Politechnika Białostocka

Wydział Elektryczny

Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja do pracowni specjalistycznej

Temat ćwiczenia:

ŚRODOWISKO MICROSOFT VISUAL C++. JĘZYK C - OGÓLNA STRUKTURA PROGRAMU

Ćwiczenie nr INF_D01

Pracownia specjalistyczna z przedmiotu:

Informatyka

Kod: **EDS1B1007**

Opracował:

dr inż. Jarosław Forenc

Białystok 2022

Materiały zostały opracowane w ramach projektu „PB2020 – Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Spis treści

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows 10.

1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Microsoft Visual Studio 2008 Standard Edition lub nowsze zawierające kompilator Microsoft Visual C++.

2. Środowisko Microsoft Visual Studio

2.1. Microsoft Visual Studio

Microsoft Visual Studio jest zintegrowanym środowiskiem programistycznym (ang. *IDE - Integrated Development Environment*) umożliwiającym tworzenie samodzielnych aplikacji konsolowych lub z graficznym interfejsem użytkownika, aplikacji sieciowych, usług sieciowych oraz serwisów internetowych. W skład środowiska wchodzą następujące narzędzia:

- Microsoft Visual C# (od wersji 2002);
- Microsoft Visual C++;
- Microsoft Visual Basic;
- Microsoft Visual J# (wersje 2002-2005);
- Microsoft Visual Web Developer ASP.NET (od wersji 2005);
- Microsoft Visual F# (od wersji 2010).

Środowisko dostępne jest w czterech edycjach:

- Microsoft Visual Studio Express;
- Microsoft Visual Studio Community;
- Microsoft Visual Studio Professional;

- Microsoft Visual Studio Test Professional;
- Microsoft Visual Studio Enterprise.

Aktualna stabilna wersja programu to Visual Studio 2022 version 17.3, która została wydana 9 sierpnia 2022 roku. W instrukcji została opisana wersja Microsoft Visual Studio 2008 Express Edition.

2.2. Microsoft Visual Studio 2008 Express Edition

Wersja Express Edition środowiska Microsoft Visual Studio jest dostępna bezpłatnie. Zawiera uproszczone wersje programów dostępnych w płatnych wersjach. Licencja pozwala także na tworzenie programów komercyjnych. W środowisku dostępne są następujące języki programowania:

- Visual Basic 2008 Express Edition;
- Visual C++ 2008 Express Edition;
- Visual C# 2008 Express Edition;
- Visual Web Developer 2008 Express Edition.

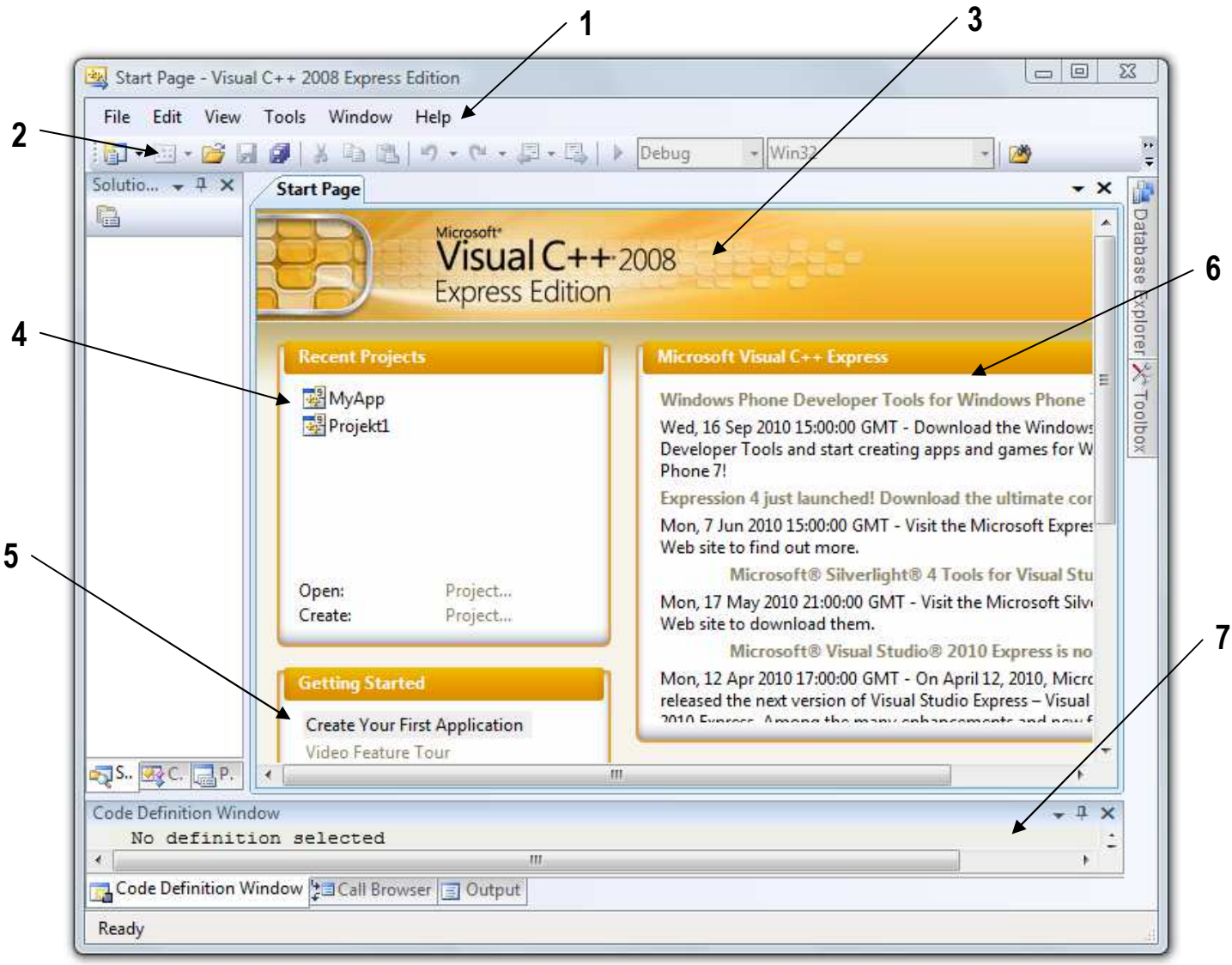
Według dokumentacji możliwe jest uruchomienie programu na następujących systemach operacyjnych: Windows XP SP2 (lub wyższy), Windows Server 2003 SP1 (lub wyższy), Windows Server 2003 R2 (lub wyższy), Windows Vista, Windows Server 2008. Nie powinno być żadnych problemów z uruchomieniem środowiska na nowszych systemach operacyjnych (Windows 7/8/10).

W Tabeli 1 zestawiono minimalne i rekomendowane wymagania sprzętowe środowiska.

Tabela 1. Wymagania sprzętowe środowiska Microsoft Visual Studio 2008 Express Edition

Parametr	Wymagania minimalne	Wymagania rekomendowane
Procesor	1,6 GHz	2,2 GHz
Pamięć RAM	384 MB	1024 MB
Karta graficzna	1024x768	1280x1024
Dysk twardy	5400 RPM	7200 RPM

Po uruchomieniu środowiska wyświetlane jest główne okno programu przedstawione na Rys. 1.



Rys. 1 Główne okno środowiska Microsoft Visual C++ 2008 Express Edition

Elementy głównego okna programu:

1 - Menu główne;

2 - Pasek narzędziowy;

3 - *Start Page* - strona startowa;

4 - *Recent Projects* - skróty do ostatnio otwieranych projektów oraz możliwość otwarcia istniejącego projektu (*Open*) lub utworzenia nowego (*Create*);

5 - *Getting Started* - łącza do dokumentacji pozwalającej zapoznać się z pracą w Visual C++ 2008;

6 - *Microsoft Visual C++ Express* - najnowsze informacje dotyczące środowiska;

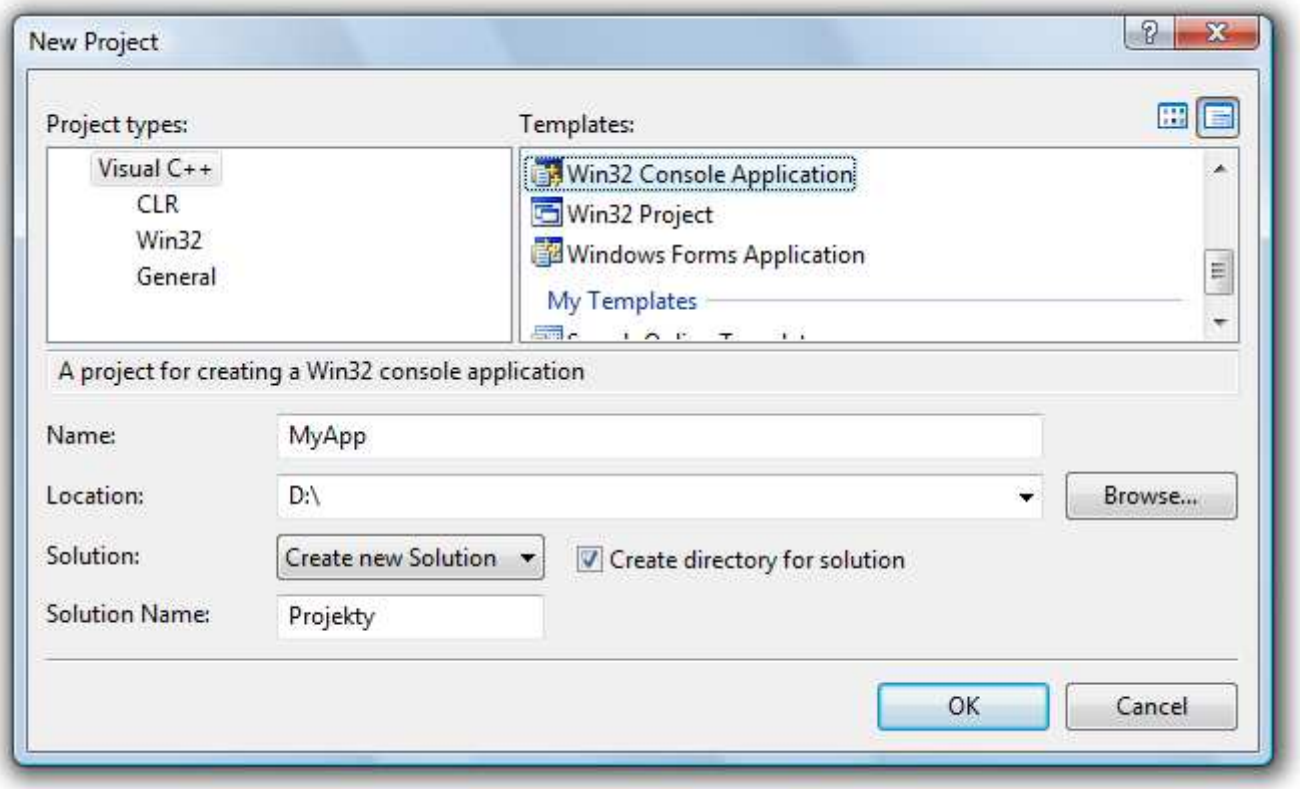
7 - *Output* - okno wyświetlające wyniki działania programu.

2.3. Utworzenie nowego projektu w Visual C++ 2008

Rozpoczęcie pracy ze środowiskiem Visual C++ 2008 wymaga utworzenia nowego projektu. Nowy projekt można utworzyć na kilka sposobów:

- wybierając w menu głównym **File → New → Project**;
- używając klawisza skrótu **Ctrl + Shift + N**;
- klikając ikonkę na pasku narzędziowym (pierwsza od lewej strony);
- klikając na karcie **Start Page → Recent Projects** opcję **Create: Project …**

Wybranie nowego projektu powoduje wyświetlenie okna (Rys. 2) przeznaczonego do określenia szablonu tworzonej aplikacji (*Templates*).



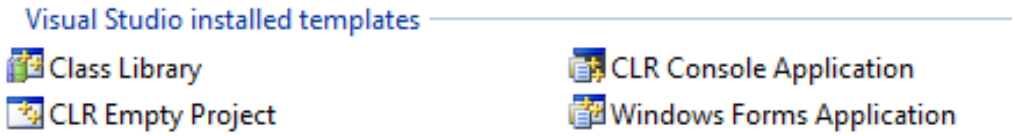
Rys. 2. Tworzenie nowego projektu

Szablony zebrane zostały w trzy grupy (*Project types*):

- **CLR** - szablony projektów przeznaczonych dla platformy .NET;
- **Win32** - szablony projektów pozwalających tworzyć kod natywny Windows (wykorzystujący WinAPI);
- **General** - inne szablony.

Grupa **CLR** zawiera następujące szablony (Rys. 3):

- **Class Library** - szablon przeznaczony do tworzenia bibliotek klas, które mogą być wykorzystane w innych projektach;
- **CLR Console Application** - szablon przeznaczony do tworzenia aplikacji konsoli systemu Windows;
- **CLR Empty Project** - szablon przeznaczony do generowania pustego projektu;
- **Windows Forms Application -** szablon przeznaczony do tworzenia aplikacji okienkowych (z graficznym interfejsem użytkownika).



Rys. 3. Szablony projektów z grupy CLR

Grupa **Win32** zawiera następujące szablony (Rys. 4):

- **Win32 Console Application** - szablon przeznaczony do tworzenia prostych aplikacji uruchamianych z Wiersza polecenia (aplikacje konsolowe);
- **Win32 Project** - szablon przeznaczony do tworzenia aplikacji systemu Windows z wykorzystaniem WinAPI.

Rys. 4. Szablony projektów z grupy Win32

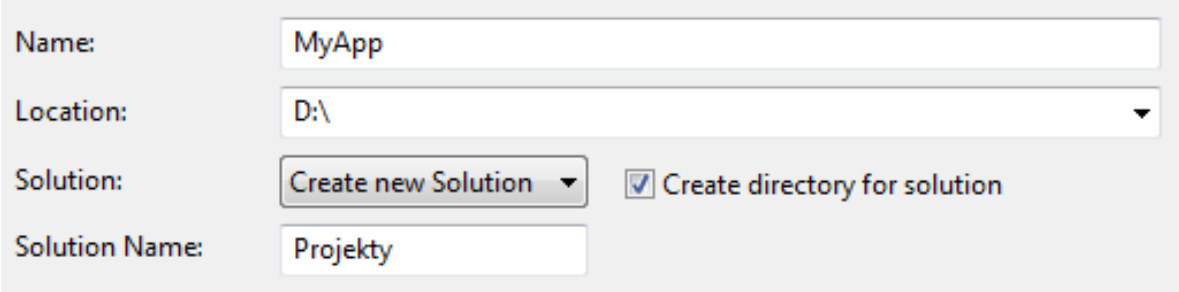
Grupa **General** zawiera następujące szablony (Rys. 5):

- **Empty Project** - szablon przeznaczony do tworzenia dowolnych aplikacji;
- **Makefile Project** - szablon przeznaczony do tworzenia aplikacji wykorzystujących przy kompilacji plik makefile.

Rys. 5. Szablony projektów z grupy General

W przypadku pisania programu w języku C należy wybrać projekt **Win32 Console Application** i wprowadzić (Rys. 6):

- nazwę projektu (**Name**), która będzie także nazwą pliku wynikowego **exe**;
- położenie plików projektu (**Location**);
- nazwę przestrzeni roboczej (**Solution Name**).



Rys. 6. Wprowadzanie danych opisujących projekt

Przestrzeń robocza (**Solution** - rozwiązanie, solucja) może składać się z wielu projektów. Każdy taki projekt jest samodzielnym programem. Wszystkie projekty wchodzące w skład przestrzeni roboczej mogą być skompilowane po wybraniu tylko jednego polecenia. Zazwyczaj projekty wchodzące w skład przestrzeni roboczej są ze sobą w pewien sposób powiązane, np. rozwiązują ten sam problem, ale różnymi metodami.

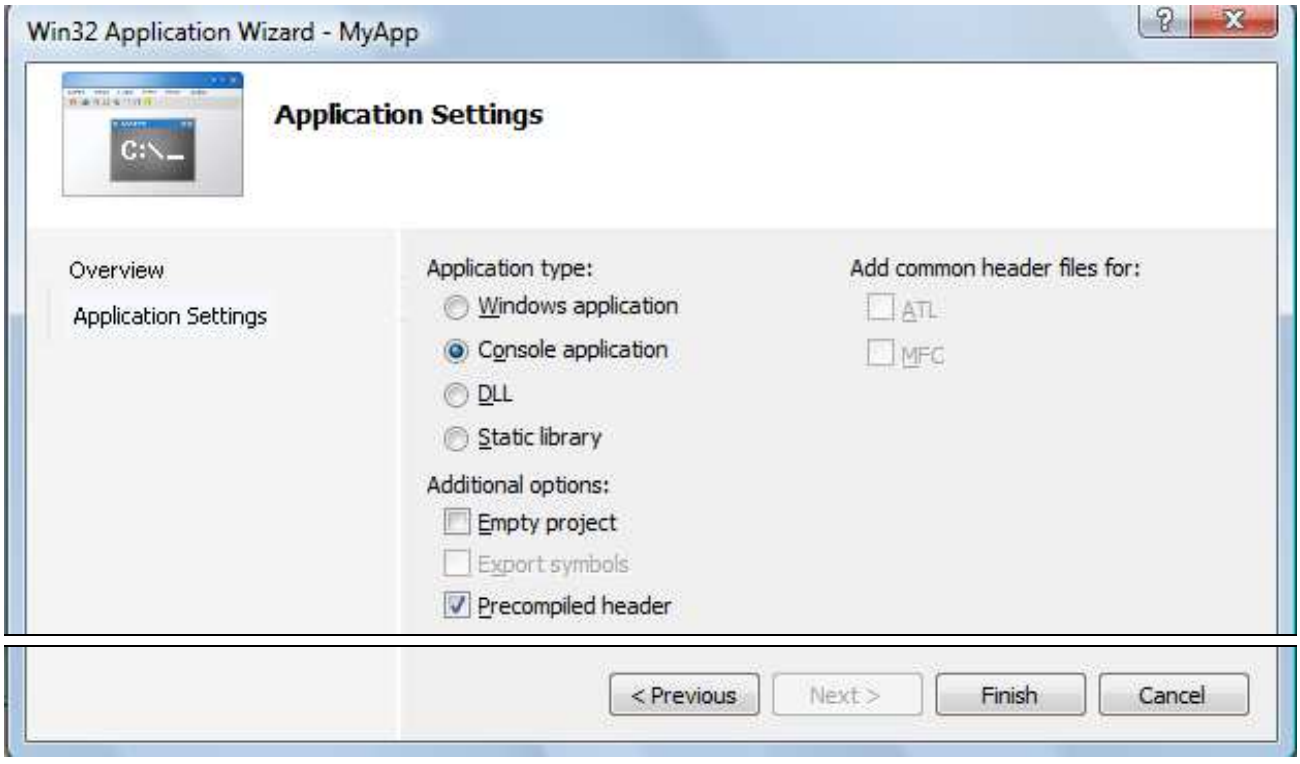
Po wybraniu typu projektu uruchamiany jest Kreator projektu (**Win32 Application Wizard**) składający się z dwóch kroków.

Pierwszy krok, **Overview** (Rys. 7), ma charakter tylko informacyjny.

Rys. 7. Kreator projektu - Overview

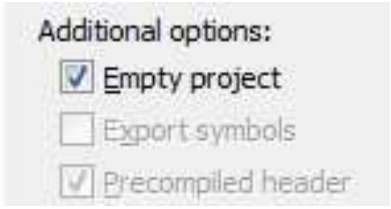
Po naciśnięciu przycisku **Next >** przechodzi się do drugiego kroku kreatora. W drugim kroku, **Application Settings** (Rys. 8), można:

- zmienić typ aplikacji (**Application type**);
- dodać pliki nagłówkowe (**Add common header files for**);
- zmienić dodatkowe opcje (**Additional options**).



Rys. 8. Kreator projektu - Application Settings

W oknie **Application Settings** zaznaczamy opcję **Empty Project** (Rys. 9).

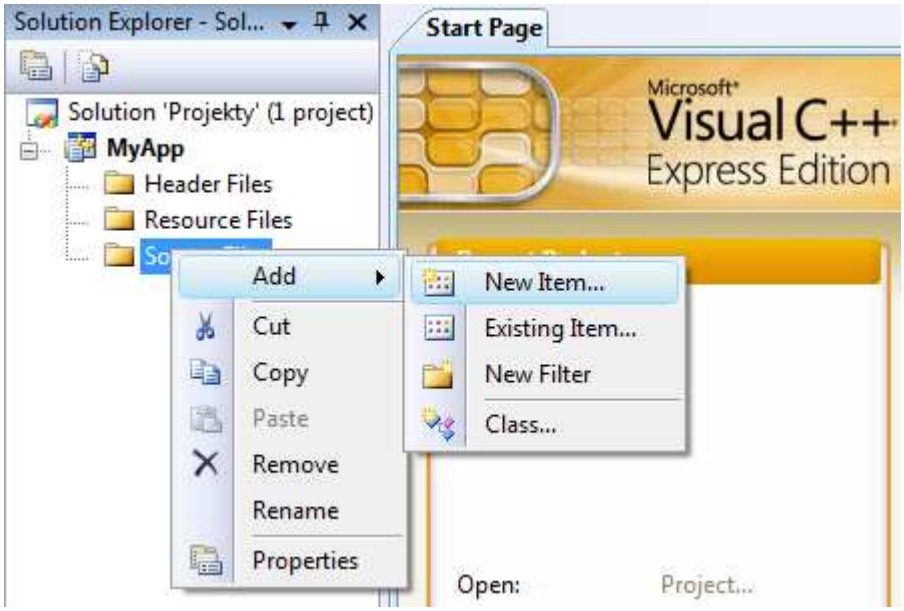


Rys. 9. Kreator projektu - zaznaczenie opcji Empty Project

Po naciśnięciu przycisku **Finish** zostanie utworzony nowy, pusty projekt.

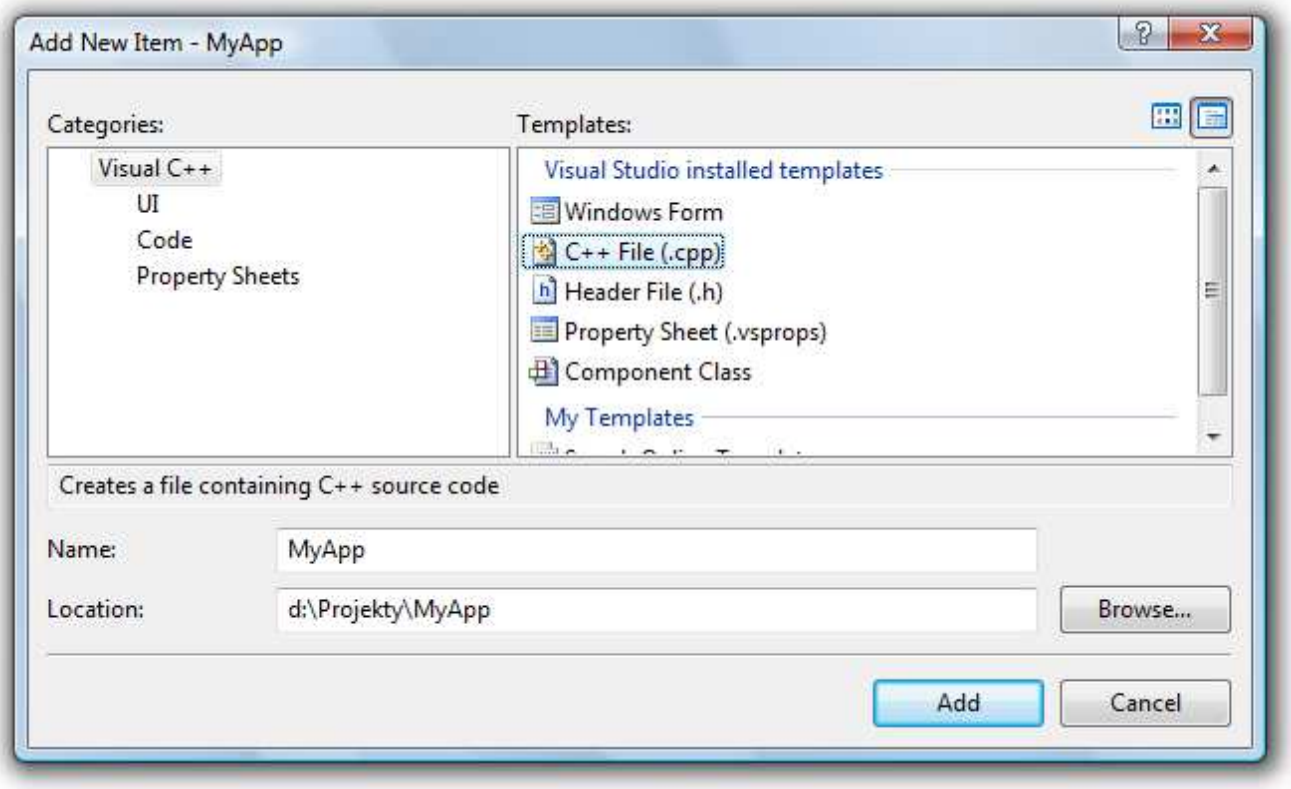
Wpisanie kodu programu wymaga dodania do projektu nowego pliku. Można to zrobić na kilka sposobów:

- wybierając z menu głównego: **File → New → File**;
- używając klawisza skrótu: **Ctrl + N**;
- klikając prawym klawiszem myszki w oknie **Solution Explorer** na **Source Files** i wybierając **Add → New Item…** (Rys. 10).



Rys. 10. Dodanie nowego pliku do projektu

Jako typ pliku (**Templates**) zaznaczamy **C++ File (.cpp)**, a następnie wprowadzamy nazwę pliku (**Name**) i ewentualnie miejsca jego zapisania na dysku (**Location**) (Rys. 11).



Rys. 11. Wprowadzenie nazwy pliku i miejsca zapisania na dysku

Wciśnięcie przycisku **Add** spowoduje utworzenie pustego pliku i jego otwarcie w edytorze kodu.

2.4. Język C

Historia powstania języka C rozpoczyna się na przełomie lat 60-tych i 70-tych XX wieku. W 1969 r. Martin Richards z University Mathematical Laboratories w Cambridge zdefiniował język BCPL. W 1970 r. Ken Thompson zdefiniował język B będący adaptacją języka BCPL dla pierwszej instalacji systemu operacyjnego Unix na komputer DEC PDP-7. Dwa lata później, w 1972 r., Dennis Ritchie z Bell Laboratories w New Jersey zdefiniował język NB (New B), nazwany później C, dla systemu Unix działającego na komputerze DEC PDP-11. W języku C zostało napisane ok. 90% kodu systemu i większość programów działających pod jego kontrolą. Dokumentacja tej wersji języka ukazał się w 1978 r. w postaci książki B.W. Kernighan, D.M. Ritchie: „*The C Programming Language*”. Był to pierwszy podręcznik do nauki języka C oraz nieformalna definicja standardu (od nazwisk autorów książki pochodzi jego nazwa - K&R).

W 1983 r. Amerykański Narodowy Instytut Standaryzacji (ANSI) powołał komitet X3J11, którego zadaniem było sformułowanie nowoczesnej i wszechstronnej definicji języka C. Komitet zakończył prace nad opracowaniem standardu ANSI w 1988 roku. Standard został zatwierdzony w 1989 roku jako ANSI X3.159-1989 „*Programming Language C*". Ta wersja języka określana jest jako ANSI C lub C89. W 1990 roku standard ANSI C został zaadoptowany przez organizację ISO w postaci normy ISO/IEC 9899:1990 (standard nazywany C90). Kolejne wersje standardu były publikowane w postaci norm ISO/IEC 9899:1999 (1999 r., standard nazwany C99), ISO/IEC 9899:2011 (2011 r., standard nazwany C11) i ISO/IEC 9899:2018 (2018 r., standard nazwany C18 lub C17).

2.5. Ogólna struktura programu w języku C

Program w języku C jest to niesformatowany plik tekstowy o odpowiedniej składni mający rozszerzenie **.c**. Najprostszy program ma następującą postać:

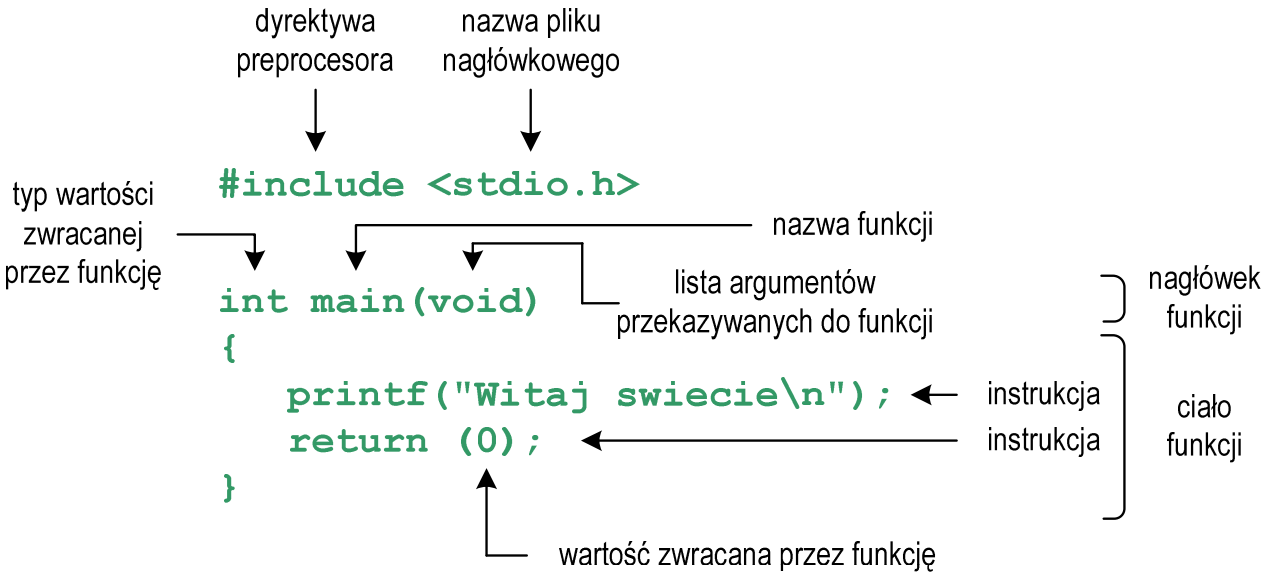
```
#include <stdio.h>

int main(void)
{
    printf("Witaj swiecie\n");
    return 0;
}
```

Program w języku C (Rys. 12) składa się z funkcji i zmiennych. Funkcje zawierają instrukcje określające wykonywane operacje, zaś zmienne przechowują wartości wykorzystywane podczas tych operacji.

Powyższy program składa się z jednej funkcji (**main()**), nie ma w nim natomiast zmiennych. Funkcja **main()** składa się z dwóch instrukcji: **printf()** i **return**. Każda instrukcja zakończona jest średnikiem.

W programie może być zdefiniowanych więcej funkcji, ale zawsze musi istnieć funkcja o nazwie **main()**, gdyż pełni ona szczególna rolę w programie - od początku tej funkcji rozpoczyna się wykonanie całego programu.



Rys. 12. Struktura programu w języku C

Funkcja w języku C rozpoczyna się od *nagłówka funkcji* (pierwszy wiersz). Zawartość funkcji ograniczona jest nawiasami klamrowymi: **{** , **}** i nazywana jest *ciałem funkcji*. *Nagłówek funkcji* i *ciało funkcji* tworzą *definicję funkcji*.

Język C rozróżnia wielkość liter, zatem nazwę funkcji **main()** nie możemy zapisać jako, np. **Main** lub **MAIN**. Podobnie jest z nazwami pozostałych funkcji i słów kluczowych języka C.

Zazwyczaj w programie poza funkcją **main()** występują inne funkcje - mogą to być funkcje napisane przez nas lub funkcje pochodzące z bibliotek. W powyższym programie do wyświetlenia tekstu na ekranie wykorzystywana jest funkcja o nazwie **printf()**. Skorzystanie z tej funkcji wymaga dołączenia do kodu programu informacji o bibliotece, w której funkcja ta została zadeklarowana. Służy do tego pierwsza linia programu: **#include <stdio.h>**. Instrukcja **#include** jest tzw. **dyrektywą preprocesora**. Dyrektywy takie wykonywane są jeszcze przed właściwą kompilacją programu (zob. Rozdz. 2.6). Dyrektywa **#include** oznacza wstawienie, w miejscu jej występowania, całej zawartości pliku **stdio.h**. Plik **stdio.h** określa standardową bibliotekę wejścia-wyjścia (ang. ***st**andar**d** **i**nput/**o**utput library*).

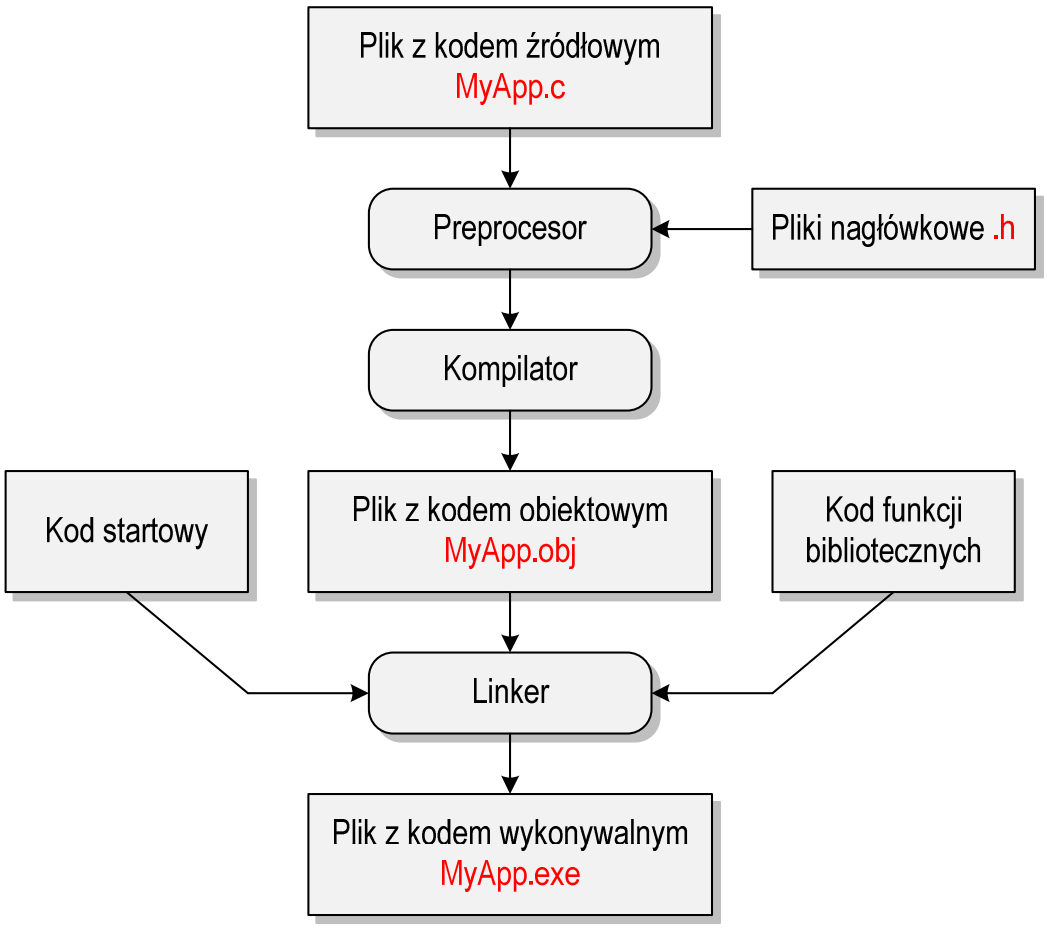
Funkcję wywołuje się podając jej nazwę (**printf()**) i (w nawiasach zwykłych) listę argumentów przekazywanych do funkcji (**"Witaj swiecie\n"**). W powyższym przykładzie do funkcji **main()** nie są przekazywane żadne argumenty, informuje o tym słowo **void** w jej nagłówku (słowo to może być pominięte).

Ciąg znaków ujęty w cudzysłów nazywa się stałą napisową (napisem, łańcuchem znaków). W powyższym łańcuchu znaków na samym jego końcu występuje sekwencja **\n** (ang. *newline character*) - reprezentuje ona znak nowego wiersza. Inaczej mówiąc powoduje ona przerwanie wypisywania tekstu w bieżącym wierszu i wznowienie wypisywania od lewego marginesu w następnym wierszu.

Instrukcja **return 0;** kończy wykonywanie funkcji **main()**, a tym samym i całego programu.

2.6. Tworzenie pliku wykonywalnego i uruchomienie programu

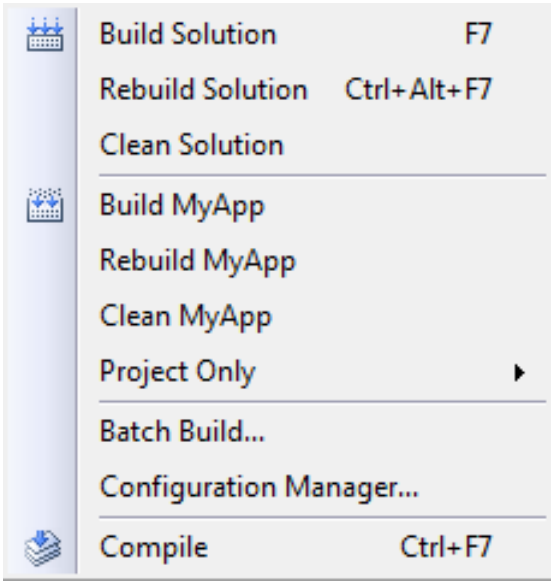
Uruchomienie programu wymaga przekształcenia pliku z kodem źródłowym na plik wykonywalny **exe**. Operacja ta składa się z dwóch kroków: kompilacji i łączenia (linkowania) (Rys. 13).



Rys. 13. Etapy tworzenia pliku wykonywalnego z kodu źródłowego

W pierwszym kroku specjalny program zwany preprocesorem analizuje kod źródłowy programu poszukując wyrażeń zaczynających się od znaku **#** (dyrektywy preprocesora). Na ich podstawie odpowiednio modyfikuje kod programu. Następnie kompilator (ang. *compiler*) kompiluje czyli przetwarza kod źródłowy programu w języku C (plik tekstowy) na kod maszynowy i zapisuje go w pliku obiektowym (ang. *object file*). Plik obiektowy jest plikiem binarnym z rozszerzeniem **.obj**. Chociaż kod maszynowy zawarty w tym pliku jest już zrozumiały dla procesora, to plik ten nie może być jeszcze uruchomiony. Brakuje w nim tzw. kodu startowego (ang. *start-up code*). Kod startowy tworzy interfejs pomiędzy programem a systemem operacyjnym. Dodatkowo do pliku obiektowego muszą być dołączone kody funkcji, które są wywoływane w programie, np. kod funkcji **printf()**. Kody te zapisane są w oddzielnych plikach (bibliotekach). Dołączaniem kodu startowego i kodu funkcji bibliotecznych do kodu obiektowego zajmuje się linker. Na koniec plik z kodem wykonywalnym (**exe**) zapisywany jest na dysku.

W środowisku Visual C++ do utworzenia pliku wykonywalnego można wybrać jedną z pozycji znajdujących się w menu głównym **Build** (Rys. 14).



Rys. 14. Menu Build do kompilacji programu

Poszczególne pozycje wykonują następujące operacje:

- **Build Solution (F7) -** buduje wszystkie projekty znajdujące się w przestrzeni roboczej (kompiluje pliki, które zmieniły się od czasu ostatniej kompilacji);

- **Rebuild Solution (Ctrl + Alt + F7) -** przebudowuje wszystkie projekty znajdujące się w przestrzeni roboczej (kompilowane są wszystkie pliki niezależnie od tego czy uległy zmianie od czasu ostatniej kompilacji);
- **Clean Solution** - usuwa wszystkie pliki będące wynikiem budowania projektów wchodzących w skład przestrzeni roboczej;
- **Build *MyApp*** - buduje aktywny projekt (o nazwie ***MyApp***);
- **Rebuild *MyApp*** - przebudowuje aktywny projekt;
- **Clean *MyApp*** - usuwa wszystkie pliki będące wynikiem budowania aktywnego projektu;
- **Project Only** - podmenu zawierające pozycje przeznaczone do budowania (**Build Only *MyApp***), przebudowania (**Rebuild Only *MyApp***), czyszczenia (**Clean Only *MyApp***) i linkowania (**Link Only *MyApp***) tylko aktualnego projektu;
- **Batch Build…** - otwiera okno budowania wsadowego;
- **Configuration Manager** - otwiera okno menadżera konfiguracji projektu;
- **Compile (Ctrl + F7)** - kompiluje edytowany plik z kodem źródłowym.

Jeśli w przestrzeni roboczej znajduje się tylko jeden projekt to najprostsza metoda kompilacji polega na wybraniu pozycji **Build Solution**. Można wtedy użyć klawisza skrótu **F7**. Przykładowy przebieg kompilacji (Compiling…) i łączenia (Linking…) przedstawiony jest poniżej.

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>Compiling manifest to resources...
1>Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
1>Copyright (C) Microsoft Corporation.  All rights reserved.
1>Linking...
1>Embedding manifest...
1>Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
1>Copyright (C) Microsoft Corporation.  All rights reserved.
1>Build log was saved at "file://d:\MyApp\Debug\BuildLog.htm"
1>MyApp - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Jeśli w kodzie programu występują błędy, to kompilator wyświetla tylko odpowiednie komunikaty, natomiast plik wykonywalny **exe** nie jest tworzony.

Skompilowany program uruchomia się poprzez wybranie pozycji **Start Without Debugging** z menu głównego **Debug** lub wciśnięcie klawisza skrótu **Ctrl + F5**.

Uruchomienie programu odbywa się automatycznie w oknie **Wiersza polecenia**. Przed uruchomieniem programu system operacyjny tworzy takie okno, a następnie uruchamia w nim program. Program wyświetla tekst „**Witaj swiecie**”. Dodatkowo środowisko Visual C++ zatrzymuje na koniec program i wyświetla komunikat: „**Aby kontynuować, naciśnij dowolny klawisz...**”. Po naciśnięciu dowolnego klawisza okno **Wiersza polecenia** jest zamykane.

Jeśli uruchomimy skompilowany program z poziomu systemu operacyjnego, to nie zobaczymy efektów jego działania. System operacyjny utworzy okno, uruchomi w nim program, program zakończy się i okno zostanie natychmiast zamknięte. Aby zaobserwować wyniki pracy programu należy zatrzymać go przed zakończeniem jego działania. Można to zrobić na kilka sposobów.

W pierwszej metodzie, przed instrukcją **return**, wywołujemy polecenie systemowe **pause**.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Witaj swiecie\n");

    system("pause");
    return 0;
}
```

Funkcja **system()** uruchamia polecenie o nazwie (ujętej w cudzysłów) przekazanej do niej jako argument. Polecenie **pause** zawiesza wykonywanie programu i wyświetla komunikat:

```
Aby kontynuować, naciśnij dowolny klawisz ...
```

Użycie w programie funkcji **system()** wymaga dołączenia pliku nagłówkowego **stdlib.h**. Po wyświetleniu komunikatu, naciśnięcie dowolnego klawisza spowoduje zakończenie programu i zamknięcie okna.

W drugiej metodzie przed instrukcją **return**, wywołujemy funkcję **getch()**.

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    printf("Witaj swiecie\n");

    getch();
    return 0;
}
```

Funkcja **getch()** zatrzymuje wykonywanie programu i czeka na wciśnięcie dowolnego klawisza. Użycie w programie funkcji **getch()** wymaga dołączenia pliku nagłówkowego **conio.h**.

2.7. Sposób zapisu kodu programu

Sposób zapisu kodu programu wpływa tylko na jego przejrzystość, a nie na kompilację i wykonanie. W tym właśnie celu w przedstawionych powyżej programach występują dodatkowe spacje przed funkcją **printf()** i słowem kluczowym **return**. Program wyświetlający tekst „**Witaj swiecie**” można zapisać także tak:

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

Środowisko Microsoft Visual Studio umożliwia automatyczne formatowanie kodu programu. W tym celu należy zaznaczyć kod programu (np. **Ctrl + A**), a następnie wcisnąć kombinację klawiszy **Ctrl + K + F**.

2.8. Struktura programu z kilkoma funkcjami, typy instrukcji w języku C

Omawiany poprzednio program, wyświetlający tekst **Witaj swiecie**, składał się tylko z jednej funkcji zdefiniowanej przez użytkownika (**main()**). W programie w języku C może występować więcej takich funkcji. Poniższy kod źródłowy zawiera trzy funkcje użytkownika: **komunikat1()**, **komunikat2()** i **main()**.

```
#include <stdio.h>

void komunikat1(void)
{
    printf("Zaczynamy...\n");  <-- 3a
}

void komunikat2(void)
{
    printf("Konczymy...\n");  <-- 3a
}
int main(void)
{
    int x;  <-- 1

    komunikat1();  <-- 3b
    x = 5;  <-- 2
    if (x > 0)  <-- 4
        printf("x to liczba dodatnia\n");  <-- 3a
    ;  <-- 5
    komunikat2();  <-- 3b

    return 0;  <-- 4
}
```

Wykonanie programu zawsze rozpoczyna się od funkcji **main()**. Gdy dochodzimy do instrukcji zawierającej funkcję **komunikat1()**, to wywołanie tej funkcji powoduje przekazanie sterowania do jej pierwszej instrukcji. Po wykonaniu wszystkich instrukcji znajdujących się w tej funkcji następuje powrót do miejsca wywołania. Następnie wykonywane są kolejne instrukcje funkcji **main()**. W przypadku wywołania funkcji **komunikat2()** sytuacja powtarza się: sterowanie przekazywane jest do pierwszej jej instrukcji, wykonywane są wszystkie instrukcje

w niej występujące i następuje powrót do miejsca wywołania. Wynikiem wykonania powyższego programu jest wyświetlenie napisów:

```
Zaczynamy...
x to liczba dodatnia
Konczymy...
```

W języku C występuje pięć typów instrukcji. W powyższym kodzie źródłowym poszczególne typy instrukcji zostały oznaczone kolejnymi liczbami:

1 - instrukcja deklaracji (deklaracja zmiennej **x** typu **int**);

2 - instrukcja przypisania (nadanie wartości **5** zmiennej **x**);

3 - instrukcja wywołania funkcji (**3a** - bibliotecznej, **3b** - użytkownika);

4 - instrukcja sterująca (instrukcja warunkowa **if**, instrukcja zwrotu **return**);

5 - instrukcja pusta.

2.9. Wyświetlanie tekstu funkcją printf()

Sekwencja **\n** w **printf()** powoduje przerwanie wypisywania tekstu w bieżącym wierszu i wznowienie wypisywania od lewego marginesu w następnym. Sekwencja **\n** może występować w dowolnym miejscu łańcucha znaków.

`printf("Witaj swiecie\n");`	`Witaj swiecie` _

`printf("Witaj\nswiecie\n");`	`Witaj`
	`swiecie`
	_

`printf("Witaj ");`
`printf("swiecie");`
`printf("\n");`
	`Witaj swiecie`
	_

W języku C istnieje kilka znaków, które pełnią specjalną funkcję w łańcuchu znaków. Nazywane są one sekwencjami sterującymi (ang. *escape sequence*). Znaki te zostały przedstawione w Tabeli 2.

Tabela 2. Sekwencje sterujące w łańcuchu formatującym funkcji **printf()**

Opis znaku	Zapis w printf()
Alarm (ang. *alert*), głośniczek wydaje dźwięk	`\a`
Backspace	`\b`
Wysunięcie strony (ang. *form feed*)	`\f`
Przejście do nowego wiersza (ang. *new line*)	`\n`
CR - Carriage Return (powrót na początek wiersza)	`\r`
Tabulacja pozioma (odstęp) (ang. *horizontal tab*)	`\t`
Tabulacja pionowa (ang. *vertical tab*)	`\v`

Istnieją także znaki, które pełnią specjalną funkcję w kodzie źródłowym i nie można ich wyświetlić w tradycyjny sposób. Znaki te oraz sposób ich zapisu w łańcuchu znaków zostały przedstawione w Tabeli 3.

Tabela 3. Wyświetlenie specjalnych znaków w funkcji **printf()**

Opis znaku	Znak	Zapis w printf()
Cudzysłów	"	`\"`
Apostrof	'	`\'`
Ukośnik (ang. *backslash*)	\	`\\`
Procent	%	`%%`

2.10. Komentarze

Komentarze służą do opisywania kodu źródłowego programu i są pomijane podczas jego kompilacji. Komentarz w języku C rozpoczyna się sekwencją znaków */**, a kończy sekwencją ***/*. Komentarz taki może obejmować więcej niż jedną linię kodu programu, np.

```
/* To jest tekst komentarza w pierwszej linii
   A to jest dalsza część komentarza          */
```

Zastosowanie sekwencji znaków // umożliwia wstawienie komentarza obejmującego tekst tylko do końca bieżącej linii kodu, np.

```
// Tekst komentarza do końca linii
```

W komentarzu, na początku kodu programu, bardzo często umieszcza się informacje o autorze programu, dacie jego powstania i przeznaczeniu.

```
/*
  Nazwa: MyApp.c
  Autor: Jarosław Forenc, Politechnika Białostocka
  Data:  01-10-2021 20:00
  Opis:  Program wyświetlający tekst "Witaj swiecie"
*/

#include <stdio.h>     // zawiera deklarację printf()
#include <stdlib.h>    // zawiera deklarację system()

int main(void)         // nagłówek funkcji main()
{
    printf("Witaj swiecie\n");

    system("pause");   // zatrzymanie programu
    return 0;
}
```

2.11. Najczęściej popełniane błędy podczas pisania programów

Podczas pisania programów komputerowych można popełnić dwa rodzaje błędów: składniowe i semantyczne. Błędy składniowe to nieprzestrzeganie zasad języka C. Błędy te są wykrywane przez kompilator, który zatrzymuje kompilację programu i wyświetla odpowiednie komunikaty. Błędy semantyczne nie są wykrywane przez kompilator. Polegają one na stosowaniu zasad języka C, ale w niewłaściwym celu (program nie działa tak jak tego oczekiwaliśmy).

W początkowej fazie nauki programowania większość popełnianych błędów są to literówki oraz błędy składni. Pouczającym może być samodzielne zrobienie błędów i zaobserwowanie reakcji kompilatora na nie.

```
1  #include <studio.h>
2
3  int main(void)
4  {
5     printf("Witaj swiecie\n");
6     return 0;
7  }
```

- błędna nazwa pliku nagłówkowego
- zamiast stdio.h jest studio.h

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp\myapp.cpp(1) : fatal error C1083: Cannot open include file: 'studio.h':
                                No such file or directory
1>Build log was saved at "file://d:\MyApp\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Liczba w nawiasie po nazwie pliku: d:\myapp\myapp\myapp.cpp(1) oznacza numer wiersza, w którym wystąpił błąd.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     printf("Witaj swiecie\n")
6     return 0;
7  }
```

- brak średnika na końcu wiersza

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp\myapp.cpp(6) : error C2143: syntax error : missing ';' before 'return'
1>Build log was saved at "file://d:\MyApp\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     printf("Witaj swiecie\n");
6     return 0;
7
```

- brak nawiasu klamrowego kończącego program

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp\myapp.cpp(7) : fatal error C1075: end of file found before the left brace '{' at
                                'd:\myapp\myapp\myapp.cpp(4)' was matched
1>Build log was saved at "file://d:\MyApp\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     printf("Witaj swiecie\n);
6     return 0;
7  }
```

- brak cudzysłowu kończącego łańcuch

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp\myapp.cpp(5) : error C2001: newline in constant
1>d:\myapp\myapp\myapp.cpp(6) : error C2143: syntax error : missing ')' before 'return'
1>Build log was saved at "file://d:\MyApp\MyApp\Debug\BuildLog.htm"
1>MyApp - 2 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int main
4  {
5     printf("Witaj swiecie\n");
6     return 0;
7  }
```

- brak nawiasów po funkcji **main**

```
1>------ Rebuild All started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp.cpp(4) : error C2470: 'main' : looks like a function definition,
                          but there is no parameter list; skipping apparent body
1>Build log was saved at "file://d:\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
========== Rebuild All: 0 succeeded, 1 failed, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     printf("Witaj swiecie\n");
6     return0;
7  }
```

- brak spacji pomiędzy **return** i **0**

```
1>------ Rebuild All started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp.cpp(6) : error C2065: 'return0' : undeclared identifier
1>Build log was saved at "file://d:\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
========== Rebuild All: 0 succeeded, 1 failed, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     Printf("Witaj swiecie\n");
6     return 0;
7  }
```

- nazwa funkcji **printf** pisana wielką literą

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp.cpp(5) : error C3861: 'Printf': identifier not found
1>Build log was saved at "file://d:\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int Main(void)
4  {
5     printf("Witaj swiecie\n");
6     return 0;
7  }
```

- nazwa funkcji **main** pisana wielką literą

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>Linking...
1>MSVCRTD.lib(crtexe.obj) : error LNK2019: unresolved external symbol _main referenced
                            in function ___tmainCRTStartup
1>d:\MyApp\Debug\MyApp.exe : fatal error LNK1120: 1 unresolved externals
1>Build log was saved at "file://d:\MyApp\MyApp\Debug\BuildLog.htm"
1>MyApp - 2 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

```
1  #include <stdio.h>
2
3  int main(void);
4  {
5     printf("Witaj swiecie\n");
6     return 0;
7  }
```

- średnik w nagłówku funkcji **main**

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>d:\myapp\myapp.cpp(4) : error C2447: '{' : missing function header (old-style formal list?)
1>Build log was saved at "file://d:\MyApp\Debug\BuildLog.htm"
1>MyApp - 1 error(s), 0 warning(s)
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     printf("Witaj swiecie\n");
6     return 0;
7  }
```

- wybrano niewłaściwy typ projektu

```
1>------ Build started: Project: MyApp, Configuration: Debug Win32 ------
1>Compiling...
1>MyApp.cpp
1>Linking...
1>MSVCRTD.lib(crtexew.obj) : error LNK2019: unresolved external symbol _WinMain@16
                             referenced in function ___tmainCRTStartup
1>d:\MyApp\Debug\MyApp.exe : fatal error LNK1120: 1 unresolved externals
1>Build log was saved at "file://d:\MyApp\Debug\BuildLog.htm"
1>MyApp - 2 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Tabela 4. Tłumaczenia wybranych komunikatów kompilatora

Komunikat	Tłumaczenie
fatal error C1083: Cannot open include file: 'studio.h': No such file or directory	błąd krytyczny C1083: Nie można otworzyć dołączanego pliku 'studio.h': Nie ma takiego pliku lub folderu (błędna nazwa pliku nagłówkowego)
error C2143: syntax error : missing ';' before 'return'	błąd C2143: błąd składni: brakujący ';' przed 'return' (na końcu wiersza powyżej return brakuje średnika)
fatal error C1075: end of file found before the left brace '{' at 'MyApp.cpp(4)' was matched	błąd krytyczny C1075: napotkano koniec pliku przed dopasowaniem nawiasu do lewego nawiasu '{' w 'MyApp.cpp(4)' (brakuje nawiasu zamykającego na końcu pliku)
error C3861: 'Printf': identifier not found	błąd C3861: 'Printf': nie znaleziono identyfikatora (nazwa funkcji napisana wielką literą - powinno być 'printf')

error C2001: newline in constant error C2143: syntax error : missing ')' before 'return'	błąd C2001: nowa linia w stałej błąd C2143: błąd składni: brakujący ')' przed 'return' (komunikaty mogą być niezrozumiałe - w rzeczywistości brakowało cudzysłowu kończącego łańcuch znaków)
error C2470: 'main' : looks like a function definition, but there is no parameter list; skipping apparent body	błąd C2470: 'main': wygląda jak definicja funkcji ale brakuje listy parametrów; omijam pozorne ciało (funkcji) (po nazwie funkcji 'main' zabrakło nawiasów: '(' i ')')
error C2065: 'return0' : undeclared identifier	błąd C2065: 'return0': niezadeklarowany identyfikator (brak spacji pomiędzy 'return' i '0')

3. Przebieg ćwiczenia

Na pracowni specjalistycznej należy wykonać wybrane zadania wskazane przez prowadzącego zajęcia. W różnych grupach mogą być wykonywane różne zadania.

1. Wykonaj poniższe polecenia:
   a) utwórz nowy projekt w środowisku Microsoft Visual C++ 2008 - jako typ projektu (szablonu) wybierz **Win32 Console Application**;
   b) dodaj do projektu nowy plik;
   c) wprowadź kod źródłowy programu wyświetlającego tekst **„Witaj swiecie”**;
   d) skompiluj i uruchom program;
   e) odszukaj na dysku katalog zawierający skompilowany plik wynikowy **exe**; uruchom program z poziomu systemu operacyjnego; sprawdź, czy można zaobserwować wyniki jego działania;
   f) dodaj do programu instrukcję zatrzymującą program przed zakończeniem jego działania (**system(”pause”)** lub **getch()**); sprawdź, czy uruchomienie programu z poziomu systemu operacyjnego pozwoli zobaczyć wyniki jego działania;

g) przekształć kod źródłowy programu tak, aby zajmował jak najmniej wierszy; skompiluj i uruchom program

h) katalog zawierający pliki stworzonego projektu skopiuj na pendrive lub skopiuj do innego katalogu na dysku lub spakuj i wyślij do siebie e-mailem.

2. Napisz program wyświetlający na ekranie wizytówkę o poniższej postaci. Pamiętaj o ramce z gwiazdek.

```
**********************************
*          Jan Kowalski          *
*  e-mail: j.kowalski@gmail.com  *
*        tel. 123-456-789        *
**********************************
```

3. Sprawdź efekt umieszczenia w łańcuchu formatującym funkcji **printf()** znaków: **\n**, **\t**, **\a**, **\b**, **\r**, **\f**.

4. Stosując funkcję **printf()** wyświetl na ekranie następujące znaki: cudzysłów (**"**), apostrof (**'**), ukośnik (\), procent (**%**).

5. Wywołaj trzykrotnie funkcję **printf()** z argumentami będącymi poniższymi łańcuchami znaków.

```
"61 62 63 64 65\n"
"\061 \062 \063 \064 \065\n"
"\x61 \x62 \x63 \x64 \x65\n"
```

Zinterpretuj znaki wyświetlane w każdym wierszu.

6. W dowolnym programie w języku C wprowadź zmiany powodujące min. 3 błędy kompilacji. Dla każdego błędu podaj: kod źródłowy zwierający błąd, otrzymany komunikat kompilatora, tłumaczenie komunikatu na język polski, wyjaśnienie na czym polegał błąd. Postaraj się, aby błędy miały inne numery niż przedstawione w instrukcji do zajęć.

4. Literatura

[1] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.

[2] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.

[3] Deitel P.J., Deitel H.: Język C. Solidna wiedza w praktyce. Wydanie VIII. Helion, Gliwice, 2020.

[4] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.

[5] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.

[6] Wileczek R.: Microsoft Visual C++ 2008. Tworzenie aplikacji dla Windows. Helion, Gliwice, 2009.

[7] http://www.cplusplus.com/reference/clibrary - C library - C++ Reference

[8] https://cpp0x.pl/dokumentacja/standard-C/1 - Standard C

[9] https://visualstudio.microsoft.com/pl/ - Microsoft Visual Studio

5. Pytania kontrolne

1. Omów sposób tworzenia projektu, kompilacji oraz uruchamiania programu w środowisku Visual C++.
2. Na wybranym przykładzie omów ogólną strukturę programu w języku C.
3. Wyjaśnij, do czego służą pliki nagłówkowe?
4. Opisz proces tworzenia pliku wynikowego (**exe**) z pliku źródłowego w języku C.

6. Wymagania BHP

Warunkiem przystąpienia do praktycznej realizacji ćwiczenia jest zapoznanie się z instrukcją BHP i instrukcją przeciw pożarową oraz przestrzeganie zasad w nich zawartych.

W trakcie zajęć laboratoryjnych należy przestrzegać następujących zasad.

- Sprawdzić, czy urządzenia dostępne na stanowisku laboratoryjnym są w stanie kompletnym, nie wskazującym na fizyczne uszkodzenie.
- Jeżeli istnieje taka możliwość, należy dostosować warunki stanowiska do własnych potrzeb, ze względu na ergonomię. Monitor komputera ustawić w sposób zapewniający stałą i wygodną obserwację dla wszystkich członków zespołu.
- Sprawdzić prawidłowość połączeń urządzeń.
- Załączenie komputera może nastąpić po wyrażeniu zgody przez prowadzącego.
- W trakcie pracy z komputerem zabronione jest spożywanie posiłków i picie napojów.
- W przypadku zakończenia pracy należy zakończyć sesję przez wydanie polecenia wylogowania. Zamknięcie systemu operacyjnego może się odbywać tylko na wyraźne polecenie prowadzącego.
- Zabronione jest dokonywanie jakichkolwiek przełączeń oraz wymiana elementów składowych stanowiska.
- Zabroniona jest zmiana konfiguracji komputera, w tym systemu operacyjnego i programów użytkowych, która nie wynika z programu zajęć i nie jest wykonywana w porozumieniu z prowadzącym zajęcia.
- W przypadku zaniku napięcia zasilającego należy niezwłocznie wyłączyć wszystkie urządzenia.
- Stwierdzone wszelkie braki w wyposażeniu stanowiska oraz nieprawidłowości w funkcjonowaniu sprzętu należy przekazywać prowadzącemu zajęcia.

- Zabrania się samodzielnego włączania, manipulowania i korzystania z urządzeń nie należących do danego ćwiczenia.
- W przypadku wystąpienia porażenia prądem elektrycznym należy niezwłocznie wyłączyć zasilanie stanowiska. Przed odłączeniem napięcia nie dotykać porażonego.