tape.
// Receive input data.
cout << "\nWhat is customer's first name? ";
cin >> first_name;
cout << "What is customer's last name? ";
cin >> last_name;
cout << "\nHow many tapes are being rented? ";
cin >> num_tapes;
cout << "Is this a Value day (Y/N)? ";
cin >> val_day;
cout << "Is this a Special Status customer (Y/N)? ";
cin >> sp_stat;
// Calculate rental amount.
discount = 0.0; // Increase discount if they are eligible.
if ((val_day == 'Y') || (sp_stat == 'Y'))
```

```
{ discount = 0.5;
rental_amt=(num_tapes*tape_charge)
(discount*num_tapes); }
// Print the bill.
cout << "\n\n** Rental Club **\n\n";
cout << first_name << " " << last_name << " rented "
<< num_tapes << " tapes\n";
printf("The total was %.2f\n", rental_amt);
printf("The discount was %.2f per tape\n", discount);
// Print extra message for Special Status customers.
if (sp_stat == 'Y')
{ cout << "\nThank them for being a Special "
<< "Status customer\n";}
return 0;
}
```

Dane wyjściowe tego programu są wyświetlane poniżej. Zwróć uwagę, że klienci specjalni mają dodatkowy komunikat na dole ekranu. Ten program, ze względu na instrukcje if, działa różnie w zależności od wprowadzonych danych. Dla stałych klientów nie przysługuje zniżka w nieważne dni.

```
*** Video Rental Computation ***
-----------------------
What is customer's first name? Jerry
What is customer's last name? Parker
How many tapes are being rented? 3
Is this a Value day (Y/N)? Y
Is this a Special Status customer (Y/N)? Y
** Rental Club **
Jerry Parker rented 3 tapes
The total was 4.50
The discount was 0.50 per tape
Thank them for being a Special Status customer
```

Operatory logiczne i ich pierwszeństwo

Kolejność pierwszeństwa matematyki, nie obejmowała operatorów logicznych. Można się zastanawiać, dlaczego operatory relacyjne i logiczne są zawarte w tabeli pierwszeństwa. Poniższa instrukcja pokazuje, dlaczego:

if ((sales < min_sal * 2 && yrs_emp > 10 * sub) ...

Bez pełnej kolejności operatorów niemożliwe jest określenie, w jaki sposób wykona się takie oświadczenie. Zgodnie z kolejnością pierwszeństwa ta instrukcja if jest wykonywana w następujący sposób:

if ((sales < (min_sal * 2)) && (yrs_emp > (10 * sub))) ...

To wciąż może być mylące, ale mniej. Najpierw wykonywane są dwa mnożenia, a następnie relacje <i>. && jest wykonywany jako ostatni, ponieważ jest najniższy w kolejności priorytetów operatorów. Aby uniknąć takich niejednoznacznych problemów, pamiętaj o stosowaniu obszernych nawiasów - nawet jeśli Twoim zamiarem jest domyślny porządek pierwszeństwa. Mądrze jest również opierać się łączeniu zbyt wielu wyrażeń w jednym teście relacyjnym. Zauważ, że || (OR) ma niższy priorytet niż && (AND). Dlatego następujące testy są równoważne:

if ((first_initial==’A’) && (last_initial==’G’) || (id==321)) ...

if (((first_initial==’A’) && (last_initial==’G’)) || (id==321)) ...

Drugi jest jaśniejszy ze względu na nawiasy, ale tabela pierwszeństwa czyni je identycznymi.

Ćwiczenia

1. Napisz program (za pomocą pojedynczej instrukcji if jeśli), aby ustalić, czy użytkownik wprowadza nieparzystą liczbę dodatnią.

2. Napisz program, który prosi użytkownika o dwa inicjały. Wyświetl komunikat informujący użytkownika, czy pierwsza litera spada alfabetycznie przed drugą.

3. Napisz grę w zgadywanie liczb. Przypisz wartość do zmiennej o nazwie numer u góry programu. Podaj monit z prośbą o pięć odpowiedzi. Uzyskaj pięć domysłów użytkownika za pomocą pojedynczego scanf () do ćwiczenia z scanf (). Sprawdź, czy któreś z domysłów pasują do numeru i wydrukuj odpowiedni komunikat, jeśli tak się stanie.

4. Napisz procedurę obliczania podatku w następujący sposób: Rodzina nie płaci podatku, jeśli jej dochód jest mniejszy niż 5000 USD. Płaci podatek w wysokości 10 procent, jeśli jego dochód wynosi od 5000 do 9 999 USD włącznie. Płaci podatek w wysokości 20 procent, jeśli dochód wynosi od 10 000 do 19 999 USD włącznie. W przeciwnym razie płaci 30-procentowy podatek.

Podsumowanie

Rozszerzono instrukcję if o znaki &&, || i! operatory logiczne. Te operatory umożliwiają połączenie kilku testów relacyjnych w jednym teście. C++ nie zawsze musi spójrzeć na każdy operator relacyjny, gdy połączysz je w wyrażenie. Zawiera wyjaśnienie instrukcji if.

Dodatkowe operatory C++

C++ ma kilka innych operatorów, których powinieneś nauczyć się oprócz tych, których nauczyłeś się wcześniej. W rzeczywistości C++ ma więcej operatorów niż większość języków programowania. Jeśli nie zapoznasz się z nimi, możesz myśleć, że programy C++ są tajemnicze i trudne do naśladowania. Duże uzależnienie C++ od operatorów i ich pierwszeństwa zapewnia wydajność, która pozwala na płynniejsze i szybsze działanie programów.

Ta sekcja nauczy Cię, co następuje:

♦?: Operator warunkowy

♦ Operator przyrostowy ++

♦ Operator dekrementacji —

♦ Operatora sizeof

♦ Operator przecinka (,)

♦ Operatory bitowe (&, | i ^)

Operator warunkowy

Operator warunkowy jest jedynym trójskładnikowym operatorem C++, wymagającym trzech operandów (w przeciwieństwie do wymagań jednoargumentowego i podwójnego argumentu binarnego). W niektórych sytuacjach operator warunkowy zastępuje logikę if-else. Operator warunkowy jest dwuczęściowym symbolem?: O następującym formacie:

wyrażenie warunkowe? wyrażenie1: wyrażenie2;

Wyrażenie warunkowe to dowolne wyrażenie w C++, które daje odpowiedź True (niezerową) lub False (zero). Jeśli wynikiem wyrażenia warunkowego jest Prawda, wyrażenie1 zostanie wykonane. W przeciwnym razie, jeśli wynikiem wyrażenia warunkowego jest False, wyrażenie2 zostanie wykonane. Wykonywane jest tylko jedno wyrażenie po znaku zapytania. Na końcu wyrażenia2 pojawia się tylko pojedynczy średnik. Wyrażenia wewnętrzne, takie jak wyrażenie1, nie mają średnika. Jeśli potrzebujesz prostej logiki „jeśli-inaczej", operator warunkowy zazwyczaj zapewnia bardziej bezpośrednią i zwięzłą metodę, chociaż zawsze powinieneś preferować czytelność niż kompaktowy kod. Aby zobaczyć operator warunkowy w pracy, rozważ następującą sekcję kodu.

```
if (a > b)

{ ans = 10; }

else

{ ans = 25; }
```

Możesz łatwo przepisać tego rodzaju kod if-else za pomocą jednego operatora warunkowego. Jeśli zmienna a jest większa niż zmienna b, ustaw zmienną ans na 10; w przeciwnym razie ustaw ans na 25.

```
A > b? (ans = 10): (ans = 25);
```

Chociaż nawiasy nie są wymagane w pobliżu wyrażenia warunkowego, aby działało, zwykle poprawiają czytelność. Czytelność tego stwierdzenia jest poprawiona poprzez użycie nawiasów, jak następuje:

(a> b)? (ans = 10): (ans = 25);

Ponieważ każde wyrażenie C++ ma wartość - w tym przypadku przypisywana wartość - to stwierdzenie może być jeszcze bardziej zwięzłe, bez utraty czytelności, poprzez przypisanie odpowiedzi z lewej strony warunku:

ans = (a> b)? (10): (25);

To wyrażenie mówi: Jeśli a jest większe niż b, przypisz 10 do ans; w przeciwnym razie przypisz 25 do ans. Niemal każda instrukcja if-else może zostać przepisana jako warunkowa i odwrotnie. Powinieneś poćwiczyć konwersję jednego do drugiego, aby zapoznać się z warunkami celu operatora.

UWAGA: Każda poprawna instrukcja C++ może być również wyrażeniem warunkowym, w tym wszystkie operatory relacyjne i logiczne, a także dowolne ich możliwe kombinacje.

Przykłady

1. Załóżmy, że przeglądasz swoje wczesne programy w C++ i zauważysz następującą sekcję kodu.

if (production > target)

{ target *= 1.10; }

else

{ target *= .90; }

Powinieneś zdawać sobie sprawę, że taką prostą instrukcję if-else można przepisać za pomocą operatora warunkowego i że skutkuje to wydajniejszym kodem. Możesz zatem zmienić go na następującą pojedynczą instrukcję.

(production > target) ? (target *= 1.10) : (target *= .90);

2. Za pomocą operatora warunkowego możesz napisać procedurę, aby znaleźć minimalną wartość między dwiema zmiennymi. Czasami nazywa się to rutyną minimum. Instrukcja , aby to zrobić, to

minimum = (var1 < var2) ? var1 : var2;

Jeśli var1 jest mniejsze niż var2, wartość var1 jest przypisana do minimum. Jeśli var2 jest mniejsze, wartość var2 jest przypisana do minimum. Jeśli zmienne są równe, wartość var2 jest przypisywana do minimum, ponieważ nie ma znaczenia, który jest przypisany.

3. Równie łatwo można napisać maksymalną procedurę:

maximum = (var1> var2)? var1: var2;

4. Idąc dalej o poprzednie przykłady, możesz również przetestować znak zmiennej. Poniższe wyrażenie warunkowe przypisuje –1 do zmiennej o nazwie znak, jeśli testvar jest mniejsze niż 0; 0, aby podpisać, jeśli testvar ma wartość zero; i +1, aby podpisać, jeśli testvaris 1 lub więcej.

sign = (testvar < 0) ? -1 : (testvar > 0);

Łatwo jest zauważyć, dlaczego wynik „mniej niż” daje wynik –1, ale druga część wyrażenia może być myląca. Działa to dobrze, ponieważ C++ 1 i 0 (odpowiednio dla True i False) zwracają wartości z testu relacyjnego. Jeśli testvar ma wartość 0 lub więcej, znak ma przypisaną odpowiedź (testvar> 0). Wartość (testvar> 0) wynosi 1, jeśli True (dlatego testvar jest większy niż 0) lub 0, jeśli testvar jest równy 0.

Poprzednie oświadczenie pokazuje wydajny operator warunkowy C++. Może to również pomóc zrozumieć, jeśli piszesz instrukcję przy użyciu typowej logiki if-else. Oto ten sam problem napisany z typową instrukcją if-else:

if (testvar < 0)

{ sign = -1; }

else

{ sign = (testvar > 0); } // testvar może być tylko

// 0 lub więcej tu

Operatory inkrementacji i dekrementacji

C++ oferuje dwa unikalne operatory, które dodają lub odejmują 1 do zmiennych lub od nich. Są to operatory zwiększania i zmniejszania: ++ i —. Tabela 11.1 pokazuje, w jaki sposób operatory te odnoszą się do innych typów wyrażeń, które widziałeś. Zauważ, że ++ i — mogą pojawić się po obu stronach zmodyfikowanej zmiennej. Jeśli ++ lub — pojawi się po lewej stronie, jest znany jako operator prefiksu. Jeśli operator pojawia się po prawej stronie, jest operatorem postfiksowym.

Operator: Przykład: Opis: Oświadczenia równoważne

++: i ++; : postfiks: i = i + 1; i + = 1;

++: ++ i; : przedrostek: i = i + 1; i + = 1;

--: i--; : postfiks: i = i - 1; i - = 1;

-- : --i; : przedrostek: i = i - 1; i - = 1;

Za każdym razem, gdy musisz dodać 1 lub odjąć 1 od zmiennej, możesz użyć tych dwóch operatorów. Jak pokazuje Tabela ,jeśli musisz zwiększyć lub zmniejszyć tylko jedną zmienną, te operatory ci to umożliwiają.

Skuteczność przyrostu i zmniejszenia

Operatory inkrementacji i dekrementacji to proste, wydajne metody dodawania 1 do zmiennej i odejmowania 1 od zmiennej. Często musisz to zrobić podczas liczenia lub przetwarzania pętli. Te dwa operatory kompilują się bezpośrednio do swoich odpowiedników języka asemblera. Prawie wszystkie komputery zawierają, w najniższym binarnym języku maszynowym, instrukcje zwiększania i zmniejszania. Jeśli używasz operatorów inkrementacji i dekrementacji w C++, upewniasz się, że kompilują się one z tymi równoważnikami niskiego poziomu. Jeśli jednak kodujesz wyrażenia w celu dodania lub odjęcia 1 (tak jak robisz to w innych językach programowania), takie jak wyrażenie i = i - 1, tak naprawdę nie upewniasz się, że C++ skompiluje tę instrukcję w jej wydajnym odpowiedniku w języku maszynowym. To, czy użyjesz przedrostka, czy przyrostka, nie ma znaczenia - jeśli samodzielnie zwiększasz lub zmniejszasz pojedyncze zmienne w liniach. Jednak łącząc te dwa operatory z innymi operatorami w jednym wyrażeniu, musisz zdawać sobie sprawę z ich różnic. Rozważ następującą sekcję programu. Tutaj wszystkie zmienne są liczbami całkowitymi, ponieważ operatory inkrementacji i dekrementacji działają tylko na zmiennych całkowitych. Zrób równy 6. Przyrost a, odejmij 1 od niego, a następnie przypisz wynik do b.

a = 6;

b = ++ a - 1;

Jakie są wartości a i b po zakończeniu tych dwóch instrukcji? Wartość a jest łatwa do ustalenia: jest zwiększana w drugim wyrażeniu, więc wynosi 7. Jednak b wynosi albo 5, albo 6, w zależności od tego, kiedy zmienna a zwiększa. Aby określić, kiedy przyrosty, rozważ następującą zasadę:

♦ Jeśli zmienna jest zwiększana lub zmniejszana za pomocą operatora prefiksu, przyrost lub spadek występuje przed użyciem wartości zmiennej w pozostałej części wyrażenia.

♦ Jeśli zmienna jest zwiększana lub zmniejszana za pomocą operatora Postfiksa, to zwiększenie lub zmniejszenie ma miejsce po użyciu wartości zmiennej w pozostałej części wyrażenia.

W poprzednim kodzie a zawiera przyrost prefiksu. Dlatego jego wartość najpierw zwiększa się do 7, a następnie 1 odejmuje się od 7, a wynik (6) przypisuje się do b. Jeśli używany jest przyrostek pofiksowy, jak w

a = 6;

b = a ++ - 1;

a wynosi 6, dlatego 5 jest przypisane do b, ponieważ a nie zwiększa się do 7, dopóki jego wartość nie zostanie użyta w wyrażeniu. Tabela pierwszeństwa w Dodatku D, „Tabela pierwszeństwa C++” pokazuje, że operatory prefiksów mają znacznie wyższy priorytet niż prawie każdy inny operator, szczególnie przyrosty i dekrementacje postfiksów o niskim priorytecie.

WSKAZÓWKA: Jeśli porządek prefiksu i postfiksu powoduje zamieszanie, podziel wyrażenia na dwa wiersze kodu i wpisz przyrost lub spadek przed lub po wyrażeniu, które go używa.

Korzystając z tej wskazówki, możesz teraz przepisać poprzedni przykład w następujący sposób:

a = 6;

b = a - 1;

a ++;

Nie ma teraz wątpliwości, kiedy jest zwiększane: przyrosty po przypisaniu b do a-1. Nawet nawiasy nie mogą zastąpić reguły Postfiks. Rozważ następujące oświadczenie.

x = p + (((amt ++)));

Jest tu zbyt wiele niepotrzebnych nawiasów, ale nawet zbędne nawiasy nie wystarczą, aby zwiększyć wartość amt przed dodaniem jej wartości do p. Przyrosty i przedrostki zawsze występują po tym, jak ich zmienne zostaną użyte w otaczającym wyrażeniu.

UWAGA: Nie próbuj zwiększać ani zmniejszać wyrażenia. Możesz zastosować te operatory tylko do zmiennych. Następujące wyrażenie jest nieprawidłowe:

sales = ++(rate * hours); // Nie dozwolony!!

Przykłady

1. Podobnie jak w przypadku wszystkich innych operatorów C++, pamiętaj o tabeli pierwszeństwa, oceniając wyrażenia zwiększające i zmniejszające się. Rysunki poniższe pokazują kilka przykładów ilustrujących te operatory.

2. Tabela pierwszeństwa nabiera jeszcze większego znaczenia, gdy zobaczysz sekcję kodu, taką jak ta pokazana na rysunku

3. Biorąc pod uwagę tabelę pierwszeństwa - i, co ważniejsze, co wiesz o relacyjnych wydajnościach C++ - jaka jest wartość ans w poniższej sekcji kodu?

int i = 1, j = 20, k = -1, l = 0, m = 1, n = 0, o = 2, p = 1;

ans = i || j– – && k ++ || ++ l&& ++ m || n– – &! o || p– –;

Z początku wydaje się to niezwykle skomplikowane. Niemniej jednak możesz po prostu na niego spojrzeć i określić wartość ans, a także wartość końcową pozostałych zmiennych. Przypomnij sobie, że gdy C++ wykonuje relację || (lub), ignoruje prawą stronę || jeśli lewą wartością jest True (każda niezerowa wartość to True). Ponieważ każda niezerowa wartość ma wartość True, C++ nie ocenia wartości po prawej stronie. Dlatego C++ wykonuje to wyrażenie, jak pokazano:

ans = i || j– – && k ++ || ++ l&& ++ m || n– – &! o || p– –;

|

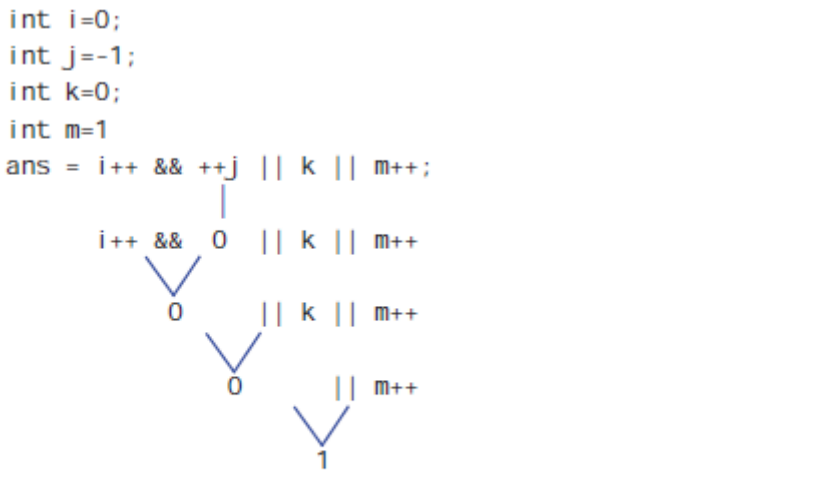
1 (PRAWDA)

```
int i=1;
int j=2;
int k=3;
ans = i++ * j - --k;
              |
      i++ * j -   2
        \
         2    -   2
              \
               0
```

ans = 0, then i increments by 1 to its final value of 2.

```
int i=1;
int j=2;
int k=3;
ans = ++i * j - k--;
        |
        2 * j - k--
          \
           4    - k--
                \
                 1
```

ans = 1, then k decrements by 1 to its final value of 2.



ans = 1, then i increments by 1 to its final value of 1,
and m increments by 1 to its final value of 2.

UWAGA: Ponieważ i jest Prawdą, C++ ocenia całe wyrażenie jako Prawda i ignoruje cały kod po pierwszym ||. Dlatego każde inne wyrażenie inkrementacji i dekrementacji jest ignorowane. Ponieważ C++ ignoruje inne wyrażenia, to wyrażenie zmienia tylko ans. Pozostałe zmienne, od j do p, nigdy nie są zwiększane ani zmniejszane, nawet jeśli kilka z nich zawiera operatory zwiększania i zmniejszania. Jeśli korzystasz z operatorów relacyjnych, pamiętaj o tym problemie i podziel wszystkie operatory inkrementacji i dekrementacji na instrukcje samodzielnie, umieszczając je w wierszach przed instrukcjami relacyjnymi, które używają ich wartości

Opeator sizeof

W C++ istnieje inny operator, który wcale nie wygląda jak operator. Wygląda jak wbudowana funkcja, ale nazywa się operatorem sizeof. W rzeczywistości, jeśli myślisz o sizeof jako wywołaniu funkcji, możesz się nie pomylić, ponieważ działa on w podobny sposób. Format sizeof wygląda następująco:

sizeof dane

lub

sizeof (typ danych)

Operator sizeof jest jednoargumentowy, ponieważ działa na jednej wartości. Ten operator generuje wynik reprezentujący rozmiar w bajtach danych lub określonego typu danych. Ponieważ większość typów danych i zmiennych wymaga różnych ilości pamięci wewnętrznej na różnych komputerach, operator sizeof umożliwia programom zachowanie spójności na różnych typach komputerów.

WSKAZÓWKA: Większość programistów C++ używa nawiasów wokół argumentu sizeof, niezależnie od tego, czy argumentem jest dane, czy typ danych. Ponieważ musisz używać nawiasów wokół argumentów typu danych i możesz ich używać wokół argumentów danych, nie zaszkodzi zawsze ich używać.

Operator sizeof jest czasem nazywany operatorem czasu kompilacji. W czasie kompilacji, a nie w czasie wykonywania, kompilator zamienia każde wystąpienie sizeof w twoim programie na wartość liczby całkowitej bez znaku . Ponieważ sizeof jest częściej używany w zaawansowanym programowaniu w C++, ten operator jest lepiej wykorzystywany w dalszej części do wykonywania większej liczby zadań zaawansowanych wymagań programistycznych. Jeśli użyjesz tablicy jako argumentu sizeof, C++ zwraca

liczbę bajtów pierwotnie zarezerwowanych dla tej tablicy. Dane w tablicy nie mają nic wspólnego ze zwróconą wartością sizeof - nawet jeśli jest to tylko tablica znaków zawierająca krótki ciąg znaków.

Przykłady

1. Załóżmy, że chcesz znać rozmiar zmiennoprzecinkowy komputera w bajtach. Możesz to ustalić, wprowadzając słowo kluczowe float w nawiasach - po sizeof- jak pokazano w następującym programie.

```
// Filename: C11SIZE1.CPP

// Wyświetla rozmiar wartości zmiennoprzecinkowych.

#include <iostream.h>

main()

{

cout << “The size of floating-point variables on \n”;

cout << “this computer is “ << sizeof(float) << “\n”;

return 0;

}
```

Ten program może generować różne wyniki na różnych komputerach. Jako argumentu sizeof można użyć dowolnego poprawnego typu danych. Na większości komputerów ten program prawdopodobnie wytwarza takie dane wyjściowe:

```
The size of floating-point variables on

this computer is: 4
```

Operator przecinka

Inny operator C++, czasami nazywany punktem sekwencyjnym, działa nieco inaczej. Jest to operator przecinka (,), który nie działa bezpośrednio na danych, ale tworzy wyrażenia od lewej do prawej. Ten operator pozwala umieścić więcej niż jedno wyrażenie w jednym wierszu, oddzielając je przecinkami. Już widziałeś jedno użycie przecinka punktu sekwencji, gdy nauczyłeś się, jak deklarować i inicjować zmienne. W poniższej sekcji kodu przecinek rozdziela instrukcje. Ponieważ przecinek kojarzy się z lewej strony, pierwsza zmienna, i, jest deklarowana i inicjowana przed drugą zmienną.

```
main()

{

int i=10, j=25;

// Pozostała część programu następuje.
```

Jednak przecinek nie jest punktem sekwencji, gdy jest używany w nawiasach funkcyjnych. Mówi się wtedy o oddzielnych argumentach, ale nie jest to punkt sekwencyjny. Rozważ następujące printf().

```
printf („% d% d% d”, i, i ++, ++ i);
```

Z takiego stwierdzenia można uzyskać wiele wyników. Przecinki służą tylko do oddzielenia argumentów printf() i nie generują sekwencji od lewej do prawej, którą robią inaczej, gdy nie są używane w

funkcjach. Dzięki przedstawionemu tutaj oświadczeniu nie masz żadnego zamówienia! Postfiks i ++ może być prawdopodobnie wykonany przed prefiksem ++ i, nawet jeśli tabela priorytetów tego nie wymaga. Tutaj kolejność oceny zależy od tego, w jaki sposób kompilator wysyła te argumenty do funkcji printf().

WSKAZÓWKA: Nie należy umieszczać operatorów inkrementacji ani operatorów dekrementacji w wywołaniach funkcji, ponieważ nie można przewidzieć kolejności ich wykonywania.

Przykłady

1. Możesz umieścić więcej niż jedno wyrażenie w linii, używając przecinka jako punktu sekwencji. Następujący program to robi.

```
// Filename: C11COM1.CPP

// Ilustruje punkt sekwencji.

#include <iostream.h>

main()

{

int num, sq, cube;

num = 5;

// Calculate the square and cube of the number.

sq = (num * num), cube = (num * num * num);

cout << “The square of “ << num << “ is “ << sq <<

“ and the cube is “ << cube;

return 0;

}
```

Niekoniecznie jest to jednak zalecane, ponieważ nie dodaje niczego do programu i faktycznie zmniejsza jego czytelność. W tym przykładzie kwadrat i sześcian są prawdopodobnie lepiej obliczane na dwóch osobnych liniach.

2. Przecinek umożliwia kilka interesujących instrukcji. Rozważ następującą sekcję kodu.

```
i = 10

j = (i = 12, i + 8);
```

Po zakończeniu wykonywania tego kodu wartość j wynosi 20 - nawet jeśli nie jest to do końca jasne. W pierwszej instrukcji i jest przypisywane 10. W drugiej instrukcji przecinek powoduje, że przypisywana jest i wartość 12, a następnie j ma wartość i + 8 lub 20.

3. W poniższej sekcji kodu, ans ma wartość 12, ponieważ przypisanie przed przecinkiem jest wykonywane w pierwszej kolejności. Pomimo tego asocjatywności operatora przypisania od prawej do lewej, punkt przecinka wymusza przypisanie 12 do x, zanim x zostanie przypisane do ans.

```
ans = (y = 8, x = 12);
```

Po zakończeniu tego fragmentu y zawiera 8, x zawiera 12, a ans zawiera również 12.

Operatory bitowe

Operatory bitowe manipulują wewnętrznymi reprezentacjami danych, a nie tylko „wartościami w zmiennych", jak robią to inni operatorzy. Te operatory bitowe wymagają zrozumienia systemu numerowania binarnego, a także pamięci komputera. Ta sekcja przedstawia operatory bitowe. Operatory bitowe są używane w zaawansowanych technikach programowania i są zwykle używane w znacznie bardziej skomplikowanych programach niż te opisane tu. Niektórzy ludzie programują w C++ od lat i nigdy nie uczą się operatorów bitowych. Niemniej ich zrozumienie może pomóc poprawić wydajność programu i umożliwić działanie na poziomie bardziej zaawansowanym niż pozwala na to wiele innych języków programowania.

Bitowe operatory logiczne

Istnieją cztery bitowe operatory logiczne, które pokazano. Operatory te działają na reprezentacjach binarnych danych liczb całkowitych. Umożliwia to programistom systemowym manipulowanie wewnętrznymi bitami w pamięci i zmiennych. Jednak operatory bitowe są przeznaczone nie tylko dla programistów systemów. Programiści aplikacji mogą również poprawić wydajność swoich programów na kilka sposobów.

Operator: Znaczenie

& : Bitowe AND

| : Bitowe włącznie OR

^ : Wyłącznie bitowe LUB

~ : Uzupełnienie bitowe 1

Każdy z operatorów bitowych porównuje dane wewnętrzne krok po kroku. Operatory bitowe odnoszą się tylko do znaku i liczby całkowitej , zmiennej i stałej, a nie dane zmiennoprzecinkowe. Ponieważ liczby binarne składają się z 1 i 0, te 1 i 0 (zwane bitami) są w porównaniu do siebie, aby uzyskać pożądany wynik dla każdego operatora bitowego. Przed przestudiowaniem przykładów należy zapoznać się z tabelą. Zawiera tabele prawdy, które opisują działanie każdego operatora bitowego na wzorcach wewnętrznych liczb całkowitych lub znaków.

Bitowe AND (&)

0 & 0 = 0

0 & 1 = 0

1 & 0 = 0

1 & 1 = 1

Bitowe włącznie OR (|)

0 | 0 = 0

0 | 1 = 1

1 | 0 = 1

1 | 1 = 1

Wyłącznie bitowe OR (^)

0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

Uzupełnienie bitowe 1 (~)

~ 0 = 1

~ 1 = 0

W bitowych tabelach prawdy możesz zastąpić 1 i 0 odpowiednio Prawdą i Fałszem, jeśli pomoże to lepiej zrozumieć wynik. W przypadku bitowej tabeli prawdy AND (&) oba bity porównywane przez operator & muszą mieć wartość True, aby wynik był prawdziwy. Innymi słowy „True i True dają True".

WSKAZÓWKA: Zamieniając jedynki i zera na True i False, możesz być w stanie powiązać operatory bitowe z normalnymi operatorami logicznymi, && i ||, których używasz do porównań.

The | operator bitowy jest czasem nazywany bitowym operatorem włączania OR. Jeśli jedna strona | operatora ma wartość 1 (prawda) - lub jeśli obie strony mają wartość 1 - wynikiem jest 1 (prawda). Operator ^ jest nazywany bitową wyłącznością OR. Oznacza to, że każda ze stron operatora ^ musi mieć wartość 1 (prawda), aby wynik miał wartość 1 (prawda), ale obie strony nie mogą być jednocześnie 1 (prawda). Operator ~, nazywany bitowym uzupełnieniem 1, odwraca każdy bit do swojej przeciwnej wartości.

UWAGA: Uzupełnienie bitowe 1 nie neguje liczby. Większość komputerów używa uzupełnienia 2 do zanegowania liczb. Uzupełnienie bitowej 1 odwraca wzór bitowy liczb, ale nie dodaje dodatkowego 1, jak wymaga uzupełnienia 2.

Możesz testować i zmieniać poszczególne bity wewnątrz zmiennych, aby sprawdzić wzorce danych. Poniższe przykłady pomagają zilustrować każdego z czterech operatorów bitowych.

Przykłady

1. Jeśli zastosujesz operator bitowy do cyfr 9 i 14, otrzymasz wynik 8. Gdy wartości binarne 9 (1001) i 14 (1110) są porównywane bitowo i na podstawie, otrzymany wzór bitowy wynosi 8 (1000).

W programie C++ możesz zakodować to porównanie bitowe w następujący sposób.

Uczyń wynik równy wartości binarnej 9 (1001) ANDed do wartości binarnej 14 (1110).

wynik = 9 i 14

Zmienna wynikowa ma wartość 8, która jest wynikiem bitowego &. 9 (binarne 1001) lub 14 (binarne 1110) - lub oba - również mogą być przechowywane w zmiennych o tym samym wyniku.

2. Kiedy zastosujesz bitowy | operator na liczby 9 i 14, otrzymasz 15. Gdy wartości binarne 9 (1001) i 14 (1110) są porównywane bitowo | w oparciu o wynikowy wzór bitowy to 15 (1111). bity wyniku to 1 (prawda) w każdej pozycji, w której 1 występuje w obu liczbach. W programie C++ możesz zakodować to porównanie bitowe jak następuje:

wynik = 9 | 14;

Zmienna wynikowa ma wartość 15, która jest wynikiem bitowego | 9 lub 14 (lub oba) również mogą być przechowywane w zmiennych.

3. Bitowe ^ zastosowane do 9 i 14 daje 7. Bitowe ^ ustawia wynikowe bity na 1, jeśli jedna liczba lub bit drugiej to 1, ale nie, jeśli oba pasujące bity są równe 1 w tym samym czasie. W programie C++ możesz zakodować to porównanie bitowe w następujący sposób:

wynik = 9 ^ 14;

Zmienna wynikowa zawiera 7 (binarnie 0111), co jest wynikiem bitowego ^. 9 lub 14 (lub oba) również mogą być przechowywane w zmiennych o tym samym wyniku.

4. Bitowe ~ po prostu neguje każdy bit. Jest to jednoargumentowy operator bitowy, ponieważ można go zastosować tylko do jednej wartości w danym momencie.

W programie C++ możesz zakodować tę bitową operację w następujący sposób:

wynik = ~ 9;

Zmienna wynikowa ma wartość 6, która jest wynikiem bitowego ~. 9 można zapisać w zmiennej o tym samym wyniku. 5. Możesz skorzystać z operatorów bitowych, aby przeprowadzić testy danych, których nie możesz wykonać równie skutecznie na inne sposoby. Załóżmy na przykład, że chcesz wiedzieć, czy użytkownik wpisał liczbę nieparzystą lub parzystą (zakładając, że wprowadzane są liczby całkowite). Za pomocą operatora modułu (%) możesz ustalić, czy reszta - po podzieleniu wartości wejściowej przez 2 - wynosi 0 lub 1. Jeśli reszta to 0, liczba jest parzysta. Jeśli reszta to 1, liczba jest nieparzysta. Operatory bitowe są bardziej wydajne niż inne operatory, ponieważ bezpośrednio porównują wzorce bitowe bez użycia żadnych operacji matematycznych. Ponieważ liczba jest parzysta, nawet jeśli jej wzór bitowy kończy się na 0, a nieparzysty, jeśli jej wzór bitowy kończy się na 1, możesz również testować na liczbach nieparzystych lub parzystych, stosując bitowe & do danych i do binarnego 1. Jest to bardziej wydajne niż przy użyciu modułu operator. Poniższy program informuje użytkowników, czy ich wartość wejściowa jest nieparzysta, czy nawet przy użyciu tej techniki. Zidentyfikuj plik i dołącz plik nagłówkowy wejścia / wyjścia. Ten program sprawdza nieparzyste lub parzyste dane wejściowe. Potrzebujesz miejsca na wpisanie numeru użytkownika, więc zadeklaruj zmienną wejściową jako liczbę całkowitą. Poproś użytkownika o numer do przetestowania. Wpisz odpowiedź użytkownika w danych wejściowych. Użyj operatora bitowego, i, aby przetestować numer. Jeśli bit po prawej stronie wejścia to 1, powiedz użytkownikowi, że liczba jest nieparzysta. Jeśli bit po prawej stronie wejścia to 0, powiedz użytkownikowi, że

liczba jest parzysta.

```
// Filename: C11ODEV.CPP

// Używa bitów i do ustalenia, czy

// liczba jest nieparzysta lub parzysta.

#include <iostream.h>

main()

{

int input; // Will hold user's number

cout << "What number do you want me to test? ";
```

```
cin >> input;

if (input & 1) // True if result is 1;

// otherwise it is false (0)

{ cout << “The number “ << input << “ is odd\n”; }

else

{ cout << “The number “ << input << “ is even\n”; }

return 0;

}
```

6. Jedyną różnicą między wzorami bitowymi dla wielkich i małych liter jest bit 5 (trzeci bit od lewej). W przypadku małych liter bit 5 to 1. W przypadku wielkich liter bit 5 to 0. Tylko bit 6 jest inny. Aby przekonwertować znak na wielkie litery, musisz wyłączyć (zmienić na 0) bit nr 5. Możesz zastosować bitowe & do znaku wejściowego i 223 (który jest binarny 11011111) poza bitem 5 i przekonwertować dowolny znak wejściowy na jego odpowiednik wielkich liter. Jeśli liczba jest już pisana wielkimi literami, bitowa i nie zmienia jej. 223 (binarny 11011111) jest nazywany maską bitową, ponieważ tak jest ,maskuje (podobnie jak taśma maskująca maskuje obszary, które nie będą malowane) bit 5, więc staje się 0, jeśli jeszcze nie jest. Poniższy program robi to, aby zapewnić, że użytkownicy wpisywali wielkie litery, gdy pytano ich o inicjały.

```
// Filename: C11UPCS1.CPP

// Konwertuje znaki wejściowe na wielkie litery

// jeśli jeszcze nie są.

#include <iostream.h>

main()

{

char first, middle, last; // Will hold user’s initials

int bitmask=223; // 11011111 in binary

cout << “What is your first initial? “;

cin >> first;

cout << “What is your middle initial? “;

cin >> middle;

cout << “What is your last initial? “;

cin >> last;

// Ensure that initials are in uppercase.

first = first & bitmask; // Turn off bit 5 if

middle = middle & bitmask; // it is not already
```

```
last = last & bitmask; // turned off.

cout << “Your initials are “ << first << “ “ <<

middle << “ “ << last;

return 0;

}
```

Poniższe dane wyjściowe pokazują, co się dzieje, gdy dwa z inicjałów są pisane małymi literami. Program konwertuje je na wielkie litery przed ich ponownym wydrukowaniem. Chociaż istnieją inne sposoby konwersji na małe litery, żaden z nich nie jest tak wydajny, jak użycie operatora bitowego.

Jaki jest twój pierwszy inicjał? g

Jaki jest twój środkowy inicjał? M.

Jaki jest twój ostatni inicjał? p

Twoje inicjały to: G M P

Ćwiczenia

1. Napisz program, który wyświetla cyfry od 1 do 10. Użyj dziesięciu różnych znaków i tylko jednej zmiennej o nazwie wynik, aby zachować wartość przed każdym znakiem. Użyj operatora inkrementacji, aby dodać 1, aby uzyskać wynik przed każdym wykresem.

2. Napisz program, który pyta użytkowników o wiek. Używając pojedynczego printf (), który zawiera operator warunkowy, wydrukuj na ekranie następujące informacje, jeśli wiek wejściowy przekracza 21 lat,

Nie jesteś nieletni.

lub wydrukuj to inaczej:

Nadal jesteś nieletni.

Ta funkcja printf() może być długa, ale pomaga zilustrować sposób działania operatora warunkowego w instrukcjach, w których logika if-else nie.

3. Użyj operatora warunkowego - i żadnych instrukcji if-else -, aby napisać następującą procedurę obliczania podatków: rodzina nie płaci podatku, jeśli jej roczne wynagrodzenie jest niższe niż 5000 USD. Opłaca 10-procentowy podatek, jeśli zakres wynagrodzenia zaczyna się od 5000 USD, a kończy na 9 999 USD. Opłaca 20-procentowy podatek, jeśli przedział wynagrodzenia zaczyna się od 10 000 USD, a kończy od 19 999 USD. W przeciwnym razie rodzina płaci 30-procentowy podatek.

4. Napisz program, który konwertuje wielką literę na małą literę, stosując maskę bitową i jeden z bitowych operatorów logicznych. Jeśli postać ma już małe litery, nie zmieniaj jej.

Podsumowanie

Teraz nauczyłeś się prawie każdego operatora w języku C++. Jak wyjaśniono, warunkowe, inkrementacja i dekrementacja to trzy operatory, które umożliwiają C++ odróżnienie się od wielu innych języków programowania. Zawsze musisz być świadomy tabeli pierwszeństwa za każdym razem, gdy z nich korzystasz, podobnie jak w przypadku wszystkich operatorów. Operatory sizeof i point point działają inaczej niż większość innych. Sizeof jest operatorem kompilacji i działa w sposób podobny do

dyrektywy #define preprocessor, ponieważ obie są zastępowane ich wartościami w czasie kompilacji. Punkt sekwencji pozwala mieć wiele instrukcji w tym samym wierszu lub w jednym wyrażeniu. Zarezerwuj punkt sekwencji dla deklarowania zmiennych tylko dlatego, że może być niejasny, gdy jest połączony z innymi wyrażeniami. Ta część kończy dyskusję na temat operatorów C++. Teraz, gdy możesz obliczyć niemal każdy wynik, którego kiedykolwiek będziesz potrzebować, nadszedł czas, aby dowiedzieć się, jak uzyskać większą kontrolę nad swoimi programami.

Pętla while

Powtarzalne możliwości komputerów czynią je dobrymi narzędziami do przetwarzania dużych ilości informacji. Konstrukcje C++ zawierają potężne, ale zwięzłe i wydajne, zapętlające się polecenia podobne do innych języków, które już znasz. Pętle while umożliwiają programom powtarzanie serii instrukcji w kółko, o ile określony warunek jest zawsze spełniony. Komputery nie nudzą się podczas wielokrotnego wykonywania tych samych zadań. Jest to jeden z powodów, dla których są one tak ważne w przetwarzaniu danych biznesowych. Tu nauczysz się, co następuje:

♦ Pętla while

♦ Pojęcie pętli

♦ Pętla do-while

♦ Różnice między pętlami if i while

♦ Funkcja exit ()

♦ Instrukcja break

♦ Liczniki i sumy rozdziału powinieneś zrozumieć pierwszą z kilku metod C++ zapewniających powtarzanie sekcji programu. Omówienie pętli w tym rozdziale obejmuje jedną z najważniejszych jakiej używa do zapętlania: tworzenie zmiennych licznika i zmiennych całkowitych.

Instrukcja while

Instrukcja while jest jedną z kilku instrukcji konstrukcyjnych C++. Każda konstrukcja (od konstrukcji) jest instrukcją języka programowania - lub serią instrukcji - sterującą zapętlaniem. Chwila, podobnie jak inne takie instrukcje, jest zapętloną instrukcją, która kontroluje wykonanie szeregu innych instrukcji. Instrukcje zapętlające powodują powtarzalne wykonywanie części programu, o ile spełniony jest określony warunek. Format instrukcji while to

while (wyrażenie testowe)

{blok jednej lub więcej instrukcji C++; }

Wymagane są nawiasy wokół wyrażenia testowego. Dopóki wyrażenie testowe ma wartość True (niezerowa), blok jednej lub więcej instrukcji C++ jest wykonywany wielokrotnie, aż wyrażenie testowe stanie się False (wartość zero). Nawiasy klamrowe są wymagane przed i po treści pętli while, chyba że chcesz wykonać tylko jedną instrukcję. Każda instrukcja w treści pętli while wymaga kończącego się średnika. Wyrażenie testu zastępczego zwykle zawiera relacyjne i być może logiczne operatory. Operatory te zapewniają warunek True-False sprawdzony w wyrażeniu testowym. Jeśli wyrażenie testowe ma wartość False, gdy program osiągnie pętlę while po raz pierwszy, treść pętli while nie zostanie w ogóle wykonana. Niezależnie od tego, czy ciało pętli while wykonuje się nie raz, raz czy wiele razy, instrukcje następujące po nawiasie zamykającym pętli while są wykonywane, jeśli wyrażenie testowe staje się False. Ponieważ wyrażenie testowe określa, kiedy pętla się kończy, treść pętli while musi zmieniać zmienne używane w wyrażeniu testowym. W przeciwnym razie wyrażenie testowe nigdy się nie zmienia, a pętla while jest powtarzana na zawsze. Jest to znane jako nieskończona pętla i należy tego unikać.

WSKAZÓWKA: Jeśli treść pętli while zawiera tylko jedną instrukcję, otaczające ją nawiasy klamrowe nie są wymagane. Dobrym nawykiem jest jednak umieszczanie instrukcji nawiasów pętli while w

nawiasach klamrowych, ponieważ jeśli trzeba później dodać instrukcje do treści pętli while, nawiasy klamrowe już tam są.

Pojęcie pętli

Korzystasz z koncepcji pętli w życiu codziennym. Za każdym razem, gdy trzeba powtórzyć tę samą procedurę, wykonuje się pętlę - podobnie jak komputer z instrukcją while. Załóżmy, że pakujesz prezenty świąteczne. Poniższe instrukcje reprezentują kroki zapętlenia (w formacie while), które wykonujesz podczas pakowania prezentów.

podczas gdy (wciąż są rozpakowane prezenty)

{Zdobądź następny prezent;

Wytnij papier do pakowania;

Zawiń prezent;

Połóż łuk na prezent;

Wypełnij kartę imienną prezentu;

Połóż zapakowany prezent z innymi; }

Niezależnie od tego, czy masz 3, 15, czy 100 prezentów do pakowania, stosuj tę procedurę (pętlę) wielokrotnie, aż każdy prezent zostanie zapakowany. Na przykład, który można łatwiej skomputeryzować, załóżmy, że chcesz zsumować wszystkie czeki napisane w poprzednim miesiącu. Możesz wykonać następującą pętlę.

podczas gdy (są jeszcze czeki z ostatniego miesiąca do zsumowania)

{Dodaj kwotę następnego czeku do sumy; }

Treść tego pseudokodu w pętli while ma tylko jedną instrukcję, ale ta pojedyncza instrukcja musi być wykonywana do momentu dodania każdej z kontroli z poprzedniego miesiąca. Kiedy ta pętla się skończy (gdy nie będzie już sumowania czeków z poprzedniego miesiąca), musisz mieć wynik. Treść pętli while może zawierać jedną lub więcej instrukcji C++, w tym dodatkowe pętle while. Twoje programy będą bardziej czytelne, jeśli przesuniesz ciało część pętli while do kilku spacji w prawo. Ilustrują to następujące przykłady.

Przykłady

1. Niektóre programy przedstawione wcześniej wymagają wkładu użytkownika za pomocą cin. Jeśli użytkownicy nie wprowadzą odpowiednich wartości, programy te wyświetlają komunikat o błędzie i proszą użytkownika o wprowadzenie innej wartości, co jest dopuszczalną procedurą. Teraz, gdy rozumiesz konstrukcję pętli while, powinieneś umieścić komunikat o błędzie w pętli. W ten sposób użytkownicy widzą komunikat w sposób ciągły, dopóki nie wpiszą odpowiednich wartości wejściowych, a nie jeden raz. Poniższy program jest krótki, ale pokazuje pętlę while, która zapewnia prawidłowe wprowadzanie danych z klawiatury. Pyta użytkowników, czy chcą kontynuować. Możesz to uczynić program większy, który wymaga zgody użytkownika, aby kontynuować. Umieść znak zachęty, u dołu ekranu tekstowego. Tekst pozostaje na ekranie, dopóki użytkownik nie powie programowi, aby kontynuował wykonywanie. Zidentyfikuj plik i dołącz niezbędny plik nagłówka. W tym programie chcesz mieć pewność, że użytkownik wpisze Y lub N. Musisz zapisać odpowiedź użytkownika, więc zadeklaruj zmienną ans jako znak. Zapytaj użytkowników, czy chcą kontynuować, i uzyskaj odpowiedź. Jeśli użytkownik nie wpisze Y ani N, poproś go o kolejną odpowiedź

```
// Filename: C12WHIL1.CPP
// Procedura wprowadzania, aby upewnić się, że użytkownik wpisuje a
// poprawna odpowiedź. Ta procedura może być częścią
// większego programu.
#include <iostream.h>
main()
{
char ans;
cout << "Do you want to continue (Y/N)? ";
cin >> ans; // Get user's answer
while ((ans != 'Y') && (ans != 'N'))
{ cout << "\nYou must type a Y or an N\n"; // Warn
// and ask
cout << "Do you want to continue (Y/N)?"; // again.
cin >> ans;
} // Body of while loop ends here.
return 0;
}
```

Zauważ, że dwie funkcje cin robią to samo. Musisz użyć początkowego cin, poza pętlą while, aby podać odpowiedź do sprawdzenia w pętli while. Jeśli użytkownik wpisze coś innego niż Y lub N, program wydrukuje komunikat o błędzie, poprosi o kolejną odpowiedź, a następnie sprawdzi nową odpowiedź. Ta metoda walidacji jest lepsza niż ta, w której czytelnik ma tylko jedną dodatkową szansę na odniesienie sukcesu. Pętla while sprawdza wyrażenie testowe na górze pętli. Dlatego pętla może nigdy nie zostać wykonana. Jeśli test jest początkowo False, pętla nie zostanie wykonana ani razu. Dane wyjściowe z tego programu są pokazane w następujący sposób. Program powtarza się w nieskończoność, aż test relacyjny ma wartość True (gdy tylko użytkownik wpisze Y lub N).

```
Czy chcesz kontynuować (T / N)? k
Musisz wpisać Y lub N
Czy chcesz kontynuować (T / N)? do
Musisz wpisać Y lub N
Czy chcesz kontynuować (T / N)? s
Musisz wpisać Y lub N
Czy chcesz kontynuować (T / N)? 5
Musisz wpisać Y lub N
```

Czy chcesz kontynuować (T / N)? Y

2. Poniższy program jest przykładem nieprawidłowej pętli while. Sprawdź, czy możesz znaleźć problem.

```
// Filename: C12WHBAD.CPP

// Niewłaściwe użycie pętli while.

#include <iostream.h>

main()

{

int a=10, b=20;

while (a > 5)

{ cout << “a is “ << a << “, and b is “ << b << “\n”;

b = 20 + a; }

return 0;

}
```

Ta pętla while jest przykładem nieskończonej pętli. Ważne jest, aby przynajmniej jedna instrukcja wewnątrz while zmieniała zmienną w wyrażeniu testowym (w tym przykładzie zmienna a); w przeciwnym razie warunek jest zawsze prawdziwy. Ponieważ zmienna a nie zmienia się w pętli while, ten program nigdy się nie zakończy.

WSKAZÓWKA: Jeśli przypadkowo napiszesz nieskończoną pętlę, musisz sam zatrzymać program. Jeśli używasz komputera, zwykle oznacza to naciśnięcie klawisza Ctrl-Break. Jeśli korzystasz z systemu UNIX, administrator systemu może być zmuszony zatrzymać wykonywanie programu.

3. Poniższy program prosi użytkowników o imię, a następnie używa pętli while, aby policzyć liczbę znaków w nazwie. Jest to program długości łańcucha; liczy znaki, aż osiągnie zero. Pamiętaj, że długość ciągu równa jest liczbie znaków w ciągu, nie licząc zera.

```
// Filename: C12WHIL2.CPP

// Liczy liczbę liter w imieniu użytkownika.

#include <iostream.h>

main()

{

char name[15]; // Will hold user’s first name

int count=0; // Will hold total characters in name

// Get the user’s first name

cout << “What is your first name? “;

cin >> name;

while (name[count] > 0) // Loop until null zero reached.
```

```
{ count++; } // Add 1 to the count.

cout << “Your name has “ << count << “ characters”;

return 0;

}
```

Pętla trwa tak długo, jak długo wartość następnego znaku w tablicy nazw jest większa od zera. Ponieważ ostatni znak w tablicy ma wartość zerową zero, testem jest Fałsz na ostatnim znaku nazwy, a instrukcja po treści pętli jest kontynuowana.

UWAGA: Wbudowana funkcja łańcucha o nazwie strlen() określa długość łańcuchów.

4. Pętla while poprzedniego programu długości łańcucha nie jest tak wydajna, jak mogłaby być. Ponieważ pętla while kończy się niepowodzeniem, gdy jej wyrażenie testowe wynosi zero, nie ma potrzeby wykonywania testu większego niż. Zmieniając wyrażenie testowe, jak pokazuje poniższy program, można poprawić efektywność liczenia długości łańcucha.

```
// Filename: C12WHIL3.CPP

// Liczy liczbę liter w imieniu użytkownika.

#include <iostream.h>

main()

{

char name[15]; // Will hold user’s first name

int count=0; // Will hold total characters in name

// Get the user’s first name

cout << “What is your first name? “;

cin >> name;

while (name[count]) // Loop until null zero is reached.

{ count++; } // Add 1 to the count.

cout << “Your name has “ << count << “ characters”;

return 0;

}
```

Pętla „do-while”

Instrukcja do-while kontroluje pętlę do-while, która jest podobna do pętli while, z tym wyjątkiem, że test relacyjny występuje na końcu (a nie na początku) pętli. Zapewnia to, że ciało pętli wykonuje się co najmniej raz. Testy „do czasu” dla pozytywnego testu relacyjnego; tak długo, jak test jest prawdziwy, ciało pętli jest kontynuowane. Format do-while jest

```
do

{blok jednej lub więcej instrukcji C++; }
```

while (wyrażenie testowe)

wyrażenie testowe musi być ujęte w nawiasy, tak jak w instrukcji while.

Przykłady

1. Poniższy program jest podobny do pierwszego, który widziałeś za pomocą pętli while (C12WHIL1.CPP), z wyjątkiem tego, że używana jest funkcja do-while. Zwróć uwagę na umiejscowienie wyrażenia testowego. Ponieważ to wyrażenie kończy pętlę, dane wejściowe użytkownika nie muszą pojawiać się przed pętlą i ponownie w jej treści.

```
// Filename: C12WHIL4.CPP

// Procedura wprowadzania, aby upewnić się, że użytkownik wpisuje a

// poprawna odpowiedź. Ta procedura może być częścią

// większego programu.

#include <iostream.h>

main()

{

char ans;

do

{ cout << "\nYou must type a Y or an N\n"; // Warn

// and ask

cout << "Do you want to continue (Y/N) ?"; // again.

cin >> ans; } // Body of while loop

// ends here.

while ((ans != 'Y') && (ans != 'N'));

return 0;

}
```

2. Załóżmy, że wprowadzasz kwoty sprzedaży do komputera, aby obliczyć sumy rozszerzone. Chcesz, aby komputer wydrukował ilość sprzedaną, numer części i sumę rozszerzoną (ilość razy cena za jednostkę), tak jak robi to następujący program.

```
// Filename: C12INV1.CPP

// Pobiera informacje o zapasach od użytkownika i drukuje

// szczegółowy wykaz zapasów z rozszerzonymi podsumowaniami.

#include <iostream.h>

#include <iomanip.h>

main()
```

```
{
int part_no, quantity;
float cost, ext_cost;
cout << "*** Inventory Computation ***\n\n"; // Title
// Get inventory information.
do
{ cout << "What is the next part number (-999 to end)? ";
cin >> part_no;
if (part_no != -999)
{ cout << "How many were bought? ";
cin >> quantity;
cout << "What is the unit price of this item? ";
cin >> cost;
ext_cost = cost * quantity;
cout << "\n" << quantity << " of # " << part_no <<
" will cost " << setprecision(2) <<
ext_cost;
cout << "\n\n\n"; // Print two blank lines.
}
} while (part_no != -999); // Loop only if part
// number is not -999.
cout << "End of inventory computation\n";
return 0;
}
```

Oto wynik tego programu:

```
*** Inventory Computation ***
What is the next part number (-999 to end)? 213
How many were bought? 12
What is the unit price of this item? 5.66
12 of # 213 will cost 67.92
What is the next part number (-999 to end)? 92
```

How many were bought? 53

What is the unit price of this item? .23

53 of # 92 will cost 12.19

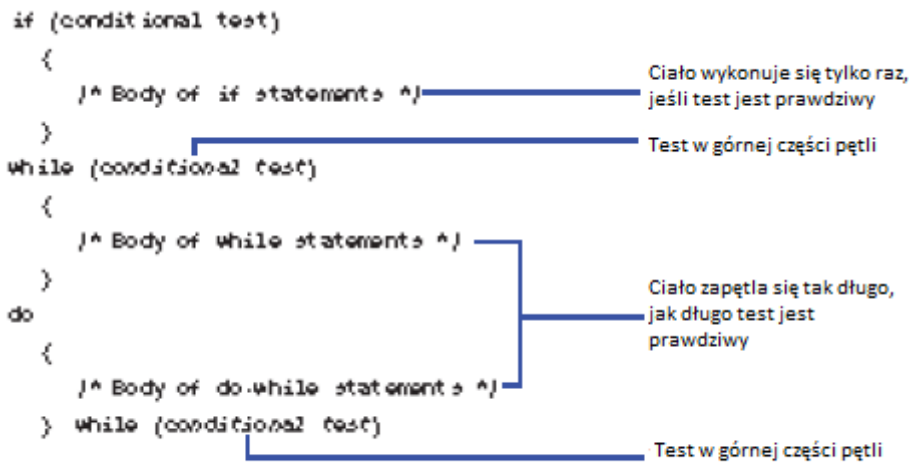
What is the next part number (-999 to end)? -999

End of inventory computation

Pętla „do-while” kontroluje wprowadzanie informacji o sprzedaży klienta. Zwróć uwagę na „wyzwalacz”, który kończy pętlę. Jeśli użytkownik wprowadzi –999 jako numer części, pętla „do-while” zostanie zamknięta, ponieważ w ekwipunku nie ma części o numerze –999. Program ten można jednak ulepszyć na kilka sposobów. Fakturę można wydrukować na drukarce, a nie na ekranie. Można również obliczyć sumę zapasów (całkowitą kwotę całego zamówienia). Dowiesz się, jak sumować takie dane w sekcji „Liczniki i sumy” w dalszej części tego rozdziału.

Pętla if versus Pętla while

Niektórzy początkujący programiści mylą instrukcję if z konstrukcjami pętli. Pętle while i do-while powtarzają sekcję kodu wiele razy, w zależności od testowanego warunku. Instrukcja if może, ale nie musi, wykonać sekcję kodu; jeśli tak, to wykonuje tę sekcję tylko raz. Użyj instrukcji if, jeśli chcesz warunkowo wykonać sekcję kodu raz, i użyj pętli while lub do-while, jeśli chcesz wykonać sekcję więcej niż raz. Rysunek pokazuje różnice między instrukcją if a dwiema pętlami while.



Funkcja exit () Instrukcja i instrukcja break

C++ udostępnia funkcję exit() jako sposób na wcześniejsze opuszczenie programu (przed jego naturalnym zakończeniem). Format exit() to exit(status); gdzie status jest opcjonalną zmienną całkowitą lub literałem. Jeśli znasz kody zwrotne swojego systemu operacyjnego, status umożliwia testowanie wyników programów w C++. W DOS status jest wysyłany do zmiennej środowiskowej poziomu błędu systemu operacyjnego, gdzie można go przetestować za pomocą plików wsadowych. Wiele razy coś się dzieje w programie, który wymaga jego zakończenia. Może to być poważny problem, na przykład błąd napędu dysku. Być może użytkownicy wskazują, że chcą wyjść z programu - możesz to powiedzieć, podając użytkownikom specjalną wartość do pisania za pomocą cin lub scanf(). Możesz

izolować funkcję exit() w wierszu samodzielnie lub w dowolnym innym miejscu, w którym może pojawić się instrukcja lub funkcja C++. Zazwyczaj exit() jest umieszczane w treści instrukcji if, aby wcześniej zakończyć program, w zależności od wyniku testu relacyjnego. Zawsze używaj pliku nagłówkowego stdlib.h, gdy używasz exit(). Ten plik opisuje działanie exit() dla twojego programu. Ilekroć używasz funkcji w programie, powinieneś znać odpowiedni plik nagłówka #include, który zwykle jest wymieniony w podręczniku kompilatora. Zamiast wychodzić z całego programu można jednak użyć instrukcji break, aby wyjść z bieżącej pętli. Format przerwy to przerwa; Instrukcja break może przejść w dowolne miejsce w programie C++, gdzie może być dowolna inna instrukcja, ale zazwyczaj pojawia się w treści pętli while lub do-while, używanej do wcześniejszego opuszczenia pętli. Poniższe przykłady ilustrują funkcję exit () i instrukcję break.

UWAGA: Instrukcja break wychodzi tylko z najbardziej bieżącej pętli. Jeśli masz pętlę while w innej pętli while, break wychodzi tylko z pętli wewnętrznej.

Przykłady

1. Oto prosty program, który pokazuje, jak działa funkcja exit(). Ten program wygląda tak, jakby drukuje kilka komunikatów na ekranie, ale tak nie jest. Ponieważ exit() pojawia się na początku kodu, program kończy pracę natychmiast po nawiasie otwierającym main().

```
// C12EXIT1.CPP

// Wychodzi wcześniej z powodu funkcji exit()

#include <iostream.h>

#include <stdlib.h> // Required for exit().

main()

{

exit(0); // Forces program to end here.

cout << “C++ programming is fun.\n”;

cout << “I like learning C++ by example!\n”;

cout << “C++ is a powerful language that is “ <<

“not difficult to learn.”;

return 0;

}
```

2. Instrukcja break nie jest tak silnym wyjściem programu, jak funkcja exit(). Podczas gdy exit() kończy cały program, break kończy tylko aktywną pętlę. Innymi słowy, przerwa jest zwykle umieszczana w pętli while lub do-while, aby „symulować” ukończoną pętlę. Instrukcja występująca po pętli jest wykonywana po wystąpieniu przerwy, ale program nie kończy pracy tak jak w przypadku exit(). Następujący program do drukowania C++ jest fajny! aż użytkownik wprowadzi literę N, aby ją zatrzymać. Komunikat jest drukowany tylko raz, ponieważ instrukcja break wymusza wcześniejsze wyjście z pętli.

```
// Filename: C12BRK.CPP
```

```
// Demonstruje instrukcję break.

#include <iostream.h>

main()

{

char user_ans;

do

{ cout << “C++ is fun! \n”;

break; // Causes early exit.

cout << “Do you want to see the message again (N/Y)? “;

cin >> user_ans;

} while (user_ans == ‘Y’);

cout << “That’s all for now\n”;

return 0;

}
```

Ten program zawsze wytwarza następujące dane wyjściowe:

C++ jest fajny!

To wszystko na teraz

Z danych wyjściowych tego programu można stwierdzić, że instrukcja break nie pozwala pętli do-while dojść do jej naturalnego zakończenia, ale powoduje jej wcześniejsze zakończenie. Końcowy cout jest drukowany, ponieważ tylko bieżąca pętla - a nie cały program - wychodzi z instrukcją break.

3. W przeciwieństwie do poprzedniego programu, przerwa zwykle pojawia się po instrukcji if. To sprawia, że jest to przerwa warunkowa, która występuje tylko wtedy, gdy relacyjny test instrukcji if jest prawdziwy. Dobrym tego przykładem jest program inwentaryzacyjny, który widziałeś wcześniej (C12INV1.CPP). Mimo że użytkownicy wpisują –999, kiedy chcą wyjść z programu, dodatkowy, jeśli test jest wymagany w czasie wykonywania. –999 kończy pętlę „do-while”, ale treść „do-while” nadal wymaga testu if, więc pozostałe monity o ilość i koszt nie są podawane. Jeśli wstawisz przerwę po przetestowaniu danych wprowadzonych przez użytkownika, jak pokazano w poniższym programie, do-while nie będzie potrzebował testu if. Przerwa kończy działanie, gdy tylko użytkownik zasygnalizuje koniec inwentarza, wprowadzając –999 jako numer części

```
// Filename: C12INV2.CPP

// Pobiera informacje o zapasach od użytkownika i drukuje

// szczegółowy wykaz zapasów z rozszerzonymi podsumowaniami.

#include <iostream.h>

#include <iomanip.h>

main()
```

```
{
int part_no, quantity;
float cost, ext_cost;
cout << "*** Inventory Computation ***\n\n"; // Title
// Get inventory information
do
{ cout << "What is the next part number (-999 to end)? ";
cin >> part_no;
if (part_no == -999)
{ break; } // Exit the loop if
// no more part numbers.
cout << "How many were bought? ";
cin >> quantity;
cout << "What is the unit price of this item? ";
cin >> cost;
cout << "\n" << quantity << " of # " << part_no <<
" will cost " << setprecision(2) << cost*quantity;
cout << "\n\n\n"; // Print two blank lines.
} while (part_no != -999); // Loop only if part
// number is not -999.
cout << "End of inventory computation\n";
return 0;
}
```

4. Możesz użyć następującego programu do sterowania dwoma innymi programami. Ten program ilustruje, w jaki sposób C++ może przekazywać informacje do DOS za pomocą exit(). To jest twój pierwszy przykład programu menu. Podobnie jak menu restauracji, program menu C++ zawiera listę możliwych wyborów użytkownika. Użytkownicy decydują, co mają zrobić komputer, z dostępnych opcji menu. Aplikacja listy mailingowej w Dodatku F, „Aplikacja listy mailingowej", używa menu dla swoich opcji użytkownika. Ten program zwraca 1 lub 2 do swojego systemu operacyjnego, w zależności od wyboru użytkownika. To zależy od działania system w celu przetestowania wartości wyjściowej i uruchomienia odpowiedniego programu.

```
// Filename: C12EXIT2.CPP
// Pyta użytkownika o jego wybór i zwraca
// ten wybór do systemu operacyjnego za pomocą exit().
```

```
#include <iostream.h>

#include <stdlib.h>

main()

{

int ans;

do

{ cout << “Do you want to:\n\n”;

cout << “\t1. Run the word processor \n\n”;

cout << “\t2. Run the database program \n\n”;

cout << “What is your selection? “;

cin >> ans;

} while ((ans != 1) && (ans != 2)); // Ensures user

// enters 1 or 2.

exit(ans); // Return value to operating system.

return 0; // Return does not ever execute due to exit().

}
```

Liczniki i sumy

Liczenie jest ważne dla wielu aplikacji. Być może będziesz musiał wiedzieć, ilu masz klientów lub ile osób przekroczyło określoną średnią w Twojej klasie. Możesz policzyć, ile czeków napisałeś w poprzednim miesiącu za pomocą komputerowego systemu książeczek czekowych. Zanim opracujesz procedury C++ do liczenia wystąpień, pomyśl o tym, jak liczysz we własnym umyśle. Jeśli dodajesz całkowitą liczbę czegoś, na przykład znaczki w swojej kolekcji znaczków lub liczbę wysłanych zaproszeń ślubnych, prawdopodobnie zrobiłbyś następujące rzeczy:

Zacznij od 0 i dodaj 1 dla każdego liczonego elementu. Po zakończeniu powinieneś mieć całkowitą liczbę (lub całkowitą liczbę). To wszystko, co robisz, gdy liczysz w C++: Przypisz 0 do zmiennej i dodaj 1 do niej za każdym razem, gdy przetwarzasz inną wartość danych. Operator inkrementacji (++) jest szczególnie przydatny do zliczania.

Przykłady

1. Aby zilustrować użycie licznika, następujący program wypisuje „Komputery są fajne!” na ekranie 10 razy. Możesz napisać program, który ma 10 instrukcji cout, ale to nie byłoby wydajne. Byłoby również zbyt uciążliwe, aby mieć 5000 instrukcji cout, jeśli chcesz wydrukować tę samą wiadomość 5000 razy. Dodając pętlę while i licznik, który zatrzymuje się po osiągnięciu określonej sumy, możesz kontrolować to drukowanie, jak pokazuje następujący program.

```
// Filename: C12CNT1.CPP

// Program do drukowania wiadomości 10 razy.
```

```
#include <iostream.h>

main()

{

int ctr = 0; // Holds the number of times printed.

do

{ cout << “Computers are fun!\n”;

ctr++; // Add one to the count,

// after each cout.

} while (ctr < 10); // Print again if fewer

// than 10 times.

return 0;

}
```

Dane wyjściowe z tego programu są pokazane w następujący sposób. Zauważ, że wiadomość jest drukowana dokładnie 10 razy.

```
Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!

Computers are fun!
```

Sercem procesu liczenia w tym programie jest następująca instrukcja.

```
ctr ++;
```

Dowiedziałeś się wcześniej, że operator inkrementacji dodaje 1 do zmiennej. W tym programie zmienna licznika jest zwiększana za każdym razem, gdy pętle do-while. Ponieważ jedyną operacją wykonywaną w tym wierszu jest przyrost ctr, przyrost przedrostka (++ ctr) daje te same wyniki.

2. Poprzedni program nie tylko dodawał do zmiennej licznika, ale także wykonywał pętlę określoną liczbę razy. Jest to powszechna metoda warunkowego wykonywania części programu przez określoną liczbę razy. Poniższy program to program haseł. Hasło jest przechowywane w zmiennej całkowitej. Użytkownik musi poprawnie wprowadzić pasujące hasło w trzech próbach. Jeśli użytkownik nie wpisze prawidłowego hasła w tym czasie, program się kończy. Jest to powszechna metoda wykorzystywana

przez komputery dial-up. Umożliwiają dzwoniącemu wypróbowanie hasła określoną liczbę razy, a następnie odłożenie słuchawki, jeśli limit ten zostanie przekroczony. Pomaga to powstrzymać ludzi przed wypróbowaniem setek różnych haseł na jednym posiedzeniu. Jeśli użytkownicy odgadną prawidłowe hasło po trzech próbach, zobaczą tajną wiadomość.

```
// Filename: C12PASS1.CPP

// Program pyta o hasło i

// sprawdź to z wewnętrznym.

#include <iostream.h>

#include <stdlib.h>

main()

{

int stored_pass = 11862;

int num_tries = 0; // Counter for password attempts.

int user_pass;

while (num_tries < 3) // Loop only three

// times.

{ cout << “What is the password (You get 3 tries...)? “;

cin >> user_pass;

num_tries++; // Add 1 to counter.

if (user_pass == stored_pass)

{ cout << “You entered the correct password.\n”;

cout << “The cash safe is behind the picture “ <<

“of the ship.\n”;

exit(0);

}

else

{ cout << “You entered the wrong password.\n”;

if (num_tries == 3)

{ cout << “Sorry, you get no more chances”; }

else

{ cout << “You get “ << (3-num_tries) <<

“ more tries...\n”;}
```

```
}

} // End of while loop.

exit(0);

return 0;

}
```

Ten program daje użytkownikom trzy szanse na wypadek pomyłki. Po trzech nieudanych próbach program kończy pracę bez wyświetlania tajnego komunikatu.

3. Poniższy program to gra polegająca na zgadywaniu liter. Zawiera komunikat informujący użytkowników, ile prób podjęli przed odgadnięciem właściwej litery. Licznik zlicza liczbę tych prób.

```
// Filename: C12GUES.CPP

/ Gra w zgadywanie liter.

#include <iostream.h>

main()

{

int tries = 0;

char comp_ans, user_guess;

// Save the computer's letter

comp_ans = 'T'; // Change to a different

// letter if desired.

cout << "I am thinking of a letter...";

do

{ cout << "What is your guess? ";

cin >> user_guess;

tries++; // Add 1 to the guess-counting variable.

if (user_guess > comp_ans)

{ cout << "Your guess was too high\n";

cout << "\nTry again...";

}

if (user_guess < comp_ans)

{ cout << "Your guess was too low\n";

cout << "\nTry again...";

}
```

```
} while (user_guess != comp_ans); // Quit when a

// match is found.

// They got it right, let them know.

cout << "*** Congratulations! You got it right! \n";

cout << "It took you only " << tries <<

" tries to guess.";

return 0;

}
```

Oto wynik tego programu:

```
I am thinking of a letter...What is your guess? E

Your guess was too low

Try again...What is your guess? X

Your guess was too high

Try again...What is your guess? H

Your guess was too low

Try again...What is your guess? O

Your guess was too low

Try again...What is your guess? U

Your guess was too high

Try again...What is your guess? Y

Your guess was too high

Try again...What is your guess? T

*** Congratulations! You got it right!

It took you only 7 tries to guess
```

Produkcja ogółem

Pisanie procedury dodawania wartości jest tak proste, jak liczenie. Zamiast dodawać 1 do zmiennej licznika, dodajesz wartość do zmiennej całkowitej. Na przykład, jeśli chcesz znaleźć całkowitą kwotę czeków w dolarach, które napisałeś w grudniu, możesz zacząć od zera (0) i dodać kwotę każdego czeku wystawionego w grudniu. Zamiast budować liczbę, budujesz sumę. Jeśli chcesz, aby C++ dodawał wartości, po prostu zainicjuj zmienną sumaryczną na zero, a następnie dodaj każdą wartość do sumy, aż do momentu włączenia wszystkich wartości.

Przykłady

1. Załóżmy, że chcesz napisać program, który doda twoje oceny do zajęć, które bierzesz. Nauczyciel poinformował cię, że uzyskasz A, jeśli zgromadzisz ponad 450 punktów. Następujący program pyta o wartości, dopóki nie wpiszesz -1. -1 oznacza, że skończyłeś wprowadzać oceny, a teraz chcesz zobaczyć sumę. Ten program drukuje również wiadomość z gratulacjami, jeśli masz wystarczającą liczbę punktów na A.

```
// Filename: C12GRAD1.CPP

// Dodaje oceny i określa, czy zdobyłeś A.

#include <iostream.h>

include <iomanip.h>

main()

{

float total_grade=0.0;

float grade; // Holds individual grades.

do

{ cout << “What is your grade? (-1 to end) “;

cin >> grade;

if (grade >= 0.0)

{ total_grade += grade; } // Add to total.

} while (grade >= 0.0); // Quit when -1 entered.

// Control begins here if no more grades.

cout << “\n\nYou made a total of “ << setprecision(1) <<

total_grade << “ points\n”;

if (total_grade >= 450.00)

{ cout << “** You made an A!!”; }

return 0;

}
```

Zauważ, że odpowiedź -1 nie jest dodawana do liczby całkowitej punktów. Ten program sprawdza -1 przed dodaniem do total_grade. Oto wynik tego programu:

```
What is your grade? (-1 to end) 87.6

What is your grade? (-1 to end) 92.4

What is your grade? (-1 to end) 78.7

What is your grade? (-1 to end) -1

You made a total of 258.7 points
```

2. Poniższy program stanowi rozszerzenie programu do obliczania ocen. Nie tylko sumuje punkty, ale także oblicza ich średnią. Aby obliczyć średnią ocen, program musi najpierw określić, ile ocen zostało wprowadzonych. Jest to subtelny problem, ponieważ liczba ocen, które należy wprowadzić, jest z góry nieznana. Dlatego za każdym razem, gdy użytkownik wprowadza prawidłową ocenę (nie -1), program musi dodać 1 do licznika, a także dodać tę ocenę do zmiennej całkowitej. Jest to procedura liczenia i sumowania kombinacji, która jest powszechna w wielu programach.

```
// Filename: C12GRAD2.CPP

// Dodaje oceny, oblicza średnią,

// i określa, czy zdobyłeś A.

#include <iostream.h>

#include <iomanip.h>

main()

{

float total_grade=0.0;

float grade_avg = 0.0;

float grade;

int grade_ctr = 0;

do

{ cout << “What is your grade? (-1 to end) “;

cin >> grade;

if (grade >= 0.0)

{ total_grade += grade; // Add to total.

grade_ctr ++; } // Add to count.

} while (grade >= 0.0); // Quit when -1 entered.

// Control begins here if no more grades.

grade_avg = (total_grade / grade_ctr); // Compute

// average.

cout << “\nYou made a total of “ << setprecision(1) <<

total_grade << “ points.\n”;

cout << “Your average was “ << grade_avg << “\n”;

if (total_grade >= 450.0)

{ cout << “** You made an A!!”; }

return 0;
```

}

Poniżej znajduje się wynik tego programu. Gratulacje! Jesteś na najlepszej drodze do zostania głównym programistą C++.

What is your grade? (-1 to end) 67.8

What is your grade? (-1 to end) 98.7

What is your grade? (-1 to end) 67.8

What is your grade? (-1 to end) 92.4

What is your grade? (-1 to end) -1

You made a total of 326.68 points.

Your average was 81.7

Ćwiczenia

1. Napisz program z pętlą „do-while", która wyświetla cyfry od 10 do 20 (włącznie), z pustą linią między każdą liczbą.

2. Napisz program kalkulatora pogody, który prosi o listę temperatur z ostatnich 10 dni, oblicza średnią i drukuje wyniki. Musisz obliczyć sumę w miarę wprowadzania danych wejściowych, a następnie podzielić ją przez 10, aby znaleźć średnią. Użyj pętli while dla 10 powtórzeń.

3. Przepisz program w ćwiczeniu 2 za pomocą pętli „do-while".

4. Napisz program, podobny do kalkulatora pogody w ćwiczeniu 2, ale uogólnij go, aby obliczał średnią z dowolnej liczby dni z temperaturą. (Wskazówka: musisz policzyć liczbę temperatur, aby obliczyć średnią końcową.)

5. Napisz program, który tworzy własną tabelę ASCII na ekranie. Nie drukuj pierwszych 31 znaków, ponieważ nie można ich wydrukować. Wydrukuj kody o numerach od 32 do 255, przechowując ich liczby w zmiennych całkowitych i drukując ich wartości ASCII za pomocą printf () i kodu formatu „% c".

Podsumowanie

Przedstawiono dwa sposoby tworzenia pętli C++: pętla while i pętla do-while. Te dwie odmiany pętli while różnią się miejscem testowania instrukcji warunków testowych. Testy while na początku pętli i testy do-while na końcu. Dlatego ciało pętli „do-while" zawsze wykonuje się co najmniej raz. Dowiedziałeś się również, że funkcja exit () i instrukcja break zwiększają elastyczność pętli while. Funkcja exit () kończy program, a instrukcja break kończy tylko bieżącą pętlę. W tym rozdziale opisano dwa najważniejsze zastosowania pętli: liczniki i sumy. Twój komputer może być wspaniałym narzędziem do dodawania i liczenia, ze względu na powtarzalne możliwości oferowane przez pętle while.

Pętla for

Pętla for pozwala powtarzać sekcje programu określoną liczbę razy. W przeciwieństwie do pętli while i do-while, for

Pętla for jest pętlą wyznaczoną. Oznacza to, że podczas pisania programu zwykle można określić liczbę iteracji pętli. Pętle while i do-while powtarzają się tylko do momentu spełnienia warunku. Pętla for robi to i więcej: kontynuuje zapętlanie, aż do osiągnięcia liczby (lub odliczania). Po osiągnięciu ostatecznej liczby pętli wykonanie jest kontynuowane z kolejną instrukcją, po kolei. Tu omówiono konstrukcję pętli for wprowadzając

♦ Instrykcja for

♦ Pojęcie pętli

♦ Zagnieżdżone dla pętli

Pętla for jest pomocnym sposobem zapętlania sekcji kodu, gdy chcesz policzyć lub zsumować określone kwoty, ale nie zastępuje ona pętli while i do-while.

Instrukcja for

Instrukcja for zawiera jedną lub więcej instrukcji C++, które tworzą treść pętli. Te instrukcje w pętli ciągle powtarzają się określoną liczbę razy. Ty jako programista będziesz kontrolować liczbę powtórzeń pętli.

Format pętli for to

for (wyrażenie początkowe; wyrażenie testowe; liczenie wyrażenia)

{Blok jednej lub więcej instrukcji C++; }

C++ ocenia wyrażenie początkowe przed rozpoczęciem pętli. Zazwyczaj wyrażenie początkowe jest instrukcją przypisania (na przykład ctr = 1;), ale może być dowolnym wyrażeniem prawnym określonym przez użytkownika. C++ ocenia wyrażenie początkowe tylko raz, u góry pętli.

UWAGA: Nie umieszczaj średnika po prawidłowym nawiasie. Jeśli to zrobisz, pętla for interpretuje treść pętli jako długie instrukcje zerowe! Kontynuowałoby zapętlanie - za każdym razem nic nie robiąc - dopóki testowe wyrażenie nie zmieni się w False

Za każdym razem, gdy treść pętli się powtarza, wykonywane jest wyrażenie liczenia, zwykle zwiększające lub zmniejszające zmienną. Test wyrażenia ma wartość True (niezerową) lub False (zero), a następnie określa, czy treść pętli powtarza się ponownie.

WSKAZÓWKA: Jeśli w ciele pętli for znajduje się tylko jedna instrukcja C++, nawiasy klamrowe nie są wymagane, ale są zalecane. Jeśli dodasz więcej instrukcji, nawiasy klamrowe już tam są, przypominając, że są one teraz potrzebne.

Koncepcja pętli

Używasz koncepcji pętli for przez całe życie. Za każdym razem, gdy trzeba powtórzyć określoną procedurę określoną liczbę razy, powtórzenie to staje się dobrym kandydatem do skomputeryzowanej pętli. Aby zilustrować koncepcję pętli for, załóżmy, że instalujesz 10 nowych żaluzji w domu. Musisz wykonać następujące czynności kroki dla każdej żaluzji:

1. Przesuń drabinę w miejsce żaluzji.

2. Weź żaluzję, młotek i gwoździe w górę drabiny.

3. Uderz żaluzją w bok domu.

4. Zejdź po drabinie.

Musisz wykonać każdy z tych czterech kroków dokładnie 10 razy, ponieważ masz 10 okiennic. Po 10 razie nie instalujesz kolejnej aluzji, ponieważ zadanie zostało zakończone. Zapętlasz procedurę składającą się z kilku kroków (blok pętli). Te kroki są ciałem pętli. To nie jest nieskończona pętla, ponieważ istnieje stała liczba okiennic; zabrakło okiennic dopiero po zainstalowaniu wszystkich 10. Dla mniej fizycznego przykładu, który można łatwiej skomputeryzować, załóżmy, że musisz wypełnić trzy deklaracje podatkowe dla każdego z nastoletnich dzieci. (Jeśli masz troje nastoletnich dzieci, prawdopodobnie potrzebujesz więcej niż komputera, aby pomóc ci przetrwać dzień!) Dla każdego dziecka musisz wykonać następujące czynności:

1. Dodaj całkowity dochód.

2. Dodaj sumę odliczeń.

3. Wypełnij zeznanie podatkowe.

4. Włóż do koperty.

5. Wyślij pocztą.

Następnie musisz powtórzyć całą procedurę jeszcze dwa razy. Zauważ, jak zaczęło się zdanie przed tymi krokami: Dla każdego dziecka. To sygnalizuje pomysł podobny do konstrukcji pętli for.

UWAGA: Pętla for testuje wyrażenie testowe na górze pętli. Jeśli wyrażenie testowe ma wartość False, gdy rozpoczyna się pętla for, ciało pętli nigdy się nie wykonuje

Wybór pętli

Dowolną konstrukcję pętli można zapisać za pomocą pętli for, loop while lub do-while. Zasadniczo używasz pętli for, gdy chcesz policzyć lub zapętlić określoną liczbę razy, i rezerwować pętle while i do-while na pętlę, dopóki warunek False nie zostanie spełniony.

Przykłady

1. Aby rzucić okiem na możliwości pętli for, ten przykład pokazuje dwa programy: jeden, który używa pętli for, a drugi nie. Pierwszy to program zliczający. Przed przestudiowaniem jego zawartości spójrz na wynik. Wyniki bardzo dobrze ilustrują koncepcję pętli for. Zidentyfikuj program i dołącz niezbędny plik nagłówka. Potrzebujesz licznika, więc uczyń ctr zmienną całkowitą.

1. Dodaj jeden do licznika.

2. Jeśli licznik jest mniejszy lub równy 10, wydrukuj jego wartość i powtórz krok pierwszy.

Program z pętlą for wygląda następująco:

```
// Nazwa pliku: C13FOR1.CPP

// Wprowadza pętlę for.

#include <iostream.h>

main()
```

```
{
int ctr;
for (ctr=1; ctr<=10; ctr++) // Uruchom ctr od razu.
// Przyrost przez pętlę.
{ cout << ctr << "\n"; } // Ciało pętli for
return 0;
}
```

Wyjście tego programu to

```
1
2
3
4
5
6
7
8
9
10
```

Oto ten sam program korzystający z pętli „do-while":

Zidentyfikuj program i dołącz niezbędny plik nagłówka. Potrzebujesz licznika, więc uczyń ctr zmienną całkowitą.

1. Dodaj jeden do licznika.

2. Wydrukuj wartość licznika.

3. Jeśli licznik jest mniejszy lub równy 10, powtórz krok pierwszy.

```
// Nazwa pliku: C13WHI1.CPP
// Symulowanie pętli for za pomocą pętli do-while.
#include <iostream.h>
main()
{
int ctr=1;
do
```

```
{ cout << ctr << “\n”; // Ciało pętli do-while.

ctr++; }

while (ctr <= 10);

return 0;

}
```

Zauważ, że pętla for jest czystszym sposobem kontrolowania procesu zapętlania. Pętla for wykonuje kilka czynności, które wymagają dodatkowych instrukcji w pętli while. W przypadku pętli nie trzeba pisać dodatkowego kodu, aby inicjalizować zmienne oraz zwiększać lub zmniejszać je. Możesz zobaczyć na pierwszy rzut oka (w wyrażeniach w instrukcji for) dokładnie, jak działa pętla, w przeciwieństwie do do-while, która zmusza cię do spojrzenia na spód pętli, aby zobaczyć, jak pętla się zatrzymuje.

2. Oba poniższe programy przykładowe dodają liczby od 100 do 200. Pierwszy używa pętli for; drugi nie. Pierwszy przykład zaczyna się od wyrażenia początkowego większego niż 1, tym samym rozpoczynając pętlę również od wyrażenia o większej liczbie. Ten program ma pętlę for:

```
// Nazwa pliku: C13FOR2.CPP

// Pokazuje sumowanie za pomocą pętli for.

#include <iostream.h>

main()

{

int total, ctr;

total = 0; // Holds a total of 100 to 200.

for (ctr=100; ctr<=200; ctr++) // ctr is 100, 101,

// 102,...200

{ total += ctr; } // Add value of ctr to each iteration.

cout << “The total is “ << total << “\n”;

return 0;

}
```

Ten sam program bez pętli for wygląda następująco:

```
// Nazwa pliku: C13WHI2.CPP

// Program sumujący korzystający z pętli „do-while”.

#include <iostream.h>

main()

{

int total=0; // Initialize total
```

```
int num=100; // Starting value

do

{ total += num; // Add to total

num++; // Increment counter

} while (num <= 200);

cout << “The total is “ << total << “\n”;;

return 0;

}
```

Oba programy generują ten wynik:

```
The total is 15150
```

Ciało pętli w obu programach wykonuje się 101 razy. Wartość początkowa to 101, a nie 1, jak w poprzednim przykładzie. Zauważ, że pętla for jest mniej złożona niż do-while, ponieważ inicjalizacja, testowanie i inkrementacja są wykonywane w pojedynczej instrukcji for.

WSKAZÓWKA: Zwróć uwagę na wcięcie ciała pętli for. Jest to dobry nawyk do rozwijania się, ponieważ ułatwia zobaczenie początku i końca ciała pętli.

3. Treść pętli for może zawierać więcej niż jedną instrukcję. Poniższy przykład wymaga pięciu par wartości danych: imion dzieci i ich wieku. Wyświetla nauczyciela przypisanego do każdego dziecka, na podstawie jego wieku. To ilustruje pętlę for z funkcjami cout, funkcją cin i instrukcją if w jej ciele. Ponieważ sprawdzane jest dokładnie pięcioro dzieci, pętla for zapewnia zakończenie programu po piątym potomku.

```
// Nazwa pliku: C13FOR3.CPP

// Program, który używa pętli do wprowadzania i drukowania

// nauczyciel przypisany do każdego dziecka.

#include <iostream.h>

main()

{

char child[25]; // Holds child’s first name

int age; // Holds child’s age

int ctr; // The for loop counter variable

for (ctr=1; ctr<=5; ctr++)

{ cout << “What is the next child’s name? “;

cin >> child;

cout << “What is the child’s age? “;

cin >> age;
```

```
if (age <= 5)
{ cout << "\n" << child << " has Mrs. "
<< "Jones for a teacher\n"; }
if (age == 6)
{ cout << "\n" << child << " has Miss "
<< "Smith for a teacher\n"; }
if (age >= 7)
{ cout << "\n" << child << " has Mr. "
<< "Anderson for a teacher\n"; }
} // Quits after 5 times
return 0;
}
```

Poniżej znajduje się wynik tego programu. Możesz ulepszyć ten program jeszcze bardziej po zapoznaniu się z instrukcją switch w następnym rozdziale.

```
What is the next child's name? Joe
What is the child's age? 5
Joe has Mrs. Jones for a teacher
What is the next child's name? Larry
What is the child's age? 6
Larry has Miss Smith for a teacher
What is the next child's name? Julie
What is the child's age? 9
Julie has Mr. Anderson for a teacher
What is the next child's name? Manny
What is the child's age? 6
Manny has Miss Smith for a teacher
What is the next child's name? Lori
What is the child's age? 5
Lori has Mrs. Jones for a teacher
```

4. Poprzednie przykłady wykorzystywały przyrost jako wyrażenie zliczające. Możesz zrobić, aby pętla for zwiększała zmienną pętli o dowolną wartość. Nie musi zwiększać o 1. Poniższy program wypisuje liczby parzyste od 1 do 20. Następnie wypisuje liczby nieparzyste od 1 do 20. W tym celu do zmiennej

licznika dodaje się dwa (zamiast jednego, jak pokazano w poprzednich przykładach) za każdym razem, gdy wykonuje się pętla.

```
// Nazwa pliku: C13EVOD.CPP

// Drukuje liczby parzyste od 1 do 20,

// następnie liczby nieparzyste od 1 do 20.

#include <iostream.h>

main()

{

int num; // The for loop variable

cout << “Even numbers below 21\n”; // Title

for (num=2; num<=20; num+=2)

{ cout << num << “ “; } // Prints every other number.

cout << “\nOdd numbers below 20\n”; // A second title

for (num=1; num<=20; num+=2)

{ cout << num << “ “; } // Prints every other number.

return 0;

}
```

W tym programie są dwie pętle. Ciało każdego z nich składa się z jednej funkcji printf(). W pierwszej połowie programu zmienna pętli, num, wynosi 2, a nie 1. Gdyby była 1, liczba 1 byłaby drukowana jako pierwsza, podobnie jak w sekcji liczb nieparzystych. Dwie instrukcje cout, które drukują tytuły, nie są częścią żadnej pętli. Gdyby tak było, program wydrukowałby tytuł przed każdym numerem. Poniżej przedstawiono wynik uruchomienia tego programu.

```
Liczby parzyste poniżej 21

2 4 6 8 10 12 14 16 18 20

Dziwne liczby poniżej 20

1 3 5 7 9 11 13 15 17 19
```

5. Możesz również zmniejszyć wartość zmiennej pętli. Jeśli to zrobisz, wartość jest odejmowana od zmiennej pętli za każdym razem przez pętlę. Poniższy przykład to przepisanie programu zliczającego. Daje efekt odwrotny, pokazując odliczanie.

```
// Nazwa pliku: C13CNTD1.CPP

// Odliczanie do startu.

#include <iostream.h>

main()

{
```

```
int ctr;

for (ctr=10; ctr!=0; ctr--)

{ cout << ctr << "\n"; } // Print ctr as it

// counts down.

cout << "*** Blast off! ***\n";

return 0;

}
```

Podczas zmniejszania zmiennej pętli wartość początkowa powinna być większa niż testowana wartość końcowa. W tym przykładzie zmienna pętli ctr odlicza od 10 do 1. Za każdym razem, gdy pętla (każda iteracja), ctr jest zmniejszana o jeden. Możesz zobaczyć, jak łatwo kontrolować pętlę, patrząc na wyniki tego programu, w następujący sposób.

```
10

9

8

7

6

5

4

3

2

1

*** Blast Off! ***
```

WSKAZÓWKA: Ten program do testu pętli ilustruje nadmiarowość, którą można wyeliminować dzięki C++. Wyrażenie testowe, ctr! = 0; mówi pętli for, aby kontynuowała zapętlenie, aż ctr nie będzie równa zero. Jeśli jednak ctr zmieni się na zero (wartość False), nie ma powodu, aby dodawać dodatkowe! = 0 (z wyjątkiem przejrzystości). Możesz przepisać pętlę for jako

```
for (ctr=10; ctr; ctr--)
```

bez utraty znaczenia. Jest to bardziej wydajne i tak integralna część C++, że powinieneś się z nim dobrze czuć. Gdy dostosujesz się do tego, utrata jasności jest niewielka.

6. Możesz także wykonać test pętli for dla czegoś innego niż wartość dosłowna. Poniższy program łączy wiele z tego, czego nauczyłeś się do tej pory. Prosi o oceny uczniów i oblicza średnią. Ponieważ w każdym semestrze może być inna liczba studentów, program najpierw pyta użytkownika o liczbę studentów. Następnie program iteruje, aż użytkownik wprowadzi taką samą liczbę wyników. Następnie oblicza średnią na podstawie sumy i liczby wprowadzonych ocen uczniów.

```
// Nazwa pliku: C13FOR4.CPP
```

```
// Oblicza średnią ocen za pomocą pętli for.
#include <iostream.h>
#include <iomanip.h>
main()
{
float grade, avg;
float total=0.0;
int num; // Total number of grades.
int loopvar; // Used to control the for loop
cout << "\n*** Grade Calculation ***\n\n"; // Title
cout << "How many students are there? ";
cin >> num; // Get total number to enter
for (loopvar=1; loopvar<=num; loopvar++)
{ cout << "\nWhat is the next student's grade? ";
cin >> grade;
total += grade; } // Keep a running total
avg = total / num;
cout << "\n\nThe average of this class is " <<
setprecision(1) << avg;
return 0;
}
```

Z powodu pętli for obliczenia całkowite i średnie nie muszą być zmieniane, jeśli zmienia się liczba uczniów.

7. Ponieważ znaki i liczby całkowite są tak ściśle powiązane w C++, możesz zwiększać zmienne znakowe w pętli for. Poniższy program wypisuje litery od A do Z za pomocą prostej pętli for.

```
// Nazwa pliku: C13FOR5.CPP
// Drukuje alfabet za pomocą prostej pętli for.
#include <iostream.h>
main()
{
char letter;
cout << "Here is the alphabet:\n";
```

```
for (letter='A'; letter<='Z'; letter++) // Loops A to Z

{ cout << " " << letter; }

return 0;

}
```

Ten program generuje następujące dane wyjściowe:

```
Oto alfabet:

ABCDEFGHIJKLMNOPQRSTU VWXYZ
```

8. Dla wyrażenia może być puste lub puste. W pętli for wszystkie wyrażenia są puste:

```
for (;;)

{ printf("Over and over..."); }
```

To dla pętli iteruje się na zawsze. Chociaż powinieneś unikać nieskończonych pętli, twój program może nakazać, abyś pozostawił puste wyrażenie for. Jeśli już zainicjowałeś wyrażenie początkowe wcześniej w programie, marnujesz czas na komputer, aby powtórzyć je w pętli for - a C++ tego nie wymaga. Poniższy program pomija wyrażenie początkowe i wyrażenie liczące, pozostawiając tylko wyrażenie testowe pętli for. Przez większość czasu musisz pominąć tylko jeden z nich. Jeśli używasz pętli for bez dwóch jej wyrażeń, rozważ zamianę z pętlą while lub do-while.

```
// Nazwa pliku: C13FOR6.CPP

// Używa tylko wyrażenia testowego w

// pętla for, która ma być liczona przez piątki.

#include <iostream.h>

main()

{

int num=5; // Starting value

cout << "\nCounting by 5s: \n"; // Title

for (; num<=100;) // Contains only the test expression.

{ cout << num << "\n";

num+=5; // Increment expression outside the loop.

} // End of the loop's body

return 0;

}
```

Dane wyjściowe z tego programu są następujące:

```
Counting by 5s:

5
```

10

15

20

25

30

35

40

45

50

55

60

65

70

75

80

85

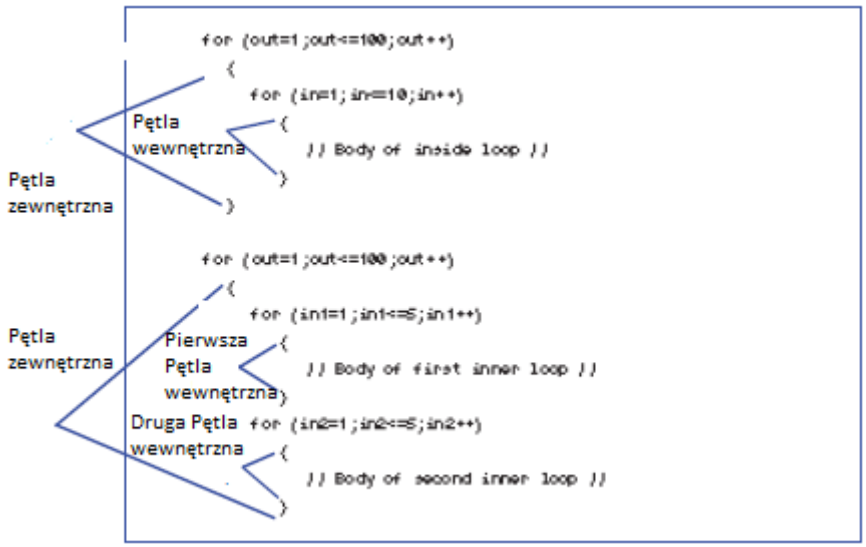
90

95

100

Zagnieżdżenie pętli

Każda instrukcja C++ może wejść do wnętrza pętli for - nawet inna dla pętli! Kiedy umieszczasz pętlę w pętli, tworzysz zagnieżdżoną pętlę. Zegar podczas wydarzenia sportowego działa jak zagnieżdżona pętla. Może się wydawać, że ta analogia nieco się rozciąga, ale naprawdę działa. Mecz piłki nożnej odlicza od 15 minut do 0. Robi to cztery razy. Pierwsze pętle odliczania od 15 do 0 (za każdą minutę). Odliczanie jest zagnieżdżone w innym, który zapętla od 1 do 4 (dla każdej z czterech ćwiartek). Jeśli twój program musi powtarzać pętlę więcej niż jeden raz, jest dobrym kandydatem do zagnieżdżonej pętli. Rysunek pokazuje dwa kontury zagnieżdżonych pętli. Możesz myśleć o pętli wewnętrznej jako zapętlonej „szybciej” niż pętli zewnętrznej. W pierwszym przykładzie wewnętrzna pętla for liczy od 1 do 10, zanim zewnętrzna pętla (zmienna out) może zakończyć swoją pierwszą iterację. Kiedy pętla zewnętrzna w końcu wykonuje iterację po raz drugi, pętla wewnętrzna zaczyna się od nowa.



Drugi kontur zagnieżdżonej pętli pokazuje dwie pętle w zewnętrznej pętli. Obie pętle wykonują się w całości, zanim pętla zewnętrzna zakończy swoją pierwszą iterację. Kiedy zewnętrzna pętla rozpoczyna drugą iterację, dwie wewnętrzne pętle powtarzają się ponownie. Zwróć uwagę na kolejność nawiasów klamrowych w każdym przykładzie. Pętla wewnętrzna zawsze kończy się, dlatego jej klamra końcowa musi nadejść przed klamrą końcową pętli zewnętrznej. Wcięcie czyni to o wiele wyraźniejszym, ponieważ można wyrównać nawiasy klamrowe każdej pętli.

UWAGA: W zagnieżdżonych pętlach pętla wewnętrzna (lub pętle) wykonuje się całkowicie przed następną iteracją pętli zewnętrznej.

Przykłady

1. Poniższy program zawiera pętlę w pętli - zagnieżdżoną pętlę. Pętla wewnętrzna liczy się i drukuje od 1 do 5. Pętla zewnętrzna liczy się od 1 do 3. Pętla wewnętrzna powtarza się w całości trzy razy. Innymi słowy, ten program wypisuje wartości od 1 do 5 i robi to trzy razy.

```
// Nazwa pliku: C13NEST1.CPP

// Wydrukuj cyfry 1-5 trzy razy.

// za pomocą zagnieżdżonej pętli.

#include <iostream.h>

main()

{

int times, num; // Outer and inner for loop variables

for (times=1; times<=3; times++)

{

for (num=1; num<=5; num++)

{ cout << num; } // Inner loop body

cout << “\n”;
```

```
} // End of outer loop

return 0;

}
```

Wcięcie jest zgodne ze standardem dla pętli; każda instrukcja w każdej pętli jest wcięta o kilka spacji. Ponieważ wewnętrzna pętla jest już wcięta, jej ciało jest wcięte jeszcze kilka spacji. Dane wyjściowe programu są następujące:

```
12345

12345

12345
```

2. Zmienna licznika zewnętrznej pętli zmienia się za każdym razem w pętli. Jeśli jedna ze zmiennych kontrolnych pętli wewnętrznej jest zmienną licznika pętli zewnętrznej, zobaczysz takie efekty, jak pokazano w poniższym programie.

```
 // Nazwa pliku: C13NEST2.CPP

// Wewnętrzna pętla kontrolowana przez zewnętrzną pętlę

// zmienna licznika.

#include <iostream.h>

main()

{

int outer, inner;

for (outer=5; outer>=1; outer--)

{ for (inner=1; inner<=outer; inner++)

{ cout << inner; } // End of inner loop.

cout << "\n";

}

return 0;

}
```

Wyjście z tego programu następuje. Wewnętrzna pętla powtarza się pięć razy (ponieważ wartość zewnętrzna odlicza od 5 do 1) i drukuje od pięciu cyfr do jednej liczby.

```
12345

1234

123

12

1
```

Poniższa tabela śledzi dwie zmienne w tym programie. Czasami musisz „grać na komputerze”, ucząc się nowej koncepcji, takiej jak zagnieżdżone pętle. Wykonując wiersz za jednym razem i zapisując zawartość każdej zmiennej, tworzysz tę tabelę.

Zmienna zewnętrzna: zmienna wewnętrzna

5: 1

5: 2

5: 3

5: 4

5: 5

4: 1

4: 2

4: 3

4: 4

3: 1

3: 2

3: 3

2: 1

2: 2

1: 1

Wskazówka dla matematyków

Instrukcja for jest identyczna z matematycznym symbolem sumowania. Podczas pisania programów symulujących symbol sumowania instrukcja for jest doskonałym kandydatem. Instrukcja zagnieżdżona jest dobra w przypadku podwójnych sumowań. Na przykład następujące podsumowanie

i = 30

Σ (i / 3 * 2)

i = 1

można przepisać jako

total = 0;

for (i=1; i<=30; i++)

{ total += (i / 3 * 2); }

4. Silnia to liczba matematyczna stosowana w teorii prawdopodobieństwa i statystyce. Silnia liczby jest iloczynem iloczynu każdej liczby od 1 do liczby, o której mowa. Na przykład silnia 4 wynosi 24, ponieważ 4 × 3 × 2 × 1 = 24. Silnia 6 wynosi 720, ponieważ 6 × 5 × 4 × 3 × 2 × 1 = 720. Silnia 1 jest z definicji

1. Zagnieżdżone pętle są dobrymi kandydatami do napisania programu silni do generowania liczb. Poniższy program pyta użytkownika o liczbę, a następnie wypisuje silnię tej liczby.

```
// Nazwa pliku: C13FACT.CPP

// Oblicza silnię liczb przez

// liczbę użytkownika.

#include <iostream.h>

main()

{

int outer, num, fact, total;

cout << “What factorial do you want to see? “;

cin >> num;

for (outer=1; outer <= num; outer++)

{ total = 1; // Initialize total for each factorial.

for (fact=1; fact<= outer; fact++)

{ total *= fact; } // Compute each factorial.

}

cout << “The factorial for “ << num << “ is “

<< total;

return 0;

}
```

Poniżej pokazano silnię siedem. Możesz uruchomić ten program, wprowadzając różne wartości, gdy zostaniesz o to poproszony i zobaczyć różne silnie. Uważaj: silnie mnożą się szybko. (Silnia 11 nie pasuje do zmiennej całkowitej). J

Jaki czynnik chcesz zobaczyć? 7

Silnia dla 7 wynosi 5040

Ćwiczenia

1. Napisz program, który drukuje na ekranie cyfry od 1 do 15. Użyj pętli for, aby kontrolować drukowanie. 2. Napisz program do drukowania cyfr 15 na 1 na ekranie. Użyj pętli for, aby kontrolować wyświetlanie.

3. Napisz program, który używa pętli for do drukowania każdej liczby nieparzystej od 1 do 100.

4. Napisz program, który pyta użytkownika o jej wiek. Użyj pętli for, aby wydrukować „Happy Birthday!” na każdy rok wieku użytkownika.

5. Napisz program, który używa pętli for do drukowania znaków ASCII od 32 do 255 na ekranie. (Wskazówka: użyj znaku konwersji% c, aby wydrukować zmienne całkowite.)

6. Używając numerów tabel ASCII, napisz program, aby wydrukować następujące dane wyjściowe, używając zagnieżdżonej pętli for. (Wskazówka: Pętla zewnętrzna powinna zapętlać od 1 do 5, a zmienna początkowa pętli wewnętrznej powinna wynosić 65, wartość ASCII A.)

ZA

AB

ABC

ABCD

ABCDE

Podsumowanie

Nauczyłeś się jak kontrolować pętle. Zamiast pisać dodatkowy kod wokół pętli while, można użyć pętli for do kontrolowania liczby iteracji w momencie definiowania pętli. Całą pętla for zawiera trzy części: wyrażenie początkowe, wyrażenie testowe i wyrażenie zliczające. Widziałeś teraz trzy konstrukcje pętli C++: pętla while, pętla do-while i pętla for. Są podobne, ale zachowują się inaczej w sposobie testowania i inicjalizacji zmiennych. Żadna pętla nie jest lepsza od innych. Problem z programowaniem powinien dyktować, której pętli użyć.

Inne opcje pętli

Po opanowaniu konstrukcji pętli powinieneś nauczyć się kilku instrukcji związanych z pętlą. Ta czeęść uczy pojęć dotyczących pętli czasowych, które umożliwiają spowolnienie programów. Spowolnienie wykonywania programu może być pomocne, jeśli chcesz wyświetlać komunikat przez określony czas lub pisać gry komputerowe z mniejszymi prędkościami, aby były praktyczne do użytku rekreacyjnego. Do sterowania pętlami można użyć dwóch dodatkowych poleceń pętli, instrukcji break i continu. Te instrukcje działają z pętlami while i pętlami.

Ta sekcja zawiera następujące informacje:

♦ Pętle czasowe

♦ Instrukcja break z pętlami for

♦ Instrukcja kontynuacji z pętlami for

Po opanowaniu tych koncepcji będziesz na dobrej drodze do pisania potężnych programów przetwarzających duże ilości danych.

Pętle czasowe

Komputery są szybkie i czasami chciałbyś, aby były jeszcze szybsze. Czasami jednak chcesz spowolnić komputer. Często trzeba spowolnić wykonywanie gier, ponieważ szybkość komputera uniemożliwia grę. Wiadomości, które pojawiają się na ekranie wiele razy, są usuwane zbyt szybko, aby użytkownik mógł je odczytać i nie opóźniać ich. Zagnieżdżona pętla jest idealnym miejscem na pętlę taktowania, która po prostu wielokrotnie przechodzi przez pętlę for lub while. Im większa wartość końcowa pętli for, tym dłuższy jest czas powtarzania się pętli. Zagnieżdżona pętla jest odpowiednia do wyświetlania komunikatów o błędach użytkownikowi. Jeśli użytkownik zażądał raportu - ale nie wprowadził wystarczającej ilości danych, aby program mógł go wydrukować - możesz wydrukować komunikat ostrzegawczy na ekranie przez kilka sekund, informując użytkowników, że nie może jeszcze poprosić o raport. Po wyświetleniu wiadomości przez kilka sekund możesz ją usunąć i dać użytkownikowi kolejną szansę. Nie ma sposobu, aby określić, ile iteracji pętla taktowania zajmuje jedną sekundę (lub minutę lub godzinę) opóźnienia, ponieważ komputery biegać z różnymi prędkościami. Dlatego musisz dostosować wartość końcową pętli czasowej, aby ustawić opóźnienie według własnych upodobań.

Przykłady

1. Pętle czasowe są łatwe do napisania - wystarczy umieścić pustą pętlę for wewnątrz programu. Poniższy program to przepisana wersja programu odliczającego (C13CNTD1.CPP), którą widziałeś w wcześniej Każda liczba w odliczaniu jest opóźniona, więc odliczanie nie wydaje się mieć miejsca natychmiast. (Dostosuj wartość opóźnienia, jeśli ten program działa zbyt wolno lub zbyt szybko na twoim komputerze.) Zidentyfikuj program i dołącz plik nagłówkowy wejścia / wyjścia. Potrzebujesz licznika i opóźnienia, więc utwórz zmienne całkowite cd i opóźnienie Uruchom licznik od 10, a opóźnienie od 1.

1. Jeśli opóźnienie jest mniejsze lub równe 30 000, dodaj 1 do jego wartości i powtórz krok pierwszy.

2. Wyświetl wartość licznika.

3. Jeśli licznik jest większy lub równy 0, odejmij 1 od jego wartości i powtórz krok pierwszy.

Wyświetl komunikat odpalenia.

```
// Nazwa pliku: C14CNTD1.CPP

// Odliczanie do startu z opóźnieniem.

#include <iostream.h>

main()

{

int cd, delay;

for (cd=10; cd>=0; cd--)

{ { for (delay=1; delay <=30000; delay++); } // Delay

// program.

cout << cd << "\n"; // Print countdown value.

} // End of outer loop

cout << "Blast off!!! \n";

return 0;

}
```

2. Następujący program pyta użytkowników o wiek. Jeśli użytkownik wprowadzi wiek poniżej 0, program wyemituje sygnał dźwiękowy (drukując znak alarmu, \a), a następnie wyświetli komunikat o błędzie przez kilka sekund za pomocą zagnieżdżonej pętli pomiaru czasu. Ponieważ liczba całkowita nie ma wystarczająco dużej wartości (na wielu komputerach) dla długiej pętli synchronizacji, należy użyć zagnieżdżonej pętli synchronizacji. (W zależności od szybkości komputera dostosuj liczby w pętli, aby wyświetlać wiadomość dłużej lub krócej.)

Program wykorzystuje rzadko spotykany znak konwersji printf () \r wewnątrz pętli. Jak pamiętacie, r jest znakiem powrotu karetki. Ten znak konwersji przesuwa kursor na początek bieżącego wiersza, umożliwiając programowi drukowanie pustych wierszy w tym samym wierszu. Ten proces zastępuje komunikat o błędzie i wygląda na to, że błąd znika z ekranu po krótkiej przerwie.

```
// Nazwa pliku: C14TIM.CPP

// Wyświetla komunikat o błędzie przez kilka sekund.

#include <stdio.h>

main()

{

int outer, inner, age;

printf("What is your age? ");

scanf(" %d", &age);

while (age <= 0)

{ printf("*** Your age cannot be that small! ***");
```

```
// Timing loop here

for (outer=1; outer<=30000; outer++)

{ for (inner=1; inner<=500; inner++); }

// Erase the message

printf("\r\n\n");

printf("What is your age? ");

scanf(" %d", &age); // Ask again

}

printf("\n\nThanks, I did not think you would actually tell");

printf("me your age!");

return 0;

}
```

UWAGA: Zauważ, że wewnętrzna pętla ma średnik (;) po instrukcji for - bez treści pętli. Tu nie ma potrzeby tworzenia korpusu pętli, ponieważ komputer krąży w pętli, aby tracić trochę czasu.

Instrukcje break i for

Pętla for została zaprojektowana do wykonywania określoną liczbę razy. W rzadkich przypadkach może być konieczne wcześniejsze zamknięcie pętli for zmienna zliczająca osiągnęła wartość końcową. Podobnie jak w przypadku pętli while, należy użyć instrukcji break, aby wcześniej wyjść z pętli for.

Instrukcja break jest zagnieżdżona w treści pętli for. Programiści rzadko samodzielnie przerywają linię i prawie zawsze pojawia się po teście if. Gdyby przerwa była sama na linii, pętla zawsze kończyłaby się wcześniej, pokonując cel pętli for.

Przykłady

1. Poniższy program pokazuje, co może się zdarzyć, gdy C++ napotka bezwarunkową instrukcję break (nie poprzedzoną instrukcją if). Zidentyfikuj program i dołącz pliki nagłówkowe wejścia / wyjścia. Potrzebujesz zmiennej do przechowywania bieżącego numeru, więc uczyń num zmienną całkowitą. Wydrukuj komunikat „Oto liczby".

1. Ustaw num równy 1. Jeśli num jest mniejszy lub równy 20, dodaj jeden za każdym razem przez pętlę.

2. Wyświetl wartość liczby.

3. Wyjdź z pętli.

Wyświetlj wiadomość do widzenia.

```
// Nazwa pliku: C14BRAK1.CPP

// Pętla for pokonana przez instrukcję break.

#include <iostream.h>

main()
```

```
{
int num;
cout << “Here are the numbers from 1 to 20\n”;
for(num=1; num<=20; num++)
{ cout << num << “\n”;
break; } // This line exits the for loop immediately.
cout << “That’s all, folks!”;
return 0;
}
```

Poniżej pokazano wynik uruchomienia tego programu. Zauważ, że przerwa natychmiast kończy pętlę for. Pętla for może równie dobrze nie znajdować się w tym programie. Oto liczby od 1 do 20

1

To wszystko, ludzie!

2. Poniższy program jest ulepszoną wersją poprzedniego przykładu. Pyta użytkowników, czy chcą zobaczyć inny numer. Jeśli tak, pętla for kontynuuje następną iterację. Jeśli nie, instrukcja break kończy pętlę for.

```
// Nazwa pliku: C14BRAK2.CPP
// A dla pętli działającej na żądanie użytkownika.
#include <iostream.h>
main()
{
int num; // Loop counter variable
char ans;
cout << “Here are the numbers from 1 to 20\n”;
for (num=1; num<=20; num++)
{ cout << num << “\n”;
cout << “Do you want to see another (Y/N)? “;
cin >> ans;
if ((ans == ‘N’) || (ans == ‘n’))
{ break; } // Will exit the for loop
// if user wants to.
}
```

```
cout << “\nThat’s all, folks!\n”;

return 0;

}
```

Poniższy ekran pokazuje przykładowe uruchomienie tego programu. Pętla for drukuje 20 liczb, o ile użytkownik nie odpowie N na monit. W przeciwnym razie break przerywa wcześniej pętlę for. Instrukcja po treści pętli jest zawsze wykonywana jako następna, jeśli nastąpi break.

```
Here are the numbers from 1 to 20

1

Do you want to see another (Y/N)? Y

2

Do you want to see another (Y/N)? Y

3

Do you want to see another (Y/N)? Y

4

Do you want to see another (Y/N)? Y

5

Do you want to see another (Y/N)? Y

6

Do you want to see another (Y/N)? Y

7

Do you want to see another (Y/N)? Y

8

Do you want to see another (Y/N)? Y

9

Do you want to see another (Y/N)? Y

10

Do you want to see another (Y/N)? N

That’s all, folks!
```

Jeśli zagnieżdżasz jedną pętlę w drugiej, przerwa kończy pętlę „najbardziej aktywną” (najbardziej wewnętrzną, w której znajduje się instrukcja break).

3. Użyj warunkowego podziału (instrukcja if, po której następuje przerwa), gdy brakuje danych. Na przykład podczas przetwarzania plików danych lub wprowadzania dużych ilości danych użytkownika możesz oczekiwać 100 liczb wejściowych i otrzymać tylko 95. Możesz użyć przerwy, aby zakończyć

pętlę for przed iteracją po raz 96. Załóżmy, że nauczyciel, który zastosował program uśredniania ocen z poprzedniego rozdziału C13FOR4.CPP), wprowadził niepoprawną całkowitą liczbę uczniów. Może napisała 16, ale jest tylko 14 uczniów. Poprzednia pętla for zapętlała 16 razy, bez względu na liczbę uczniów, ponieważ zależy ona od liczby nauczycieli. Poniższy program uśredniania ocen jest bardziej wyrafinowany niż poprzedni. Pyta nauczyciela o całkowitą liczbę uczniów, ale jeśli nauczyciel chce, może wpisać –99 jako wynik ucznia. –99 nie jest uśredniane; jest używany jako wartość wyzwalająca, która pozwala wydostać się z pętli for przed jej normalnym zakończeniem.

```
// Nazwa pliku: C14BRAK3.CPP

// Oblicza średnią ocen za pomocą pętli for,

// pozwalając na wcześniejsze wyjście z instrukcją break.

#include <iostream.h>

#include <iomanip.h>

main()

{

float grade, avg;

float total=0.0;

int num, count=0; // Total number of grades and counter

int loopvar; // Used to control for loop

cout << "\n*** Grade Calculation ***\n\n"; // Title

cout << "How many students are there? ";

cin >> num; // Get total number to enter.

for (loopvar=1; loopvar<=num; loopvar++)

{ cout << "\nWhat is the next student's " <<

"grade? (-99 to quit) ";

cin >> grade;

if (grade < 0.0) // A negative number

// triggers break.

{ break; } // Leave the loop early.

count++;

total += grade; } // Keep a running total.

avg = total / count;

cout << "\n\nThe average of this class is "<<

setprecision(1) << avg;
```

return 0;

}

Zauważ, że ocena jest testowana na mniej niż 0, a nie –99,0. Nie można wiarygodnie użyć wartości zmiennoprzecinkowych do porównania w celu uzyskania równości (ze względu na ich reprezentacje na poziomie bitów). Ponieważ żadna ocena nie jest ujemna, każda liczba ujemna wyzwala instrukcję break. Poniżej pokazano, jak działa ten program.

*** Obliczanie klasy ***

Ilu jest tam uczniów? 10

Jaka jest ocena następnego ucznia? (-99, aby wyjść) 87

Jaka jest ocena następnego ucznia? (-99, aby wyjść) 97

Jaka jest ocena następnego ucznia? (-99, aby wyjść) 67

Jaka jest ocena następnego ucznia? (-99, aby wyjść) 89

Jaka jest ocena następnego ucznia? (-99, aby wyjść) 94

Jaka jest ocena następnego ucznia? (-99, aby wyjść) -99

Średnia tej klasy to: 86,8

Instrukcja continue

Instrukcja break wychodzi z pętli wcześniej, ale instrukcja continue wymusza na komputerze wykonanie kolejnej iteracji pętli. Jeśli umieścisz instrukcję Kontynuuj w treści pętli for lub a, komputer zignoruje wszelkie instrukcje w pętli, która następuje dalej.

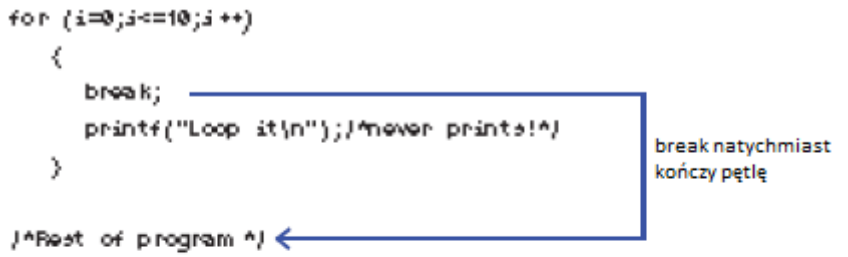
Format continue to

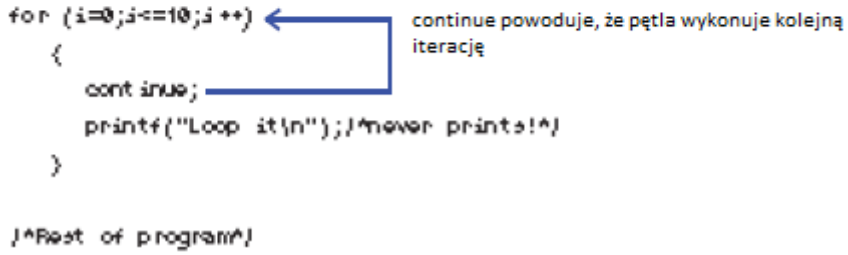
continue

Instrukcja continue jest używana, gdy dane w treści pętli są złe, poza zakresem lub nieoczekiwane. Zamiast działać na złych danych, możesz wrócić do początku pętli i wypróbować inną wartość danych. Poniższe przykłady pomagają zilustrować użycie instrukcji continue.

WSKAZÓWKA: Instrukcja continue wymusza nową iterację dowolnej z trzech konstrukcji pętli: pętla for, pętla while i pętla do-while.

Rycina pokazuje różnicę między instrukcjami break i continue.





Przykłady

1. Chociaż następujący program wydaje się drukować liczby od 1 do 10, po każdym z nich następuje „Programowanie w C++”, ale tak nie jest. continue w treści pętli for powoduje wcześniejsze zakończenie pętli. Pierwsze cout w pętli for wykonuje się, ale drugie nie - ze względu na continue.

// Nazwa pliku: C14CON1.CPP

// Demonstruje użycie instrukcji kontynuacji.

#include <iostream.h>

main()

{

int ctr;

for (ctr=1; ctr<=10; ctr++) // Loop 10 times.

{ cout << ctr << “ “;

continue; // Causes body to end early.

cout << “C++ Programming\n”;

}

return 0;

}

Ten program generuje następujące dane wyjściowe:

1 2 3 4 5 6 7 8 9 10

W niektórych kompilatorach pojawia się komunikat ostrzegawczy podczas kompilacji tego typu programu. Kompilator rozpoznaje, że drugi kod jest nieosiągalnym kodem - nigdy nie jest wykonywany

z powodu instrukcji kontynuacji. Z tego powodu większość programów nie używa kontynuacji, z wyjątkiem instrukcji if. To sprawia, że jest to warunkowa instrukcja kontynuacji, która jest bardziej przydatna. Poniższe dwa przykłady pokazują warunkowe użycie kontynuacji.

2. Ten program prosi użytkowników o pięć małych liter, po jednym na raz, i drukuje ich ekwiwalenty wielkich liter. Wykorzystuje tabelę ASCII , aby zapewnić, że użytkownicy wpisują małe litery. (Są to litery, których numery ASCII mieszczą się w zakresie od 97 do 122.) Jeśli użytkownicy nie wpisują małej litery, program ignoruje błąd w instrukcji kontynuacji.

```
// Nazwa pliku: C14CON2.CPP

// Drukuje wielkie odpowiedniki pięciu małych liter.

#include <iostream.h>

main()

{

char letter;

int ctr;

for (ctr=1; ctr<=5; ctr++)

{ cout << “Please enter a lowercase letter “;

cin >> letter;

if ((letter < 97) || (letter > 122)) // See if

// out-of-range.

{ continue; } // Go get another

letter -= 32; // Subtract 32 from ASCII value.

// to get uppercase.

cout << “The uppercase equivalent is “ <<

letter << “\n”;

}

return 0;

}
```

Ze względu na instrukcję cntinue tylko małe litery są konwertowane na wielkie.

3. Załóżmy, że chcesz uśrednić pensje pracowników swojej firmy, którzy zarabiają ponad 10 000 USD rocznie, ale masz tylko ich miesięczne wynagrodzenie brutto. Przydatny może być następujący program. Monituje o każde miesięczne wynagrodzenie pracownika, annualizuje je (mnożąc przez 12) i oblicza średnią. Instrukcja Ciąg dalszy gwarantuje, że pensje mniejsze lub równe 10 000 USD są ignorowane w obliczeniach średnich. Umożliwia „spadnięcie” pozostałych zarobków. Jeśli wpiszesz -1 jako miesięczne wynagrodzenie, program zakończy pracę i wydrukuje wynik średniej.

```
// Nazwa pliku: C14CON3.CPP
```

```
// Średnie zarobki powyżej 10 000 $
#include <iostream.h>
#include <iomanip.h>
main()
{
float month, year; // Monthly and yearly salaries
float avg=0.0, total=0.0;
int count=0;
do
{ cout << “What is the next monthly salary (-1) “ <<
“to quit)? “;
cin >> month;
if ((year=month*12.00) <= 10000.00) // Do not add
{ continue; } // low salaries.
if (month < 0.0)
{ break; } // Quit if user entered -1.
count++; // Add 1 to valid counter.
total += year; // Add yearly salary to total.
} while (month > 0.0);
avg = total / (float)count; // Compute average.
cout << “\n\nThe average of high salaries “ <<
“is $” << setprecision(2) << avg;
return 0;
}
```

Zauważ, że ten program używa zarówno instrukcji continuei, jak i instrukcji break. Program wykonuje jedną z trzech rzeczy, w zależności od danych wejściowych każdego użytkownika. Dodaje się do sumy, kontynuuje kolejną iterację, jeśli wynagrodzenie jest zbyt niskie, lub wychodzi z pętli while (i obliczeń średnich), jeśli użytkownik wpisze -1.

Poniższy ekran przedstawia dane wyjściowe z tego programu:

The following display is the output from this program:

What is the next monthly salary (-1 to quit)? 500.00

What is the next monthly salary (-1 to quit)? 2000.00

What is the next monthly salary (-1 to quit)? 750.00

What is the next monthly salary (-1 to quit)? 4000.00

What is the next monthly salary (-1 to quit)? 5000.00

What is the next monthly salary (-1 to quit)? 1200.00

What is the next monthly salary (-1 to quit)? -1

The average of high salaries is $36600.00

Ćwiczenia

1. Napisz program, który wyświetla C++ jest zabawny na ekranie przez dziesięć sekund. (Wskazówka: Może być konieczne dostosowanie pętli synchronizacji.)

2. Spraw, aby program w ćwiczeniu 1 flashował komunikat C++ jest fajny przez dziesięć sekund. (Wskazówka: może być konieczne użycie kilku pętli pomiaru czasu).

3. Napisz program uśredniania ocen dla klasy 20 uczniów. Zignoruj ocenę mniejszą niż 0 i kontynuuj do momentu wpisania wszystkich 20 ocen uczniów lub do momentu, gdy użytkownik wpisze –99, aby zakończyć program wcześniej.

4. Napisz program, który drukuje cyfry od 1 do 14 w jednej kolumnie. Po prawej stronie liczb parzystych wydrukuj kwadrat każdej liczby. Po prawej stronie liczb nieparzystych wydrukuj sześcian każdej liczby (liczba podniesiona do trzeciej potęgi).

Podsumowanie

Poznałeś kilka dodatkowych sposobów używania i modyfikowania pętli programu. Dodając pętle czasowe, instrukcje kontynuacji i instrukcje break, możesz lepiej kontrolować zachowanie każdej pętli. Możliwość wcześniejszego wyjścia (za pomocą instrukcji break) lub wcześniejszego przejścia do następnej pętli (za pomocą instrukcji continue) daje większą swobodę podczas przetwarzania różnych typów danych.

Instrukcje switch i goto

Ta częśćł koncentruje się na instrukcji switch. Poprawia także konstrukcje if i else-if, usprawniając decyzje wielokrotnego wyboru twoich programów. Instrukcja switch nie zastępuje instrukcji if, ale lepiej jest używać switch, gdy programy muszą wykonać jedną z wielu różnych akcji. Instrukcje switch i break działają razem. Prawie każda używana instrukcja switch zawiera co najmniej jedną instrukcję break w treści . Poznasz instrukcję goto, chociaż jest ona rzadko używana.

Tu opisano:

♦ Instrukcja switch używana do wyboru

♦ Instrukcja goto używana do rozgałęziania z jednej części twojego program do innego

Jeśli opanowałeś instrukcję if, nie powinieneś mieć problemów z przedstawionymi tu pojęciami. Ucząc się instrukcji switch, powinieneś być w stanie z łatwością pisać menu i programy do wprowadzania danych wielokrotnego wyboru.

Instrukcja switch

Instrukcja switch jest czasami nazywana instrukcją wielokrotnego wyboru. Instrukcja switch umożliwia Twojemu programowi wybór z kilku alternatyw. Format instrukcji switch jest nieco dłuższy niż format innych instrukcji, które widziałeś. Oto instrukcja switch:

switch (expression)

{ case (expression1): { one or more C++ statements; }

case (expression2): { one or more C++ statements; }

case (expression3): { one or more C++ statements; }

.

.

.

default: { one or more C++ statements; }

}

Wyrażenie może być wyrażeniem całkowitym, znakiem, literałem lub zmienną. Podwyrażenia (wyrażenie1, wyrażenie2 itd.) Mogą być dowolnymi innymi wyrażeniami całkowitymi, znakami, literałami lub zmiennymi. Liczba wyrażeń wielkości liter po linii przełączania zależy od aplikacji. Jedna lub więcej instrukcji C++ to dowolny blok kodu C++. Jeśli blok ma tylko jedną instrukcję, nawiasy klamrowe nie są potrzebne, ale są zalecane. Domyślna linia jest opcjonalna; większość (ale nie wszystkie) instrukcji przełączników zawiera wartości domyślne. Domyślna linia nie musi być ostatnią linią korpusu przełącznika. Jeśli wyrażenie pasuje do wyrażenia 1, wykonywane są instrukcje po prawej stronie wyrażenia 1. Jeśli wyrażenie pasuje do wyrażenia2, wykonywane są instrukcje po prawej stronie wyrażenia2. Jeśli żadne z wyrażeń nie pasuje do wyrażenia przełączającego, wykonywany jest domyślny blok obserwacji. Wyrażenie wielkości liter nie wymaga nawiasów, ale nawiasy czasem ułatwiają znalezienie wartości.

WSKAZÓWKA: Użyj instrukcji break po każdym bloku instrukcji, aby zapobiec wykonywaniu instrukcji od „upadku” do pozostałych instrukcji

Korzystanie z instrukcji switch jest łatwiejsze niż jej format może to pokazać. Wszędzie, gdzie może iść kombinacja instrukcji if-else-if, zwykle można umieścić jaśniejszą instrukcję switch. Instrukcja switch jest znacznie łatwiejsza do wykonania niż instrukcja if-in-if-in-an-if, jak musieliście wcześniej napisać. Jednak kombinacje instrukcji if i else-if nie są trudne do naśladowania. Gdy test relacyjny, który określa wybór, jest złożony i zawiera wiele && i || operatorów, instrukcja if może być lepszym kandydatem. Instrukcja switch jest preferowana, gdy możliwości wielokrotnego wyboru oparte są na pojedynczym literale, zmiennej lub wyrażeniu.

WSKAZÓWKA: Ułóż zestawienia przypadków w najczęściej wykonywanym lub najrzadziej wykonywanym porządku, aby zwiększyć szybkość programu.

Poniższe przykłady wyjaśniają instrukcję switch. Porównują instrukcję switch z instrukcją if, aby zobaczyć różnicę.

Przykłady

1. Załóżmy, że piszesz program, który nauczy twoje dziecko liczenia. Twój program poprosi dziecko o numer. Następnie emituje sygnał dźwiękowy (dzwoni dzwonek alarmowy komputera) tyle razy, ile potrzeba, aby dopasować ten numer. Poniższy program zakłada, że dziecko naciska klawisz numeryczny od 1 do 5. Ten program używa kombinacji if-else-if, aby osiągnąć tę metodę nauczania liczenia i wydawania sygnałów dźwiękowych. Zidentyfikuj program i dołącz niezbędny plik nagłówka. Chcesz wydać dźwięk i przesunąć kursor do następnego wiersza, więc zdefiniuj zmienną globalną o nazwie BEEP, która to robi. Potrzebujesz zmiennej, aby utrzymać odpowiedź użytkownika, więc uczyń num zmienną całkowitą. Poproś użytkownika o numer. Przypisz numer użytkownika do num. Jeśli num to 1, zadzwoń raz BEEP. Jeśli num to 2, zadzwoń BEEP dwa razy. Jeśli num to 3, zadzwoń BEEP trzy razy. Jeśli num to 4, zadzwoń BEEP cztery razy. Jeśli num to 5, zadzwoń BEEP pięć razy.

```
// Nazwa pliku: C15BEEP1.CPP

// Sygnał dźwiękowy określoną liczbę razy.

#include <iostream.h>

// Zdefiniuj sygnał dźwiękowy, aby zapisać powtarzające się printf () s

// w całym programie.

# zdefiniować BEEP cout << „\ a \ n”

main()

{

int num;

// Request a number from the child

// (you might have to help).

cout << “Please enter a number “;

cin >> num;

// Use multiple if statements to beep.

if (num == 1)
```

```
{ BEEP; }

else if (num == 2)

{ BEEP; BEEP; }

else if (num == 3)

{ BEEP; BEEP; BEEP; }

else if (num == 4)

{ BEEP; BEEP; BEEP; BEEP; }

else if (num == 5)

{ BEEP; BEEP; BEEP; BEEP; BEEP; }

return 0;

}
```

Jeśli dziecko wprowadzi coś innego niż 1 do 5, nie będą emitowane żadne sygnały dźwiękowe. Ten program wykorzystuje dyrektywę #define preprocessor w celu zdefiniowania skrótu do funkcji alarmu. W tym przypadku BEEP jest nieco bardziej czytelny, o ile pamiętasz, że BEEP nie jest poleceniem, ale jest zastępowane przez cout wszędzie tam, gdzie się pojawia. Wadą tego typu programu „jeśli w to" jest jego czytelność. Zanim wciśniesz ciało każdego, jeśli i jeszcze, program jest zbyt daleko w prawo. Nie ma miejsca na więcej niż pięć lub sześć możliwości. Co ważniejsze, ten typ logiki jest trudny do naśladowania. Ponieważ wiąże się to z wyborem wielokrotnego wyboru, instrukcja przełączania jest znacznie lepsza w użyciu, jak widać w następującej, ulepszonej wersji.

```
// Nazwa pliku: C15BEEP2.CPP

// Sygnał dźwiękowy określoną liczbę razy za pomocą przełącznika.

#include <iostream.h>

// Zdefiniuj sygnał dźwiękowy, aby zapisać powtarzające się sygnały dźwiękowe

// w całym programie.

# zdefiniować BEEP cout << „\ a \ n"

main()

{

int num;

// Request from the child (you might have to help).

cout << "Please enter a number ";

cin >> num;

switch (num)

{ case (1): { BEEP;

break; }
```

```
case (2): { BEEP; BEEP;

break; }

case (3): { BEEP; BEEP; BEEP;

break; }

case (4): { BEEP; BEEP; BEEP; BEEP;

break; }

case (5): { BEEP; BEEP; BEEP; BEEP; BEEP;

break; }

}

return 0;

}
```

Ten przykład jest znacznie wyraźniejszy niż poprzedni. Wartość num kontroluje wykonanie - wykonywany jest tylko przypadek pasujący do num. Wcięcie pomaga rozdzielić każdy case. Jeśli dziecko wprowadzi liczbę inną niż 1 do 5, nie będą emitowane żadne sygnały dźwiękowe, ponieważ nie ma wyrażenia, które pasowałoby do żadnej innej wartości, i nie ma domyślnej wielkości liter. Ponieważ dyrektywa preprocesora BEEP jest tak krótka, że możesz umieścić więcej niż jeden w jednym wierszu. Nie jest to jednak wymagane. Blok instrukcji następujących po sprawie może również mieć więcej niż jedną instrukcję. Jeśli więcej niż jedno wyrażenie jest takie samo, wykonywane jest tylko pierwsze wyrażenie.

2. Jeśli dziecko nie wpisze 1, 2, 3, 4 lub 5, nic się nie dzieje w poprzednim programie. Poniżej znajduje się ten sam program zmodyfikowany, aby skorzystać z opcji domyślnej. Domyślny blok instrukcji jest wykonywany, jeśli żaden z poprzednich przypadków nie jest zgodny.

```
// Nazwa pliku: C15BEEP3.CPP

// Sygnał dźwiękowy określoną liczbę razy za pomocą switch.

#include <iostream.h>

// Zdefiniuj sygnał dźwiękowy, aby zapisać powtarzające się sygnały dźwiękowe

// w całym programie.

# zdefiniować BEEP cout << „\ a \ n”

main()

{

int num;

// Request a number from the child (you might have to help).

cout << “Please enter a number “;

cin >> num;
```

```
switch (num)

{ case (1): { BEEP;

break; }

case (2): { BEEP; BEEP;

break; }

case (3): { BEEP; BEEP; BEEP;

break; }

case (4): { BEEP; BEEP; BEEP; BEEP;

break; }

case (5): { BEEP; BEEP; BEEP; BEEP; BEEP;

break; }

default: { cout << “You must enter a number from “ <<

“1 to 5\n”;

cout << “Please run this program again\n”;

break; }

}

return 0;

}
```

Przerwa na końcu domyślnego przypadku może wydawać się zbędna. W końcu żadne inne instrukcje nie są wykonywane przez „wypadnięcie” z przypadku domyślnego. W każdym razie dobrym zwyczajem jest przerywanie domyślnego przypadku. Jeśli przesuniesz domyślną wartość wyższą w przełączniku (nie musi to być ostatnia opcja przełącznika), będziesz bardziej skłonny do przeniesienia z nią przerwy (tam, gdzie jest to potrzebne).

3. Aby pokazać znaczenie używania instrukcji break w każdym wyrażeniu przypadku, oto ten sam program dźwiękowy bez instrukcji break.

```
// Nazwa pliku: C15BEEP4.CPP

// Nieprawidłowo emituje sygnał dźwiękowy za pomocą przełącznika.

#include <iostream.h>

// Zdefiniuj sygnał dźwiękowy printf (), aby zapisać powtarzające się krzyki

// w całym programie.

# zdefiniować BEEP cout << „\ a \ n”

main()

{
```

```
int num;
// Request a number from the child
// (you might have to help).
cout << “Please enter a number “;
cin >> num;
switch (num) // Warning!
{ case (1): { BEEP; } // Without a break, this code
case (2): { BEEP; BEEP; } // falls through to the
case (3): { BEEP; BEEP; BEEP; } // rest of the beeps!
case (4): { BEEP; BEEP; BEEP; BEEP; }
case (5): { BEEP; BEEP; BEEP; BEEP; BEEP; }
default: { cout << “You must enter a number “ <<
“from 1 to 5\n”;
cout << “Please run this program again\n”; }
}
return 0;
}
```

Jeśli użytkownik wpisze 1, program wyemituje 15 sygnałów dźwiękowych! Nie ma przerwy, aby powstrzymać egzekucję przed innymi sprawami. W przeciwieństwie do innych języków programowania, takich jak Pascal, instrukcja przełączania C++ wymaga wstawiania instrukcji break między poszczególnymi przypadkami, jeśli chcesz wykonać tylko jeden przypadek. To niekoniecznie jest wadą. Kompromis polegający na określeniu instrukcji break daje większą kontrolę nad sposobem obsługi określonych przypadków, jak pokazano w następnym przykładzie.

4. Ten program kontroluje wyświetlanie podsumowań sprzedaży na koniec dnia. Najpierw pyta o dzień tygodnia. Jeśli dzień jest od poniedziałku do czwartku, drukowana jest suma dzienna. Jeśli dniem jest piątek, drukowana jest suma tygodniowa i suma dzienna. Jeśli dzień przypada na koniec miesiąca, drukowana jest również miesięczna suma sprzedaży. W prawdziwej aplikacji sumy te pochodziłyby raczej z napędu dyskowego, niż byłyby przypisywane na początku programu. Ponadto zamiast drukowanych pojedynczych danych sprzedaży prawdopodobnie wydrukowano by pełny dzienny, tygodniowy i miesięczny raport z wielu sum sprzedaży. Jesteś na najlepszej drodze, aby dowiedzieć się więcej o rozszerzaniu możliwości swoich programów w C++. Na razie skoncentruj się na instrukcji switch i jej możliwościach. Każdy typ raportu dla wielkości sprzedaży jest obsługiwany przez hierarchię zestawień spraw. Ponieważ dzienna kwota jest ostatnim przypadkiem, jest to jedyny wydrukowany raport, jeśli dzień tygodnia przypada od poniedziałku do czwartku. Jeśli dniem tygodnia jest piątek, drugi przypadek drukuje tygodniową sumę sprzedaży, a następnie spada do sumy dziennej (ponieważ należy także wydrukować sumę dzienną piątku). Jeśli jest koniec miesiąca, pierwszy przypadek kończy się, przechodząc do tygodniowej sumy, a następnie do dziennej sumy sprzedaży. Inne języki, które nie oferują takiej elastyczności, są bardziej ograniczające.

```
// Nazwa pliku: C15SALE.CPP
// Drukuje dzienne, tygodniowe i miesięczne sumy sprzedaży.
#include <iostream.h>
#include <stdio.h>
main()
{
float daily=2343.34; // Later, these figures
float weekly=13432.65; // come from a disk file
float monthly=43468.97; // instead of being assigned
// as they are here.
char ans;
int day; // Day value to trigger correct case.
// Month is assigned 1 through 5 (for Monday through
// Friday) or 6 if it is the end of the month. Assume
// a weekly and a daily prints if it is the end of the
// month, no matter what the day is.
cout << “Is this the end of the month? (Y/N) “;
cin >> ans;
if ((ans==’Y’) || (ans==’y’))
{ day=6; } // Month value
else
{ cout << “What day number, 1 through 5 (for Mon-Fri)” <<
“ is it? “;
cin >> day; }
switch (day)
{ case (6): printf(“The monthly total is %.2f \n”,
monthly);
case (5): printf(“The weekly total is %.2f \n”,
weekly);
default: printf(“The daily total is %.2f \n”, daily);
}
```

```
return 0;

}
```

5. Kolejność instrukcji case nie jest ustalona. Możesz zmienić kolejność instrukcji, aby były bardziej wydajne. Jeśli przez większość czasu wybierany jest tylko jeden lub dwa przypadki, umieść je w górnej części instrukcji switch. Na przykład w poprzednim programie większość raportów firmy jest codziennie, ale dzienna opcja jest trzecia w zestawieniach przypadków. Dzięki zmianie kolejności instrukcji case, tak aby dzienny raport był na górze, możesz przyspieszyć ten program, ponieważ C++ nie musi skanować dwóch wyrażeń case, które rzadko wykonuje.

```
// Nazwa pliku: C15DEPT1.CPP

// Wyświetl wiadomość w zależności od wprowadzonego działu.

#include <iostream.h>

main()

{

char choice;

do // Display menu and ensure that user enters a

// correct option.

{ cout << "\nChoose your department: \n";

cout << "S - Sales \n";

cout << "A - Accounting \n";

cout << "E - Engineering \n";

cout << "P - Payroll \n";

cout << "What is your choice? ";

cin >> choice;

// Convert choice to uppercase (if they

// entered lowercase) with the ASCII table.

if ((choice>=97) && (choice<=122))

{ choice -= 32; } // Subtract enough to make

// uppercase.

} while ((choice!='S')&&(choice!='A')&&

(choice!='E')&&(choice!='P'));

// Put Engineering first because it occurs most often.

switch (choice)

{ case ('E') : { cout << "\n Your meeting is at 2:30";
```

```
break; }

case (‘S’) : { cout << “\n Your meeting is at 8:30”;

break; }

case (‘A’) : { cout << “\n Your meeting is at 10:00”;

break; }

case (‘P’) : { cout << “\n Your meeting has been “ <<

“canceled”;

break; }

}

return 0;
```

Instrukcja Goto

Wczesne języki programowania nie oferowały elastycznych konstrukcji udostępnianych przez C++, takich jak pętle, pętle while i instrukcje switch. Jedynym sposobem zapętlenia i porównania były instrukcje goto. C++ nadal zawiera goto, ale inne konstrukcje są bardziej wydajne, elastyczne i łatwiejsze do naśladowania w programie. Instrukcja goto powoduje, że twój program przeskakuje w inne miejsce, zamiast wykonywać następną instrukcję po kolei. Format instrukcji goto to

Goto instrukcja etykiety

Etykieta instrukcji jest nazywana tak jak zmienne. Etykieta instrukcji nie może mieć takiej samej nazwy jak komenda C++, funkcja C++ lub inna zmienna w programie. Jeśli używasz instrukcji goto, musi istnieć etykieta instrukcji w innym miejscu programu, do którego rozgałęzia się goto. Wykonanie wtedy kontynuuje od instrukcji z etykietą wyciągu. Etykieta instrukcji poprzedza wiersz kodu. Śledź wszystkie etykiety instrukcji dwukropkiem (:), aby C++ rozpoznał je jako etykiety, a nie zmienne. Dotychczas nie widziałeś etykiet instrukcji w programach C++, ponieważ żaden z nich nie potrzebował ich. Etykieta instrukcji jest opcjonalna, chyba że masz instrukcję goto. Poniższe cztery wiersze kodu mają inną etykietę instrukcji. To nie jest program, ale pojedyncze linie, które mogą być zawarte w programie. Zauważ, że etykiety wyciągów znajdują się po lewej stronie.

```
pay: cout << „Umieść czeki w drukarce \ n”;

Ponownie: cin >> name;

EndIt: cout << „To jest całe przetwarzanie. \ n ”;

CALC: kwota = (ogółem / .5) * 1,15;
```

Etykiety instrukcji nie mają na celu zastąpienia komentarzy, chociaż ich nazwy odzwierciedlają poniższy kod. Etykiety wyciągów dają tagom goto znacznik, do którego można przejść. Gdy twój program znajdzie goto, przechodzi do instrukcji oznaczonej etykietą instrukcji. Następnie program kontynuuje wykonywanie sekwencyjne, aż do następnego goto zmienia kolejność ponownie (lub do zakończenia programu).

WSKAZÓWKA: Użyj identyfikujących etykiet linii. Powtarzające się obliczenia zasługują na etykietę taką jak CalcIt, a nie x15z. Mimo że oba są dozwolone, pierwszy jest lepszym wskazaniem celu kodu.

Używaj goto rozważnie

Goto nie jest uważane za dobrą instrukcję programistyczną, gdy jest nadużywana. Istnieje tendencja, szczególnie dla początkujących programistów, do dołączania zbyt wielu instrukcji goto do programu. Kiedy program rozgałęzia się w dowolnym miejscu, trudno jest go śledzić. Niektóre osoby nazywają programy zawierające wiele instrukcji goto „kodem spaghetti". Aby wyeliminować instrukcje goto i pisać programy o lepszej strukturze, użyj innych konstrukcji zapętlających i przełączających opisanych w poprzednich kilku częściach. Goto niekoniecznie jest złym stwierdzeniem - jeśli jest stosowane rozsądnie. Kiedy zaczynasz dzielić swoje programy na mniejsze moduły zwane funkcjami, goto staje się coraz mniej ważne, gdy piszesz coraz więcej funkcji. Na razie zapoznaj się z goto, aby zrozumieć programy, które go używają. Pewnego dnia może być konieczne poprawienie kodu osoby, która używała goto.

Przykłady

1. Poniższy program ma problem, który jest bezpośrednim wynikiem goto, ale nadal jest jedną z najlepszych ilustracji instrukcji goto. Program składa się z nieskończonej pętli (lub pętli nieskończonej). Wykonane zostaną pierwsze trzy linie (po nawiasie otwierającym), a następnie goto w czwartej linii powoduje wykonanie pętli z powrotem do początku i powtórzenie pierwszych trzech linii. Goto kontynuuje to, dopóki nie naciśniesz Ctrl-Break lub poprosisz administratora systemu o anulowanie programu. Zidentyfikuj program i dołącz plik nagłówkowy wejścia / wyjścia. Chcesz wydrukować wiadomość, ale podziel ją na trzy linie. Chcesz, aby komunikat się powtarzał, więc oznacz pierwszą linię, a następnie użyj goto, aby wrócić do tej linii.

```
// Nazwa pliku: C15GOTO1.CPP

// Program pokazujący użycie goto. Ten program się kończy

// tylko wtedy, gdy użytkownik naciśnie Ctrl-Break.

#include <iostream.h>

main()

{

Again: cout << "This message \n";

cout << "\t keeps repeating \n";

cout << "\t\t over and over \n";

goto Again; // Repeat continuously.

return 0;

}
```

Zauważ, że etykieta instrukcji (ponownie w poprzednim przykładzie) ma dwukropek oddzielający ją od reszty wiersza, ale nie ma dwukropka z etykietą w instrukcji goto. Oto wynik uruchomienia tego programu.

```
Ta wiadomość

ciągle się powtarza
```

raz po raz

Ta wiadomość

ciągle się powtarza

raz po raz

Ta wiadomość

ciągle się powtarza

raz po raz

Ta wiadomość

ciągle się powtarza

raz po raz

Ta wiadomość

ciągle się powtarza

raz po raz

Ta wiadomość

ciągle się powtarza

raz po raz

Ta wiadomość

ciągle się powtarza

raz po raz

Ta wiadomość

2. Czasami łatwiej jest odczytać kod programu, pisząc etykiety instrukcji w osobnych wierszach. Pamiętaj, że pisanie możliwych do utrzymania programów jest celem każdego dobrego programisty. Ułatwienie czytania programów jest bardzo ważne podczas ich pisania. Poniższy program jest tym samym programem powtarzalnym, co pokazany w poprzednim przykładzie, z tym wyjątkiem, że etykieta instrukcji jest umieszczona w osobnym wierszu.

```
// Nazwa pliku: C15GOTO2.CPP

// Program pokazujący użycie goto. Ten program się kończy

// tylko wtedy, gdy użytkownik naciśnie Ctrl-Break.

#include <iostream.h>

main()

{

Again:
```

```
cout << “This message \n”;

cout << “\t keeps repeating \n”;

cout << “\t\t over and over \n”;

goto Again; // Repeat continuously

return 0;

}
```

Wiersz następujący po etykiecie instrukcji jest tym, który jest wykonywany jako następny, po przekazaniu kontroli (przez goto) do etykiety. Oczywiście są to głupie przykłady. Prawdopodobnie nie chcesz pisać programów z nieskończonymi pętlami. Goto jest stwierdzeniem, które najlepiej poprzedzić if; w ten sposób goto ostatecznie przestaje się rozgałęziać bez interwencji użytkownika.

3. Poniższy program jest jednym z najgorzej napisanych programów w historii! To uosobienie kodu spaghetti! Jednak staraj się postępować zgodnie z nim i zrozumieć jego wyniki. Rozumiejąc przepływ tego wyjścia, możesz doskonalić swoje zrozumienie goto. Możesz także docenić fakt, że reszta tej książki używa goto tylko wtedy, gdy jest to potrzebne, aby program był bardziej przejrzysty.

```
// Nazwa pliku: C15GOTO3.CPP

// Ten program demonstruje nadużywanie goto.

#include <iostream.h>

main()

{

goto Here;

First:

cout << “A \n”;

goto Final;

There:

cout << “B \n”;

goto First;

Here:

cout << “C \n”;

goto There;

Final:

return 0;

}
```

Na pierwszy rzut oka program wydaje się drukować pierwsze trzy litery alfabetu, ale instrukcje goto sprawiają, że drukują w odwrotnej kolejności, C, B, A. Chociaż program nie jest dobrze

zaprojektowanym programem, pewne wcięcie wiersze bez etykiet instrukcji sprawiają, że jest on trochę bardziej czytelny. Umożliwia to szybkie oddzielenie etykiet instrukcji od pozostałego kodu, jak widać z następującego programu.

```
// Nazwa pliku: C15GOTO4.CPP

// Ten program demonstruje nadużywanie goto.

#include <iostream.h>

main()

{

goto Here;

First:

cout << “A \n”;

goto Final;

There:

cout << “B \n”;

goto First;

Here:

cout << “C \n”;

goto There;

Final:

return 0;

}
```

Lista programu jest nieco łatwiejsza do śledzenia niż poprzednia, mimo że oba robią to samo. Pozostałe programy w tej książce z etykietami instrukcji również używają takiego wcięcia. Z pewnością zdajesz sobie sprawę, że ten wynik jest lepiej wytwarzany przez następujące trzy linie.

```
cout << „C \ n”;

cout << „B \ n”;

cout << „A \ n”;
```

Ostrzeżenie goto warto powtórzyć: używaj goto oszczędnie i tylko wtedy, gdy jego użycie czyni twój program bardziej czytelnym i łatwym w utrzymaniu. Zwykle możesz używać znacznie lepszych poleceń.

Ćwiczenia

1. Napisz program, używając instrukcji switch, która pyta użytkowników o wiek, a następnie drukuje komunikat „Możesz głosować!” jeśli mają 18 lat, „Możesz adoptować!” jeśli mają 21 lat lub „Czy naprawdę jesteś taki młody?” dla każdego wieku.

2. Napisz program sterowany menu dla lokalnej firmy telewizji kablowej. Oto jak oszacować opłaty: Jeśli jesteś w odległości 20 mil poza granicami miasta, płacisz 12,00 $ miesięcznie; 21 do 30 mil poza granicami miasta, płacisz 23,00 $ miesięcznie; Od 31 do 50 mil poza granicami miasta płacisz 34,00 $. Usługa nie przysługuje nikomu poza 50 milami. Monituj użytkowników o menu określające odległość pobytu od granic miasta.

3. Napisz program, który oblicza opłaty parkingowe dla wielopoziomowego garażu. Zapytaj, czy kierowca jest w samochodzie, czy w ciężarówce. Naładuj kierowcę 2,00 $ za pierwszą godzinę, 3,00 $ za drugą i 5,00 $ za ponad 2 godziny. Jeśli jest to ciężarówka, dodaj 1,00 $ do całkowitej opłaty. (Wskazówka: użyj jednego przełącznika i jednego instrukcji if).

4. Zmodyfikuj poprzedni problem z parkowaniem, aby opłata zależała od pory dnia parkowania pojazdu. Jeśli pojazd jest zaparkowany przed 8 rano, nalicz opłaty w ćwiczeniu 3. Jeśli pojazd jest zaparkowany po 8 rano i przed 17 po południu, nalicz dodatkową opłatę za użytkowanie w wysokości 50 centów. Jeśli pojazd zostanie zaparkowany po godzinie 17:00, odliczyć 50 centów od obliczonej ceny. Musisz poprosić użytkowników o godzinę rozpoczęcia w menu, jak następuje.

1. Przed 8 rano

2. Przed 17:00

3. Po 17:00

Podsumowanie

Widziałeś teraz instrukcję switch i jej opcje. Dzięki niemu możesz poprawić czytelność skomplikowanego wyboru „jeśli-inaczej-jeśli”. Zmiana jest szczególnie dobra, gdy możliwych jest kilka wyników, w zależności od wyboru użytkownika. Instrukcja goto powoduje bezwarunkową gałąź i czasami może być trudna do naśladowania. Instrukcja goto nie jest obecnie często używana i prawie zawsze można użyć lepszej konstrukcji. Powinieneś jednak znać jak najwięcej C++, na wypadek, gdybyś musiał pracować nad programami napisanymi przez innych. To kończy rozdział o sterowaniu programem. Następna sekcja przedstawia funkcje napisane przez użytkownika. Do tej pory korzystałeś z wbudowanych funkcji C++, takich jak strcpy() i printf(). Teraz nadszedł czas, aby napisać własny.

Pisanie funkcji C++

Komputery nigdy się nie nudzą. Wykonują te same dane wejściowe, wyjściowe i obliczenia, jakich wymaga Twój program - tak długo, jak chcesz, aby to zrobiły. Możesz skorzystać z ich powtarzalnej natury, patrząc na swoje programy w nowy sposób: jako serię małych procedur, które wykonują się zawsze, gdy ich potrzebujesz, tyle razy, ile potrzebujesz. Ten rozdział podchodzi do tematu nieco inaczej niż poprzednie rozdziały. Koncentruje się na nauce pisania własnych funkcji, które są modułami kodu wykonywanymi i kontrolowanymi przez funkcję main(). Jak dotąd programy składały się z jednej długiej funkcji o nazwie main(). Jak się tutaj nauczysz, głównym celem funkcji main() jest kontrola wykonania innych następujących po nim funkcji .Opisano:

♦ Potrzebę funkcji

♦ Jak śledzić funkcje

♦ Jak pisać funkcje

♦ Jak dzwonić i wracać z funkcji

Podkreślono zastosowanie programowania strukturalnego, zwanego czasem programowaniem modułowym. C++ został zaprojektowany w taki sposób, że programista może pisać programy w kilku modułach, a nie w jednym długim bloku. Dzieląc program na kilka mniejszych procedur (funkcji), możesz izolować problemy, szybciej pisać poprawne programy i tworzyć programy łatwiejsze w utrzymaniu.

Podstawy funkcji

Kiedy zbliżasz się do aplikacji, która musi być zaprogramowana, najlepiej nie siadać przy klawiaturze i zacząć pisać. Najpierw pomyśl najpierw o programie i co powinien zrobić. Jednym z najlepszych sposobów ataku na program jest rozpoczęcie od ogólnego celu, a następnie podziel go na kilka mniejszych zadań. Nigdy nie powinieneś tracić z oczu ogólnego celu, ale pomyśl także o tym, jak poszczególne elementy mogą się ze sobą łączyć, aby osiągnąć ten cel. Kiedy w końcu usiądziesz i zaczniesz kodować problem, kontynuuj myślenie w kategoriach pasujących elementów. Nie podchodź do programu tak, jakby to był jeden ogromny problem; raczej kontynuuj pisanie tych małych kawałków indywidualnie. Nie oznacza to, że musisz pisać osobne programy, aby zrobić wszystko. Możesz przechowywać razem poszczególne elementy całego programu - jeśli wiesz, jak pisać funkcje. Następnie możesz używać tych samych funkcji w wielu różnych programach. Programy C++ nie są podobne do programów BASIC ani FORTRAN. C++ został zaprojektowany, aby zmusić cię do myślenia w modularnym lub podprogramowym, funkcjonalnym stylu. Dobrzy programiści C++ piszą programy, które składają się z wielu małych funkcji, nawet jeśli ich programy wykonują jedną lub więcej z tych funkcji tylko raz. Funkcje te działają razem, aby stworzyć program szybciej i łatwiej niż w przypadku, gdyby program musiał zostać napisany od zera.

WSKAZÓWKA: Zamiast kodować jeden długi program, napisz kilka mniejszych procedur zwanych funkcjami. Jedna z tych funkcji musi być nazwana main(). Funkcja main() jest zawsze uruchamiana jako pierwsza. Nie musi być pierwszy w programie, ale zwykle tak jest.

Rozwiązywanie problemów

Jeśli twój program robi bardzo dużo, podziel go na kilka funkcji. Każda funkcja powinna wykonywać tylko jedno podstawowe zadanie. Na przykład, jeśli piszesz program C++ w celu pobrania listy znaków z klawiatury, alfabetycznie, a następnie wyświetlenia ich na ekranie, możesz - ale nie powinieneś -

napisać wszystkie te instrukcje w jednej dużej funkcji main(), jak pokazuje następujący szkielet C++ (zarys programu):

```
main()

{

// :

// C++ code to retrieve a list of characters.

// :

// C++ code to alphabetize the characters.

// :

// C++ code to print the alphabetized list on-screen.

// :

return 0;

}
```

Ten szkielet nie jest dobrym sposobem na napisanie tego programu. Chociaż możesz wpisać ten program tylko w kilku wierszach kodu, o wiele lepiej jest zacząć rozbijać każdy program na odrębne zadania, aby proces ten stał się dla Ciebie nawykiem. Nie powinieneś używać main() do robienia wszystkiego - w rzeczywistości, użyj main() do robienia bardzo mało, z wyjątkiem wywoływania każdej z funkcji, która faktycznie wykonuje pracę. Lepszym sposobem organizacji tego programu jest napisanie osobnej funkcji dla każdego zadania, które program ma wykonać. Nie oznacza to, że każda funkcja musi mieć tylko jedną linię. Oznacza to raczej, że uczynisz każdą funkcję blokiem konstrukcyjnym, który wykonuje tylko jedno odrębne zadanie w programie. Poniższy zarys programu pokazuje lepszy sposób napisania właśnie opisanego programu:

```
main()

{

getletters(); // Calls a function to retrieve the numbers.

alphabetize(); // Calls a function to alphabetize

// letters.

printletters(); // Calls a function to print letters

// on-screen.

return 0; // Returns to the operating system.

}

getletters()

{

// :
```

```
// C++ code to get a list of characters.
// :
return 0; // Returns to main().
}
alphabetize()
{
// :
// C++ code to alphabetize the characters
// :
return 0; // Returns to main().
}
printletters()
{
// :
// C++ code to print the alphabetized list on-screen
// :
return 0; // Returns to main().
}
```

Zarys programu pokazuje znacznie lepszy sposób pisania tego programu. Pisanie zajmuje więcej czasu, ale jest o wiele bardziej zorganizowane. Jedyną czynnością, którą wykonuje funkcja main(), jest kontrolowanie innych funkcji przez wywoływanie ich w określonej kolejności. Każda osobna funkcja wykonuje swoje instrukcje, a następnie wraca do main(), po czym main() wywołuje następną funkcję, dopóki nie pozostanie więcej funkcji. Funkcja main() następnie zwraca kontrolę komputera na system operacyjny. Nie przejmuj się zbytnio 0, które następuje po instrukcji return. Funkcje C++ zwracają wartości. Jak dotąd funkcje, które widziałeś, zwróciły zero, a ta zwracana wartość została zignorowana.

WSKAZÓWKA: Dobrą zasadą jest to, że funkcja nie powinna mieć więcej niż jednego ekranu. Jeśli jest dłuższy, prawdopodobnie robisz za dużo w jednej funkcji i dlatego powinieneś podzielić go na dwie lub więcej funkcji.

Pierwsza funkcja o nazwie main() jest tym, czego wcześniej używałeś do przechowywania całego programu. Od tego momentu, we wszystkich oprócz najmniejszych programów, main() po prostu kontroluje inne funkcje, które działają. Te wykazy nie są przykładami prawdziwych programów w C++; zamiast tego są szkieletami lub zarysami programów. Na podstawie tych konturów łatwiej jest opracować rzeczywisty pełny program. Zanim przejdziesz do klawiatury, aby napisać program taki jak ten, wiedz, że istnieją cztery odrębne sekcje: główna funkcja wywołująca funkcję main(), funkcja wprowadzania danych z klawiatury, funkcja alfabetyczna i funkcja drukowania. Nigdy nie trać z oczu oryginalnego problemu z programowaniem. (Korzystając z opisanego powyżej podejścia, nigdy tego nie zrobisz!) Ponownie spójrz na procedurę wywołania main() w poprzednim programie. Zauważ, że

możesz rzucić okiem na main() i poznać ogólny program, bez przeszkadzania pozostałym instrukcjom. To dobry przykład programowania strukturalnego, modułowego. Duży problem z programowaniem jest podzielony na odrębne, oddzielne moduły zwane funkcjami, a każda funkcja wykonuje jedno podstawowe zadanie w kilku instrukcjach C++.

Więcej podstaw funkcji

Niewiele powiedziano o funkcjach nazewnictwa i pisania, ale zapewne rozumiesz już wiele celów poprzedniej listy. Funkcje C++ ogólnie przestrzegają następujących zasad:

1. Każda funkcja musi mieć nazwę.

2. Nazwy funkcji są tworzone i przypisywane przez programistę (ciebie!) Zgodnie z tymi samymi regułami, które dotyczą zmiennych nazewnictwa: Mogą zawierać do 32 znaków, muszą zaczynać się od litery i mogą składać się z liter, cyfr, oraz znak podkreślenia (_).

3. Wszystkie nazwy funkcji mają jeden zestaw nawiasów zaraz po nich. Pomaga to odróżnić je od zmiennych. Nawiasy mogą, ale nie muszą, zawierać coś. Jak dotąd wszystkie takie nawiasy w tej książce były puste.

4. Treść każdej funkcji, rozpoczynająca się bezpośrednio po nawiasie zamykającym nazwę funkcji, musi być otoczona nawiasami klamrowymi. Oznacza to, że blok zawierający jedną lub więcej instrukcji stanowi treść każdej funkcji.

WSKAZÓWKA: używaj znaczących nazw funkcji. Funkcja Calc_balance () jest bardziej opisowa niż xy3 ().

Chociaż kontur pokazany na poprzedniej liście jest dobrym przykładem kodu strukturalnego, można go ulepszyć, używając znaku podkreślenia (_) w nazwach funkcji. Czy widzisz, jak get_letters() i print_letters() są znacznie łatwiejsze do odczytania niż getletters() i printletters()?

UWAGA: Pamiętaj, aby używać znaku podkreślenia (_), a nie myślnika (-) podczas nazywania funkcji i zmiennych. Jeśli użyjesz łącznika, C++ generuje mylące komunikaty o błędach.

Poniższa lista pokazuje przykład funkcji C++. Możesz już sporo powiedzieć o tej funkcji. Wiesz na przykład, że nie jest to kompletny program, ponieważ nie ma funkcji main(). (Wszystkie programy muszą mieć funkcję main()). Wiesz również, że nazwa funkcji to calc_it, ponieważ nawiasy następują po tej nazwie. Nawiasy te zawierają coś w sobie. Wiesz również, że ciało funkcji jest zamknięte w bloku nawiasów klamrowych. Wewnątrz tego bloku jest mniejszy blok, ciało pętli while. Wreszcie rozpoznajesz, że instrukcja return jest ostatnim wierszem funkcji

```
calc_it(int n)

{

// Funkcja do wyświetlania kwadratów liczb

int square;

while (square <= 250)

{ square = n * n;

cout << “The square of “ << n <<

“ is “ << square << “\n”;
```

```
n++; } // A block in the function.

return 0;

}
```

WSKAZÓWKA: Nie wszystkie funkcje wymagają instrukcji return dla ostatniego wiersza, ale zaleca się, aby zawsze zawierać jedną, ponieważ pomaga to pokazać zamiar powrotu do funkcji wywołującej w tym momencie. W dalszej części dowiesz się, że returnt jest wymagany w niektórych przypadkach. Na razie rozwijaj nawyk dołączania deklaracji return.

Funkcje połączeń i zwrotów

Czytałeś dużo o „wywoływaniu funkcji" i „przywracaniu kontroli". Chociaż możesz już zrozumieć te wyrażenia z ich kontekstu, prawdopodobnie możesz się ich lepiej nauczyć, ilustrując, co oznacza wywołanie funkcji. Wywołanie funkcji w C++ jest jak objazd na autostradzie. Wyobraź sobie, że podróżujesz „drogą" funkcji podstawowej o nazwie main(), a następnie natrafisz na instrukcję wywołującą funkcję. Musisz tymczasowo opuścić funkcję main() i wykonać wywołaną funkcję. Po zakończeniu tej funkcji (osiągnięciu instrukcji return) sterowanie programem powraca do main(). Innymi słowy, po zakończeniu objazdu powracasz na „główną" trasę i kontynuujesz podróż. Sterowanie jest kontynuowane, ponieważ main() wywołuje inne funkcje.

UWAGA: Zasadniczo podstawowa funkcja kontrolująca wywołania funkcji i ich kolejność nazywa się funkcją wywołującą. Funkcje kontrolowane przez funkcję wywołującą nazywane są funkcjami wywoływanymi.

Kompletny program C++ z funkcjami wyjaśni tę koncepcję. Poniższy program wyświetla kilka komunikatów na ekranie. Każda wyświetlona wiadomość jest określona przez kolejność funkcji. Zanim zaczniesz się zbytnio martwić o to, co robi ten program, poświęć trochę czasu na przestudiowanie jego struktury. Zauważ, że w programie są zdefiniowane trzy funkcje: main(), next_fun() i third_fun(). Czwarta funkcja jest również używana, ale jest to wbudowana funkcja printf() C++. Trzy zdefiniowane funkcje pojawiają się sekwencyjnie. Ciało każdego z nich jest zamknięte w nawiasy klamrowe, a na końcu każdego z nich znajduje się instrukcja return. Jak zobaczysz z programu, po dyrektywie #include jest coś nowego. Pierwszy wiersz każdej funkcji wywoływanej przez main() znajduje się tutaj, a także pojawia się nad rzeczywistą funkcją. C++ wymaga tych prototypów. Na razie po prostu je zignoruj i przestudiuj ogólny format programów wielofunkcyjnych.

```
// C16FUN1.CPP

// Poniższy program ilustruje wywołania funkcji.

#include <stdio.h>

next_fun (); // Prototypy.

third_fun ();

main() // main() jest zawsze pierwszą uruchomioną funkcją C++.

{

printf("First function called main() \n");

next_fun(); // Second function is called here.

third_fun(); // This function is called here.
```

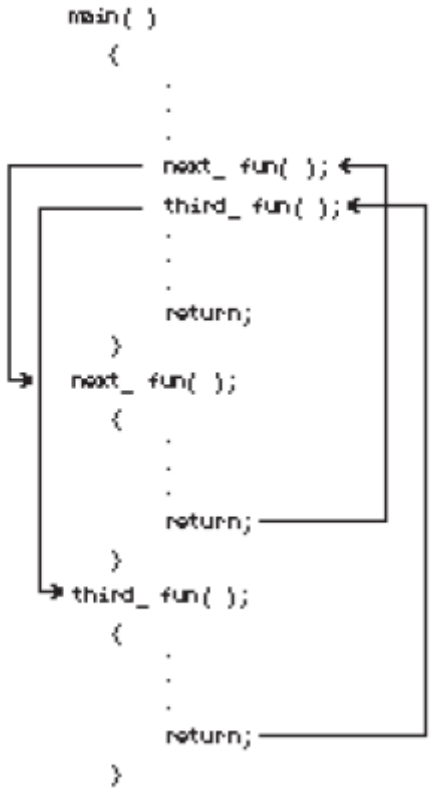
```
printf(“main() is completed \n”); // All control
// returns here.
return 0; // Control is returned to
//the operating system.
} // This brace concludes main().
next_fun() // Second function.
// Parentheses always required.
{
printf(“Inside next_fun() \n”); // No variables are
// defined in the program.
return 0; // Control is now returned to main().
}
third_fun() // Last function in the program.
{
printf(“Inside third_fun() \n”);
return 0; // Always return from all functions.
}
```

Dane wyjściowe tego programu są następujące:

```
Pierwsza funkcja o nazwie main()
Inside next_fun ()
Inside Third_fun ()
main() jest zakończone
```

Rysunek pokazuje śledzenie wykonania tego programu. Zauważ, że main() kontroluje, która z pozostałych funkcji jest wywoływana, a także kolejność wywoływania. Sterowanie zawsze powraca do funkcji wywołującej po zakończeniu wywoływanej funkcji. Aby wywołać funkcję, po prostu wpisz jej nazwę - w tym nawiasy - i podążaj za nią średnikiem. Pamiętaj, że średniki podążają za wszystkimi instrukcjami wykonywalnymi w C++, a wywołanie funkcji (czasami nazywane wywołaniem funkcji) jest instrukcją wykonywalną. Wykonywanie jest wywoływanym kodem funkcji. Każda funkcja może wywoływać dowolną inną funkcję. W poprzednim programie main() jest jedyną funkcją wywołującą inne funkcje. Teraz możesz powiedzieć, że następująca instrukcja jest wywołaniem funkcji:

```
print_total();
```



Ponieważ print_total nie jest poleceniem C++ ani nazwą wbudowanej funkcji, musi to być zmienna lub nazwa zapisanej funkcji. Tylko nazwy funkcji kończą się nawiasami, więc musi to być wywołanie funkcji lub początek kodu funkcji. Z dwóch ostatnich możliwości musi to być wywołanie funkcji, ponieważ kończy się średnikiem. Gdyby nie miał średnika, musiałby być początkiem definicji funkcji. Kiedy definiujesz funkcję (wpisując nazwę funkcji i jej późniejszy kod w nawiasach klamrowych), nigdy nie podążaj za nazwą średnikiem. Zwróć uwagę w poprzednim programie, że main(), next_fun () i third_fun () nie mają średników, gdy pojawiają się w treści programu. Średnik występuje po nazwie tylko w main(), gdzie wywoływane są te funkcje.

UWAGA: Nigdy nie definiuj funkcji w innej funkcji. Cały kod funkcji musi być wymieniony kolejno w całym programie. Nawias zamykający funkcji musi pojawić się, zanim będzie można wyświetlić kod innej funkcji.

Przykłady

1. Załóżmy, że piszesz program, który wykonuje następujące czynności. Po pierwsze, pyta użytkowników o ich działy. Następnie, jeśli prowadzą księgowość, otrzymują raport działu księgowości. Jeśli są inżynierami, otrzymują raport działu technicznego. Wreszcie, jeśli zajmują się marketingiem, otrzymują raport działu marketingu. Następuje szkielet takiego programu. Kod funkcji main() jest pokazany w całości, ale pokazany jest tylko szkielet pozostałych funkcji. Instrukcja switch jest idealną instrukcją wywołującą funkcję dla takich wyborów wielokrotnego wyboru.

```
// Szkielet programu raportów departamentalnych.

#include <iostream.h>

main()

{
```

```
int choice;
do
{ cout << “Choose your department from the “ <<
“following list\n”;
cout << “\t1. Accounting \n”;
cout << “\t2. Engineering \n”;
cout << “\t3. Marketing \n”;
cout << “What is your choice? “;
cin >> choice;
} while ((choice<1) || (choice>3)); // Ensure 1, 2,
// or 3 is chosen.
switch choice
{ case(1): { acct_report(); // Call accounting function.
break; } // Don’t fall through.
case(2): { eng_report(); // Call engineering function.
break; }
case(3): { mtg_report(); // Call marketing function.
break; }
}
return 0; // Program returns to the operating
// system when finished.
}
acct_report()
{
// :
// Accounting report code goes here.
// :
return 0;
}
eng_report()
{
```

```
// :
// Engineering report code goes here.
// :
return 0;
}
mtg_report()
{
// :
// Marketing report code goes here.
// :
return 0;
}
```

Treści instrukcji switch zwykle zawierają wywołania funkcji. Można powiedzieć, że te instrukcje przypadków wykonują funkcje. Na przykład acct_report(); (który jest pierwszym wierszem pierwszego przypadku) nie jest nazwą zmiennej ani poleceniem C++. Jest to nazwa funkcji zdefiniowana później w programie. Jeśli użytkownicy wprowadzą 1 w menu, uruchomiona zostanie funkcja o nazwie acct_report(). Po zakończeniu kontrola wraca do treści pierwszej sprawy, a jej instrukcja break powoduje zakończenie instrukcji switch. Funkcja main() wraca do DOS (lub do zintegrowanego środowiska C++, jeśli go używasz), gdy wykonywana jest jego instrukcja return.

2. W poprzednim przykładzie procedura main() nie jest bardzo modułowa. Wyświetla menu, ale nie w osobnej funkcji, tak jak powinno. Pamiętaj, że main() robi bardzo niewiele poza kontrolą innych funkcji, które wykonują całą pracę. Oto przepis tego przykładowego programu z czwartą funkcją wyświetlania menu na ekranie. To jest naprawdę modułowy przykład, w którym każda funkcja wykonuje jedno zadanie. Ponownie, ostatnie trzy funkcje są pokazane tylko jako kod szkieletowy, ponieważ celem tutaj jest po prostu zilustrowanie wywoływania i powrotu funkcji.

```
// Drugi szkielet wydziałowego programu raportów.
#include <iostream.h>
main()
{
int choice;
do
{ menu_print(); // Call function to print the menu.
cin >> choice;
} while ((choice<1) || (choice>3)); // Ensure 1, 2,
// or 3 is chosen.
```

```
switch choice
{ case(1): { acct_report(); // Call accounting function.
break; } // Don’t fall through.
case(2): { eng_report(); // Call engineering function.
break; }
case(3): { mtg_report(); // Call marketing function.
break; }
}
return 0; // Program returns to the operating system
// when finished.
}
menu_print()
{
cout << “Choose your department from the following"
"list\n”;
cout << “\t1. Accounting \n”;
cout << “\t2. Engineering \n”;
cout << “\t3. Marketing \n”;
cout << “What is your choice? “;
return 0; // Return to main().
}
acct_report()
{
// :
// Accounting report code goes here.
// :
return 0;
}
eng_report()
{
// :
```

```
// Engineering report code goes here.
// :
return 0;
}
mtg_report()
{
// :
// Marketing report code goes here.
// :
return 0;
}
```

Funkcja drukowania menu nie musi być zgodna z main(). Ponieważ jednak jest to pierwsza wywoływana funkcja, najlepiej ją tam zdefiniować.

3. Kluczem jest czytelność, więc programy podzielone na osobne funkcje dają lepszy napisany kod. Możesz pisać i testować każdą funkcję, pojedynczo. Po napisaniu ogólnego zarysu programu możesz wypisać kilka wywołań funkcji w main() i zdefiniować ich szkielety po main(). Ciało każdej funkcji początkowo powinno składać się z jednej instrukcji return, więc program kompiluje się w formacie szkieletu. Po zakończeniu każdej funkcji możesz skompilować i przetestować program. Umożliwia to szybsze opracowywanie dokładniejszych programów. Oddzielne funkcje umożliwiają innym (którzy mogą później zmodyfikować Twój program) łatwe znalezienie konkretnej funkcji bez wpływu na resztę programu. Innym użytecznym nawykiem, popularnym wśród wielu programistów C++, jest oddzielanie funkcji od siebie za pomocą komentarza składającego się z linii gwiazdek (*) lub myślników (-). Ułatwia to, szczególnie w dłuższych programach, sprawdzenie, gdzie funkcja zaczyna się i kończy. Poniżej znajduje się kolejna lista poprzedniego programu, ale teraz z czterema funkcjami wyraźniej oddzielonymi tego typu linią komentarza.

```
// Trzeci szkielet wydziałowego programu raportowego.
#include <iostream.h>
main()
{
int choice;
do
{ menu_print(); // Call function to print the menu.
cin >> choice;
} while ((choice<1) || (choice>3)); // Ensure 1, 2,
// or 3 is chosen.
```

```
switch choice
{ case(1): { acct_report(); // Call accounting function.
break; } // Don't fall through.
case(2): { eng_report(); // Call engineering function.
break; }
case(3): { mtg_report(); // Call marketing function.
break; }
}
return 0; // Program returns to the operating system
// when finished.
}
//*********************************************************
menu_print()
{
cout << "Choose your department from the following"
"list\n";
cout << "\t1. Accounting \n";
cout << "\t2. Engineering \n";
cout << "\t3. Marketing \n";
cout << "What is your choice? ";
return 0; // Return to main().
}
//*********************************************************
acct_report()
{
// :
// Accounting report code goes here.
// :
return 0;
}
//*********************************************************
```

```
eng_report()

{

// :

// Engineering report code goes here.

// :

return 0;

}

//*******************************************************

mtg_report()

{

// :

// Marketing report code goes here.

// :

return 0;

}
```

Z powodu ograniczeń miejsca nie wszystkie wykazy programów w tej książce rozdzielają funkcje w ten sposób. Może się jednak okazać, że łatwiej jest śledzić listę, jeśli umieścisz te oddzielające komentarze między funkcjami.

4. Możesz wykonać funkcję więcej niż raz, po prostu wywołując ją z więcej niż jednego miejsca w programie. Jeśli umieścisz wywołanie funkcji w ciele pętli, funkcja będzie wykonywana wielokrotnie, aż pętla się zakończy. Poniższy program wypisuje komunikat C++ is Fun! kilka razy na ekranie - do przodu i do tyłu - za pomocą funkcji. Zauważ, że main() nie wykonuje każdego wywołania funkcji. Druga funkcja, name_print (), wywołuje funkcję o nazwie reverse_print (). Śledź wykonanie tego programu.

```
// Nazwa pliku: C16FUN2.CPP

// Drukuje C++ jest fajny! kilka razy na ekranie.

#include <iostream.h>

name_print();

reverse_print();

one_per_line();

main()

{

int ctr; // To control loops

for (ctr=1; ctr<=5; ctr++)
```

```
{ name_print(); } // Calls function five times.
one_per_line(); // Calls the program's last
// function once.
return 0;
}
//*********************************************************
name_print()
{
// Prints C++ is Fun! across a line, separated by tabs.
cout << "C++ is Fun!\tC++ is Fun!\tC++ is Fun!
\tC++ is Fun!\n";
cout << "C++ i s F u n !\tC++ i s F u n ! " <<
"\tC++ i s F u n !\n";
reverse_print(); // Call next function from here.
return 0; // Returns to main().
}
//*********************************************************
reverse_print()
{
// Prints several C++ is Fun! messages,
// in reverse, separated by tabs.
cout << "!nuF si ++C\t!nuF si ++C\t!nuF si ++C\t\n";
return 0; // Returns to name_print().
}
//*********************************************************
one_per_line()
{
// Prints C++ is Fun! down the screen.
cout << "C++\n \ni\ns\n \nF\nu\nn\n!\n";
return 0; // Returns to main()
}
```

Oto wynik tego programu:

C++ is Fun! C++ is Fun! C++ is Fun! C++ is Fun!

C++ i s F u n ! C++ i s F u n ! C++ i s F u n !

!nuF si ++C !nuF si ++C !nuF si ++C

C++ is Fun! C++ is Fun! C++ is Fun! C++ is Fun!

C++ i s F u n ! C++ i s F u n ! C++ i s F u n !

!nuF si ++C !nuF si ++C !nuF si ++C

C++ is Fun! C++ is Fun! C++ is Fun! C++ is Fun!

C++ i s F u n ! C++ i s F u n ! C++ i s F u n !

!nuF si ++C !nuF si ++C !nuF si ++C

C++ is Fun! C++ is Fun! C++ is Fun! C++ is Fun!

C++ i s F u n ! C++ i s F u n ! C++ i s F u n !

!nuF si ++C !nuF si ++C !nuF si ++C

C++ is Fun! C++ is Fun! C++ is Fun! C++ is Fun!

C++ i s F u n ! C++ i s F u n ! C++ i s F u n !

!nuF si ++C !nuF si ++C !nuF si ++C

C++

i

s

F

u

n

!

Podsumowanie

Jesteś teraz narażony na naprawdę ustrukturyzowane programy. Zamiast pisać długi program, możesz podzielić go na osobne funkcje. Ta metoda izoluje twoje procedury, więc otaczający kod nie zaśmieca twojego programu i nie wprowadza zamieszania. Funkcje wprowadzają tylko nieco większą złożoność, polegającą na sposobie rozpoznawania wartości zmiennych przez funkcje programu. Następny rozdział (Rozdział 17, „Zakres zmiennych") pokazuje, jak zmienne są obsługiwane między funkcjami, i pomaga wzmocnić umiejętności programowania strukturalnego

Zakres zmiennej

Pojęcie zakresu zmiennego jest najważniejsze przy pisaniu funkcji. Zakres zmiennych określa, które funkcje rozpoznają określone zmienne. Jeśli funkcja rozpoznaje zmienną, zmienna jest widoczna dla tej funkcji. Zakres zmiennych chroni zmienne w jednej funkcji przed innymi funkcjami, które mogą je zastąpić. Jeśli funkcja nie potrzebuje dostępu do zmiennej, funkcja ta nie powinna widzieć ani zmieniać tej zmiennej. Innymi słowy, zmienna nie powinna być „widoczna" dla tej konkretnej funkcji.

Zapoznasz się z

♦ Zmiennymi globalnymi i lokalnymi

♦ Przekazywaniem argumentów

♦ Zmiennymi automatycznymi i statycznymi

♦ Przekazywanie parametrów

Poprzednia część wprowadziła koncepcję używania innej funkcji dla każdego zadania. Ta koncepcja jest o wiele bardziej przydatna, gdy poznasz lokalny i globalny zakres zmiennych.

Zmienne globalne a lokalne

Jeśli programowałeś tylko w języku BASIC, koncepcja zmiennych lokalnych i globalnych może być dla Ciebie nowa. W wielu interpretowanych wersjach BASIC wszystkie zmienne są globalne, co oznacza, że cały program zna każdą zmienną i ma możliwość zmiany dowolnej z nich. Jeśli użyjesz zmiennej o nazwie SPRZEDAŻ u góry programu, nawet ostatnia linia w programie może używać SPRZEDAŻY. (Jeśli nie znasz języka BASIC, nie rozpaczaj – będzie to jeden nawyk, który musisz przełamać!) Zmienne globalne mogą być niebezpieczne. Części programu mogą przypadkowo zmienić zmienną, której nie należy zmieniać. Załóżmy na przykład, że piszesz program, który śledzi zapasy w sklepie spożywczym. Możesz śledzić wartości procentowe sprzedaży, rabaty, ceny detaliczne, ceny hurtowe, ceny produktów, ceny produktów mlecznych, ceny dostawy, zmiany cen, wartości procentowe podatku od sprzedaży, marże wakacyjne, obniżki po okresie świątecznym i tak dalej. Ogromna liczba cen w takim systemie jest myląca. Pisząc program do śledzenia każdej ceny, łatwo byłoby błędnie wywołać zarówno ceny produktów mlecznych d_price, jak i dostarczone ceny d_price. Albo C++ nie pozwoli ci tego zrobić (nie możesz zdefiniować tej samej zmiennej dwa razy) lub nadpiszesz wartość używaną do czegoś innego. Cokolwiek się stanie, śledzenie wszystkich tych różnych - ale podobnie nazwanych - cen powoduje, że program ten jest mylący z pisaniem. Zmienne globalne mogą być niebezpieczne, ponieważ kod może przypadkowo zastąpić zmienną zainicjowaną gdzie indziej w programie. Lepiej jest uczynić każdą zmienną lokalną w swoich programach. Wtedy mogą to zrobić tylko funkcje, które powinny móc zmieniać zmienne. Zmienne lokalne można zobaczyć (i zmienić) tylko z funkcji, w której są zdefiniowane. Dlatego jeśli funkcja definiuje zmienną jako lokalną, zakres tej zmiennej jest chroniony. Zmienna nie może być używana, zmieniana ani usuwana przez żadną inną funkcję bez specjalnego programowania, o którym wkrótce się dowiesz. Jeśli używasz tylko jednej funkcji, main(), pojęcie lokalne i globalne jest akademickie. Jednak wiesz z rozdziału 16, „Pisanie funkcji C++", że programy jednofunkcyjne nie są zalecane. Najlepiej jest pisać modułowe, ustrukturyzowane programy złożone z wielu mniejszych funkcji. Dlatego powinieneś wiedzieć, jak zdefiniować zmienne jako lokalne tylko dla tych funkcji, które ich używają.

Definiowanie zakresu zmiennej

Kiedy po raz pierwszy poznałeś zmienne, nauczyłeś się, że możesz definiować zmienne w dwóch miejscach:

♦ Zanim zostaną użyte w funkcji

♦ Przed nazwą funkcji, na przykład main()

Wszystkie przykłady deklarowały zmienne za pomocą pierwszej metody. Musisz jeszcze zobaczyć przykład drugiej metody. Ponieważ większość tych programów składa się całkowicie z jednej funkcji main(), nie było powodu, aby różnicować te dwie metody. Dopiero po rozpoczęciu korzystania z kilku funkcji w jednym programie te dwie metody definicji zmiennych stają się krytyczne. Ważne są następujące reguły, specyficzne dla zmiennych lokalnych i globalnych:

♦ Zmienna jest lokalna tylko wtedy, gdy zostanie zdefiniowana po nawiasie otwierającym bloku, zwykle na górze funkcji.

♦ Zmienna jest globalna tylko wtedy, gdy zostanie zdefiniowana poza funkcją.

Wszystkie zmienne, które widziałeś do tej pory, były lokalne. Wszystkie zostały zdefiniowane natychmiast po nawiasach otwierających main(). Dlatego są one lokalne dla main() i tylko main() może ich używać. Inne funkcje nie mają pojęcia, że te zmienne nawet istnieją, ponieważ należą tylko do main(). Kiedy funkcja (lub blok) kończy się, wszystkie lokalne zmienne są niszczone.

WSKAZÓWKA: Wszystkie zmienne lokalne znikają (tracą swoją definicję) po zakończeniu bloku.

Zmienne globalne są widoczne („znane”) od punktu definicji do końca programu. Jeśli zdefiniujesz zmienną globalną, dowolny wiersz w pozostałej części programu - bez względu na to, ile funkcji i linii kodu za nią podąży - będzie mógł używać tej zmiennej globalnej.

Przykłady

1. Poniższa sekcja kodu definiuje dwie zmienne lokalne, i i j.

```
main()

{

int i, j; // Local because they’re

// defined after the brace.

// Rest of main() goes here.

}
```

Te zmienne są widoczne dla main(), a nie dla żadnej innej funkcji, która może następować lub być wywoływana przez main().

2. Następująca sekcja kodu definiuje dwie zmienne globalne, g i h.

```
#include <iostream.h>

int g, h; // Globalne, ponieważ są

// zdefiniowane przed funkcją.

main()
```

```
{
// main()'s code goes here.
}
```

Nie ma znaczenia, czy Twoje linie #include idą przed czy po deklaracjach zmiennych globalnych.

3. Zmienne globalne mogą pojawić się przed dowolną funkcją. W poniższym programie main() nie używa zmiennych. Jednak obie funkcje po main() mogą wykorzystywać sprzedaż i zysk, ponieważ zmienne te są globalne.

```
// Nazwa pliku: C17GLO.CPP
// Program zawierający dwie zmienne globalne.
#include <iostream.h>
do_fun ();
third_fun (); // Prototyp omówiony później.
{
cout << "No variables defined in main() \n\n";
do_fun(); // Call the first function.
return 0;
}
float sales, profit; // Two global variables.
do_fun()
{
sales = 20000.00; // This variable is visible
// from this point down.
profit = 5000.00; // As is this one. They are
// both global.
cout << "The sales in the second function are " <<
sales << "\n";
cout << "The profit in the second function is " <<
profit << "\n\n";
third_fun(); // Call the third function to
// show that globals are visible.
return 0;
}
```

```
third_fun()

{

cout << “In the third function: \n”;

cout << “The sales in the third function are “ <<

sales << “\n”;

cout << “The profit in the third function is “ <<

profit << “\n”;

// If sales and profit were local, they would not be

// visible by more than one function.

return 0;

}
```

Zauważ, że funkcja main() nigdy nie może wykorzystywać sales i profit, ponieważ nie są widoczne dla main() - mimo że są globalne. Pamiętaj, że zmienne globalne są widoczne tylko z punktu definicji w dół w programie. Instrukcje pojawiające się przed definicjami zmiennych globalnych nie mogą używać tych zmiennych. Oto wynik uruchomienia tego programu

```
No variables defined in main()

The sales in the second function are 20000

The profit in the second function is 5000

In the third function:

The sales in the third function are 20000

The profit in the third function is 5000
```

WSKAZÓWKA: Zadeklaruj wszystkie zmienne globalne u góry programów. Nawet jeśli możesz je zdefiniować później (pomiędzy dowolnymi dwiema funkcjami), możesz je znaleźć szybciej, jeśli zadeklarujesz je na górze.

4. Poniższy program używa zmiennych lokalnych i globalnych. Teraz powinno być dla ciebie oczywiste, że j i p są lokalne, a i iz są globalne.

```
// Nazwa pliku: C17GLLO.CPP

// Program ze zmiennymi lokalnymi i globalnymi.

// Zmienne lokalne Zmienne globalne

// j, p i, z

#include <iostream.h>

pr_again (); // Prototyp

int i = 0; // Zmienna globalna, ponieważ jest
```

```
// zdefiniowano poza main().

main()

{

float p ; // Local to main() only.

p = 9.0; // Puts value in global variable.

cout << i << “, “ << p << “\n”; // Prints global i

// and local p.

pr_again(); // Calls next function.

return 0; // Returns to DOS.

}

float z = 9.0; // Global variable because it’s

// defined before a function.

pr_again()

{

int j = 5; // Local to only pr_again().

cout << j << “, “ << z; // This can’t print p!.

cout << “, “ << i << “\n”;

return 0; // Return to main().

}
```

Mimo że j jest zdefiniowane w funkcji wywoływanej przez main(), main() nie może używać j, ponieważ j jest lokalny dla pr_again (). Po zakończeniu pr_again () j nie jest już zdefiniowane. Zmienna z jest globalna od punktu definicji w dół. Dlatego main() nie może drukować z. Ponadto funkcja pr_again () nie może wypisać p, ponieważ p jest lokalne tylko dla main(). Zanim przejdziesz dalej, upewnij się, że możesz rozpoznać zmienne lokalne i globalne. Małe studium tutaj ułatwia zrozumienie reszty tego rozdziału.

5. Dwie zmienne mogą mieć tę samą nazwę, o ile są lokalne dla dwóch różnych funkcji. Są to odrębne zmienne, nawet jeśli mają takie same nazwy. Poniższy krótki program wykorzystuje dwie zmienne, obie o nazwie wiek. Mają dwie różne wartości i są uważane za dwie różne zmienne. Pierwszy age jest lokalny dla main(), a drugi wiek jest lokalny dla get_age ().

```
// Nazwa pliku: C17LOC1.CPP

// Dwie różne zmienne lokalne o tej samej nazwie.

#include <iostream.h>

get_age (); // Prototyp

main()
```

```
{
int age;
cout << “What is your age? “;
cin >> age;
get_age(); // Call the second function.
cout << “main()’s age is still “ << age << “\n”;
return 0;
}
get_age()
{
int age; // A different age. This one
// is local to get_age().
cout << “What is your age again? “;
cin >> age;
return 0;
}
```

Wyjście tego programu następuje. Uważnie przestudiuj ten wynik. Zauważ, że ostatnie cout main() nie drukuje nowo zmienionego wieku. Drukuje raczej wiek znany z funkcji main() - age lokalny dla funkcji main(). Mimo że mają takie same nazwy, wiek main() nie ma z tym nic wspólnego wiek get_age (). Równie dobrze mogą mieć dwie różne nazwy zmiennych.

```
What is your age? 28
What is your age again? 56
main()’s age is still 28
```

Należy zachować ostrożność podczas nazywania zmiennych. Posiadanie dwóch zmiennych o tej samej nazwie jest mylące. Łatwo byłoby pomylić się, zmieniając później ten program. Jeśli te zmienne naprawdę muszą być oddzielne, nazwij je inaczej, na przykład old_age i new_age lub ag1 i ag2. Pomaga to zapamiętać, że są różne.

6. Kilka razy nakładanie się lokalnych nazw zmiennych nie wprowadza zamieszania, ale należy zachować ostrożność, aby nie przesadzić. Programiści często używają tej samej nazwy zmiennej, co zmienna licznika w pętli for. Na przykład dwie zmienne lokalne w następującym programie mają tę samą nazwę.

```
// Nazwa pliku: C17LOC2.CPP
// Używanie dwóch zmiennych lokalnych o tej samej nazwie
jako zmienne zliczające.
```

```
#include <iostream.h>

do_fun (); // Prototyp

main()

{

int ctr; // Loop counter.

for (ctr=0; ctr<=10; ctr++)

{ cout << “main()’s ctr is “ << ctr << “\n”; }

do_fun(); // Call second function.

return 0;

}

do_fun()

{

int ctr;

for (ctr=10; ctr>=0; ctr--)

{ cout << “do_fun()’s ctr is “ << ctr << “\n”; }

return 0; // Return to main().

}
```

Chociaż jest to nonsensowny program, który po prostu drukuje od 0 do 10, a następnie drukuje od 10 do 0, pokazuje, że użycie ctr dla obu nazw funkcji nie stanowi problemu. Zmienne te nie przechowują ważnych danych, które muszą zostać przetworzone; są raczej dla zmiennych zliczających pętle. Wywołanie ich obu ctr prowadzi do niewielkiego zamieszania, ponieważ ich użycie ogranicza się do kontrolowania pętli. Ponieważ pętla for inicjuje i zwiększa zmienne, jedna funkcja nigdy nie polega na drugiej funkcji ctr , aby cokolwiek zrobić.

7. Uważaj na tworzenie zmiennych lokalnych o tej samej nazwie w tej samej funkcji. Jeśli zdefiniujesz zmienną lokalną na początku funkcji, a następnie zdefiniujesz inną zmienną lokalną o tej samej nazwie w nowym bloku, C++ używa tylko najbardziej wewnętrznej zmiennej, dopóki jej blok się nie skończy. Poniższy przykład pomaga wyjaśnić ten mylący problem. Program zawiera jedną funkcję z trzema zmiennymi lokalnymi. Sprawdź, czy możesz znaleźć te trzy zmienne.

```
// Nazwa pliku: C17MULI.CPP

// Program z wieloma zmiennymi lokalnymi o nazwie i.

#include <iostream.h>

main()

{

int i; // Outer i
```

```
i = 10;

{ int i; // New block's i

i = 20; // Outer i still holds a 10.

cout << i << " " << i << "\n"; // Prints 20 20.

{ int i; // Another new block and local variable.

i = 30; // Innermost i only.

cout << i << " " << i <<

" " << i << "\n"; // Prints 30 30 30.

} // Innermost i is now gone forever.

} // Second i is gone forever (its block ended).

cout << i << " " << i << " " <<

i << "\n"; // Prints 10 10 10.

return 0;

} // main() ends and so do its variables.
```

Wszystkie zmienne lokalne są lokalne dla bloku, w którym zostały zdefiniowane. Ten program ma trzy bloki, każdy zagnieżdżony w innym. Ponieważ możesz zdefiniować zmienne lokalne bezpośrednio po nawiasie otwierającym blok, w tym programie są trzy różne zmienne i. Lokalne i znika całkowicie, gdy kończy się jego blok (po osiągnięciu klamry zamykającej). C++ zawsze drukuje zmienną, którą interpretuje jako najbardziej lokalną - tę, która znajduje się w najbardziej wewnętrznym bloku.

Oszczędnie używaj zmiennych globalnych

Być może zadajesz sobie pytanie: „Dlaczego muszę rozumieć zmienne globalne i lokalne?" W tym momencie jest to zrozumiałe pytanie, szczególnie jeśli programujesz głównie w języku BASIC. Oto sedno: Zmienne globalne mogą być niebezpieczne. Kod może przypadkowo zastąpić zmienną, która została zainicjowana w innym miejscu w programie. Lepiej, aby każda zmienna w twoim programie była lokalna dla funkcji, która ma do niej dostęp. Przeczytaj jeszcze raz ostatnie zdanie. Nawet jeśli wiesz już, jak tworzyć zmienne globalne, powinieneś tego unikać! Staraj się nigdy nie używać innej zmiennej globalnej. Używanie zmiennych globalnych może wydawać się łatwiejsze, gdy piszesz programy posiadające więcej niż jedną funkcję: Jeśli sprawisz, że każda zmienna używana przez każdą funkcję będzie globalna, nigdy nie będziesz musiał się martwić, czy jest ona widoczna dla dowolnej funkcji. Z drugiej strony funkcja może przypadkowo wyznaczyć zmienną globalną, jeśli nie było to twoją intencją. Jeśli utrzymujesz zmienne lokalne tylko dla funkcji, które ich potrzebują, chronisz ich wartości, a także utrzymujesz swoje programy w pełni modułowe.

Potrzeba przekazywania zmiennych

Właśnie poznałeś różnicę między zmiennymi lokalnymi i globalnymi. Zobaczyłeś, że ustawiając zmienne lokalne, chronisz ich wartości, ponieważ funkcja, która widzi zmienną, jest jedyną, która może ją modyfikować. Co jednak robisz, jeśli masz zmienną lokalną, której chcesz używać w dwóch lub więcej funkcjach? Innymi słowy, może być konieczne dodanie zmiennej z klawiatury w jednej funkcji i wydrukowanie w innej funkcji. Jeśli zmienna jest lokalna tylko dla pierwszej funkcji, w jaki sposób druga

może uzyskać do niej dostęp? Masz dwa rozwiązania, jeśli więcej niż jedna funkcja musi współdzielić zmienną. Po pierwsze, możesz zadeklarować zmienną globalnie. To nie jest dobry pomysł, ponieważ chcesz, aby tylko te dwie funkcje miały dostęp do zmiennej, ale wszystkie funkcje mają do niej dostęp, gdy jest ona globalna. Inną alternatywą - i zdecydowanie lepszą - jest przekazywanie zmiennej lokalnej z jednej funkcji do drugiej. Ma to dużą zaletę: zmienna jest znana tylko tym dwóm funkcjom. Reszta programu nadal nie ma do niego dostępu.

UWAGA: Nigdy nie przekazuj zmiennej globalnej do funkcji. Zresztą nie ma powodu, aby przekazywać zmienne globalne, ponieważ są one już widoczne dla wszystkich funkcji.

Gdy przekazujesz zmienną lokalną z jednej funkcji do drugiej, przekazujesz argument z pierwszej funkcji do następnej. Możesz przekazać więcej niż jeden argument (zmienną) na raz, jeśli chcesz, aby kilka zmiennych lokalnych było wysyłanych z jednej funkcji do drugiej. Funkcja odbierająca otrzymuje parametr (zmienną) od funkcji, która go wysyła. Nie przejmuj się zbytnio tym, jak je nazywasz - argumentami lub parametrami. Ważną rzeczą do zapamiętania jest to, że wysyłasz zmienne lokalne z jednej funkcji do drugiej.

UWAGA: Przekazałeś już argumenty do parametrów, gdy przekazałeś dane do operatora cout. Literały, zmienne i wyrażenia w nawiasach okrągłych są argumentami. Wbudowana funkcja cout odbiera te wartości (zwane parametrami po stronie odbierającej) i wyświetla je.

Potrzebna jest trochę więcej terminologii, zanim zobaczysz kilka przykładów. Gdy funkcja przekazuje argument, nazywana jest funkcją wywołującą. Funkcja, która odbiera argument (zwana parametrem, gdy jest odbierany) jest nazywana funkcją odbierającą. Aby przekazać zmienną lokalną z jednej funkcji do drugiej, należy umieścić zmienną lokalną w nawiasach zarówno w funkcji wywołującej, jak i odbierającej. Na przykład lokalne i globalne przykłady przedstawione wcześniej nie przekazały zmiennych lokalnych z main() do do_fun (). Jeśli nazwa funkcji ma puste nawiasy, nic nie jest do niej przekazywane. Biorąc to pod uwagę, poniższy wiersz przekazuje dwie zmienne, total i rabat, do funkcji o nazwie do_fun ().

```
do_fun (ogółem, zniżka);
```

Czasami mówi się, że zmienna lub funkcja jest zdefiniowana. Nie ma to nic wspólnego z dyrektywą #define preprocessor, która definiuje literały. Zmienne definiujesz za pomocą instrukcji takich jak:

```
int i, j;

int m = 9;

float x;

char ara [] = „Tulsa”;
```

Te instrukcje mówią programowi, że potrzebujesz tych zmiennych do zarezerwowania. Funkcja jest definiowana, gdy kompilator C++ odczytuje pierwszą instrukcję w funkcji, która opisuje nazwę i kiedy odczytuje wszelkie zmienne, które mogły zostać przekazane również do tej funkcji. Nigdy nie podążaj za definicją funkcji za pomocą średnika, ale zawsze podążaj za instrukcją, która wywołuje funkcję za pomocą średnika.

UWAGA: W przypadku niektórych purystów w C++ zmienna jest deklarowana tylko podczas pisania int i; i tylko prawdziwie zdefiniowane, gdy przypisujesz mu wartość, taką jak i = 7 ;. Mówią, że zmienna jest zarówno deklarowana, jak i definiowana, kiedy deklarujesz zmienną i przypisujesz jej wartość w tym samym czasie, na przykład int i = 7 ;.

Poniższy program zawiera dwie definicje funkcji: main() i pr_it(). Aby przećwiczyć przekazywanie zmiennej do funkcji, zadeklaruj i jako zmienną całkowitą i ustaw ją na pięć. Funkcja przekazywania (lub wywoływania) to main(), a funkcją odbierającą jest pr_it(). Przekaż zmienną i do funkcji pr_it(), a następnie wróć do main().

```
main() // Definicja funkcji main().

{

int i = 5; // Definiuje zmienną całkowitą.

pr_it (i); // Wywołuje pr_it ().

// funkcja i przekazuje ją

zwraca 0; // Powraca do systemu operacyjnego.

}

pr_it (int i) // Definicja funkcji pr_it ().

{

cout << i << „\ n”; // Wywołuje operatora cout.

zwraca 0; // Powraca do main().

}
```

Ponieważ przekazany parametr jest traktowany jak zmienna lokalna w funkcji odbierającej, cout w pr_it () wypisuje 5, mimo że funkcja main() zainicjowała tę zmienną. Gdy przekazujesz argumenty do funkcji, funkcja odbierająca nie jest świadoma typów danych zmiennych przychodzących. Dlatego musisz podać typ danych każdego parametru przed nazwą parametru. W poprzednim przykładzie definicja pr_it () (pierwszy wiersz funkcji) zawiera typ, int, zmiennej przychodzącej i. Zauważ, że funkcja wywołująca main() nie musi wskazywać typu zmiennej. W tym przykładzie main() zna już typ zmiennej i (liczba całkowita); tylko pr_it () musi wiedzieć, że i jest liczbą całkowitą.

WSKAZÓWKA: Zawsze deklaruj typy parametrów w funkcji odbierającej. Poprzedź każdy parametr w nawiasie funkcji znakiem int, float lub dowolnym typem danych każdej przekazywanej zmiennej.

Przykłady

1. Oto funkcja main(), która zawiera trzy zmienne lokalne. main() przekazuje jedną z tych zmiennych do pierwszej funkcji, a dwie z nich do drugiej funkcji.

```
// Nazwa pliku: C17LOC3.CPP

// Przekaż trzy zmienne lokalne do funkcji.

#include <iostream.h>

#include <iomanip.h>

pr_init (char initial); // Prototypy omówione później.

pr_other (int age, float salary);

main()
```

```
{
char initial; // Three variables local to
// main().
int age;
float salary;
// Fill these variables in main().
cout << “What is your initial? “;
cin >> initial;
cout << “What is your age? “;
cin >> age;
cout << “What is your salary? “;
cin >> salary;
pr_init(initial); // Call pr_init() and
// pass it initial.
pr_other(age, salary); // Call pr_other() and
// pass it age and salary.
return 0;
}
pr_init(char initial) // Never put a semicolon in
// the function definition.
{
cout << “Your initial is “ << initial << “\n”;
return 0; // Return to main().
}
pr_other(int age, float salary) // Must type both parameters.
{
cout << “You look young for “ << age << “\n”;
cout << “And “ << setprecision(2) << salary <<
“ is a LOT of money!”;
return 0; // Return to main().
}
```

2. Funkcja odbierająca może zawierać własne zmienne lokalne. Dopóki nazwy nie są takie same, te zmienne lokalne nie powodują konfliktu z przekazanymi. W poniższym programie druga funkcja odbiera przekazywaną zmienną z main() i definiuje własną zmienną lokalną o nazwie price_per.

```
// Nazwa pliku: C17LOC4.CPP

// Druga funkcja ma własną zmienną lokalną.

#include <iostream.h>

#include <iomanip.h>

compute_sale (int galons); // Prototypy omówione później.

main()

{

int gallons;

cout << “Richard’s Paint Service \n”;

cout << “How many gallons of paint did you buy? “;

cin >> gallons; // Get gallons in main().

compute_sale(gallons); // Compute total in function.

return 0;

}

compute_sale(int gallons)

{

float price_per = 12.45; // Local to compute_sale().

cout << “The total is “ << setprecision(2) <<

(price_per*(float)gallons) << “\n”;

// Had to type cast gallons because it was integer.

return 0; // Return to main().
```

3. Poniższe przykładowe linie kodu sprawdzają Twoje umiejętności rozpoznawania funkcji wywoływania i odbierania funkcji. Rozpoznanie różnicy to połowa sukcesu.

```
do_it()
```

Poprzedni fragment musi być pierwszym wierszem nowej funkcji, ponieważ nie kończy się średnikiem.

```
do_it2(sales);
```

Ta linia wywołuje funkcję o nazwie do_it2 (). Funkcja wywołująca przekazuje zmienną o nazwie sales do do_it2 ().

```
pr_it(float total)
```

Poprzedni wiersz jest pierwszym wierszem funkcji, która odbiera zmiennoprzecinkową zmienną z innej funkcji, która ją wywołała. Wszystkie funkcje odbierające muszą określać typ każdej przekazywanej zmiennej.

pr_them(float total, int number)

Jest to pierwszy wiersz funkcji, która odbiera dwie zmienne - jedna jest zmienną zmiennoprzecinkową, a druga liczbą całkowitą. Ta linia nie może wywoływać funkcji pr_them, ponieważ na końcu linii nie ma średnika.

Zmienne automatyczne a zmienne statyczne

Terminy automatyczny i statyczny opisują, co dzieje się ze zmiennymi lokalnymi, gdy funkcja powraca do procedury wywoływania. Domyślnie wszystkie zmienne lokalne są automatyczne, co oznacza, że są kasowane po zakończeniu ich funkcji. Możesz wyznaczyć zmienną jako automatyczną, poprzedzając jej definicję terminem auto. Słowo kluczowe auto jest opcjonalne w przypadku zmiennych lokalnych, ponieważ domyślnie są one automatyczne. Dwie instrukcje po nawiasie otwierającym main() deklarują automatyczne zmienne lokalne:

main()

{

int i;

auto float x;

auto float x;

// Reszta main() idzie tutaj.

Ponieważ auto jest ustawieniem domyślnym, nie trzeba dołączać terminu auto do x.

UWAGA: Programiści C++ rzadko używają słowa kluczowego auto w przypadku zmiennych lokalnych, ponieważ domyślnie są one automatyczne.

Przeciwieństwem zmiennej automatycznej jest zmienna statyczna. Wszystkie zmienne globalne są statyczne i, jak wspomniano, wszystkie zmienne statyczne zachowują swoje wartości. Dlatego, jeśli zmienna lokalna jest statyczna, również zachowuje swoją wartość po zakończeniu funkcji - na wypadek, gdyby funkcja została wywołana po raz drugi. Aby zadeklarować zmienną jako statyczną, podczas definiowania umieść słowo kluczowe static przed zmienną. Poniższa sekcja kodu definiuje trzy zmienne, i, j oraz k. Zmienna i jest automatyczna, ale j i k są statyczne.

my_fun() // Początek definicji nowej funkcji.

{

int i;

static j=25; // Both j and k are static variables.

static k=30;

Zawsze deklaruj wartość początkową zmiennej statycznej, kiedy ją deklarujesz, jak pokazano tutaj w dwóch ostatnich wierszach. Ta wartość początkowa jest umieszczana w zmiennej statycznej tylko przy

pierwszym uruchomieniu my_fun (). Jeśli nie przypiszesz zmiennej statycznej wartości początkowej, C++ inicjuje ją do zera.

WSKAZÓWKA: Zmienne statyczne są przydatne podczas pisania funkcji, które śledzą liczbę lub sumują się do sumy. Gdyby zmienne zliczające lub sumujące były lokalne i automatyczne, ich wartości zniknęłyby po zakończeniu funkcji - niszcząc sumy.

Automatyczne i statyczne reguły dla zmiennych lokalnych

Lokalne zmienne automatyczne znikają po zakończeniu bloku. Wszystkie zmienne lokalne są domyślnie automatyczne. Możesz poprzedzić zmienną (kiedy ją zdefiniujesz) słowem kluczowym auto lub możesz ją pominąć; zmienna jest nadal automatyczna, a jej wartość ulega zniszczeniu po zakończeniu bloku lokalnego. Lokalne zmienne statyczne nie tracą swoich wartości po zakończeniu funkcji. Pozostają lokalne dla tej funkcji. Gdy funkcja jest wywoływana po raz pierwszy, wartość zmiennej statycznej jest nadal na miejscu. Deklarujesz zmienną statyczną, umieszczając słowo kluczowe static przed definicją zmiennej.

Przykłady

1. Rozważ ten program:

```
// Nazwa pliku: C17STA1.CPP

// Próbuje użyć zmiennej statycznej

// bez deklaracji statycznej.

#include <iostream.h>

triple_it (int ctr);

main()

{

int ctr; // Used in the for loop to

// call a function 25 times.

for (ctr=1; ctr<=25; ctr++)

{ triple_it(ctr); } // Pass ctr to a function

// called triple_it().

return 0;

}

triple_it(int ctr)

{

int total=0, ans; // Local automatic variables.

// Triples whatever value is passed to it

// and adds the total.
```

```
ans = ctr * 3; // Triple number passed.

total += ans; // Add triple numbers as this is called.

cout << “The number “ << ctr << “ multiplied by 3 is “

<< ans << “\n”;

if (total > 300)

{ cout << “The total of triple numbers is over 300 \n”; }

return 0;

}
```

To nonsensowny program, który niewiele robi, ale możesz wyczuć, że coś jest nie tak. Program przekazuje liczby od 1 do 25 do funkcji o nazwie triple_it. Funkcja potroi liczbę i wyświetli ją. Zmienna o nazwie total jest początkowo ustawiona na 0. Chodzi tutaj o to ,że dodaje każdą liczbę trzykrotnie i wyświetli wiadomość, gdy suma jest większa niż 300. Jednak cout nigdy się nie wykonuje. Dla każdego z 25 razy wywołania tego podprogramu suma jest resetowana do 0. Suma zmienna jest zmienną automatyczną, której wartość jest kasowana i inicjowana za każdym razem, gdy wywoływana jest jej procedura. Następny przykład to rozwiązuje.

2. Jeśli chcesz, aby suma zachowała swoją wartość po zakończeniu procedury, musisz ustawić ją jako statyczną. Ponieważ zmienne lokalne są domyślnie automatyczne, należy dołączyć słowo kluczowe static, aby zastąpić to ustawienie domyślne. Następnie wartość zmiennej całkowitej jest zachowywana przy każdym wywołaniu podprogramu. Poniższe poprawia błąd w poprzednim programie.

```
// Nazwa pliku: C17STA2.CPP

// Używa zmiennej statycznej z deklaracją statyczną.

#include <iostream.h>

triple_it (int ctr);

main()

{

int ctr; // Used in the for loop to

// call a function 25 times.

for (ctr=1; ctr<=25; ctr++)

{ triple_it(ctr); } // Pass ctr to a function

// called triple_it().

return 0;

}

triple_it(int ctr)

{
```

```
static int total=0; // Local and static

int ans; // Local and automatic

// total is set to 0 only the first time this

// function is called.

// Triples whatever value is passed to it and adds

// the total.

ans = ctr * 3; // Triple number passed.

total += ans; // Add triple numbers as this is called.

cout << “The number “ << ctr << “ multiplied by 3 is “

<< ans << “\n”;

if (total > 300)

{ cout << “The total of triple numbers is over 300 \n”; }

return 0;

}
```

Wyjście tego programu następuje. Zauważ, że cout funkcji jest wyzwalany, mimo że total jest zmienną lokalną. Ponieważ suma jest statyczna, jej wartość nie jest kasowana po zakończeniu funkcji. Gdy main() wywołuje funkcję po raz drugi, poprzednia wartość sumy (w momencie opuszczenia procedury) nadal istnieje.

```
The number 1 multiplied by 3 is 3

The number 2 multiplied by 3 is 6

The number 3 multiplied by 3 is 9

The number 4 multiplied by 3 is 12

The number 5 multiplied by 3 is 15

The number 6 multiplied by 3 is 18

The number 7 multiplied by 3 is 21

The number 8 multiplied by 3 is 24

The number 9 multiplied by 3 is 27

The number 10 multiplied by 3 is 30

The number 11 multiplied by 3 is 33

The number 12 multiplied by 3 is 36

The number 13 multiplied by 3 is 39

The number 14 multiplied by 3 is 42
```

The number 15 multiplied by 3 is 45

The number 16 multiplied by 3 is 48

The number 17 multiplied by 3 is 51

The number 18 multiplied by 3 is 54

The number 19 multiplied by 3 is 57

The number 20 multiplied by 3 is 60

The number 21 multiplied by 3 is 63

The number 22 multiplied by 3 is 66

The number 23 multiplied by 3 is 69

The number 24 multiplied by 3 is 72

The number 25 multiplied by 3 is 75

Nie oznacza to, że lokalne zmienne statyczne stają się globalne. Program główny nie może odwoływać się, używać, drukować ani zmieniać sumy, ponieważ jest lokalny dla drugiej funkcji. Statyczny oznacza po prostu, że wartość zmiennej lokalnej jest nadal dostępna, jeśli program ponownie wywoła funkcję.

Trzy problemy z przekazywaniem parametrów

Aby w pełni zrozumieć programy z kilkoma funkcjami, musisz nauczyć się trzech dodatkowych pojęć:

♦ Przekazywanie argumentów (zmiennych) według wartości (zwane także „przez kopiowanie")

♦ Przekazywanie argumentów (zmiennych) przez adres (zwane także „przez referencję")

♦ Zwracanie wartości z funkcji

Pierwsze dwa pojęcia dotyczą sposobu przekazywania i odbierania zmiennych lokalnych. Trzecia koncepcja opisuje, w jaki sposób funkcje odbierające wysyłają wartości z powrotem do funkcji wywołujących. Rozdział 18, „Przekazywanie wartości", kończy tę dyskusję, wyjaśniając te trzy metody przekazywania parametrów i zwracania wartości

Ćwiczenia

1. Napisz program, który pyta w main() o wiek psa użytkownika. Napisz drugą funkcję o nazwie people (), która oblicza wiek psa w latach ludzkich (mnożąc wiek psa przez siedem).

2. Napisz funkcję, która zlicza liczbę wywołań. Nazwij funkcję count_it (). Nic nie przekazuj. W treści count_it () wypisz następujący komunikat: Liczba wywołań tej funkcji to: ## gdzie ## jest liczbą. (Wskazówka: Ponieważ zmienna musi być lokalna, ustaw ją na statyczną i zainicjuj ją do zera podczas pierwszej definicji).

3. Poniższy program zawiera kilka problemów. Niektóre z tych problemów powodują błędy. Jednym z problemów nie jest błąd, ale zła lokalizacja deklaracji zmiennej. (Wskazówka: Znajdź wszystkie zmienne globalne.) Sprawdź, czy możesz zauważyć niektóre problemy i przepisz program, aby działał lepiej.

// Nazwa pliku: C17BAD.CPP

// Program z niewłaściwym wykorzystaniem deklaracji zmiennych.

```
#include <iostream.h>

# zdefiniuj NUM 10

do_var_fun (); // Prototypy omówione później.

char city [] = „Miami”;

int count;

main()

{

int abc;

count = NUM;

abc = 5;

do_var_fun();

cout << abc << “ “ << count << “ “ << pgm_var << “ “

<< xyz;

return 0;

}

int pgm_var = 7;

do_var_fun()

{

char xyz = ‘A’;

xyz = ‘b’;

cout << xyz << “ “ << pgm_var << “ “ abc << “ “ << city;

return 0;

}
```

Podsumowanie

Przekazywanie parametrów jest konieczne, ponieważ zmienne lokalne są lepsze niż globalne. Zmienne lokalne są chronione we własnych procedurach, ale czasami muszą być współużytkowane z innymi procedurami. Jeśli dane lokalne mają pozostać w tych zmiennych (w przypadku ponownego wywołania funkcji w tym samym programie), zmienne powinny być statyczne, ponieważ w przeciwnym razie ich automatyczne wartości znikną.

Przekazywanie wartości

C++przekazuje zmienne między funkcjami przy użyciu dwóch różnych metod. Ten, którego używasz, zależy od tego, jak chcesz zmienić przekazywane zmienne. W tej sekcji opisano te dwie metody. Pojęcia omówione tutaj nie są nowe w języku C ++. Inne języki programowania, takie jak Pascal, FORTRAN i QBasic, przekazują parametry przy użyciu podobnych technik. Język komputerowy musi mieć możliwość przekazywania informacji między funkcjami, zanim będzie można go nazwać prawdziwie ustrukturyzowanym. Ta część zawiera następujące informacje:

♦ Przekazywanie zmiennych według wartości

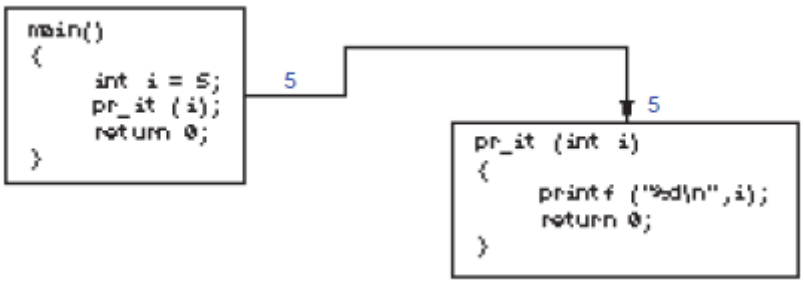
♦ Przekazywanie tablic według adresu

♦ Przekazywanie nonarrays przez adres

Zwróć szczególną uwagę, ponieważ większość programów w pozostałej części książki opiera się na metodach opisanych w tym rozdziale.

Przekazywanie według wartości (według kopii)

Dwa sformułowania „przekazywanie według wartości" i „przekazywanie przez kopię" oznaczają to samo w kategoriach komputerowych. Niektóre podręczniki i programiści C++stwierdzają, że argumenty są przekazywane przez wartość, a niektóre twierdzą, że są przekazywane przez kopię. Oba te wyrażenia opisują jedną z dwóch metod przekazywania argumentów do funkcji odbierających. (Druga metoda nazywana jest „przez adres" lub „przez odniesienie". Ta metoda została omówiona w dalszej części rozdziału.) Gdy argument (zmienna lokalna) jest przekazywany przez wartość, kopia wartości zmiennej jest wysyłana do - i jest przypisany do parametru funkcji odbierającej. Jeśli wartość przekazuje więcej niż jedną zmienną, kopia każdej z ich wartości jest wysyłana do parametrów funkcji odbierającej i przypisywana do nich. Rysunek pokazuje przekazywanie w akcji. Wartość i - nie zmienna - jest przekazywana do wywoływanej funkcji, która odbiera ją jako zmienną i. Istnieją dwie zmienne o nazwie i, a nie jedna. Pierwszy jest lokalny dla main(), a drugi lokalny dla pr_it(). Obie mają te same nazwy, ale ponieważ są lokalne dla swoich funkcji, nie ma konfliktu. Zmienna nie musi być wywoływana i w obu funkcjach, a ponieważ wartość i jest wysyłana do funkcji odbierającej, nie ma znaczenia, jak funkcja odbierająca wywołuje zmienną, która odbiera tę wartość.



W takim przypadku, podczas przekazywania i odbierania zmiennych między funkcjami, najlepiej jest zachować te same nazwy. Mimo że nie są to te same zmienne, mają tę samą wartość. W tym przykładzie wartość 5 jest przekazywana z i (main)) do i_pr_it(). Ponieważ kopia wartości i (a nie sama zmienna) jest przekazywana do funkcji odbierającej, gdyby pr_it () zmienił i, zmieniłby tylko swoją kopię i, a nie main() i. Ten fakt naprawdę oddziela funkcje i zmienne. Masz teraz technikę przekazywania kopii zmiennej do funkcji odbierającej, przy czym funkcja odbierająca nie jest w stanie zmodyfikować zmiennej funkcji wywołującej. Wszystkie zmienne, które do tej pory widziałeś w C ++, są przekazywane

przez wartość. Nie musisz robić nic specjalnego, aby przekazywać zmienną wartość, z wyjątkiem przekazania ich na listę argumentów funkcji wywołującej i otrzymania ich na liście parametrów funkcji odbierającej.

UWAGA: Domyślną metodą przekazywania parametrów jest wartość, jak właśnie opisano, chyba że przekazujesz tablice. Tablice są zawsze przekazywane inną metodą, według adresu, opisaną w dalszej części.

Przykłady

1. Poniższy program pyta użytkowników o ich wagę. Następnie przekazuje ten ciężar do funkcji, która oblicza równoważną wagę na Księżycu. Zauważ, że druga funkcja wykorzystuje przekazaną wartość i oblicza ją. Po przekazaniu wagi do drugiej funkcji, funkcja ta może traktować wagę tak, jakby była zmienną lokalną. Zidentyfikuj program i dołącz niezbędny plik wejściowy / wyjściowy. Chcesz obliczyć wagę użytkownika na Księżycu. Ponieważ musisz gdzieś utrzymać wagę użytkownika, zadeklaruj zmienną wagę jako liczbę całkowitą. Potrzebujesz także funkcji wykonującej obliczenia, więc utwórz funkcję o nazwie moon (). Zapytaj użytkownika, ile on lub ona waży. Podaj wagę odpowiedzi użytkownika. Teraz przekaż wagę użytkownika do funkcji moon (), która dzieli wagę przez sześć, aby określić równoważną wagę na Księżycu. Wyświetlaj wagę użytkownika na Księżycu. Zakończyłeś, więc opuść funkcję moon (), a następnie opuść

główna funkcja.

```
// Nazwa pliku: C18PASS1.CPP

// Oblicz wagę użytkownika w drugiej funkcji.

#include <iostream.h>

moon(weight); // Call the moon() function and

// pass it the weight.

return 0; // Return to the operating system.

}

moon(int weight) // Declare the passed parameter.

{

// Moon weights are 1/6th earth's weights

weight /= 6; // Divide the weight by six.

cout << "You weigh only " << weight <<

" pounds on the moon!";

return 0; // Return to main().

}
```

Dane wyjściowe tego programu są następujące:

```
How many pounds do you weigh? 120

You weigh only 20 pounds on the moon!
```

2. Możesz zmienić nazwę przekazywanych zmiennych w funkcji odbierającej. Różnią się one od zmiennej funkcji przekazującej. Poniżej przedstawiono ten sam program jak w przykładzie 1, z tą różnicą, że funkcja odbierająca wywołuje przekazaną zmienną earth_weight. Nowa zmienna, o nazwie moon_weight, jest lokalna dla wywoływanej funkcji i jest używana do równoważnej masy księżyca.

```
// Nazwa pliku: C18PASS2.CPP

// Oblicz wagę użytkownika w drugiej funkcji.

#include <iostream.h>

moon(int earth_weight);

main()

{

int weight; // main()’s local weight.

cout << “How many pounds do you weigh? “;

cin >> weight;

moon(weight); // Call the moon() function and

// pass it the weight.

return 0; // Return to the operating system.

}

moon(int earth_weight) // Declare the passed parameter.

{

int moon_weight; // Local to this function.

// Moon's weights are 1/6th of earth’s weights.

moon_weight = earth_weight / 6; // Divide weight by six.

cout << “You only weigh “ << moon_weight <<

“ pounds on the moon!”;

return 0; // Return to main().

}
```

Wynikowe wyjście jest identyczne jak w poprzednim programie. Zmiana nazwy przekazanej zmiennej nic nie zmienia.

3. Następny przykład przekazuje trzy zmienne - trzech różnych typów - do wywoływanej funkcji. Na liście parametrów funkcji odbierającej należy zadeklarować każdy z tych typów zmiennych. Ten program monituje użytkowników o podanie trzech wartości w funkcji main(). Funkcja main() następnie przekazuje te zmienne do funkcji odbierającej, która oblicza i drukuje wartości związane z tymi przekazywanymi zmiennymi. Gdy wywoływana funkcja modyfikuje zmienną przekazaną do funkcji, zauważ to jeszcze raz nie wpływa to na zmienną funkcji wywołującej. Gdy zmienne są przekazywane przez wartość, wartość - nie zmienna - jest przekazywana.

```
// Nazwa pliku: C18PASS3.CPP
// Uzyskaj informacje o ocenach dla ucznia.
#include <iostream.h>
#include <iomanip.h>
check_grade(char lgrade, float average, int tests);
main()
{
char lgrade; // Letter grade.
int tests; // Number of tests not yet taken.
float average; // Student's average based on 4.0 scale.
cout << "What letter grade do you want? ";
cin >> lgrade;
cout << "What is your current test average? ";
cin >> average;
cout << "How many tests do you have left? ";
cin >> tests;
check_grade(lgrade, average, tests); // Calls function
// and passes three variables by value.
return 0;
}
check_grade(char lgrade, float average, int tests)
{
switch (tests)
{
case (0): { cout << "You will get your current grade "
<< "of " << lgrade;
break; }
case (1): { cout << "You still have time to bring " <<
"up your average";
cout << "of " << setprecision(1) <<
average << "up. Study hard!";
```

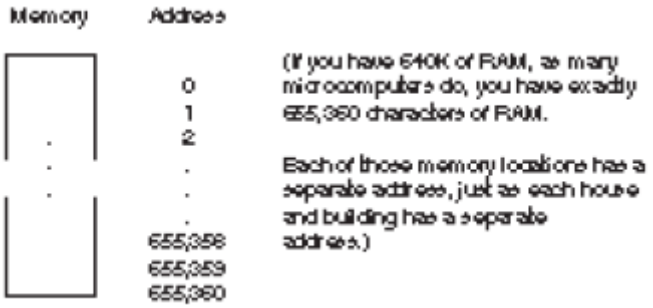
```
break; }

default: { cout << “Relax. You still have plenty of “

<< “time.”;

break; }

}

return 0;

}
```

Przekazywanie przez adres (przez odniesienie)

Te dwa wyrażenia „według adresu” i „przez odniesienie” oznaczają to samo. W poprzedniej sekcji opisano przekazywanie argumentów według wartości (lub kopii). Ta sekcja uczy, jak przekazywać argumenty według adresu. Gdy przekazujesz argument (zmienną lokalną) według adresu, adres zmiennej jest wysyłany do parametru funkcji odbierającej i przypisywany do niego. (Jeśli przekażesz więcej niż jedną zmienną według adresu, każdy z ich adresów jest wysyłany do parametrów funkcji odbierającej i przypisywany do nich).

Zmienne adresy

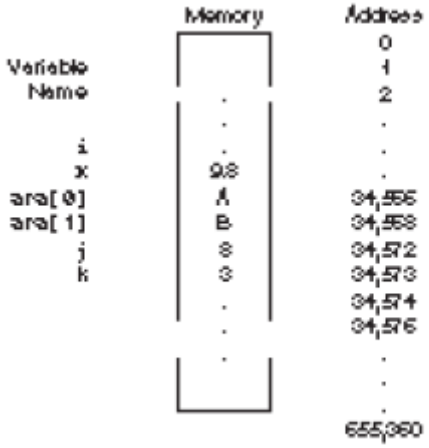
Wszystkie zmienne w pamięci (RAM) są przechowywane pod adresami pamięci.Aby uzyskać więcej informacji na temat wewnętrznej reprezentacji pamięci, zobacz Dodatek A, „Adresowanie pamięci, przegląd binarny i przegląd szesnastkowy”.



Kiedy mówisz C ++, aby zdefiniował zmienną (taką jak int i;), prosisz C++ o znalezienie nieużywanego miejsca w pamięci i przypisanie tego miejsca (lub adresu pamięci) do i. Kiedy twój program używa zmiennej o nazwie i, C++ podchodzi do adresu i używa wszystkiego, co tam jest. Jeśli zdefiniujesz pięć zmiennych w następujący sposób,

```
int i;

float x=9.8;

char ara[2] = {‘A’, ‘B’};

int j=8, k=3;
```

C++ może dowolnie umieścić je w pamięci pod adresami pokazanymi na rysunku.



Nie wiesz, co zawiera zmienna o nazwie i, ponieważ jeszcze jej nie umieściłeś. Przed użyciem i należy zainicjować go wartością. (Wszystkie zmienne - oprócz zmiennych znakowych - zwykle zajmują więcej niż 1 bajt pamięci.)

Przykładowy program

Adres zmiennej, a nie jej wartość, jest kopiowany do funkcji odbierającej po przekazaniu zmiennej przez adres. W C++ wszystkie tablice są automatycznie przekazywane przez adres. (W rzeczywistości kopia ich adresu jest przekazywana, ale zrozumiesz to lepiej, gdy dowiesz się więcej o tablicach i wskaźnikach). Następująca ważna zasada obowiązuje w przypadku programów, które przekazują według adresu: Za każdym razem, gdy przekazujesz zmienną według adresu, jeśli funkcja odbierająca zmienia zmienną, jest również zmieniana w funkcji wywołującej. Dlatego jeśli przekażesz tablicę do funkcji, a funkcja zmieni tablicę, zmiany te będą nadal obowiązywały w tablicy, gdy powróci ona do funkcji wywołującej. W przeciwieństwie do przekazywania przez wartość, przekazywanie według adresu daje możliwość zmiany zmiennej w wywoływanej funkcji i utrzymania tych zmian w funkcji wywołującej. Poniższy przykładowy program pomaga zilustrować tę koncepcję.

```
// Nazwa pliku: C18ADD1.CPP

// Przykład podania adresu.

#include <iostream.h>

#include <string.h>

change_it (char c [4]); // Prototyp omówiony później

main()

{

char name[4]=”ABC”;

change_it(name); // Passes by address because

// it is an array.

cout << name << “\n”; // Called function can

// change array.
```

return 0;

}

change_it(char c[4]) // You must tell the function

// that c is an array.

{

cout << c << “\n”; // Print as it is passed.

strcpy(c, “USA”); // Change the array, both

// here and in main().

return 0;

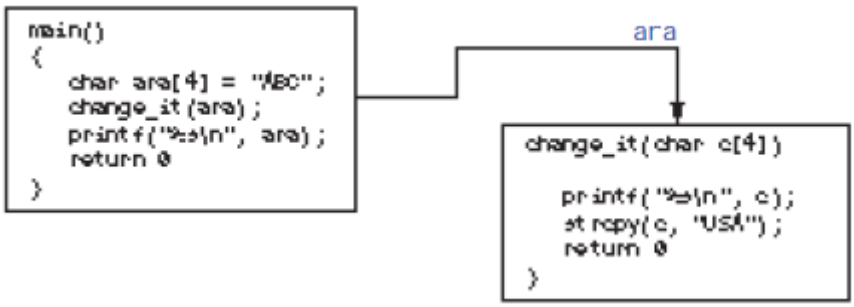
}

Oto wynik tego programu:

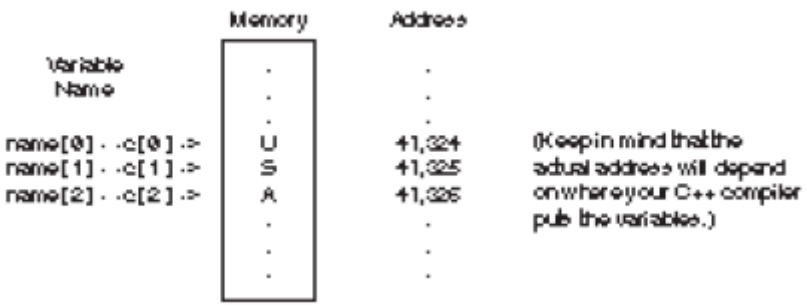
ABC

USA

W tym momencie nie powinieneś mieć problemów ze zrozumieniem, że tablica jest przekazywana z main() do funkcji o nazwie change_it (). Mimo że change_it () wywołuje tablicę c, odnosi się do tej samej tablicy przekazywanej przez funkcję main() (name). Rysunek pokazuje sposób przekazywania tablicy. Chociaż adres tablicy - a nie jej wartość - jest przekazywany z nazwy na c, obie tablice są takie same.



Zanim przejdziemy dalej, kilka dodatkowych komentarzy jest w porządku. Ponieważ adres nazwy jest przekazywany do funkcji - mimo że tablica nazywa się c w funkcji odbierającej - nadal jest taką samą tablicą jak nazwa. Rysunek pokazuje, jak C++wykonuje to zadanie na poziomie adresu pamięci.



Tablica zmiennych jest określana jako nazwa w main() i jako c w change_it(). Ponieważ adres nazwy jest kopiowany do odbiorcy funkcja zmienna jest zmieniana bez względu na to, jak się ją wywołuje. Ponieważ change_it () zmienia tablicę, tablica jest zmieniana również w main().

Przykłady

1. Możesz teraz użyć funkcji do wypełnienia tablicy danymi wejściowymi użytkownika. Poniższa funkcja prosi użytkowników o podanie imienia w funkcji o nazwie get_name(). Gdy użytkownicy wpisują nazwę w tablicy, jest ona również wpisywana w tablicy main(). Funkcja main() przekazuje tablicę do pr_name(), gdzie jest wyświetlana. (Gdyby tablice były przekazywane przez wartość, ten program nie działałby. Tylko wartość tablicy byłaby przekazywana do wywoływanych funkcji).

```
// Nazwa pliku: C18ADD2.CPP

// Uzyskaj nazwę w tablicy, a następnie wydrukuj ją za pomocą

// oddzielne funkcje.

#include <iostream.h>


get_name(char name[25]); // Prototypes discussed later.

print_name(char name[25]);

main()

{

char name[25];

get_name(name); // Get the user's name.

print_name(name); // Print the user's name.

return 0;

}

get_name(char name[25]) // Pass the array by address.

{

cout << "What is your first name? ";

cin >> name;

return 0;

}

print_name(char name[25])

{

cout << "\n\n Here you are, " << name;

return 0;
```

}

Gdy przekazujesz tablicę, pamiętaj, aby określić typ tablicy na liście parametrów funkcji odbierającej. Jeśli poprzedni program zadeklarował przekazaną tablicę za pomocą

get_name(char name)

funkcja get_name() interpretowałaby to jako zmienną jednoznakową, a nie tablicę znaków. Nigdy nie musisz umieszczać rozmiaru tablicy w nawiasach. Poniższa instrukcja działa również jako pierwszy wiersz get_name ().

get_name (char name [])

Większość programistów C++umieszcza rozmiar tablicy w nawiasach, aby wyjaśnić rozmiar tablicy, nawet jeśli rozmiar nie jest potrzebny.

2. Wielu programistów przekazuje tablice znaków do funkcji, aby je usunąć. Oto funkcja o nazwie clear_it (). Oczekuje dwóch parametrów: tablicy znaków i całkowitej liczby elementów zadeklarowanych dla tej tablicy. Tablica jest przekazywana przez adres (jak wszystkie tablice), a liczba elementów, num_els, jest przekazywana przez wartość (jak wszystkie tablice nonarra). Po zakończeniu funkcji tablica jest czyszczona (wszystkie jej elementy są zerowane). Kolejne funkcje, które z niego korzystają, mogą mieć pustą tablicę.

clear_it (char ara [10], int num_els)

{

int ctr;

for (ctr = 0; ctr <num_els; ctr ++)

{ara [ctr] = „\ 0”; }

return 0;

}

Nawiasy po ara nie muszą zawierać liczby, jak opisano w poprzednim przykładzie. 10 w tym przykładzie jest po prostu symbolem zastępczym dla nawiasów. Każda wartość (lub żadna wartość) również by działała.

Przekazywanie nonarrays przez adres

Teraz powinieneś zobaczyć różnicę między przekazywaniem zmiennych według adresu i wartości. Tablice mogą być przekazywane według adresu, a nonarrays mogą być przekazywane według wartości. Możesz zastąpić domyślną wartość domyślną dla nonarrays. Jest to czasem pomocne, ale nie zawsze jest zalecane, ponieważ wywoływana funkcja może uszkodzić wartości w pliku ,nazywana funkcją. Jeśli chcesz, aby zmienna nonarray została zmieniona w funkcji odbierającej, a także chcesz, aby zmiany zachowały się w funkcji wywołującej, musisz zastąpić wartość domyślną i przekazać zmienną przez adres. (Powinieneś zrozumieć tę sekcję lepiej, gdy dowiesz się, jak tablice i wskaźniki odnoszą się.) Aby przekazać nonarray przez adres, musisz poprzedzić argument w funkcji odbierającej znakiem ampersand (&). Może ci to brzmieć dziwnie (i tak jest w tym momencie). Niewielu programistów C++zastępuje domyślne przekazywanie według adresu. Kiedy dowiesz się o wskaźnikach później, nie powinieneś tego robić. Większość programistów C++ nie lubi zaśmiecać swojego kodu tymi dodatkowymi znakami ampersand, ale miło jest wiedzieć, że możesz zastąpić domyślny jeśli to

konieczne. Poniższe przykłady pokazują, w jaki sposób przekazywać zmienne niebędące tablicami według adresu.

Przykłady

1. Poniższy program przekazuje zmienną według adresu z funkcji main() do funkcji. Funkcja zmienia to i wraca do main(). Ponieważ zmienna jest przekazywana przez adres, main() rozpoznaje nową wartość.

```
// Nazwa pliku: C18ADD3.CPP

// Zademonstruj przekazywanie nonarrays według adresu.

#include <iostream.h>

do_fun (int & amt); // Prototypy omówione później.

main()

{

int amt;

amt = 100; // Assign a value in main().

cout << “In main(), amt is “ << amt << “\n”;

do_fun(amt); // Pass amt by address

cout << “After return, amt is “ << amt << “ in main()\n”;

return 0;

}

do_fun(int &amt) // Inform function of

// passing by address.

{

amt = 85; // Assign new value to amt.

cout << “In do_fun(), amt is “ << amt << “\n”;

return 0;

}
```

Dane wyjściowe z tego programu są następujące:

```
In main(), amt is 100

In do_fun(), amt is 85

After return, amt is 85 in main()
```

Zauważ, że amt zmienił się w wywołanej funkcji. Ponieważ został przekazany przez adres, jest również zmieniany w funkcji wywołującej.

2. Możesz użyć funkcji, aby uzyskać wartości klawiatury użytkownika. Funkcja main() rozpoznaje te wartości, o ile przekazujesz je według adresu. Poniższy program oblicza stopy sześcienne w basenie. W jednej funkcji żąda szerokości, długości i głębokości. W innej funkcji oblicza sześcienne stopy wody. Wreszcie, w trzeciej funkcji, wypisuje odpowiedź. Funkcja main() jest wyraźnie funkcją sterującą, przekazującą zmienne między tymi funkcjami według adresu.

```
// Nazwa pliku: C18POOL.CPP

// Oblicza stopy sześcienne w basenie.

#include <iostream.h>

get_values(int &length, int &width, int &depth);

calc_cubic(int &length, int &width, int &depth, int &cubic);

print_cubic(int &cubic);

main()

{

int length, width, depth, cubic;

get_values(length, width, depth);

calc_cubic(length, width, depth, cubic);

print_cubic(cubic);

return 0;

}

get_values(int &length, int &width, int &depth)

{

cout << “What is the pool’s length? “;

cin >> length;

cout << “What is the pool’s width? “;

cin >> width;

cout << “What is the pool’s average depth? “;

cin >> depth;

return 0;

}

calc_cubic(int &length, int &width, int &depth, int &cubic)

{

cubic = (length) * (width) * (depth);
```

```
return 0;

}

print_cubic(int &cubic)

{

cout << "\nThe pool has " << cubic << " cubic feet\n";

return 0;

}
```

Dane wyjściowe są następujące:

What is the pool's length? 16

What is the pool's width? 32

What is the pool's average depth? 6

The pool has 3072 cubic feet

Wszystkie zmienne w funkcji muszą być poprzedzone znakiem ampersand, jeśli mają być przekazywane przez adres.

Ćwiczenia

1. Napisz funkcję main() i drugą funkcję, którą wywołuje main(). Zapytaj użytkowników o ich roczny dochód w main(). Przenieś dochód do drugiej funkcji i wydrukuj wiadomość gratulacyjną, jeśli użytkownik zarobi więcej niż 50 000 USD, lub wiadomość zachęty, jeśli użytkownik zarobi mniej.

2. Napisz program trzyfunkcyjny, składający się z następujących funkcji:

main()

fun1()

fun2()

Zadeklaruj 10-elementową tablicę znaków w main(), wypełnij ją literami od A do J w fun1 (), a następnie wydrukuj tę tablicę wstecz w fun2 ().

3. Napisz program, którego funkcja main() przekazuje liczbę do funkcji o nazwie print_aster (). Funkcja print_aster () drukuje tyle gwiazdek w linii na ekranie. Jeśli parametr print_aster () przejdzie liczbę większą niż 80, wyświetl błąd, ponieważ większość ekranów nie może wydrukować więcej niż 80 znaków w tym samym wierszu. Po zakończeniu wykonywania przywróć kontrolę do main(), a następnie wróć do systemu operacyjnego.

4. Napisz funkcję, której adres poda dwie wartości całkowite. Funkcja powinna zadeklarować trzecią zmienną lokalną. Użyj trzeciej zmiennej jako zmiennej pośredniej i zamień wartości obu przekazanych liczb całkowitych. Załóżmy na przykład, że funkcja wywołująca przekazuje twoją funkcję old_pay i new_pay jak w swap_it (old_pay, new_pay); Funkcja swap_it () odwraca dwie wartości, więc gdy formant powróci do funkcji wywołującej, wartości old_pay i new_pay są zamieniane.

Podsumowanie

Teraz masz pełne zrozumienie różnych metod przekazywania danych do funkcji. Ponieważ będziesz używać zmiennych lokalnych w jak największym stopniu, musisz wiedzieć, jak przekazywać zmienne lokalne między funkcjami, ale także trzymać te zmienne z dala od funkcji, które ich nie potrzebują. Możesz przekazać dane na dwa sposoby: według wartości i adresu. Gdy przekazujesz dane według wartości, która jest domyślną metodą dla nonarrays, przekazywana jest tylko kopia zawartości zmiennej. Jeśli wywoływana funkcja modyfikuje parametry, zmienne te nie są modyfikowane w funkcji wywołującej. Gdy przekazujesz dane według adresu, tak jak ma to miejsce w przypadku tablic i zmiennych nonarray poprzedzonych znakiem ampersand, funkcja odbierająca może zmienić dane w obu funkcjach. Za każdym razem, gdy przekazujesz wartości, musisz upewnić się, że pasują do siebie liczbą i typem. Jeśli ich nie dopasujesz, możesz mieć problemy. Załóżmy na przykład, że przekazujesz tablicę i zmienną zmiennoprzecinkową, ale w funkcji odbierającej otrzymujesz zmienną zmiennoprzecinkową, po której następuje tablica. Dane nie osiągają poprawnie funkcji odbierającej, ponieważ typy danych parametrów nie pasują do przekazywanych zmiennych.

Funkcja Zwracane wartości i prototypy

Do tej pory przekazywałeś zmienne do funkcji tylko w jednym kierunku - funkcja wywołująca przekazała dane do funkcji odbierającej. Musisz jeszcze zobaczyć, w jaki sposób dane są przekazywane z funkcji odbierającej do funkcji wywołującej. Gdy przekazujesz zmienne według adresu, dane są zmieniane w obu funkcjach - ale różni się to od przekazywania danych z powrotem. Ten rozdział koncentruje się na pisaniu wartości zwracanych przez funkcje, które poprawiają moc programowania. Po nauczeniu się przekazywania i zwracania wartości musisz prototypować własne funkcje, a także wbudowane funkcje C++, takie jak cout i cin. Prototypując swoje funkcje, zapewniasz dokładność przekazywanych i zwracanych wartości. Ta część zawiera następujące informacje:

♦ Zwracanie wartości z funkcji

♦ Funkcje prototypowania

♦ Zrozumienie plików nagłówkowych

Zwracając wartości z funkcji, czynisz swoje funkcje w pełni modułowymi. Mogą teraz wyróżniać się spośród innych funkcji. Mogą odbierać i zwracać wartości oraz pełnić rolę bloków składających się na całą aplikację.

Zwracane wartości funkcji

Do tej pory wszystkie funkcje były podprogramami lub podfunkcjami. Podprogram C++ jest funkcją wywoływaną z innej funkcji, ale nie zwraca żadnych wartości. Różnica między podprogramami a funkcjami nie jest tak istotna w C++, jak w innych językach. Wszystkie funkcje, niezależnie od tego, czy są to podprogramy, czy funkcje zwracające wartości, są zdefiniowane w ten sam sposób. Możesz przekazać zmienne do każdego z nich, jak widzieliście w tej części książki. Funkcje zwracające wartości oferują nowe podejście do programowania. Oprócz przekazywania danych w jedną stronę, od wywołania do funkcji odbierającej, możesz przekazać dane z funkcji odbierającej do funkcji wywołującej. Jeśli chcesz zwrócić wartość z funkcji do jej funkcji wywołującej, wstaw wartość zwracaną po zwrocie komunikat. Aby jeszcze bardziej wyjaśnić zwracaną wartość, wielu programistów umieszcza nawiasy wokół zwracanej wartości, jak pokazano w następującej składni:

return (wartość zwracana);

UWAGA: Nie zwracaj zmiennych globalnych. Nie ma takiej potrzeby, ponieważ ich wartości są już znane w całym kodzie.

Funkcja wywołująca musi mieć zastosowanie dla wartości zwracanej. Załóżmy na przykład, że napisałeś funkcję obliczającą średnią z dowolnych trzech zmiennych całkowitych przekazanych do niego. Jeśli zwrócisz średnią, funkcja wywołująca musi otrzymać tę zwracaną wartość. Poniższy przykładowy program pomaga zilustrować tę zasadę.

```
// Nazwa pliku: C19AVG.CPP

// Oblicza średnią z trzech wartości wejściowych.

#include <iostream.h>

int calc_av(int num1, int num2, int num3); //Prototype

main()

{
```

```
int num1, num2, num3;

int avg; // Holds the return value.

cout << “Please type three numbers (such as 23 54 85) “;

cin >> num1 >> num2 >> num3;

// Call the function, pass the numbers,

// and accept the return value amount.

avg = calc_av(num1, num2, num3);

cout << “\n\nThe average is “ << avg; // Print the

// return value.

return 0;

}

int calc_av(int num1, int num2, int num3)

{

int local_avg; // Holds the average for these numbers.

local_avg = (num1+num2+num3) / 3;

return (local_avg);

}
```

Oto przykładowe wyjście z programu:

```
Please type three numbers (such as 23 54 85) 30 40 50

The average is 40
```

Przestudiuj dokładnie ten program. Jest podobny do wielu, które widziałeś, ale teraz, gdy funkcja zwraca wartość, należy wziąć pod uwagę kilka dodatkowych punktów. Może to pomóc przejść przez ten program kilka linii na raz. Pierwsza część main() jest podobna do innych programów, które widziałeś. Deklaruje lokalne zmienne: trzy dla danych wejściowych użytkownika i jedna dla obliczonej średniej. cout i cin są ci znane. Wywołanie funkcji funkcji calc_av() jest również znane; przekazuje trzy zmienne (num1, num2 i num3) przez wartość do calc_av(). (Jeśli przekaże je adres, ampersand (&) musiałby poprzedzać każdy argument, jak omówiono w rozdziale 18.) Funkcja odbierająca calc_av () wydaje się podobna do innych, które widziałeś. Jedyną różnicą jest to, że pierwsza linia, linia definicji funkcji, ma jeden dodatek - int przed jej nazwą. Jest to typ zwracanej wartości. Zawsze należy poprzedzać nazwę funkcji zwracanym typem danych. Jeśli nie określisz typu, C++ zakłada typ int. Dlatego jeśli ten przykład nie ma typu zwracanego, działałby równie dobrze, ponieważ przyjęto by typ zwracany int. Ponieważ zmienna zwracana z funkcji calc_av () jest liczbą całkowitą, typ zwracanej wartości int jest umieszczany przed nazwą calc_av (). Widać również, że instrukcja return calc_av () zawiera zwracaną wartość, local_avg. Jest to zmienna wysyłana z powrotem do funkcji wywołującej main(). Do funkcji wywołującej można zwrócić tylko jedną zmienną. Chociaż funkcja może otrzymać więcej niż jeden parametr, może zwrócić tylko jedną wartość funkcji wywołującej. Jeśli funkcja odbierająca modyfikuje więcej niż jedną wartość z funkcji wywołującej, musisz przekazać parametry

według adresu; nie można zwrócić wielu wartości za pomocą instrukcji return. Po funkcji odbierającej calc_av () zwraca wartość, main() musi coś zrobić z tą zwróconą wartością. Do tej pory sam widziałeś wywołania funkcji na liniach. Zauważ w main(), że wywołanie funkcji pojawia się po prawej stronie następującej instrukcji przypisania:

avg = calc_av (num1, num2, num3);

Gdy funkcja calc_av () zwraca swoją wartość - średnią z trzech liczb - ta wartość zastępuje wywołanie funkcji. Jeśli średnia obliczona w calc_av () wynosi 40, kompilator C++ interpretuje następującą instrukcję zamiast wywołania funkcji:

avg = 40;

Wpisałeś wywołanie funkcji po prawej stronie znaku równości, ale program zastępuje wywołanie funkcji jego wartością zwracaną, gdy nastąpi powrót. Innymi słowy, funkcja zwracająca wartość staje się tą wartością. Musisz umieścić taką funkcję w dowolnym miejscu dowolnej zmiennej lub literału (zwykle po prawej stronie znaku równości, w wyrażeniu lub w cout). Poniżej przedstawiono nieprawidłowy sposób dzwonienia

calc_av ():

calc_av (num1, num2, num3);

Gdybyś to zrobił, C++ nie miałby gdzie umieścić wartości zwracanej.

UWAGA: Wywołania funkcji, które zwracają wartości, zwykle nie pojawiają się same w wierszach. Ponieważ wywołanie funkcji jest zastąpione wartością zwracaną, powinieneś zrobić coś z tą wartością zwracaną (na przykład przypisać ją do zmiennej lub użyć w wyrażeniu). Zwracane wartości można zignorować, ale zazwyczaj nie pozwala to na ich utworzenie.

Przykłady

1. Poniższy program przekazuje liczbę do funkcji o nazwie double(). Funkcja podwaja liczbę i zwraca wynik.

```
// Nazwa pliku: C19DOUB.CPP

// Podwaja numer użytkownika.

#include <iostream.h>

int double (int num);

main()

{

int number; // Holds user's input.

int d_number; // Holds double the user's input.

cout << "What number do you want doubled? ";

cin >> number;

d_number = doub(number); // Assigns return value.

cout << number << " doubled is " << d_number;
```

```
return 0;

}

int doub(int num)

{

int d_num;

d_num = num * 2; // Doubles the number.

return (d_num); // Returns the result.

}
```

Program generuje dane wyjściowe takie jak to:

```
What number do you want doubled? 5

5 doubled is 10
```

2. Zwracane wartości funkcji mogą być używane wszędzie tam, gdzie używane są literały, zmienne i wyrażenia. Poniższy program jest podobny do poprzedniego. Różnica polega na funkcji main(). Wywołanie funkcji jest wykonywane nie na samej linii, ale z cout. To jest zagnieżdżone wywołanie funkcji. Wywołujesz funkcję wbudowaną cout, używając wartości zwracanej z jednej z funkcji programu o nazwie Dougl (). Ponieważ wywołanie metody double () jest zastępowane wartością zwracaną, cout ma wystarczającą ilość informacji, aby kontynuować, gdy tylko funkcja return () zwróci. Daje to main() mniejszy narzut, ponieważ nie potrzebuje już zmiennej o nazwie d_number, chociaż musisz sam zdecydować, czy ten program jest łatwiejszy w utrzymaniu. Czasami rozsądnie jest zawrzeć wywołania funkcji w innych wyrażeniach; innym razem łatwiej jest wywołać funkcję i przypisać jej wartość zwrotną do zmiennej przed jej użyciem.

```
// Filename: C19SUMD.CPP

// Compute the sum of the digits.

#include <iostream.h>

int sum(int num); // Prototype

main()

{

int num, sumd;

cout << “Please type a number: “;

cin >> num;

sumd = sum(num);

cout << “The sum of the digits is “ << sumd;

return 0;

}
```

```
int sum(int num)
{
int ctr; // Local loop counter.
int sumd=0; // Local to this function.
if (num <= 0) // Check whether parameter is too small.
{ sumd = num; } // Returns parameter if too small.
else
{ for (ctr=1; ctr<=num; ctr++)
{ sumd += ctr; }
}
return(sumd);
}
```

Oto przykładowe dane wyjściowe z tego programu:

```
Please type a number: 6
The sum of the digits is 21
```

4. Poniższy program zawiera dwie funkcje zwracające wartości. Pierwsza funkcja, maximum (), zwraca większą z dwóch liczb wprowadzonych przez użytkownika. Drugi, minimum (), zwraca mniejszy.

```
// Nazwa pliku: C19MINMX.CPP
// Znajduje wartości minimalne i maksymalne w funkcjach.
#include <iostream.h>
int maximum (int num1, int num2); // Prototypy
int minimum (int num1, int num2);
main()
{
int num1, num2; // User's two numbers.
int min, max;
cout << "Please type two numbers (such as 46 75) ";
cin >> num1 >> num2;
max = maximum(num1, num2); // Assign the return
min = minimum(num1, num2); // value of each
// function to variables.
```

```
cout << “The minimum number is “ << min << “\n”;

cout << “The maximum number is “ << max << “\n”;

return 0;

}

int maximum(int num1, int num2)

{

int max; // Local to this function only.

max = (num1 > num2) ? (num1) : (num2);

return (max);

}

int minimum(int num1, int num2)

{

int min; // Local to this function only.

min = (num1 < num2) ? (num1) : (num2);

return (min);

}
```

Oto przykładowe wyjście z tego programu:

```
The minimum number is 55

The maximum number is 72
```

Jeśli użytkownik wpisze tę samą liczbę, minimalna i maksymalna to to samo. Te dwie funkcje można przekazać dowolne dwie wartości całkowite. W tak prostym przykładzie, jak ten, użytkownik z pewnością już wie, która liczba jest niższa lub wyższa. Celem takiego przykładu jest pokazanie, jak kodować zwracane wartości. Możesz użyć podobnych funkcji w bardziej użytecznej aplikacji, na przykład znaleźć najlepiej opłacanego pracownika z pliku dyskowego listy płac.

Prototypy funkcji

Słowo prototyp jest czasem definiowane jako model. W C++ prototyp funkcji modeluje rzeczywistą funkcję. Przed ukończeniem badania funkcji, parametrów i zwracanych wartości należy zrozumieć, jak prototypować każdą funkcję w programie. C++ wymaga prototypowania wszystkich funkcji w programie. Podczas prototypowania informujesz C++ o typach parametrów funkcji i ewentualnej wartości zwracanej. Aby prototypować funkcję, skopiuj linię definicji funkcji na górę programu (bezpośrednio przed lub po linii #include <iostream.h>). Umieść średnik na końcu linii definicji funkcji, a otrzymasz prototyp. Wiersz definicji (pierwszy wiersz funkcji) zawiera typ zwracany, nazwę funkcji i typ każdego argumentu, więc prototyp funkcji służy jako model następującej po nim funkcji. Jeśli funkcja nie zwraca wartości lub jeśli do tej funkcji nie przekazano żadnych argumentów, nadal należy ją prototypować. Umieść słowo kluczowe void zamiast typu zwracanego parametru lub parametrów. main() jest jedyną funkcją, której nie trzeba prototypować, ponieważ jest to samo-prototypowanie; co

oznacza, że main() nie jest wywoływana przez inną funkcję. Za pierwszym razem, gdy main() pojawi się w twoim programie (zakładając, że zastosujesz standardowe podejście i uczynisz main() pierwszą funkcją programu), zostanie on wykonany. Jeśli funkcja nic nie zwraca, void musi być typem zwracanym. Unieważnij nawiasy argumentów prototypów funkcji bez argumentów. Wszystkie funkcje muszą pasować do ich prototypów. Wszystkie główne () funkcje w tej książce zwróciły 0. Dlaczego? Teraz wiesz wystarczająco dużo, aby odpowiedzieć na to pytanie. Ponieważ main() jest samo-prototypowaniem, a ponieważ słowo kluczowe void nigdy nie pojawiło się przed main() w tych programach, C++ przyjął typ int. Wszystkie funkcje C++ prototypowane jako zwracające int lub te bez prototypu zwracanego typu danych przyjmują int. Jeśli nie chcesz wstawiać return 0; na końcu funkcji main() musisz wstawić void przed main() jak w: void main() // main() auto-prototypy, aby nic nie zwracać. Możesz spojrzeć na instrukcję i stwierdzić, czy jest to prototyp, czy definicja funkcji (pierwszy wiersz funkcji) po średniku na końcu. Wszystkie prototypy, chyba że wykonasz autoprototyp główny (), kończą się średnikiem.

Prototyp bezpieczeństwa

Prototypowanie chroni przed błędami programowania. Załóżmy, że piszesz funkcję, która oczekuje dwóch argumentów: liczby całkowitej, po której następuje wartość zmiennoprzecinkowa. Oto pierwszy wiersz takiej funkcji:

my_fun (int num, liczba zmiennoprzecinkowa)

Co jeśli przekażesz niepoprawne typy danych do my_fun()? Jeśli miałbyś wywołać tę funkcję, przekazując jej dwa literały, zmiennoprzecinkowy, po którym następuje liczba całkowita, jak w

my_fun (23,43, 5); // Wywołaj funkcję my_fun ().

funkcja nie otrzyma poprawnych parametrów. Oczekuje liczby całkowitej, po której następuje liczba zmiennoprzecinkowa, ale zrobiłeś coś przeciwnego i wysłałeś jej liczbę zmiennoprzecinkową, po której następuje liczba całkowita. W zwykłych programach C niedopasowane argumenty, takie jak te, nie generują komunikatu o błędzie, nawet jeśli dane nie są poprawnie przekazywane. C++ wymaga prototypów, więc nie można wysłać niewłaściwych typów danych do funkcji (lub oczekiwać, że zostanie zwrócony niewłaściwy typ danych). Prototypowanie poprzedniej funkcji powoduje:

void my_fun (int num, liczba zmiennoprzecinkowa); // Prototyp

Robiąc to, każesz kompilatorowi sprawdzić dokładność tej funkcji. Informujesz kompilator, aby niczego nie oczekiwał po instrukcji return, nawet 0, (ze względu na słowo kluczowe void) i oczekiwał liczby całkowitej, po której następuje nawias zmiennoprzecinkowy. Jeśli złamiesz którąkolwiek z zasad prototypu, kompilator poinformuje cię o problemie i możesz go naprawić.

Prototypuj wszystkie funkcje

Powinieneś prototypować każdą funkcję w swoim programie. Jak opisano powyżej, prototyp definiuje (dla pozostałej części programu), jakie funkcje mają następować, ich typy zwracane i typy parametrów. Powinieneś także prototypować wbudowane funkcje C++, takie jak printf () i scanf (), jeśli ich używasz.

Pomyśl o tym, jak prototypujesz printf (). Nie zawsze przekazujesz mu te same typy parametrów, ponieważ drukujesz różne dane przy każdym printf (). Pisanie prototypowych funkcji jest łatwe: prototyp jest zasadniczo pierwszą linią funkcji. Funkcje prototypowania, których nie piszesz, mogą wydawać się trudne, ale tak nie jest - robiłeś to już z każdym programem w tej książce! Projektanci C++ zdali sobie sprawę, że wszystkie funkcje muszą być prototypowane. Zdali sobie również sprawę, że nie można prototypować wbudowanych funkcji, więc zrobili to za Ciebie i umieścili prototypy w plikach

nagłówkowych na dysku. Dołączasz prototypy printf () i scanf () do każdego programu, który używał ich w tej książce, z następującą instrukcją:

#include <stdio.h>

W pliku stdio.h znajduje się prototyp wielu funkcji wejścia i wyjścia C++. Dysponując prototypami tych funkcji, upewniasz się, że nie można im przekazać złych wartości. Jeśli ktoś próbuje przekazać niepoprawne wartości, C++ łapie problem. Ponieważ printf() i scanf() nie są bardzo często używane w C++, operatorzy cout i cin mają własny plik nagłówka o nazwie iostream.h, który, jak widzieliście, jest również zawarty w programach tej książki. Plik iostream.h tak naprawdę nie zawiera prototypów dla cout i cin, ponieważ są one operatorami, a nie funkcjami, ale iostream.h zawiera kilka niezbędnych definicji, aby cout i cin działały. Pamiętaj też, że iomanip.h musi być dołączony, jeśli używasz modyfikatora setw lub setprecision w cout. Za każdym razem, gdy korzystasz z nowej wbudowanej funkcji C++ lub operatora manipulującego, sprawdź instrukcję kompilatora, aby znaleźć nazwę pliku prototypu, który ma zostać uwzględniony. Prototypowanie jest głównym powodem, dla którego powinieneś zawsze dołączać pasujący plik nagłówkowy, gdy używasz wbudowanych funkcji C++. Funkcja strcpy (), którą widziałeś w poprzednich rozdziałach, wymaga następującego wiersza:

#include <string.h>

To jest plik nagłówka funkcji strcpy (). Bez tego program nie działa.

Przykłady

1. Prototypuj wszystkie funkcje we wszystkich programach oprócz main(). Nawet main() musi być prototypowany, jeśli nic nie zwraca (nawet 0). Poniższy program zawiera dwa prototypy: jeden dla main(), ponieważ nic nie zwraca, a drugi dla wbudowanych funkcji printf() i scanf().

```
// Nazwa pliku: C19PRO1.CPP

// Oblicza podatek od sprzedaży

#include <stdio.h> // Wbudowane funkcje prototypu.

void main (void);

void main (void)

{

float total_sale;

float tax_rate = .07; // Assume seven percent

// tax rate.

printf(“What is the sale amount? “);

scanf(“ %f”, &total_sale);

total_sale += (tax_rate * total_sale);

printf(“The total sale is %.2f”, total_sale);

return; // No 0 required!

}
```

Zauważ, że instrukcja return main() wymagała tylko średnika po niej. Dopóki prototypujesz main() za pomocą typu void return, ostatnia linia w main() może być return; zamiast wpisywać return 0; za każdym razem.

2. Poniższy program prosi użytkownika o numer w main() i przekazuje go do ascii(). Funkcja ascii() zwraca znak ASCII pasujący do numeru użytkownika. Ten przykład ilustruje typ zwracanego znaku. Funkcje mogą zwracać dowolny typ danych.

```
// Nazwa pliku: C19ASC.CPP

// Drukuje znak ASCII numeru użytkownika.

// Następują prototypy.

#include <iostream.h>

char ascii (int num);

void main()

{

int num;

char asc_char;

cout << “Enter an ASCII number? “;

cin >> num;

asc_char = ascii(num);

cout << “The ASCII character for “ << num

<< “ is “ << asc_char;

return;

}

char ascii(int num)

{

char asc_char;

asc_char = char(num); // Type cast to a character.

return (asc_char);

}
```

Dane wyjściowe z tego programu są następujące:

```
Enter an ASCII number? 67

The ASCII character for 67 is C
```

3. Załóżmy, że musisz obliczyć wynagrodzenie netto dla firmy. Mnożycie godziny przepracowane przez wynagrodzenie godzinowe, a następnie odliczacie podatki, aby obliczyć wynagrodzenie netto. Poniższy

program zawiera funkcję, która robi to za Ciebie. Wymaga trzech argumentów: przepracowanych godzin, wynagrodzenia za godzinę i stawki podatkowej (w postaci dziesiętnej zmiennoprzecinkowej, takiej jak 0,30 na 30 procent). Funkcja zwraca wynagrodzenie netto. Program wywołujący main() testuje funkcję, wysyłając do niej trzy różne wartości listy płac i drukując trzy wartości zwrotu

```
// Nazwa pliku: C19NPAY.CPP

// Definiuje funkcję obliczającą wynagrodzenie netto.

#include <iostream.h> // Potrzebny do cout i cin.

void main (void);

float netpayfun(float hours, float rate, float taxrate);

void main(void)

{

float net_pay;

net_pay = netpayfun(40.0, 3.50, .20);

cout << “The pay for 40 hours at $3.50/hr., and a 20% “

<< “tax rate is $”;

cout << net_pay << “\n”;

net_pay = netpayfun(50.0, 10.00, .30);

cout << “The pay for 50 hours at $10.00/hr., and a 30% “

<< “tax rate is $”;

cout << net_pay << “\n”;

net_pay = netpayfun(10.0, 5.00, .10);

cout << “The pay for 10 hours at $5.00/hr., and a 10% “

<< “ tax rate is $”;

cout << net_pay << “\n”;

return;

}

float netpayfun(float hours, float rate, float taxrate)

{

float gross_pay, taxes, net_pay;

gross_pay = (hours * rate);

taxes = (taxrate * gross_pay);

net_pay = (gross_pay - taxes);
```

return (net_pay);

}

Ćwiczenia

1. Napisz program, który zawiera dwie funkcje. Pierwsza funkcja zwraca kwadrat liczby całkowitej przekazanej do niej, a druga funkcja zwraca kostkę. Prototypuj main(), więc nie musisz zwracać wartości.

2. Napisz funkcję, która zwraca obszar podwójnej precyzji koła, biorąc pod uwagę, że jest do niego przekazywany promień podwójnej precyzji. Wzór na obliczenie powierzchni koła to

powierzchnia = 3,14159 * (promień * promień)

3. Napisz funkcję, która zwraca wartość wielomianu przy tej formule:

9x4 + 15x2 + x1

Załóżmy, że x jest przekazywany z main() i jest dostarczany przez użytkownika.

Podsumowanie

Nauczyłeś się, jak budować własną kolekcję funkcji. Pisząc funkcję, możesz chcieć użyć jej w więcej niż jednym programie - nie ma potrzeby ponownego tworzenia koła. Wielu programistów pisze przydatne funkcje i używa ich w więcej niż jednym programie. Teraz rozumiesz znaczenie funkcji prototypowania. Powinieneś prototypować wszystkie swoje funkcje i dołączyć odpowiedni plik nagłówka, gdy używasz jednej z wbudowanych funkcji C++. Ponadto, gdy funkcja zwraca wartość inną niż liczba całkowita, musisz prototypować, aby C++ rozpoznał zwracaną wartość niecałkowitą.

Domyślne argumenty i przeciążenie funkcji

Wszystkie funkcje, które otrzymują argumenty, nie muszą być przesyłane wartości. C++ umożliwia określenie domyślnych list argumentów. Możesz pisać funkcje, które przyjmują wartości argumentów, nawet jeśli nie przekażesz im żadnych argumentów. C++ umożliwia także pisanie więcej niż jednej funkcji o tej samej nazwie. Nazywa się to funkcjami przeciążania. Dopóki ich listy argumentów się różnią, funkcje są zróżnicowane przez C++. Ta część zawiera następujące informacje:

♦ Domyślne listy argumentów

♦ Przeciążone funkcje

♦ Zmiana nazwy

Domyślne listy argumentów i przeciążone funkcje nie są dostępne w zwykłym C. C++ rozszerza możliwości twoich programów, zapewniając te procedury oszczędzające czas.

Domyślne listy argumentów

Załóżmy, że piszesz program, który musi wyświetlić komunikat na ekranie przez krótki okres czasu. Na przykład przekazujesz funkcji komunikat o błędzie przechowywany w tablicy znaków, a funkcja drukuje komunikat o błędzie przez pewien okres czasu. Prototypem takiej funkcji może być:

void pr_msg (char note []);

Dlatego, aby zażądać, aby pr_msg() wyświetlił wiersz „Włącz drukarkę”, wywołujesz go w ten sposób:

pr_msg („Włącz drukarkę”); // Przekazuje wiadomość do wydrukowania.

To polecenie drukuje komunikat „Włącz drukarkę” przez około pięć sekund. Aby zażądać, aby pr_msg() wydrukował wiersz „Naciśnij dowolny klawisz, aby kontynuować ...”, wywołujesz go w ten sposób:

pr_msg („Naciśnij klawisz, aby kontynuować ...”); // Przekazuje wiadomość.

Gdy piszesz więcej o programie, zaczynasz zdawać sobie sprawę, że drukujesz jeden komunikat, na przykład komunikat „Włącz drukarkę”, częściej niż jakikolwiek inny komunikat. Wygląda na to, że funkcja pr_msg() odbiera ten komunikat znacznie częściej niż jakikolwiek inny. Może tak być w przypadku pisania programu, który wydrukował wiele raportów do drukarki. Nadal będziesz używać pr_msg() do innych opóźnionych wiadomości, ale najczęściej używany jest komunikat „Włącz drukarkę”. Zamiast ciągle wywoływać funkcję, wpisując ten sam komunikat za każdym razem, możesz skonfigurować prototyp dla pr_msg(), aby domyślnie „Włącz drukarkę” w ten sposób:

void pr_msg (char note [] = ”Włącz drukarkę”); // Prototyp

Po prototypowaniu pr_msg() przy użyciu domyślnej listy argumentów, C++ zakłada, że chcesz przekazać funkcję „Włącz drukarkę”, chyba że przesłonisz domyślną, przekazując do niej coś innego. Na przykład w main () wywołujesz pr_msg () w ten sposób:

pr_msg (); // C++ zakłada, że masz na myśli „Włącz drukarkę”.

Ułatwia to pracę programistyczną. Ponieważ przez większość czasu chcesz, aby pr_msg() wypisał „Włącz drukarkę”, domyślna lista argumentów zajmuje się wiadomością i nie musisz przekazywać wiadomości podczas wywoływania funkcji. Jednak te kilka razy, kiedy chcesz przekazać coś innego, po prostu przekaż inną wiadomość. Na przykład, aby pr_msg() wypisał „Niepoprawną wartość”, wpisz:

pr_msg („Niepoprawna wartość”); // Przekaż nową wiadomość.

WSKAZÓWKA: Za każdym razem, gdy wywołujesz funkcję kilka razy i okazuje się, że przekazujesz tej funkcji te same parametry przez większość czasu, rozważ użycie domyślnej listy argumentów.

Wiele domyślnych argumentów

Możesz podać więcej niż jeden domyślny argument na liście prototypów. Oto prototyp funkcji z trzema domyślnymi argumentami:

float funk1 (int i = 10, float x = 7,5, char c = ’A’);

Istnieje kilka sposobów na wywołanie tej funkcji. Oto kilka próbek:

funk1();

Zakładane są wszystkie wartości domyślne.

funk1(25);

Do argumentu liczby całkowitej wysyłana jest wartość 25, a dla pozostałych przyjmuje się wartości domyślne.

funk1 (25, 31,25);

25 jest wysyłane do argumentu liczb całkowitych, 31.25 do argumentu zmiennoprzecinkowego i dla argumentu znakowego przyjmuje się domyślną wartość „A”.

UWAGA: Jeśli tylko niektóre argumenty funkcji są argumentami domyślnymi, te domyślne argumenty muszą pojawić się po lewej stronie listy argumentów. Po lewej stronie argumentów domyślnych nie mogą pojawiać się argumenty domyślne. To jest nieprawidłowy domyślny prototyp argumentu:

float func2 (int i = 10, float x, char c, long n = 10.232);

Jest to nieprawidłowe, ponieważ domyślny argument pojawia się po lewej stronie argumentu innego niż domyślny. Aby to naprawić, musisz przenieść dwa domyślne argumenty w skrajną lewą stronę (początek) listy argumentów. Dlatego, przestawiając prototyp (i wynikające z niego wywołania funkcji) w następujący sposób, C++ umożliwia osiągnięcie tego samego celu, jak przy poprzedniej linii:

float func2 (float x, char c, int i = 10, long n = 10.232);

Przykłady

1. Oto kompletny program ilustrujący funkcję drukowania komunikatów opisaną wcześniej. Funkcja main() po prostu wywołuje funkcję opóźnionego drukowania wiadomości trzy razy, za każdym razem przekazując jej inny zestaw list argumentów.

```
// Nazwa pliku: C20DEF1.CPP

// Ilustruje domyślną listę argumentów.

#include <iostream.h>

void pr_msg (char note [] = ”Włącz drukarkę”); // Prototyp.

void main ()

{

pr_msg(); // Prints default message.
```

```
pr_msg(“A new message”); // Prints another message.

pr_msg(); // Prints default message again.

return;

}

void pr_msg(char note[]) // Only prototype contains defaults.

{

long int delay;

cout << note << “\n”;

for (delay=0; delay<500000; delay++)

{ ; /* Do nothing while waiting */ }

return;

}
```

Program generuje następujące dane wyjściowe:

```
Turn printer on

A new message

Turn printer on
```

Pętla opóźnień powoduje wyświetlanie każdej linii przez kilka sekund lub dłużej, w zależności od szybkości komputera, aż wszystkie trzy linie zostaną wydrukowane.

2. Poniższy program ilustruje użycie domyślnych kilku argumentów. main() wywołuje funkcję de_fun() pięć razy, wysyłając de_fun() pięć zestawów argumentów. Funkcja de_fun() drukuje pięć różnych rzeczy w zależności od listy argumentów main().

```
// Nazwa pliku: C20DEF2.CPP

// Pokazuje domyślną listę argumentów z kilkoma parametrami.

#include <iostream.h>

#include <iomanip.h>

void de_fun(int i=5, long j=40034, float x=10.25,

char ch=’Z’, double d=4.3234); // Prototype

void main()

{

de_fun(); // All defaults used.

de_fun(2); // First default overridden.

de_fun(2, 75037); // First and second default overridden.
```

```
de_fun(2, 75037, 35.88); // First, second, and third

de_fun(2, 75037, 35.88, 'G'); // First, second, third,

// and fourth

de_fun(2, 75037, 35.88, 'G', .0023); // No defaulting.

return;

}

void de_fun(int i, long j, float x, char ch, double d)

{

cout << setprecision(4) << "i: " << i << " " << "j: " << j;

cout << " x: " << x << " " << "ch: " << ch;

cout << " d: " << d << "\n";

return;

}
```

Oto wynik tego programu:

```
i: 5 j: 40034 x: 10.25 ch: Z d: 4.3234

i: 2 j: 40034 x: 10.25 ch: Z d: 4.3234

i: 2 j: 75037 x: 10.25 ch: Z d: 4.3234

i: 2 j: 75037 x: 35.88 ch: Z d: 4.3234

i: 2 j: 75037 x: 35.88 ch: G d: 4.3234

i: 2 j: 75037 x: 35.88 ch: G d: 0.0023
```

Zauważ, że każde wywołanie de_fun () daje inne wyjście, ponieważ main() wysyła inny zestaw parametrów za każdym razem, gdy main () wywołuje de_fun().

Przeciążone funkcje

W przeciwieństwie do zwykłego C, C++ pozwala mieć więcej niż jedną funkcję o tej samej nazwie. Innymi słowy, możesz mieć trzy funkcje o nazwie abs() w tym samym programie. Funkcje o tych samych nazwach nazywane są funkcjami przeciążonymi. C++ wymaga, aby każda przeciążona funkcja różniła się listą argumentów. Funkcje przeciążone pozwalają mieć podobne funkcje, które działają na różnych typach danych. Załóżmy na przykład, że napisałeś funkcję, która zwróciła wartość bezwzględną dowolnej liczby, którą do niej przekazałeś. Wartość bezwzględna liczby jest jej dodatnim odpowiednikiem. Na przykład wartość bezwzględna 10,25 wynosi 10,25, a wartość bezwzględna -10,25 wynosi 10,25. Wartości bezwzględne są używane w obliczeniach odległości, temperatury i masy. Różnica w wadze dwojga dzieci jest zawsze dodatnia. Jeśli Joe waży 65 funtów, a Mary waży 55 funtów, ich różnica wynosi dodatnie 10 funtów. Możesz odjąć 65 od 55 (–10) lub 55 od 65 (+10), a różnica masy jest zawsze wartością bezwzględną wyniku. Załóżmy, że trzeba napisać funkcję wartości bezwzględnej dla liczb całkowitych i funkcję wartości bezwzględnej dla liczb zmiennoprzecinkowych. Bez przeciążenia funkcji potrzebne są te dwie funkcje:

```
int iabs(int i) // Returns absolute value of an integer.
{
if (i < 0)
{ return (i * -1); } // Makes positive.
else
{ return (i); } // Already positive.
}
float fabs(float x) // Returns absolute value of a float.
{
if (x < 0.0)
{ return (x * -1.0); } // Makes positive.
else
{ return (x); } // Already positive.
}
```

Bez przeciążania, jeśli miałeś zmiennoprzecinkową zmienną, dla której potrzebna była wartość bezwzględna, przekazujesz ją do funkcji fabs () jak w:

```
ans = fabs(weight);
```

Jeśli potrzebujesz wartości bezwzględnej zmiennej całkowitej, przekazujesz ją do funkcji iabs () jak w:

```
ians = iabs(age);
```

Ponieważ kod tych dwóch funkcji różni się tylko listami parametrów, są idealnymi kandydatami na przeciążone funkcje. Wywołaj obie funkcje abs (), prototypuj obie i koduj każdą z nich osobno w swoim programie. Po przeciążeniu dwóch funkcji (z których każda działa na dwóch różnych typach parametrów o tej samej nazwie), przekazujesz wartość zmiennoprzecinkową lub liczbę całkowitą do abs (). Kompilator C++ określa, którą funkcję chcesz wywołać.

UWAGA: Jeśli dwie lub więcej funkcji różnią się tylko typami zwracanych danych, C++ nie może ich przeciążać. Dwie lub więcej funkcji, które różnią się jedynie typami zwracanych znaków, muszą mieć różne nazwy i nie mogą być przeciążone.

Ten proces znacznie upraszcza programowanie. Zamiast pamiętać kilka różnych nazw funkcji, musisz zapamiętać tylko jedną nazwę funkcji. C++ przekazuje argumenty do właściwej funkcji.

UWAGA: C++ używa manglingu nazw w celu realizacji przeciążonych funkcji. Zrozumienie manglingu nazw pomaga Ci stać się zaawansowanym programistą C++. Kiedy C++ zdaje sobie sprawę, że przeciążasz dwie lub więcej funkcji o tej samej nazwie, każda z nich różni się tylko jego lista parametrów, C++ zmienia nazwę funkcji i dodaje litery na końcu nazwy funkcji, które pasują do parametrów. Różne kompilatory C++ robią to inaczej. Aby zrozumieć, co robi kompilator, weź funkcję wartości bezwzględnej opisaną wcześniej. C++ może zmienić funkcję wartości bezwzględnej liczby

całkowitej na absi (), a funkcję wartości bezwzględnej zmiennoprzecinkowej na absf (). Gdy wywołujesz funkcję za pomocą tego wywołania funkcji:

ians = abs (age);

C++ określa, że chcesz wywołać funkcję absi (). O ile wiesz, C++ nie zmienia nazw; nigdy nie widzisz różnic nazw w kodzie źródłowym swojego programu. Jednak kompilator wykonuje zmianę nazwy, dzięki czemu może śledzić różne funkcje o tej samej nazwie.

Przykłady

1. Oto pełny program wartości bezwzględnej opisany w poprzednim tekście. Zauważ, że obie funkcje są prototypowane. (Dwa prototypy sygnalizują C++, że musi wykonać kodowanie nazw, aby określić prawidłowe nazwy funkcji do wywołania.)

```
// Nazwa pliku: C20OVF1.CPP

// Przeciąża dwie funkcje wartości bezwzględnej.

#include <iostream.h> // Prototyp cout i cin.

#include <iomanip.h> // Prototype setprecision (2).

int abs (int i); // abs () jest dwukrotnie przeciążony

float abs (float x); // jak pokazują te prototypy.

void main ()

{

int ians; // To hold return values.

float fans;

int i = -15; // To pass to the two overloaded functions.

float x = -64.53;

ians = abs(i); // C++ calls the integer abs().

cout << “Integer absolute value of -15 is “ << ians << “\n”;

fans = abs(x); // C++ calls the floating-point abs().

cout << “Float absolute value of -64.53 is “ <<

setprecision(2) << fans << “\n”;

// Notice that you no longer have to keep track of two

// different names. C++ calls the appropriate

// function that matches the parameters.

return;

}
```

```
int abs(int i) // Integer absolute value function
{
if (i < 0)
{ return (i * -1); } // Makes positive.
else
{ return (i); } // Already positive.
}
float abs(float x) // Floating-point absolute value function
{
if (x < 0.0)
{ return (x * -1.0); } // Makes positive.
else
{ return (x); } // Already positive.
}
```

Dane wyjściowe z tego programu są następujące:

```
Integer absolute value of -15 is 15
Float absolute value of -64.53 is 64.53
```

2. Gdy będziesz pisać coraz więcej programów C++, zobaczysz wiele zastosowań przeciążonych funkcji. Poniższy program to program demonstracyjny pokazujący, w jaki sposób można zbudować własne funkcje wyjściowe dostosowane do potrzeb. main() wywołuje trzy funkcje o nazwie output(). Za każdym razem, gdy jest wywoływane, main () przekazuje inną funkcję do funkcji. Kiedy main() przekazuje output() ciąg, output() drukuje ciąg, sformatowany do szerokości (przy użyciu manipulatora setw() opisanego później) o długości 30 znaków. Kiedy main() przekazuje output() liczbę całkowitą, output() drukuje liczbę całkowitą o szerokości pięciu. Kiedy main() przekazuje output() wartość zmiennoprzecinkową, output() wypisuje wartość na dwa miejsca po przecinku i uogólnia dane wyjściowe różnych typów danych. Nie musisz formatować własnych danych. output () poprawnie formatuje dane i musisz zapamiętać tylko jedną nazwę funkcji, która wyprowadza wszystkie trzy typy danych.

```
// Nazwa pliku: C20OVF2.CPP
// Wyprowadza trzy różne typy
// dane o tej samej nazwie funkcji.
#include <iostream.h>
#include <iomanip.h>
void output (char []); // Prototypy dla przeciążonych funkcji.
void output (int i);
```

```
void wyjście (float x);
void main ()
{
char name[] = “C++ By Example makes C++ easy!”;
int ivalue = 2543;
float fvalue = 39.4321;
output(name); // C++ chooses the appropriate function.
output(ivalue);
output(fvalue);
return;
}
void output(char name[])
{
cout << setw(30) << name << “\n”;
// The width truncates string if it is longer than 30.
return;
}
void output(int ivalue)
{
cout << setw(5) << ivalue << “\n”;
// Just printed integer within a width of five spaces.
return;
}
void output(float fvalue)
{
cout << setprecision(2) << fvalue << “\n”;
// Limited the floating-point value to two decimal places.
return;
}
```

Oto wynik tego programu:

C++ By Example makes C++ easy!

2543

39.43

Każdy z trzech wierszy, zawierający trzy różne wiersze informacji, został wydrukowany z tym samym wywołaniem funkcji.

Ćwiczenia

1. Napisz program, który zawiera dwie funkcje. Pierwsza funkcja zwraca kwadrat liczby całkowitej przekazanej do niej, a druga funkcja zwraca kwadrat liczby zmiennoprzecinkowej przekazanej do niej.

2. Napisz program, który oblicza wynagrodzenie netto na podstawie wartości wpisanych przez użytkownika. Zapytaj użytkownika o przepracowane godziny, stawkę za godzinę i stawkę podatkową. Ponieważ większość pracowników pracuje 40 godzin tygodniowo i zarabia 5,00 USD za godzinę, użyj tych wartości jako wartości domyślnych w funkcji obliczającej wynagrodzenie netto. Jeśli użytkownik naciśnie klawisz Enter w odpowiedzi na pytania, użyj wartości domyślnych.

Podsumowanie

Domyślne listy argumentów i przeciążone funkcje przyspieszają czas programowania. Nie musisz już określać wartości typowych argumentów. Nie musisz pamiętać kilku różnych nazw tych funkcji, które wykonują podobne procedury i różnią się jedynie typami danych. Pozostała część książki omawia wcześniejsze koncepcje, dzięki czemu możesz korzystać z oddzielnych, modułowych funkcji i danych lokalnych. Jesteś gotowy, aby dowiedzieć się więcej o tym, jak C++ wykonuje dane wejściowe i wyjściowe.

Wejście / wyjście urządzenia i znaki

W przeciwieństwie do wielu języków programowania, C++ nie zawiera poleceń wejściowych ani wyjściowych. C++ jest niezwykle przenośnym językiem; program C++, który kompiluje i działa na jednym komputerze, jest w stanie także skompilować i uruchomić na innym komputerze. Większość niezgodności między komputerami wynika z ich mechaniki wejścia / wyjścia. Każde inne urządzenie wymaga innej metody wykonywania operacji we / wy (wejście / wyjście). Umieszczając wszystkie możliwości I / O we wspólnych funkcjach dostarczanych z kompilatorem każdego komputera, a nie w instrukcjach C++, projektanci C++ upewnili się, że programy nie są powiązane z konkretnym sprzętem dla danych wejściowych i wyjściowych. Kompilator należy zmodyfikować dla każdego komputera, dla którego został napisany. Dzięki temu kompilator działa z określonym komputerem i jego urządzeniami. Autorzy kompilatora zapisują funkcje We / Wy dla każdej maszyny; gdy program C++ zapisuje znak na ekranie, działa tak samo, niezależnie od tego, czy masz kolorowy ekran komputera, czy terminal UNIX X / Windows. W tym rozdziale przedstawiono dodatkowe sposoby wprowadzania i wyprowadzania danych oprócz funkcji cin i cout, które widziałeś w całej książce. Zapewniając funkcje we / wy oparte na znakach, C++ zapewnia podstawowe funkcje we / wy potrzebne do pisania zaawansowanych procedur wprowadzania danych i drukowania. Ta część zapozna cię ze

♦ Strumieniem wejścia i wyjścia

♦ Przekierowaniem I / O

♦ Drukowaniem na drukarce

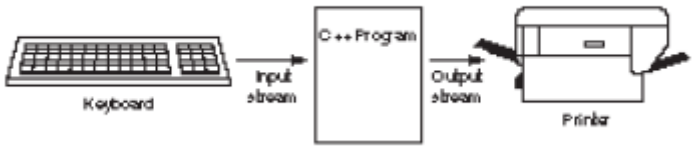
♦ Funkcjami we / wy znaków

♦ Buforowaniem i niebuforowaniem wejścia / wyjścia

Do czasu zakończenia tej sekcji zrozumiesz podstawowe wbudowane funkcje we / wy dostępne w C++. Wprowadzanie i wyprowadzanie znaków, po jednym znaku na raz, może brzmieć jak wolna metoda We / Wy. Wkrótce zorientujesz się, że we / wy postaci faktycznie umożliwiają tworzenie potężniejszych funkcji we / wy niż cin i cout.

Strumień i charakter we / wy

C++ wyświetla wszystkie dane wejściowe i wyjściowe jako strumienie znaków. Niezależnie od tego, czy Twój program odbiera dane z klawiatury, pliku na dysku, modemu lub myszy, C++ wyświetla tylko strumień znaków. C++ nie musi wiedzieć, jaki typ urządzenia dostarcza dane wejściowe; system operacyjny obsługuje specyfikację urządzenia. Projektanci C++ chcą, aby twoje programy działały na znakach danych bez względu na fizyczną metodę. Ten strumień I / O oznacza, że możesz używać tych samych funkcji do odbierania danych z klawiatury jak z modemu. Tych samych funkcji można użyć do zapisu do pliku dyskowego, drukarki lub ekranu. Oczywiście musisz mieć jakiś sposób na przekierowanie strumienia wejściowego lub wyjściowego do odpowiedniego urządzenia, ale funkcje we / wy każdego programu działają w podobny sposób. Rysunek ilustruje tę koncepcję.



Znak specjalny nowej linii: / n

Przenośność jest kluczem do sukcesu C++. Niewiele firm ma zasoby, aby przepisać każdy program, którego używają po zmianie sprzętu komputerowego. Potrzebują języka programowania, który działa na wielu platformach (kombinacje sprzętowe). C++ osiąga prawdziwą przenośność lepiej niż prawie jakikolwiek inny język programowania.

Jest to spowodowane przenośnością, że C++ używa ogólnego znaku nowej linii, \n, a nie specyficznych sekwencji powrotu karetki i sekwencji wstawiania wiersza, których używają inne języki. Właśnie dlatego C++ używa \t dla tab, a także innych znaków kontrolnych używanych w funkcjach I / O. Jeśli C++ użyje kodu ASCII do przedstawienia tych znaków specjalnych, twoje programy nie będą przenośne. Napisałbyś program C++ na jednym komputerze i użyłbyś wartości powrotu karetki, takiej jak 12, ale 12 może nie być wartością powrotu karetki na innym komputerze. Używając znaku nowej linii i innych znaków kontrolnych dostępnych w C++, upewniasz się, że twój program jest kompatybilny z każdym komputerem, na którym jest skompilowany. Konkretne kompilatory zastępują rzeczywiste kody komputera kodami sterującymi w twoich programach.

Standardowe urządzenia

Poniżej pokazano listę standardowych urządzeń I / O. C++ zawsze zakłada, że dane wejściowe pochodzą ze standardowego wejścia, co oznacza standardowe urządzenie wejściowe.

Zwykle jest to klawiatura, chociaż można zmienić tę domyślną trasę. C++ zakłada, że wszystkie dane wyjściowe są przesyłane do standardowego wyjścia standardowego.

W słowach stdin i stdout nie ma nic magicznego; jednak wiele osób po raz pierwszy uczy się ich znaczenia w C++.

Opis: Nazwa C++: Nazwa MS-DOS

Ekran: standardowe: CON:

Klawiatura: stdin: CON:

Drukarka: stdprn: PRN: lub LPT1:

Port szeregowy: stdaux: AUX: lub COM1:

Komunikaty o błędach: stderr: CON:

Pliki na dysku: brak: Nazwa pliku

Poświęć chwilę na przestudiowanie tabeli. Może wydawać się mylące, że trzy urządzenia noszą nazwę CON :. MS-DOS rozróżnia urządzenie ekranowe o nazwie CON: (co oznacza konsolę), a urządzenie klawiaturowe o nazwie CON: od kontekstu strumienia danych. Jeśli wyślesz strumień wyjściowy (strumień znaków) do CON :, MS-DOS automatycznie przekieruje go na ekran. Jeśli zażądasz danych wejściowych od CON :, MS-DOS pobiera dane wejściowe z klawiatury. (Te wartości domyślne są prawdziwe, dopóki nie przekierowano tych urządzeń, jak pokazano poniżej.) MS-DOS wysyła również wszystkie komunikaty o błędach na ekran (CON :).

UWAGA: Jeśli chcesz przekierować We / Wy do drugiej drukarki lub portu szeregowego.

Przekierowanie urządzeń z MS-DOS

Powodem, dla którego cout przechodzi na ekran, jest po prostu fakt, że standardowe wyjście jest kierowane na ekran na większości komputerów. Powodem wejścia cin z klawiatury jest to, że większość

komputerów uważa klawiaturę za standardowe urządzenie wejściowe, standardowe. Po skompilowaniu programu C++ nie wysyła danych na ekran ani nie pobiera ich z klawiatury. Zamiast tego program wysyła dane wyjściowe na standardowe wyjście i odbiera dane wejściowe ze standardowego wejścia. System operacyjny kieruje dane do odpowiedniego urządzenia. MS-DOS umożliwia przekierowanie I / O z ich domyślnych lokalizacji na inne urządzenia za pomocą symbolu przekierowania wyjściowego,> i symbolu przekierowania wejściowego <. Celem naszym nie jest zagłębianie się w przekierowanie systemu operacyjnego. Aby dowiedzieć się więcej na temat obsługi I / O, przeczytaj dobrą książkę na ten temat. Zasadniczo symbol przekierowania wyjścia informuje system operacyjny, że chcesz, aby standardowe wyjście trafiło na urządzenie inne niż domyślny (ekran). Symbol przekierowania wejścia kieruje wejście z klawiatury do innego urządzenia wejściowego. Poniższy przykład ilustruje, jak to się robi w MS-DOS.

Przykłady

1. Załóżmy, że piszesz program, który używa tylko cin i cout dla danych wejściowych i wyjściowych. Zamiast odbierać dane z klawiatury, chcesz, aby program pobierał dane z pliku o nazwie MYDATA. Ponieważ cin odbiera dane wejściowe ze standardowego wejścia, musisz przekierować standardowe wejście. Po skompilowaniu programu w pliku o nazwie MYPGM.EXE możesz przekierować jego dane wejściowe z klawiatury za pomocą następującego polecenia DOS:

C:> MYPGM <MYDATA

Oczywiście możesz podać pełną nazwę ścieżki przed nazwą programu lub nazwą pliku. Istnieje jednak niebezpieczeństwo przekierowania wszystkich takich wyników. Wszystkie dane wyjściowe, w tym monity ekranowe dotyczące wprowadzania danych z klawiatury, trafiają do MYDATA. W większości przypadków jest to prawdopodobnie nie do przyjęcia; nadal chcesz .prompts i niektóre wiadomości, aby przejść do ekranu. W następnej sekcji dowiesz się, jak oddzielić we / wy i wysłać część danych wyjściowych do jednego urządzenia, takiego jak ekran, a resztę do innego urządzenia, takiego jak plik lub drukarka.

2. Możesz także skierować wyjście programu do drukarki, wpisując:

C:> MYPGM> PRN:

Przekieruj wyjście MYPGM do drukarki.

3. Jeśli program wymagał dużo danych wejściowych, a dane te były przechowywane w pliku ANSWERS, możesz zastąpić domyślne urządzenie klawiatury, z którego korzysta cin, jak w:

C:> MYPGM <ODPOWIEDZI

Program czyta z pliku ANSWERS za każdym razem wymagane jest wejście cin .

4. Możesz łączyć symbole przekierowań. Jeśli chcesz wprowadzić dane z pliku dyskowego ODPOWIEDZI i wysłać dane wyjściowe do drukarki, wykonaj następujące czynności:

C:> MYPGM <ODPOWIEDZI> PRN:

WSKAZÓWKA: Możesz skierować dane wyjściowe do drukarki szeregowej lub drugiego portu drukarki równoległej, zastępując PR1: lub LPT2:.

Drukowanie sformatowanego wyjścia na drukarce

Wysyłanie danych wyjściowych programu do drukarki jest łatwe za pomocą funkcji ofstream. Format ofstream to

ofstream device (nazwa_urządzenia);

Poniższe przykłady pokazują, jak połączyć cout i ofstream, aby pisać zarówno na ekranie, jak i drukarce.

Przykład

Poniższy program pyta użytkownika o twoje imię i nazwisko. Następnie drukuje imię i nazwisko, najpierw na drukarce.

```
// Nazwa pliku: C21FPR1.CPP

// Drukuje nazwę na drukarce.

#include <fstream.h>

void main ()

{

char first[20];

char last[20];

cout << “What is your first name? “;

cin >> first;

cout << “What is your last name? “;

cin >> last;

// Send names to the printer.

ofstream prn(“PRN”);

prn << “In a phone book, your name looks like this: \n”;

prn << last << “, “ << first << “\n”;

return;

}
```

Funkcje we / wy znaków

Ponieważ wszystkie wejścia / wyjścia są w rzeczywistości znakowymi wejściami / wyjściami, C++ zapewnia wiele funkcji, których można używać do wprowadzania i wyprowadzania znaków. Funkcje cout i cin są nazywane sformatowanymi funkcjami We / Wy, ponieważ zapewniają kontrolę formatowania nad wejściem i wyjściem. Funkcje cout i cin nie są funkcjami znakowymi I / O. Nie ma nic złego w używaniu cout do sformatowanego wyjścia, ale cin ma wiele problemów, jak widzieliście. Zobaczysz teraz, jak pisać własne procedury wprowadzania znaków, aby zastąpić cin, a także korzystać z funkcji wyjścia znaków, aby przygotować cię do nadchodzącej sekcji w tej książce na temat plików dyskowych.

Funkcje get() i put()

Najbardziej podstawowymi funkcjami We / Wy znaków są get() i put(). Funkcja put() zapisuje pojedynczy znak na standardowym urządzeniu wyjściowym (ekran, jeśli nie przekierowujesz go z

systemu operacyjnego). Funkcja get() wprowadza pojedynczy znak ze standardowego urządzenia wejściowego (domyślnie klawiatury). Format dla get() to

device.get (char_var);

Urządzenie get() może być dowolnym standardowym urządzeniem wejściowym. Jeśli otrzymywałeś znaki z klawiatury, używasz cin jako urządzenia. Jeśli inicjujesz modem i chcesz odbierać z niego znaki, użyj ofstream, aby otworzyć modem i odczytać z urządzenia. Format put() to

device.put (char_val);

Char_val może być zmienną znakową, wyrażeniem lub stałą. Dane znakowe wyprowadzasz za pomocą put (). Urządzenie może być dowolnym standardowym urządzeniem wyjściowym. Aby napisać znak do drukarki, otwórz PRN za pomocą ofstream.

Przykłady

1. Poniższy program pyta użytkownika o jego lub jego inicjały po jednym znaku. Zauważ, że program używa zarówno cout, jak i put(). Cout jest nadal użyteczny w przypadku sformatowanych danych wyjściowych, takich jak wiadomości wysyłane do użytkownika. Pisanie pojedynczych znaków najlepiej jest osiągnąć za pomocą put(). Program musi wywołać dwie funkcje get() dla każdego wpisanego znaku. Gdy odpowiesz na monit get(), wpisując znak, a następnie klawisz Enter, C++ interpretuje dane wejściowe jako strumień dwóch znaków. get() najpierw otrzymuje literę, którą wpisałeś, a następnie musi otrzymać \ n (nowa linia, dostarczana do C++ po naciśnięciu Enter). Istnieją przykłady, które rozwiązują ten problem podwójnego get ().

```
// Nazwa pliku: C21CH1.CPP

// Wprowadza get() i put().

#include <fstream.h>

void main ()

{

char in_char; // Holds incoming initial.

char first, last; // Holds converted first and last initial.

cout << “What is your first name initial? “;

cin.get(in_char); // Waits for first initial.

first = in_char;

cin.get(in_char); // Ignores newline.

cout << “What is your last name initial? “;

cin.get(in_char); // Waits for last initial.

last = in_char;

cin.get(in_char); // Ignores newline.

cout << “\nHere they are: \n”;
```

```
cout.put(first);

cout.put(last);

return;

}
```

Oto wynik tego programu:

```
What is your first name initial? G

What is your last name initial? P

Here they are:

GP
```

2. Możesz lepiej dodać znak powrotu karetki do miejsca wyjściowego. Aby wydrukować dwa inicjały w dwóch osobnych wierszach, użyj put(), aby wstawić znak nowej linii do cout, tak jak robi to następujący program:

```
// Filename: C21CH2.CPP

// Introduces get() and put() and uses put() to output

newline.

#include <fstream.h>

void main()

{

char in_char; // Holds incoming initial.

char first, last; // Holds converted first and last

// initial.

cout << “What is your first name initial? “;

cin.get(in_char); // Waits for first initial.

first = in_char;

cin.get(in_char); // Ignores newline.

cout << “What is your last name initial? “;

cin.get(in_char); // Waits for last initial.

last = in_char;

cin.get(in_char); // Ignores newline.

cout << “\nHere they are: \n”;

cout.put(first);

cout.put(‘\n’);
```

cout.put(last);

return;

}

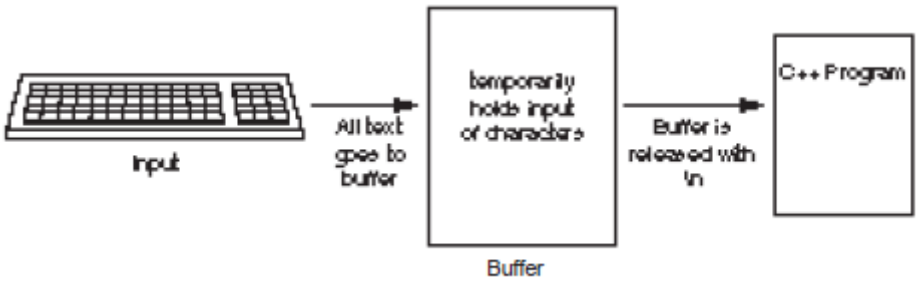
3. Może być łatwiej zdefiniować znak nowej linii jako stałą. Na górze programu masz:

const char NEWLINE = '\ n'

Następnie put () brzmi:

cout.put (NEWLINE);

Niektórzy programiści wolą definiować swoje stałe formatowania znaków i odwoływać się do nich po nazwie. Od Ciebie zależy, czy chcesz użyć tej metody, czy też nadal będziesz używać stałej znaków \ n w put(). Funkcja get() jest buforowaną funkcją wejściową. Podczas wpisywania znaków dane nie trafiają natychmiast do programu, a raczej do bufora. Bufor jest częścią pamięci (i nie ma nic wspólnego z buforami typu PC z wyprzedzeniem) zarządzanym przez C++. Rysunek pokazuje, jak działa ta buforowana funkcja. Kiedy twój program zbliża się do get(), program tymczasowo czeka na wpisanie danych wejściowych. Program nie wyświetla znaków, ponieważ idą do bufora pamięci. Praktycznie nie ma ograniczeń co do wielkości bufora; wypełnia się danymi wejściowymi, dopóki nie naciśniesz Enter. Naciśnięcie klawisza Enter sygnalizuje komputerowi zwolnienie bufora do programu.



Większość komputerów akceptuje buforowane lub niebuforowane dane wejściowe. Funkcja getch() pokazana w dalszej części tego rozdziału nie jest buforowana. Dzięki get() wszystkie dane wejściowe są buforowane. Buforowany tekst wpływa na czas wprowadzania programu. Twój program nie otrzymuje żadnych znaków z get(), dopóki nie naciśniesz Enter. Dlatego jeśli zadajesz pytanie takie jak Czy chcesz ponownie zobaczyć raport (T / N)? i użyj get() do wprowadzenia, użytkownik może nacisnąć Y, ale program nie odbiera danych wejściowych, dopóki użytkownik nie naciśnie Enter. Y i Enter są następnie wysyłane, po jednym znaku na raz, do programu, w którym przetwarza dane wejściowe. Jeśli chcesz natychmiastowej odpowiedzi na wpisanie przez użytkownika (np. INKEY $ w BASIC pozwala), musisz użyć getch().

WSKAZÓWKA: Korzystając z buforowanych danych wejściowych, użytkownik może wpisać ciąg znaków w odpowiedzi na pętlę za pomocą metody get(), odbierać znaki i poprawiać dane za pomocą klawisza Backspace przed naciśnięciem klawisza Enter. Gdyby dane wejściowe nie były buforowane, Backspace byłby innym znakiem danych.

Przykład

C21CH2.CPP musi odrzucić znak nowej linii. W tym celu przypisano znak wejściowy - od get() - do dodatkowej zmiennej. Oczywiście get() zwraca wartość (wpisany znak). W takim przypadku dopuszczalne jest zignorowanie zwracanej wartości, nie używając znaku zwróconego przez get(). Wiesz, że użytkownik musi nacisnąć klawisz Enter (aby zakończyć wprowadzanie), więc dopuszczalne jest odrzucenie go za pomocą nieużywanego wywołania funkcji get(). Podczas wprowadzania ciągów znaków, takich jak nazwy i zdania, cin pozwala na wprowadzenie tylko jednego słowa na raz. Poniższy ciąg pyta użytkownika o pełne imię i nazwisko w tych dwóch wierszach:

cout << “What are your first and last names? “;

cin >> names; // Receive name in character array names.

Nazwy tablic otrzymują tylko imię; cin ignoruje wszystkie dane na prawo od pierwszego miejsca. Możesz zbudować własną funkcję wprowadzania za pomocą get(), która nie ma ograniczenia na jedno słowo. Jeśli chcesz otrzymać ciąg znaków od użytkownika, na przykład jego imię i nazwisko, możesz wywołać funkcję get_in_str() pokazaną w następnym programie. Funkcja main() definiuje tablicę i pyta użytkownika o nazwę. Po monicie program wywołuje funkcję get_in_str() i buduje tablicę wejściową znak za pomocą get(). Ta funkcja zapętla się, używając pętli while, dopóki użytkownik nie naciśnie klawisza Enter (sygnalizowanego znakiem nowej linii, \n, do C++) lub dopóki nie zostanie wpisana maksymalna liczba znaków. Możesz chcieć użyć tej funkcji we własnych programach. Pamiętaj, aby przekazać mu tablicę znaków i liczbę całkowitą, która zawiera maksymalny rozmiar tablicy (nie chcesz, aby łańcuch wejściowy był dłuższy niż tablica znaków zawierająca t). Gdy formant powraca do funkcji main() (lub dowolnej innej funkcji o nazwie et_in_str()), tablica ma pełne dane wejściowe użytkownika, w tym spacje.

```
// Filename: C21IN.CPP

// Program that builds an input string array using get().

#include <fstream.h>

void get_in_str(char str[], int len);

const int MAX=25; // Size of character array to be typed.

void main()

{

char input_str[MAX]; // Keyboard input fills this.

cout << “What is your full name? “;

get_in_str(input_str, MAX); // String from keyboard

cout << “After return, your name is “ << input_str << “\n”;

return;

}

/ ********************************************** **********

// Poniższa funkcja wymaga ciągu znaków i maksimum

// długość łańcucha zostanie mu przekazana. Akceptuje dane wejściowe
```

```
// z klawiatury i wysyła dane wejściowe z klawiatury w ciągu.

// Po powrocie procedura wywołująca ma dostęp do ciągu.

// ********************************************** **********

void get_in_str(char str[ ], int len)

{

int i = 0; // index

char input_char; // character typed

cin.get(input_char); // Get next character in string.

while (i < (len - 1) && (input_char != '\n'))

{

str[i] = input_char; // Build string a character

i++; // at a time.

cin.get(input_char); // Receive next character in string.

}

str[i] = '\0'; // Make the char array a string.

return;

}
```

UWAGA: Pętla sprawdza długość len - 1, aby zaoszczędzić miejsce na zero zerujące na końcu łańcucha wejściowego.

Funkcje getch() i putch()

Funkcje getch() i putch() nieznacznie różnią się od funkcji We / Wy poprzedniego znaku. Ich format jest podobny do get() i put(); czytają z klawiatury i piszą na ekranie i nie można ich przekierować, nawet z systemu operacyjnego. Formaty getch() i putch() to

```
int_var = getch ();

i

putch (int_var);
```

getch() i putch() nie są standardowymi funkcjami AT&T C++, ale zwykle są dostępne w większości kompilatorów C++. getch () i putch () są funkcjami niebuforowanymi. Funkcja wyjściowa znaku putch () jest funkcją odbicia lustrzanego dla getch(); jest to niebuforowana funkcja wyjściowa. Ponieważ prawie każde wykonane urządzenie wyjściowe, z wyjątkiem ekranu i modemu, jest z natury buforowane, putch() skutecznie robi to samo, co put().

Kolejną różnicą w getch() od innych funkcji wprowadzania znaków jest to, że getch() nie echo znaków wejściowych na ekranie, gdy je odbiera. Kiedy wpisujesz znaki w odpowiedzi na get(), widzisz znaki podczas pisania (gdy są one wysyłane do bufora). Jeśli chcesz zobaczyć znaki otrzymane przez getch(), musisz wykonać getch() za pomocą putch(). Przydaje się echo znaków na ekranie, aby użytkownik mógł

sprawdzić, czy on lub on wpisał poprawnie. Niektórzy programiści chcą, aby użytkownik nacisnął klawisz Enter po odpowiedzi na monit lub po wybraniu z menu. Wydaje im się, że dodatkowy czas poświęcony buforowaniu danych daje użytkownikowi więcej czasu na decyzję, czy on lub on chce udzielić takiej odpowiedzi; użytkownik może nacisnąć klawisz Backspace i poprawić dane wejściowe przed naciśnięciem klawisza Enter. Inni programiści lubią chwytać odpowiedź użytkownika na jednoznaczną odpowiedź, taką jak odpowiedź menu, i natychmiast na nią reagować. Uważają, że naciśnięcie Enter jest dodatkowym i niepotrzebnym obciążeniem dla użytkownika, dlatego używają getch(). Wybór nalezy do ciebie. Powinieneś zrozumieć zarówno buforowane, jak i niebuforowane dane wejściowe, abyś mógł używać obu.

WSKAZÓWKA: Możesz także użyć getche(). getche() jest niebuforowanym wejściem identycznym jak getch(), z tym wyjątkiem, że znaki wejściowe są wyświetlane na ekranie echem (wyświetlane), gdy użytkownik je wpisuje. Użycie getche() zamiast getch() zapobiega konieczności wywoływania funkcji putch() w celu echa wejścia użytkownika na ekran.

Przykład

Poniższy program pokazuje funkcje getch() i putch(). Użytkownik jest proszony o wprowadzenie pięciu liter. Te pięć liter dodaje się (za pomocą pętli for) do tablicy znaków o nazwie litery. Podczas uruchamiania tego programu zauważ, że znaki nie są wyświetlane na ekranie podczas pisania. Ponieważ getch() nie jest buforowany, program faktycznie odbiera każdy znak, dodaje go do tablicy i zapętla ponownie, gdy je wpisujesz. (Gdyby były to buforowane dane wejściowe, program nie wykonałby pętli przez pięć iteracji, dopóki nie naciśniesz Enter.) Druga pętla drukuje pięć liter za pomocą putch(). Trzecia pętla drukuje pięć liter na drukarce za pomocą put().

```
// Nazwa pliku: C21GCH1.CPP

// Używa getch () i putch () dla danych wejściowych i wyjściowych.

#include <fstream.h>

#include <conio.h>

void main ()

{

int ctr; // for loop counter

char letters[5]; // Holds five input characters. No

// room is needed for the null zero

// because this array never will be

// treated as a string.

cout << “Please type five letters... \n”;

for (ctr = 0; ctr < 5; ctr++)

{

letters[ctr] = getch(); // Add input to array.

}
```

```
for (ctr = 0; ctr < 5; ctr++) // Print them to screen.

{

putch(letters[ ctr ]);

}

ofstream prn("PRN");

for (ctr = 0; ctr < 5; ctr++) // Print them to printer.

{

prn.put(letters[ ctr ]);

}

return;

}
```

Po uruchomieniu tego programu nie naciskaj Enter po pięciu literach. Funkcja getch() nie używa Enter. Pętla kończy się automatycznie po piątej literze z powodu niebuforowanego wejścia i pętli for.

Ćwiczenia

1. Napisz program, który poprosi użytkownika o pięć liter i wydrukuje je w odwrotnej kolejności na ekranie, a następnie na drukarce.

2. Napisz miniaturowy program do pisania, używając get() i put(). W pętli zdobywaj znaki, dopóki użytkownik nie naciśnie Enter, podczas gdy on lub ona otrzymuje linię wprowadzania danych przez użytkownika. Napisz wiersz danych wejściowych użytkownika do drukarki. Ponieważ get () jest buforowane, nic nie trafia do drukarki, dopóki użytkownik nie naciśnie Enter na końcu każdego wiersza tekstu. (Użyj funkcji wprowadzania ciągów znaków pokazanej w C21IN.CPP.)

3. Dodaj putch() w pierwszej pętli C21CH1.CPP (pierwszy przykładowy program get () tego rozdziału), aby znaki były wyświetlane na ekranie, gdy użytkownik je wpisuje.

4. Palindrom to słowo lub wyrażenie pisane tak samo do przodu i do tyłu. Dwa przykładowe palindromy to Madam I'm Adam

Napisz program w C++, który poprosi użytkownika o wyrażenie. Twórz dane wejściowe, po jednym znaku, używając funkcji wprowadzania znaków, takiej jak get (). Po uzyskaniu pełnego ciągu (zapisz go w tablicy znaków) określ, czy fraza jest palindromem. Musisz przefiltrować znaki specjalne (niealfabetyczne), przechowując tylko znaki alfabetyczne do drugiej tablicy znaków. Musisz także przekonwertować znaki na wielkie litery podczas ich przechowywania. Pierwszy palindrom staje się:

MADAMIMADAM

Używając jednej lub więcej pętli for lub while, możesz teraz przetestować frazę, aby ustalić, czy jest ona palindromem. Wydrukuj wynik testu na drukarce. Przykładowe dane wyjściowe powinny wyglądać następująco:

„Madam I'm Adam" to palindrom.

Podsumowanie

Teraz powinieneś zrozumieć ogólne metody stosowane przez programy C++ dla danych wejściowych i wyjściowych. Pisząc na standardowych urządzeniach I / O, C++ zapewnia przenośność. Jeśli napiszesz program dla jednego komputera, działa on na innym. Gdyby C++ miał pisać bezpośrednio na określonym sprzęcie, programy nie działałyby na każdym komputerze. Jeśli nadal chcesz korzystać ze sformatowanych funkcji we / wy, takich jak cout, możesz to zrobić. Funkcja ofstream () umożliwia zapisywanie sformatowanych danych wyjściowych na dowolnym urządzeniu, w tym na drukarce. Metody znaków we / wy mogą wydawać się prymitywne i są, ale dają elastyczność w budowaniu i tworzeniu własnych funkcji wejściowych. Zademonstrowano jedną z najczęściej używanych funkcji C++, funkcję we / wy znaków budujących łańcuchy. Ciągowe funkcje We / Wy bazują na przedstawionych tutaj zasadach. Będziesz zaskoczony szerokimi funkcjami manipulacji znakami i ciągami dostępnymi również w tym języku.

Funkcje znakowe, łańcuchowe i numeryczne

C++ zapewnia wiele wbudowanych funkcji oprócz funkcji cout, getch() i strcpy(), które widziałeś do tej pory. Te wbudowane funkcje zwiększają produktywność i oszczędzają czas programowania. Nie musisz pisać tyle kodu, ponieważ wbudowane funkcje wykonują dla Ciebie wiele przydatnych zadań.

Tu zapoznasz się z

♦ Funkcjami konwersji znaków

♦ Funkcjami testowania znaków i ciągów

♦ Funkcjami manipulacji ciągami

♦ Łańcuchowymi funkcjamie we / wy

♦ Funkcjami matematycznymi, trygonometrycznymi logarytmicznymi

♦ Przetwarzaniem liczb losowych

Funkcje znaków

W tej sekcji opisano wiele funkcji znaków dostępnych w AT&T C++. Zasadniczo do funkcji przekazuje się argumenty znakowe, a funkcje zwracają wartości, które można zapisać lub wydrukować. Korzystając z tych funkcji, przenosisz większość swojej pracy do C++ i pozwalasz jej wykonywać bardziej żmudne operacje na znakach i ciągach danych.

Funkcje testowania znaków

Kilka funkcji sprawdza pewne cechy twoich danych postaci. Możesz określić, czy dane postaci są alfabetyczne, cyfrowe, wielkie, małe i wiele innych. Musisz przekazać zmienną znakową lub dosłowny argument do funkcji (umieszczając argument w nawiasach funkcji) podczas jego wywoływania. Te funkcje zwracają wynik Prawda lub Fałsz, dzięki czemu można przetestować ich zwracane wartości wewnątrz instrukcji if lub pętli while.

UWAGA: Wszystkie funkcje znakowe przedstawione w tej sekcji są prototypowane w pliku nagłówkowym ctype.h. Pamiętaj, aby dołączyć ctype.h na początku wszelkich programów, które ich używają.

Testy alfabetyczne i cyfrowe

Następujące funkcje sprawdzają warunki alfabetyczne:

♦ isalpha(c): Zwraca wartość True (niezerową), jeśli c jest wielką lub małą literą. Zwraca False (zero), jeśli c nie jest literą.

♦ islower(c): Zwraca wartość True (niezerową), jeśli c jest małą literą. Zwraca Fałsz (zero), jeśli c nie jest małą literą.

♦ isupper(c): Zwraca wartość True (niezerową), jeśli c jest wielką literą. Zwraca False (zero), jeśli c nie jest wielką literą.

Pamiętaj, że każda niezerowa wartość to True w C++, a zero to zawsze False. Jeśli użyjesz zwracanych wartości tych funkcji w teście relacyjnym, Prawdziwa wartość zwracana nie zawsze wynosi 1 (może to

być dowolna wartość niezerowa), ale zawsze jest uważana za prawdę w teście. Następujące funkcje testują cyfry:

♦ isdigit(c): Zwraca True (niezerową), jeśli c jest cyfrą od 0 do 9. Zwraca False (zero), jeśli c nie jest cyfrą.

♦ isxdigit(c): Zwraca True (niezerową), jeśli c jest dowolną cyfrą szesnastkową od 0 do 9 lub A, B, C, D, E, F, a, b, c, d, e lub f. Zwraca False (zero), jeśli c jest czymś innym. (Więcej informacji na temat szesnastkowego systemu numerowania znajduje się w załączniku A, „Adresowanie pamięci, przegląd binarny i przegląd szesnastkowy").

UWAGA: Chociaż niektóre funkcje znakowe sprawdzają cyfry, argumenty są nadal danymi znakowymi i nie można ich używać w obliczeniach matematycznych, chyba że oblicza się za pomocą wartości znaków ASCII

Następująca funkcja testuje argumenty numeryczne lub alfabetyczne: isalnum(c): Zwraca True (niezerową), jeśli c jest cyfrą od 0 do 9 lub znakiem alfabetu (wielkimi lub małymi literami).

UWAGA: Do tych funkcji można przekazać tylko wartość znaku lub wartość całkowitą zawierającą wartość ASCII znaku. Nie można przekazać całej tablicy znaków do funkcji znakowych. Jeśli chcesz przetestować elementy tablicy znaków, musisz przekazać tablicę jeden element na raz.

Przykład

Poniższy program prosi użytkowników o ich inicjały. Jeśli użytkownik wpisze coś innego niż znaki alfabetyczne, program wyświetli błąd i ponownie zapyta. Zidentyfikuj program i dołącz pliki nagłówkowe wejścia / wyjścia. Program prosi użytkownika o swój pierwszy inicjał, więc zadeklaruj początkową zmienną znakową, aby zawierała odpowiedź użytkownika.

1. Poproś użytkownika o jego pierwszą inicjał i uzyskaj odpowiedź użytkownika. 2. Jeśli odpowiedź nie była alfabetyczna, powiedz to użytkownikowi i powtórz krok pierwszy. Wydrukuj wiadomość z podziękowaniami na ekranie.

```
// Nazwa pliku: C22INI.CPP

// Prosi o pierwszą inicjał i testy

// aby upewnić się, że odpowiedź jest poprawna.

#include <iostream.h>

#include <ctype.h>

void main()

{

char initial;

cout << "What is your first initial? ";

cin >> initial;

while (!isalpha(initial))

{
```

```
cout << “\nThat was not a valid initial! \n”;

cout << “\nWhat is your first initial? “;

cin >> initial;

}

cout << “\nThanks!”;

return;

}
```

To użycie operatora not (!) Jest dość jasne. Program kontynuuje zapętlanie, o ile wprowadzony znak nie jest alfabetyczny.

Specjalne funkcje testowania znaków

Kilka funkcji znakowych przydaje się, gdy musisz czytać z pliku dyskowego, modemu lub innego urządzenia systemu operacyjnego, z którego kierujesz dane wejściowe. Te funkcje nie są używane tak często, jak funkcje znaków, które widziałeś w poprzedniej sekcji, ale są przydatne do testowania określonych znaków pod kątem czytelności. Pozostałe funkcje testowania znaków są następujące:

♦ iscntrl(c): Zwraca wartość True (niezerową), jeśli c jest znakiem kontrolnym (dowolny znak z tabeli ASCII o numerach od 0 do 31). Zwraca False (zero), jeśli c nie jest znakiem kontrolnym.

♦ isgraph(c): Zwraca wartość True (niezerową), jeśli c jest dowolnym drukowalnym znakiem (znakiem niekontrolowanym) oprócz spacji. Zwraca False (zero), jeśli c jest spacją lub czymkolwiek innym niż znak, który można wydrukować.

♦ isprint(c): Zwraca wartość True (niezerową), jeśli c jest znakiem do wydruku (znak niekontrolowany) od ASCII 32 do ASCII 127, łącznie ze spacją. Zwraca Fałsz (zero), jeśli c nie jest znakiem do wydrukowania.

♦ ispunct(c): Zwraca True (niezerową), jeśli c jest dowolnym znakiem interpunkcyjnym (dowolnym znakiem do wydruku innym niż spacja, litera lub cyfra). Zwraca False (zero), jeśli c nie jest znakiem interpunkcyjnym.

♦ isspace(c): Zwraca wartość True (niezerową), jeśli c jest spacją, znakiem nowej linii (\ n), znakiem powrotu karetki (\ r), tabulatorem (\ t) lub znakiem tabulacji pionowej (\ v). Zwraca False (zero), jeśli c jest czymś innym.

Funkcje konwersji znaków

Dwie pozostałe funkcje postaci są przydatne. Zamiast znaków testowych funkcje te zmieniają znaki na ich małe lub wielkie odpowiedniki.

♦ tolower(c): Konwertuje c na małe litery. Nic się nie zmieni, jeśli podasz tolower () małą literę lub znak niealfabetyczny.

♦ toupper(c): Konwertuje c na wielkie litery. Nic się nie zmieni, jeśli podasz toupper () wielką literę lub znak niealfabetyczny.

Funkcje te zwracają zmienione wartości znaków. Te funkcje są przydatne do wprowadzania danych przez użytkownika. Załóżmy, że zadajesz użytkownikom pytanie tak lub nie, na przykład:

Czy chcesz wydrukować czeki (T / N)?

Przed opracowaniem toupper() i tolower () trzeba było sprawdzić zarówno Y, jak i Y, aby wydrukować czeki. Zamiast testować oba warunki, możesz przekonwertować znak na wielkie litery i przetestować dla Y.

Przykład

Oto program, który drukuje odpowiednią wiadomość, jeśli użytkownik jest dziewczynką lub chłopcem. Program testuje G i B po przekonwertowaniu danych wejściowych użytkownika na wielkie litery. Nie ma potrzeby sprawdzania małych liter. Zidentyfikuj program i dołącz pliki nagłówkowe wejścia / wyjścia. Program zadaje użytkownikowi pytanie wymagające odpowiedzi alfabetycznej, więc zadeklaruj zmienną znakową ans zawierająca odpowiedź użytkownika. Zapytaj, czy użytkownik jest dziewczynką czy chłopcem, i zapisz odpowiedź użytkownika w ans. Użytkownik musi nacisnąć Enter, więc włącz, a następnie odrzuć klawisz Enter. Zmień wartość ans na wielkie litery. Jeśli odpowiedź brzmi G, wydrukuj wiadomość. Jeśli odpowiedź brzmi B, wydrukuj inną wiadomość. Jeśli odpowiedź jest inna, wydrukuj inną wiadomość.

```
// Nazwa pliku: C22GB.CPP

// Określa, czy użytkownik wpisał G czy B.

#include <iostream.h>

#include <conio.h>

#include <ctype.h>

void main()

{

char ans; // Holds user's response.

cout << "Are you a girl or a boy (G/B)? ";

ans=getch(); // Get answer.

getch(); // Discard newline.

cout <<ans<<"\n";

ans = toupper(ans); // Convert answer to uppercase.

switch (ans)

{ case ('G'): { cout << "You look pretty today!\n";

break; }

case ('B'): { cout << "You look handsome today!\n";

break; }

default : { cout << "Your answer makes no sense!\n";

break; }

}
```

return;

}

Oto wynik działania programu:

Are you a girl or a boy (G/B)? B

You look handsome today!

Funkcje ciągów

Niektóre z najpotężniejszych wbudowanych funkcji C++ to funkcje łańcuchowe. Wykonują większość żmudnej pracy, dla której pisałeś do tej pory kod, np. Wprowadzanie ciągów z klawiatury i porównywanie ciągów. Podobnie jak w przypadku funkcji znakowych, nie ma potrzeby „wymyślać koła", pisząc kod, gdy wbudowane funkcje wykonują to samo zadanie. Korzystaj z tych funkcji w jak największym stopniu. Teraz, gdy dobrze znasz podstawy języka C++, możesz opanować funkcje łańcuchowe. Pozwalają skoncentrować się na głównym celu programu, zamiast spędzać czas na kodowaniu własnych funkcji łańcuchowych.

Przydatne funkcje ciągów

Możesz użyć garści przydatnych funkcji ciągów do testowania i konwersji ciągów. Widziałeś już (we wcześniejszych rozdziałach) funkcję łańcuchową strcpy (), która kopiuje ciąg znaków do tablicy znaków.

UWAGA: Wszystkie funkcje łańcucha w tej sekcji są prototypowane w pliku nagłówkowym string.h. Pamiętaj, aby dołączyć string.h na początku każdego programu, który korzysta z funkcji string. Funkcje ciągów, które testują lub manipulują ciągami:

♦ strcat(s1, s2): Łączy (scala) ciąg s2 z końcem tablicy znaków s1. Tablica s1 musi mieć wystarczająco dużo zarezerwowanych elementów, aby pomieścić oba łańcuchy.

♦ strcmp(s1, s2): Porównuje ciąg s1 ze łańcuchem s2 w kolejności alfabetycznej, element po elemencie. Jeśli s1 alfabetycznie przed s2, strcmp () zwraca wartość ujemną. Jeśli s1 i s2 są tymi samymi łańcuchami, strcmp () zwraca 0. Jeśli s1 alfabetycznie po s2, strcmp () zwraca wartość dodatnią.

♦ strlen(s1): Zwraca długość s1. Pamiętaj, że długość łańcucha to liczba znaków, nie licząc zera zero. Liczba znaków zdefiniowana dla tablicy znaków nie ma nic wspólnego z długością łańcucha.

WSKAZÓWKA: Przed użyciem strcat () do konkatenacji łańcuchów, użyj strlen (), aby upewnić się, że łańcuch docelowy (łańcuch jest konkatenowany) jest wystarczająco duży, aby pomieścić oba łańcuchy.

Funkcje ciągów we / wy

W poprzednich kilku częściach użyłeś funkcji wprowadzania znaków, cin.get(), aby zbudować ciągi wejściowe. Teraz możesz zacząć korzystać z ciągów wejściowych i wyjściowych funkcji. Chociaż celem funkcji budowania ciągów było nauczenie cię specyfiki języka, te funkcje we / wy są znacznie łatwiejsze w użyciu niż pisanie funkcji wprowadzania znaków. Funkcje wejściowe i wyjściowe ciągu są wymienione w następujący sposób:

♦ gets(s): Przechowuje dane wejściowe ze standardowego wejścia (zwykle skierowane na klawiaturę) do ciągu o nazwie s.

♦ puts(s): Wysyła ciąg s na standardowe wyjście (zwykle kierowane na ekran przez system operacyjny).

♦ fgets(s, len, dev): Przechowuje dane wejściowe ze standardowego urządzenia określonego przez dev (np. Stdin lub stdaux) w ciągu s. Jeśli wprowadzono więcej niż len znaków, fgets () je odrzuca.

♦ fputs(s, dev): Wysyła ciąg s do standardowego urządzenia określonego przez dev.

Te cztery funkcje ułatwiają wprowadzanie i wyprowadzanie ciągów. Pracują w parach. Oznacza to, że ciągi wejściowe za pomocą gets() są zwykle wyprowadzane za pomocą puts(). Ciągi wejściowe za pomocą fgets() są zwykle wyprowadzane za pomocą fputs().

WSKAZÓWKA: gets() zastępuje funkcję wprowadzania ciągów znaków, którą widziałeś we wcześniejszych częściach

Zakończ gets() lub fgets () przez naciśnięcie Enter. Każda z tych funkcji obsługuje znaki kończące ciąg w nieco inny sposób, w następujący sposób:

gets () Wejście nowego wiersza staje się zerowym zerem (\ 0).

puts() Null na końcu ciągu staje się znakiem nowej linii (\ n).

fgets() Pozostaje wpis nowej linii, a po nim dodawane jest zero.

fputs() Zerowane jest zero, a znak nowej linii nie jest dodawany.

Dlatego po wprowadzeniu ciągów za pomocą gets() C++ umieszcza znak kończący ciąg w ciągu, w którym naciskasz Enter. To tworzy ciąg wejściowy. (Bez zerowego zera dane wejściowe nie byłyby ciągiem.) Kiedy wyprowadzasz ciąg, zerowe zero na końcu łańcucha staje się znakiem nowej linii. Jest to preferowane, ponieważ nowa linia znajduje się na końcu wiersza wyniku, a kursor zaczyna się automatycznie w następnym wierszu. Ponieważ fgets () i fputs () mogą wprowadzać i wyprowadzać ciągi z urządzeń, takich jak pliki dyskowe i modemy telefoniczne, krytyczne może być zachowanie przychodzących znaków nowej linii dla zachowania integralności danych. Podczas wysyłania ciągów do tych urządzeń nie chcesz, aby C++ wstawiał dodatkowe znaki nowego wiersza.

UWAGA: Ani get(), ani fgets() nie zapewnia, że jego łańcuchy wejściowe są wystarczająco duże, aby pomieścić przychodzące dane. Od Ciebie zależy, czy w tablicy znaków zarezerwowana zostanie wystarczająca ilość miejsca, aby pomieścić pełne dane wejściowe.

Warto zwrócić uwagę na ostatnią funkcję, chociaż nie jest to funkcja łańcuchowa. Jest to funkcja fflush(), która wypłukuje (opróżnia) dowolne standardowe urządzenie wymienione w nawiasach. Aby opróżnić klawiaturę ze wszystkich danych wejściowych, należy napisać w następujący sposób:

fflush (stdin);

Podczas wprowadzania i wyprowadzania ciągów czasami w buforze klawiatury pojawia się dodatkowy znak nowej linii. Poprzednia odpowiedź na get() lub getc() mogła zawierać dodatkowy nowy wiersz, którego zapomniałeś odrzucić. Kiedy program wydaje się ignorować get (), być może będziesz musiał wstawić fflush (stdin) przed gets(). Opróżnienie standardowego urządzenia wejściowego nie powoduje żadnych szkód, a użycie go może wyczyścić strumień wejściowy, dzięki czemu Twój następny get() będzie działał poprawnie. Możesz także opróżnić standardowe urządzenia wyjściowe za pomocą fflush(), aby wyczyścić strumień wyjściowy ze wszystkich wysłanych do niego znaków. UWAGA: Plik nagłówka dla fflush() znajduje się w stdio.h.

Przykład

Poniższy program pokazuje, jak łatwo jest używać funkcji get() i puts(). Program żąda nazwy książki od użytkownika za pomocą pojedynczego wywołania funkcji gets(), a następnie drukuje tytuł książki za pomocą puts().

Zidentyfikuj program i dołącz pliki nagłówkowe wejścia / wyjścia. Program pyta użytkownika o nazwę książki. Zadeklaruj tablicę znaków z 30 elementami, aby zachować odpowiedź użytkownika. Poproś użytkownika o tytuł książki i zapisz odpowiedź użytkownika w tablicy książek. Wyświetl ciąg zapisany w książce na urządzeniu wyjściowym, prawdopodobnie swoim ekranem. Wywietl wiadomość z podziękowaniem.

```
// C22GPS1.CPP

// Ciągi danych wejściowych i wyjściowych.

#include <iostream.h>

#include <stdio.h>

#include <string.h>

void main()

{

char book[30];

cout << “What is the book title? “;

gets(book); // Get an input string.

puts(book); // Display the string.

cout << “Thanks for the book!\n”;

return;

}
```

Dane wyjściowe programu są następujące:

```
What is the book title? Mary and Her Lambs

Mary and Her Lambs

Thanks for the book!
```

Konwertowanie ciągów na liczby

Czasami musisz przekonwertować liczby zapisane w ciągach znaków na numeryczny typ danych. AT&T C++ zapewnia trzy funkcje, które pozwalają to zrobić:

♦ atoi(s): Konwertuje s na liczbę całkowitą. Nazwa oznacza alfabetyczną liczbę całkowitą.

♦ atol(s): Konwertuje s na długą liczbę całkowitą. Nazwa oznacza liczbę całkowitą alfabetyczną.

♦ atof(s): Konwertuje s na liczbę zmiennoprzecinkową. Nazwa oznacza alfabet do zmiennoprzecinkowego.

UWAGA: Te trzy funkcje ato() są prototypowane w pliku nagłówkowym stdlib.h. Pamiętaj, aby dołączyć stdlib.h na początku każdego programu, który korzysta z funkcji ato().

Ciąg musi zawierać prawidłową liczbę. Oto ciąg, który można przekonwertować na liczbę całkowitą:

„1232”

Ciąg musi zawierać ciąg cyfr wystarczająco krótki, aby pasował do docelowego typu danych liczbowych. Poniższego ciągu nie można przekonwertować na liczbę całkowitą za pomocą funkcji atoi():

„-1232495.654”

Można go jednak przekonwertować na liczbę zmiennoprzecinkową za pomocą funkcji atof (). C++ nie może wykonać żadnych obliczeń matematycznych z takimi ciągami, nawet jeśli ciągi zawierają cyfry reprezentujące liczby. Dlatego przed wykonaniem arytmetyki należy przekonwertować dowolny ciąg znaków na jego numeryczny odpowiednik.

UWAGA: Jeśli przekażesz ciąg do funkcji ato (), a ciąg nie będzie zawierał prawidłowej reprezentacji liczby, funkcja ato () zwróci 0.

Funkcje te stają się bardziej przydatne po zapoznaniu się z plikami dyskowymi i wskaźnikami.

Funkcje numeryczne

W tej sekcji przedstawiono wiele wbudowanych funkcji numerycznych C++. Podobnie jak w przypadku funkcji łańcuchowych, funkcje te oszczędzają czas, konwertując i obliczając liczby, zamiast konieczności pisania funkcji, które robią to samo. Wiele z nich to trygonometryczne i zaawansowane funkcje matematyczne. Niektóre z tych funkcji numerycznych mogą być używane rzadko, ale są one dostępne, jeśli są potrzebne. W tej sekcji omówiono standardowe funkcje wbudowane w C++. Po opanowaniu pojęć z tej części jesteś gotowy, aby dowiedzieć się więcej o tablicach i wskaźnikach. Gdy rozwijasz coraz więcej umiejętności w C++, możesz polegać na tych funkcjach numerycznych, łańcuchowych i znakowych, pisząc mocniejsze programy.

Przydatne funkcje matematyczne

Kilka wbudowanych funkcji numerycznych zwraca wyniki na podstawie zmiennych numerycznych i literałów do nich przekazanych. Nawet jeśli napiszesz tylko kilka programów naukowych i inżynierskich, niektóre z tych funkcji są przydatne.

UWAGA: Wszystkie funkcje matematyczne i trygonometryczne są prototypowane w pliku nagłówkowym math.h. Pamiętaj, aby dołączyć go na początku każdego programu, który korzysta z funkcji numerycznych. Oto funkcje wymienione wraz z ich opisami:

♦ ceil(x): Funkcja ceil () lub pułap zaokrągla liczby w górę do najbliższej liczby całkowitej.

♦ fabs(x): Zwraca wartość bezwzględną x. Wartość bezwzględna liczby jest jej dodatnim odpowiednikiem.

WSKAZÓWKA: Wartość bezwzględna jest stosowana do odległości (które zawsze są dodatnie), pomiarów dokładności, różnic wieku i innych obliczeń wymagających pozytywnego wyniku.

♦ floor(x): Funkcja floor () zaokrągla liczby w dół do najbliższej liczby całkowitej.

♦ fmod(x, y): Funkcja fmod () zwraca resztę zmiennoprzecinkową (x / y) z tym samym znakiem co x, a y nie może wynosić zero. Ponieważ operator modułu (%) działa tylko z liczbami całkowitymi, funkcja ta służy do znalezienia pozostałej części liczb zmiennoprzecinkowych.

♦ pow (x, y): Zwraca x podniesione do potęgi y lub xy. Jeśli x jest mniejsze lub równe zero, y musi być liczbą całkowitą. Jeśli x jest równe zero, y nie może być ujemne.

♦ sqrt(x): Zwraca pierwiastek kwadratowy z x; x musi być większe lub równe zero.

N-ty pierwiastek

Żadna funkcja nie zwraca n-tego pierwiastka z liczby, tylko pierwiastek kwadratowy. Innymi słowy, nie można wywołać funkcji, która daje czwarty pierwiastek z 65 536. (Nawiasem mówiąc, 16 jest czwartym pierwiastkiem z 65 536, ponieważ 16 razy 16 razy 16 razy 16 = 65 536). Możesz jednak użyć sztuczki matematycznej, aby zasymulować n-ty pierwiastek. Ponieważ C++ umożliwia podniesienie liczby do potęgi ułamkowej - za pomocą funkcji pow() - możesz podnieść liczbę do n-tego pierwiastka, podnosząc ją do potęgi (1 / n). Na przykład, aby znaleźć czwarty pierwiastek z 65 536, możesz wpisać:

root = pow (65536,0, (1,0 / 4,0));

Zauważ, że przecinek dziesiętny utrzymuje liczby w formacie zmiennoprzecinkowym. Jeśli pozostawisz je jako liczby całkowite, takie jak

root = pow (65536, (1/4));

C++ daje nieprawidłowe wyniki. Funkcja pow() i większość innych funkcji matematycznych wymaga wartości zmiennoprzecinkowych jako argumentów. Aby zapisać siódmy pierwiastek z 78 125 w zmiennej o nazwie root, na przykład wpisz

root = pow (78125.0, (1.0 / 7.0));

To przechowuje 5.0 w katalogu głównym, ponieważ 5 7 równa się 78 125. Umiejętność obliczenia n-tego źródła jest przydatna w programach naukowych, a także w aplikacjach finansowych, takich jak problemy z wartością pieniądza.

Przykład

Poniższy program używa funkcji fabs() do obliczenia różnicy między dwoma grupami wiekowymi.

```
// Nazwa pliku: C22ABS.CPP

// Oblicza różnicę między dwoma wiekami.

#include <iostream.h>

#include <matematyka.h>

void main()

{

{

float age1, age2, diff;

cout << "\nWhat is the first child's age? ";
```

```
cin >> age1;

cout << “What is the second child’s age? “;

cin >> age2;

// Calculates the positive difference.

diff = age1 - age2;

diff = fabs(diff); // Determines the absolute value.

cout << “\nThey are “ << diff << “ years apart.”;

return;

}
```

Wyjście z tego programu następuje. Ze względu na fabs() kolejność grup wiekowych nie ma znaczenia. Bez wartości bezwzględnej program ten spowodowałby ujemną różnicę wieku, gdyby pierwszy wiek był mniejszy niż drugi. Ponieważ wiek jest stosunkowo mały, w tym przykładzie zastosowano zmienne zmiennoprzecinkowe. C++ automatycznie konwertuje argumenty zmiennoprzecinkowe do podwójnej precyzji podczas przekazywania ich do fabs().

Jaki jest wiek pierwszego dziecka? 10

Jaki jest wiek drugiego dziecka? 12

Są w odstępie 2 lat.

Funkcje trygonometryczne

Do aplikacji trygonometrycznych dostępne są następujące funkcje:

♦ cos(x): Zwraca cosinus kąta x, wyrażony w radianach.

♦ sin(x): Zwraca sinus kąta x, wyrażony w radianach.

♦ tan(x): Zwraca tangens kąta x, wyrażony w radianach.

Są to prawdopodobnie najmniej używane funkcje. Nie oznacza to jednak umniejszenia pracy programistów naukowych i matematycznych, którzy ich potrzebują. Z pewnością są wdzięczni, że C++ dostarcza te funkcje! W przeciwnym razie programiści musieliby napisać własne funkcje, aby wykonać te trzy podstawowe obliczenia trygonometryczne. Większość kompilatorów C++ zapewnia dodatkowe funkcje trygonometryczne, w tym hiperboliczne odpowiedniki tych trzech funkcji.

WSKAZÓWKA: Jeśli musisz przekazać kąt wyrażony w stopniach do tych funkcji, zamień stopnie kąta na radiany, mnożąc stopnie przez π / 180,0 (π wynosi około 3,14159).

Funkcje logarytmiczne

Trzy wysoce matematyczne funkcje są czasami używane w biznesie i matematyce. Są one wymienione w następujący sposób:

♦ exp(x): Zwraca podstawę logarytmu naturalnego (e) podniesioną do potęgi określonej przez x (ex); e jest wyrażeniem matematycznym przybliżonej wartości 2,718282.

♦ log(x): Zwraca logarytm naturalny argumentu x, zapisany matematycznie jako ln (x). x musi być dodatnie.

♦ log10(x): Zwraca logarytm dziesiętny argumentu x, zapisany matematycznie jako log10 (x). x musi być dodatnie.

Przetwarzanie liczb losowych

Losowe wydarzenia zdarzają się każdego dnia. Budzisz się i jest słonecznie lub deszczowo. Masz dobry lub zły dzień. Dostajesz telefon od starego przyjaciela lub nie. Wartość twojego portfela akcji może wzrosnąć lub spaść. Losowe zdarzenia są szczególnie ważne w grach. Częścią zabawy w grach jest twoje szczęście z rzucaniem kostkami lub ciągnięciem kart w połączeniu z umiejętnościami gry. Symulacja zdarzeń losowych jest ważnym zadaniem dla komputerów. Komputery są jednak maszynami skończonymi; przy takim samym wejściu zawsze wytwarzają ten sam wynik. Ten fakt może stworzyć nudne gry! Projektanci C++ znali tę komplikację komputerową i znaleźli sposób na jej przezwyciężenie. Napisali funkcję generującą liczby losowe o nazwie rand(). Możesz na przykład użyć rand(), aby obliczyć rzut kości lub narysować kartę. Aby wywołać funkcję rand() i przypisać zwróconą liczbę losową do przetestowania, użyj następującej składni:

test = rand ();

Funkcja rand() zwraca liczbę całkowitą od 0 do 32 767. Nigdy nie używaj argumentów w nawiasach rand(). Za każdym razem, gdy wywołujesz rand() w tym samym programie, otrzymujesz inny numer. Jeśli jednak ciągle uruchamiasz ten sam program, funkcja rand() zwraca ten sam zestaw liczb losowych. Jednym ze sposobów otrzymania innego zestawu liczb losowych jest wywołanie funkcji srand(). Format srand() jest następujący:

srand (seed);

gdzie seed jest zmienną całkowitą lub literałem. Jeśli nie wywołasz srand(), C++ zakłada wartość początkową 1.

UWAGA: Funkcje rand() i srand() są prototypowane w pliku nagłówkowym stdlib.h. Pamiętaj, aby dołączyć stdlib.h na początku każdego programu, który używa rand() lub srand().

Wartość początkowa resetuje lub resetuje generator liczb losowych, więc następna liczba losowa jest oparta na nowej wartości początkowej. Jeśli wywołasz srand() z inną wartością początkową na górze programu, rand() zwróci inną liczbę losową przy każdym uruchomieniu programu.

Dlaczego musisz to zrobić?

Istnieje znaczna debata wśród programistów C++ na temat generatora liczb losowych. Wielu uważa, że liczby losowe powinny być naprawdę losowe i że nie powinny same musieć inicjować generatora. Uważają, że C++ powinien wykonać własne wewnętrzne inicjowanie, gdy poprosisz o losową liczbę. Jednak wiele aplikacji nie działałoby, gdyby generator liczb losowych był dla ciebie losowy. Komputery są wykorzystywane w biznesie, poznaniu i badaniach, aby symulować wzorzec wydarzeń w świecie rzeczywistym. Naukowcy muszą być w stanie powtarzać te symulacje w kółko. Chociaż zdarzenia wewnątrz symulacji mogą być od siebie losowe, przeprowadzanie symulacji nie może być losowe, jeśli badacze mają zbadać kilka różnych efektów. Matematycy i statystycy muszą także powtarzać losowe wzorce dla swoich analiz, szczególnie gdy pracują z teoriami ryzyka, prawdopodobieństwa i gier. Ponieważ tak wielu użytkowników komputerów musi powtarzać wzorce liczb losowych, projektanci C++ mądrze wybrali, aby dać programistom możliwość zachowania tych samych wzorców losowych lub ich zmiany. Zalety znacznie przewyższają wadę dołączenia dodatkowego wywołania funkcji srand ().

Jeśli chcesz generować inny zestaw liczb losowych przy każdym uruchomieniu programu, musisz ustalić, w jaki sposób kompilator C++ odczytuje zegar systemowy komputera. Możesz użyć licznika sekund z zegara do uruchomienia generatora liczb losowych, aby wyglądał na naprawdę losowy.

Ćwiczenia

1. Napisz program, który pyta użytkowników o wiek. Jeśli użytkownik wpisze coś innego niż dwie cyfry, wyświetl komunikat o błędzie.

2. Napisz program, który przechowuje hasło w tablicy znaków o nazwie pass. Zapytaj użytkowników o hasło. Użyj strcmp (), aby poinformować użytkowników, czy wpisali prawidłowe hasło. Użyj ciągowych funkcji we / wy dla wszystkich danych wejściowych i wyjściowych programu.

3. Napisz program, który zaokrągla w górę i zaokrągla liczby –10,5, –5,75 i 2,75.

4. Zapytaj użytkowników o ich nazwy. Wydrukuj każde imię w odwrotnej wielkości; wydrukuj pierwszą literę każdego imienia małymi literami, a resztę nazwiska dużymi literami.

5. Napisz program, który prosi użytkowników o pięć tytułów filmowych. Wydrukuj najdłuższy tytuł. Używaj tylko ciągów we / wy i funkcji manipulacji przedstawionych w tym rozdziale.

6. Napisz program, który oblicza pierwiastek kwadratowy, pierwiastek kostny i czwarty pierwiastek liczb od 10 do 25 włącznie

7. Zapytaj użytkowników o tytuły swoich ulubionych piosenek. Odrzuć wszystkie znaki specjalne w każdym tytule. Wydrukuj słowa w tytule, po jednym w wierszu. Na przykład, jeśli wpiszą „Moja prawdziwa miłość to moja”, „Och, moja!”, Powinieneś podać następujące dane:

Mój

Prawdziwe

Miłość

Jest

Moje

O

Moje

8. Zapytaj użytkowników o imiona 10 dzieci. Używając strcmp () dla każdej nazwy, napisz program, aby wydrukować nazwę, która jest pierwsza w alfabecie.

Podsumowanie

Nauczyłeś się funkcji znakowych, łańcuchowych i numerycznych, które zapewnia C++. Dołączając plik nagłówka ctype.h, możesz testować i konwertować znaki wpisywane przez użytkownika. Funkcje te mają wiele przydatnych celów, takich jak konwertowanie odpowiedzi użytkownika na wielkie litery.

Ułatwia to testowanie danych wejściowych użytkownika. Funkcje I / O ciągów zapewniają większą kontrolę zarówno nad wprowadzaniem ciągów, jak i numerycznymi. Możesz otrzymać ciąg cyfr z klawiatury i przekonwertować je na liczbę za pomocą funkcji ato (). Funkcje porównywania i łączenia ciągów umożliwiają testowanie i zmianę zawartości więcej niż jednego ciągu. Funkcje oszczędzają czas programowania, ponieważ przejmują niektóre zadania komputerowe, pozwalając Ci skoncentrować

się na programach. Funkcje numeryczne C++ zaokrąglają liczby i manipulują nimi, generują wyniki trygonometryczne i logarytmiczne oraz generują liczby losowe. Teraz, gdy poznałeś większość wbudowanych funkcji C++, możesz poprawić swoją umiejętność pracy z tablicami. Rozdział 23, „Wprowadzenie do tablic”, poszerza Twoją wiedzę o tablicach znaków i pokazuje, jak tworzyć tablice dowolnego typu danych.

Przedstawiamy tablice

Omówimy różne typy tablic. Znacie już tablice znaków, które są jedyną metodą przechowywania ciągów znaków w języku C++. Tablica znaków nie jest jednak jedynym rodzajem tablicy, którego można użyć. W C++ istnieje tablica dla każdego typu danych. Ucząc się, jak przetwarzać tablice, znacznie poprawiasz moc i wydajność swoich programów. Ta część wprowadza

♦ Podstawy tablic nazw, typów danych i indeksów dolnych

♦ Inicjalizacja tablicy w czasie deklaracji

♦ Inicjowanie tablicy podczas wykonywania programu

♦ Wybieranie elementów z tablic

Tablice nie są trudne aby ich używać, ale ich moc sprawia, że dobrze nadają się do bardziej zaawansowanego programowania.

Podstawy tablic

Chociaż widziałeś tablice używane jako ciągi znaków, nadal musisz mieć przegląd tablic w ogóle. Tablica to lista więcej niż jednej zmiennej o tej samej nazwie. Nie wszystkie listy zmiennych są tablicami. Na przykład poniższa lista czterech zmiennych nie kwalifikuj się jako tablica.

sales bonus_92 first_initial ctr

To jest lista zmiennych (cztery z nich), ale nie jest to tablica, ponieważ każda zmienna ma inną nazwę. Możesz się zastanawiać, jak więcej niż jedna zmienna może mieć tę samą nazwę; wydaje się to naruszać reguły dotyczące zmiennych. Jeśli dwie zmienne mają tę samą nazwę, w jaki sposób C++ może określić, do którego odniesienia się odnosi, używając tej nazwy? Zmienne tablicowe lub elementy tablicowe są różnicowane przez indeks dolny, który jest liczbą w nawiasach kwadratowych. Załóżmy, że chcesz zapisać imię osoby w tablicy znaków zwanej imieniem. Możesz to zrobić za pomocą

char name [] = „Ray Krebbs”;

lub

char name [11] = „Ray Krebbs”;

Ponieważ C++ rezerwuje dodatkowy element dla zerowego zera na końcu każdego łańcucha, nie musisz określać 11, dopóki inicjujesz tablicę wartością. Nazwa zmiennej jest tablicą, ponieważ nawiasy następują po jej nazwie. Tablica ma jedną nazwę, nazwę i zawiera 11 elementów. Każdy element jest postacią.

UWAGA: Wszystkie indeksy tablicy zaczynają się od 0.

Możesz manipulować poszczególnymi elementami w tablicy, odwołując się do ich indeksów dolnych. Na przykład następujący napis drukuje inicjały Raya. Wydrukuj pierwszy i piąty element tablicy o nazwie name.

cout << nazwa [0] << „“ << nazwa [4];

Możesz zdefiniować tablicę jako dowolny typ danych w C++. Możesz mieć tablice liczb całkowitych, długie tablice liczb całkowitych, podwójne tablice zmiennoprzecinkowe, krótkie tablice liczb całkowitych i tak dalej. C++ rozpoznaje, że nawiasy [] po nazwie tablicy oznaczają, że definiujesz tablicę,

a nie pojedynczą zmienną inną niż tablica. Poniższy wiersz definiuje tablicę o nazwie wiek, składającą się z pięciu liczb całkowitych:

age [5];

Pierwszy element w tablicy wieków to epoki [0]. Drugi element to wiek [1], a ostatni to wiek [4]. Ta deklaracja wieku nie przypisuje wartości do elementów, więc nie wiesz, co jest w wieku, a twój program nie wyzeruje automatycznie dla ciebie wieku. Oto kilka definicji tablic:

int weights[25], sizes[100]; // Deklarujedwie tablice liczb całkowitych.

float salaries[8]; // Deklaruje tablicę zmiennoprzecinkową.

double temps[50]; Deklaruje tablicę zmiennoprzecinkową double

char letters[15]; // Dekalruje tablicę znaków.

Kiedy deklarujesz tablicę, instruujesz C++, aby zarezerwował określoną liczbę lokalizacji pamięci dla tej tablicy. C++ chroni te elementy. W poprzednich wierszach kodu, jeśli przypisujesz wartość do liter [2], nie zastępujesz żadnych danych dotyczących wag, rozmiarów, wynagrodzeń ani tempa. Ponadto, jeśli przypiszesz wartość do rozmiarów [94], nie zastąpisz danych zapisanych w wagach, pensjach, tymczasach lub listach. Każdy element w tablicy zajmuje tyle samo miejsca, co zmienna nonarray tego samego typu danych. Innymi słowy, każdy element w tablicy znaków zajmuje jeden bajt. Każdy element w tablicy liczb całkowitych zajmuje dwa lub więcej bajtów pamięci - w zależności od wewnętrznej architektury komputera. To samo dotyczy każdego innego typu danych. Twój program może odwoływać się do elementów za pomocą formuł do indeksów dolnych. Tak długo, jak indeks dolny może być liczbą całkowitą, możesz używać literału, zmiennej lub wyrażenia dla indeksu dolnego. Wszystkie poniższe są odniesieniami do poszczególnych elementów tablicy:

ara [4]

sales [ctr + 1]

bonus [month]

salary [month [i] * 2]

Wszystkie elementy tablicy są przechowywane w sposób ciągły, jeden za drugim. Należy o tym pamiętać, zwłaszcza podczas pisania bardziej zaawansowanych programów. Zawsze możesz liczyć na pierwszy element tablicy poprzedzający drugi. Drugi element jest zawsze umieszczany bezpośrednio przed trzecim i tak dalej. Pamięć nie jest „wyścielana”; co oznacza, że C++ gwarantuje, że nie ma dodatkowej przestrzeni między elementami tablicy. Dotyczy to tablic znaków, tablic liczb całkowitych, tablic zmiennoprzecinkowych i wszystkich innych typów tablic. Jeśli wartość zmiennoprzecinkowa zajmuje cztery bajty pamięci na twoim komputerze, następny element w tablicy zmiennoprzecinkowej zawsze zaczyna się dokładnie cztery bajty po poprzednim elemencie.

Rozmiar tablic

Funkcja sizeof() zwraca liczbę bajtów potrzebną do przechowywania jej argumentu. Jeśli zażądasz rozmiaru nazwy tablicy, sizeof() zwraca liczbę bajtów zarezerwowanych dla całej tablicy. Załóżmy na przykład, że deklarujesz tablicę liczb całkowitych 100 elementów zwanych wynikami. Jeśli chcesz znaleźć rozmiar tablicy, jak poniżej:

n = sizeof(wyniki);

n zawiera 200 lub 400 bajtów, w zależności od liczby całkowitej komputera. Funkcja sizeof () zawsze zwraca zarezerwowaną ilość pamięci, bez względu na to, jakie dane znajdują się w tablicy. Dlatego zawartość tablicy znaków - nawet jeśli zawiera bardzo krótki ciąg - nie wpływa na rozmiar tablicy pierwotnie zarezerwowanej w pamięci. Jeśli jednak zażądasz rozmiaru pojedynczego elementu tablicy, jak poniżej:

n = sizeof(wyniki [6]);

n zawiera 2 lub 4 bajty, w zależności od liczby całkowitej komputera.

Nigdy nie wolno przekraczać granic tablicy. Załóżmy na przykład, że chcesz śledzić zwolnienia i kody wynagrodzeń pięciu pracowników. Możesz zarezerwować dwie tablice do przechowywania takich danych, jak poniżej:

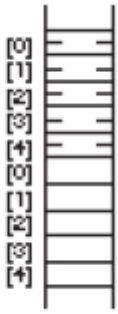
int wyjątki [5]; // Posiada do pięciu zwolnień pracowników.

char sal_codes [5]; // Przechowuje do pięciu kodów pracowników.

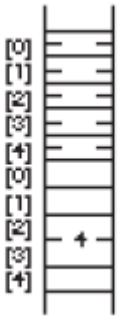
Rysunek pokazuje, jak C++ rezerwuje pamięć dla tych tablic. Liczba przyjmuje dwubajtową liczbę całkowitą, chociaż może się różnić na niektórych komputerach. Zauważ, że C++ rezerwuje pięć elementów na wyłączenia z deklaracji tablicowej. C++ zaczyna rezerwować pamięć dla sal_codes po zarezerwowaniu wszystkich pięciu elementów dla wyjątków. Jeśli zadeklarujesz kilka kolejnych zmiennych - lokalnie lub globalnie - po tych dwóch wierszach, C++ zawsze chroni te zarezerwowane pięć elementów dla wyjątków i kodów_sal. Ponieważ C++ ma swój udział w ochronie danych w tablicy, ty też musisz. Jeśli zarezerwujesz pięć elementów dla wyjątków, masz pięć elementów tablicy liczb całkowitych zwanych wyjątkami [0], wyjątkami [1], wyjątkami [2], wyjątkami [3] i wyjątkami [4]. C++ nie chroni więcej niż pięciu elementów dla wyjątków! Załóżmy, że umieściłeś wartość w elemencie zwolnień, którego nie zarezerwowałeś:

exemptions [6] = 4; // Przypisz wartość do

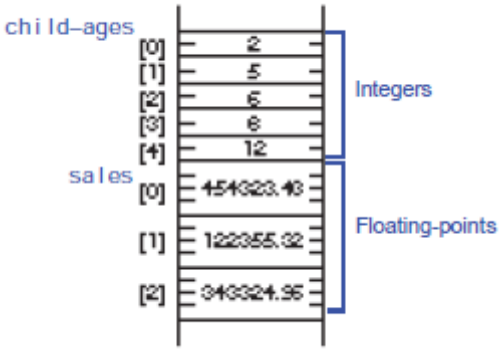
// element poza zasięgiem.



C++ pozwala ci to zrobić - ale wyniki są szkodliwe! C++ zastępuje inne dane (w tym przypadku sal_codes [2] i sal_codes [3], ponieważ są one zarezerwowane w miejscu siódmego elementu wyjątków). Rysunek pokazuje szkodliwe wyniki przypisywania wartości elementowi spoza zakresu



Chociaż możesz zdefiniować tablicę dowolnego typu danych, nie możesz zadeklarować tablicy ciągów. Łańcuch nie jest zmiennym typem danych C++.



Chociaż C++ nie inicjuje automatycznie elementów tablicy, jeśli zainicjujesz niektóre, ale nie wszystkie elementy, kiedy deklarujesz tablicę, C++ kończy zadanie, przypisując resztę do zera.

WSKAZÓWKA: Aby zainicjować każdy element dużej tablicy jednocześnie do zera, zadeklaruj całą tablicę i zainicjuj tylko pierwszą wartość do zera. C++ wypełnia resztę tablicy do zera.

Załóżmy na przykład, że musisz zarezerwować miejsce w macierzy na dane dotyczące zysków z trzech poprzednich miesięcy, a także trzech kolejnych miesięcy. Musisz zarezerwować sześć elementów pamięci, ale znasz wartości tylko dla pierwszych trzech. Możesz zainicjować tablicę w następujący sposób:

double bonus [6] = {67654,43, 46472,34, 63451,93};

Ponieważ jawnie zainicjowałeś trzy elementy, C++ inicjuje resztę do zera. Jeśli użyjesz cout do wydrukowania całej tablicy, po jednym elemencie w wierszu, otrzymasz:

67654,43

46472,34

63451,93

00000,00

00000,00

00000,00

UWAGA: Zawsze deklaruj tablicę z maksymalną liczbą indeksów dolnych, chyba że tablica zostanie zainicjowana w tym samym czasie. Następująca deklaracja tablicowa jest nielegalna:

int count []; // Zła deklaracja tablicy!

C++ nie wie, ile elementów zarezerwować do zliczenia, więc nie rezerwuje żadnego. Jeśli następnie przypiszesz wartości do niezarezerwowanych elementów, możesz (i prawdopodobnie) nadpisze inne dane. Jedynym czasem, w którym można pozostawić puste nawiasy, jest przypisanie wartości do tablicy, na przykład:

int [] = {15, 9, 22, -8, 12}; // Dobra definicja.

C++ może określić z listy wartości, ile elementów należy zarezerwować. W tym przypadku C++ rezerwuje pięć elementów do zliczenia.

Przykłady

1. Załóżmy, że chcesz śledzić średnie giełdowe z ostatnich 90 dni. Zamiast przechowywać je w 90 różnych zmiennych, znacznie łatwiej jest przechowywać je w tablicy. Możesz zadeklarować tablicę w następujący sposób:

float stock[90];

Pozostała część programu może przypisywać wartości do średnich.

2. Załóżmy, że właśnie skończyłeś lekcje na lokalnym uniwersytecie i chcesz uśrednić swoje sześć klas. Poniższy program inicjuje jedną tablicę dla nazwy szkoły, a drugą dla sześciu klas. Treść programu uśrednia sześć wyników.

```
// Nazwa pliku: C23ARA1.CPP

// Średnio sześć wyników testu.

#include <iostream.h>

#include <iomanip.h>

void main ()

{

s_name[] = “Tri Star University”;

float scores[6] = {88.7, 90.4, 76.0, 97.0, 100.0, 86.7};

float average=0.0;

int ctr;

// Computes total of scores.

for (ctr=0; ctr<6; ctr++)

{ average += scores[ctr]; }

// Computes the average.

average /= float(6);
```

```
cout << “At “ << s_name << “, your class average is “

<< setprecision(2) << average << “\n”;

return;

}
```

Dane wyjściowe są następujące:

```
At Tri Star University, your class average is 89.8
```

Zauważ, że korzystanie z tablic znacznie ułatwia przetwarzanie list informacji. Zamiast uśredniać sześć zmiennych o różnych nazwach, można użyć pętli for, aby przejść przez każdy element tablicy. Gdybyś musiał uśrednić 1000 liczb, nadal możesz to zrobić za pomocą prostej pętli for, jak w tym przykładzie. Gdyby 1000 zmiennych nie było w tablicy, ale zostały indywidualnie nazwane, musiałbyś napisać znaczną ilość kodu, aby je dodać.

3. Poniższy program jest rozszerzoną wersją poprzedniego. Drukuje sześć wyników przed obliczeniem średniej. Zauważ, że musisz drukować elementy tablicy indywidualnie; nie można wydrukować całej tablicy za jednym razem. (Możesz wydrukować całą tablicę znaków za pomocą cout, ale tylko jeśli zawiera ciąg znaków zakończony znakiem null.)

```
// Nazwa pliku: C23ARA2.CPP

// Drukuje i uśrednia sześć wyników testu.

#include <iostream.h>

#include <iomanip.h>

void pr_scores (liczba zmiennoprzecinkowa []); // Prototyp

void main ()

{

char s_name[] = “Tri Star University”;

float scores[6] = {88.7, 90.4, 76.0, 97.0, 100.0, 86.7};

float average=0.0;

int ctr;

// Call function to print scores.

pr_scores(scores);

// Computes total of scores.

for (ctr=0; ctr<6; ctr++)

{ average += scores[ctr]; }

// Computes the average.

average /= float(6);

cout << “At “ << s_name << “, your class average is “
```

```
<< setprecision(2) << average;

return;

}

void pr_scores(float scores[6])

{

// Prints the six scores.

int ctr;

cout << “Here are your scores:\n”; // Title

for (ctr=0; ctr<6; ctr++)

cout << setprecision(2) << scores[ctr] << “\n”;

return;

}
```

Aby przekazać tablicę do funkcji, musisz podać tylko jej nazwę. Na liście parametrów funkcji odbierającej należy podać typ tablicy i dołączyć jej nawiasy, które informują funkcję, że jest to tablica. (Nie musisz jawnie podawać rozmiaru tablicy na liście parametrów odbiorczych, jak pokazano w prototypie).

4. Aby poprawić łatwość konserwacji programów, zdefiniuj wszystkie rozmiary tablic za pomocą instrukcji const. Co jeśli wziąłeś cztery klasy w następnym semestrze, ale nadal chciałbyś korzystać z tego samego programu? Możesz go zmodyfikować, zmieniając wszystkie cyfry od 6 do 4, ale jeśli zdefiniowałeś rozmiar tablicy za pomocą stałej, musisz zmienić tylko jeden wiersz, aby zmienić limity indeksu dolnego programu. Zauważ, jak następujący program używa stałej dla liczby klas.

```
// Nazwa pliku: C23ARA3.CPP

// Drukuje i uśrednia sześć wyników testu.

#include <iostream.h>

#include <iomanip.h>

void pr_scores (liczba zmiennoprzecinkowa []);

const int CLASS_NUM = 6; // Stała zawiera rozmiar tablicy.

void main ()

{

char s_name [] = „Tri Star University”;

float scores[CLASS_NUM] = {88.7, 90.4, 76.0, 97.0,

100.0, 86.7};

float average=0.0;

int ctr;
```

```
// Calls function to print scores.

pr_scores(scores);

// Computes total of scores.

for (ctr=0; ctr<CLASS_NUM; ctr++)

{ average += scores[ctr]; }

// Computes the average.

average /= float(CLASS_NUM);

cout << “At “ << s_name << “, your class average is “

<< setprecision(2) << average;

return;

}

void pr_scores(float scores[CLASS_NUM])

// Prints the six scores.

int ctr;

cout << “Here are your scores:\n”; // Title

for (ctr=0; ctr<CLASS_NUM; ctr++)

cout << setprecision(2) << scores[ctr] << “\n”;

return;

}
```

W tak prostym przykładzie użycie stałej maksymalnego indeksu dolnego może nie wydawać się dużą zaletą. Jeśli piszesz większy program, który przetwarzał kilka tablic, zmiana stałej w górnej części programu byłaby znacznie łatwiejsza niż wyszukiwanie programu dla każdego wystąpienia tego odwołania do tablicy. Używanie stałych dla rozmiarów tablic ma dodatkową zaletę, chroniąc cię przed przekroczeniem granic indeksu dolnego. Nie musisz pamiętać indeksu dolnego podczas przechodzenia przez tablice; możesz zamiast tego użyć stałej.

Inicjalizacja elementów w programie

Rzadko kiedy znasz ich zawartość, kiedy je deklarujesz. Zwykle wypełniasz tablicę danymi wejściowymi użytkownika lub danymi pliku dyskowego. Pętla for jest doskonałym narzędziem do zapętlania tablic po wypełnieniu ich wartościami.

UWAGA: Nazwa tablicy nie może pojawić się po lewej stronie instrukcji przypisania.

Nie można przypisać jednej tablicy do drugiej. Załóżmy, że chcesz skopiować tablicę o nazwie total_sales do drugiej tablicy o nazwie save_sales. Nie możesz tego zrobić za pomocą następującej instrukcji przypisania:

```
save_sales = total_sales; // Nieważny!
```

Zamiast tego musisz kopiować tablice jeden element na raz, używając pętli, tak jak robi to następująca sekcja kodu: Chcesz skopiować jedną tablicę do drugiej. Musisz zrobić to jeden element na raz, więc potrzebujesz licznika. Zainicjuj zmienną o nazwie ctr na 0; wartość ctr reprezentuje pozycję w tablicy.

1. Przypisz element zajmujący pozycję w pierwszej tablicy reprezentowanej przez wartość ctr do tej samej pozycji w drugiej tablicy.

2. Jeśli licznik jest mniejszy niż rozmiar tablicy, dodaj go do licznika. Powtórz krok pierwszy.

```
for (ctr=0; ctr<ARRAY_SIZE; ctr++)

{ saved_sales[ctr] = total_sales[ctr]; }
```

Poniższe przykłady ilustrują metody inicjowania tablic w programie. Po zapoznaniu się z przetwarzaniem dysku w dalszej części książki nauczysz się czytać wartości tablic z pliku dysku.

Przykłady

1. Poniższy program używa operatora przypisania do przypisania 10 temperatur do tablicy.

```
// Nazwa pliku: C23ARA4.CPP

// Wypełnia tablicę 10 wartościami temperatur.

#include <iostream.h>

#include <iomanip.h>

const int NUM_TEMPS = 10;

void main ()

{

float temps [NUM_TEMPS];

int ctr;

temps[0] = 78.6; // Subscripts always begin at 0.

temps[1] = 82.1;

temps[2] = 79.5;

temps[3] = 75.0;

temps[4] = 75.4;

temps[5] = 71.8;

temps[6] = 73.3;

temps[7] = 69.5;

temps[8] = 74.1;

temps[9] = 75.7;

// Print the temps.
```

```
cout << “Daily temperatures for the last “ <<

NUM_TEMPS << “ days:\n”;

for (ctr=0; ctr<NUM_TEMPS; ctr++)

{ cout << setprecision(1) << temps[ctr] << “\n”; }

return;

}
```

2. Poniższy program używa pętli for i cin do przypisania ośmiu liczb całkowitych wprowadzonych indywidualnie przez użytkownika. Następnie program wypisuje sumę liczb.

```
// Nazwa pliku: C23TOT.CPP

// Sumuje osiem wartości wejściowych od użytkownika.

#include <iostream.h>

const int NUM = 8;

void main ()

{

int nums [NUM];

int total = 0; // Przechowuje łącznie osiem liczb użytkownika.

int ctr;

for (ctr=0; ctr<NUM; ctr++)

{ cout << “Please enter the next number...”;

cin >> nums[ctr];

total += nums[ctr]; }

cout << “The total of the numbers is “ << total << “\n”;

return;

}
```

3. Nie musisz uzyskiwać dostępu do tablicy w tej samej kolejności, w jakiej ją zainicjowałeś. Rozdział 24, „Przetwarzanie macierzy”, pokazuje, jak zmienić kolejność macierzy. Możesz także użyć indeksu dolnego, aby wybrać elementy z tablicy wartości.

Poniższy program wymaga danych sprzedaży za poprzednie 12 miesięcy. Użytkownicy mogą następnie wpisać miesiąc, który chcą zobaczyć. Następnie drukowane są dane dotyczące sprzedaży w tym miesiącu, bez przeszkadzania liczbom z innych miesięcy. W ten sposób zaczynasz budować program wyszukiwania w celu znalezienia żądanych danych: Przechowujesz dane w tablicy (lub w pliku dyskowym, który można odczytać w tablicy, jak dowiesz się później), a następnie zaczekaj na żądanie użytkownika, aby zobaczyć określone fragmenty danych.

```
// Nazwa pliku: C23SAL.CPP
```

```
// Przechowuje dwanaście miesięcy sprzedaży i
// drukuje wybrane.
#include <iostream.h>
#include <ctype.h>
#include <conio.h>
#include <iomanip.h>
const int NUM = 12;
void main ()
{
float sales[NUM];
int ctr, ans;
int req_month; // Holds user's request.
// Fill the array.
cout << "Please enter the twelve monthly sales values\n";
for (ctr=0; ctr<NUM; ctr++)
{ cout << "What are sales for month number "
<< ctr+1 << "? \n";
cin >> sales[ctr]; }
// Wait for a requested month.
for (ctr=0; ctr<25; ctr++)
{ cout << "\n"; } // Clears the screen.
cout << "*** Sales Printing Program ***\n";
cout << "Prints any sales from the last " << NUM
<< " months\n\n";
do
{ cout << "For what month (1-" << NUM << ") do you want "
<< "to see a sales value? ";
cin >> req_month;
// Adjust for zero-based subscript.
cout << "\nMonth " << req_month <<
"'s sales are " << setprecision(2) <<
```

```
sales[req_month-1];

cout << "\nDo you want to see another (Y/N)? ";

ans=getch();

ans=toupper(ans);

} while (ans == 'Y');

return;

}
```

Zwróć uwagę na pomocną procedurę czyszczenia ekranu, która drukuje 23 znaki nowego wiersza. Ta procedura przewija ekran, aż będzie pusty. (Większość kompilatorów ma lepszą wbudowaną funkcję czyszczenia ekranu, ale standard AT&T C++ nie oferuje takiej funkcji, ponieważ kompilator jest zbyt ściśle powiązany z konkretnym sprzętem.) Poniżej znajduje się drugi ekran tego programu. Po wprowadzeniu do tablicy 12 wartości sprzedaży, można zażądać jednej lub wszystkich pojedynczo, po prostu podając numer miesiąca (numer indeksu dolnego).

```
*** Sales Printing Program ***

Prints any sales from the last 12 months

For what month (1-12) do you want to see a sales value? 2

Month 2's sales are 433.22

Do you want to see another (Y/N)?

For what month (1-12) do you want to see a sales value? 5

Month 5's sales are 123.45

Do you want to see another (Y/N)?
```

Ćwiczenia

1. Napisz program do przechowywania wieku sześciu znajomych w jednej tablicy. Przechowuj każdy z sześciu grup wiekowych za pomocą operatora przypisania. Wydrukuj wiek na ekranie.

2. Zmodyfikuj program w ćwiczeniu 1, aby wydrukować wiek w odwrotnej kolejności.

3. Napisz prosty program danych do śledzenia ocen stacji radiowych (1, 2, 3, 4 lub 5) za poprzednie 18 miesięcy. Użyj cin, aby zainicjować tablicę z ocenami. Wydrukuj oceny na ekranie z odpowiednim tytułem.

4. Napisz program do przechowywania liczb od 1 do 100 w tablicy 100 elementów całkowitych. (Wskazówka: indeksy dolne powinny zaczynać się od 0, a kończyć na 99.)

5. Napisz program, którego właściciel małej firmy może użyć do śledzenia klientów. Przypisz każdemu klientowi numer (od 0). Za każdym razem, gdy klient coś kupuje, zapisz sprzedaż w elemencie, który odpowiada numerowi klienta (to znaczy następny nieużywany element tablicy). Gdy właściciel sklepu zasygnalizuje koniec dnia, wydrukuj raport składający się z każdego numeru klienta z pasującą sprzedażą, całkowitą wartością sprzedaży i średnią wartością sprzedaży na klienta.

Podsumowanie

Teraz wiesz, jak deklarować i inicjować tablice składające się z różnych typów danych. Możesz zainicjować tablicę, kiedy ją deklarujesz lub w treści twojego programu. Elementy tablicy są znacznie łatwiejsze do przetworzenia niż inne zmienne, ponieważ każdy ma inną nazwę. C++ ma zaawansowane techniki sortowania i wyszukiwania, dzięki którym Twoje programy są jeszcze bardziej użyteczne. Następny rozdział przedstawia te techniki i pokazuje jeszcze inne sposoby uzyskiwania dostępu do elementów tablicy.

Przetwarzanie tablic

C++ zapewnia wiele sposobów dostępu do tablic. Jeśli programowałeś w innych językach komputerowych, przekonasz się, że niektóre techniki indeksowania tablic w C++ są unikalne. Tablice w języku C++ są ściśle powiązane ze wskaźnikami. Ponieważ wskaźniki są tak potężne i ponieważ poznanie tablic stanowi dobrą podstawę do poznania wskaźników, w tym rozdziale podjęto próbę szczegółowego opisania sposobu odwoływania się do tablic. W tym rozdziale omówiono różne typy przetwarzania tablic. Dowiesz się, jak przeszukiwać tablicę pod kątem jednej lub więcej wartości, znajdować najwyższe i najniższe wartości w tablicy oraz sortować tablicę w kolejności numerycznej lub alfabetycznej. Tu przedstawiono następujące pojęcia:

♦ Przeszukiwanie tablic

♦ Znajdowanie najwyższych i najniższych wartości w tablicach

♦ Sortowanie tablic

♦ Zaawansowane indeksowanie za pomocą tablic

Wielu programistów postrzega tablice jako punkt zwrotny. Zrozumienie przetwarzania tablic sprawia, że twoje programy są dokładniejsze i pozwalają na wydajniejsze programowanie.

Przeszukiwanie tablic

Tablice są jednym z podstawowych sposobów przechowywania danych w programach C++. Wiele rodzajów programów nadaje się do przetwarzania list (tablic) danych, takich jak program płac pracowniczych, badania naukowe kilku chemikaliów lub przetwarzanie kont klientów. Jak wspomniano poprzednio, dane z tablicy zwykle są odczytywane z pliku dyskowego. Późniejsze rozdziały opisują przetwarzanie plików na dysku. Na razie powinieneś zrozumieć, jak manipulować tablicami, aby zobaczyć dane dokładnie tak, jak chcesz. Czasami tak się dzieje, ale nie zawsze jest to najodpowiedniejsza metoda patrzenia na dane. Załóżmy na przykład, że liceum używało raportów C++ w swoich programach. Załóżmy również, że szkoła chciała zobaczyć listę 10 najlepszych średnich ocen. Nie można wydrukować pierwszych 10 średnich ocen na liście średnich uczniów, ponieważ 10 najlepszych GPA może nie pojawiać się (i prawdopodobnie nie będzie) jako pierwszych 10 elementów tablicy. Ponieważ GPA nie byłyby w żadnej sekwencji, program musiałby sortować tablicę w kolejności numerycznej, od wysokich GPA do niskich, lub przeszukać tablicę pod kątem 10 najwyższych GPA. Potrzebujesz metody umieszczania tablic w określonej kolejności. Nazywa się to sortowaniem tablicy. Podczas sortowania tablicy ustawiasz ją w określonej kolejności, na przykład w kolejności alfabetycznej lub numerycznej. Słownik jest posortowany, podobnie jak książka telefoniczna. Kiedy odwracasz kolejność sortowania, nazywa się to malejącym sortowaniem. Na przykład, jeśli chcesz spojrzeć na listę wszystkich pracowników w kolejności malejącej, pensje będą najlepiej drukowane jako najlepiej opłacani pracownicy.

Zanim nauczysz się sortować, pomocne byłoby nauczenie się wyszukiwania wartości w tablicy. To wstępny krok w nauce sortowania. Co się stanie, jeśli jeden z tych uczniów otrzyma zmianę oceny? Komputer musi mieć dostęp do oceny danego ucznia, aby ją zmienić (bez wpływu na innych). Jak pokazuje następna sekcja, programy mogą wyszukiwać określone elementy tablicy.

UWAGA: C++ zapewnia metodę sortowania i przeszukiwania list ciągów, ale nie zrozumiesz, jak to zrobić, dopóki nie dowiesz się o wskaźnikach. Przykłady i algorytmy sortowania i wyszukiwania przedstawione tutj pokazują sortowanie i wyszukiwanie tablic liczb. Te same koncepcje będą miały

zastosowanie (i faktycznie będą znacznie bardziej użyteczne w aplikacjach „rzeczywistych"), gdy nauczysz się, jak przechowywać listy nazw w C++.

Poszukiwanie wartości

Nie musisz znać żadnych nowych poleceń, aby przeszukać tablicę pod kątem wartości. Zasadniczo potrzebne są instrukcje if i pętla for . Aby wyszukać tablicę pod kątem określonej wartości, spójrz na każdy element w tej tablicy i porównaj ją z instrukcją if, aby sprawdzić, czy są one zgodne. Jeśli nie, przeszukujesz tablicę. Jeśli zabraknie elementów tablicy przed znalezieniem wartości, nie ma jej w tablicy. Możesz wykonać kilka różnych rodzajów wyszukiwania. Może być konieczne znalezienie najwyższej lub najniższej wartości na liście liczb. Jest to przydatne, gdy masz dużo danych i chcesz poznać ich ekstremum (takie jak najwyższy i najniższy region sprzedaży w twoim oddziale). Możesz także wyszukać tablicę, aby sprawdzić, czy zawiera pasującą wartość. Na przykład można sprawdzić, czy element znajduje się już w ekwipunku, wyszukując dopasowanie w tablicy numerów części. Następujące programy ilustrują niektóre z tych technik wyszukiwania tablicowego.

Przykłady

1. Aby znaleźć najwyższą liczbę w tablicy, porównaj każdy element z pierwszym. Jeśli znajdziesz wyższą wartość, stanie się ona podstawą dla reszty tablicy. Kontynuuj, aż dojdziesz do końca tablicy, a uzyskasz najwyższą wartość, jak pokazuje poniższy program. Zidentyfikuj program i dołącz plik nagłówkowy we / wy. Chcesz znaleźć najwyższą wartość w tablicy, więc zdefiniuj rozmiar tablicy jako stałą, a następnie zainicjuj tablicę. Pętlę przez tablicę, porównując każdy element do najwyższej wartości. Jeśli element jest wyższy niż najwyższa zapisana wartość, zapisz element jako nową wysoką wartość. Wyświetl najwyższą wartość znalezioną w tablicy.

```
// Nazwa pliku: C24HIGH.CPP

// Znajduje najwyższą wartość w tablicy.

#include <iostream.h>

const int SIZE = 15;

void main()

{

// Puts some numbers in the array.

int ara[SIZE]={5,2,7,8,36,4,2,86,11,43,22,12,45,6,85};

int high_val, ctr;

high_val = ara[0]; // Initializes with first

// array element.

for (ctr=1; ctr<SIZE; ctr++)

{ // Stores current value if it is

// the higher than the highest.

if (ara[ctr] > high_val)

{ high_val = ara[ctr]; }
```

```
}

cout << “The highest number in the list is “

<< high_val << “\n”;

return;

}
```

Dane wyjściowe programu są następujące:

The highest number in the list is 86.

Musisz zapisać element wtedy i tylko wtedy, gdy jest on wyższy niż ten, który porównujesz. Znalezienie najmniejszej liczby w tablicy jest równie łatwe, z tym wyjątkiem, że określasz, czy każdy kolejny element tablicy jest mniejszy niż najniższa znaleziona do tej pory wartość.

2. Poniższy przykład rozwija się w stosunku do poprzedniego, znajdując najwyższą i najniższą wartość. Najpierw zapisz pierwszy element tablicy zarówno w najwyższej, jak i najniższej zmiennej, aby rozpocząć wyszukiwanie. Zapewnia to, że każdy element po tym jest testowany, aby sprawdzić, czy jest wyższy czy niższy od pierwszego. W tym przykładzie użyto również funkcji rand() , aby wypełnić tablicę losowymi wartościami od 0 do 99 poprzez zastosowanie operatora modułu (%) i 100 względem dowolnej wartości rand() produkuje. Program drukuje całą tablicę przed uruchomieniem wyszukiwania najwyższej i najniższej.

```
/ Nazwa pliku: C24HILO.CPP

// Znajduje najwyższą i najniższą wartość w tablicy.

#include <iostream.h>

#include <stdlib.h>

const int SIZE = 15;

void main()

{

int ara[SIZE];

int high_val, low_val, ctr;

// Fills array with random numbers from 0 to 99.

for (ctr=0; ctr<SIZE; ctr++)

{ ara[ctr] = rand() % 100; }

// Prints the array to the screen.

cout << “Here are the “ << SIZE << “ random numbers:\n”;

for (ctr=0; ctr<SIZE; ctr++)

{ cout << ara[ctr] << “\n”; }

cout << “\n\n”; // Prints a blank line.
```

```
high_val = ara[0]; // Initializes first element to
// both high and low.
low_val = ara[0];
for (ctr=1; ctr<SIZE; ctr++)
{ // Stores current value if it is
// higher than the highest.
if (ara[ctr] > high_val)
{ high_val = ara[ctr]; }
if (ara[ctr] < low_val)
{ low_val = ara[ctr]; }
}
cout << “The highest number in the list is “ <<
high_val << “\n”;
cout << “The lowest number in the list is “ <<
low_val << “\n”;
return;
}
```

Oto wynik tego programu:

```
Here are the 15 random numbers:
46
30
82
90
56
17
95
15
48
26
4
58
```

71

79

92

The highest number in the list is 95

The lowest number in the list is 4

3. Następny program wypełnia tablicę numerami części z wykazu. Musisz użyć wyobraźni, ponieważ tablica zapasów zwykle wypełniałaby więcej tablic, była inicjowana z pliku dyskowego i była częścią większego zestawu tablic, które zawierają opisy, ilości, koszty, ceny sprzedaży i tak dalej. W tym przykładzie instrukcje przypisania inicjują tablicę. Ważną ideą tego programu nie jest inicjalizacja tablicy, ale metoda jej przeszukiwania.

UWAGA: Jeśli nowo wprowadzony numer części jest już zapisany, program poinformuje użytkownika. W przeciwnym razie numer części zostanie dodany na końcu tablicy.

```
// Nazwa pliku: C24SERCH.CPP

// Przeszukuje tablicę numerów części pod kątem wartości wejściowej. Gdyby

// wprowadzony numer części nie znajduje się w tablicy, jest

// dodany. Jeśli numer części znajduje się w tablicy, pojawia się komunikat

// jest drukowane.

#include <iostream.h>

const int MAX = 100;

void fill_parts(long int parts[MAX]);

void main()

{

long int search_part; // Holds user request.

long int parts[MAX];

int ctr;

int num_parts=5; // Beginning inventory count.

fill_parts(parts); // Fills the first five elements.

do

{

cout << “\n\nPlease type a part number...”;

cout << “(-9999 ends program) “;

cin >> search_part;

if (search_part == -9999)
```

```
{ break; } // Exits loop if user wants.
// Scans array to see whether part is in inventory.
for (ctr=0; ctr<num_parts; ctr++) // Checks each item.
{ if (search_part == parts[ctr]) // If it is in
// inventory...
{ cout << "\nPart " << search_part <<
" is already in inventory";
break;
}
else
{ if (ctr == (num_parts-1) ) // If not there,
// adds it.
{ parts[num_parts] = search_part; // Adds to
// end of array.
num_parts++;
cout << search_part <<
" was added to inventory\n";
break;
}
}
}
} while (search_part != -9999); // Loops until user
// signals end.
return;
}
void fill_parts(long int parts[MAX])
{
// Assigns five part numbers to array for testing.
parts[0] = 12345;
parts[1] = 24724;
parts[2] = 54154;
```

parts[3] = 73496;

parts[4] = 83925;

return;

}

Oto wynik tego programu:

Please type a part number...(-9999 ends program) 34234

34234 was added to inventory

Please type a part number...(-9999 ends program) 83925

Part 83925 is already in inventory

Please type a part number...(-9999 ends program) 52786

52786 was added to inventory

Please type a part number...(-9999 ends program) -9999

Sortowanie tablic

Wiele razy musisz posortować jedną lub więcej tablic. Załóżmy, że weźmiesz listę liczb, zapisz każdą liczbę na osobnej kartce papieru i wyrzuć wszystkie kartki w powietrze. Kroki, które wykonujesz - tasowanie i zmiana kolejności kawałków papieru oraz próba uporządkowania ich - są podobne do czynności wykonywanych przez komputer w celu sortowania liczb lub danych znakowych. Ponieważ sortowanie tablic wymaga wymiany wartości elementów tam iz powrotem, pomaga to, jeśli najpierw poznasz technikę zamiany zmiennych. Załóżmy, że masz dwie zmienne o nazwach score1 i score2. Co jeśli chcesz odwrócić ich wartości (wstawienie score2 do zmiennej score1 i vice versa)? Nie możesz tego zrobić:

wynik1 = wynik2; // Nie zamienia dwóch wartości.

wynik2 = wynik1;

Dlaczego to nie działa? W pierwszym wierszu wartość score1 zostaje zastąpiona wartością score2. Po zakończeniu pierwszego wiersza zarówno wynik1, jak i wynik2 zawierają tę samą wartość. Dlatego druga linia nie może pracować zgodnie z życzeniem. Aby zamienić dwie zmienne, musisz użyć trzeciej zmiennej do przechowywania wyniku pośredniego. (Jest to jedyna funkcja tej trzeciej zmiennej.) Na przykład, aby zamienić score1 i score2, użyj trzeciej zmiennej (w tym kodzie zwanej hold_score), jak w

hold_score = wynik1; // Te trzy linie poprawnie

wynik1 = wynik2; // zamień score1 i score2.

score2 = hold_score;

Wymienia to wartości dwóch zmiennych. Istnieje kilka różnych sposobów sortowania tablic. Metody te obejmują sortowanie bąbelkowe, szybkie sortowanie i sortowanie powłokowe. Podstawowym celem każdej metody jest porównanie każdego elementu tablicy z innym elementem tablicy i zamiana, jeśli wyższa wartość jest mniejsza od drugiej. Teoria leżąca u podstaw tych rodzajów wykracza poza zakres tej książki, jednak sortowanie bąbelkowe jest jedną z najłatwiejszych do zrozumienia. Wartości w tablicy są porównywane ze sobą, para na raz, i zamieniane, jeśli nie są w kolejności jeden po drugim.

Najniższa wartość ostatecznie „unosi się” na górę tablicy, jak bąbelek w szklance sody. Rysunek 24.2 pokazuje listę liczb przed, podczas i po sortowaniu bąbelkowym. Sortowanie bąbelkowe przechodzi przez tablicę i porównuje pary liczb w celu ustalenia, czy należy je zamienić. Może być konieczne wykonanie kilku przebiegów przez tablicę, zanim zostanie ona ostatecznie posortowana (nie są już potrzebne kolejne przebiegi). Inne rodzaje poprawiają sortowanie bąbelkowe. Procedura sortowania bąbelkowego jest łatwa do zaprogramowania, ale jest wolniejsza w porównaniu do wielu innych metod. Następujące programy pokazują sortowanie bąbelkowe w akcji.

Przykłady

1. Poniższy program przypisuje do tablicy 10 liczb losowych od 0 do 99, a następnie sortuje tablicę. Zagnieżdżona pętla for jest idealna do sortowania liczb w tablicy (jak pokazano w funkcji sort_array ()). Zagnieżdżone dla pętli zapewniają ładny mechanizm pracy na parach wartości, wymieniając je w razie potrzeby. Gdy zewnętrzna pętla odlicza listę, odnosząc się do każdego elementu, wewnętrzna pętla porównuje każdą z pozostałych wartości z tymi elementami tablicy.

```
// Nazwa pliku: C24SORT1.CPP

// Sortuje i drukuje listę liczb.

const int MAX = 10;

#include <iostream.h>

#include <stdlib.h>

void fill_array(int ara[MAX]);

void print_array(int ara[MAX]);

void sort_array(int ara[MAX]);

void main()

{

int ara[MAX];

fill_array(ara); // Puts random numbers in the array.

cout << “Here are the unsorted numbers:\n”;

print_array(ara); // Prints the unsorted array.

sort_array(ara); // Sorts the array.

cout << “\n\nHere are the sorted numbers:\n”;

print_array(ara); // Prints the newly sorted array.

return;

}

void fill_array(int ara[MAX])

{

// Puts random numbers in the array.
```

```
int ctr;
for (ctr=0; ctr<MAX; ctr++)
{ ara[ctr] = (rand() % 100); } // Forces number to
// 0-99 range.
return;
}
void print_array(int ara[MAX])
{
// Prints the array.
int ctr;
for (ctr=0; ctr<MAX; ctr++)
{ cout << ara[ctr] << "\n"; }
return;
}
void sort_array(int ara[MAX])
{
// Sorts the array.
int temp; // Temporary variable to swap with
int ctr1, ctr2; // Need two loop counters to
// swap pairs of numbers.
for (ctr1=0; ctr1<(MAX-1); ctr1++)
{ for (ctr2=(ctr1+1); ctr2<MAX; ctr2++) // Test pairs.
{ if (ara[ctr1] > ara[ctr2]) // Swap if this
{ temp = ara[ctr1]; // pair is not in order.
ara[ctr1] = ara[ctr2];
ara[ctr2] = temp; // "Float" the lowest
// to the highest.
}
}
}
return;
```

}

Wyjście z tego programu pojawi się następnie. Jeśli dowolne dwa losowo wygenerowane liczby są takie same, sortowanie bąbelkowe działałoby poprawnie, umieszczając je obok siebie na liście.

Here are the unsorted numbers:

46

30

82

90

56

17

95

15

48

26

Here are the sorted numbers:

15

17

26

30

46

48

56

82

90

95

2. Poniższy program jest taki sam jak poprzedni, z tą różnicą, że drukuje listę liczb w kolejności malejącej. Sortowanie malejące jest równie łatwe do napisania jak sortowanie rosnące. Przy sortowaniu rosnącym (od niskich do wysokich) porównujesz pary wartości, sprawdzając, czy pierwsza jest większa od drugiej. Przy sortowaniu malejącym testujesz zobacz, czy pierwszy jest mniejszy niż drugi.

// Nazwa pliku: C24SORT2.CPP

// Sortuje i drukuje listę liczb w odwrotnej kolejności

// i kolejność malejąca.

```
const int MAX = 10;
#include <iostream.h>
#include <stdlib.h>
void fill_array(int ara[MAX]);
void print_array(int ara[MAX]);
void sort_array(int ara[MAX]);
void main()
{
int ara[MAX];
fill_array(ara); // Puts random numbers in the array.
cout << “Here are the unsorted numbers:\n”;
print_array(ara); // Prints the unsorted array.
sort_array(ara); // Sorts the array.
cout << “\n\nHere are the sorted numbers:\n”;
print_array(ara); // Prints the newly sorted array.
return;
}
void fill_array(int ara[MAX])
{
// Puts random numbers in the array.
int ctr;
for (ctr=0; ctr<MAX; ctr++)
{ ara[ctr] = (rand() % 100); } // Forces number
// to 0-99 range.
return;
}
void print_array(int ara[MAX])
{
// Prints the array
int ctr;
for (ctr=0; ctr<MAX; ctr++)
```

```
{ cout << ara[ctr] << "\n"; }
return;
}
void sort_array(int ara[MAX])
{
// Sorts the array.
int temp; // Temporary variable to swap with.
int ctr1, ctr2; // Need two loop counters
// to swap pairs of numbers.
for (ctr1=0; ctr1<(MAX-1); ctr1++)
{ for (ctr2=(ctr1+1); ctr2<MAX; ctr2++) // Test pairs
// Notice the difference in descending (here)
// and ascending.
{ if (ara[ctr1] < ara[ctr2]) // Swap if this
{ temp = ara[ctr1]; // pair is not in order.
ara[ctr1] = ara[ctr2];
ara[ctr2] = temp; // "Float" the lowest
// to the highest.
}
}
}
return;
}
```

WSKAZÓWKA: Możesz zapisać funkcje sortowania poprzednich programów w dwóch osobnych plikach o nazwach sort_ascend i sort_descend. Gdy musisz posortować dwie różne tablice, # włącz te pliki do swoich programów. Co więcej, skompiluj każdą z tych procedur osobno i połącz tę, której potrzebujesz z programem. (Musisz sprawdzić instrukcję kompilatora, aby dowiedzieć się, jak to zrobić.) Możesz sortować tablice znaków równie łatwo, jak sortujesz tablice numeryczne. C++ używa zestawu znaków ASCII do swoich porównań sortowania.

Zaawansowane odwołania do tablic

Dotychczasowa notacja tablicowa jest powszechna w językach programowania komputerowego. Większość języków używa indeksów dolnych w nawiasach (lub nawiasach) w celu odniesienia do poszczególnych elementów tablicy. Na przykład wiesz, że następujące odwołania do tablic opisują

pierwszy i piąty element tablicy o nazwie sprzedaż (pamiętaj, że początkowy indeks dolny ma zawsze wartość 0):

sales[0]

sales[4]

C++ zapewnia inne podejście do odwoływania się do tablic. Chociaż tytuł tej sekcji zawiera słowo „zaawansowane", to metoda odwoływania się do tablicy nie jest trudna. Jest jednak zupełnie inaczej, zwłaszcza jeśli znasz podejście do innego języka programowania. Nie ma nic złego w odniesieniu do elementów tablicy w sposób, w jaki widzieliście do tej pory, jednak drugie podejście, unikalne dla C i C++, będzie pomocne, gdy poznasz wskaźniki. W rzeczywistości programiści C++, którzy programowali od kilku lat, rzadko używają notacji w indeksie dolnym W C++ nazwa tablicy to nie tylko etykieta, z której mogą korzystać programy. Dla C++ nazwa tablicy jest rzeczywistym adresem, od którego zaczyna się pierwszy element w pamięci. Załóżmy, że zdefiniowałeś tablicę o nazwie amounts z następującym stwierdzeniem:

int amount [6] = {4, 1, 3, 7, 9, 2};

Rysunek 24.3 pokazuje, jak ta tablica jest przechowywana w pamięci. Rysunek pokazuje tablicę zaczynającą się od adresu 405,332. (Rzeczywiste adresy zmiennych są określane przez komputer podczas ładowania i uruchamiania skompilowanego programu.) Zauważ, że nazwa tablicy, kwoty, znajduje się gdzieś w pamięci i zawiera adres kwot [0] lub 403 532.

Do tablicy można odwoływać się za pomocą zwykłej notacji w indeksie dolnym lub modyfikując adres tablicy. Poniższe odnoszą się do trzeciego elementu amounts:

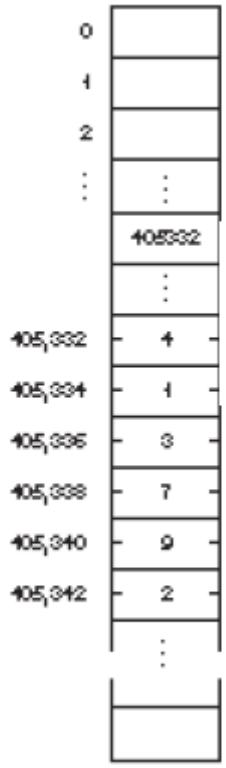
amounts[3] i (amounts + 3) [0]

Ponieważ C++ traktuje nazwę tablicy jako adres w pamięci, który zawiera lokalizację pierwszego elementu tablicy, nic nie stoi na przeszkodzie, aby użyć innego adresu jako adresu początkowego i odwoływać się stamtąd. Idąc dalej o krok, każdy z poniższych odnosi się również do trzeciego elementu amounts:

(amounts + 0) [3] i (amounts + 2) [1] i (amounts -2) [5]

(1 + amounts) [2] i (3 + amounts) [0] i (amounts + 1) [2]

Możesz wydrukować dowolny z tych elementów tablicy za pomocą cout.



Gdy wyświetlasz ciągi wewnątrz tablic znaków, odwołując się do tablic według ich zmodyfikowanych adresów są bardziej przydatne niż z liczbami całkowitymi. Załóżmy, że zachowałeś trzy ciągi znaków w jednej tablicy znaków. Możesz zainicjować tę tablicę za pomocą następującej instrukcji:

char names [] = {'T', 'e', 'd', '\ 0', 'E', 'v', 'a', '\ 0', 'S', 'a', 'm ',' \ 0 '};

Nazwa tablicy, nazwy, zawiera adres pierwszego elementu, nazwy [0] (litera T).

PRZESTROGA: Musisz zawrzeć nazwy tablic w nawiasach, jeśli chcesz zmodyfikować nazwę, jak pokazano w tych przykładach. Poniższe nie są równoważne:

(2+amounts)[1] and 2+amounts[1]

Pierwszy przykład dotyczy amounts[3] (czyli 7). Drugi przykład przyjmuje wartość amounts[1] (która w tym przykładzie wynosi 1) tablica) i dodaje do niej 2 (co daje wartość 3). Ta druga metoda odwoływania się do tablic może wydawać się większym problemem niż jest warta, ale nauczenie się odwoływania do tablic w ten sposób znacznie ułatwi przejście do wskaźników. Nazwa tablicy jest w rzeczywistości wskaźnikiem, ponieważ tablica zawiera adres pierwszego elementu tablicy („wskazuje” na początek tablicy). Musisz jeszcze zobaczyć tablicę znaków, która zawiera więcej niż jeden ciąg, ale C++ pozwala na to. Problem z taką tablicą polega na tym, jak odwołujesz się, a zwłaszcza jak drukujesz, drugi i trzeci ciąg. Jeśli miałbyś wydrukować tę tablicę za pomocą cout:

cout << names;

C++ wypisze następujące:

Ted

Ponieważ cout wymaga adresu początkowego, możesz wydrukować trzy ciągi znaków z następującymi coutami:

cout << names; // Prints Ted

cout << (names+4); // Prints Eva

cout << (names+8); // Prints Sam

Aby przetestować twoje zrozumienie, co drukują następujące kreacje?

cout << (names+1);

cout << (names+6);

Pierwsze cout drukuje ed. Edowane znaki zaczynają się od (nazwy + 1), a cout przestaje drukować, gdy osiągnie zero. Drugie cout drukuje a. Dodanie szóstki do adresu w nazwach daje adres, pod którym znajduje się a. „Łańcuch" ma tylko jeden znak, ponieważ zero zero pojawia się w tablicy bezpośrednio po a. Podsumowując tablice znaków, następujące elementy odnoszą się do poszczególnych elementów tablicy (pojedyncze znaki):

names[2] and (names+1)[1]

Poniższe informacje odnoszą się tylko do adresów i jako takie można wydrukować pełne ciągi znaków z cout:

names i (names+4)

UWAGA: Nigdy nie używaj kodu sterującego % c printf () do wyś1)ietlania odwołania do adresu, nawet jeśli ten adres zawiera znak. Wydrukuj ciągi znaków, podając adres za pomocą% s, a pojedyncze znaki, określając element znaku za pomocą% c.

Poniższe przykłady różnią się nieco od większości, które widziałeś. Nie wykonują one pracy w „świecie rzeczywistym", ale zostały zaprojektowane jako przykłady do nauki w celu zapoznania się z tą nową metodą odwoływania się do tablic. Następne kilka rozdziałów opisuje te metody.

Przykłady

1. Poniższy program przechowuje liczby od 100 do 600 w tablicy, a następnie drukuje elementy przy użyciu nowej metody indeksowania tablicy.

// Nazwa pliku: C24REF1.CPP

// Drukuj elementy tablicy liczb całkowitych na różne sposoby.

#include <iostream.h>

void main()

{

int num[6] = {100, 200, 300, 400, 500, 600};

cout << "num[0] is \t" << num[0] << "\n";

cout << "(num+0)[0] is \t" << (num+0)[0] << "\n";

cout << "(num-2)[2] is \t" << (num-2)[2] << "\n\n";

cout << "num[1] is \t" << num[1] << "\n";

cout << "(num+1)[0] is \t" << (num+1)[0] << "\n\n";

```
cout << “num[5] is \t” << num[5] << “\n”;
cout << “(num+5)[0] is \t” << (num+5)[0] << “\n”;
cout << “(num+2)[3] is \t” << (num+2)[3] << “\n\n”;
cout << “(3+num)[1] is \t” << (3+num)[1] << “\n”;
cout << “3+num[1] is \t” << 3+num[1] << “\n”;
return;
}
```

Oto wynik tego programu:

```
num[0] is 100
(num+0)[0] is 100
(num-2)[2] is 100
num[1] is 200
(num+1)[0] is 200
num[5] is 600
(num+5)[0] is 600
(num+2)[3] is 600
(3+num)[1] is 500
3+num[1] is 203
```

2. Poniższy program drukuje ciągi i znaki z tablicy znaków. Wszystkie krzyże drukują się poprawnie.

```
// Nazwa pliku: C24REF2.CPP
// Drukuje elementy i ciągi z tablicy.
#include <iostream.h>
void main()
{
char names[]={‘T’,’e’,’d’,’\0',’E’,’v’,’a’,’\0',
’S’, ‘a’,’m’,’\0'};
// Must use extra percent (%) to print %s and %c.
cout << “names “ << names << “\n”;
cout << “names+0 “ << names+0 << “\n”;
cout << “names+1 “ << names+1 << “\n”;
cout << “names+2 “ << names+2 << “\n”;
```

```
cout << “names+3 “ << names+3 << “\n”;
cout << “names+5 “ << names+5 << “\n”;
cout << “names+8 “ << names+8 << “\n\n”;
cout << “(names+0)[0] “ << (names+0)[0] << “\n”;
cout << “(names+0)[1] “ << (names+0)[1] << “\n”;
cout << “(names+0)[2] “ << (names+0)[2] << “\n”;
cout << “(names+0)[3] “ << (names+0)[3] << “\n”;
cout << “(names+0)[4] “ << (names+0)[4] << “\n”;
cout << “(names+0)[5] “ << (names+0)[5] << “\n\n”;
cout << “(names+2)[0] “ << (names+2)[0] << “\n”;
cout << “(names+2)[1] “ << (names+2)[1] << “\n”;
cout << “(names+1)[4] “ << (names+1)[4] << “\n\n”;
return;
}
```

Sprawdź wyniki pokazane poniżej, porównując je z programem. Dowiesz się więcej na temat ciągów, znaków i odwołań do tablicy znaków, studiując ten jeden przykład, niż z 20 stron opisu tekstowego.

```
names Ted
names+0 Ted
names+1 ed
names+2 d
names+3
names+5 va
names+8 Sam
(names+0)[0] T
(names+0)[1] e
(names+0)[2] d
(names+0)[3]
(names+0)[4] E
(names+0)[5] v
(names+2)[0] d
(names+2)[1]
```

(names+1)[4] v

Ćwiczenia

1. Napisz program do przechowywania sześciu grup wiekowych znajomych w jednej tablicy. Przypisuj grupy wiekowe w losowej kolejności. Drukuj epoki, od niskiego do wysokiego, na ekranie.

2. Zmodyfikuj program w ćwiczeniu 1, aby drukować wieki w kolejności malejącej.

3. Korzystając z nowego podejścia do tablic indeksowania, przepisz programy w ćwiczeniach 1 i 2. Zawsze umieszczaj 0 w nawiasach indeksu dolnego, modyfikując zamiast tego adres (użyj (ages + 3) [0] zamiast ery [3]).

4. Czasami w programach, które muszą śledzić więcej niż jedną listę powiązanych wartości, używane są tablice równoległe. Załóżmy na przykład, że trzeba utrzymywać zapasy, śledząc liczby całkowite, ceny i ilości każdego elementu. Wymagałoby to trzech tablic: tablica liczb całkowitych, tablica cen zmiennoprzecinkowych i tablica liczb całkowitych. Każda tablica miałaby taką samą liczbę elementów (łączna liczba części w ekwipunku). Napisz program do prowadzenia takiego inwentarza i zarezerwuj wystarczającą liczbę elementów na 100 części w ekwipunku. Przedstaw użytkownikowi ekran wprowadzania. Gdy użytkownik wprowadzi numer części, wyszukaj tablicę numerów części. Po zlokalizowaniu pozycji części wydrukuj odpowiednią cenę i ilość. Jeśli część nie istnieje, włącz możliwość dodania jej do zapasów wraz z odpowiednią ceną i ilością.

Podsumowanie

Zaczynasz dostrzegać prawdziwą moc języków programowania. Tablice umożliwiają wyszukiwanie i sortowanie list wartości. Sortowanie i wyszukiwanie to, co robią najlepiej komputery; komputery mogą szybko skanować setki, a nawet tysiące wartości, szukając dopasowania. Ręczne skanowanie plików papieru w poszukiwaniu odpowiedniej liczby zajmuje znacznie więcej czasu. Przechodząc przez tablice, twój program może szybko skanować, drukować, sortować i obliczać listę wartości. Masz teraz narzędzia do sortowania list liczb, a także wyszukiwania wartości na liście. Wykorzystanych tu koncepcji użyjesz również do sortowania i przeszukiwania list danych ciągów znaków, gdy dowiesz się nieco więcej o tym, jak C++ manipuluje ciągami znaków i wskaźnikami. Aby pomóc w budowaniu solidnych podstaw dla tego i bardziej zaawansowanego materiału, teraz możesz umieć odwoływać się do elementów tablicy bez użycia tradycyjnych indeksów dolnych. Po opanowaniu tego rozdziału następny będzie łatwy .Nie wszystkie listy danych nadają się do macierzy, ale powinieneś być na nie przygotowany, kiedy ich potrzebujesz.

Tablice wielowymiarowe

Niektóre dane mieszczą się na listach, takie jak dane omówione w poprzednich dwóch rozdziałach, a inne dane lepiej nadają się do tabel informacji. Ten rozdział posuwa tablice o krok dalej. W poprzednich rozdziałach wprowadzono tablice jednowymiarowe; tablice, które mają tylko jeden indeks dolny i reprezentują listy wartości. Tu przedstawiono tablice o więcej niż jednym wymiarze, zwane tablicami wielowymiarowymi. Tablice wielowymiarowe, czasami nazywane tabelami lub macierzami, mają co najmniej dwa wymiary (wiersze i kolumny). Wiele razy mają więcej niż dwa. W tym rozdziale przedstawiono następujące pojęcia:

♦ Tablice wielowymiarowe

♦ Rezerwowanie pamięci dla tablic wielowymiarowych

♦ Umieszczanie danych w tablicach wielowymiarowych

♦ Używanie zagnieżdżonych pętli do przetwarzania tablic wielowymiarowych

Jeśli rozumiesz tablice jednowymiarowe, powinieneś to zrobić ,jeśli nie ma problemu ze zrozumieniem tablic, które mają więcej niż jeden wymiar.

Podstawy macierzy wielowymiarowych

Tablica wielowymiarowa to tablica z więcej niż jednym indeksem dolnym. Podczas gdy tablica jednowymiarowa jest listą wartości, tablica wielowymiarowa symuluje tablicę wartości lub wielokrotność wartości tabeli. Najczęściej używana tabela jest dwuwymiarowa tablica (tablica z dwoma indeksami dolnymi). Załóżmy, że drużyna softballowa chce śledzić rekordy mrugnięcia swoich graczy. Zespół rozegrał 10 gier, aw drużynie jest 15 graczy. Tabela pokazuje rekord drużyny.

Gracz: Gra

Imię i nazwisko: 1 2 3 4 5 6 7 8 9 10

Adams :2 1 0 0 2 3 3 1 1 2

Berryhill : 1 0 3 2 5 1 2 2 1 0

Downing : 1 0 2 1 0 0 0 0 2 0

Edwards : 0 3 6 4 6 4 5 3 6 3

Franks : 2 2 3 2 1 0 2 3 1 0

Grady : 1 3 2 0 1 5 2 1 2 1

Howard : 3 1 1 1 2 0 1 0 4 3

Jones : 2 2 1 2 4 1 0 7 1 0

Martin : 5 4 5 1 1 0 2 4 1 5

Powers : 2 2 3 1 0 2 1 3 1 2

Smith : l 1 1 2 1 3 4 1 0 3 2

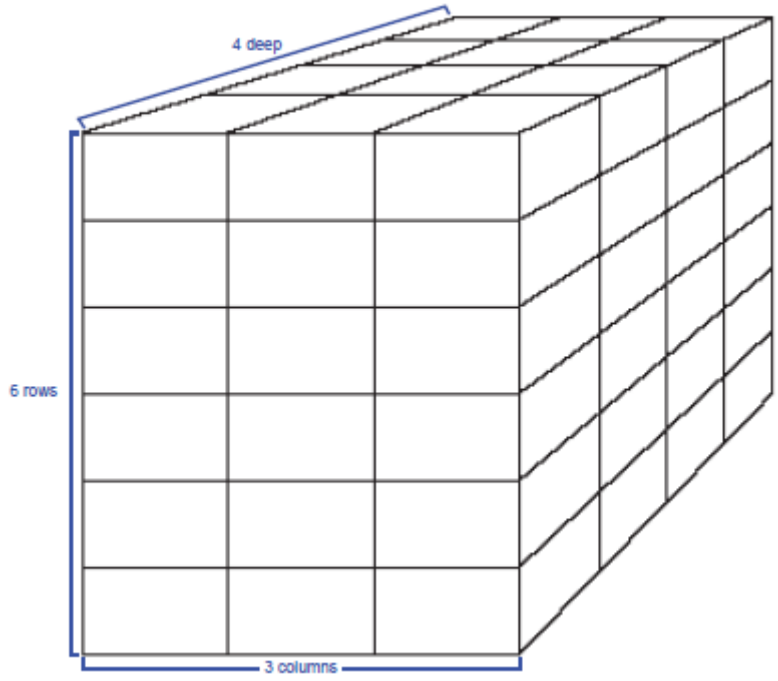
Smithtown : 1 0 1 2 1 0 3 4 1 2

Townsend : 0 0 0 0 0 0 1 0 0 0

Ulmer : 2 2 2 2 2 1 1 3 1 3

Williams : 2 3 1 0 1 2 1 2 0 3

Czy widzisz, że tablica do softballa jest tablicą dwuwymiarową? Ma wiersze (pierwszy wymiar) i kolumny (drugi wymiar). Dlatego nazywa się to dwuwymiarową tabelą z 15 rzędami i 10 kolumnami. (Ogólnie liczba rzędów jest określona jako pierwsza.) Każdy wiersz ma nazwę gracza, a każda kolumna ma przypisany numer gry, ale nie są one częścią rzeczywistych danych. Dane składają się tylko z 150 wartości (15 wierszy na 10 kolumn). Dane w dwuwymiarowej tabeli zawsze są tego samego rodzaju danymi; w tym przypadku każda wartość jest liczbą całkowitą. Gdyby była to tabela wynagrodzeń, każdy element byłby zmiennoprzecinkowy po przecinku. Liczba wymiarów, w tym przypadku dwa, odpowiada wymiarom w świecie fizycznym. Tablica jednowymiarowa jest linią lub listą wartości. Dwa wymiary reprezentują zarówno długość, jak i szerokość. Piszesz na kartce papieru w dwóch wymiarach; dwa wymiary reprezentują płaską powierzchnię. Trzy wymiary reprezentują szerokość, długość i głębokość. Widziałeś filmy 3D. Obrazy mają nie tylko szerokość i wysokość, ale także wydają się mieć głębokość. Rysunek pokazuje, jak wygląda trójwymiarowa tablica, jeśli ma głębokość czterech, sześciu rzędów i trzech kolumn. Zauważ, że trójwymiarowa tabela przypomina sześcian. Trudno jest wyobrazić sobie więcej niż trzy wymiary. Możesz jednak pomyśleć o każdym wymiarze po trzech jako o innym wystąpieniu. Innymi słowy, lista rekordów mrugnięcia jednego gracza może być przechowywana w tablicy. Uderzenie zespołu (jak pokazano w tabeli 25.1) jest dwuwymiarowe. Liga, złożona z kilku rekordów mrugnięcia drużyn, reprezentuje trójwymiarowy stół. Każda drużyna (głębokość tabeli) ma wiersze i kolumny danych o mrugnięciu. Jeśli jest więcej niż jedna liga, jest to inny wymiar (inny zestaw danych). C++ umożliwia przechowywanie kilku wymiarów, chociaż dane „rzeczywiste” rzadko wymagają więcej niż dwóch lub trzech.



Rezerwowanie tablic wielowymiarowych

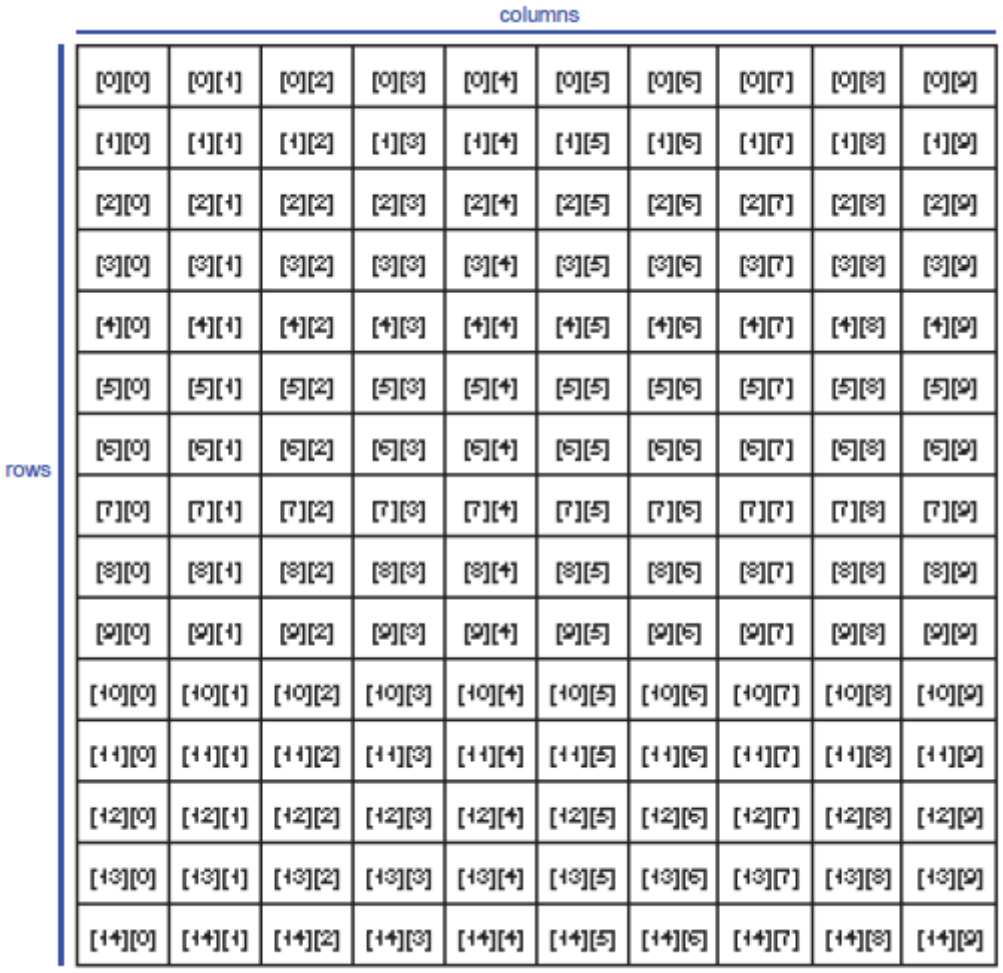
Kiedy rezerwujesz tablicę wielowymiarową, musisz poinformować C++, że tablica ma więcej niż jeden wymiar, umieszczając więcej niż jeden indeks dolny w nawiasach po nazwie tablicy. Musisz umieścić osobną liczbę w nawiasach dla każdego wymiaru w tabeli. Na przykład, aby zarezerwować dane zespołu

z tabeli powyżej, należy użyć następującej deklaracji tablic wielowymiarowych. Zadeklaruj tablicę liczb całkowitych zwaną zespołami z 15 rzędami i 10 kolumnami. zespoły wewnętrzne [15] [10]; // Zastrzega tabelę dwuwymiarową.

PRZESTROGA: W przeciwieństwie do innych języków programowania, C++ wymaga zawarcia każdego wymiaru w nawiasach. Nie rezerwuj wielowymiarowej pamięci macierzowej:

Int teams [15,10]; // Nieprawidłowa deklaracja tabeli.

Właściwe zarezerwowanie tabeli drużyn tworzy stół zawierający 150 elementów. Rysunek pokazuje, jak wygląda indeks dolny każdego elementu.



Gdybyś musiał śledzić trzy drużyny, każda z 15 graczami i 10 grami, trójwymiarowy stół zostałby utworzony w następujący sposób:

int teams [3] [15] [10]; // Zastrzega trójwymiarowy stół.

Tworząc tabelę dwuwymiarową, zawsze umieszczaj najpierw maksymalną liczbę wierszy, a drugą maksymalną liczbę kolumn. C++ zawsze używa 0 jako indeksu początkowego każdego wymiaru. Ostatnim elementem, prawym dolnym elementem tabeli drużyn, są teams[2][14][9].

Przykłady

1. Załóżmy, że chcesz śledzić rachunki za media za rok. Możesz przechowywać 12 miesięcy czterech narzędzi w dwuwymiarowej tabeli liczb zmiennoprzecinkowych, co pokazuje następująca deklaracja tablicowa:

float utilities[12][4]; // Reserves 48 elements.

Możesz obliczyć całkowitą liczbę elementów w tablicy wielowymiarowej, mnożąc indeksy dolne. Ponieważ 12 razy 4 to 48, w tej tablicy jest 48 elementów (12 wierszy, 4 kolumny). Każdy z tych elementów jest zmiennoprzecinkowym typem danych.

2. Jeśli śledziłeś narzędzia o wartości pięciu lat, musisz dodać dodatkowy wymiar. Pierwszy wymiar to lata, drugi to miesiące, a ostatni to poszczególne narzędzia. Oto jak zarezerwować miejsce:

Float utilities [5] [12] [4]; // Rezerwuje 240 elementów.

Mapowanie tablic do pamięci

C++ podchodzi do tablic wielowymiarowych nieco inaczej niż większość języków programowania. Korzystając z indeksów dolnych, nie musisz rozumieć wewnętrznej reprezentacji tablic wielowymiarowych. Jednak większość programistów C++ uważa, że głębsze zrozumienie tych tablic jest ważne, szczególnie kiedy programujesz zaawansowane aplikacje. Dwuwymiarowa tablica jest w rzeczywistości tablicą tablic. Programujesz tablice wielowymiarowe, jakby były tabelami z wierszami i kolumnami. Tablica dwuwymiarowa jest w rzeczywistości tablicą jednowymiarową, ale każdy z jej elementów nie jest liczbą całkowitą, zmiennoprzecinkową lub znakiem, ale inną tablicą. Wiedza, że tablica wielowymiarowa jest tablicą innych tablic, ma kluczowe znaczenie podczas przekazywania i odbierania takich tablic. C++ przekazuje wszystkie tablice, w tym tablice wielowymiarowe, według adresu. Załóżmy, że używasz tablicy liczb całkowitych o nazwie score, zarezerwowanej jako tabela 5 na 6. Możesz przekazać scores do funkcji o nazwie print_it () w następujący sposób:

print_it (scores); // Przekazuje tabelę do funkcji.

Funkcja print_it() musi identyfikować typ przekazywanego do niej parametru. Funkcja print_it() musi także rozpoznać, że parametr jest tablicą. Gdyby wyniki były jednowymiarowe, możesz odebrać jako

print_it (int scores []) // Działa tylko, jeśli scores

// jest jednowymiarowa.

lub

print_it (int scores [10]) // Zakładanie scores

// ma 10 elementów.

Gdyby scores były tabelą wielowymiarową, należałoby wyznaczyć każdą parę nawiasów i umieścić maksymalną liczbę indeksów dolnych w nawiasach, jak w

print_it (int scores [5] [6]) // Poinformuj print_it () o

// wymiarze tablicy.

lub

print_it (int scores [] [6]) // Poinformuj print_it () o

// wymiarz tablicy.

Zauważ, że nie musisz jawnie podawać maksymalnego indeksu dolnego dla pierwszego wymiaru, gdy otrzymujesz tablice wielowymiarowe, ale musisz wyznaczyć drugi. Gdyby wyniki były tabelą trójwymiarową o wymiarach 10 na 5 na 6, otrzymalibyśmy ją za pomocą print_it () as

print_it (int scores [] [5] [6]) // Tylko pierwszy wymiar

// jest opcjonalny.

lub

print_it (int scores [10] [5] [6]) // Poinformuj print_it () o

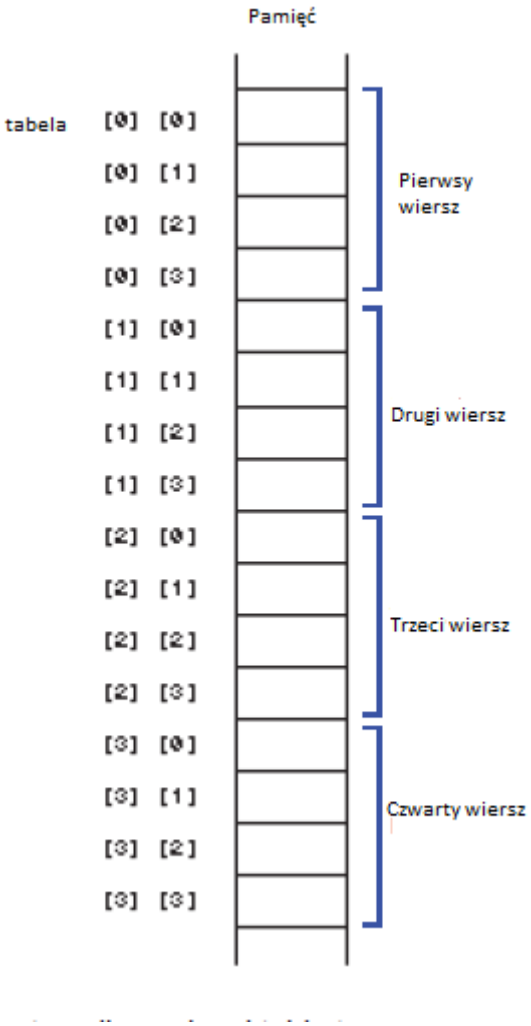
// wymiary tablicy.

Nie powinieneś zbytnio martwić się o fizyczny sposób przechowywania tabel. Chociaż tablica dwuwymiarowa jest w rzeczywistości tablicą tablic (i każda z tych tablic zawiera inną tablicę, jeśli jest to tablica trójwymiarowa), można użyć indeksów dolnych do programowania tablic wielowymiarowych tak, jakby były przechowywane w wierszach i kolumnach zamówienie. Tablice wielowymiarowe są przechowywane w kolejności wierszy. Załóżmy, że chcesz śledzić tabelę 3 na 4. Góra rysunku pokazuje jak ta tabela (i jej indeksy) są wizualizowane. Pomimo dwuwymiarowej organizacji tabel, pamięć nadal jest sekwencyjna. C++ musi mapować tablice wielowymiarowe na jednowymiarową pamięć i robi to w kolejności rzędów. Każdy wiersz zapełnia pamięć przed zapisaniem następnego wiersza. Rysunek pokazuje, w jaki sposób tabela 3 na 4 jest mapowana do pamięci. Cały pierwszy wiersz (tabela [0] [0] do tabeli [0] [3]) jest zapisany najpierw w pamięci przed drugim rzędem. Tabela jest w rzeczywistości tablicą tablic i, jak się dowiedziałeś w poprzednich rozdziałach, elementy tablicy są zawsze przechowywane sekwencyjnie w pamięci. Dlatego też pierwszy wiersz (tablica) całkowicie wypełnia pamięć przed drugim wierszem. Rysunek pokazuje, w jaki sposób tablice dwuwymiarowe mapowane są do pamięci.

Definiowanie tablic wielowymiarowych

C++ nie jest wybredny co do sposobu definiowania tablicy wielowymiarowej podczas inicjowania jej w czasie deklaracji. Jak w przypadku pojedynczego wymiaru

tablice, inicjujesz tablice wielowymiarowe za pomocą nawiasy klamrowe określające wymiary. Ponieważ tablica wielowymiarowa jest tablicą tablic, podczas inicjowania można zagnieżdżać nawiasy klamrowe.

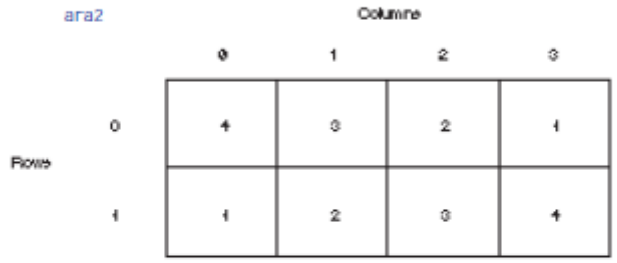


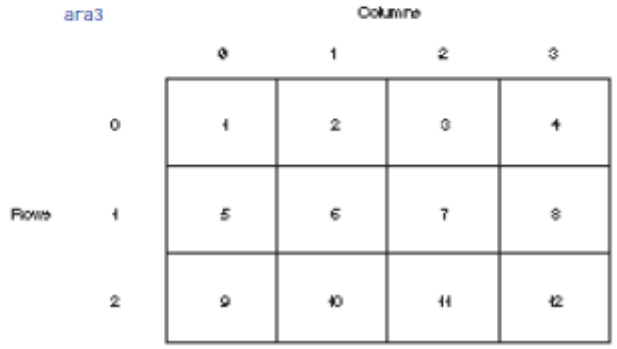
Poniższe trzy definicje tablic wypełniają trzy tablice ara1, ara2 i ara3, jak pokazano na rysunku 25.4:

int ara1 [5] = {8, 5, 3, 25, 41}; // Tablica jednowymiarowa.

int ara2 [2] [4] = {{4, 3, 2, 1}, {1, 2, 3, 4}};

int ara3 [3] [4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};





Zauważ, że tablice wielowymiarowe są przechowywane w kolejności wierszy. W ara3 pierwszy wiersz otrzymuje pierwsze cztery elementy definicji (1, 2, 3 i 4).

WSKAZÓWKA: Aby wielowymiarowa inicjalizacja tablicy była zgodna z indeksami tablicy, niektórzy programiści lubią pokazywać, w jaki sposób tablice są wypełniane. Ponieważ programy w C++ są dowolne, możesz zainicjować ara2 i ara3 jako

int ara2 [2] [4] = {{4, 3, 2, 1}, // Robi dokładnie to samo

{1, 2, 3, 4}}; // rzecz jak poprzednio.

int ara3 [3] [4] = {{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}; // Wizualnie więcej

// oczywiste.

Możesz zainicjalizować tablicę wielowymiarową, tak jakby była pojedynczo wymiarowa w C++. Jeśli to zrobisz, musisz śledzić kolejność wierszy. Na przykład następujące dwie definicje również rezerwują pamięć dla i inicjalizują ara2 i ara3:

int ara2 [2] [4] = {4, 3, 2, 1, 1, 2, 3, 4};

int ara3 [3] [4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};

Nie ma różnicy między inicjowaniem ara2 i ara3 z zagnieżdżonymi nawiasami klamrowymi lub bez nich. Zagnieżdżone nawiasy klamrowe wydają się pokazywać wymiary i sposób, w jaki C++ wypełnia je nieco lepiej, ale wybór korzystania z nawiasów klamrowych należy do Ciebie.

WSKAZÓWKA: Tablice wielowymiarowe (o ile nie są globalne) nie są inicjowane do określonych wartości, chyba że zostaną przypisane im wartości w czasie deklaracji lub w programie. Podobnie jak w przypadku tablic jednowymiarowych, jeśli zainicjujesz jeden lub więcej elementów, ale nie wszystkie, C++ wypełnia resztę zerami. Jeśli chcesz wypełnić całą tablicę wielowymiarową zerami, możesz to zrobić, wykonując następujące czynności:

Float sales [3] [4] [7] [2] = {0}; // Wypełnia całą sprzedaż zerami.

Ostatnią kwestią do rozważenia jest sposób, w jaki kompilator wyświetla tablice wielowymiarowe. Wiele osób programuje w C++ od lat, ale nigdy nie rozumie, w jaki sposób tabele są przechowywane wewnętrznie. Dopóki używasz indeksów dolnych, wewnętrzna reprezentacja tabeli nie powinna mieć znaczenia. Kiedy dowiesz się o zmiennych wskaźnikowych, możesz chcieć wiedzieć, w jaki sposób C++ przechowuje twoje tabele na wypadek, gdybyś chciał odwoływać się do nich za pomocą wskaźników (jak pokazano w kilku następnych rozdziałach). Rysunek pokazuje sposób, w jaki C++ przechowuje w pamięci tabelę 3 na 4. W przeciwieństwie do tablic jednowymiarowych każdy element jest przechowywany w sposób ciągły, ale zwróć uwagę, jak C++ wyświetla dane. Ponieważ tabela jest tablicą tablic, nazwa tablicy zawiera adres początku tablicy podstawowej. Każdy z tych elementów wskazuje na zawarte w nim tablice (dane w każdym rzędzie). Ten zakres przechowywania tabeli służy wyłącznie do celów informacyjnych w tym momencie. W miarę doskonalenia się w języku C++ i pisania potężniejszych programów, które manipulują pamięcią wewnętrzną, warto przejrzeć tę metodę przechowywania tabel.

Tabele i pętle

Jak pokazują poniższe przykłady, zagnieżdżone dla pętli są przydatne, gdy chcesz zapętlić każdy element tabeli wielowymiarowej.

Na przykład sekcja kodu,

for (row = 0; row<2; row ++)

{for (col = 0; col <3; col ++)

{cout << wiersz << „„ << col ”\ n”; }

}

twory następujące dane wyjściowe:

0 0

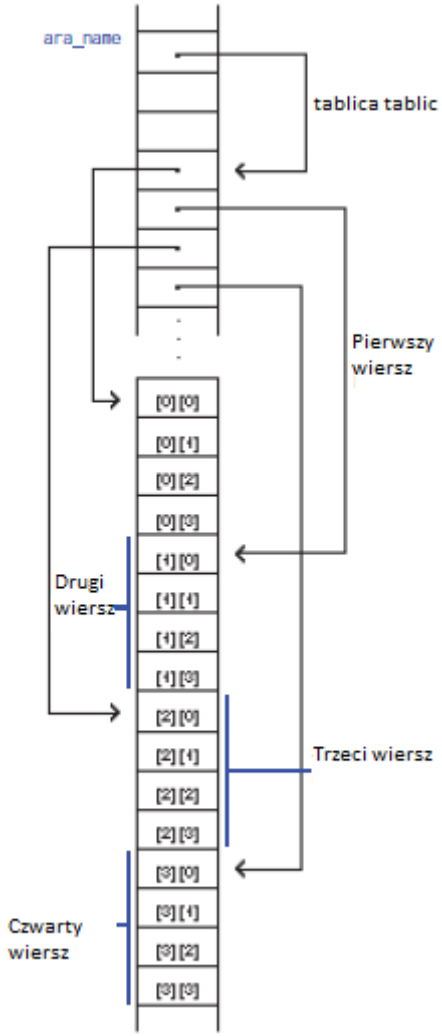
0 1

0 2

1 0

1 1

1 2



Te liczby są indeksami dolnymi, w kolejności rzędów, dla dwurzędowych przez trójkolumnową tabelę zwymiarowaną tabelą int [2] [3];

Zauważ, że dla pętli jest tyle, ile jest indeksów w tablicy (dwa). Pętla zewnętrzna reprezentuje pierwszy indeks dolny (wiersze), a pętla wewnętrzna reprezentuje drugi indeks dolny (kolumny). Zagnieżdżona pętla przechodzi przez każdy element tabeli. Możesz użyć cin, gets (), get i innych funkcji wejściowych do wypełnienia tabeli, a także możesz przypisać wartości do elementów podczas deklarowania tabeli. Częściej dane pochodzą z plików danych na dysku. Niezależnie od tego, która metoda przechowuje wartości w tablicach wielowymiarowych, zagnieżdżone dla pętli są doskonałymi instrukcjami sterującymi do przechodzenia przez indeksy dolne. Poniższe przykłady pokazują, w jaki sposób zagnieżdżone pętle działają z tablicami wielowymiarowymi.

Przykłady

1. Poniższe stwierdzenia zastrzegają wystarczającą liczbę elementów pamięci dla ocen stacji telewizyjnej (od A do D) na tydzień:

char ratings [7] [48];

Te oświadczenia zawierają wystarczającą liczbę elementów, aby pomieścić siedem dni (wierszy) ocen dla każdego 30-minutowego przedziału czasowego (48 z nich dziennie). Każdy element w tabeli jest

zawsze tego samego typu. W tym przypadku każdy element jest zmienną znakową. Niektóre są inicjowane za pomocą następujących instrukcji przypisania:

show[3] [12] = „B”; // Przechowuje B w 4 rzędzie, 13 kolumnie.

show [1] [5] = „A”; // Przechowuje C w 2. rzędzie, 6. kolumnie.

show [6] [20] = getch (); // Przechowuje literę, którą wpisuje użytkownik.

2. Firma komputerowa sprzedaje dyski w dwóch rozmiarach: 3 1/2 cala i 5 1/2 cala. Każdy dysk ma jedną z czterech pojemności: jednostronną podwójną gęstość, dwustronną podwójną gęstość, jednostronną wysoką gęstość i dwustronną wysoką gęstość. Inwentaryzacja dysku dobrze nadaje się do dwuwymiarowego stołu. Firma ustaliła, że dyski mają następujące ceny detaliczne:

Podwójna gęstość: wysoka gęstość

Pojedyncze - podwójne: Pojedyncze - podwójne

3 1/2 cala: 2,30 - 2,75: 3,20 -3,50

5 1/4 cala: 1,75 - 2,10: 2,60 2,95

Firma chce przechowywać cenę każdego dysku w tabeli dla łatwego dostępu. Poniższy program przechowuje ceny z instrukcjami przypisania.

```
// Nazwa pliku: C25DISK1.CPP

// Przypisuje ceny dysku do tabeli.

#include <iostream.h>

#include <iomanip.h>

void main()

{

float disks[2][4]; // Table of disk prices.

int row, col; // Subscript variables.

disks[0][0] = 2.39; // Row 1, column 1

disks[0][1] = 2.75; // Row 1, column 2

disks[0][2] = 3.29; // Row 1, column 3

disks[0][3] = 3.59; // Row 1, column 4

disks[1][0] = 1.75; // Row 2, column 1

disks[1][1] = 2.19; // Row 2, column 2

disks[1][2] = 2.69; // Row 2, column 3

disks[1][3] = 2.95; // Row 2, column 4

// Print the prices.

for (row=0; row<2; row++)
```

```
{ for (col=0; col<4; col++)

{ cout << "$" << setprecision(2) <<

disks[row][col] << "\n"; }

}

return;

}
```

Program wyświetla ceny w następujący sposób

$2.39

$2.75

$3.29

$3.59

$1.75

$2.19

$2.69

$2.95

Wyświetla je pojedynczo, bez opisowych tytułów. Mimo że dane wyjściowe nie są oznaczone, ilustrują sposób użycia instrukcji przypisania do zainicjowania tabeli oraz sposób zagnieżdżania pętli w celu wydrukowania elementów.

3. Poprzednia inwentaryzacja dysku byłaby wyświetlana lepiej, gdyby dane wyjściowe miały tytuły opisowe. Przed dodaniem tytułów pomocne jest sprawdzenie, jak wydrukować tabelę w rodzimym formacie wierszy i kolumn. Zazwyczaj do drukowania wierszy i kolumn używa się zagnieżdżonej pętli for, takiej jak w poprzednim przykładzie. Nie powinieneś jednak wypisywać znaku nowej linii przy każdym cout. Jeśli to zrobisz, zobaczysz jedną wartość w wierszu, jak w danych wyjściowych poprzedniego programu, która nie jest formatem wiersza i kolumny tabeli. Nie chcesz widzieć każdej ceny dysku w jednym wierszu, ale chcesz, aby każdy wiersz tabeli był drukowany w osobnym wierszu. Musisz wstawić cout << „\ n"; aby wysłać kursor do następnego wiersza za każdym razem, gdy zmienia się numer wiersza. Drukowanie nowego wiersza po każdym wierszu powoduje wydrukowanie tabeli w formacie wiersza i kolumny, ponieważ ten program pokazuje:

```
// Nazwa pliku: C25DISK2.CPP

// Przypisuje ceny dysku do tabeli

#include <iostream.h>

#include <iomanip.h>

void main()

{

float disks[2][4]; // Table of disk prices.
```

```
int row, col;
disks[0][0] = 2.39; // Row 1, column 1
disks[0][1] = 2.75; // Row 1, column 2
disks[0][2] = 3.29; // Row 1, column 3
disks[0][3] = 3.59; // Row 1, column 4
disks[1][0] = 1.75; // Row 2, column 1
disks[1][1] = 2.19; // Row 2, column 2
disks[1][2] = 2.69; // Row 2, column 3
disks[1][3] = 2.95; // Row 2, column 4
// Print the prices
for (row=0; row<2; row++)
{ for (col=0; col<4; col++)
{ cout << "$" << setprecision(2) <<
disks[row][col] << "\t";
}
cout << "\n"; // Prints a new line after each row.
}
return;
}
```

Oto dane wyjściowe cen dysków w natywnej kolejności tabel:

2,39 USD 2,75 USD 3,29 USD 3,59 USD

1,75 USD 2,19 USD 2,69 USD 2,95 USD

4. Aby dodać tytuły, po prostu wydrukuj rząd tytułów przed pierwszym rzędem wartości, a następnie wydrukuj nowy tytuł kolumny przed każdą kolumną, jak pokazano w następującym programie:

```
// Nazwa pliku: C25DISK3.CPP
// Przypisuje ceny dysku do tabeli
// i drukuje je w formacie tabeli z tytułami.
#include <iostream.h>
#include <iomanip.h>
void main()
{
```

```
float disks[2][4]; // Table of disk prices.
int row, col;
disks[0][0] = 2.39; // Row 1, column 1
disks[0][1] = 2.75; // Row 1, column 2
disks[0][2] = 3.29; // Row 1, column 3
disks[0][3] = 3.59; // Row 1, column 4
disks[1][0] = 1.75; // Row 2, column 1
disks[1][1] = 2.19; // Row 2, column 2
disks[1][2] = 2.69; // Row 2, column 3
disks[1][3] = 2.95; // Row 2, column 4
// Print the column titles.
cout << "\tSingle-sided\tDouble-sided\tSingle-sided\t" <<
"Double-sided\n";
cout << "\tDouble-density\tDouble-density\tHigh-density" <<
"\tHigh-density\n";
// Print the prices
for (row=0; row<2; row++)
{ if (row == 0)
{ cout << "3-1/2\"\t"; } // Need \" to
// print quotation.
else
{ cout << "5-1/4\"\t"; }
for (col=0; col<4; col++) // Print the current row.
{ cout << setprecision(2) << "$" << disks[row][col]
<< "\t\t";
}
cout << "\n"; // Print a newline after each row.
}
return;
}
```

Oto wynik tego programu:

Jednostronny: Dwustronny - Jednostronny: Dwustronny

Podwójna gęstość: Podwójna gęstość - Wysoka gęstość: Wysoka gęstość

3-1 / 2 ": 2,39 USD: 2,75 USD - 3,29 USD: 3,59 USD

5-1 / 4 ": 1,75 USD: 2,19 USD - 2,69 USD: 2,95 USD

Ćwiczenia

1. Napisz program, który przechowuje i drukuje liczby od 1 do 21 w tabeli 3 na 7. (Wskazówka: pamiętaj, że C++ zaczyna indeksy dolne na 0.)

2. Napisz program, który rezerwuje miejsce na dane sprzedaży na trzy lata dla pięciu sprzedawców. Użyj instrukcji przypisania, aby wypełnić tabelę danymi, a następnie wydrukuj ją, jedna wartość na wiersz.

3. Zamiast używać instrukcji przypisania, użyj funkcji cin, aby wypełnić dane sprzedawców z ćwiczenia 2.

4. Napisz program, który śledzi oceny z pięciu klas, z których każda ma 10 uczniów. Wprowadź dane za pomocą funkcji cin. Wydrukuj tabelę w rodzimym formacie wierszy i kolumn.

Podsumowanie

Teraz wiesz, jak tworzyć, inicjować i przetwarzać wielowymiarowo tablice. Chociaż nie wszystkie dane mieszczą się w zwartym formacie tabel, wiele się dzieje. Korzystanie z zagnieżdżonych pętli ułatwia przechodzenie przez wielowymiarową tablicę.

Wskaźniki

C++ ujawnia swoją prawdziwą moc poprzez zmienne wskaźnikowe. Zmienne wskaźnikowe (lub wskaźniki, jak się je zwykle nazywa) to zmienne zawierające adresy innych zmiennych. Wszystkie zmienne, które widziałeś do tej pory, zawierały wartości danych. Rozumiesz, że zmienne przechowują różne typy danych: znak, liczba całkowita, zmiennoprzecinkowa i tak dalej. Zmienne wskaźnikowe zawierają lokalizację zwykłych zmiennych danych; w rzeczywistości wskazują na dane, ponieważ przechowują adres danych. Podczas pierwszej nauki języka C++ uczniowie tego języka unikają wskaźników, myśląc, że wskaźniki będą trudne. Wskaźniki nie muszą być trudne. W rzeczywistości po dłuższej pracy z nimi okaże się, że są one łatwiejsze w użyciu niż tablice (i znacznie bardziej elastyczne).

W tej części przedstawiono następujące pojęcia:

♦ Wskaźniki

♦ Wskaźniki różnych typów danych

♦ „Adres" (&) operatora

♦ Operator dereferencji (*)

♦ Tablice wskaźników

Wskaźniki oferują wysoce wydajny sposób uzyskiwania dostępu i zmiany danych. Ponieważ wskaźniki zawierają rzeczywisty adres twoich danych, twój kompilator ma mniej pracy do znalezienia tych danych w pamięci. Wskaźniki nie muszą łączyć danych z konkretnymi nazwami zmiennych. Wskaźnik może wskazywać na nienazwaną wartość danych. Dzięki wskaźnikom zyskujesz „inny widok" swoich danych.

Wprowadzenie do zmiennych wskaźnikowych

Wskaźniki są zmiennymi. Przestrzegają wszystkich normalnych zasad nazewnictwa zwykłych zmiennych niepunktowych. Podobnie jak w przypadku zmiennych zwykłych, przed użyciem należy zadeklarować zmienne wskaźnikowe. Istnieje typ wskaźnika dla każdego typu danych w C++; istnieją wskaźniki całkowite, wskaźniki znakowe, zmiennoprzecinkowe i tak dalej. Możesz zadeklarować wskaźniki globalne lub wskaźniki lokalne, w zależności od tego, gdzie je zadeklarujesz. Jedyną różnicą między zmiennymi wskaźnikowymi a zmiennymi regularnymi są przechowywane przez nich dane. Wskaźniki nie zawierają danych w zwykłym znaczeniu tego słowa. Wskaźniki zawierają adresy danych. Jeśli potrzebujesz szybkiego przeglądu adresów i pamięci. W C++ są dwa operatory wskaźnika:

& Operator „adresu"

* Operator dereferencji

Nie pozwól, aby ci operatorzy cię rzucili; mogłeś je wcześniej zobaczyć! & Jest bitowym operatorem AND (z rozdziału 11, „Dodatkowe operatory C++"), a * oznacza oczywiście mnożenie. Są to tak zwane przeciążone operatory. Wykonują więcej niż jedną funkcję, w zależności od tego, jak używasz ich w swoich programach. C++ nie myli * mnożenia, gdy używasz go jako operatora dereferencji ze wskaźnikami. Za każdym razem, gdy zobaczysz „i używane ze wskaźnikami", pomyśl o słowach „adres". Operator & zawsze wytwarza adres pamięci tego, co poprzedza. Operator *, w połączeniu ze wskaźnikami, deklaruje wskaźnik lub odchyla jego wartość. Następna sekcja wyjaśnia każdego z tych operatorów.

Deklarowanie wskaźników

Ponieważ musisz zadeklarować wszystkie wskaźniki przed ich użyciem, najlepszym sposobem na rozpoczęcie nauki o wskaźnikach jest zrozumienie, jak je deklarować i definiować. W rzeczywistości deklarowanie wskaźników jest prawie tak proste, jak deklarowanie zmiennych regularnych. W końcu wskaźniki są zmiennymi. Jeśli musisz zadeklarować zmienną, która utrzymuje twój wiek, możesz to zrobić za pomocą następującej deklaracji zmiennej:

int age = 30; // Zadeklaruj zmienną, aby utrzymać mój wiek.

Deklarowanie wieku w ten sposób robi kilka rzeczy. Umożliwia C++ identyfikację zmiennej o nazwie wiek i zarezerwowanie pamięci dla tej zmiennej. Użycie tego formatu pozwala również C++ rozpoznać, że będziesz przechowywać tylko liczby całkowite w wieku, a nie dane zmiennoprzecinkowe lub podwójne zmiennoprzecinkowe. Deklaracja wymaga również, aby C++ przechowywał wartość 30 lat po tym, jak zarezerwuje miejsce dla wieku. Gdzie C++ zapisał się w pamięci? Jako programista nie powinieneś się przejmować tym, gdzie starzeją się sklepy C++. Nie musisz znać adresu zmiennej, ponieważ nigdy nie będziesz odnosić się do wieku przez jego adres. Jeśli chcesz obliczyć wiek lub wydrukować wiek, nazywasz go po nazwie, wiek.

WSKAZÓWKA: Nadaj sensowi nazwy zmiennych wskaźnika. Nazwa file_ptr ma większą sens niż x13 dla zmiennej wskazującej plik, chociaż dowolna nazwa jest dozwolona.

Załóżmy, że chcesz zadeklarować zmienną wskaźnika. Ta zmienna wskaźnikowa nie zachowa twojego wieku, ale będzie wskazywać na wiek, zmienną, która utrzymuje twój wiek. (Dlaczego chcesz to zrobić, wyjaśniono w tym i kilku następnych rozdziałach.) P_age może być dobrą nazwą dla zmiennej wskaźnika. Rysunek pokazuje, co chcesz zrobić. Na rysunku przyjęto age przechowywany w C++ pod adresem 350 606. Jednak Twój kompilator C++ arbitralnie określa adres age, więc może to być cokolwiek.

Nazwa p_age nie ma nic wspólnego ze wskaźnikami, poza tym, że jest to nazwa, którą wymyśliłeś, aby wskaźnik się zestarzał. Tak jak możesz nazwać zmienne cokolwiek (o ile nazwa jest zgodna z prawnymi regułami nazewnictwa zmiennych), p_age może równie łatwo zostać nazwany house, x43344, space_trek lub cokolwiek innego, co chcesz to nazwać. To wzmacnia pogląd, że wskaźnik jest tylko zmienną zarezerwowaną w twoim programie. Twórz znaczące nazwy zmiennych, nawet dla zmiennych wskaźnikowych. p_age to dobra nazwa dla zmiennej, która wskazuje na wiek (tak jak ptr_age i ptr_to_age). Aby zadeklarować zmienną wskaźnika p_age, należy zaprogramować następujące elementy:

int * p_age; // Deklaruje wskaźnik liczby całkowitej.

Podobnie do deklaracji wieku, ta deklaracja zastrzega zmienną o nazwie p_age. Jednak zmienna p_age nie jest normalną zmienną całkowitą. Z powodu operatora dereferencji * C++ wie, że ma to być

zmienna wskaźnikowa. Niektórzy programiści C++ wolą deklarować taką zmienną bez spacji po *, w następujący sposób:

int * p_age; // Deklaruje wskaźnik liczby całkowitej.

Obie metody są w porządku, ale musisz pamiętać, że * nie jest częścią nazwy. Kiedy później użyjesz p_age, nie będziesz poprzedzać nazwy znakiem *, chyba że w tej chwili będziesz ją odsuwał (jak pokazują późniejsze przykłady).

WSKAZÓWKA: Ilekroć operator dereferencji * pojawia się w definicji zmiennej, deklarowana zmienna jest zawsze zmienną wskaźnikową.

Rozważ deklarację dla p_age, jeśli nie ma tam gwiazdki: C++ pomyślałby, że deklarujesz zwykłą zmienną całkowitą. * Jest ważne, ponieważ każe C++ interpretować p_age jako zmienną wskaźnikową, a nie jako normalną zmienną danych.

Przypisywanie wartości do wskaźników

p_age jest wskaźnikiem liczb całkowitych. To jest bardzo ważne. p_age może wskazywać tylko na wartości całkowite, nigdy na zmiennoprzecinkowe, podwójne zmiennoprzecinkowe, a nawet zmienne znakowe. Jeśli potrzebujesz wskazać zmienną zmiennoprzecinkową, możesz to zrobić ze wskaźnikiem zadeklarowanym jako

float *point; // Declares a floating-point pointer.

Podobnie jak w przypadku każdej zmiennej automatycznej, C++ nie inicjuje wskaźników podczas ich deklarowania. Jeśli zadeklarowałeś p_age zgodnie z wcześniejszym opisem i chciałeś, aby p_age wskazywał na wiek, musisz jawnie przypisać p_age do adresu wieku. Następująca instrukcja to robi:

p_age = i wiek; // Przypisz adres wieku do p_age.

Jaka jest teraz wartość w p_age? Nie wiesz dokładnie, ale wiesz, że to adres wieku, gdziekolwiek to jest. Zamiast przypisywać adres wieku p_age za pomocą operatora przypisania, możesz zadeklarować i inicjować wskaźniki w tym samym czasie. Te linie deklarują i inicjują age i p_age:

int age=30; // Declares a regular integer

// variable, putting 30 in it.

int *p_age=&age; // Declares an integer pointer,

// initializing it with the address

// of p_age.

Te dwa wiersze tworzą zmienne opisane na rysunku 26.1. Jeśli chcesz wydrukować wartość wieku, możesz to zrobić za pomocą następującego polecenia:

cout << age; // Prints the value of age.

Możesz także wydrukować wartość wieku w następujący sposób:

cout << * p_age; // Dereferencje p_age.

Operator dereferencji tworzy wartość, która mówi wskaźnikowi, gdzie ma wskazywać. Bez znaku * ostatni kod wydrukowałby adres (adres wieku). Z *, cout wypisuje wartość pod tym adresem. Możesz przypisać inną wartość do wieku za pomocą następującego oświadczenia:

age=41; // Assigns a new value to age.

Możesz również przypisać wartość do wieku w następujący sposób:

* p_age = 41;

Ta deklaracja przypisuje 41 do wartości, na którą wskazuje p_age.

WSKAZÓWKA: * pojawia się przed zmienną wskaźnikową tylko w dwóch miejscach - kiedy deklarujesz zmienną wskaźnikową i gdy odznaczasz zmienną wskaźnikową (aby znaleźć dane, na które wskazuje).

Wskaźniki i parametry

Teraz, gdy rozumiesz wskaźnik * i operatory, możesz wreszcie zobaczyć, dlaczego wymagania scanf () nie były tak surowe, jak się początkowo wydawały. Przekazując zmienną zwykłą do scanf (), trzeba było poprzedzić zmienną operatorem &. Na przykład poniższy skanf () pobiera od użytkownika trzy wartości całkowite:

scanf („% d% d% d”, & num1, & num2 i & num3);

Ta funkcja scanf () nie przekazuje trzech zmiennych, ale przekazuje adresy trzech zmiennych. Ponieważ scanf() zna dokładnie zapisy tych parametrów w pamięci (ponieważ ich adresy zostały przekazane), trafia na te adresy i umieszcza wartości wejściowe klawiatury na tych adresach.

Jest to jedyny sposób, w jaki scanf() może działać. Jeśli je zdałeś zmienne według kopii, bez wstawiania operatora „adres” (&) przed nimi scanf() pobiera dane z klawiatury i wypełnia kopię zmienne, ale nie rzeczywiste zmienne num1, num2 i num3. Kiedy scanf() zwrócił kontrolę do twojego programu, nie miałeś wartości wejściowych. Oczywiście operator cin nie ma

Wymaga ampersand (&) i jest łatwiejszy w użyciu dla większości programów w C++. Możesz przypomnieć sobie z rozdziału 18, „Przekazywanie wartości”, że możesz zastąpić normalne domyślne ustawienie C++ przekazywania przez kopiowanie (lub „przez wartość").

Aby przekazać adres, należy otrzymać zmienną poprzedzoną znakiem & w funkcji odbierającej. Następująca funkcja odbiera próby według adresu:

pr_it (int & try); // Odbierz liczby całkowite w pr_it () przez

// adres (pr normalnie by to otrzymał

// próbuje przez kopię).

Teraz, gdy rozumiesz operatory & i *, możesz całkowicie zrozumieć przekazywanie parametrów nonarray przez adres do funkcji. (Tablice domyślnie przekazują według adresu bez konieczności używania &.)

Przykłady

1. Następująca sekcja kodu deklaruje trzy zmienne regularne trzech różnych typów danych i trzy odpowiadające zmienne wskaźnikowe:

char initial = „Q”; // Deklaruje trzy zmienne regularne

int num = 40; // trzech różnych typów.

float sales = 2321,59;

char * p_initial = & initial; // Deklaruje trzy wskaźniki.

int * ptr_num = & num; // Nazwy wskaźników i odstępy

float * sales_add = & sales; // po * nie są krytyczne.

2. Podobnie jak zwykłe zmienne, możesz inicjalizować wskaźniki za pomocą instrukcji przypisania. Nie musisz ich inicjować, kiedy je deklarujesz. Następne kilka wierszy kodu jest równoważnych kodowi z przykładu 1:

char initial; // Deklaruje trzy zmienne regularne

int num; // trzech różnych typów.

float sales;

char * p_initial; // Deklaruje trzy wskaźniki, ale robi

int * ptr_num; // jeszcze ich nie inicjuje.

float * sales_add;

initial = = „Q”; // Inicjuje zmienne regularne

num = 40; // z wartościami.

sales = 2321,59;

p_initial = & initial; // Inicjuje wskaźniki za pomocą

ptr_num = & num; // ich adresy

sales_add = & sales; // odpowiednie zmienne.

Zauważ, że nie umieszczasz operatora * przed nazwami zmiennych wskaźnika podczas przypisywania im wartości. Poprzedziłbyś zmienną wskaźnikową znakiem * tylko wtedy, gdy byłaby to dereferencja.

UWAGA: W tym przykładzie zmiennym wskaźnikowym można było przypisać adresy zmiennych regularnych, zanim zmienne regularne otrzymały wartości. Nie byłoby różnicy w działaniu. Wskaźniki mają przypisane adresy zmiennych zmiennych, bez względu na to, jakie są dane w zmiennych zmiennych. Zachowaj typ danych każdego wskaźnika zgodny z odpowiednią zmienną. Nie przypisuj zmiennej zmiennoprzecinkowej do adresu liczby całkowitej. Na przykład nie można wykonać następującej instrukcji przypisania:

p_initial = & sales; // Nieprawidłowe przypisanie wskaźnika.

ponieważ p_initial może wskazywać tylko dane znakowe, a nie dane zmiennoprzecinkowe.

3. Poniższy program jest przykładem, który powinieneś studiować dokładnie. Pokazuje więcej na temat wskaźników i operatorów wskaźników,

i i *, niż może zrobić kilka stron tekstu.

// Nazwa pliku: C26POINT.CPP

// Demonstruje użycie deklaracji wskaźnika

// i operatorzy.

```
#include <iostream.h>

void main()

{

int num=123; // A regular integer variable.

int *p_num; // Declares an integer pointer.

cout << “num is “ << num << “\n”; // Prints value of num.

cout << “The address of num is “ << &num << “\n”;

// Prints num’s location.

p_num = &num; // Puts address of num in p_num,

// in effect making p_num point

// to num.

// No * in front of p_num.

cout << “*p_num is “ << *p_num << “\n”; // Prints value

// of num.

cout << “p_num is “ << p_num << “\n”; // Prints location

// of num.

return;

}
```

Oto wynik tego programu:

```
num is 123

Adres num to 0x8fbd0ffe

p_num is 0x8fbd0ffe
```

Jeśli uruchomisz ten program, prawdopodobnie uzyskasz różne wyniki dla wartości p_num, ponieważ twój kompilator umieści wartość num w innym miejscu, w zależności od konfiguracji pamięci. Wartość p_num jest wyświetlana w systemie szesnastkowym, ponieważ jest to adres pamięci. Rzeczywisty adres nie ma jednak znaczenia. Ponieważ wskaźnik p_num zawsze zawiera adres num, a ponieważ można wyrejestrować p_num, aby uzyskać wartość num, rzeczywisty adres nie jest krytyczny.

4. Poniższy program zawiera funkcję, która zamienia wartości dwóch dowolnych liczb całkowitych przekazanych do niego. Możesz pamiętać, że funkcja może zwrócić tylko jedną wartość. Dlatego wcześniej nie można było napisać funkcji, która zmieniła dwie różne wartości i zwróciła obie wartości do funkcji wywołującej. Aby zamienić dwie zmienne (odwracając ich wartości do sortowania, jak widzieliśmy w rozdziale 24, „Przetwarzanie tablic”), musisz mieć możliwość przekazania obu zmiennych według adresu. Następnie, gdy funkcja odwraca zmienne, zmienne funkcji wywołującej również są zamieniane. Zwróć uwagę na użycie przez operatorów operatorów dereferencyjnych przed każdym wystąpieniem wartości num1 i num2. Nie ma znaczenia, pod którym adresem są przechowywane

numery num1 i num2, ale należy upewnić się, że dereferencje zostały usunięte z adresów przekazanych do funkcji. Pamiętaj, aby otrzymywać argumenty z prefiksem i funkcjami, które odbierają adres, tak jak tutaj.

Zidentyfikuj program i dołącz plik nagłówkowy we / wy. Ten program zamienia dwie liczby całkowite, więc zainicjuj dwie zmienne całkowite w main (). Przekaż zmienne do funkcji zamiany, zwanej swap_them, a następnie zmień ich wartości. Wydrukuj wyniki zamiany w main().

```
// Nazwa pliku: C26SWAP.CPP

// Program zawierający funkcję zamiany

// przekazano do niego dwie dowolne liczby całkowite

#include <iostream.h>

void swap_them(int &num1, int &num2);

void main()

{

int i=10, j=20;

cout << "\n\nBefore swap, i is " << i <<

" and j is " << j << "\n\n";

swap_them(i, j);

cout << "\n\nAfter swap, i is " << i <<

" and j is " << j << "\n\n";

return;

}

void swap_them(int &num1, int &num2)

{

int temp; // Variable that holds

// in-between swapped value.

temp = num1; // The calling function's variables

num1 = num2; // (and not copies of them) are

num2 = temp; // changed in this function.

return;

}
```

Tablice wskaźników

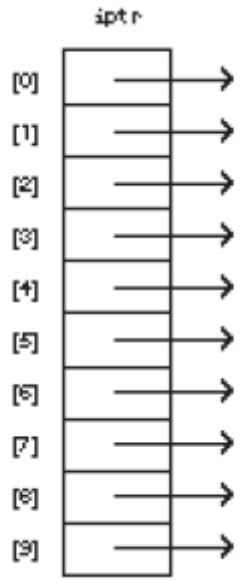
Jeśli musisz zarezerwować wiele wskaźników dla wielu różnych wartości, możesz zadeklarować tablicę wskaźników. Wiesz, że możesz zarezerwować tablicę znaków, liczb całkowitych, długich liczb

całkowitych i wartości zmiennoprzecinkowych, a także tablicę każdego innego dostępnego typu danych. Możesz także zarezerwować tablicę wskaźników, przy czym każdy wskaźnik jest wskaźnikiem określonego typu danych. Następujące rezerwuje tablicę 10 zmiennych wskaźników całkowitych:

int * iptr [10]; // Zastrzega tablicę 10 wskaźników całkowitych

Rysunek pokazuje, jak C++ wyświetla tę tablicę. Każdy element ma adres (po przypisaniu wartości), który wskazuje na inne wartości w pamięci. Każda wskazana wartość musi być liczbą całkowitą. Możesz przypisać element z adresu iptr do adresu, tak jak w przypadku zmiennych wskaźnika innych niż tablica. Możesz sprawić, aby iptr [4] wskazywał adres zmiennej całkowitej o nazwie wiek, przypisując ją w następujący sposób:

iptr [4] = i age; // Ustaw iptr [4] na adres wieku.



Następujące rezerwuje tablicę 20 zmiennych znaków wskaźnika:

char * cpoint [20]; // Tablica 20 wskaźników postaci.

Znów gwiazdka nie jest częścią nazwy tablicy. Gwiazdka informuje C++, że jest to tablica wskaźników liczb całkowitych, a nie tylko tablica liczb całkowitych. Niektórzy początkujący studenci C++ są zdezorientowani, gdy widzą taką deklarację. Wskaźniki to jedno, ale rezerwowanie miejsca na tablice wskaźników ma tendencję do zamaczania nowicjuszy. Jednak rezerwowanie miejsca na tablice wskaźników jest łatwe do zrozumienia. Usuń gwiazdkę z poprzedniej deklaracji w następujący sposób:

char cpoint [20];

a ty właśnie zarezerwowałeś prosty zestaw 20 znaków. Dodanie gwiazdki mówi C++, aby poszła o krok dalej: zamiast tablicy zmiennych znakowych potrzebujesz tablicy zmiennych wskazujących znaki. Zamiast tego, aby każdy element był zmienną znakową, każdy element ma adres wskazujący znaki. Zastrzeganie tablic wskaźników będzie o wiele bardziej znaczące po zapoznaniu się ze strukturami w kilku kolejnych rozdziałach. Podobnie jak w przypadku zwykłych zmiennych niepunktujących, tablica znacznie ułatwia przetwarzanie kilku zmiennych wskaźnikowych. Za pomocą indeksu dolnego można odwoływać się do każdej zmiennej (elementu) bez konieczności używania innej nazwy zmiennej dla każdej wartości.

Podsumowanie

Deklarowanie i używanie wskaźników może w tym momencie wydawać się kłopotliwe. Po co przypisywać * p_num wartość, gdy łatwiej (i jaśniej) przypisać wartość bezpośrednio do num? Jeśli zadajesz sobie to pytanie, prawdopodobnie rozumiesz wszystko, co powinieneś i jesteś gotowy, aby zacząć uczyć się prawdziwej mocy wskaźników: łączenie wskaźników i przetwarzanie tablic.

Wskaźniki i tablice

Tablice i wskaźniki są ściśle powiązane w języku programowania C++. Możesz adresować tablice tak, jakby były wskaźnikami, i adresować wskaźniki tak, jakby były tablicami. Możliwość przechowywania wskaźników i tablic oraz uzyskiwania do nich dostępu daje możliwość przechowywania ciągów danych w elementach tablicy. Bez wskaźników nie można przechowywać ciągów danych w tablicach, ponieważ w C++ nie ma podstawowego typu danych ciągu (brak zmiennych ciągu, tylko literały ciągu). Tu przedstawiono następujące pojęcia:

♦ Tablice nazw i wskaźników

♦ Wskaźniki znakow

♦ Arytmetyka wskaźnika

♦ Szeregowane tablice danych ciągu

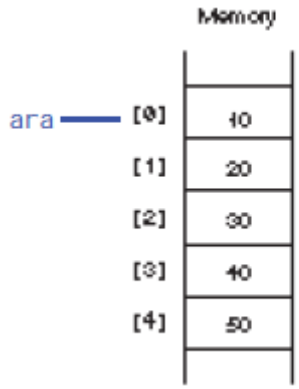
Oisano pojęcia, które będą używane przez większość twojego przyszłego programowania w C++. Manipulowanie wskaźnikami jest ważne dla języka programowania C++.

Nazwy tablic jako wskaźniki

Nazwa tablicy to tylko wskaźnik, nic więcej. Aby to udowodnić, załóżmy, że masz następującą deklarację tablicową:

int ara [5] = {10, 20, 30, 40, 50};

Jeśli wyświetlisz arę [0], zobaczysz 10. Ponieważ teraz w pełni rozumiesz tablice, jest to wartość, której możesz się spodziewać. Ale co by było, gdybyś wydrukował *arę? Czy *ara wydrukuje coś? Jeśli tak to co? Jeśli pomyślałeś, że zostanie wydrukowany komunikat o błędzie, ponieważ ara nie jest wskaźnikiem, ale tablicą, pomylisz się. Nazwa tablicy to wskaźnik. Jeśli wyświetlisz *ara, zobaczysz również 10. Przypomnij sobie, jak tablice są przechowywane w pamięci. Rysunek pokazuje, w jaki sposób ara byłaby mapowana w pamięci. Nazwa tablicy, ara, jest niczym więcej niż wskaźnikiem wskazującym pierwszy element tablicy. Dlatego jeśli wyrejestrujesz ten wskaźnik, wyrejestrujesz wartość zapisaną w pierwszym elemencie tablicy, czyli 10. Dereferencja ara jest dokładnie tym samym, co odwołanie do ara [0], ponieważ oba generują tę samą wartość.



Teraz widzisz, że możesz odwoływać się do tablicy za pomocą indeksów dolnych lub za pomocą dereferencji wskaźnika. Czy możesz użyć notacji wskaźnikowej do wydrukowania trzeciego elementu ara? Tak i masz już do tego narzędzia. Następujący cout wypisuje ara [2] (trzeci element ara) bez użycia indeksu dolnego:

cout << * (ara + 2); // Wyświetla ara [2].

Wyrażenie * (ara + 2) wcale nie jest niejasne, jeśli pamiętasz, że nazwa tablicy jest tylko wskaźnikiem, który zawsze wskazuje na pierwszy element tablicy. * (ara + 2) pobiera adres zapisany w ara, dodaje dwa do adresu i odznacza tę lokalizację. Prawda jest następująca:

ara + 0 punktów do ara [0]

ara + 1 punktów do ara [1]

ara + 2 punkty do ara [2]

ara + 3 punkty do ara [3]

ara + 4 punkty do ara [4]

Dlatego do drukowania, przechowywania lub obliczania za pomocą elementu tablicowego można użyć zapisu w indeksie dolnym lub wskaźnika. Ponieważ nazwa tablicy zawiera adres pierwszego elementu tablicy, musisz wyrejestrować wskaźnik, aby uzyskać wartość elementu.

Lokalizacje wewnętrzne

C++ zna wewnętrzne wymagania dotyczące rozmiaru danych dotyczące znaków, liczb całkowitych, liczb zmiennoprzecinkowych i innych typów danych na komputerze. Dlatego, ponieważ ara jest tablicą liczb całkowitych i ponieważ każdy element w tablicy liczb całkowitych zużywa dwa do czterech bajtów pamięci, w zależności od komputera, C++ dodaje dwa lub cztery bajty do adresu, jeśli odwołujesz się do tablic, jak pokazano. Jeśli napiszesz * (ara + 3) w odniesieniu do ara [3], C++ doda sześć lub dwanaście bajtów do adresu ara, aby uzyskać trzeci element. C++ nie dodaje rzeczywistych trzech. Nie musisz się tym martwić, ponieważ C++ obsługuje te elementy wewnętrzne. Kiedy piszesz *(ara + 3), faktycznie żądasz, aby C++ dodał trzy liczby całkowite adresu na adres ara. Gdyby ara była tablicą zmiennoprzecinkową, C++ dodałby do niej trzy adresy zmiennoprzecinkowe.

Zalety wskaźnika

Chociaż tablice są w rzeczywistości wskaźnikami w przebraniu, są to specjalne typy wskaźników. Nazwa tablicy jest stałą wskaźnika, a nie zmienną wskaźnika. Nie można zmienić wartości nazwy tablicy, ponieważ nie można zmienić stałych. To wyjaśnia, dlaczego nie można przypisać tablicy nowych wartości podczas wykonywania programu. Na przykład, nawet jeśli cname jest tablicą znaków, następujące elementy nie są poprawne w C++:

cname = „Christine Chambers”; // Nieprawidłowe przypisanie tablicy.

Nazwy tablicy cname nie można zmienić, ponieważ jest to stała. Nie próbowałbyś wykonać następujących czynności

5 = 4 + 8 * 21; // Nieprawidłowe przypisanie

ponieważ nie można zmienić stałej 5 na żadną inną wartość. C++ wie, że nie możesz przypisać niczego do 5, a C++ drukuje komunikat o błędzie, jeśli spróbujesz zmienić 5. C++ wie również, że nazwa tablicy jest stała i nie możesz zmienić tablicy na inną wartość. (Możesz przypisać wartości do tablicy tylko w czasie deklaracji, jeden element na raz podczas wykonywania lub za pomocą funkcji takich jak strcpy ().) Daje to najważniejszy powód, aby nauczyć się wskaźników: wskaźników (z wyjątkiem tablic oznaczonych jako wskaźniki) są zmiennymi. Możesz zmienić zmienną wskaźnika, a dzięki temu przetwarzanie praktycznie dowolnych danych, w tym tablic, jest o wiele bardziej wydajne i elastyczne.

Przykłady

1. Zmieniając wskaźniki, sprawiasz, że wskazują one różne wartości w pamięci. Poniższy program pokazuje, jak zmienić wskaźniki. Program najpierw definiuje dwie wartości zmiennoprzecinkowe. Wskaźnik zmiennoprzecinkowy wskazuje pierwszą zmienną, v1, i jest używany w kreacji. Wskaźnik jest następnie zmieniany, aby wskazywał na drugą zmienną zmiennoprzecinkową, v2.

```
// Nazwa pliku: C27PTRCH.CPP

// Zmienia wartość zmiennej wskaźnikowej.

#include <iostream.h>

#include <iomanip.h>

void main()

{

float v1=676.54; // Defines two

float v2=900.18; // floating-point variables.

float * p_v; / Defines a floating-point pointer.

p_v = &v1; // Makes pointer point to v1.

cout << “The first value is “ << setprecision(2) <<

*p_v << “\n”; // Prints 676.54.

p_v = &v2; // Changes the pointer so it

// points to v2.

cout << “The second value is “ << setprecision(2) <<

*p_v << “\n”; // Prints 900.18.

return;

}
```

Ponieważ mogą zmieniać wskaźniki, większość programistów C++ używa wskaźników zamiast tablic. Ponieważ tablice są łatwe do zadeklarowania, programiści C++ czasami deklarują tablice, a następnie używają wskaźników do odwołania się do tych tablic. Jeśli dane tablicy się zmienią, wskaźnik pomaga to zmienić.

2. Możesz użyć notacji wskaźnikowej i wskaźników referencyjnych jako tablic z notacją tablicową. Poniższy program deklaruje tablicę liczb całkowitych i wskaźnik liczb całkowitych wskazujący początek tablicy. Wartości tablic i wskaźników są drukowane przy użyciu notacji w indeksie dolnym. Następnie program używa notacji tablicowej do wydrukowania wartości tablic i wskaźników. Przestudiuj dokładnie ten program. Widzisz wewnętrzne działanie tablic i notacji wskaźnikowej.

```
// Nazwa pliku: C27ARPTR.CPP

// Odwołuje się do tablic takich jak wskaźniki i

// wskaźniki jak tablice.
```

```
#include <iostream.h>
void main()
{
int ctr;
int iara[5] = {10, 20, 30, 40, 50};
int *iptr;
iptr = iara; // Make iptr point to array's first
// element. This would work also:
// iptr = &iara[0];
cout << "Using array subscripts:\n";
cout << "iara\tiptr\n";
for (ctr=0; ctr<5; ctr++)
{ cout << iara[ctr] << "\t" << iptr[ctr] << "\n"; }
cout << "\nUsing pointer notation:\n";
cout << "iara\tiptr\n";
for (ctr=0; ctr<5; ctr++)
{ cout << *(iara+ctr) << "\t" << *(iptr+ctr) << "\n"; }
return;
}
```

Oto wynik programu:

```
Using array subscripts:
iara iptr
10 10
20 20
30 30
40 40
50 50
Using pointer notation:
iara iptr
10 10
20 20
```

30 30

40 40

50 50

Używanie wskaźników znaków

Możliwość zmiany wskaźników najlepiej widać podczas pracy z ciągami znaków w pamięci. Możesz przechowywać ciągi znaków w tablicach znaków lub wskazywać je za pomocą wskaźników znaków. Rozważ następujące dwie definicje ciągów:

char cara [] = „C++ is fun”; // Tablica zawierająca ciąg

char * cptr = „C++ By Example”; // Wskaźnik do łańcucha

Rysunek pokazuje, jak C++ przechowuje te dwa ciągi w pamięci. C++ przechowuje oba w zasadzie w ten sam sposób. Znasz definicję tablicy. Podczas przypisywania łańcucha do wskaźnika znaku, C++ znajduje wystarczającą ilość wolnej pamięci do przechowywania łańcucha i przypisania adresu pierwszego znaku do wskaźnika. Poprzednie dwie instrukcje definicji łańcucha wykonują prawie dokładnie to samo; jedyna różnica między nimi jest to, że dwa wskaźniki można łatwo wymienić (nazwa tablicy i wskaźniki znaków). Ponieważ cout drukuje ciągi znaków zaczynające się od nazwy tablicy lub wskaźnika, aż do osiągnięcia zerowego zera, możesz wydrukować każdy z tych ciągów za pomocą następujących instrukcji cout:

cout << „String 1:„ << cara << „\ n”;

cout << „String 2:„ << cptr << „\ n”;

Drukujesz ciągi w tablicach i ciągi wskazane w ten sam sposób. Możesz się zastanawiać, jaką przewagę ma jedna metoda przechowywania łańcuchów w porównaniu do drugiej. Pozornie niewielka różnica między tymi przechowywanymi ciągami stanowi dużą różnicę po ich zmianie. Załóżmy, że chcesz zapisać ciąg Hello w dwóch ciągach. Nie można przypisać ciągu do tablicy w następujący sposób:

cara = „Hello”; // Nieważny

Ponieważ nie można zmienić nazwy tablicy, nie można jej przypisać nowej wartości. Jedynym sposobem na zmianę zawartości tablicy jest przypisanie znaków tablicy z ciągu elementu jednocześnie lub za pomocą wbudowanej funkcji, takiej jak strcpy (). Możesz jednak ustawić tablicę znaków na nowy ciąg w następujący sposób:

cptr = „Hello”; // Zmień wskaźnik tak

// wskazuje na nowy ciąg.

WSKAZÓWKA: Jeśli chcesz przechowywać dane wejściowe użytkownika w ciągu wskazanym przez wskaźnik, najpierw musisz zarezerwować wystarczającą ilość miejsca dla tego ciągu wejściowego. Najłatwiejszym sposobem na to jest zarezerwowanie tablicy znaków, a następnie przypisanie wskaźnika do początkowego elementu tej tablicy:

char input[81]; // Holds a string as long as

// 80 characters.

char *iptr=input; // Also could have done this:

// char *iptr=&input[0];

Teraz możesz wprowadzić ciąg znaków za pomocą wskaźnika:

gets(iptr); // Make sure iptr points to

// ciąg wpisany przez użytkownika.

Możesz używać manipulacji wskaźnikami, arytmetyki i modyfikacji na łańcuch wejściowy.

Przykłady

1. Załóżmy, że chcesz zapisać imię i nazwisko swojej siostry i wydrukować je. Zamiast korzystać z tablic, możesz użyć wskaźnika postaci. Następujący program właśnie to robi.

```
// Nazwa pliku: C27CP1.CPP

// Przechowuje nazwę we wskaźniku postaci.

#include <iostream.h>

void main()

{

char *c=”Bettye Lou Horn”;

cout << “My sister’s name is “ << c << “\n”;

return;

}
```

Spowoduje to wydrukowanie następujących elementów:

My sister’s name is Bettye Lou Horn

2. Załóżmy, że musisz zmienić ciąg wskazany przez wskaźnik znaku. Jeśli twoja siostra zmieniła nazwisko na Henderson, twój program może wyświetlić oba ciągi w następujący sposób: Zidentyfikuj program i dołącz plik nagłówkowy we / wy. Ten program używa wskaźnika znakowego c, aby wskazać literał łańcuchowy w pamięci. Wskaż literał łańcucha i wydrukuj łańcuch. Ustaw wskaźnik znaku na nowy ciąg literału, a następnie wydrukuj nowy ciąg.

```
// Nazwa pliku: C27CP2.CPP

// Ilustruje zmianę ciągu znaków.

#include <iostream.h>

void main()

{

char *c=”Bettye Lou Horn”;

cout << “My sister’s maiden name was “ << c << “\n”;

c = “Bettye Lou Henderson”; // Assigns new string to c.

cout << “My sister’s married name is “ << c << “\n”;

return;

}
```

Dane wyjściowe są następujące:

My sister’s maiden name was Bettye Lou Horn

My sister’s married name is Bettye Lou Henderson

3. Nie używaj wskaźników znaków do zmiany stałych ciągów. Może to pomylić kompilator i prawdopodobnie nie uzyskasz oczekiwanych rezultatów. Poniższy program jest podobny do tych, które właśnie widziałeś. W tym przykładzie zamiast umieszczania wskaźnika znaku na nowym ciągu, podjęto próbę zmiany zawartości oryginalnego ciągu.

```
// Nazwa pliku: C27CP3.CPP

// Ilustruje nieprawidłową zmianę łańcucha znaków.

#include <iostream.h>

void main ()

{

char *c=”Bettye Lou Horn”;

cout << “My sister’s maiden name was “ << c << “\n”;

c += 11; // Makes c point to the last name

// (the twelfth character).

c = “Henderson”; // Assigns a new string to c.

cout << “My sister’s married name is “ << c << “\n”;

return;

}
```

Program wydaje się zmieniać nazwisko z Horn na Henderson, ale tak nie jest. Oto wynik tego programu:

```
My sister’s maiden name was Bettye Lou Horn

My sister’s married name is Henderson
```

Dlaczego nie wydrukowano pełnego ciągu? Ponieważ adres wskazany przez c został zwiększony o 11, c nadal wskazuje na Hendersona, więc to wszystko zostało wydrukowane.

4. Możesz zgadywać, jak naprawić poprzedni program. Zamiast wydrukować ciąg zapisany w punkcie c po przypisaniu go do Hendersona, możesz chcieć go zmniejszyć o 11, aby wskazywał na pierwotną lokalizację, początek nazwy. Kod do wykonania tej czynności jest następujący, ale nie działa zgodnie z oczekiwaniami. Przestudiuj program przed przeczytaniem wyjaśnienia.

```
// Nazwa pliku: C27CP4.C

// Ilustruje nieprawidłową zmianę łańcucha znaków.

#include <iostream.h>

void main ()

{

char *c=”Bettye Lou Horn”;

cout << “My sister’s maiden name was “ << c << “\n”;

c += 11; // Makes c point to the last

// name (the twelfth character).

c = “Henderson”; // Assigns a new string to c.
```

```
c -= 11; // Makes c point to its

// original location (???).

cout << “My sister’s married name is “ << c << “\n”;

return;

}
```

Ten program produkuje śmieci przy drugim cout. W tym programie są dwa dosłowne ciągi znaków. Kiedy po raz pierwszy przypisujesz c do Bettye Lou Horn, C++ rezerwuje miejsce w pamięci dla stałego łańcucha i umieszcza początkowy adres łańcucha w c. Gdy program przypisuje następnie c Hendersonowi, C++ znajduje miejsce na inną stałą znaków, jak pokazano na rysunku 27.3. Jeśli odejmiesz 11 od położenia c, po tym, jak wskazuje nowy ciąg Henderson, c wskazuje na obszar pamięci, którego program nie używa. Nie ma gwarancji, że dane do wydrukowania pojawią się przed ciągiem ciągłym Henderson. Jeśli chcesz manipulować częściami łańcucha, musisz zrobić to jednocześnie, podobnie jak w przypadku tablic.

Wskaźnik arytmetyczny

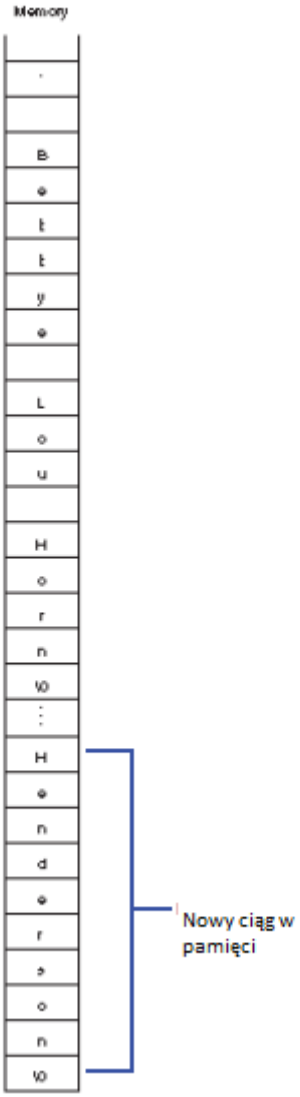
Widziałeś przykład arytmetyki wskaźnika, gdy uzyskiwałeś dostęp do elementów tablicy za pomocą notacji wskaźnika. Do tej pory powinieneś czuć się komfortowo z faktem, że oba odwołania do tablic lub wskaźników są identyczne:

ara [sub] i * (ara + sub)

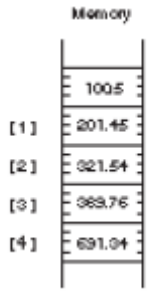
Możesz zwiększyć lub zmniejszyć wskaźnik. Jeśli zwiększysz wskaźnik, adres wewnątrz zmiennej wskaźnika zwiększy się. Jednak wskaźnik nie zawsze zwiększa się o jeden

Załóżmy, że f_ptr jest wskaźnikiem zmiennoprzecinkowym indeksującym pierwszy element tablicy liczb zmiennoprzecinkowych. Możesz zainicjować f_ptr w następujący sposób:

```
float fara [] = {100,5, 201,45, 321,54, 389,76, 691,34};

f_ptr = fara;
```



Rysunek pokazuje, jak te zmienne wyglądają w pamięci. Każda wartość zmiennoprzecinkowa w tym przykładzie zajmuje cztery bajty pamięci.



Jeśli wydrukujesz wartość * f_ptr, zobaczysz 100,5. Załóżmy, że zwiększasz f_ptr o jeden za pomocą następującej instrukcji:

f_ptr ++;

C++ nie dodaje go do adresu w f_ptr, nawet jeśli wydaje się, że należy go dodać. W tym przypadku, ponieważ wartości zmiennoprzecinkowe zajmują po cztery bajty na tym komputerze, C++ dodaje cztery

do f_ptr. Skąd C++ wie, ile bajtów dodać do f_ptr? C++ wie z deklaracji wskaźnika, ile bajtów wskaźników pamięci zajmuje. Dlatego musisz zadeklarować wskaźnik o poprawnym typie danych. Po zwiększeniu f_ptr, jeśli miałbyś wydrukować * f_ptr, zobaczyłbyś 201.45, drugi element w tablicy. Gdyby C++ dodał tylko jeden adres do f_ptr, f_ptr wskazywałby tylko na drugi bajt, 100.5. Spowoduje to wyrzucenie śmieci na ekran.

UWAGA: Po zwiększeniu wskaźnika, C++ dodaje do wskaźnika jeden pełny rozmiar typu danych (w bajtach), a nie jeden bajt. Podczas zmniejszania wskaźnika C++ odejmuje jeden pełny rozmiar typu danych (w bajtach) od wskaźnika.

Przykłady

1. Poniższy program definiuje tablicę z pięcioma wartościami. Wskaźnik liczb całkowitych jest następnie inicjowany w celu wskazania pierwszego elementu w tablicy. Reszta programu wypisuje zdereferencjonowaną wartość wskaźnika, a następnie zwiększa wskaźnik, aby wskazywał następną liczbę całkowitą w tablicy. Aby pokazać, co się dzieje, rozmiar wartości całkowitych jest wydrukowany na dole programu. Ponieważ (w tym przypadku) liczby całkowite zajmują dwa bajty, C++ zwiększa wskaźnik o dwa, aby wskazywał następną liczbę całkowitą. (Liczby całkowite są dwa bajty od siebie.)

```
// Nazwa pliku: C27PTI.CPP

// Zwiększa wskaźnik poprzez tablicę liczb całkowitych.

#include <iostream.h>

void main ()

{

int iara[] = {10,20,30,40,50};

int *ip = iara; // The pointer points to

// The start of the array.

cout << *ip << "\n";

ip++; // Two are actually added.

cout << *ip << "\n";

ip++; // Two are actually added.

cout << *ip << "\n";

ip++; // Two are actually added.

cout << *ip << "\n";

ip++; // Two are actually added.

cout << *ip << "\n\n";

cout << "The integer size is " << sizeof(int);

cout << " bytes on this machine \n\n";

return;
```

```
}
```

Oto wynik działania programu:

```
10

20

30

40

50

Rozmiar całkowity to dwa bajty na tym komputerze
```

2. Oto ten sam program wykorzystujący tablicę znaków i wskaźnik znaków. Ponieważ znak zajmuje tylko jeden bajt pamięci, zwiększenie wskaźnika znaku w rzeczywistości dodaje tylko jeden wskaźnik; potrzebny jest tylko jeden, ponieważ znaki dzieli tylko jeden bajt.

```
// Nazwa pliku: C27PTC.CPP

// Zwiększa wskaźnik poprzez tablicę znaków.

#include <iostream.h>

void main ()

{

char cara[] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’};

char *cp = cara; // The pointers point to

// the start of the array.

cout << *cp << “\n”;

cp++; // One is actually added.

cout << *cp << “\n”;

cp++; // One is actually added.

cout << *cp << “\n”;

cp++; // One is actually added.

cout << *cp << “\n”;

cp++; // One is actually added.

cout << *cp << “\n\n”;

cout << “The character size is “ << sizeof(char);

cout << “ byte on this machine\n”;

return;

}
```

3. Następny program pokazuje wiele sposobów dodawania, odejmowania i odwoływania się do tablic i wskaźników. Program definiuje tablicę zmiennoprzecinkową i wskaźnik zmiennoprzecinkowy. Treść programu drukuje wartości z tablicy przy użyciu notacji tablicowej i wskaźnikowej.

```
// Nazwa pliku: C27ARPT2.CPP

// Kompleksowe odniesienie do tablic i wskaźników.

#include <iostream.h>

void main ()

{

float ara[] = {100.0, 200.0, 300.0, 400.0, 500.0};

float *fptr; // Floating-point pointer.

// Make pointer point to array's first value.

fptr = &ara[0]; // Also could have been this:

// fptr = ara;

cout << *fptr << "\n"; // Prints 100.0

fptr++; // Points to next floating-point value.

cout << *fptr << "\n"; // Prints 200.0

fptr++; // Points to next floating-point value.

cout << *fptr << "\n"; // Prints 300.0

fptr++; // Points to next floating-point value.

cout << *fptr << "\n"; // Prints 400.0

fptr++; // Points to next floating-point value.

cout << *fptr << "\n"; // Prints 500.0

fptr = ara; // Points to first element again.

cout << *(fptr+2) << "\n"; // Prints 300.00 but

// does not change fptr.

// References both array and pointer using subscripts.

cout << (fptr+0)[0] << " " << (ara+0)[0] << "\n";

// 100.0 100.0

cout << (fptr+1)[0] << " " << (ara+1)[0] << "\n";

// 200.0 200.0

cout << (fptr+4)[0] << " " << (ara+4)[0] << "\n";

// 500.0 500.0
```

return;

}

Poniżej przedstawiono dane wyjściowe z tego programu:100.0

200.0

300.0

400.0

500.0

300.0

100.0 100.0

200.0 200.0

500.0 500.0

Tablice ciągów

Teraz jesteś gotowy na jedną z najbardziej przydatnych aplikacji wskaźników znaków: przechowywanie tablic ciągów znaków. W rzeczywistości nie można przechowywać tablicy ciągów, ale można przechowywać tablicę wskaźników znaków, a każdy wskaźnik znaków może wskazywać łańcuch w pamięci. Definiując tablicę wskaźników znaków, definiujesz poszarpaną krawędź tablicy. Tablica o nierównych krawędziach jest podobna do tabeli dwuwymiarowej, z tym wyjątkiem, że każdy wiersz zawiera inną liczbę znaków (zamiast być tej samej długości). Słowo nierówne pochodzi od użycia edytorów tekstu. Edytor tekstu zazwyczaj może drukować tekst w pełni uzasadniony lub z nierównym prawym marginesem. Kolumny tego akapitu są w pełni uzasadnione, ponieważ lewa i prawa kolumna są wyrównane równomiernie. Listy pisane ręcznie i na maszynie do pisania (pamiętasz, czym jest maszyna do pisania?) Mają na ogół niewyrównane marginesy. Trudno jest pisać, więc każda linia kończy się dokładnie w tej samej prawej kolumnie. Wszystkie dwuwymiarowe tabele, które do tej pory widziałeś, są w pełni uzasadnione. Na przykład, jeśli zadeklarujesz tabelę znaków z pięcioma wierszami i 20 kolumnami, każdy wiersz będzie zawierał tę samą liczbę znaków. Możesz zdefiniować tabelę za pomocą następującej instrukcji:

imiona znaków [5] [20] = {{„„ George ”},

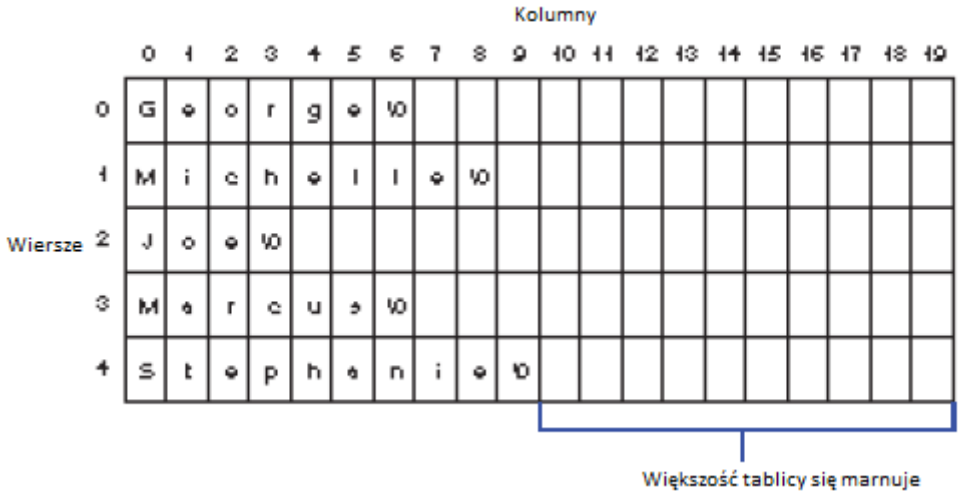
{„Michelle”},

{„Joe”},

{„Marcus”},

{„Stephanie”}};

Ta tabela jest pokazana na rysunku. Zauważ, że duża część tablicy to zmarnowane miejsce. Każdy wiersz zajmuje 20 znaków, mimo że dane w każdym wierszu zajmują znacznie mniej znaków. Niewypełnione elementy zawierają zera zerowe, ponieważ C++ niweluje wszystkie elementy, których nie inicjujesz w tablicach. Ten typ tabeli zużywa zbyt dużo pamięci



Aby rozwiązać problem marnowania pamięci w pełni uzasadnionych tabelach, należy zadeklarować jednowymiarową tablicę wskaźników znaków. Każdy wskaźnik wskazuje łańcuch w pamięci, a łańcuchy nie muszą być tej samej długości. Oto definicja takiej tablicy:

char *names[5]={ {“George”},

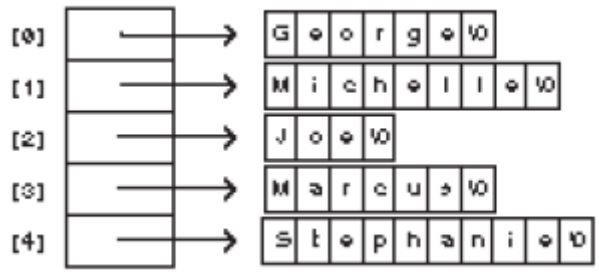
{“Michelle”},

{“Joe”},

{“Marcus”},

{“Stephanie”} };

Ta tablica jest jednowymiarowa. Definicja nie powinna cię mylić, chociaż jest to coś, czego nie widziałeś. Gwiazdka przed nazwami tworzy tablicę wskaźników. Typ danych wskaźników to znak. Ciągi nie są przypisywane do elementów tablicy, ale są wskazywane przez elementy tablicy. Rysunek pokazuje tę tablicę wskaźników. Ciągi są przechowywane w innym miejscu w pamięci. Ich rzeczywiste lokalizacje nie są krytyczne, ponieważ każdy wskaźnik wskazuje na postać początkową. Ciągi nie marnują żadnych danych. Każdy ciąg zajmuje tylko tyle pamięci, ile potrzebuje ciąg i jego końcowe zero. Dzięki temu dane mają nierówny wygląd.



Aby wyświetlić pierwszy ciąg, użyłbyś tego cout:

cout << *names; // Prints George

Aby wydrukować drugi ciąg, użyłbyś tego cout:

cout << *(names+1); // Prints Michelle

Za każdym razem, gdy wyreżyserujesz dowolny element wskaźnika za pomocą operatora * dereferencji, uzyskasz dostęp do jednego z ciągów w tablicy. Możesz użyć elementu pozbawionego odniesień w dowolnym miejscu, w którym używasz stałej łańcuchowej lub tablicy znaków (za pomocą strcpy (), strcmp () itd.).

WSKAZÓWKA: Praca ze wskaźnikami do łańcuchów jest znacznie wydajniejsza niż bezpośrednia praca z łańcuchami. Na przykład sortowanie listy ciągów zajmuje dużo czasu, jeśli są przechowywane jako w pełni uzasadniona tabela. Sortowanie ciągów wskazywanych przez tablicę wskaźników jest znacznie szybsze. Podczas sortowania zamieniasz tylko wskaźniki, a nie całe łańcuchy.

Przykłady

1. Oto pełny program, który używa tablicy wskaźników z pięcioma nazwami. Pętla for kontroluje funkcję cout, drukując każdą nazwę w danych ciągu. Teraz możesz zobaczyć, dlaczego nauka o notacji wskaźnikowej dla tablic się opłaca!

```
// Nazwa pliku: C27PTST1.CPP

// Wyświetla ciągi wskazane przez tablicę.

#include <iostream.h>

void main ()

{

char *name[5]={ {“George”}, // Defines a ragged-edge

{“Michelle”}, // array of pointers to

{“Joe”}, // strings.

{“Marcus”},

{“Stephanie”} };

int ctr;

for (ctr=0; ctr<5; ctr++)

{ cout << “String #” << (ctr+1) <<

“ is “ << *(name+ctr) << “\n”; }

return;

}
```

Poniżej przedstawiono dane wyjściowe z tego programu:

```
String #1 is George

String #2 is Michelle

String #3 is Joe

String #4 is Marcus
```

String #5 is Stephanie

2. Poniższy program przechowuje dni tygodnia w tablicy. Gdy użytkownik wpisze liczbę od 1 do 7, zostanie wyświetlony dzień tygodnia, który pasuje do tej liczby (przy niedzieli 1), usuwając odwołanie ze wskaźnika wskazującego na ten ciąg.

```
// Nazwa pliku: C27PTST2.CPP

// Drukuje dzień tygodnia na podstawie wartości wejściowej.

#include <iostream.h>

void main ()

{

char *days[] = {“Sunday”, // The seven separate sets

“Monday”, // of braces are optional.

“Tuesday”,

“Wednesday”,

“Thursday”,

“Friday”,

“Saturday”};

int day_num;

do

{ cout << “What is a day number (from 1 to 7)? “;

cin >> day_num;

} while ((day_num<1) || (day_num>7)); // Ensures

// an accurate number.

day_num--; // Adjusts for subscript.

cout << “The day is “ << *(days+day_num) << “\n”;

return;

}
```

Ćwiczenia

1. Napisz program do przechowywania nazw członków rodziny w tablicy znaków wskaźników. Wydrukuj nazwiska.

2. Napisz program, który prosi użytkownika o 15 średnich dziennych notowań giełdowych i przechowuje te średnie w tablicy zmiennoprzecinkowej. Używając tylko notacji wskaźnikowej, wypisz tablicę do przodu i do tyłu. Ponownie używając tylko notacji wskaźnikowej, wydrukuj najwyższe i najniższe notowania giełdowe na liście.

3. Zmodyfikuj sortowanie bąbelkowe pokazane w Rozdziale 24, „Przetwarzanie macierzy”, tak aby sortowało się za pomocą notacji wskaźnikowej. Dodaj ten rodzaj bąbelków do programu w ćwiczeniu 2, aby wydrukować średnie giełdowe w porządku rosnącym

4. Napisz program, który żąda od użytkownika 10 tytułów utworów. Przechowuj tytuły w tablicy wskaźników znaków (tablica poszarpana). Wydrukuj oryginalne tytuły, wydrukuj tytuły alfabetyczne i wydrukuj tytuły w odwrotnej kolejności alfabetycznej (od Z do A).

Podsumowanie

Zasługujesz na przerwę! Rozumiesz teraz podstawy wskaźników i notacji tablicowych C++. Po opanowaniu tej sekcji, jesteś na dobrej drodze do myślenia w C++ podczas projektowania programów. Programiści C++ wiedzą, że tablice C++ są ukrytymi wskaźnikami i odpowiednio je programują. Możliwość korzystania z tablic o nierównych krawędziach ma dwie zalety: Możesz przechowywać tablice danych ciągów bez marnowania dodatkowego miejsca i możesz szybko zmieniać wskaźniki bez konieczności przenoszenia danych ciągów w pamięci. W miarę postępów w zaawansowanych koncepcjach C++ docenisz czas spędzony na doskonaleniu notacji wskaźnikowej.

Struktury

Korzystając ze struktur, możesz grupować dane i pracować z zgrupowanymi danymi jako całością. Przetwarzanie danych biznesowych wykorzystuje pojęcie struktur w prawie każdym programie. Możliwość manipulowania kilkoma zmiennymi jako pojedynczą grupą ułatwia zarządzanie programami.

W tej sekcji przedstawiono następujące pojęcia:

♦ Definicje struktury

♦ Inicjalizacja struktur

♦ Operator kropki (.)

♦ Przypisanie struktury

♦ Zagnieżdżone struktury

Wprowadzenie do struktur

Struktura to zbiór jednego lub więcej typów zmiennych. Jak wiadomo, każdy element w tablicy musi być tego samego typu danych i należy odwoływać się do całej tablicy po nazwie. Każdy element (zwany członkiem) w strukturze może mieć inny typ danych. Załóżmy, że chcesz śledzić swoją kolekcję płyt CD. Możesz śledzić następujące informacje o każdej płycie CD:

Tytuł

Artysta

Liczba utworów

Koszt

Data zakupu

W tej strukturze CD byłoby pięć składowych członków.

WSKAZÓWKA: Jeśli programowałeś w innych językach komputerowych lub kiedykolwiek korzystałeś z programu bazodanowego, struktury C++ są analogiczne do rekordów plików, a członkowie są analogiczni do pól w tych rekordach.

Po podjęciu decyzji o członkach musisz zdecydować, jakie dane wpisze każda składowa. Tytuł i artysta to tablice znaków, liczba utworów jest liczbą całkowitą, koszt zmiennoprzecinkowy, a data to kolejna tablica znaków. Informacje te są reprezentowane w następujący sposób:

Nazwa członka: typ danych

Tytuł znaku: tablica 25 znaków

Postać wykonawcy: tablica 20 znaków

Liczba utworów: Integer

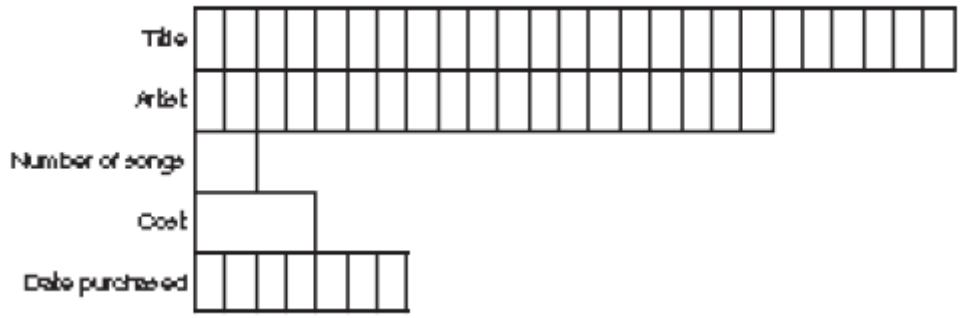
Koszt: zmiennoprzecinkowy

Data zakupu: tablica ośmiu znaków

Każda zdefiniowana struktura może mieć powiązaną nazwę struktury zwaną znacznikiem struktury. Znaczniki struktury w większości przypadków nie są wymagane, ale na ogół najlepiej jest zdefiniować jeden dla każdej struktury w programie. Znacznik struktury nie jest nazwą zmiennej. W przeciwieństwie do nazw tablic, które odwołują się do tablicy jako zmiennych, znacznik struktury jest po prostu etykietą formatu struktury. Sam nazywasz znaczniki struktury, stosując te same reguły nazewnictwa dla zmiennych. Jeśli nadasz strukturze CD znacznik struktury o nazwie cd_collection, informujesz C++, że znacznik o nazwie cd_collection wygląda jak dwie tablice znaków, po których następuje liczba całkowita, wartość zmiennoprzecinkowa i końcowa tablica znaków. Znacznik struktury jest właściwie nowo zdefiniowanym typem danych, który programista definiuje. Kiedy chcesz zapisać liczbę całkowitą, nie musisz definiować w C++ czym jest liczba całkowita. C++ już rozpoznaje liczbę całkowitą. Jeśli jednak chcesz przechowywać dane kolekcji CD, C++ nie jest w stanie rozpoznać, jaki format ma kolekcja CD. Musisz powiedzieć C++ (używając opisanego tutaj przykładu), że potrzebujesz nowego typu danych. Ten typ danych będzie twoim znacznikiem struktury, w tym przykładzie nazwanym cd_collection, i wygląda jak poprzednio opisana struktura (dwie tablice znaków, liczba całkowita, liczba zmiennoprzecinkowa i tablica znaków).

UWAGA: Żadna pamięć nie jest zarezerwowana dla znaczników struktury. Znacznik struktury to Twój własny typ danych. C++ nie rezerwuje pamięci dla typu danych liczb całkowitych, dopóki nie zadeklarujesz zmiennej liczby całkowitej. C++ nie rezerwuje pamięci dla struktury, dopóki nie zadeklarujesz zmiennej struktury.

Rysunek pokazuje strukturę CD, przedstawiając graficznie typy danych w strukturze. Zauważ, że jest pięć składowych, a każda składowa ma inny typ danych. Cała struktura nazywa się cd_collection, ponieważ jest to znacznik struktury.



Przykłady

1. Załóżmy, że zostałeś poproszony o napisanie programu dla systemu inwentaryzacji firmy. Firma używa systemu ewidencji kartotek do śledzenia następujących pozycji:

Nazwa przedmiotu

Ilość w magazynie

Ilość na zamówienie

Cena detaliczna

Cena hurtowa

Byłoby to idealne zastosowanie w przypadku struktury zawierającej pięć elementów. Przed zdefiniowaniem struktury należy określić typy danych każdego członka. Po zadaniu pytań na temat

zakresu danych (musisz znać największą nazwę elementu i najwyższą możliwą ilość, która pojawiłaby się w celu zapewnienia, że Twoje typy danych mogą przechowywać dane), decydujesz się użyć następującego znacznika struktury i typów danych:

Składowa: typ danych

Nazwa przedmiotu: Tablica 20 znaków

Ilość w magazynie: długa int

Ilość na zamówienie: długa int

Cena detaliczna: podwójna

Cena hurtowa: podwójna

2. Załóżmy, że ta sama firma chciała również, abyś napisał program do śledzenia swoich miesięcznych i rocznych wynagrodzeń oraz wydrukował raport na koniec roku, który pokazywał indywidualne wynagrodzenie każdego miesiąca i łączne wynagrodzenie na koniec roku. Jak wyglądałaby struktura? Bądź ostrożny! Ten typ danych prawdopodobnie nie potrzebuje struktury. Ponieważ wszystkie miesięczne pensje muszą być tego samego typu danych, zmiennoprzecinkowa lub podwójna tablica zmiennoprzecinkowa ładnie utrzymuje miesięczne pensje bez złożoności struktury. Struktury są przydatne do śledzenia danych, które należy pogrupować, takich jak dane magazynowe, nazwisko i dane adresowe klienta lub plik danych pracownika.

Definiowanie struktur

Aby zdefiniować strukturę, musisz użyć instrukcji struct. Instrukcja struct definiuje nowy typ danych z więcej niż jednym członkiem dla twojego programu. Format instrukcji struct to

struct [structure tag]

{

member definition;

member definition;

:

member definition;

} [one or more structure variables];

Jak wspomniano wcześniej, znacznik struktury jest opcjonalny (stąd nawiasy w formacie). Każda definicja elementu jest normalną definicją zmiennej, taką jak int i; lub sprzedaż typu float [20]; lub dowolna inna poprawna definicja zmiennej, w tym wskaźniki zmiennych, jeśli struktura wymaga wskaźnika jako elementu. Na końcu definicji struktury, przed końcowym średnikiem, możesz określić jedną lub więcej zmiennych struktury. Jeśli podasz zmienną strukturalną, poprosisz C++ o zarezerwowanie miejsca dla tej zmiennej. Dzięki temu C++ rozpoznaje, że zmienna nie jest liczbą całkowitą, znakiem ani żadnym innym wewnętrznym typem danych. C++ rozpoznaje również, że zmienna musi być typem przypominającym strukturę. Może wydawać się dziwne, że członkowie nie rezerwują miejsca, ale nie robią tego. Zmienne struktury tak. Staje się to jasne w poniższych przykładach. Oto sposób zadeklarowania struktury CD:

struct cd_collection

```
{
char title[25];
char artist[20];
int num_songs;
float price;
char date_purch[9];
} cd1, cd2, cd3;
```

Zanim przejdziesz dalej, powinieneś być w stanie odpowiedzieć na następujące pytania dotyczące tej struktury:

♦ Co to jest znacznik struktury?

♦ Ile jest składowych?

♦ Jakie są typy danych składowych?

♦ Jakie są nazwy składowych?

♦ Ile jest zmiennych strukturalnych?

♦ Jakie są ich nazwy?

Znacznik struktury nazywa się cd_collection. Istnieje pięć elementów, dwie tablice znaków, liczba całkowita, zmiennoprzecinkowa i tablica znaków. Nazwiska członków to tytuł, wykonawca, num_songs, cena i date_purch. Istnieją trzy zmienne struktury - cd1, cd2 i cd3.

WSKAZÓWKA: Często można wizualizować zmienne struktury jako system inwentaryzacji plików kart. Rysunek pokazuje, jak możesz przechowywać swoją kolekcję płyt CD w pliku karty 3 na 5. Każda płyta CD pobiera jedną kartę (reprezentowaną przez zmienną struktury), która zawiera informacje o tej płycie CD (elementy struktury).

Gdybyś miał 1000 płyt CD, musiałbyś zadeklarować 1000 zmiennych strukturalnych. Oczywiście nie chciałbyś wymieniać tylu zmiennych struktury na końcu definicji struktury. Aby pomóc zdefiniować

struktury dla dużej liczby wystąpień, musisz zdefiniować tablicę struktury. Na razie skoncentruj się na zapoznaniu się z definicjami konstrukcji.

Przykłady

1. Oto definicja struktury aplikacji inwentaryzacyjnej opisana wcześniej w tym rozdziale.

```
struct inventory
{
char item_name[20];
long int in_stock;
long int order_qty;
float retail;
float wholesale;
} item1, item2, item3, item4;
```

Zdefiniowano cztery zmienne struktury zapasów. Każda zmienna struktury - item1, item2, item3 i item4 - wygląda jak struktura.

2. Załóżmy, że firma chce śledzić swoich klientów i personel. Poniższe dwie definicje struktur utworzyłyby pięć zmiennych struktury dla każdej struktury. Ten przykład, mający pięciu pracowników i pięciu klientów, jest bardzo ograniczony, ale pokazuje, jak można zdefiniować struktury.

```
struct employees
{
char emp_name[25]; // Employee's full name.
char address[30]; // Employee's address.
char city[10];
char state[2];
long int zip;
double salary; // Annual salary.
} emp1, emp2, emp3, emp4, emp5;
struct customers
{
char cust_name[25]; // Customer's full name.
char address[30]; // Customer's address.
char city[10];
char state[2];
```

long int zip;

double balance; // Balance owed to company.

} cust1, cust2, cust3, cust4, cust5;

Każda struktura ma podobne dane. W dalszej części dowiesz się, jak skonsolidować podobne definicje elementów, tworząc struktury zagnieżdżone.

PORADA: Umieść komentarze po prawej stronie składowych, aby udokumentować cel członków.

Inicjalizacja danych struktury

Istnieją dwa sposoby inicjowania elementów struktury. Możesz zainicjować składowe, kiedy deklarujesz strukturę, i możesz zainicjować strukturę w treści programu. Większość programów stosuje tę drugą metodę, ponieważ nie zawsze znasz struktury danych podczas pisania programu. Oto przykład struktury zadeklarowanej i zainicjowanej w tym samym czasie:

struct cd_collection

{

char title[25];

char artist[20];

int num_songs;

float price;

char date_purch[9];

} cd1 = {“Red Moon Men”, “Sam and the Sneeds”,

12, 11.95, “02/13/92”};

Kiedy po raz pierwszy dowiadujesz się o strukturach, możesz mieć ochotę zainicjować składowe indywidualnie wewnątrz struktury, np

char artist [20] = „Sam and the Sneeds”; // Nieważny

Nie można zainicjować poszczególnych członków, ponieważ nie są to zmienne. Możesz przypisywać tylko wartości do zmiennych. Jedyną zmienną struktury w tej strukturze jest cd1. Nawiasy klamrowe muszą obejmować dane, które inicjujesz, w zmiennych strukturalnych, tak jak obejmują dane podczas inicjowania tablic. Ta metoda inicjowania zmiennych struktury staje się uciążliwa, gdy istnieje kilka zmiennych struktury (jak zwykle są). Umieszczenie danych w kilku zmiennych, każdy zestaw danych w nawiasach klamrowych, staje się nieporządny i zajmuje zbyt dużo miejsca w kodzie. Co ważniejsze, zwykle nie znasz nawet zawartości zmiennych strukturalnych. Zasadniczo użytkownik wprowadza dane, które mają być przechowywane w strukturach, lub czyta je z pliku dyskowego. Lepszym podejściem do inicjowania struktur jest użycie operatora kropki (.). Operator kropki jest jednym ze sposobów inicjowania poszczególnych elementów zmiennej struktury w treści twojego programu. Za pomocą operatora kropki możesz traktować każdy element struktury prawie tak, jakby był zwykłą zmienną niestrukturalną. Format operatora kropki to

nazwa_zmiennej_struktury. nazwa_składowej

Po nazwie zmiennej struktury musi zawsze występować operator kropki, a nazwa elementu musi zawsze pojawiać się za operatorem kropki. Korzystanie z operatora kropki jest łatwe, jak pokazują poniższe przykłady.

Przykłady

1. Oto prosty program wykorzystujący strukturę kolekcji CD i operator kropki do inicjalizacji struktury. Zauważ, że program traktuje skłdowe jak zmienne regularne w połączeniu z operatorem kropki. Zidentyfikuj program i dołącz niezbędny plik nagłówka. Zdefiniuj zmienną struktury CD z pięcioma elementami. Wypełnij zmienną struktury CD danymi, a następnie wydrukuj ją.

```
// Nazwa pliku: C28ST1.CPP

// Inicjalizacja struktury za pomocą kolekcji CD.

#include <iostream.h>

#include <string.h>

void main ()

{

struct cd_collection

{

char title[25];

char artist[20];

int num_songs;

float price;

char date_purch[9];

} cd1;

// Initialize members here.

strcpy(cd1.title, “Red Moon Men”);

strcpy(cd1.artist, “Sam and the Sneeds”);

cd1.num_songs=12;

cd1.price=11.95;

strcpy(cd1.date_purch, “02/13/92”);

// Print the data to the screen.

cout << “Here is the CD information:\n\n”;

cout << “Title: “ << cd1.title << “\n”;

cout << “Artist: “ << cd1.artist << “\n”;

cout << “Songs: “ << cd1.num_songs << “\n”;
```

```
cout << “Price: “ << cd1.price << “\n”;

cout << “Date purchased: “ << cd1.date_purch << “\n”;

return;

}
```

Oto wynik tego programu:

```
Here is the CD information:

Title: Red Moon Men

Artist: Sam and the Sneeds

Songs: 12

Price: 11.95

Date purchased: 02/13/92
```

2. Korzystając z operatora kropki, możesz odbierać dane struktury z klawiatury za pomocą dowolnej znanej funkcji wprowadzania danych, takiej jak cin, gets () i get. Poniższy program prosi użytkownika o informacje dla studentów. Aby przykład był dość krótki, w programie zdefiniowano tylko dwóch studentów.

```
// Nazwa pliku: C28ST2.CPP

// Struktura danych wejściowych z danymi ucznia.

#include <iostream.h>

#include <string.h>

#include <iomanip.h>

#include <stdio.h>

void main ()

{

struct students

{

char name[25];

int age;

float average;

} student1, student2;

// Get data for two students.

cout << “What is first student’s name? “;

gets(student1.name);
```

```
cout << “What is the first student’s age? “;
cin >> student1.age;
cout << “What is the first student’s average? “;
cin >> student1.average;
fflush(stdin); // Clear input buffer for next input.
cout << “\nWhat is second student’s name? “;
gets(student2.name);
cout << “What is the second student’s age? “;
cin >> student2.age;
cout << “What is the second student’s average? “;
cin >> student2.average;
// Print the data.
cout << “\n\nHere is the student information you “ <<
“entered:\n\n”;
cout << “Student #1:\n”;
cout << “Name: “ << student1.name << “\n”;
cout << “Age: “ << student1.age << “\n”;
cout << “Average: “ << setprecision(2) << student1.average
<< “\n”;
cout << “\nStudent #2:\n”;
cout << “Name: “ << student2.name << “\n”;
cout << “Age: “ << student2.age << “\n”;
cout << “Average: “ << student2.average << “\n”;
return;
}
```

Oto wynik tego programu:

```
What is first student’s name? Larry
What is the first student’s age? 14
What is the first student’s average? 87.67
What is second student’s name? Judy
What is the second student’s age? 15
```

What is the second student's average? 95.38

Here is the student information you entered:

Student #1:

Name: Larry

Age: 14

Average: 87.67

Student #2:

Name: Judy

Age: 15

Average: 95.38

3. Zmienne struktury są przekazywane przez kopię, a nie przez adres jak tablice. Dlatego jeśli wypełnisz strukturę w funkcji, musisz zwrócić ją do funkcji wywołującej, aby funkcja wywołująca mogła rozpoznać strukturę lub użyć globalnych zmiennych struktury, co na ogół nie jest zalecane.

WSKAZÓWKA: Dobrym rozwiązaniem problemu lokalnej / globalnej struktury jest: Zdefiniuj swoje struktury globalnie bez żadnych zmiennych struktur. Zdefiniuj wszystkie zmienne struktury lokalnie do funkcji, które ich potrzebują. Tak długo, jak twoja definicja struktury jest globalna, możesz deklarować lokalne zmienne struktury z tej struktury. Wszystkie kolejne przykłady wykorzystują tę metodę. Znacznik struktury odgrywa ważną rolę w problemie lokalnym / globalnym. Użyj znacznika struktury, aby zdefiniować lokalne zmienne struktury. Poniższy program jest podobny do poprzedniego. Zauważ, że struktura ucznia jest zdefiniowana globalnie bez żadnych zmiennych struktury. W każdej funkcji lokalne zmienne struktury są deklarowane przez odniesienie do znacznika struktury. Tag struktury zapobiega konieczności ponownego definiowania elementów struktury za każdym razem, gdy definiujesz nową zmienną struktury.

```
// Nazwa pliku: C28ST3.CPP

// Struktura danych wejściowych z danymi ucznia przekazywanymi do funkcji.

#include <iostream.h>

#include <string.h>

#include <stdio.h>

#include <iomanip.h>

struct students fill_structs(struct students student_var);

void pr_students(struct students student_var);

struct students // A global structure.

{

char name[25];

int age;
```

```
float average;
}; // No memory reserved.
void main()
{
students student1, student2; // Defines two
// local variables.
// Call function to fill structure variables.
student1 = fill_structs(student1); // student1
// is passed by copy, so it must be
// returned for main() to recognize it.
student2 = fill_structs(student2);
// Print the data.
cout << "\n\nHere is the student information you";
cout << " entered:\n\n";
pr_students(student1); // Prints first student's data.
pr_students(student2); // Prints second student's data.
return;
}
struct students fill_structs(struct students student_var)
{
// Get student's data
fflush(stdin); // Clears input buffer for next input.
cout << "What is student's name? ";
gets(student_var.name);
cout << "What is the student's age? ";
cin >> student_var.age;
cout << "What is the student's average? ";
cin >> student_var.average;
return (student_var);
}
void pr_students(struct students student_var)
```

```
{
cout << “Name: “ << student_var.name << “\n”;
cout << “Age: “ << student_var.age << “\n”;
cout << “Average: “ << setprecision(2) <<
student_var.average << “\n”;
return;
}
```

Prototyp i definicja funkcji fill_structs() może wydawać się skomplikowana, ale jest zgodna z tym samym schematem. Przed nazwą funkcji należy zadeklarować nieważność lub ustawić zwracany typ danych, jeśli funkcja zwraca wartość. Funkcja fill_structs () zwraca wartość, a typem zwracanej wartości jest struktura studentów.

4. Ponieważ dane dotyczące struktury są niczym więcej niż zwykłymi zmiennymi zgrupowanymi razem, nie krępuj się obliczać za pomocą elementów struktury. Tak długo, jak używasz operatora kropki, możesz traktować elementy struktury tak jak inne zmienne. Poniższy przykład wymaga salda klienta i wykorzystuje stopę dyskontową zawartą w strukturze klienta do obliczenia nowego salda. Aby skrót był krótki, dane struktury są inicjowane w zmiennym czasie deklaracji. Ten program w rzeczywistości nie wymaga struktur, ponieważ używany jest tylko jeden klient. Można zastosować pojedyncze zmienne, ale nie ilustrują one koncepcji obliczania za pomocą struktur.

```
// Nazwa pliku: C28CUST.CPP
// Aktualizuje saldo klienta w strukturze.
#include <iostream.h>
#include <iomanip.h>
struct customer_rec
{
char cust_name [25];
double balance;
float dis_rate;
} ;
void main()
{
struct customer_rec customer = {“Steve Thompson”,
431.23, .25};
cout << “Before the update, “ << customer.cust_name;
cout << “ has a balance of $” << setprecision(2) <<
```

```
customer.balance << “\n”;

// Update the balance

customer.balance *= (1.0-customer.dis_rate);

cout << “After the update, “ << customer.cust_name;

cout << “ has a balance of $” << customer.balance << “\n”;

return;

}
```

5. Możesz skopiować elementy jednej zmiennej struktury do elementów innej, o ile obie struktury mają ten sam format. Niektóre starsze wersje C++ wymagają kopiowania każdego elementu osobno, gdy chcesz skopiować jedną zmienną struktury do drugiej, ale AT&T C++ ułatwia powielanie zmiennych struktury. Poniższy program deklaruje trzy zmienne strukturalne, ale inicjuje tylko pierwszą z danymi. Pozostałe dwie są następnie inicjowane przez przypisanie im pierwszej zmiennej struktury.

```
// Nazwa pliku: C28STCPY.CPP

// Demonstruje przypisywanie jednej struktury do drugiej.

#include <iostream.h>

#include <iomanip.h>

struct student

{

char st_name[25];

char grade;

int age;

float average;

};

void main()

{

student std1 = {“Joe Brown”, ‘A’, 13, 91.4};

struct student std2, std3; // Not initialized

std2 = std1; // Copies each member of std1

std3 = std1; // to std2 and std3.

cout << “The contents of std2:\n”;

cout << std2.st_name << “ “ << std2.grade << “ “;

cout << std2.age << “ “ << setprecision(1) << std2.average
```

<< “\n\n”;

cout << “The contents of std3:\n”;

cout << std3.st_name << “ “ << std3.grade << “ “;

cout << std3.age << “ “ << std3.average << “\n”;

return;

}

Oto wynik działania programu:

The contents of std2

Joe Brown, A, 13, 91.4

The contents of std3

Joe Brown, A, 13, 91.4

Zauważ, że każdy element std1 został przypisany do std2 i std3 z dwoma pojedynczymi przypisaniami.

Struktury zagnieżdżone

C++ daje możliwość zagnieżdżenia jednej definicji struktury w innej. Oszczędza to czas, gdy piszesz programy korzystające z podobnych struktur. Musisz zdefiniować wspólnych członków tylko raz w ich własnej strukturze, a następnie użyć tej struktury jako członka w innej strukturze. Poniższe definicje struktur ilustrują ten punkt:

struct employees

{

char emp_name[25]; // Employee’s full name.

char address[30]; // Employee’s address.

char city[10];

char state[2];

long int zip;

double salary; // Annual salary.

};

struct customers

{

char cust_name[25]; // Customer’s full name.

char address[30]; // Customer’s address.

char city[10];

char state[2];

long int zip;

double balance; // Balance owed to company.

};

Struktury te przechowują różne dane. Jedna struktura dotyczy danych pracowników, a druga przechowuje dane klientów. Chociaż dane powinny być przechowywane osobno (nie chcesz wysyłać klientowi wypłaty!), Definicje struktur nakładają się na siebie i można je skonsolidować, tworząc trzecią strukturę. Załóżmy, że utworzyłeś następującą strukturę:

struct address_info

{

char address[30]; // Common address information.

char city[10];

char state[2];

long int zip;

};

Ta struktura może być następnie użyta jako element członkowski w innych strukturach takich jak ta:

struct employees

{

char emp_name[25]; // Employee's full name.

address_info e_address; // Employee's address.

double salary; // Annual salary.

};

struct customers

{

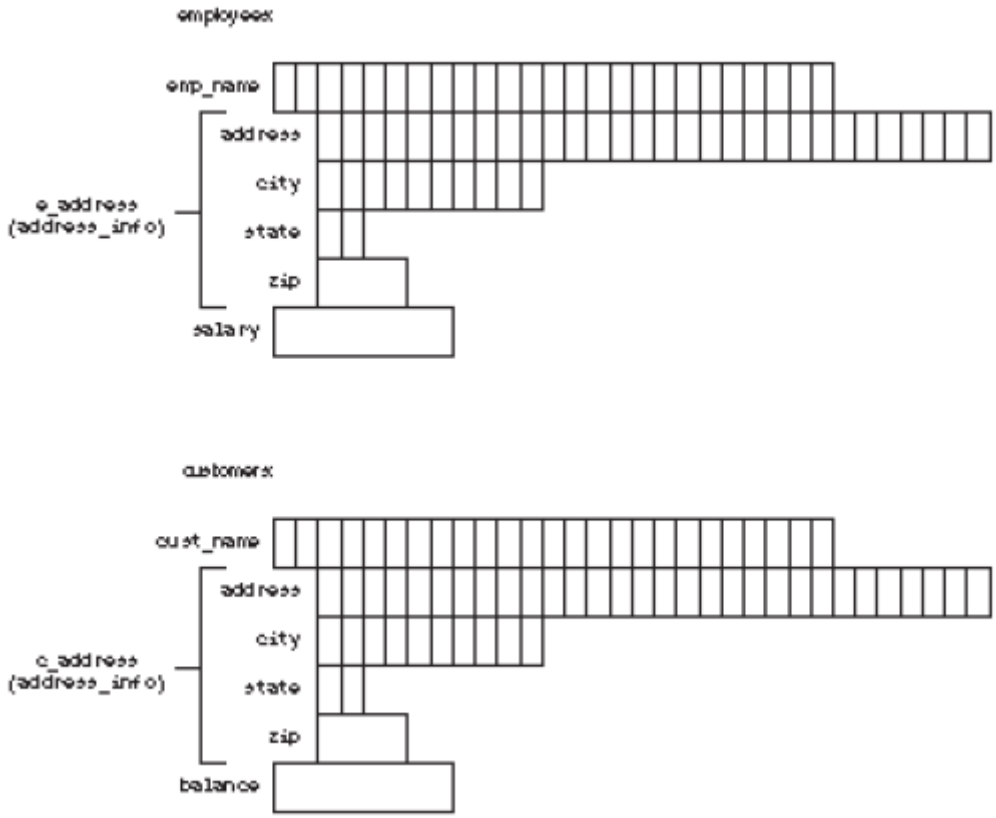
char cust_name[25]; // Customer's full name.

address_info c_address; // Customer's address.

double balance; // Balance owed to company.

};

Ważne jest, aby zdać sobie sprawę, że istnieją w sumie trzy struktury i że mają one tagi adres_info, pracownicy i klienci. Ilu członków ma struktura pracowników? Jeśli odpowiedziałeś na trzy, masz rację. W pracownikach i klientach jest trzech członków. Struktura pracowników ma strukturę tablicy znaków, po której następuje struktura adres_info, po której następuje podwójne zmiennoprzecinkowe wynagrodzenie. Rysunek pokazuje, jak wyglądają te struktury.



Po zdefiniowaniu struktury, staje się ona nowym typem danych w programie i może być używana wszędzie tam, gdzie może się pojawić typ danych (taki jak int, float itp.). Możesz przypisać wartości członków za pomocą operatora kropki. Aby przypisać saldo klienta, wpisz coś takiego:

customer.balance = 5643,24;

Zagnieżdżona struktura może wydawać się stanowić problem. Jak przypisać wartość do jednego z zagnieżdżonych elementów? Używając operatora kropki, musisz zagnieździć operatora kropki tak samo, jak zagnieżdżasz definicje struktury. Przypisujesz wartość do kodu pocztowego klienta w następujący sposób:

customer.c_address.zip = 34312;

Aby przypisać wartość do kodu pocztowego pracownika, wykonaj następujące czynności:

employee.e_address.zip = 59823;

Ćwiczenia

1. Napisz strukturę w programie, który śledzi zapasy taśm w sklepie wideo. Upewnij się, że struktura zawiera tytuł taśmy, długość taśmy (w minutach), początkową cenę zakupu taśmy, cenę wypożyczenia taśmy i datę premiery filmu.

2. Napisz program, korzystając ze struktury zadeklarowanej w ćwiczeniu 1. Zdefiniuj trzy zmienne struktury i zainicjuj je, gdy deklarujesz zmienne danymi. Wydrukuj dane na ekranie.

3. Napisz program nauczyciela, aby śledzić nazwiska 10 uczniów, ich wiek, oceny literowe i iloraz inteligencji. Użyj 10 różnych nazw zmiennych struktur i pobierz dane dla uczniów w pętli for z

klawiatury. Wydrukuj dane na drukarce, gdy nauczyciel zakończy wprowadzanie informacji dla wszystkich uczniów.

Podsumowanie

Dzięki strukturom możesz grupować dane w bardziej elastyczny sposób niż w przypadku tablic. Twoje struktury mogą zawierać członków różnych typów danych. Struktury można zainicjować w czasie deklaracji lub podczas programu za pomocą operatora kropki. Struktury stają się jeszcze potężniejsze, gdy deklarujesz tablice zmiennych struktur. Rozdział 29, „Tablice struktur”, pokazuje, jak zadeklarować kilka zmiennych struktur bez nadawania im każdej innej nazwy. Umożliwia to szybsze przechodzenie między strukturami za pomocą konstrukcji pętli.

Tablice struktur

Ta częśćł opiera się na poprzednim, pokazując, jak utworzyć wiele struktur danych. Po utworzeniu tablicy struktur można przechowywać wiele wystąpień wartości danych. Tablice struktur nadają się do przechowywania pełnego pliku pracownika, pliku zapasów lub dowolnego innego zestawu danych, który pasuje do formatu struktury. Podczas gdy tablice zapewniają wygodny sposób przechowywania kilku wartości tego samego typu, tablice struktur przechowują kilka wartości różnych typów razem, pogrupowanych jako struktury.

Tu przedstawiono następujące pojęcia:

♦ Tworzenie tablic struktur

♦ Inicjowanie tablic struktur

♦ Odwoływanie się do elementów z tablicy struktur

♦ Tablice jako składowe

Wielu programistów C ++ używa tablic struktur jako preludium do przechowywania swoich danych w pliku dyskowym. Możesz wprowadzić i obliczyć dane na dysku w tablicach struktur, a następnie zapisać te struktury w pamięci. Tablice struktur zapewniają również sposób przechowywania danych i odczyt z dysku.

Deklarowanie tablic struktur

Łatwo jest zadeklarować tablicę struktur. Podaj liczbę zarezerwowanych struktur w nawiasach tablicowych, kiedy deklarujesz zmienną struktury. Rozważ następującą definicję struktury:

struct stores

{ int employees;

int registers;

double sales;

} store1, store2, store3, store4, store5;

Ta struktura nie powinna być trudna do zrozumienia, ponieważ w deklaracji struktury nie są używane nowe polecenia. Ta deklaracja struktury tworzy pięć zmiennych struktury.

Jeśli czwarty sklep zwiększył liczbę pracowników o trzy, możesz zaktualizować numer pracownika sklepu za pomocą następującego oświadczenia o przypisaniu:

store4.employees + = 3; // Dodaj trzy do tego sklepu

// liczba pracowników.

Załóżmy, że piąty sklep właśnie został otwarty i chcesz zainicjować jego członków danymi. Jeśli sklepy są siecią, a nowy sklep to podobnie jak jeden z pozostałych, możesz rozpocząć inicjowanie danych sklepu, przypisując każdemu z jego członków te same dane, co dane innego sklepu, na przykład:

store5 = store2; // Zdefiniuj wartości początkowe dla składowej store5.

Takie deklaracje struktur są odpowiednie dla niewielkiej liczby struktur, ale gdyby sklepy były siecią krajową, pięć zmiennych struktury nie wystarczyłoby. Załóżmy, że było 1000 sklepów. Nie chcesz

tworzyć 1000 różnych zmiennych sklepu i pracować z każdą z nich osobno. O wiele łatwiej byłoby stworzyć tablicę struktur sklepu. Rozważ następującą deklarację struktury:

struct stores

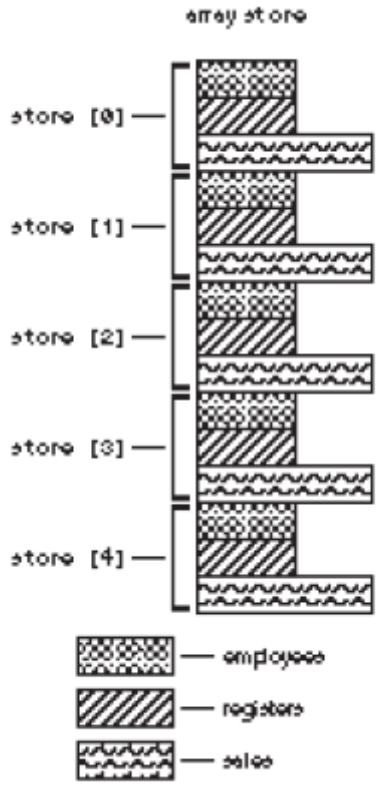
{ int employees;

int registers;

double sales;

} store[1000];

W jednej szybkiej deklaracji kod ten tworzy 1000 struktur sklepów, z których każda zawiera trzy elementy. Rysunek pokazuje, jak te zmienne strukturalne pojawiają się w pamięci. Zwróć uwagę na nazwę każdej zmiennej struktury: store [0], store [1], store [2] itd.



PRZESTROGA: Uważaj, aby w komputerze nie zabrakło pamięci podczas tworzenia dużej liczby struktur. Tablice struktur szybko zużywają cenną pamięć. Być może będziesz musiał utworzyć mniej struktur, przechowując więcej danych w plikach dyskowych i mniej danych w pamięci. store[2] jest elementem tablicowym. Ten element tablicy, w przeciwieństwie do innych, które widziałeś, jest zmienną strukturalną. Dlatego zawiera trzy elementy, z których każdy można odwoływać się do operatora kropki. Operator kropki działa w ten sam sposób dla elementów tablicy struktur , podobnie jak w przypadku regularnych zmiennych struktury. Jeśli liczba pracowników piątego sklepu (sklepu [4]) wzrośnie o trzy, możesz to zrobić aktualizując zmienną struktury w następujący sposób:

store[4].employees += 3;// Dodaj trzy do tego sklepu liczbę pracowników.

Możesz przypisywać sobie nawzajem całe struktury, również używając notacji tablicowej. Aby przypisać wszystkich członków 20 sklepu do 45 sklepu, wykonaj następujące czynności

store [44] = store [19]; // Skopiuj wszystkich członków z 20. sklep do 45.

Nadal obowiązują tutaj reguły tablic. Każdy element tablicy o nazwie store ma ten sam typ danych. Typ danych store to struktura stores Jak w przypadku każdej tablicy, każdy element musi być tego samego typu danych; nie można mieszać typów danych w tej samej tablicy. Typem danych tej tablicy jest utworzona przez Ciebie struktura zawierająca trzy elementy. Typ danych dla store[316] jest taki sam dla store[981] i store[74]. Nazwa tablicy, store, jest wskaźnikiem stałym do elementu początkowego tablicy, store[0]. Dlatego możesz użyć notacji wskaźnikowej do odniesienia do sklepów. Aby przypisać store[60] tę samą wartość, co store[23], możesz odwołać się do dwóch takich elementów:

* (store+ 60) = * (store + 23);

Możesz także łączyć notacje tablicowe i wskaźnikowe, takie jak

store [60] = * (store + 23);

i otrzymuj te same wyniki.

Możesz zwiększyć sprzedaż sklepu [8] o 40 procent, używając wskaźnika lub indeksu dolnego, jak również

store[8]. sales = (* (store + 8)). ssales * 1,40;

Wymagana jest dodatkowa para nawiasów, ponieważ operator kropki ma pierwszeństwo przed symbolem dereferencyjnym w hierarchii operatorów C ++

Oczywiście w tym przypadku kodowi nie pomaga notacja wskaźnika. Poniżej przedstawiono znacznie wyraźniejszy sposób na zwiększenie sprzedaży o 40 procent:

store[8].sales *= 1.40;

Poniższe przykłady budują system wprowadzania danych o zapasach dla firmy wysyłkowej za pomocą szeregu struktur. Jest bardzo mało nowych rzeczy, które musisz znać podczas pracy z tablicami struktur. Aby poczuć się komfortowo z tablicami zapisów struktur, skoncentruj się na notacji używanej podczas uzyskiwania dostępu do tablic struktur i ich członków.

Zachowaj prostą notację tablic

Nigdy nie uzyskasz dostępu do sprzedaży członków w ten sposób:

store.sales [8] = 3234,54; // Nieważne

Indeksy tablicy podążają tylko za elementami tablicy. sales nie jest tablicą; został zadeklarowany jako podwójna liczba zmiennoprzecinkowa. store nigdy nie można używać bez indeksu dolnego (chyba że używasz notacji wskaźnikowej). Oto poprawiona wersja poprzedniego oświadczenia o przypisaniu:

store[8].sales=3234.54; // Poprawne przypisanie

// wartości.

Przykłady

1. Załóżmy, że pracujesz dla firmy wysyłkowej, która sprzedaje dyski. Masz za zadanie napisać program śledzący dla 125 różnych dysków, które sprzedajesz. Musisz śledzić następujące informacje:

Pojemność w megabajtach

Czas dostępu w milisekundach

Kod dostawcy (A, B, C lub D)

Koszt

Cena £

Ponieważ w ekwipunku jest 125 różnych napędów dyskowych, dane ładnie pasują do tablicy struktur. Każdy element tablicy jest strukturą zawierającą pięć elementów opisanych na liście. Następująca definicja struktury definiuje spis:

```
struct inventory

{

long int storage;

int access_time;

char vendor_code;

double code;

double price;

} drive[125]; // Definiuje 125 wystąpień struktury
```

2. Podczas pracy z szerokim wachlarzem struktur, pierwszą troską powinno być to, w jaki sposób dane wprowadzane są do elementów tablicy. Najlepsza metoda wprowadzania danych zależy od aplikacji. Na przykład, jeśli konwertujesz ze starszego skomputeryzowanego systemu spisów, musisz napisać program do konwersji, który odczytuje plik spisu w jego rodzimym formacie i zapisuje go w nowym pliku w formacie wymaganym przez twoje programy C ++. To nie jest łatwe zadanie. Wymaga to dużej wiedzy na temat systemu, z którego dokonuje się konwersji. Jeśli piszesz skomputeryzowany system ewidencji po raz pierwszy, praca jest nieco łatwiejsza, ponieważ nie musisz konwertować starych plików. Nadal musisz zdać sobie sprawę, że ktoś musi wpisać dane do komputera. Musisz napisać program do wprowadzania danych, który odbiera każdy element ekwipunku z klawiatury i zapisuje go w pliku dyskowym. Powinieneś dać użytkownikowi szansę na edycję danych asortymentowych, aby poprawić dane, które pierwotnie wpisał nieprawidłowo. Jednym z powodów wprowadzenia plików dyskowych jest to, że formaty i struktury plików dyskowych mają wspólną więź. Gdy przechowujesz dane w strukturze lub częściej w szeregu struktur, możesz łatwo zapisać te dane do pliku dyskowego za pomocą prostych poleceń We / Wy dysku. Poniższy program pobiera tablicę struktur napędów dyskowych pokazaną w poprzednim przykładzie i dodaje funkcję wprowadzania danych, aby użytkownik mógł wprowadzać dane do tablicy struktur. Program jest sterowany z menu. Podczas uruchamiania programu użytkownik ma możliwość dodania danych, wydrukowania danych na ekranie lub wyjścia z programu. Ponieważ jeszcze nie widziałeś komend dyskowych we / wy, dane w tablicy struktur znikają po zakończeniu programu. Jak wspomniano wcześniej, zapisanie tych struktur na dysku jest łatwym zadaniem po nauczeniu się komend We / Wy dysku C ++. Na razie skoncentruj się na manipulacji strukturą. Ten program jest dłuższy niż wiele, które wcześniej widziałeś, ale jeśli śledziłeś dyskusje o strukturze i operatorze kropki, nie powinieneś mieć większych problemów z przestrzeganiem kodu. Zidentyfikuj program i dołącz niezbędne pliki nagłówkowe. Zdefiniuj strukturę, która opisuje format każdego elementu zapasów. Utwórz tablicę struktur o nazwie dysk. Wyświetl menu, które daje użytkownikowi wybór wprowadzania nowych danych inwentaryzacyjnych, wyświetlania danych na ekranie lub wychodzenia z programu. Jeśli użytkownik chce wprowadzić nowe

elementy zapasów, poproś użytkownika o każdy element i zapisz dane w tablicy struktur. Jeśli użytkownik chce zobaczyć ekwipunek, przejrzyj każdy element ekwipunku w tablicy, wyświetlając każdy na ekranie.

```
// Filename: C29DSINV.CPP

//Program do wprowadzania danych dla firmy zajmującej się napędami dysków.

#include <iostream.h>

#include <stdlib.h>

#include <iomanip.h>

#include <stdio.h>

struct inventory // Global structure definition.

{

long int storage;

int access_time;

char vendor_code;

float cost;

float price;

}; // No structure variables defined globally.

void disp_menu(void);

struct inventory enter_data();

void see_data(inventory disk[125], int num_items);

void main()

{

inventory disk[125]; // Local array of structures.

int ans;

int num_items=0; // Number of total items

// in the inventory.

do

{

do

{ disp_menu(); // Display menu of user choices.

cin >> ans; // Get user's request.

} while ((ans<1) || (ans>3));
```

```
switch (ans)
{ case (1): { disk[num_items] = enter_data(); // Enter
// disk data.
num_items++; // Increment number of items.
break; }
case (2): { see_data(disk, num_items); // Display
// disk data.
break; }
default : { break; }
}
} while (ans!=3); // Quit program
// when user is done.
return;
}
void disp_menu(void)
{
cout << "\n\n*** Disk Drive Inventory System ***\n\n";
cout << "Do you want to:\n\n";
cout << "\t1. Enter new item in inventory\n\n";
cout << "\t2. See inventory data\n\n";
cout << "\t3. Exit the program\n\n";
cout << "What is your choice? ";
return;
}
inventory enter_data()
{
inventory disk_item; // Local variable to fill
// with input.
cout << "\n\nWhat is the next drive's storage in bytes? ";
cin >> disk_item.storage;
cout << "What is the drive's access time in ms? ";
```

```
cin >> disk_item.access_time;
cout << “What is the drive’s vendor code (A, B, C, or D)? “;
fflush(stdin); // Discard input buffer
// before accepting character.
disk_item.vendor_code = getchar();
getchar(); // Discard carriage return
cout << “What is the drive’s cost? “;
cin >> disk_item.cost;
cout << “What is the drive’s price? “;
cin >> disk_item.price;
return (disk_item);
}
void see_data(inventory disk[125], int num_items)
{
int ctr;
cout << “\n\nHere is the inventory listing:\n\n”;
for (ctr=0;ctr<num_items;ctr++)
{
cout << “Storage: “ << disk[ctr].storage << “\t”;
cout << “Access time: “ << disk[ctr].access_time << “\n”;
cout << “Vendor code: “ << disk[ctr].vendor_code << “\t”;
cout << setprecision(2);
cout << “Cost: $” << disk[ctr].cost << “\t”;
cout << “Price: $” << disk[ctr].price << “\n”;
}
return;
}
```

Istnieje wiele funkcji i funkcji sprawdzania błędów, które można dodać, ale ten program jest podstawą bardziej kompleksowego systemu ewidencji. Możesz łatwo dostosować go do innego rodzaju ekwipunku, kolekcji taśm wideo, kolekcji monet lub dowolnego innego systemu śledzenia, zmieniając definicję struktury i nazwy członków w całym programie.

Tablice jako składowe

Składowe struktur mogą być tablicami. Składowe tablicy nie stwarzają żadnych nowych problemów, ale należy zachować ostrożność podczas uzyskiwania dostępu do poszczególnych elementów tablicy. Śledzenie tablic struktur zawierających elementy tablic może wydawać się dużym nakładem pracy z twojej strony, ale nie ma w tym nic. Rozważ następującą definicję struktury. To oświadczenie deklaruje tablicę 100 struktur, z których każda zawiera informacje o liście płac dla firmy. Dwie składowe, name i department, to tablice.

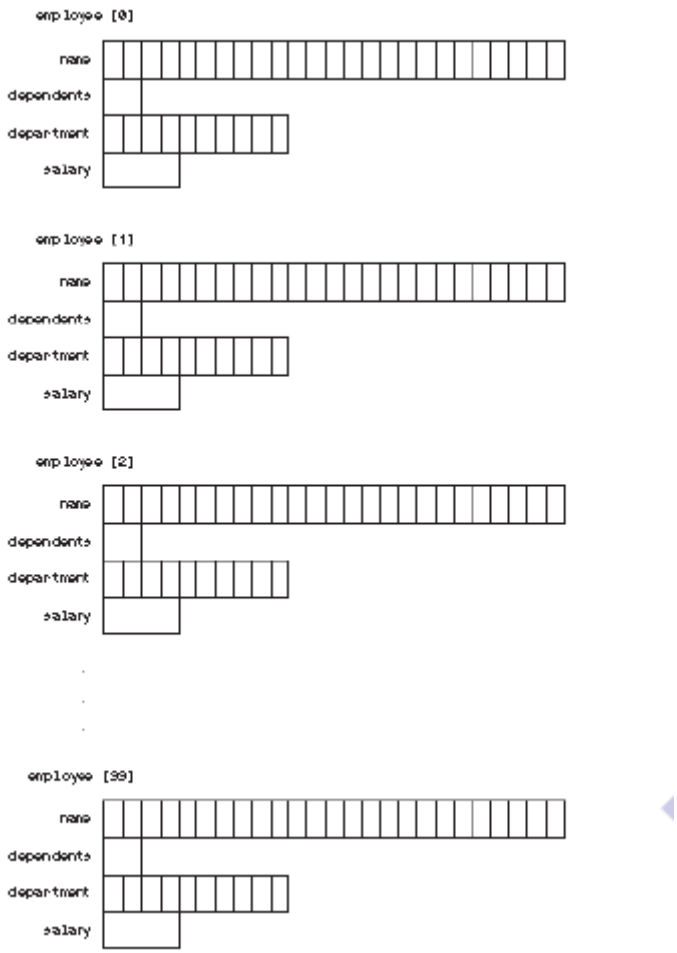
```
struct payroll

{ char name[25]; // Employee name array.

int dependents;

char department[10]; // Department name array.

float salary;

} employee[100]; // An array of 100 employees.
```

Rysunek pokazuje, jak te struktury wyglądają.

Pierwszy i trzeci element to tablice. name to tablica 25 znaków, a department to tablica 10 znaków. Załóżmy, że musisz zapisać inicjał 25. pracownika w zmiennej znakowej. Zakładając, że inicjał jest już zadeklarowany jako zmienna znakowa, następująca instrukcja przypisuje inicjał pracownika do zmiennej inicjał:

```
initial = employee[24].name[0];
```

Podwójne indeksy dolne mogą wyglądać na mylące, ale operator kropki wymaga zmiennej struktury po lewej stronie (pracownik [24]) i elementu po prawej stronie (pierwszy element tablicy nazwy). Możliwość odwoływania się do tablic składowych sprawia, że przetwarzanie danych znakowych w strukturach jest proste.



Przykłady

1. Załóżmy, że pracownica wyszła za mąż i chcesz zmienić jej nazwisko w pliku listy płac. (Zdarza się, że jest 45 pracownikiem w szeregu struktur.) Biorąc pod uwagę strukturę płac opisaną w poprzedniej sekcji, przypisałoby to nowej nazwie jej strukturze:

strcpy (pracownik [44] .name, „Mary Larson”); // Przypisz

// nowe nazwisko.

Gdy odwołujesz się do zmiennej struktury za pomocą operatora kropki, możesz używać zwykłych poleceń i funkcji do przetwarzania danych w elementach struktury.

2. Księgarnia chce skatalogować swój spis książek. Poniższy program tworzy tablicę 100 struktur. Każda struktura zawiera kilka rodzajów zmiennych, w tym tablice. Ten program jest częścią do wprowadzania danych w większym systemie zapasów. Przestudiuj odniesienia do członków, aby zobaczyć, w jaki sposób używane są tablice członków.

// Filename: C29BOOK.CPP

// Bookstore data-entry program.

#include <iostream.h>

#include <stdio.h>

```
#include <ctype.h>
struct inventory
{ char title[25]; // Book's title.
char pub_date[19]; // Publication date.
char author[20]; // Author's name.
int num; // Number in stock.
int on_order; // Number on order.
float retail; // Retail price.
};
void main()
{
inventory book[100];
int total=0; // Total books in inventory.
int ans;
do // This program enters data into the structures.
{ cout << "Book #" << (total+1) << ":\n", (total+1);
cout << "What is the title? ";
gets(book[total].title);
cout << "What is the publication date? ";
gets(book[total].pub_date);
cout << "Who is the author? ";
gets(book[total].author);
cout << "How many books of this title are there? ";
cin >> book[total].num;
cout << "How many are on order? ";
cin >> book[total].on_order;
cout << "What is the retail price? ";
cin >> book[total].retail;
fflush(stdin);
cout << "\nAre there more books? (Y/N) ";
ans=getchar();
```

```
fflush(stdin); // Discard carriage return.

ans=toupper(ans); // Convert to uppercase.

if (ans=='Y')

{ total++;

continue; }

} while (ans=='Y');

return;

}
```

Potrzebujesz znacznie więcej, aby uczynić z tego użyteczny program asortymentowy. Ćwiczenie na końcu tej częśći zaleca sposoby ulepszenia tego programu poprzez dodanie procedury drukowania oraz wyszukiwania tytułu i autora. Jedną z pierwszych rzeczy, które powinieneś zrobić, jest umieszczenie procedury wprowadzania danych w osobnej funkcji, aby kod był bardziej modułowy. Ponieważ ten przykład jest tak krótki i ponieważ program wykonuje tylko jedno zadanie (wprowadzanie danych), nie było żadnej korzyści z umieszczenia zadania wprowadzania danych w osobnej funkcji.

3. Oto wyczerpujący przykład kroków, które możesz wykonać, aby napisać program w C ++. Powinieneś zacząć rozumieć język C ++ na tyle, aby zacząć pisać zaawansowane programy. Załóżmy, że zostałeś zatrudniony przez lokalną księgarnię, aby napisać system inwentaryzacji czasopism. Musisz śledzić następujące elementy:

Tytuł czasopisma (maksymalnie 25 znaków)

Wydawca (maksymalnie 20 znaków)

Miesiąc (1, 2, 3, ... 12)

Rok publikacji

Liczba kopii w magazynie

Liczba kopii na zamówienie

Cena czasopisma (dolary i centy)

Załóżmy, że przewiduje się maksymalnie 1000 czasopism , które sklep będzie kiedykolwiek nosił. Oznacza to, że potrzebujesz 1000 wystąpień struktury, a nie 1000 magazynów ogółem. Oto dobra definicja struktury takiego spisu:

```
struct mag_info

{ char title[25];

char pub[25];

int month;

int year;

int stock_copies;

int order_copies;
```

```
float price;

} mags[1000]; // Define 1000 occurrences.
```

Ponieważ ten program składa się z więcej niż jednej funkcji, najlepiej jest deklarować strukturę globalnie, a zmienne struktury lokalnie w funkcjach, które ich potrzebują. Ten program potrzebuje trzech podstawowych funkcji: main() funkcji sterującej, funkcji wprowadzania danych i funkcji drukowania danych. Możesz dodać znacznie więcej, ale to dobry początek dla systemu zapasów. Aby długość tego przykładu była rozsądna, załóż, że użytkownik chce wprowadzić kilka czasopism, a następnie je wydrukować. (Aby uczynić program bardziej „użytecznym", należy dodać menu, aby użytkownik mógł kontrolować, kiedy on lub on dodaje i drukuje informacje, oraz powinien dodać więcej możliwości sprawdzania błędów i edycji). Oto przykład kompletnych danych -wejście i program do drukowania z prototypami. Tablice struktur są przekazywane między funkcjami z main ().

```
// Filename: C29MAG.CPP

// Magazine inventory program for adding and displaying

// a bookstore's magazines.

#include <iostream.h>

#include <ctype.h>

#include <stdio.h>

struct mag_info

{ char title[25];

char pub[25];

int month;

int year;

int stock_copies;

int order_copies;

float price;

};

mag_info fill_mags(struct mag_info mag);

void print_mags(struct mag_info mags[], int mag_ctr);

void main()

{

mag_info mags[1000];

int mag_ctr=0; // Number of magazine titles.

char ans;

do
```

```
{ // Assumes there is
// at least one magazine filled.
mags[mag_ctr] = fill_mags(mags[mag_ctr]);
cout << “Do you want to enter another magazine? “;
fflush(stdin);
ans = getchar();
fflush(stdin); // Discards carriage return.
if (toupper(ans) == ‘Y’)
{ mag_ctr++; }
} while (toupper(ans) == ‘Y’);
print_mags(mags, mag_ctr);
return; // Returns to operating system.
}
void print_mags(mag_info mags[], int mag_ctr)
{
int i;
for (i=0; i<=mag_ctr; i++)
{ cout << “\n\nMagazine “ << i+1 << “:\n”;// Adjusts for
// subscript.
cout << “\nTitle: “ << mags[i].title << “\n”;
cout << “\tPublisher: “ << mags[i].pub << “\n”;
cout << “\tPub. Month: “ << mags[i].month << “\n”;
cout << “\tPub. Year: “ << mags[i].year << “\n”;
cout << “\tIn-stock: “ << mags[i].stock_copies << “\n”;
cout << “\tOn order: “ << mags[i].order_copies << “\n”;
cout << “\tPrice: “ << mags[i].price << “\n”;
}
return;
}
mag_info fill_mags(mag_info mag)
{
```

```
puts(“\n\nWhat is the title? “);

gets(mag.title);

puts(“Who is the publisher? “);

gets(mag.pub);

puts(“What is the month (1, 2, ..., 12)? “);

cin >> mag.month;

puts(“What is the year? “);

cin >> mag.year;

puts(“How many copies in stock? “);

cin >> mag.stock_copies;

puts(“How many copies on order? “);

cin >> mag.order_copies;

puts(“How much is the magazine? “);

cin >> mag.price;

return (mag);

}
```

Ćwiczenia

1. Napisz program, który przechowuje tablicę nazwisk znajomych, numerów telefonów i adresów i drukuje je na dwa sposoby: z ich nazwiskiem, adresem i numerem telefonu, lub tylko z ich nazwiskiem i numerem telefonu na liście telefonów.

2. Dodaj funkcję sortowania do programu w ćwiczeniu 1, aby móc drukować nazwiska znajomych w kolejności alfabetycznej. (Wskazówka: musisz sprawić, by członek trzymający te nazwiska był wskaźnikiem postaci).

3. Rozwiń program do wprowadzania danych książek, C29BOOK.CPP, dodając funkcje ułatwiające korzystanie z niego (takie jak wyszukiwanie książki według autora, tytułu i drukowanie spisu książek na zamówienie).

Podsumowanie

Opanowałeś struktury i tablice struktur. Wiele przydatnych programów do inwentaryzacji i śledzenia można napisać przy użyciu struktur. Dzięki możliwości tworzenia tablic struktur możesz teraz utworzyć kilka wystąpień danych. Kolejnym krokiem w procesie uczenia się C ++ jest zapisanie tych struktur i innych danych w plikach dyskowych. Następne dwa rozdziały omawiają koncepcje przetwarzania plików dyskowych.

Pliki sekwencyjne

Do tej pory każdy przykład przetwarzał dane, które znajdowały się na liście programów lub pochodziły z klawiatury. Przypisałeś stałe i zmienne innym zmiennym i utworzyłeś nowe wartości danych z wyrażeń. Programy otrzymały również dane wejściowe za pomocą cin, gets() i funkcje wprowadzania znaków. Dane utworzone przez użytkownika i przypisane do zmiennych z instrukcjami przypisania są wystarczające dla niektórych aplikacji. Ze względu na duże ilości danych, które większość rzeczywistych aplikacji musi przetworzyć, potrzebujesz lepszego sposobu przechowywania tych danych. Dla wszystkich oprócz najmniejszych programów komputerowych rozwiązaniem są pliki dyskowe. Po zapisaniu danych na dysku komputer pomaga wprowadzać, znajdować, zmieniać i usuwać dane. Komputer i C++ to po prostu narzędzia do zarządzania danymi i ich przetwarzania. Ten rozdział koncentruje się na koncepcjach przetwarzania dysku i plików i uczy pierwszej z dwóch metod dostępu do dysku, sekwencyjnego dostępu do plików. W tej części przedstawiono następujące pojęcia:

♦ Przegląd plików dyskowych

♦ Rodzaje plików

♦ Przetwarzanie danych na dysku

♦ Sekwencyjny dostęp do plików

♦ Funkcje we / wy pliku

Po zakończeniu będziesz gotowy zająć się bardziej zaawansowanymi metodami dostępu do plików losowych opisanymi w następnych częściach. Jeśli zaprogramowałeś skomputeryzowane pliki danych w innym języku programowania, możesz być zaskoczony, jak C++ pożycza z innych języków programowania, zwłaszcza BASIC, podczas pracy z plikami dyskowymi. Jeśli dopiero zaczynasz przetwarzać pliki dyskowe, pliki dyskowe można łatwo tworzyć i czytać.

Dlaczego warto korzystać z dysku?

Typowy system komputerowy ma znacznie mniej pamięci niż dysk twardy. Dysk twardy zawiera znacznie więcej danych, niż zmieści się w pamięci RAM komputera. Jest to główny powód używania dysku do przechowywania danych. Pamięć dyskowa, ponieważ jest nieulotna, również trwa dłużej; po wyłączeniu komputera pamięć dysku nie jest usuwana, a pamięć RAM jest usuwana. Ponadto, gdy zmieniają się Twoje dane, Ty (lub, co ważniejsze, użytkownicy), nie musisz edytować programu i szukać zestawu instrukcji przypisania. Zamiast tego użytkownicy uruchamiają wcześniej napisane programy, które wprowadzają zmiany w danych na dysku. Utrudnia to na początku programowanie, ponieważ trzeba pisać programy, aby zmienić dane na dysku. Jednak osoby nieprogramujące mogą wtedy używać programów i modyfikować dane bez znajomości C++.

Pojemność dysku sprawia, że jest to idealne miejsce do przechowywania danych oraz programów. Zastanów się, co by się stało, gdyby wszystkie dane musiały być przechowywane z instrukcjami przypisania programu. Co się stanie, jeśli Urząd Ubezpieczeń Społecznych w Waszyngtonie poprosi cię o napisanie programu C++ do obliczania, uśredniania, filtrowania, sortowania i drukowania nazwiska i adresu każdej osoby w jego plikach? Czy chcesz, aby Twój program zawierał miliony instrukcji przypisania? Nie tylko nie chcesz, aby program przechowywał tak dużo danych, ale nie mógł tego zrobić, ponieważ tylko stosunkowo niewielkie ilości danych mieszczą się w programie, zanim zabraknie pamięci RAM. Przechowywanie danych na dysku jest znacznie mniej ograniczone, ponieważ masz więcej miejsca. Twój dysk może pomieścić tyle danych, ile masz miejsca na dysku. Ponadto, jeśli

wymagania programu wzrosną, zwykle możesz zwiększyć ilość miejsca na dysku, podczas gdy nie zawsze możesz dodać więcej pamięci RAM do swojego komputera.

UWAGA: C++ nie ma dostępu do specjalnej rozszerzonej lub rozszerzonej pamięci, którą mają niektóre komputery.

Podczas pracy z plikami dyskowymi C++ nie musi uzyskiwać dostępu do dużej ilości pamięci RAM, ponieważ C++ odczytuje dane z dysku i przetwarza tylko części danych naraz. Nie wszystkie dane na dysku muszą znajdować się w pamięci RAM, aby C++ mógł je przetworzyć. C++ odczytuje niektóre dane, przetwarza je, a następnie czyta więcej. Jeśli C++ wymaga danych na dysku po raz drugi, ponownie odczytuje to miejsce na dysku.

Rodzaje dostępu do plików na dysku

Twoje programy mogą uzyskiwać dostęp do plików na dwa sposoby: poprzez dostęp sekwencyjny lub dostęp losowy. Twoja aplikacja określa metodę, którą powinieneś wybrać. Tryb dostępu do pliku określa sposób odczytu, zapisu, zmiany i usuwania danych z pliku. Dostęp do niektórych plików można uzyskać na dwa sposoby, sekwencyjnie i losowo, o ile programy są poprawnie zapisane, a dane umożliwiają dostęp do obu typów plików. Dostęp do pliku sekwencyjnego należy uzyskać w tej samej kolejności, w jakiej plik został zapisany. Jest to analogiczne do taśm kasetowych: muzyka odtwarzana jest w tej samej kolejności, w jakiej został nagrany. (Możesz szybko przewijać utwory do przodu lub do tyłu, których nie chcesz słuchać, ale ich kolejność decyduje o tym, co musisz zrobić, aby odtworzyć utwór, którego chcesz.) Trudno, a czasem nie, wstawianie danych do środek pliku ekwiwalentnego. Jak łatwo wstawić nowy utwór w środku dwóch innych utworów na taśmie? Jedynym sposobem, aby naprawdę dodać lub usunąć rekordy ze środka pliku sekwencyjnego, jest utworzenie całkowicie nowego pliku, który łączy zarówno stare, jak i nowe rekordy. Może się wydawać, że pliki sekwencyjne są ograniczone, ale okazuje się, że wiele aplikacji poddaje się przetwarzaniu plików sekwencyjnych.

W przeciwieństwie do plików sekwencyjnych możesz uzyskać dostęp do plików o swobodnym dostępie w dowolnej kolejności. Pomyśl o danych w pliku o swobodnym dostępie jak o utworach na płycie kompaktowej lub płycie; możesz przejść bezpośrednio do dowolnego utworu, który chcesz, bez konieczności odtwarzania lub przewijania do przodu innych utworów. Jeśli chcesz odtworzyć pierwszą piosenkę, szóstą, a następnie czwartą, możesz to zrobić. Kolejność odtwarzania nie ma nic wspólnego z kolejnością, w jakiej utwory zostały pierwotnie nagrane. Dostęp do plików losowych czasami wymaga więcej programowania, ale nagradza Twój wysiłek bardziej elastyczną metodą dostępu do plików.

Pojęcia dotyczące plików sekwencyjnych

Istnieją trzy operacje, które można wykonać na dysku sekwencyjnym z plikiem. Możesz

♦ Utworzyć pliki dyskowe

♦ Dodawać do plików na dysku

♦ Czytać z plików dyskowych

Twoja aplikacja określa, co musisz zrobić. Jeśli tworzysz plik dyskowy po raz pierwszy, musisz go utworzyć i zapisać w nim dane początkowe. Załóżmy, że chcesz utworzyć plik danych klienta. Utworzyłbyś nowy plik i zapisałbyś w nim swoich obecnych klientów. Dane klienta mogą pierwotnie znajdować się w tablicach, tablicach struktur, wskazywanych przez wskaźniki lub umieszczanych przez użytkownika w regularnych zmiennych. Z biegiem czasu, gdy baza klientów rośnie, możesz dodawać nowych klientów do pliku (nazywane dołączaniem do pliku). Gdy dodajesz na końcu pliku, dołączasz do tego pliku. Gdy Twoi klienci wchodzą do Twojego sklepu, odczytujesz ich informacje z pliku danych

klienta. Jednak przetwarzanie dysku klienta jest przykładem jednej z wad plików sekwencyjnych. Załóżmy, że klient się porusza i chce, abyś zmienił jego lub jej adres w swoich plikach. Pliki dostępu sekwencyjnego nie nadają się do zmiany przechowywanych w nich danych. Trudno jest również usunąć informacje z plików sekwencyjnych. Losowe pliki, , zapewniają znacznie łatwiejsze podejście do zmiany i usuwania danych. Podstawowym podejściem do zmiany lub usuwania danych z pliku dostępu sekwencyjnego jest utworzenie nowego, ze starego, ze zaktualizowanymi danymi. Ze względu na łatwość aktualizacji zapewnianą przez pliki o swobodnym dostępie, ta część koncentruje się na tworzeniu, czytaniu i dodawaniu do plików sekwencyjnych.

Otwieranie i zamykanie plików

Przed utworzeniem, zapisaniem lub odczytaniem pliku dyskowego należy go otworzyć. Jest to analogiczne do otwierania szafki na dokumenty przed rozpoczęciem pracy z plikiem przechowywanym w szafce. Gdy skończysz z plikiem szafki, zamkniesz szufladę z plikami. Po zakończeniu należy także zamknąć plik dyskowy. Po otwarciu pliku dyskowego musisz jedynie poinformować C++ o nazwie pliku i o tym, co chcesz zrobić (napisać, dodać lub odczytać z). C++ i system operacyjny współpracują ze sobą, aby upewnić się, że dysk jest gotowy i utworzyć pozycję w katalogu plików (jeśli tworzą plik) dla nazwy pliku. Po zamknięciu pliku C++ zapisuje wszystkie pozostałe dane do pliku, zwalnia plik z programu i aktualizuje katalog plików, aby odzwierciedlić nowy rozmiar pliku.

PRZESTROGA: Musisz upewnić się, że instrukcja FILES = w pliku CONFIG.SYS jest wystarczająco duża, aby pomieścić maksymalną liczbę otwartych plików dyskowych, przy czym jeden pozostanie dla programu C++. Jeśli nie masz pewności, jak to zrobić, zapoznaj się z instrukcją obsługi systemu DOS lub książką dla początkujących o systemie DOS.

Aby otworzyć plik, musisz wywołać funkcję open(). Aby zamknąć plik, wywołaj funkcję close(). Oto format tych dwóch wywołań funkcji:

file_ptr.open (nazwa pliku, dostęp);

i

file_ptr.close ();

file_ptr to specjalny typ wskaźnika, który wskazuje tylko pliki, a nie zmienne danych. Twój system operacyjny obsługuje dokładną lokalizację danych w pliku dyskowym. Nie chcesz się martwić o dokładny numer ścieżki i sektora danych na dysku. Dlatego pozwalasz aby file_ptr wskazywał dane, które odczytujesz i zapisujesz. Twój program musi jedynie ogólnie zarządzać file_ptr, podczas gdy C++ i twój system operacyjny zajmują się lokalizowaniem rzeczywistych danych fizycznych. nazwa_pliku to ciąg (lub wskaźnik znakowy wskazujący na ciąg) zawierający prawidłową nazwę pliku dla twojego komputera. nazwa_pliku może zawierać pełną ścieżkę do dysku i katalogu. Możesz podać nazwę pliku wielkimi lub małymi literami. access musi być jedną z wartości

Tryb: Opis

app: otwórz plik do dołączenia (dodania do niego).

ate: do końca pliku po otwarciu.

in: Otwórz plik do odczytu.

out: otwórz plik do zapisu.

binarny: Otwórz plik w trybie binarnym.

trunc: odrzuć zawartość, jeśli plik istnieje

nocreate: jeśli plik nie istnieje, otwieranie kończy się niepowodzeniem.

noreplace: Jeśli plik istnieje, otwieranie kończy się niepowodzeniem, chyba że dołączanie lub próba zakończenia pliku przy otwieraniu.

Domyślny tryb dostępu do pliku to tryb tekstowy. Plik tekstowy to plik ASCII, zgodny z większością innych języków programowania i aplikacjami. Pliki tekstowe nie zawsze zawierają tekst w znaczeniu tego słowa. Wszelkie dane, które musisz przechowywać, mogą znajdować się w pliku tekstowym. Programy odczytujące pliki ASCII mogą odczytywać dane utworzone jako pliki tekstowe C++. Omówienie dostępu do plików binarnych znajduje się w poniższym polu.

Tryby binarne

Jeśli określisz dostęp binarny, C++ utworzy lub odczyta plik w formacie binarnym. Pliki danych binarnych są „ściśnięte" - zajmują mniej miejsca niż pliki tekstowe. Wadą korzystania z plików binarnych jest to, że inne programy nie zawsze mogą odczytać pliki danych. Tylko programy C++ napisane w celu uzyskania dostępu do plików binarnych mogą je odczytywać i zapisywać. Zaletą plików binarnych jest oszczędność miejsca na dysku, ponieważ pliki danych są bardziej kompaktowe. Poza trybem dostępu w funkcji open() nie używasz żadnych dodatkowych poleceń, aby uzyskać dostęp do plików binarnych w swoich programach C++. Format binarny to format pliku specyficzny dla systemu. Innymi słowy, nie wszystkie komputery mogą odczytać plik binarny utworzony na innym komputerze. Jeśli otworzysz plik do zapisu, C++ go utworzy. Jeśli plik o tej nazwie już istnieje, C++ zastępuje stary plik bez ostrzeżenia. Podczas otwierania plików należy zachować ostrożność, aby nie zastąpić istniejących danych, które chcesz zapisać.

Jeśli podczas otwierania pliku wystąpi błąd, C++ nie tworzy poprawnego wskaźnika pliku. Zamiast tego C++ tworzy wskaźnik pliku równy zero. Na przykład, jeśli otworzysz plik do wyjścia, ale użyjesz niepoprawnej nazwy dysku, C++ nie będzie mógł otworzyć pliku i ustawi wskaźnik pliku na zero. Zawsze sprawdzaj wskaźnik pliku podczas pisania programów dyskowych, aby upewnić się, że plik został poprawnie otwarty.

WSKAZÓWKA: Początkujący programiści lubią otwierać wszystkie pliki na początku swoich programów i zamykać je na końcu. To nie zawsze jest najlepsza metoda. Otwórz pliki bezpośrednio przed uzyskaniem do nich dostępu i zamknij je natychmiast po zakończeniu. Ten nawyk chroni pliki, ponieważ są one zamykane natychmiast po ich zakończeniu. Zamknięty plik jest bardziej chroniony w przypadku niefortunnego (ale możliwego) przypadku awarii zasilania lub awarii komputera.

Ta sekcja zawiera wiele informacji na temat teorii dostępu do plików. Poniższe przykłady pomagają zilustrować te pojęcia.

Przykłady

1. Załóżmy, że chcesz utworzyć plik do przechowywania rekordów płatności domu za poprzedni rok. Oto kilka pierwszych wierszy programu, który tworzy na dysku plik HOUSE.DAT:

```
#include <fstream.h>

main()

{

ofstream file_ptr; // Declares a file pointer for writing
```

file_ptr.open(“house.dat”, ios::out); // Creates the file

Pozostała część programu zapisuje dane do pliku. Program nigdy więcej nie musi odwoływać się do nazwy pliku. Program korzysta ze zmiennej file_ptr w celu odniesienia do pliku. Przykłady w następnych kilku sekcjach ilustrują, w jaki sposób. Nie ma nic specjalnego w file_ptr, oprócz jego nazwy (chociaż nazwa ma w tym przypadku znaczenie). Możesz nazwać zmienne wskaźnika pliku XYZ lub a908973, jeśli chcesz, ale te nazwy nie będą miały znaczenia. Musisz dołączyć plik nagłówkowy fstream.h, ponieważ zawiera on definicję deklaracji ofstream i ifstream. Nie musisz się martwić o cechy fizyczne. File_ptr „wskazuje” na dane w pliku podczas jego zapisywania. Umieść deklaracje w swoich programach, w których deklarujesz inne zmienne i tablice.

WSKAZÓWKA: Ponieważ pliki nie są częścią twojego programu, może być przydatne deklarowanie wskaźników plików na całym świecie. W przeciwieństwie do danych w zmiennych rzadko istnieje powód, aby wskaźniki plików były lokalne.

Przed zakończeniem pracy z programem należy zamknąć plik. Następująca funkcja close () zamyka plik house:

file_ptr.close(); // Close the house payment file.

2. Jeśli chcesz, możesz wpisać pełną nazwę ścieżki w nazwie pliku. Poniższy plik otwiera domowy plik płatności w podkatalogu na dysku D:

file_ptr.open („d: \ mydata \ house.dat”, ios :: out);

3. Jeśli chcesz, możesz zapisać nazwę pliku w tablicy znaków lub wskazać ją wskaźnikiem. Każda z poniższych sekcji kodu jest równoważna:

char fn [] = „house.dat”; // Nazwa pliku w tablicy znaków.

file_ptr.open (fn, ios :: out); // Tworzy plik.

char * mój_plik = „house.dat”; // Wskazano nazwę pliku.

file_ptr.open (mój_plik, ios :: out); // Tworzy plik.

// Pozwól użytkownikowi wprowadzić nazwę pliku.

cout << „Jak nazywa się plik gospodarstwa domowego? „;

dostaje (nazwa pliku); // Nazwa pliku musi być tablicą lub

// wskaźnik znaków.

file_ptr.open (nazwa pliku, ios :: out); // Tworzy plik.

Bez względu na to, jak określisz nazwę pliku podczas otwierania pliku, zamknij plik wskaźnikiem pliku. Ta funkcja close () zamyka otwarty plik, bez względu na to, jakiej metody użyłeś do otwarcia pliku:

file_ptr.close (); // Zamknij plik płatności domu.

4. Powinieneś sprawdzić wartość zwracaną z open (), aby upewnić się, że plik został poprawnie otwarty. Oto kod po open (), który sprawdza błąd:

#include <fstream.h>

main()

{

ofstream file_ptr; // Declares a file pointer.

file_ptr.open(“house.dat”, ios::out); // Creates the file.

if (!file_ptr)

{ cout << “Error opening file.\n”; }

else

{

// Rest of output commands go here.

5. Możesz otworzyć i zapisać kilka plików w tym samym programie. Załóżmy, że chcesz odczytać dane z pliku listy płac i utworzyć zapasowy plik danych listy płac. Musisz otworzyć bieżący plik listy płac w trybie odczytu, a plik kopii zapasowej w trybie wyjścia. Dla każdego otwartego pliku w programie musisz zadeklarować inny wskaźnik pliku. Wskaźniki plików używane przez instrukcję wejściową i wyjściową określają, na którym pliku działają. Jeśli musisz otworzyć wiele plików, możesz zadeklarować tablicę wskaźników plików. Oto sposób na otwarcie dwóch plików listy płac:

#include <fstream.h>

ifstream file_in; // Input file

ofstream file_out; // Output file

main()

{

file_in.open(“payroll.dat”, ios::in); // Existing file

file_out.open(“payroll.BAK”, ios::out); // New file

When you finish with these files, be sure to close them with

these two close() function calls:

file_in.close();

file_out.close();

Zapis do pliku

Każda funkcja wejściowa lub wyjściowa wymagająca urządzenia wykonuje dane wejściowe i wyjściowe za pomocą plików. Widziałeś już większość z nich. Najpopularniejsze funkcje we / wy pliku

get() i put()

gets() i puts()

Możesz także użyć file_ptr, tak jak w przypadku cout lub cin. Poniższe wywołanie funkcji odczytuje trzy liczby całkowite z pliku wskazywanego przez file_ptr:

file_ptr >> num1 >> num2 >> num3; // Odczytuje trzy zmienne.

Zawsze istnieje więcej niż jeden sposób zapisywania danych w pliku dyskowym. Przez większość czasu będzie działać więcej niż jedna funkcja. Na przykład, jeśli napiszesz wiele nazw do pliku, zarówno puts(), jak i file_ptr << działają. Możesz także pisać nazwy używając put(). Powinieneś korzystać z dowolnej funkcji, z której czujesz się najlepiej. Jeśli chcesz znaku nowej linii (\ n) na końcu każdej linii w pliku, file_ptr << i put () są prawdopodobnie łatwiejsze niż put(), ale wszystkie trzy wykonają zadanie.

WSKAZÓWKA: Każda linia w pliku nazywa się rekordem. Umieszczając znak nowego wiersza na końcu rekordów pliku, ułatwia wprowadzanie tych rekordów.

Przykłady

1. Poniższy program tworzy plik o nazwie NAMES.DAT. Program zapisuje pięć nazw w pliku dyskowym za pomocą file_ptr <<.

```
// Filename: C30WR1.CPP

// Writes five names to a disk file.

#include <fstream.h>

ofstream fp;

void main()

{

fp.open("NAMES.DAT", ios::out); // Creates a new file.

fp << "Michael Langston\n";

fp << "Sally Redding\n";

fp << "Jane Kirk\n";

fp << "Stacy Wikert\n";

fp << "Joe Hiquet\n";

fp.close(); // Release the file.

return;

}
```

Aby uprościć ten pierwszy przykład, nie sprawdzono błędów w funkcji open (). Następne kilka przykładów sprawdza błąd.

NAMES.TXT to plik danych tekstowych. Jeśli chcesz, możesz wczytać ten plik do edytora tekstu (użyj polecenia edytora tekstu do odczytu plików ASCII) lub użyj polecenia MS-DOS TYPE (lub równoważnego polecenia systemu operacyjnego), aby wyświetlić ten plik na ekranie. Jeśli wyświetlisz NAMES.TXT, zobaczysz:

```
Michael Langston

Sally Redding

Jane Kirk

Stacy Wikert
```

Joe Hiquet

2. Poniższy plik zapisuje liczby od 1 do 100 w pliku o nazwie NUMS.1.

```
// Filename: C30WR2.CPP

// Writes 1 to 100 to a disk file.

#include <fstream.h>

ofstream fp;

void main()

{

int ctr;

fp.open("NUMS.1", ios::out); // Creates a new file.

if (!fp)

{ cout << "Error opening file.\n"; }

else

{

for (ctr = 1; ctr < 101; ctr++)

{ fp << ctr << " "; }

}

fp.close();

return;

}
```

Liczby nie są zapisywane po jednym w wierszu, ale ze spacją między każdą z nich. Format file_ptr << określa format danych wyjściowych. Podczas zapisywania danych w plikach dyskowych należy pamiętać, że dane należy odczytać później. Musisz użyć funkcji wejściowych „odbicie lustrzane", aby odczytać dane, które wyprowadzasz do plików.

Zapisanie do drukarki

Funkcje takie jak open() i inne nie zostały zaprojektowane do zapisu tylko do plików. Zostały zaprojektowane do pisania na dowolnym urządzeniu, w tym plikach, ekranie i drukarce. Jeśli musisz zapisać dane w drukarce, możesz traktować drukarkę jak plik. Poniższy program otwiera wskaźnik pliku przy użyciu nazwy MS-DOS dla drukarki znajdującej się na LPT1 (nazwa MS-DOS dla pierwszego portu drukarki równoległej):

```
// Filename: C30PRNT.CPP

// Prints to the printer device

#include <fstream.h>

ofstream prnt; // Points to the printer.
```

```
void main()

{

prnt.open(“LPT1”, ios::out);

prnt << “Printer line 1\n”; // 1st line printed.

prnt << “Printer line 2\n”; // 2nd line printed.

prnt << “Printer line 3\n”; // 3rd line printed.

prnt.close();

return;

}
```

Upewnij się, że drukarka jest włączona i ma papier przed uruchomieniem tego programu. Po uruchomieniu programu na drukarce jest drukowane:

Printer line 1

Printer line 2

Printer line 3

Dodawanie do pliku

Możesz łatwo dodawać dane do istniejącego pliku lub tworzyć nowe pliki, otwierając plik w trybie dostępu dołączania. Pliki danych na dysku rzadko są statyczne; rosną prawie codziennie z powodu (miejmy nadzieję!) zwiększonego biznesu. Możliwość dodawania danych już znajdujących się na dysku jest bardzo przydatna. Pliki otwierane w celu dołączenia do dołączenia (przy użyciu ios :: app) nie muszą istnieć. Jeśli plik istnieje, C++ dołącza dane na końcu pliku podczas ich zapisywania.

Jeśli plik nie istnieje, C++ tworzy plik (podobnie jak w przypadku otwierania pliku w celu uzyskania dostępu do zapisu).

Przykład

Poniższy program dodaje trzy kolejne nazwy do pliku NAMES.DAT utworzonego we wcześniejszym przykładzie.

```
// Filename: C30AP1.CPP

// Adds three names to a disk file.

#include <fstream.h>

ofstream fp;

void main()

{

fp.open(“NAMES.DAT”, ios::app); // Adds to file.

fp << “Johnny Smith\n”;

fp << “Laura Hull\n”;
```

```
fp << “Mark Brown\n”;

fp.close(); // Release the file.

return;

}
```

Oto jak teraz wygląda plik:

Michael Langston

Sally Redding

Jane Kirk

Stacy Wikert

Joe Hiquet

Johnny Smith

Laura Hull

Mark Brown

UWAGA: Jeśli plik nie istnieje, C++ go utworzy i zapisze w pliku trzy nazwy.

Zasadniczo wystarczy zmienić tryb dostępu funkcji open (), aby zmienić program do tworzenia plików w program do dołączania plików.

Czytanie z pliku

Gdy dane znajdą się w pliku, musisz być w stanie je odczytać. Musisz otworzyć plik w trybie dostępu do odczytu. Istnieje kilka sposobów odczytu danych. Możesz odczytywać dane znakowe jeden znak na raz lub jeden ciąg na raz. Wybór zależy od formatu danych. Pliki otwierane w celu uzyskania dostępu do odczytu (przy użyciu ios :: in) muszą już istnieć, lub C++ spowoduje błąd. Nie można odczytać pliku, który nie istnieje. open() zwraca zero, jeśli plik nie istnieje po otwarciu go do odczytu. Kolejne zdarzenie ma miejsce podczas odczytu plików. W końcu odczytasz wszystkie dane. Późniejszy odczyt powoduje błędy, ponieważ nie ma więcej danych do odczytania. C++ zapewnia rozwiązanie wystąpienia końca pliku. Jeśli spróbujesz odczytać z pliku, z którego całkowicie odczytałeś dane, C++ zwraca wartość zero. Aby znaleźć warunek końca pliku, sprawdź zerowanie podczas odczytywania informacji z plików.

Przykłady

1. Ten program prosi użytkownika o nazwę pliku i drukuje zawartość pliku na ekranie. Jeśli plik nie istnieje, program wyświetla komunikat o błędzie.

```
// Filename: C30RE1.CPP

// Reads and displays a file.

#include <fstream.h>

#include <stdlib.h>

ifstream fp;

void main()
```

```
{
char filename[12]; // Holds user's filename.
char in_char; // Input character.
cout << "What is the name of the file you want to see? ";
cin >> filename;
fp.open(filename, ios::in);
if (!fp)
{
cout << "\n\n*** That file does not exist ***\n";
exit(0); // Exit program.
}
while (fp.get(in_char))
{ cout << in_char; }
fp.close();
return;
}
```

Oto wynikowy wynik, gdy żądany jest plik NAMES.DAT:

```
Jaka jest nazwa pliku, który chcesz zobaczyć? NAMES.DAT
Michael Langston
Sally Redding
Jane Kirk
Stacy Wikert
Joe Hiquet
Johnny Smith
Laura Hull
Mark Brown
```

Ponieważ znaki nowej linii znajdują się w pliku na końcu każdej nazwy, nazwy pojawiają się na ekranie, po jednym w wierszu. Jeśli spróbujesz odczytać plik, który nie istnieje, program wyświetli następujący komunikat:

```
*** Ten plik nie istnieje ***
```

2. Ten program odczytuje jeden plik i kopiuje go do innego. Możesz użyć takiego programu do wykonania kopii zapasowej ważnych danych na wypadek uszkodzenia oryginalnego pliku. Program

musi otworzyć dwa pliki, pierwszy do odczytu, a drugi do zapisu. Wskaźnik pliku określa, który z dwóch plików jest otwierany.

```
// Filename: C30RE2.CPP
// Makes a copy of a file.
#include <fstream.h>
#include <stdlib.h>
ifstream in_fp;
ofstream out_fp;
void main()
{
char in_filename[12]; // Holds original filename.
char out_filename[12]; // Holds backup filename.
char in_char; // Input character.
cout << “What is the name of the file you want to back up?
“;
cin >> in_filename;
cout << “What is the name of the file “;
cout << “you want to copy “ << in_filename << “ to? “;
cin >> out_filename;
in_fp.open(in_filename, ios::in);
if (!in_fp)
{
cout << “\n\n*** “ << in_filename << “ does not exist
***\n”;
exit(0); // Exit program
}
out_fp.open(out_filename, ios::out);
if (!out_fp)
{
cout << “\n\n*** Error opening “ << in_filename << “
***\n”;
```

```
exit(0); // Exit program
}
cout << "\nCopying...\n"; // Waiting message.
while (in_fp.get(in_char))
{ out_fp.put(in_char); }
cout << "\nThe file is copied.\n";
in_fp.close();
out_fp.close();
return;
}
```

Ćwiczenia

1. Napisz program, który tworzy plik zawierający następujące dane:

Twoje imię

Twój adres

Twój numer telefonu

Twój wiek

2. Napisz drugi program, który odczyta i wydrukuje plik danych utworzony w ćwiczeniu 1.

3. Napisz program, który pobiera dane utworzone w ćwiczeniu 1 i zapisuje je na ekranie jedno słowo w wierszu.

4. Napisz program na komputery PC, który tworzy kopię zapasową dwóch ważnych plików:

AUTOEXEC.BAT i CONFIG.SYS. Wywołaj pliki kopii zapasowych AUTOEXEC.SAV i CONFIG.SAV.

5. Napisz program, który czyta plik i tworzy nowy plik z tymi samymi danymi, z wyjątkiem odwrócenia wielkości liter w drugim pliku. Wszędzie tam, gdzie w pierwszym pliku pojawiają się wielkie litery, pisz małe litery do nowego pliku i wszędzie tam, gdzie małe litery pojawiają się w pierwszym pliku, pisz wielkie litery do nowego pliku.

Podsumowanie

Możesz teraz wykonać jedno z najważniejszych wymagań przetwarzania danych: zapis i odczyt do i z plików dyskowych. Przedtem można było przechowywać dane tylko w zmiennych. Krótka żywotność zmiennych (trwają one tylko tak długo, jak działa Twój program) czyniło długotrwałe przechowywanie danych niemożliwe. Możesz teraz zapisywać duże ilości danych w plikach dyskowych do późniejszego przetworzenia. Odczytywanie i zapisywanie plików sekwencyjnych wymaga nauki większej liczby pojęć niż rzeczywistych poleceń lub funkcji. Funkcje open () i close () to najważniejsze funkcje, których nauczyłeś się w tym rozdziale. Jesteś już zaznajomiony z większością funkcji I / O potrzebnych do pobierania danych do iz plików dyskowych. Zaraz dowiesz się, jak tworzyć i wykorzystywać pliki o swobodnym dostępie. Programując z losowym dostępem do pliku, możesz odczytać wybrane dane z pliku, a także zmienić dane bez konieczności przepisywania całego pliku.

Pliki o swobodnym dostępie

Tu przedstawiono pojęcie losowego dostępu do plików. Losowy dostęp do plików umożliwia odczytywanie lub zapisywanie dowolnych danych z pliku dyskowego bez konieczności odczytywania lub zapisywania wszystkich danych przed nim. Możesz szybko wyszukiwać, dodawać, pobierać, zmieniać i usuwać informacje w pliku o swobodnym dostępie. Chociaż potrzebujesz kilku nowych funkcji do losowego dostępu do plików, okazuje się, że dodatkowy wysiłek opłaca się elastycznością, mocą i szybkością dostępu do dysku. Ta część wprowadza

♦ Pliki o swobodnym dostępie

♦ Rekordy plików

♦ Funkcja seekg ()

♦ Funkcje we / wy pliku specjalnego przeznaczenia

Dzięki sekwencyjnym i losowo dostępnym plikom C++ możesz robić wszystko, co chcesz zrobić z danymi na dysku.

Losowe zapisy plików

Losowe pliki pokazują moc przetwarzania danych w C++. Sekwencyjne przetwarzanie plików jest powolne, chyba że wczytasz cały plik do tablic i przetworzysz go w pamięci. Jak wyjaśniono w rozdziale 30, masz jednak znacznie więcej miejsca na dysku niż pamięć RAM, a większość plików na dysku nawet nie mieści się w pamięci RAM jednocześnie. Dlatego potrzebujesz sposobu szybkiego odczytu poszczególnych fragmentów danych z pliku w dowolnej kolejności i przetwarzania ich pojedynczo. Ogólnie rzecz biorąc, czytasz i zapisujesz rekordy plików. Rekord do pliku jest analogiczny do struktury C++. Rekord to zbiór jednej lub więcej wartości danych (zwanych polami), które odczytujesz i zapisujesz na dysku. Zasadniczo przechowujesz dane w strukturach i zapisujesz struktury na dysku, gdzie są one nazywane rekordami. Kiedy czytasz rekord z dysku, zwykle odczytujesz ten rekord do zmiennej struktury i przetwarzasz go za pomocą programu. W przeciwieństwie do większości języków programowania, nie wszystkie dane dyskowe dla programów C++ muszą być przechowywane w formacie rekordu. Zazwyczaj zapisujesz strumień znaków w pliku dyskowym i uzyskujesz dostęp do tych danych sekwencyjnie lub losowo, wczytując je do zmiennych i struktur. Proces losowego uzyskiwania dostępu do danych w pliku jest prosty. Pomyśl o plikach danych dużej organizacji kart kredytowych. Po dokonaniu zakupu sklep dzwoni do wystawcy karty kredytowej w celu uzyskania autoryzacji. Miliony nazwisk znajdują się w plikach firmy wydającej karty kredytowe. Firma kart kredytowych nie może szybko odczytać każdego rekordu sekwencyjnie z dysku znajdującego się przed twoim. Pliki sekwencyjne nie umożliwiają szybkiego dostępu. W wielu sytuacjach nie jest możliwe wyszukiwanie pojedynczych rekordów w pliku danych z dostępem sekwencyjnym. Firmy wydające karty kredytowe muszą korzystać z losowego dostępu do plików, aby ich komputery mogły przejść bezpośrednio do Twojej płyty, podobnie jak bezpośrednio do utworu na dysku CD lub albumie. Funkcje, których używasz, różnią się od funkcji sekwencyjnych, ale siła wynikająca z uczenia się dodanych funkcji jest warta wysiłku. Kiedy twój program odczytuje i zapisuje pliki losowo, traktuje plik jak dużą tablicę. Dzięki tablicom wiesz, że możesz dodawać, drukować lub usuwać wartości w dowolnej kolejności. Nie musisz zaczynać od pierwszego elementu tablicy, kolejno patrząc na następny, dopóki nie uzyskasz potrzebnego elementu. Możesz wyświetlić plik o swobodnym dostępie w ten sam sposób, uzyskując dostęp do danych w dowolnej kolejności. Większość losowych rekordów plików to rekordy o stałej długości. Każdy rekord (zwykle wiersz w pliku) zajmuje tyle samo miejsca na dysku. Większość plików sekwencyjnych, które czytałeś i pisałeś w poprzednich rozdziałach, były rekordami o zmiennej długości.

Podczas sekwencyjnego odczytu lub zapisu nie ma potrzeby zapisywania rekordów o stałej długości, ponieważ każda wartość wpisywana jest po jednym znaku, słowie, łańcuchu lub liczbie i szukasz potrzebnych danych. Dzięki rekordom o stałej długości komputer może lepiej obliczyć, gdzie na dysku znajduje się żądany rekord. Chociaż marnujesz trochę miejsca na dysku przy użyciu rekordów o stałej długości (ze względu na spacje wypełniające niektóre pola), zalety losowego dostępu do plików rekompensują „zmarnowane" miejsce na dysku (gdy dane nie wypełniają rozmiaru struktury) .

WSKAZÓWKA: Dzięki plikom o swobodnym dostępie możesz odczytywać lub zapisywać rekordy w dowolnej kolejności. Dlatego nawet jeśli chcesz wykonać sekwencyjny odczyt lub zapis pliku, możesz użyć przetwarzania o dostępie swobodnym i „losowo" odczytać lub zapisać plik w kolejności numerów rekordów sekwencyjnych.

Otwieranie plików o swobodnym dostępie

Podobnie jak w przypadku plików sekwencyjnych, musisz otwierać pliki o swobodnym dostępie przed ich odczytaniem lub zapisaniem. Możesz użyć dowolnego z trybów dostępu do odczytu wymienionych w rozdziale 30 (takich jak ios :: in) tylko do losowego odczytu pliku. Aby jednak zmodyfikować dane w pliku, należy otworzyć plik w jednym z trybów aktualizacji.

Tryb: Opis

app: otwórz plik do dołączenia (dodanie do niego)

ate: po otwarciu przejdź do końca pliku

in: Otwórz plik do odczytu

out: Otwórz plik do zapisu

binarny: Otwórz plik w trybie binarnym

trunc: odrzuć zawartość, jeśli plik istnieje

nocreate: jeśli plik nie istnieje, otwieranie kończy się niepowodzeniem

noreplace: Jeśli plik istnieje, otwieranie kończy się niepowodzeniem, chyba że dołączanie lub próba zakończenia pliku przy otwieraniu

W C++ tak naprawdę nie ma różnicy między plikami sekwencyjnymi a plikami losowymi. Różnica między plikami nie jest fizyczna, ale polega na metodzie dostępu do nich i ich aktualizacji.

Przykłady

1. Załóżmy, że chcesz napisać program do utworzenia pliku z nazwiskami znajomych. Wystarczy następujące wywołanie funkcji open (), zakładając, że fp jest deklarowany jako wskaźnik pliku:

fp.open("NAMES.DAT", ios::out);

if (!fp)

{ cout << "\n*** Cannot open file ***\n"; }

Tryb dostępu open update() nie jest potrzebny, jeśli tworzysz tylko plik. Co jednak, jeśli chcesz utworzyć plik, zapisać do niego nazwy i dać użytkownikowi szansę zmiany dowolnej z tych nazw przed zamknięciem pliku? Następnie musisz otworzyć plik w ten sposób:

fp.open("NAMES.DAT", ios::in | ios::out);

if (!fp)

cout << “\n*** Cannot open file ***\n”;

kod umożliwia utworzenie pliku, a następnie zmianę danych zapisanych w pliku.

2. Podobnie jak w przypadku plików sekwencyjnych, jedyną różnicą między użyciem binarnego trybu dostępu open() i trybu tekstowego jest to, że utworzony plik jest bardziej zwarty i oszczędza miejsce na dysku. Nie można jednak odczytać tego pliku z innych programów jako pliku tekstowego ASCII. Poprzednią funkcję open() można przepisać, aby utworzyć i umożliwić aktualizację pliku binarnego. Wszystkie pozostałe polecenia i funkcje filerelowane działają dla plików binarnych, podobnie jak w przypadku plików tekstowych.

fp.open(“NAMES.DAT”, ios::in | ios::out | ios::binary);

if (!fp)

cout << “\n*** Cannot open file ***\n”;

Funkcja seekg ()

C++ udostępnia funkcję, która umożliwia czytanie do określonego punktu w pliku danych o swobodnym dostępie. To jest funkcja seekg (). Format seekg () to

file_ptr.seekg (long_num, origin);

file_ptr to wskaźnik do pliku, do którego chcesz uzyskać dostęp, zainicjowany instrukcją open (). long_num to liczba bajtów w pliku, który chcesz pominąć. C++ nie czyta tylu bajtów, ale dosłownie pomija dane o liczbę bajtów określoną w long_num. Pomijanie bajtów na dysku jest znacznie szybsze niż ich odczytywanie. Jeśli long_num ma wartość ujemną, C++ przeskakuje do tyłu w pliku (pozwala to na ponowne odczytanie danych kilka razy). Ponieważ pliki danych mogą być duże, musisz zadeklarować long_num jako długą liczbę całkowitą, aby pomieścić dużą liczbę bajtów. origin to wartość, która mówi C++, od czego zacząć pomijanie bajtów określonych przez long_num. pochodzenie może być dowolną z trzech wartości

Opis: Pochodzenie: Ekwiwalent

Początek pliku: SEEK_SET: ios :: beg

Aktualna pozycja pliku: SEEK_CUR: ios :: cur

Koniec pliku: SEEK_END: ios :: end

Początki SEEK_SET, SEEK_CUR i SEEK_END są zdefiniowane w stdio.h. Odpowiedniki ios :: beg, ios :: cur i ios :: end są zdefiniowane w fstream.h.

UWAGA: W rzeczywistości wskaźnik pliku odgrywa znacznie ważniejszą rolę niż zwykłe „wskazywanie pliku” na dysku. Wskaźnik pliku stale wskazuje dokładną lokalizację następnego bajtu do odczytu lub zapisu. Innymi słowy, podczas odczytywania danych z pliku sekwencyjnego lub dostępu losowego wskaźnik pliku zwiększa się z każdym odczytanym bajtem. Używając seekg(), możesz przesunąć wskaźnik pliku do przodu lub do tyłu w pliku.

Przykłady

1. Bez względu na to, jak daleko w czytanym pliku znajduje się następująca funkcja seekg () ustawia wskaźnik pliku z powrotem na początku pliku:

fp.seekg (0L, SEEK_SET); // Ustaw wskaźnik pliku na początku.

Stała 0L przekazuje długą liczbę całkowitą 0 do funkcji seekg ().

Bez L C++ przekazuje zwykłą liczbę całkowitą, co nie odpowiada prototypowi seekg (), który znajduje się w fstream.h. Rozdział 4, „Zmienne i literały", wyjaśnił użycie sufiksów typu danych w stałych numerycznych, ale sufiksy nie były używane do tej pory.

Ta funkcja seekg() dosłownie czyta „przenieś wskaźnik pliku o 0 bajtów od początku pliku".

2. Poniższy przykład czyta plik o nazwie MYFILE.TXT dwa razy, raz, aby wysłać plik na ekran, a raz, aby wysłać plik do drukarki. Używane są trzy wskaźniki plików, po jednym dla każdego urządzenia (pliku, ekranu i drukarki).

```
// Filename: C31TWIC.CPP

// Writes a file to the printer, rereads it,

// and sends it to the screen.

#include <fstream.h>

#include <stdlib.h>

#include <stdio.h>

ifstream in_file; // Input file pointer.

ofstream scrn; // Screen pointer.

ofstream prnt; // Printer pointer.

void main()

{

char in_char;

in_file.open("MYFILE.TXT", ios::in);

if (!in_file)

{

cout << "\n*** Error opening MYFILE.TXT ***\n";

exit(0);

}

scrn.open("CON", ios::out); // Open screen device.

while (in_file.get(in_char))

{ scrn << in_char; } // Output characters to the screen.

scrn.close(); // Close screen because it is no

// longer needed.
```

```
in_file.seekg(0L, SEEK_SET); // Reposition file pointer.

prnt.open("LPT1", ios::out); // Open printer device.

while (in_file.get(in_char))

{ prnt << in_char; } // Output characters to the

// printer.

prnt.close(); // Always close all open files.

in_file.close();

return;

}
```

Możesz także zamknąć, a następnie ponownie otworzyć plik, aby ustawić wskaźnik pliku na początku, ale użycie seekg() jest bardziej wydajną metodą. Oczywiście można było używać zwykłych funkcji We / Wy do pisania na ekranie, zamiast konieczności otwierania ekranu jako oddzielnego urządzenia.

3. Następująca funkcja seekg() ustawia wskaźnik pliku na 30 bajcie w pliku. (Następny odczytany bajt to 31 bajt.)

```
file_ptr.seekg (30L, SEEK_SET); // Wskaźnik pozycji pliku

// na 30 bajcie.
```

Ta funkcja seekg() dosłownie czyta „przenieś wskaźnik pliku o 30 bajtów od początku pliku". Jeśli zapisujesz struktury w pliku, możesz szybko wyszukać dowolną strukturę w pliku za pomocą funkcji sizeof(). Załóżmy, że chcesz 123 wystąpienie struktury oznaczonej inventory. Wyszukiwałbyś za pomocą następującej funkcji seekg ():

```
file_ptr.seekg ((123L * sizeof (struct inventory)), SEEK_SET);
```

4. Poniższy program zapisuje litery alfabetu do pliku o nazwie ALPH.TXT. Funkcja seekg () jest następnie używana do odczytu i wyświetlania dziewiątej i siedemnastej litery (I i Q).

```
// Filename: C31ALPH.CPP

// Stores the alphabet in a file, then reads

// two letters from it.

#include <fstream.h>

#include <stdlib.h>

#include <stdio.h>

fstream fp;

void main()

{

char ch; // Holds A through Z.

// Open in update mode so you can read file after writing to it.
```

```
fp.open(“alph.txt”, ios::in | ios::out);

if (!fp)

{

cout << “\n*** Error opening file ***\n”;

exit(0);

}

for (ch = ‘A’; ch <= ‘Z’; ch++)

{ fp << ch; } // Write letters.

fp.seekg(8L, ios::beg); // Skip eight letters, point to I.

fp >> ch;

cout << “The first character is “ << ch << “\n”;

fp.seekg(16L, ios::beg); // Skip 16 letters, point to Q.

fp >> ch;

cout << “The second character is “ << ch << “\n”;

fp.close();

return;

}
```

5. Aby wskazać koniec pliku danych, możesz użyć funkcji seekg(), aby ustawić wskaźnik pliku na ostatnim bajcie. Kolejne seekg() powinny następnie użyć ujemnej wartości long_num, aby przeskoczyć do tyłu w pliku. Następująca funkcja seekg() powoduje, że wskaźnik pliku wskazuje na koniec pliku:

```
file_ptr.seekg (0L, SEEK_END); // Plik pozycji

// wskaźnika na końcu.
```

Ta funkcja seekg() dosłownie czyta „przenieś wskaźnik pliku o 0 bajtów z końca pliku”. Wskaźnik pliku wskazuje teraz znacznik końca pliku, ale można przeszukiwać wstecz (), aby znaleźć inne dane w pliku.

6. Poniższy program odczytuje plik ALPH.TXT (utworzony w ćwiczeniu 4) do tyłu, drukując każdy znak, gdy przeskakuje z powrotem do pliku.

```
// Filename: C31BACK.CPP

// Reads and prints a file backwards.

#include <fstream.h>

#include <stdlib.h>

#include <stdio.h>

ifstream fp;
```

```
void main()
{
int ctr; // Steps through the 26 letters in the file.
char in_char;
fp.open("ALPH.TXT", ios::in);
if (!fp)
{
cout << "\n*** Error opening file ***\n";
exit(0);
}
fp.seekg(-1L, SEEK_END); // Point to last byte in
// the file.
for (ctr = 0; ctr < 26; ctr++)
{
fp >> in_char;
fp.seekg(-2L, SEEK_CUR);
cout << in_char;
}
fp.close();
return;
}
```

Ten program używa również wartości początkowej SEEK_CUR. Ostatnia funkcja seekg() w programie szuka dwóch bajtów wstecz od bieżącej pozycji, a nie początku ani końca, jak w poprzednich przykładach. Pętla for pod koniec programu wykonuje metodę „przeskocz dwa bajty wstecz, przeczytaj jeden bajt dalej", aby przeskoczyć plik do tyłu.

7. Poniższy program wykonuje te same czynności, co w przykładzie 4 (C31ALPH.CPP), z jednym dodatkiem. Kiedy zostaną znalezione litery I i Q, litera x jest zapisywana nad I i Q. Funkcja seekg () musi zostać użyta do utworzenia kopii zapasowej jednego bajtu w pliku, aby zastąpić właśnie przeczytaną literę.

```
// Filename: C31CHANG.CPP
// Stores the alphabet in a file, reads two letters from it,
// and changes those letters to xs.
#include <fstream.h>
```

```
#include <stdlib.h>
#include <stdio.h>
fstream fp;
void main()
{
char ch; // Holds A through Z.
// Open in update mode so you can read file after writing to it.
fp.open("alph.txt", ios::in | ios::out);
if (!fp)
{
cout << "\n*** Error opening file ***\n";
exit(0);
}
for (ch = 'A'; ch <= 'Z'; ch++)
{ fp << ch; } // Write letters
fp.seekg(8L, SEEK_SET); // Skip eight letters, point to I.
fp >> ch;
// Change the Q to an x.
fp.seekg(-1L, SEEK_CUR);
fp << 'x';
cout << "The first character is " << ch << "\n";
fp.seekg(16L, SEEK_SET); // Skip 16 letters, point to Q.
fp >> ch;
cout << "The second character is " << ch << "\n";
// Change the Q to an x.
fp.seekg(-1L, SEEK_CUR);
fp << 'x';
fp.close();
return;
}
```

Plik o nazwie ALPH.TXT wygląda teraz tak:

ABCDEFGHxJKLMNOPxRSTUVWXYZ

Ten program stanowi podstawę bardziej kompletnego programu do zarządzania plikami danych. Po opanowaniu funkcji seekg () i zaznajomieniu się z plikami danych dyskowych zaczniesz pisać programy przechowujące bardziej zaawansowane struktury danych i dostęp do nich. Użytkownik ma możliwość zmiany nazw i adresów znajdujących się już w pliku. Korzystając z dostępu losowego, program wyszukuje i zmienia wybrane dane bez przepisywania całego pliku dysku.

Inne przydatne funkcje we / wy

Dostępnych jest kilka dodatkowych funkcji dyskowych we / wy, które mogą okazać się przydatne. Są tu wymienione dla kompletności. Podczas wykonywania bardziej wydajnych operacji we / wy na dysku możesz znaleźć zastosowanie dla wielu z tych funkcji. Każda z tych funkcji jest prototypowana w pliku nagłówkowym fstream.h.

♦ read (array, count): Odczytuje dane określone przez count do tablicy lub wskaźnika określonego przez array. read () nazywa się buforowaną funkcją We / Wy. read() umożliwia odczyt dużej ilości danych za pomocą jednego wywołania funkcji.

♦ write (array, count): Zapisuje liczbę bajtów tablicy do określonego pliku. write () to buforowana funkcja I / O. write() umożliwia zapisanie dużej ilości danych w jednym wywołaniu funkcji.

♦ usuń (nazwa pliku): usuwa plik nazwany nazwą pliku. Metoda remove () zwraca wartość 0, jeśli plik został pomyślnie usunięty, a wartość -1, jeśli wystąpił błąd.

Wiele z tych (i innych wbudowanych funkcji we / wy, których uczysz się w swojej karierze programistycznej w C++) to przydatne funkcje, które można powielić, korzystając z tego, co już wiesz. Buforowane funkcje plików we / wy umożliwiają odczytywanie i zapisywanie całych tablic (w tym tablic struktur) na dysku w jednym wywołaniu funkcji.

Przykłady

1. Następujący program żąda nazwy pliku od użytkownika i usuwa plik z dysku za pomocą funkcji remove().

```
// Filename: C31ERAS.CPP

// Erases the file specified by the user.

#include <stdio.h>

#include <iostream.h>

void main()

{

char filename[12];

cout << “What is the filename you want me to erase? “;

cin >> filename;

if (remove(filename) == -1)

{ cout << “\n*** I could not remove the file ***\n”; }
```

```
else

{ cout << "\nThe file " << filename << " is now removed\n";}

return;

}
```

2. Następująca funkcja jest częścią większego programu, który odbiera dane inwentarza w szeregu struktur od użytkownika. Do tej funkcji przekazywana jest nazwa tablicy i liczba elementów (zmiennych struktury) w tablicy. Funkcja write() następnie zapisuje pełny zestaw struktur do pliku dyskowego wskazanego przez fp.

```
void write_str(inventory items[ ], int inv_cnt)

{

fp.write(items, inv_cnt * sizeof(inventory);

return;

}
```

Gdyby tablica ekwipunku miała 1000 elementów, ta funkcja jednowierszowa nadal zapisywałaby całą tablicę do pliku dyskowego. Możesz użyć funkcji read (), aby odczytać całą tablicę struktur z dysku w jednym wywołaniu funkcji.

Ćwiczenia

1. Napisz program, który poprosi użytkownika o listę pięciu nazw, a następnie zapisze je w pliku. Przewiń plik i wyświetl jego zawartość na ekranie za pomocą funkcji seekg() i get().

2. Przepisz program w ćwiczeniu 1, aby wyświetlał co drugi znak w pliku nazw.

3. Napisz program, który odczytuje znaki z pliku. Jeśli wprowadzany znak jest małą literą, zmień go na wielką. Jeśli… znak wejściowy jest wielką literą, zmień go na małe litery. Nie zmieniaj innych znaków w pliku.

4. Napisz program, który wyświetla liczbę znaków niealfabetycznych w pliku.

5. Napisz program oceniania dla nauczyciela. Pozwól nauczycielowi wprowadzić do 10 ocen uczniów. Każdy student ma trzy oceny z semestru. Przechowuj nazwiska uczniów i ich trzy stopnie w szeregu struktur i przechowuj dane na dysku. Ustaw program za pomocą menu. Uwzględnij opcje dodawania większej liczby uczniów, przeglądania danych pliku lub drukowania ocen na drukarce z obliczoną klasą średnią

Podsumowanie

C++ obsługuje pliki o swobodnym dostępie z kilkoma funkcjami. Funkcje te obejmują sprawdzanie błędów, pozycjonowanie wskaźnika plików oraz otwieranie i zamykanie plików. Masz teraz narzędzia, które musisz zapisać dane programu C++ na dysk w celu przechowywania i wyszukiwania. Aplikacja listy mailingowej w Dodatku F oferuje kompletny przykład manipulacji plikami o swobodnym dostępie. Program umożliwia wprowadzanie nazw i adresów, przechowywanie ich na dysku, edycję, zmianę i drukowanie z pliku dyskowego.

Wprowadzenie do programowania obiektowego

Najpopularniejszym obecnie obiektowym językiem programowania jest C++. C++zapewnia klasy - które są jego obiektami. Klasy naprawdę odróżniają C++od C. W rzeczywistości, zanim powstała nazwa C++, język C++nosił nazwę „C z klasami”. Ta część ma na celu przybliżyć Cię do świata programowania obiektowego, często nazywanego OOP. Na tych kilku krótkich stronach prawdopodobnie nie staniesz się mistrzem OOP, jednak jesteś gotowy, aby poszerzyć swoją wiedzę na temat C++. W tej sekcji przedstawiono następujące pojęcia:

♦ Klasy C++

♦ Funkcje składowe

♦ Konstruktory

♦ Destruktory

Część kończy wprowadzenie do języka C++.

Co to jest klasa?

Klasa to zdefiniowany przez użytkownika typ danych, który przypomina strukturę. Klasa może mieć członków danych, ale w przeciwieństwie do struktur, które widziałeś do tej pory, klasy mogą również mieć funkcje członków. Członkowie danych mogą być dowolnego typu, niezależnie od tego, czy są zdefiniowani przez język, czy przez ciebie. Funkcje składowe mogą manipulować danymi, tworzyć i niszczyć zmienne klas, a nawet redefiniować operatory C++, aby działały na obiekty klasy. Klasy mają kilka typów elementów, ale wszystkie dzielą się na dwie kategorie: elementy danych i funkcje elementów.

Składowe danych

Składowe danych mogą być dowolnego typu. Oto prosta klasa:

```
// Klasa kuli.

class Sphere

{

public:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

};
```

Zwróć uwagę, jak ta klasa przypomina struktury, które już widziałeś, z wyjątkiem publicznego słowa kluczowego. Klasa Sphere ma czterech członków danych: r, x, y i z. W tym przypadku publiczne słowo kluczowe odgrywa ważną rolę; identyfikuje klasę Sfera jako strukturę. W rzeczywistości w C++klasa publiczna jest fizycznie identyczna ze strukturą. Na razie zignoruj publiczne słowo kluczowe; wyjaśniono to w dalszej części tego rozdziału.

Funkcje składowe

Klasa może również mieć funkcje składowe (elementy klasy, które manipulują elementami danych). Jest to jedna z głównych cech, która odróżnia klasę od struktury. Oto znowu klasa Sphere z dodanymi funkcjami składowymi:

```
#include <math.h>

const float PI = 3.14159;

// A sphere class.

class Sphere

{

public:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

Sphere(float xcoord, float ycoord, float zcoord, float radius)

{ x = xcoord; y = ycoord; z = zcoord; r = radius; }

~Sphere() { }

float volume()

{

return (r * r * r * 4 * PI / 3);

}

float surface_area()

{

return (r * r * 4 * PI);

}

};
```

Ta klasa Sphere ma cztery funkcje składowe: Sphere(), ~Sphere(), volume() i surface_area(). Klasa traci podobieństwo do struktury. Te funkcje składowe są bardzo krótkie. (Ten o dziwnej nazwie ~ Sphere () ,nie ma w nim kodu.) Gdyby kody funkcji składowych były znacznie dłuższe, tylko prototypy pojawiłyby się w klasie, a kod funkcji składowych pojawiłby się później w program. Programiści C++ nazywają obiekty danych klasy, ponieważ klasy nie tylko przechowują dane. Klasy działają na danych; w efekcie klasa jest obiektem, który sam się manipuluje. Wszystkie dane, które do tej pory widziałeś, to dane pasywne (dane zmanipulowane przez kod w programie). Funkcje członków klas faktycznie manipulują danymi klasy. W tym przykładzie członek klasy Sphere () jest funkcją specjalną. Jest to funkcja konstruktora, a jej nazwa musi zawsze być taka sama jak jej klasa. Jego podstawowym zastosowaniem jest zadeklarowanie nowej instancji klasy.

Przykłady

1. Poniższy program używa klasy Sphere () do zainicjowania zmiennej klasy (zwanej instancją klasy) i wydrukowania jej.

```
// Filename: C32CON.CPP

// Demonstrates use of a class constructor function.

#include <iostream.h>

const float PI = 3.14159; // Approximate value of pi.

// A sphere class.

class Sphere

{

public:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

Sphere(float xcoord, float ycoord,

float zcoord, float radius)

{ x = xcoord; y = ycoord; z = zcoord; r = radius; }

~Sphere() { }

float volume()

{

return (r * r * r * 4 * PI / 3);

}

float surface_area()

{

return (r * r * 4 * PI);

}

};

void main()

{

Sphere s(1.0, 2.0, 3.0, 4.0);

cout << “X = “ << s.x << “, Y = “ << s.y

<< “, Z = “ << s.z << “, R = “ << s.r << “\n”;

return;
```

}

Uwaga: W OOP funkcja main() (i wszystkie wywołania) staje się mniejsza, ponieważ funkcje składowe zawierają kod, który manipuluje wszystkimi danymi klas.

Rzeczywiście, ten program wygląda inaczej niż te, które widziałeś do tej pory. Ten przykład to Twoja pierwsza prawdziwa ekspozycja na programowanie OOP. Oto wynik tego programu:

X = 1, Y = 2, Z = 3, R = 4

Ten program ilustruje funkcję konstruktora Sphere(). Funkcja konstruktora jest jedyną funkcją składową wywoływaną przez program. Zwróć uwagę, że funkcja członka ~ Sphere() skonstruowała s i zainicjowała również swoje elementy danych.

Inną specjalną funkcją jest funkcja destruktor ~ Sphere (). Zauważ, że ma on również taką samą nazwę jak klasa, ale z tyldą (~) jako przedrostkiem. Funkcja destruktor nigdy nie przyjmuje argumentów i nigdy nie zwraca wartości. Zauważ też, że ten destruktor nic nie robi. Większość niszczycieli robi bardzo niewiele. Jeśli destruktor nie ma rzeczywistego celu, nie musisz go określać. Kiedy zmienna klasy wykracza poza zakres, pamięć przydzielona dla tej zmiennej klasy jest zwracana do systemu (innymi słowy następuje automatyczne zniszczenie). Programiści używają funkcji destruktora do zwolnienia pamięci zajmowanej przez dane klasy w zaawansowanych aplikacjach C++. Podobnie, jeśli konstruktor nie spełnia żadnej konkretnej funkcji, nie musisz jej deklarować. C++ przydziela pamięć dla zmiennej klasy, gdy definiujesz zmienną klasy, tak jak ma to miejsce w przypadku wszystkich innych zmiennych. Gdy dowiesz się więcej o programowaniu w C++, zwłaszcza gdy zaczniesz używać zaawansowanej koncepcji dynamicznego przydzielania pamięci, konstruktory i destruktory stają się bardziej przydatne.

2. Aby zilustrować, że wywoływany jest destruktor ~ Sphere () (po prostu nic nie robi), możesz wstawić do konstruktora instrukcję cout, jak pokazano w następnym programie:

```
// Filename: C32DES.CPP

// Demonstrates use of a class destructor function.

#include <iostream.h>

#include <math.h>

const float PI = 3.14159; // Approximate value of pi.

// A sphere class

class Sphere

{

public:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

Sphere(float xcoord, float ycoord,

float zcoord, float radius)

{ x = xcoord; y = ycoord; z = zcoord; r = radius; }
```

```
~Sphere()
{
cout << “Sphere (“ << x << “, “ << y
<< “, “ << z << “, “ << r << “) destroyed\n”;
}
float volume()
{
return (r * r * r * 4 * PI / 3);
}
float surface_area()
{
return (r * r * 4 * PI);
}
};
void main(void)
{
Sphere s(1.0, 2.0, 3.0, 4.0);
// Construct a class instance.
cout << “X = “ << s.x << “, Y = “
<< s.y << “, Z = “ << s.z << “, R = “ << s.r << “\n”;
return;
}
```

Oto wynik tego programu:

```
X = 1, Y = 2, Z = 3, R = 4
Sphere (1, 2, 3, 4) destroyed
```

Zauważ, że main() nie wywołał jawnie funkcji destruktora, ale ~Sphere() została wywołana automatycznie, gdy instancja klasy wykroczyła poza zakres.

3. Pozostałe funkcje składowe czekają na użycie. Poniższy program używa funkcji volume() i surface_area():

```
// Filename: C32MEM.CPP
// Demonstrates use of class member functions.
#include <iostream.h>
```

```
#include <math.h>
const float PI = 3.14159; // Approximate value of pi.
// A sphere class.
class Sphere
{
public:
float r; // Radius of sphere
float x, y, z; // Coordinates of sphere
Sphere(float xcoord, float ycoord,
float zcoord, float radius)
{ x = xcoord; y = ycoord; z = zcoord; r = radius; }
~Sphere()
{
cout << “Sphere (“ << x << “, “ << y
<< “, “ << z << “, “ << r << “) destroyed\n”;
}
float volume()
{
return (r * r * r * 4 * PI / 3);
}
float surface_area()
{
return (r * r * 4 * PI);
}
}; // End of class.
void main()
{
Sphere s(1.0, 2.0, 3.0, 4.0);
cout << “X = “ << s.x << “, Y = “ << s.y
<< “, Z = “ << s.z << “, R = “ << s.r << “\n”;
cout << “The volume is “ << s.volume() << “\n”;
```

```
cout << “The surface area is “

<< s.surface_area() << “\n”;

}
```

Funkcje volume() i surface_area() mogły zostać wprowadzone w linii. Oznacza to, że kompilator osadza funkcje w kodzie, zamiast wywoływać je jako funkcje. W C32MEM.CPP istnieje zasadniczo osobna funkcja, która jest wywoływana przy użyciu danych w Sphere(). Po ustawieniu go w linii Sphere() zasadniczo staje się makrem i jest rozwijany w kodzie.

4. W poniższym programie zmieniono funkcję volume() na funkcję liniową, tworząc bardziej wydajny program:

```
// Filename: C32MEM1.CPP

// Demonstrates use of in-line class member functions.

#include <iostream.h>

#include <math.h>

const float PI = 3.14159; // Approximate value of pi.

// A sphere class.

class Sphere

{

public:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

Sphere(float xcoord, float ycoord, float zcoord, float radius)

{ x = xcoord; y = ycoord; z = zcoord; r = radius; }

~Sphere()

{

cout << “Sphere (“ << x << “, “ << y

<< “, “ << z << “, “ << r << “) destroyed\n”;

}

inline float volume()

{

return (r * r * r * 4 * PI / 3);

}

float surface_area()
```

```
{
return (r * r * 4 * PI);
}
};
void main()
{
Sphere s(1.0, 2.0, 3.0, 4.0);
cout << “X = “ << s.x << “, Y = “ << s.y
<< “, Z = “ << s.z << “, R = “ << s.r << “\n”;
cout << “The volume is “ << s.volume() << “\n”;
cout << “The surface area is “ << s.surface_area() << “\n”;
}
```

Funkcje wbudowane rozwijają się, aby wyglądać tak w kompilatorze:

```
// C32MEM1A.CPP
// Demonstrates use of in-line class member functions.
#include <iostream.h>
#include <math.h>
const float PI = 3.14159; // Approximate value of pi.
// A sphere class
class Sphere
{
public:
float r; // Radius of sphere
float x, y, z; // Coordinates of sphere
Sphere(float xcoord, float ycoord, float zcoord, float radius)
{ x = xcoord; y = ycoord; z = zcoord; r = radius; }
~Sphere()
{
cout << “Sphere (“ << x << “, “ << y
<< “, “ << z << “, “ << r << “) destroyed\n”;
}
```

```
inline float volume()
{
return (r * r * r * 4 * PI / 3);
}
float surface_area()
{
return (r * r * 4 * PI);
}
};
void main()
{
Sphere s(1.0, 2.0, 3.0, 4.0);
cout << “X = “ << s.x << “, Y = “ << s.y
<< “, Z = “ << s.z << “, R = “ << s.r << “\n”;
cout << “The volume is “ << (s.r * s.r * s.r * 4 * PI / 3)
<< “\n”;
cout << “The surface area is “ << s.surface_area() << “\n”;
}
```

Zaletą korzystania z funkcji wbudowanych jest to, że wykonują się one szybciej - nie wiąże się to z narzutem wywołania funkcji, ponieważ w rzeczywistości żadna funkcja nie jest wywoływana. Wadą jest to, że jeśli twoje funkcje są często używane, twoje programy stają się coraz większe wraz z ich rozszerzaniem.

Domyślne argumenty składowe

Możesz także domyślnie podać argumenty funkcji członkowskich. Załóżmy domyślnie, że współrzędna y kuli będzie równa 2,0, współrzędna z będzie równa 2,5, a promień będzie równy 1,0. Przepisanie funkcji konstruktora z poprzedniego przykładu w celu wykonania tego powoduje, że kod:

```
Sphere(float xcoord, float ycoord = 2.0, float zcoord = 2.5,
float radius = 1.0)
{ x = xcoord; y = ycoord; z = zcoord; r = radius; }
```

You can create a sphere with the following instructions:

```
Sphere s(1.0); // Use all default
Sphere t(1.0, 1.1); // Override y coord
Sphere u(1.0, 1.1, 1.2); // Override y and z
```

Sphere v(1.0, 1.1, 1.2, 1.3); // Override all default

Przykłady

1. Domyślne argumenty są używane w następującym kodzie.

```
// Filename: C32DEF.CPP
// Demonstrates use of default arguments in
// class member functions.
#include <iostream.h>
#include <math.h>
const float PI = 3.14159; // Approximate value of pi.
// A sphere class.
class Sphere
{
public:
float r; // Radius of sphere
float x, y, z; // Coordinates of sphere
Sphere(float xcoord, float ycoord = 2.0,
float zcoord = 2.5, float radius = 1.0)
{ x = xcoord; y = ycoord; z = zcoord; r = radius; }
~Sphere()
{
cout << “Sphere (“ << x << “, “ << y
<< “, “ << z << “, “ << r << “) destroyed\n”;
}
inline float volume()
{
return (r * r * r * 4 * PI / 3);
}
float surface_area()
{
return (r * r * 4 * PI);
}
```

```
};
void main()
{
Sphere s(1.0); // use all default
Sphere t(1.0, 1.1); // override y coord
Sphere u(1.0, 1.1, 1.2); // override y and z
Sphere v(1.0, 1.1, 1.2, 1.3); // override all defaults
cout << “s: X = “ << s.x << “, Y = “ << s.y
<< “, Z = “ << s.z << “, R = “ << s.r << “\n”;
cout << “The volume of s is “ << s.volume() << “\n”;
cout << “The surface area of s is “ << s.surface_area() << “\n”;
cout << “t: X = “ << t.x << “, Y = “ << t.y
<< “, Z = “ << t.z << “, R = “ << t.r << “\n”;
cout << “The volume of t is “ << t.volume() << “\n”;
cout << “The surface area of t is “ << t.surface_area() << “\n”;
cout << “u: X = “ << u.x << “, Y = “ << u.y
<< “, Z = “ << u.z << “, R = “ << u.r << “\n”;
cout << “The volume of u is “ << u.volume() << “\n”;
cout << “The surface area of u is “ << u.surface_area() << “\n”;
cout << “v: X = “ << v.x << “, Y = “ << v.y
<< “, Z = “ << v.z << “, R = “ << v.r << “\n”;
cout << “The volume of v is “ << v.volume() << “\n”;
cout << “The surface area of v is “ << v.surface_area() << “\n”;
return;
}
```

Oto wynik tego programu:

Oto wynik tego programu:

s: X = 1, Y = 2, Z = 2,5, R = 1

Objętość s wynosi 4,188787

Pole powierzchni s wynosi 12.56636

t: X = 1, Y = 1,1, Z = 2,5, R = 1

Objętość t wynosi 4,188787

Pole powierzchni t wynosi 12,56636

u: X = 1, Y = 1,1, Z = 1,2, R = 1

Objętość u wynosi 4,188787

Pole powierzchni u wynosi 12.56636

v: X = 1, Y = 1,1, Z = 1,2, R = 1,3

Objętość v wynosi 9.202764

Pole powierzchni v wynosi 21,237148

Sfera (1, 1.1, 1.2, 1.3) zniszczona

Sfera (1, 1.1, 1.2, 1) zniszczona

Sfera (1, 1.1, 2.5, 1) zniszczona

Sfera (1, 2, 2,5, 1) zniszczona

Zauważ, że kiedy używasz wartości domyślnej, musisz także użyć innych wartości domyślnych po jej prawej stronie. Podobnie, gdy zdefiniujesz parametr funkcji jako mający wartość domyślną, każdy parametr po jego prawej stronie musi również mieć wartość domyślną.

2. Możesz także wywołać więcej niż jednego konstruktora; nazywa się to przeciążeniem konstruktora. Jeśli masz więcej niż jednego konstruktora, wszystkie o tej samej nazwie klasy, musisz nadać każdemu z nich inną listę parametrów, aby kompilator mógł określić, którego zamierzasz użyć. Powszechnym zastosowaniem przeciążonych konstruktorów jest tworzenie niezainicjowanego obiektu na odbierającym końcu zadania, jak widać tutaj:

```
// C32OVCON.CPP

// Demonstruje użycie przeciążonych konstruktorów.

#include <iostream.h>

#include <math.h>

const float PI = 3.14159; // Approximate value of pi.

// A sphere class.

class Sphere

{

public:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

Sphere() { /* doesn't do anything... */ }

Sphere(float xcoord, float ycoord,
```

```
float zcoord, float radius)
{ x = xcoord; y = ycoord; z = zcoord; r = radius; }
~Sphere()
{
cout << “Sphere (“ << x << “, “ << y
<< “, “ << z << “, “ << r << “) destroyed\n”;
}
inline float volume()
{
return (r * r * r * 4 * PI / 3);
}
float surface_area()
{
return (r * r * 4 * PI);
}
};
void main()
{
Sphere s(1.0, 2.0, 3.0, 4.0);
Sphere t; // No parameters (an uninitialized sphere).
cout << “X = “ << s.x << “, Y = “ << s.y
<< “, Z = “ << s.z << “, R = “ << s.r << “\n”;
t = s;
cout << “The volume of t is “ << t.volume() << “\n”;
cout << “The surface area of t is “ << t.surface_area()
<< “\n”;
return;
}
```

Widoczność składowej klasy

Przypomnijmy, że klasa Sphere() miała etykietę public. Zadeklarowanie etykiety publicznej jest konieczne, ponieważ domyślnie wszyscy członkowie klasy są prywatni. Członkowie prywatni nie mogą być dostępne tylko przez funkcję członka. Aby dane lub funkcje składowe mogły być używane przez

inne programy, muszą być jawnie zadeklarowane jako publiczne. W przypadku klasy Sphere() prawdopodobnie chcesz ukryć rzeczywiste dane przed innymi klasami. Chroni to integralność danych. Następny program dodaje funkcje cube() i square(), aby wykonać niektóre funkcje funkcji volume() i surface_area (). Nie ma potrzeby, aby inne funkcje korzystały z tych funkcji członkowskich.

```
// Nazwa pliku: C32VISIB.CPP

// Demonstruje użycie etykiet widoczności klasy.

#include <iostream.h>

#include <math.h>

const float PI = 3.14159; // Approximate value of pi.

// A sphere class.

class Sphere

{

private:

float r; // Radius of sphere

float x, y, z; // Coordinates of sphere

float cube() { return (r * r * r); }

float square() { return (r * r); }

public:

Sphere(float xcoord, float ycoord, float zcoord, float radius)

{ x = xcoord; y = ycoord; z = zcoord; r = radius; }

~Sphere()

{

cout << “Sphere (“ << x << “, “ << y

<< “, “ << z << “, “ << r << “) destroyed\n”;

}

float volume()

{

return (cube() * 4 * PI / 3);

}

float surface_area()

{

return (square() * 4 * PI);
```

```
}

};

void main()

{

Sphere s(1.0, 2.0, 3.0, 4.0);

cout << “The volume is “ << s.volume() << “\n”;

cout << “The surface area is “ << s.surface_area() << “\n”;

return;

}
```

Zauważ, że wiersz pokazujący elementy danych musiał zostać usunięty z main(). Członkowie danych nie są już bezpośrednio dostępni, z wyjątkiem funkcji członka klasy Sphere. Innymi słowy, main() nigdy nie może bezpośrednio manipulować elementami danych, takimi jak riz, nawet jeśli wywołuje funkcję konstruktora, która je utworzyła. Prywatne dane członka są widoczne tylko w funkcjach członka. To jest prawdziwa moc ukrywania danych; nawet twój własny kod nie może dostać się do danych! Zaletą jest zdefiniowanie określonych funkcji wyszukiwania danych, wyświetlania danych i zmieniających dane funkcji składowych, które main () musi wywoływać w celu manipulowania danymi składowymi. Za pomocą tych funkcji członkowskich konfigurujesz bufor między programem a strukturami danych programu. Jeśli zmienisz sposób przechowywania danych, nie musisz zmieniać funkcji main () ani żadnych funkcji wywoływanych przez main (). Musisz tylko zmienić funkcje składowe tej klasy.

Ćwiczenie

Skonstruuj klasę do przechowywania akt osobowych. Użyj następujących członków danych i zachowaj ich prywatność:

```
char name[25];

float salary;

char date_of_birth[9];
```

Utwórz dwa konstruktory, jeden do zainicjowania rekordu jego niezbędnymi wartościami, a drugi do utworzenia niezainicjowanego rekordu. Utwórz funkcje członkowskie, aby zmienić imię, wynagrodzenie i datę urodzenia osoby.

Podsumowanie

Poznałeś już klasy, typ danych, który odróżnia C++ od swojego poprzednika, C. Było to tylko pobieżne spojrzenie na programowanie obiektowe. Widziałeś jednak, że OOP oferuje zaawansowany widok twoich danych, łącząc dane z funkcjami członka, które manipulują tymi danymi.