

# Projektowanie aplikacji w MS Visual Studio

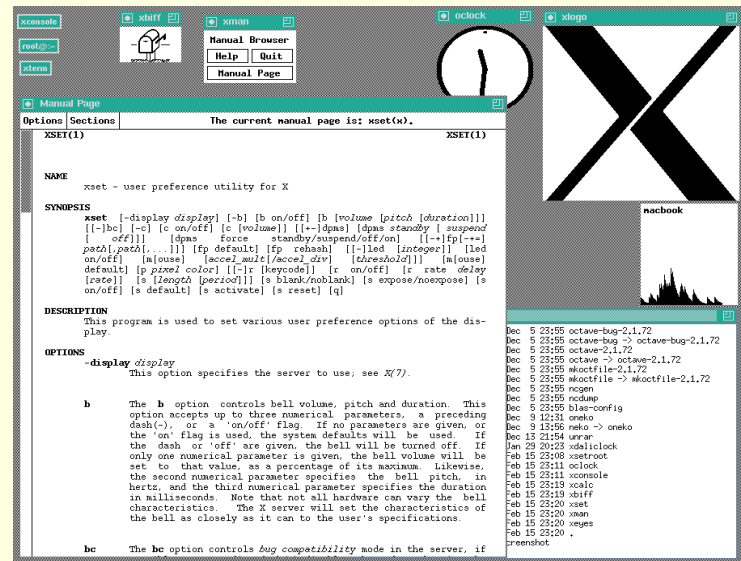
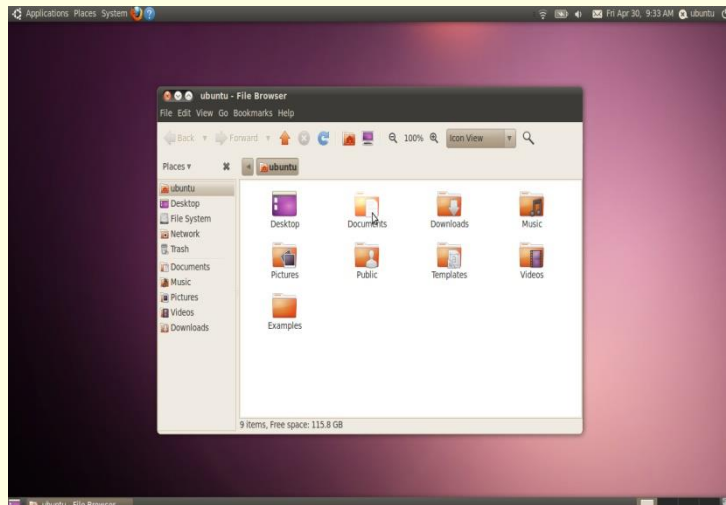
# Rys historyczny

Wszystkie języki programowania wyższego i niższego rzędu swoje działanie opierają na zasadach wprowadzonych przez systemy operacyjne. Dlatego, aby zrozumieć historię i budowę języka programowania C/C++ należy najpierw omówić systemy operacyjne, jakie stoją u podstaw rozwoju technik programistycznych.

Unix Time Sharing System, zwany w skrócie Unix, jest jednym z pierwszych systemów operacyjnych, jakie zostały wprowadzone do użytku powszechnego. Rok 1969 jest uznawany jako początek prac nad rozwojem tego systemu operacyjnego. Początkowo prace były prowadzone w Unix System Laboratories (USL) przez Dennisa Ritchie i Kena Thompsona. W latach 70 i 80 system ten zdobył jednak dużą popularność w środowiskach naukowych i inżynierskich. W efekcie rozpoczęto prace nad komercjalizacją systemu Unix. Zadania tego podjął się ośrodek Uniwersytetu Kalifornijskiego w Berkeley (ang. Berkeley Software Distribution - BSD). Zaowocowało to powstaniem wielu kolejnych odmian i implementacji. Najbardziej znane współczesnemu użytkownikowi pochodne systemu Unix to Linux oraz Mac OS.

W roku 1984 Richard Stallman zapoczątkował prace w ramach projektu Free Software Foundation (FSF). Celem prowadzonych prac było stworzenie bezpłatnie rozpowszechnianego systemu operacyjnego Unix, który nie tylko ujednoliciłby działania informatyków i programistów, ale również pozwalał na łatwe i bezpieczne zarządzanie zasobami danych. W tym celu utworzona została tzw. publiczna licencja GNU typu open source, która stała się podstawą otwartej współpracy pomiędzy programistami z różnych krajów. To właśnie na podstawie tej licencji Czytelnik może korzystać z różnego rodzaju oprogramowania udostępnianego bezpłatnie na różnych stronach internetowych. Sam system UNIX jest obecnie zarejestrowanym i chronionym prawnie znakiem towarowym The Open Group [[www.opengroup.org](http://www.opengroup.org)]. Unix miał ogromny wkład w rozwój wielu systemów operacyjnych. Twórcy tego systemu zastosowali podczas jego tworzenia wiele nowatorskich rozwiązań oraz założeń projektowych. Jego autorzy wprowadzili takie podstawowe dzisiaj pojęcia jak idea hierarchicznego systemu plików czy reprezentacja poszczególnych składników systemu w postaci plików (również dla urządzeń peryferyjnych).

Wbrew ówczesnym praktykom, kod Unix stworzono w języku wysokopoziomowym, dlatego jego późniejsza migracja na kolejne platformy sprzętowe była bardzo łatwa do wykonania. Pulpit użytkownika systemu Unix, przedstawiony na poniższej ilustracji, w pewien sposób był wersją podstawową dla rozwoju późniejszych systemów i przypomina znane z nich pulpity użytkownika. Kolejnym znanym systemem operacyjnym jest Linux. Linus Torvalds będąc studentem uniwersytetu w Helsinkach w roku 1991r. rozpoczął prace nad stworzeniem jądra systemu podobnego do Unix. Celem jego działań było stworzenie powszechnego systemu, który byłby używany bezpłatnie na całym świecie. Systemu, który byłby otwarty i umożliwiał rozwijanie własności systemów operacyjnych wszystkim chętnym programistom.



Dlatego tak ważnym było uznanie licencji open source, pochodnej licencji UNG, jako podstawy działań. GPL (ang. General Public License) jest jedną z najbardziej znanych licencji oprogramowania i jedną z tych, na której podstawie udostępniane jest jądro Linux. Jądro systemu Linux stanowi wspólną cechę wszystkich produktów wchodzących w skład rodziny unixo podobnych systemów operacyjnych. Do jądra dołączone są narzędzia systemowe oraz biblioteki z projektu GNU. Obecnie jest on dostępny w formie licznych dystrybucji. Poszczególne wersje składają się z jądra oraz charakterystycznego zestawu pakietów oprogramowania. Aktualnie największe zastosowanie Linux ma w środowiskach serwerowych, gdzie komercyjne wsparcie oferują również tak znane firmy branży IT jak IBM, Sun Microsystems, Dell, Hewlett Packard, Red Hat i Novell. Linux jest również bardzo często stosowany w szerokiej gamie komputerów PC oraz wszelkiego rodzaju urządzeniach przenośnych i routerach. Dlatego też system Linux zaczyna być używany na całym świecie i staje się jednym z najpopularniejszych systemów operacyjnych. Niezależnie od omawianego nurtu ogólnodostępnego oprogramowania rozwijał się system operacyjny Windows. Prezentacja pierwszego graficznego środowiska z rodziny Windows Microsoft odbyła się w listopadzie 1985r.

Wówczas była to w zasadzie jedynie graficzna nakładka na MS-DOS (ang. Microsoft Isk Operating System), który sam w sobie jest środowiskiem tekstowym. Kolejne wersje Windows miały za zadanie stworzyć platformę programistyczną. Jednak w zasadzie wszystkie systemy tej linii startują z poziomu DOS i są z nim zgodne. Firma Microsoft oficjalnie zaprzestała rozwoju MS-DOS w 1993r. wraz z wydaniem systemu Microsoft Windows NT. Usunięcie systemu MS-DOS z Microsoft Windows było skomplikowanym przedsięwzięciem, ponieważ wiele ówczesnych programów i gier było zaprojektowanych wyłącznie dla systemu DOS. Pierwszą wersją opartą jedynie o poszczególne elementy MS-DOS jest Microsoft Windows XP, gdzie dostępne są jedynie niektóre funkcje poprzez interpreter poleceń uruchamiany komendą *cmd*.

```
Starten von MS-DOS...

HIMEM testet den erweiterten Speicher...beendet.

This driver is provided by Oak Technology, Inc..
DTI-91X ATAPI CD-ROM device driver, Rev D91XU352
(C)Copyright Oak Technology Inc. 1987-1997
Device Name       : CDROM
Transfer Mode      : Programmed I/O
Number of drives   : 1

C:\>C:\DOS\SMARTDRV.EXE /X
MSCDEX Version 2.23
Copyright (C) Microsoft Corp. 1986-1993. Alle Rechte vorbehalten.
Laufwerk D: = Treiber CDROM Gerät 0
C:\>_
```



Nakładka na MS-DOS, a później rodzina systemów operacyjnych Windows zdominowała światowy rynek komputerów osobistych ze względu na łatwość obsługi, instalacji urządzeń zewnętrznych oraz szeroką gamę dostępnych gier, muzyki, wideo i aplikacji multimedialnych. Jak podają źródła internetowe [marketshare.hitslink.com] w styczniu 2010r. systemy z rodziny Microsoft Windows były zainstalowane na 92,02% komputerów osobistych na świecie. Natomiast na terenie Polski rodzina Windows stanowi 98,99% udziałów w rynku [ranking.pl]. Zatem słusznym jest stwierdzenie, że w chwili obecnej jest to jeden z najczęściej stosowanych systemów w Polsce i Europie. Różnica pomiędzy rodziną Windows a Linux polega na licencji udostępniania użytkownikowi systemu operacyjnego. Podczas gdy Linux jest otwarty i można go modyfikować, Windows jest własnością Microsoftu i użytkownik może jedynie aktualizować system na podstawie udostępnionych przez autorów poprawek. Microsoft oferuje kompleksowe rozwiązania dla firm i użytkowników domowych, zatem w większości przypadków operowanie systemem Windows jest łatwiejsze i wymaga mniejszej znajomości komputera niż w przypadku użytkownika Linux.



Microsoft między innymi udostępnia Visual Studio 2010 w wersji Express Edition, używane przez autorów książki, do stworzenia przedstawionego w niej oprogramowania w języku C/C++. Produkt ten umożliwi nam pisanie kodu źródłowego programu do skompilowania zarówno pod systemem Linux jak i Windows.



# Rozpoczynamy pracę

Programowanie w środowisku Linux jest możliwe na dwa sposoby. Pierwszym jest posiadanie zainstalowanego systemu Linux na naszym biurkowym komputerze wraz z dostępnymi kompilatorami *gcc* czy *g++*. Innym powszechnym sposobem jest programowanie poprzez dostęp do aplikacji serwera zdalnego. Serwer zdalny jest to komputer, który jest udostępniony w sieci poprzez właściwego mu administratora wybranym użytkownikom. Każdy z nich otrzymuje od administratora login oraz odpowiadające mu hasło. Dane te umożliwiają logowanie do systemu np. za pomocą standardowego protokołu sieciowego SSH (ang. Secure Shell) w architekturze klient – serwer. Do ustanowienia takiego połączenia potrzebny jest tzw. klient, czyli program umożliwiający połączenie.

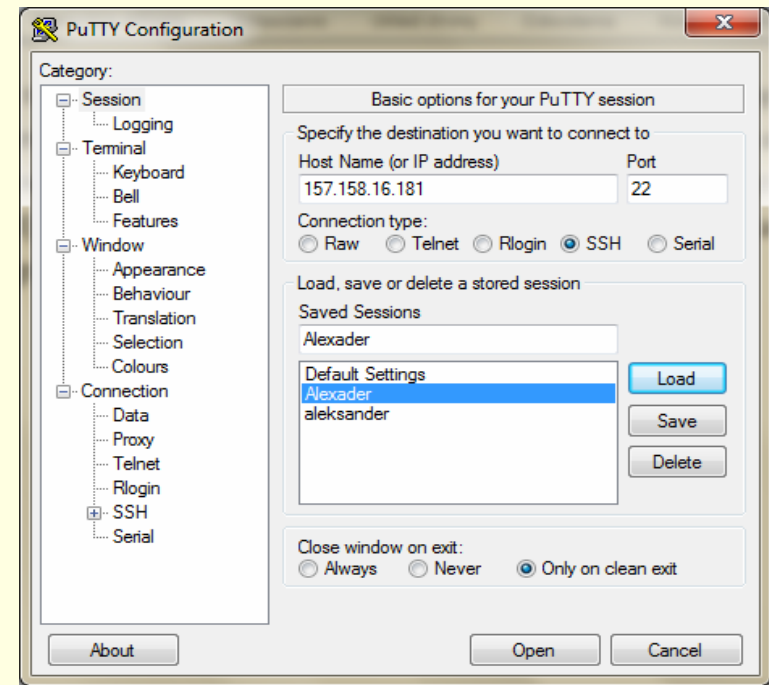
W rozdziale tym omówione zostanie połączenie z serwerem opartym o system Linux uzyskane ze stacji roboczej pracującej pod systemem z rodziny Windows. W naszym przypadku wykorzystany zostanie popularny program *PuTTY* dostępny w sieci [[www.putty.org](http://www.putty.org)] na licencji open source.

Pod systemem z rodziny Windows umożliwia on konsolowe połączenie z wybranym przez użytkownika zdalnym serwerem pracującym w systemie Linux.

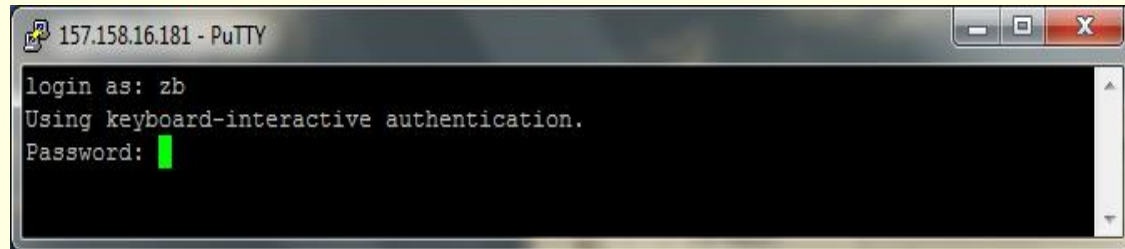
Na powyższej ilustracji został pokazany ekran użytkownika programu *PuTTY*. W książce uwagę Czytelnika chcemy skierować na programowanie, dlatego zajmiemy się jedynie opisem podstawowych funkcji omawianego programu. Program *PuTTY* umożliwia dowolne ustawienie koloru tła ekranu oraz wielkości i rozmiaru czcionki,

Czytelnik zechce samodzielnie wypróbować różne opcje i możliwości.

Interesującymi dla połączenia typu klient – serwer są pola *IP address* oraz *port*.

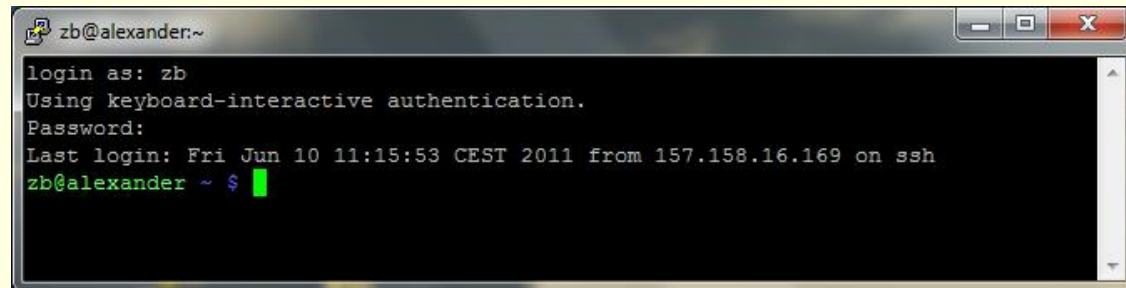


Pierwsze jest polem, w którym należy umieścić nazwę serwera. Adres możemy opisać w dwojaki sposób. Pierwszym jest nazwa w postaci adresu IP, który odpowiada uszeregowaniu serwera w poszczególnej sieci i domenie, której jest członkiem. Odpowiednikiem takiego zapisu jest zapis domenowy. Po uzyskaniu połączenia z serwerem programista otrzymuje zapytanie, wysłane przez serwer o dane użytkownika próbującego zalogować się do zdalnego systemu. Ekran, jaki wyświetli się w portalu poprzez protokół SSH został pokazany na ilustracji.



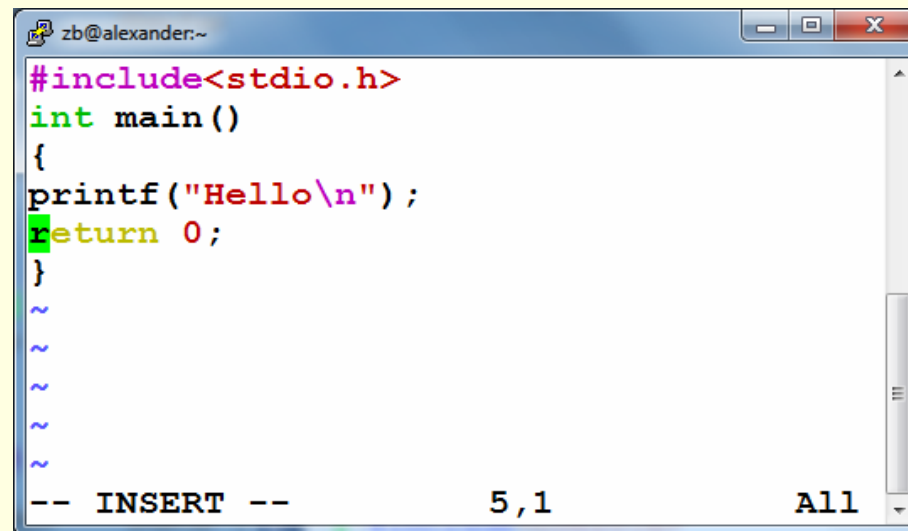
```
157.158.16.181 - PuTTY
login as: zb
Using keyboard-interactive authentication.
Password: █
```

Po zalogowaniu się do systemu Linux otrzymujemy „znak zachęty” \$, który umożliwia wprowadzanie dalszych poleceń użytkownikowi, co pokazuje kolejna ilustracja.



```
zb@alexander:~
login as: zb
Using keyboard-interactive authentication.
Password:
Last login: Fri Jun 10 11:15:53 CEST 2011 from 157.158.16.169 on ssh
zb@alexander ~ $ █
```

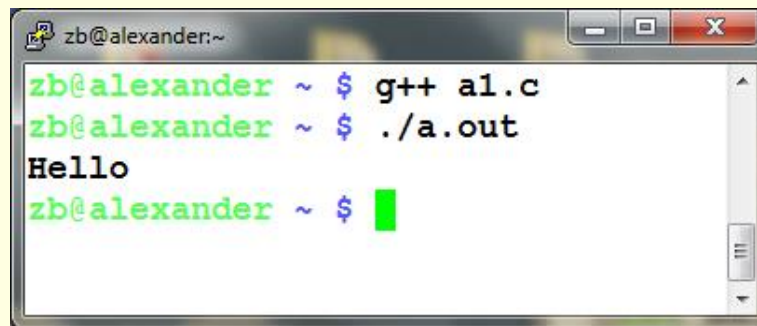
Jeśli Czytelnik korzysta z własnego komputera domowego z systemem Linux, będzie operował na konsoli systemowej, która bezpośrednio umożliwia połączenie ze zdalnym serwerem. Połączenie odbywa się w analogiczny sposób jak pokazano na ilustracjach w omawianym przypadku. Po wykonaniu połączenia uruchomiona konsola programisty pracuje i możemy już przystąpić do napisania naszego pierwszego programu. W tym celu należy wywołać jeden z dostępnych edytorów np. *vi*, *nano* lub *joe*. Trzeba jednak pamiętać, że wywołanie edytora łączy się bezpośrednio z nazwą pliku. Wywołanie takie będzie mieć postać: *vi nazwa\_pliku.c* przedzielając spacją polecenie i wpisaną nazwę pliku. Utworzymy teraz standardowy plik z rozszerzeniem c np. *a1.c*.

A screenshot of a terminal window titled 'zb@alexander:~'. The window shows a C program being edited in the vi editor. The code is as follows:

```
#include<stdio.h>
int main()
{
printf("Hello\n");
return 0;
}
~
~
~
~
~
```

The status bar at the bottom of the window displays '-- INSERT --', the cursor position '5,1', and the word 'All'.

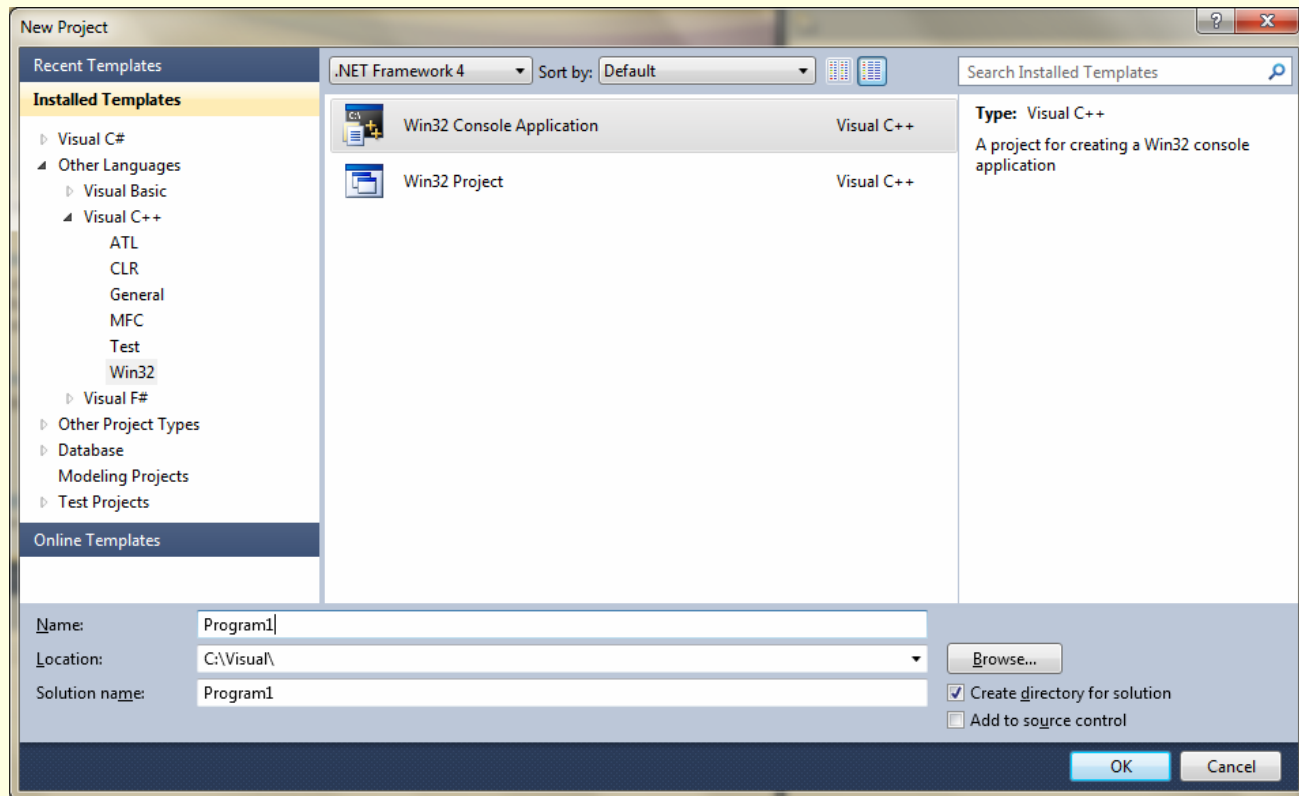
Do kompilacji stworzonego kodu swojego pierwszego programu Czytelnik zechce użyć kompilatora *gcc* albo *g++* wpisując w linii poleceń komendę pokazaną na poniższej ilustracji. Program jest domyślnie kompilowany do programu wykonywalnego o nazwie *a.out*. Stworzony w ten sposób program jest uruchamiany poleceniem *./a.out*. Przeszukuje ono domyślny katalog w celu odnalezienia programu o nazwie *a.out*. Jak widzimy napisany program drukuje napis „Hello” i kończy swoją pracę.

A screenshot of a terminal window with a title bar showing 'zb@alexander:~'. The terminal contains the following text: 'zb@alexander ~ \$ g++ a1.c', 'zb@alexander ~ \$ ./a.out', 'Hello', and 'zb@alexander ~ \$' followed by a green cursor block. The window has standard Linux window controls (minimize, maximize, close) in the top right corner.

```
zb@alexander ~ $ g++ a1.c
zb@alexander ~ $ ./a.out
Hello
zb@alexander ~ $
```

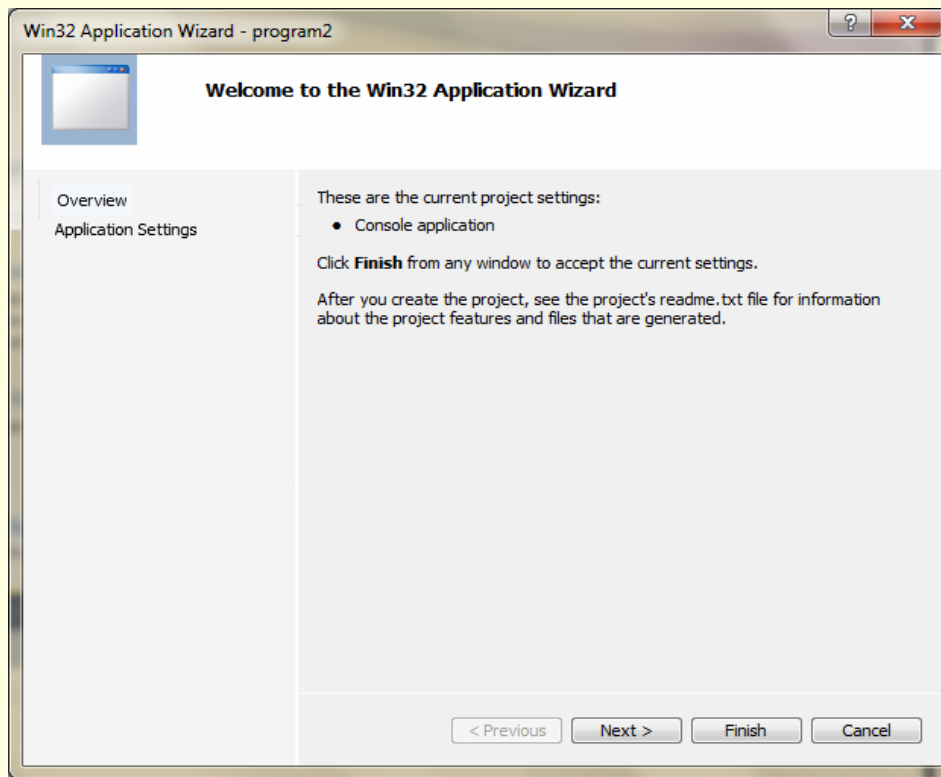
Bardzo ważną uwagą jest zasada wykonywalności kodu pod systemem Linux. W przypadku systemu Linux rozszerzenie pliku nie decyduje o wykonywalności zapisanego programu tylko nadaje mu uprawnienia. Do kompilowania programu za pomocą kompilatora *gcc* jest wymagane rozszerzenie *c*. Czytelnik powinien zwrócić szczególną uwagę, aby pisane przez niego programy miały rozszerzenie *c*, jak to jest pokazane na powyższej ilustracji

Spróbujmy teraz nasz program wykonać za pomocą aplikacji Microsoft Visual Studio 2010. W tym celu po uruchomieniu aplikacji wybieramy: *Microsoft Visual Studio 2010* → *New Project...* → *Visual C++* → *Win32 Console Application*, co zostało pokazane na kolejnej ilustracji.

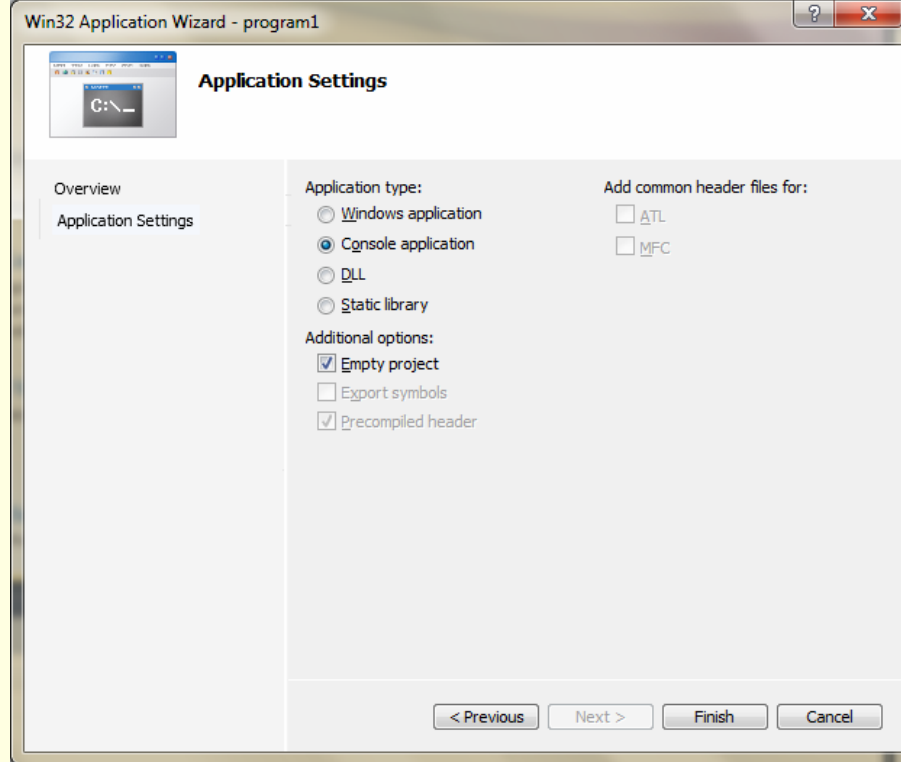




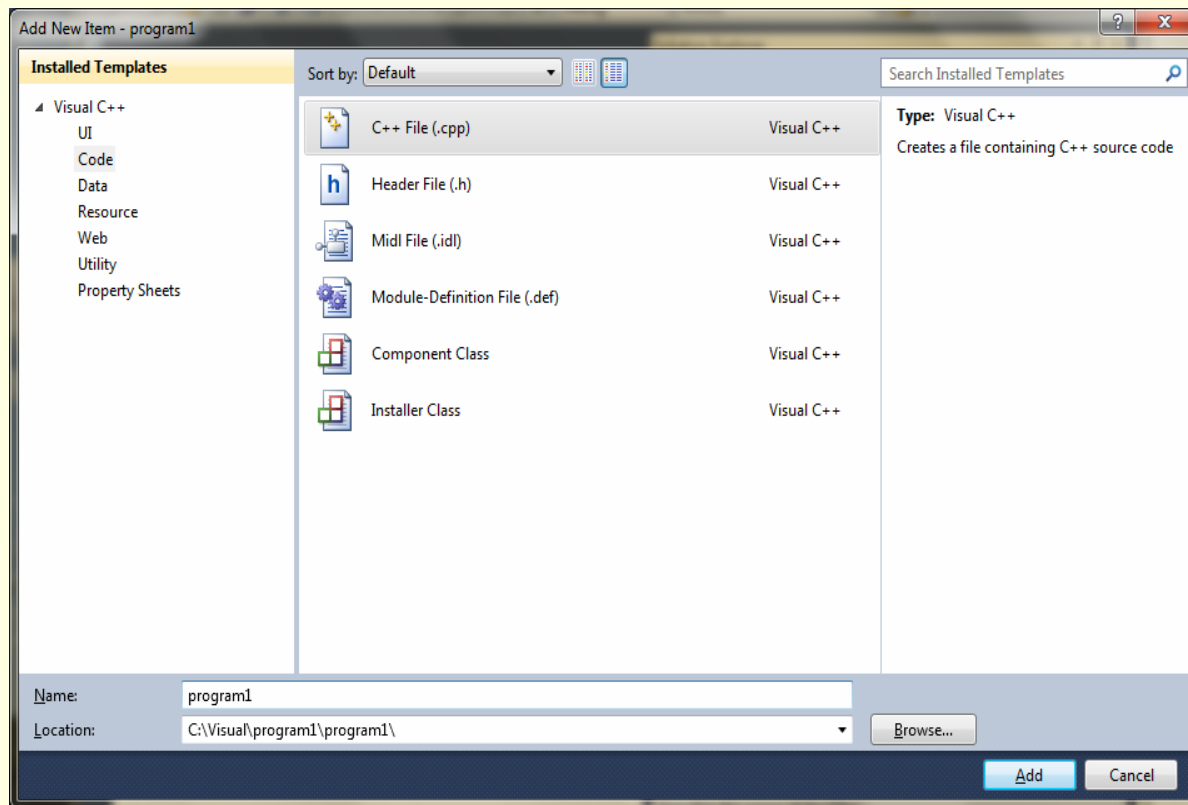
Następnie koniecznym jest ustawienie ścieżki definiującej dołączonej do naszego projektu. Można to uczynić podając bezpośrednio z klawiatury zadeklarowaną ścieżkę lub za pomocą przycisku *Browse*. Wybierając tę opcję jako łatwiejszą, Czytelnik zechce wybrać docelowy katalog na jednej z dostępnych mu na komputerze partycji. Należy pamiętać, że aby zapisać plik czy katalog na jednej z dostępnych partycji Czytelnik musi mieć uprawnienia zapisu danych. Uprawnienia takie udziela administrator danego systemu. Następnie należy wpisać nazwę programu (zwanego również projektem) w pole *Name* i zaakceptować całą operację klikając przycisk *OK*. Jeśli Czytelnik korzysta z własnego komputera powinien posiadać wszelkie prawa administracyjne, aby omawiana operacja nie będzie stanowiła problemu. Natomiast w przypadku korzystania z komputera jedynie udostępnianego użytkownikowi nadrzędnym jest ustawienie uprawnień dla konta użytkownika w domenie. Zatem Czytelnik być może będzie posiadał dostęp jedynie do wybranych katalogów lub partycji. O uprawnienia takie należy zapytać administratora systemu, jaki użytkuje Czytelnik lub osobę posiadającą odpowiednie uprawnienia w domenie.



W następnym okienku wybieramy „pusty projekt” i zatwierdzamy przyciskiem *Finish*. W tym momencie Visual Studio zapyta się o typ aplikacji, jaką chce stworzyć użytkownik. Jak widać na ilustracji Czytelnik ma do dyspozycji wiele dostępnych opcji. W omawianym przez autorów przypadku używane będą jedynie opcje dla trybu *Konsole Application*.



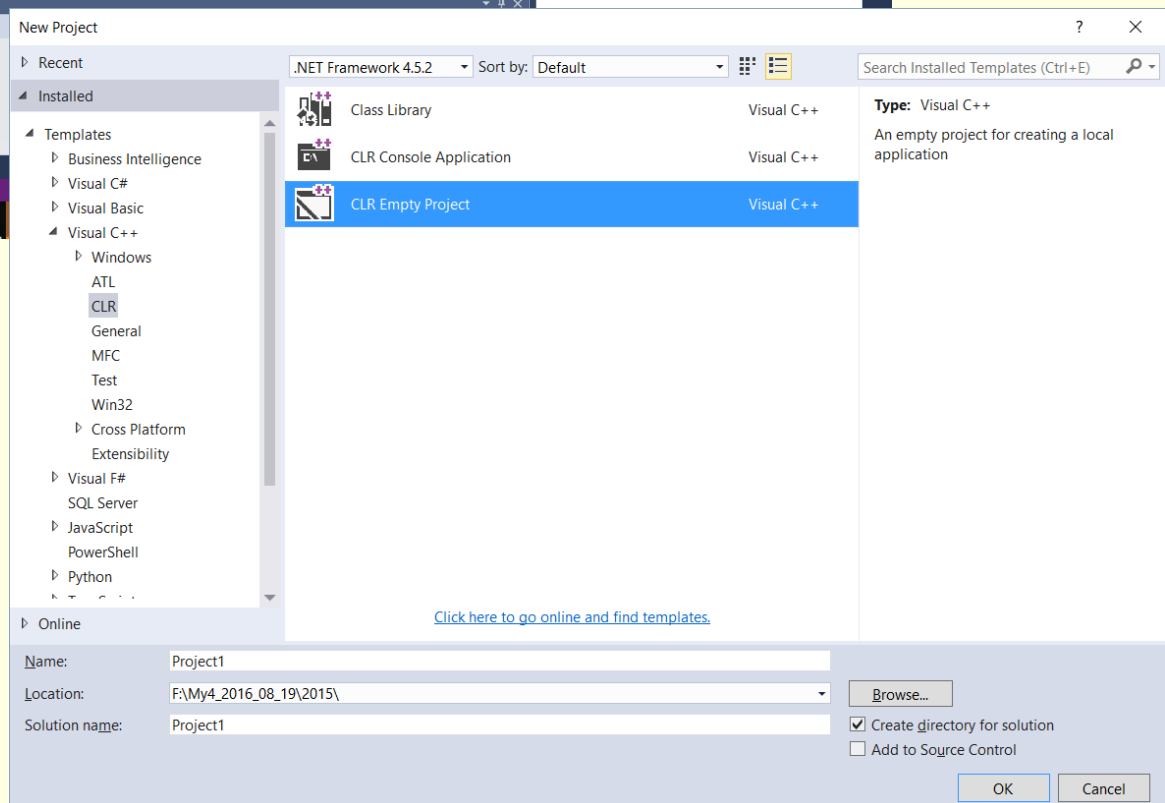
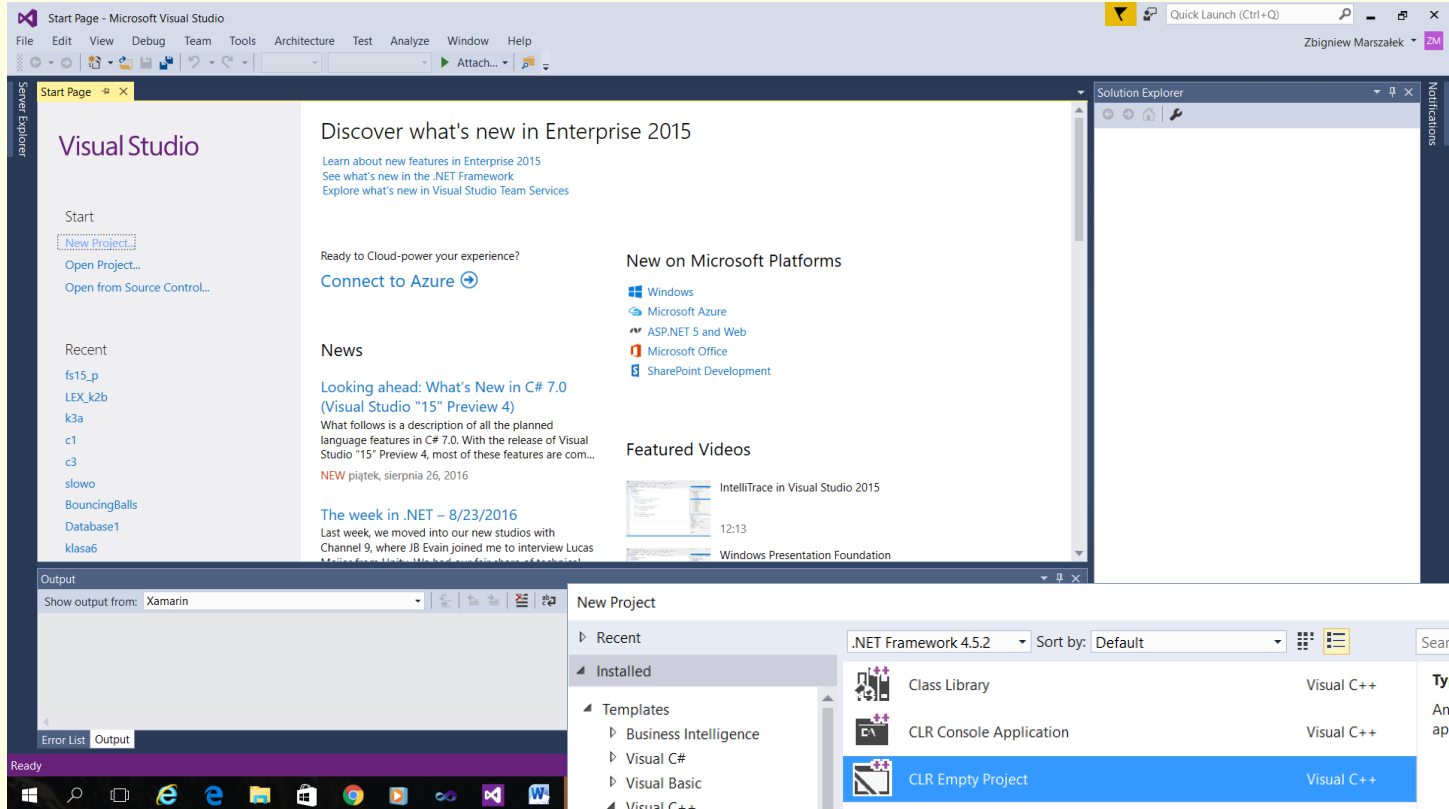
Taki typ jest niczym innym jak trybem bez rozbudowanego interfejsu graficznego. Oznacza to, że wykonywany kod będzie widoczny w oknie konsoli systemu Windows, czyli w postaci „czarnego okna” z wyświetlanym tekstem. Opcje, jakie należy wybrać dla utworzenia „pustego projektu” w postaci aplikacji konsolowej zostały pokazane na ilustracjach. W wyniku wykonanych operacji otrzymujemy „pusty projekt”. Czytelnik, aby móc wpisywać kod programu przyciska *CTRL+Shift+A*. W oknie dialogowym w kolumnie *Installed Templates* wybieramy opcję *Code*. Jako plik utworzony w systemie Windows wybieramy *C++ File(.cpp)*.



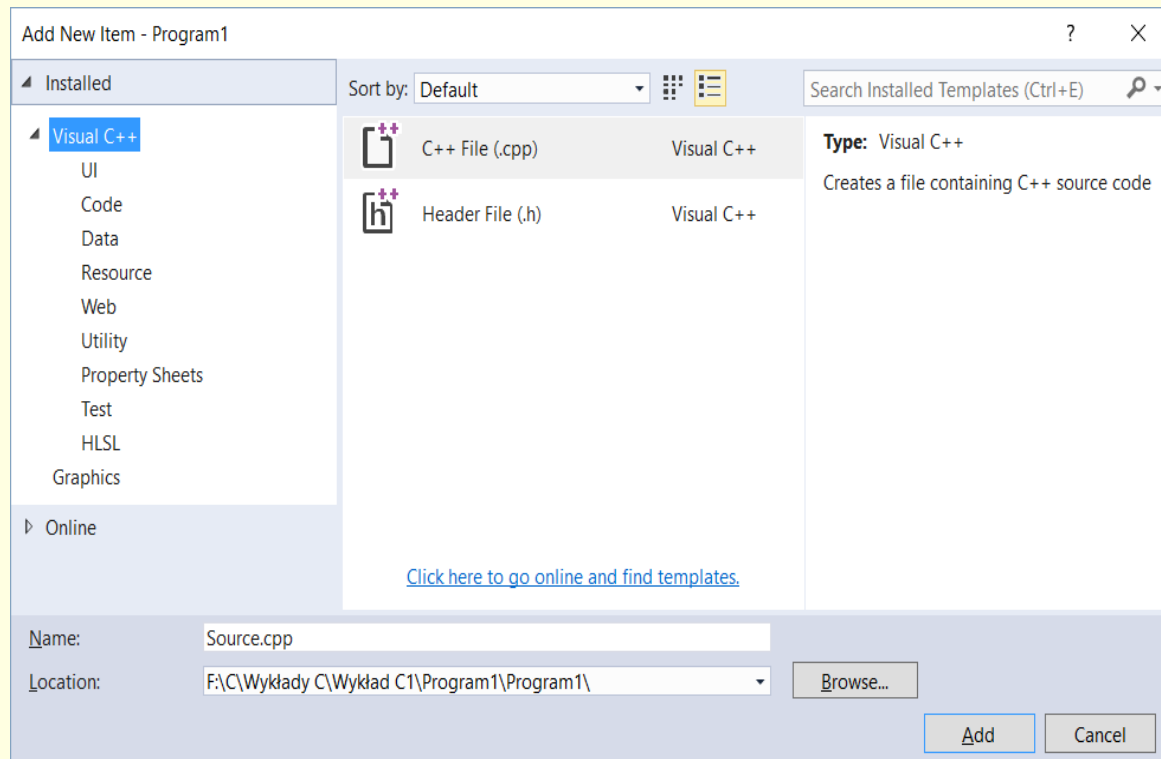
W polu *Name*: wpisujemy własną nazwę programu dla lokalizacji, jaką Czytelnik ustalił we wcześniejszych operacjach. Należy zauważyć, że MS Visual Studio samoczynnie będzie wpisywał wybraną przez użytkownika ścieżkę we wszystkie adresy dotyczące bieżącego projektu. Okno aplikacji i wykonane w nim operacje zatwierdzamy przyciskiem *Add*

# Wybór projektu w Visual Studio 2015

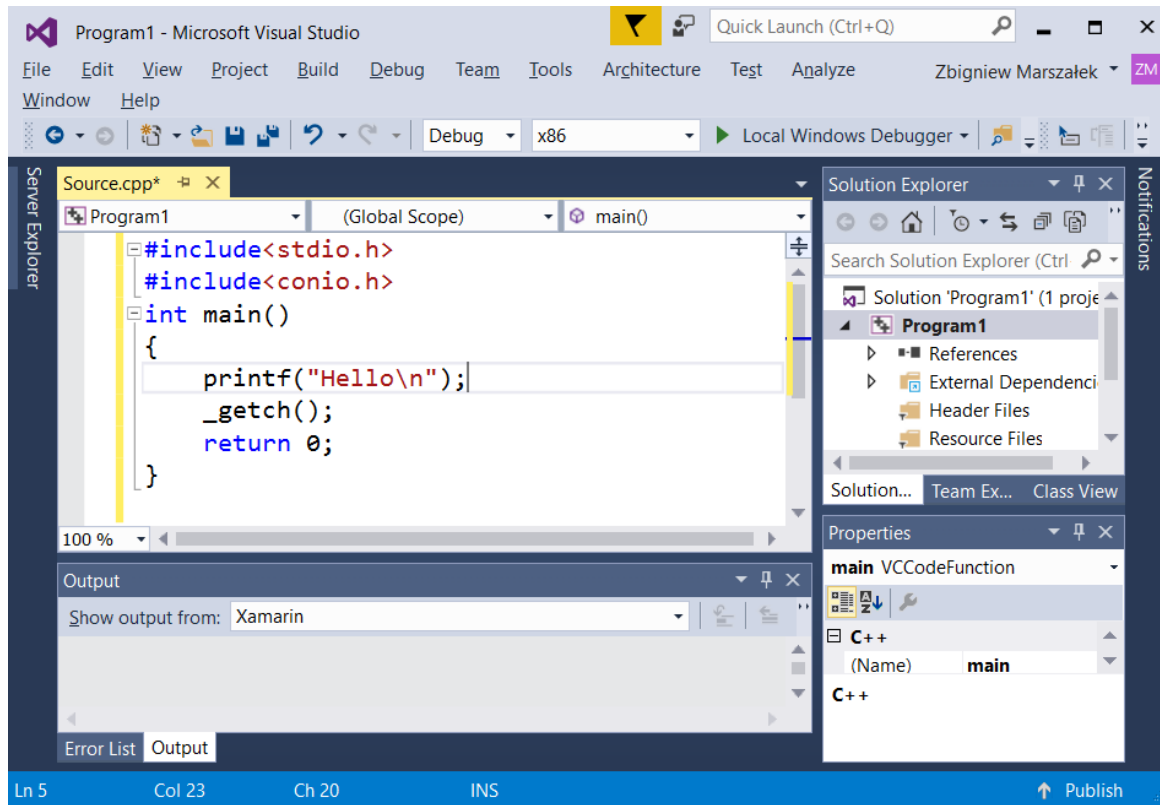
Firma Microsoft dokonuje bardzo istotnych zmian w składni języka C i C++ w bardzo krótkim okresie czasu. Wystarczy wspomnieć, że w ostatnich latach ukazały się cztery kolejne wersje programu Visual Studio: 2010, 2012, 2013 i 2015. Co więcej w działaniu kompilatorów są również zasadnicze różnice. Tak więc o ile kompilator w wersji 2010 przyjmował wszystkie instrukcje języka C, tak w wersji 2015 nie przyjmuje w ogóle instrukcji wprowadzania danych takich `scanf` i `fscanf`. Instrukcje te zostały usunięte przez firmę Microsoft jako instrukcje niebezpieczne dla środowiska Systemy Windows i zastąpione nowymi instrukcjami. Wyjściem z tej sytuacji jest rezygnacja z pracy z projektami Win32 i przejście na projekty CLR, które akceptują podstawową składnię języka C. Budowę projektu umożliwiającego pisanie programu w języku C rozpoczynamy od uruchomienia programu Visual Studio w wersji 2013 lub 2015 jak pokazano na rysunku. Następnie wybieramy Templates→Visual C++→CLR→CLR Empty Project i zmieniamy nazwę projektu.





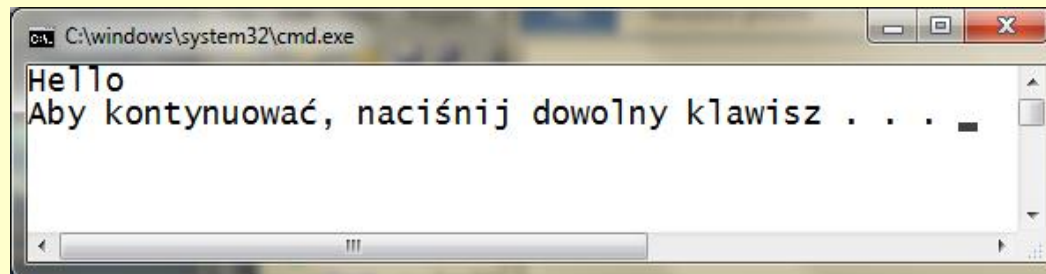


Klikamy w zakładkę Add i otrzymujemy pustą planszę, na której możemy wpisywać kod naszego programu w języku C.



1)	<code>#include&lt;stdio.h&gt;</code>	// deklaracje zastosowanych bibliotek
2)	<code>#include&lt;conio.h&gt;</code>	
3)	<code>int main()</code>	// kod głównej funkcji programu
4)	<code>{</code>	
5)	<code>printf("Hello\n");</code>	// funkcja wydruku tekstu
6)	<code>_getch();</code>	
7)	<code>return 0;</code>	
8)	<code>}</code>	

Do zatrzymania pojawiającego się komunikatu w trybie Debuggowania dodaliśmy bibliotekę *conio.h* i funkcję *\_getch()*. Czytelnik sprawdzi reakcje systemu po usunięciu tych fragmentów z kodu tworzonej aplikacji. Oczywiście możemy również skopiować program napisany wcześniej pod systemem Linux i uruchomić go za pomocą zakładki *Start Without Debugging* lub skrótem klawiszowym *Ctrl+F5* bez śledzenia kodu programu przez MS Visual Studio. Jak widzimy nie ma wielu różnic w strukturze programów uruchamianych pod systemem Linux i Windows, co stanowi ogromne ułatwienie dla programistów. Okno wykonania stworzonej przez Czytelnika aplikacji zostało zamieszczone na ilustracji.



Przyjrzyjmy się teraz dokładnie naszemu pierwszemu programowi, śledząc, co znajduje się w poszczególnych liniijkach kodu.

1) pierwsza i 2) druga linijka kodu zawiera polecenie dołączenia bibliotek. Oznacza to, że w trakcie kompilacji zostaną dołączone do kodu programu informacje zawarte we wskazanej przez programistę bibliotece. W tym przypadku dołączona zostanie biblioteka *stdio.h*, w której zdefiniowane są funkcje wprowadzania znaków z klawiatury oraz wypisywania znaków na urządzenia zewnętrzne np. *printf()*, *scanf()*. Jako znak będziemy rozumieli jeden bajt reprezentujący jeden poszczególny znak zapisany w kodzie ASCII. W ten sposób można zapamiętać w każdym bajcie duże i małe litery angielskie, cyfry i znaki specjalne np. interpunkcyjne. W programie komputer rozpoznaje znak, jako liczbę całkowitą z zakresu od -128 do 127. Oznacza to, że należy uważać, aby odpowiednio interpretować wczytywane bajty. Inaczej mówiąc nasze programy będą działać poprawnie, jeżeli będziemy używali języka angielskiego.

3) trzecia linijka kodu zawiera deklarację funkcji *main()*. Jest to główna funkcja w kodzie, od której program rozpoczyna działanie. Funkcja ta po zakończeniu wykonywania operacji przekazuje wartość całkowitą 0 do systemu operacyjnego za pomocą instrukcji *return 0;*

4) i 8) linijka kodu zawiera nawiasy klamrowe, które dla funkcji *main()* oznaczają kolejne operacje, które należy wykonać w ramach programu.

5) piąta linijka zawiera deklarację użycia funkcji *printf()*. Wypisuje ona zadany w cudzysłowach ciąg znaków na ekran. Jak widać w tym przypadku jest to słowo *Hello*. Czytelnik zwróci uwagę na zamieszczenie sekwencji *\n*. W kodzie programu oznacza to chęć przejścia do nowej linii. W języku C/C++ pojedynczy znak umieszczamy w pojedynczych cudzysłowach np. 'a' natomiast ciągi znaków, czyli słowa czy całe zdania w podwójnych cudzysłowach tak jak to ma miejsce w naszym kodzie. Należy tutaj zwrócić uwagę na różnice pomiędzy omawianymi systemami. W systemie operacyjnym Linux znak nowej linii jest zapisany w jednym bajcie, natomiast w systemie z rodziny Windows są to dwa kolejne bajty.

6) szósta linijka wywołuje funkcję `_getch()`. Jest ona zdefiniowana w bibliotece `conio.h`, którą wraz ze wszystkimi zawartymi w niej informacjami dołączyliśmy do kodu programu deklaracją `#include` w linijce drugiej. Wywołanie funkcji `_getch()` powoduje zatrzymanie programu do czasu „przyciśnięcia” dowolnego znaku na klawiaturze. Co widać na ilustracji pokazującej ekran konsoli wykonującej napisany program. System wypisze na ekran komentarz: „Aby kontynuować, naciśnij dowolny klawisz ...” po czym praca konsoli będzie zawieszona do momentu wykonania tej operacji przez użytkownika, a następnie konsola zakończy pracę. Deklaracja taka nie jest potrzebna w przypadku aplikacji wywołanej już w oknie konsoli ze względu na to, że stanowi ona w danym momencie środowisko pracy użytkownika.

7) siódma linijka oznacza operacje zwrócenia na wyjście 0 po zakończeniu pracy programu.



# Najpopularniejsze biblioteki w języku C/C++

Biblioteka (ang. library) jest to zbiór zbudowanych wcześniej funkcji, które zostały opisane w specyficzny sposób, aby umożliwić korzystanie z nich w dowolnym programie. Języki programowania mają swoje własne biblioteki, w których zawarte są podstawowe funkcje matematyczne, operujące na danych czy umożliwiające łączenie z różnymi urządzeniami. Natomiast nowe biblioteki najczęściej tworzy się i wykorzystuje w wyspecjalizowanych programach. Obszerniejszą informację o bibliotekach języka C/C++ można znaleźć w literaturze [75, 81]. Dobrą praktyką programisty jest wykorzystywanie istniejących już bibliotek i zapisanych w nich funkcji w nowych programach. Przykładowo, aby wypisać kilka słów na ekranie programista musi napisać własną funkcję drukującą znaki na wyjście standardowe lub może użyć funkcji *printf()* zawartej w bibliotece *stdio.h*. Biblioteki posiadają swoje pliki nagłówkowe, które zawierają deklaracje funkcji bibliotecznych oraz często zawarte są w nich komentarze, jak używać danej funkcji. Aby dołączyć bibliotekę do programu na samej górze kodu używamy polecenia *#include<>*, gdzie w nawiasy wpisujemy nazwę dołączanej biblioteki.

# Biblioteka matematyczna *math.h*

W bibliotece matematycznej zawarte są nie tylko przydatne stałe w postaci definicji, ale również wiele funkcji matematycznych m.in. funkcja logarytmiczna, znane funkcje trygonometryczne, funkcja podnoszenia liczby do określonej potęgi, pierwiastkowania oraz inne. Znajdują się tam również stałe liczbowe takie jak liczba  $\pi$  czy  $e$ .

# Biblioteka narzędzi ogólnego użytku *stdlib.h*

W tej bibliotece zawarte są funkcje różnego rodzaju, włącznie z generatorem liczb losowych, funkcjami przeszukującymi i sortującymi, funkcjami konwersji oraz funkcjami zarządzania pamięcią.

# Biblioteka do wykonywania operacji wejścia / wyjścia *stdio.h*

Biblioteka zawiera funkcje wykonujące operacje wejścia – wyjścia na strumieniach. Biblioteka ta umożliwia korzystanie ze strumieni przy wykonywaniu operacji z urządzeniami fizycznymi jak klawiatura, drukarka, terminal czy inne media udostępniane w systemie.

## Biblioteka do operacji na znakach *ctype.h*

Biblioteka klasyfikowania i przekształcania poszczególnych znaków oraz operacji na *string.h*. Biblioteka stworzona do pracy z napisami (ciągami znaków, tekstami).

# Biblioteka *assert.h*

Zawiera pomoce do debuggowania programów. Jeśli testowany warunek logiczny przyjmuje wartość *fałsz*, na standardowe wyjście błędów wypisywany jest komunikat o błędzie, zawierający m.in. argument wywołania makra, nazwę funkcji wywołania, nazwę pliku źródłowego oraz numer linii. Program jest przerywany poprzez wywołanie funkcji *abort()*. W ten sposób możemy oznaczyć w programie niezmienniki, czyli warunki, które niezależnie od wartości zmiennych muszą pozostać prawdziwe. Jeśli asercja zawiedzie, oznacza to, że popełniliśmy błąd w algorytmie, piszemy nadając zmiennym wartości, których nigdy nie powinny mieć albo nastąpiła po drodze sytuacja wyjątkowa, na przykład związana z obsługą operacji wejścia – wyjścia.

# Biblioteka obsługi czasu *time.h*

Biblioteka udostępnia kilka typów danych umożliwiających wykonywanie operacji na czasie, takie jak dodawanie czy odejmowanie.

Ponieważ „czysty” język C sam w sobie praktycznie nie ma mechanizmów do obsługi np. wejścia-wyjścia, dlatego biblioteki są ważnym dodatkiem. Udostępniają one wiele funkcji realizujących takie działania jak: wejście/wyjście, obsługa plików, zarządzanie pamięcią, przeszukiwanie i sortowanie, obliczenia matematyczne oraz przetwarzanie łańcuchów.