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: **EDS1B 1007**

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 Community 2019 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, serwisów internetowych oraz aplikacji mobilnych.

Visual Studio wspiera bardzo dużo języków programowania. Do wbudowanych języków należą m.in. C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS. Natomiast wsparcie dla języków takich jak Python, Ruby, Node.js i M dobywa się poprzez odpowiednie pluginy.

Środowisko dostępne jest obecnie w trzech edycjach:

- Microsoft Visual Studio Community 2022;
- Microsoft Visual Studio Professional 2022;
- Microsoft Visual Studio Enterprise 2022.

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 Community 2019.

2.2. Microsoft Visual Studio Community 2019

Wersja Community środowiska Microsoft Visual Studio 2019 jest dostępna bezpłatnie dla indywidualnych deweloperów, do zastosowań akademickich i rozwiązań typu open source.

Według dokumentacji program może być zainstalowany i uruchomiony w następujących systemach operacyjnych (zalecane są wersje 64-bitowe):

- Windows 10 w wersji 1703 lub nowszej: Home, Professional, Education i Enterprise;
- Windows Server 2016 / 2019: Standard i Datacenter;
- Windows 8.1: Core, Professional i Enterprise;
- Windows Server 2012 R2: Essentials, Standard, Datacenter;
- Windows 7 z dodatkiem SP1: Home Premium, Professional, Enterprise, Ultimate.

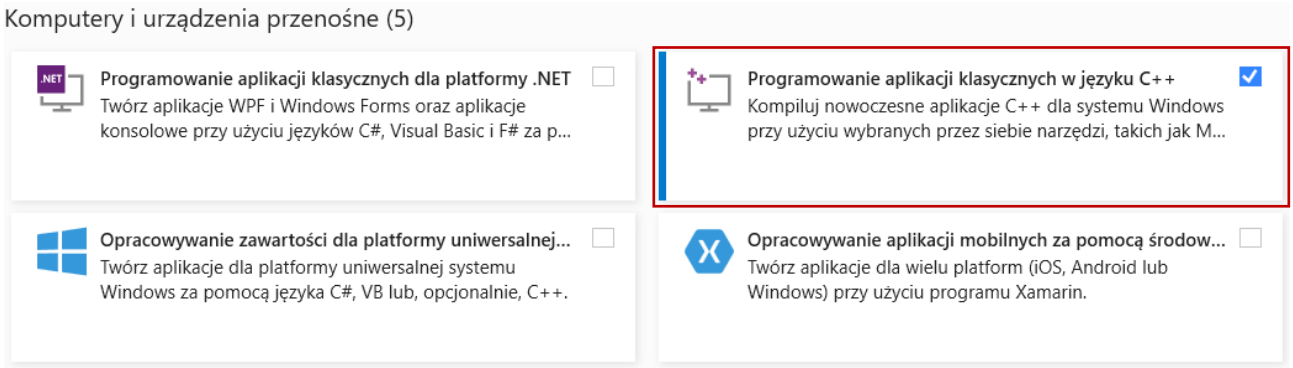
Do wymagań sprzętowych środowiska należą:

- procesor 1,8 GHz lub szybszy (zalecany czterordzeniowy lub lepszy);
- 2 GB pamięci RAM; zalecane 8 GB pamięci RAM;
- miejsce na dysku twardym: minimalnie 800 MB, a maksymalnie 210 GB dostępnego miejsca (w zależności od instalowanych funkcji); typowe instalacje wymagają 20–50 GB wolnego miejsca;
- dysk twardy: aby zwiększyć wydajność, system Windows i program Visual Studio należy zainstalować na dysku SSD;
- karta wideo obsługująca rozdzielczość ekranu co najmniej 720p (1280 x 720); program Visual Studio będzie działać najlepiej przy rozdzielczości WXGA (1366 x 768) lub wyższej.

Program Visual Studio jest dostępny w następujących językach: angielskim, chińskim (uproszczony i tradycyjny), czeskim, francuskim, niemieckim, włoskim, japońskim, koreańskim, polskim, portugalskim (Brazylia), rosyjskim, hiszpańskim i tureckim. Język programu Visual Studio można wybrać podczas instalacji. Instalator programu Visual Studio jest dostępny w tych samych czternastu językach.

Do zainstalowania programu Visual Studio są wymagane uprawnienia administratora.

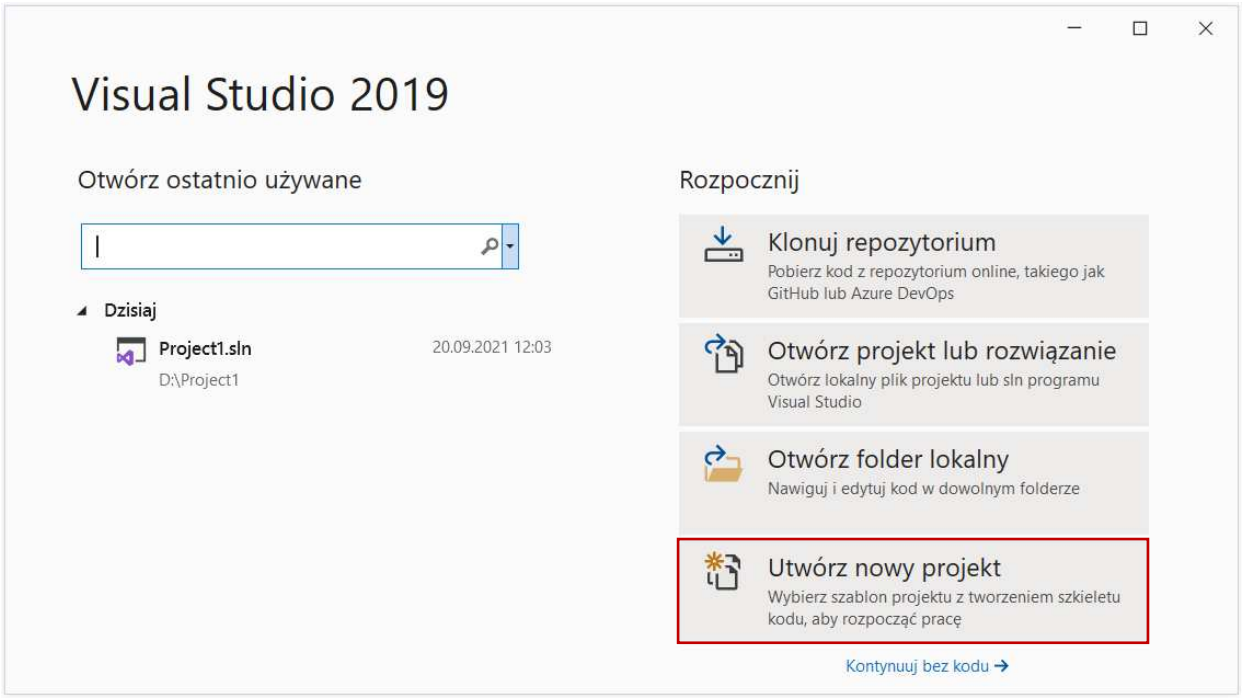
Wykorzystanie środowiska Visual Studio do pisania programów na pracowni specjalistycznej z przedmiotu **Informatyka** wymaga podczas instalacji zaznaczenia elementu: **Programowanie aplikacji klasycznych w języku C++** (Rys. 1).



Rys. 1 Fragment okna instalatora programu Microsoft Visual Studio 2019

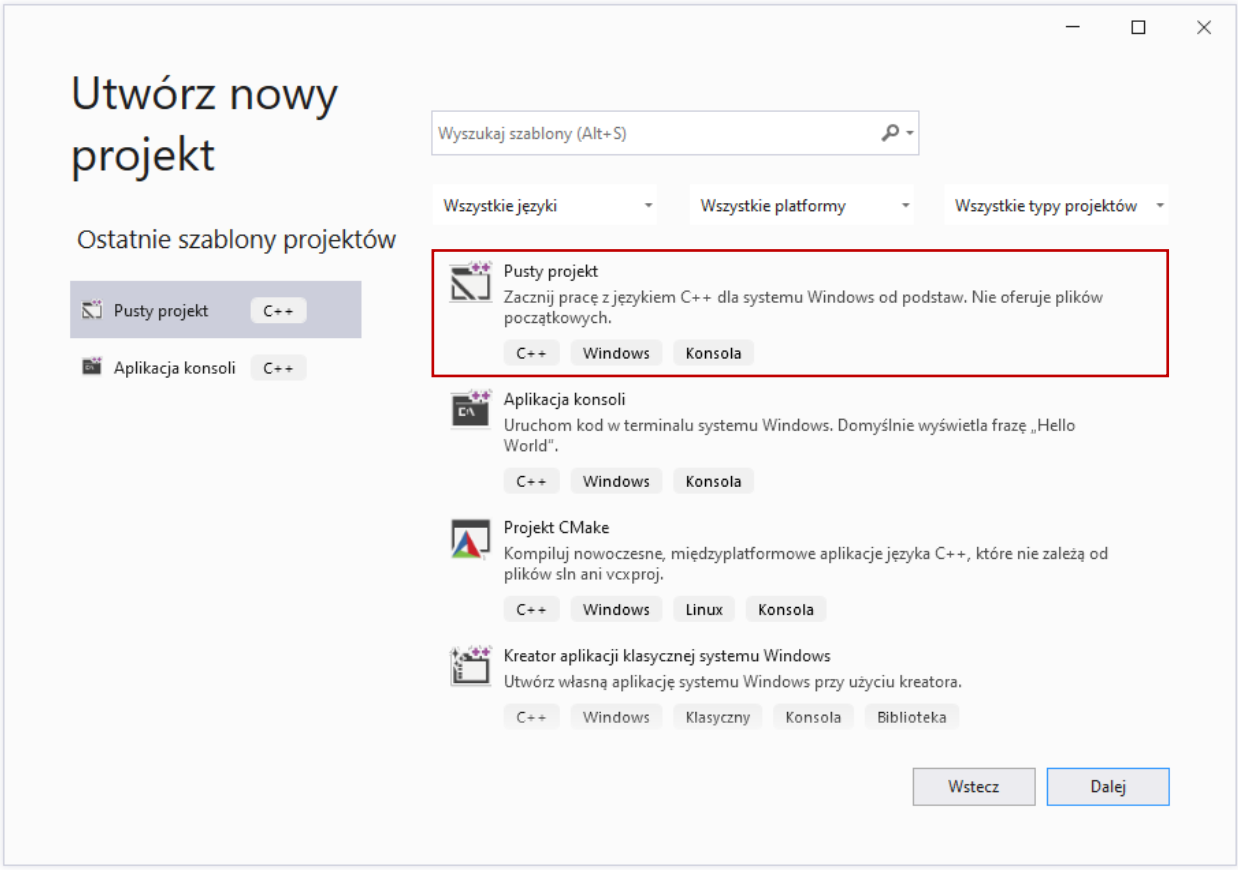
2.3. Utworzenie nowego projektu w Visual Studio 2019

Rozpoczęcie pracy ze środowiskiem Visual Studio 2019 wymaga utworzenia nowego projektu. Po uruchomieniu programu na ekranie pojawia się okno pokazane na Rys. 2. W oknie tym wybieramy opcję: **Utwórz nowy projekt**



Rys. 2 Rozpoczęcie pracy z programem Visual Studio 2019

Jako typ projektu wskazujemy: **Pusty projekt** (Rys. 3) i naciskamy przycisk **Dalej**.

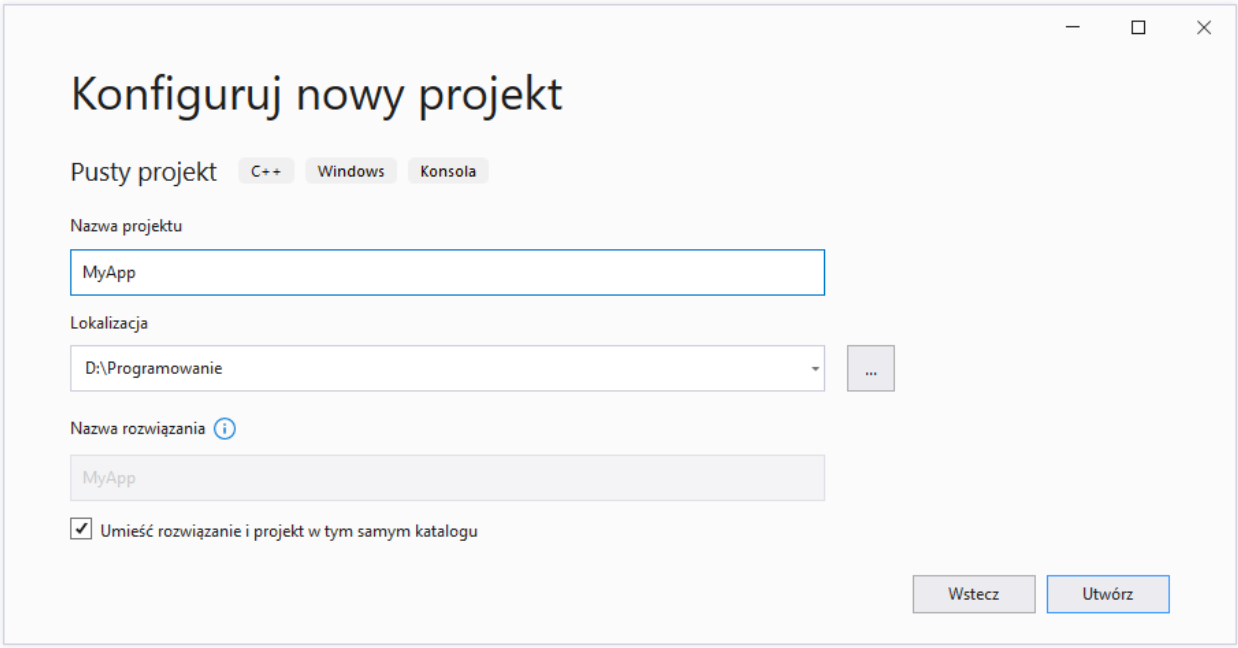


Rys. 3 Wybieranie szablonu nowego projektu

W kolejnym kroku wprowadzamy (Rys. 4):

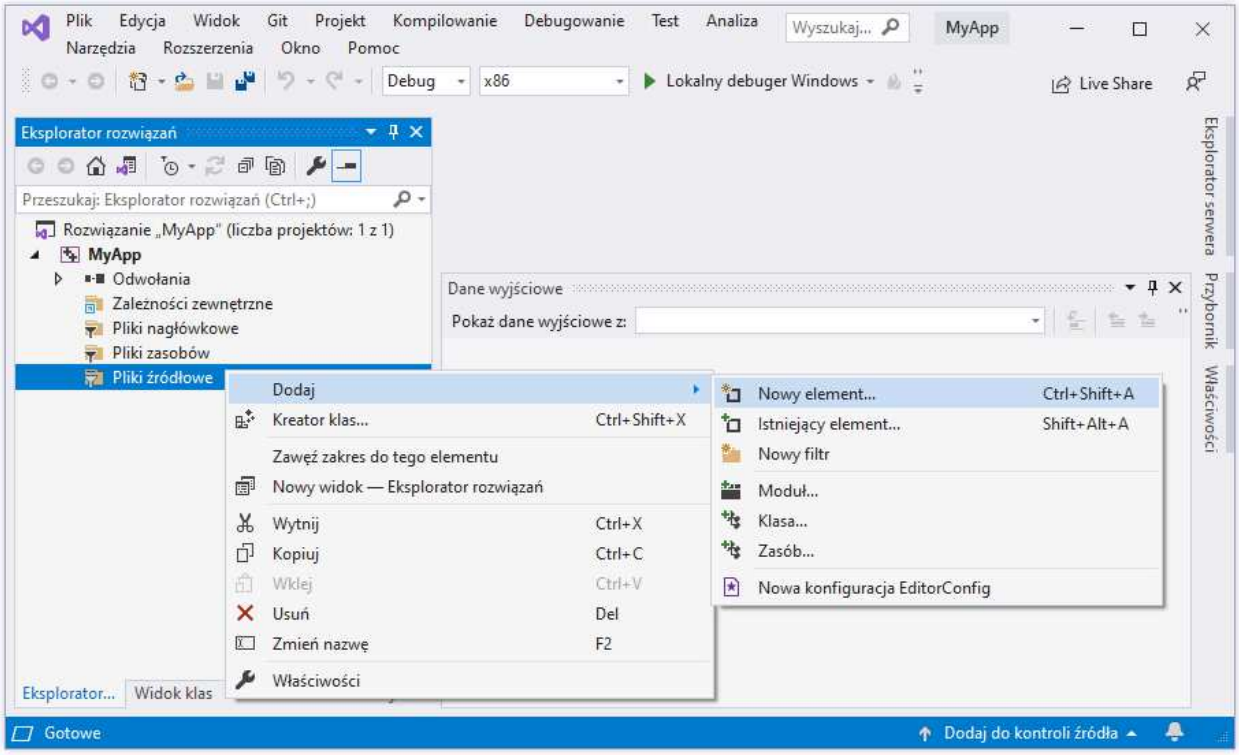
- nazwę naszego projektu (**Nazwa projektu**), która będzie także nazwą pliku wynikowego **exe**;
- położenie plików projektu (**Lokalizacja**);
- opcjonalnie nazwę rozwiązania (**Nazwa rozwiązania**).

Rozwiązanie (ang. *Solution*) może składać się z wielu projektów. Każdy taki projekt jest samodzielnym programem. Wszystkie projekty wchodzące w skład rozwiązania mogą być skompilowane po wybraniu tylko jednego polecenia. Zazwyczaj projekty wchodzące w skład rozwiązania są ze sobą w pewien sposób powiązane, np. rozwiązują ten sam problem, ale różnymi metodami. Zaznaczenie opcji **Umieść rozwiązanie i projekt w tym samym katalogu** spowoduje, że rozwiązanie będzie składało się z tylko jednego projektu. Tworzenie nowego projektu kończymy wciskając przycisk **Utwórz**.



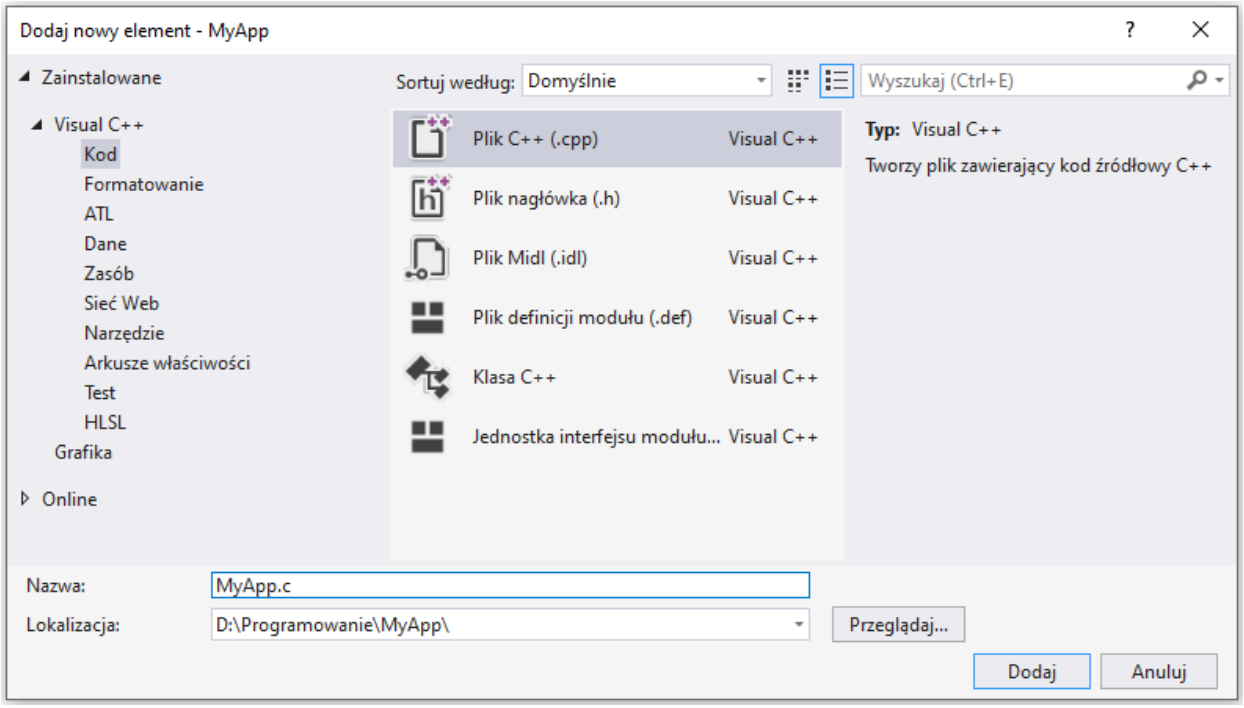
Rys. 4 Konfiguracja nowego projektu

Utworzony projekt jest pusty (nie zawiera żadnego pliku, w którym można wpisać kod programu). Nowy plik można dodać do projektu używając klawisza skrótu: **Ctrl + Shift + A** lub klikając prawym klawiszem myszki w oknie **Eksplorator rozwiązań** na **Pliki źródłowe** i wybierając **Dodaj → Nowy element…** (Rys. 5).



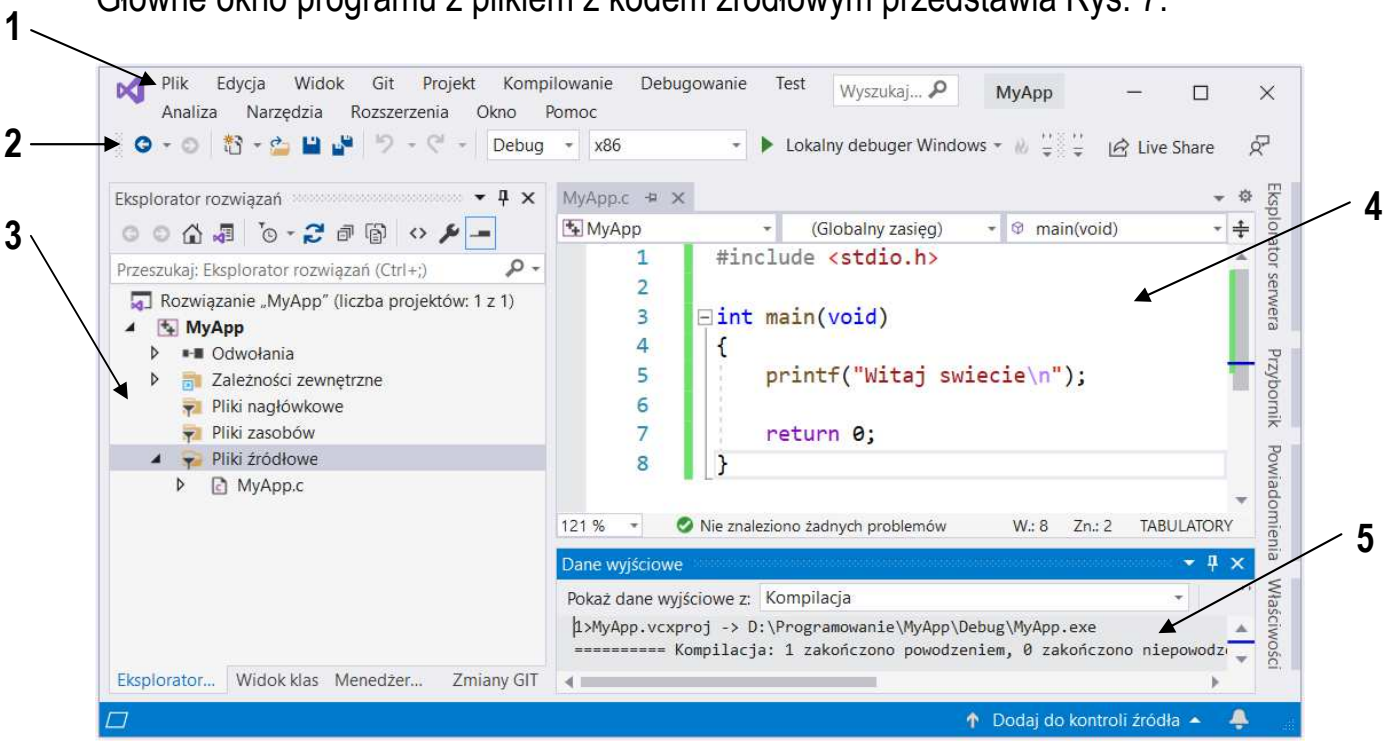
Rys. 5 Dodanie nowego pliku do projektu

Jako typ pliku wybieramy **Plik C++ (.cpp)**, a następnie wprowadzamy nazwę pliku (**Nazwa**) z rozszerzeniem **.c** (nie **.cpp**) i ewentualnie miejsce jego zapisania na dysku (**Lokalizacja**) (Rys. 6).



Rys. 6 Wprowadzenie nazwy pliku i miejsca zapisania na dysku

Główne okno programu z plikiem z kodem źródłowym przedstawia Rys. 7.



Rys. 7 Główne okno środowiska Microsoft Visual Studio 2019

Elementy głównego okna programu:

1 - Menu główne;

2 - Pasek narzędziowy;

3 - *Eksplorator rozwiązań* - zawiera m.in. informacje o aktualnym rozwiązaniu i projekcie;

4 - Edytor kodu;

5 - *Dane wyjściowe* - zawiera m.in. wyniki kompilacji programu.

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:

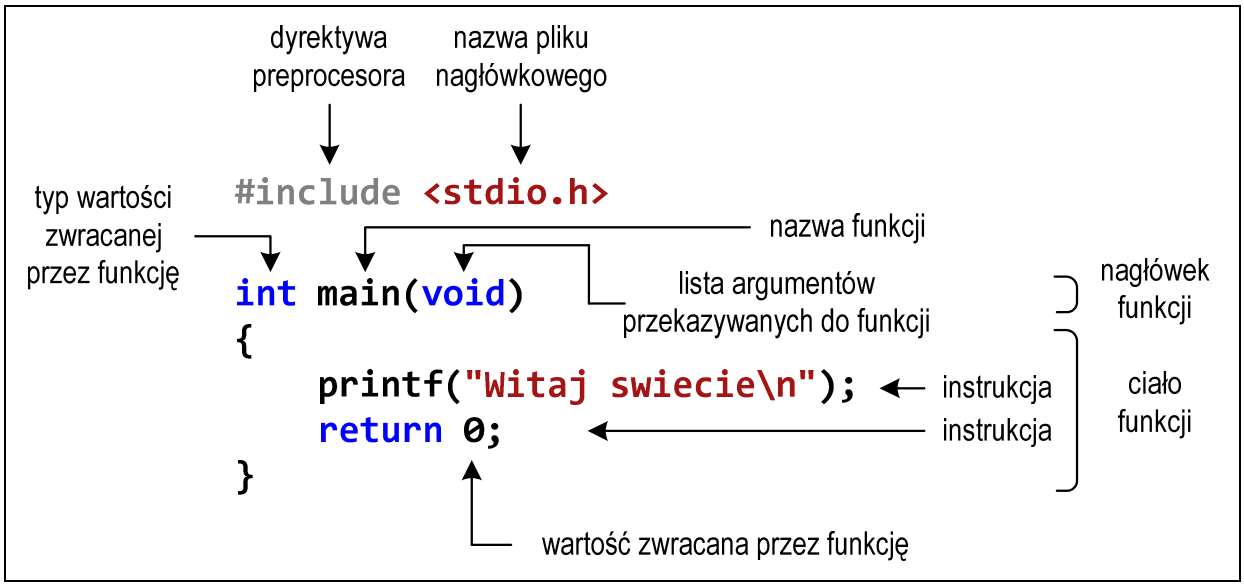
```
#include <stdio.h>

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

Program w języku C (Rys. 8) 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. 8. 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*).

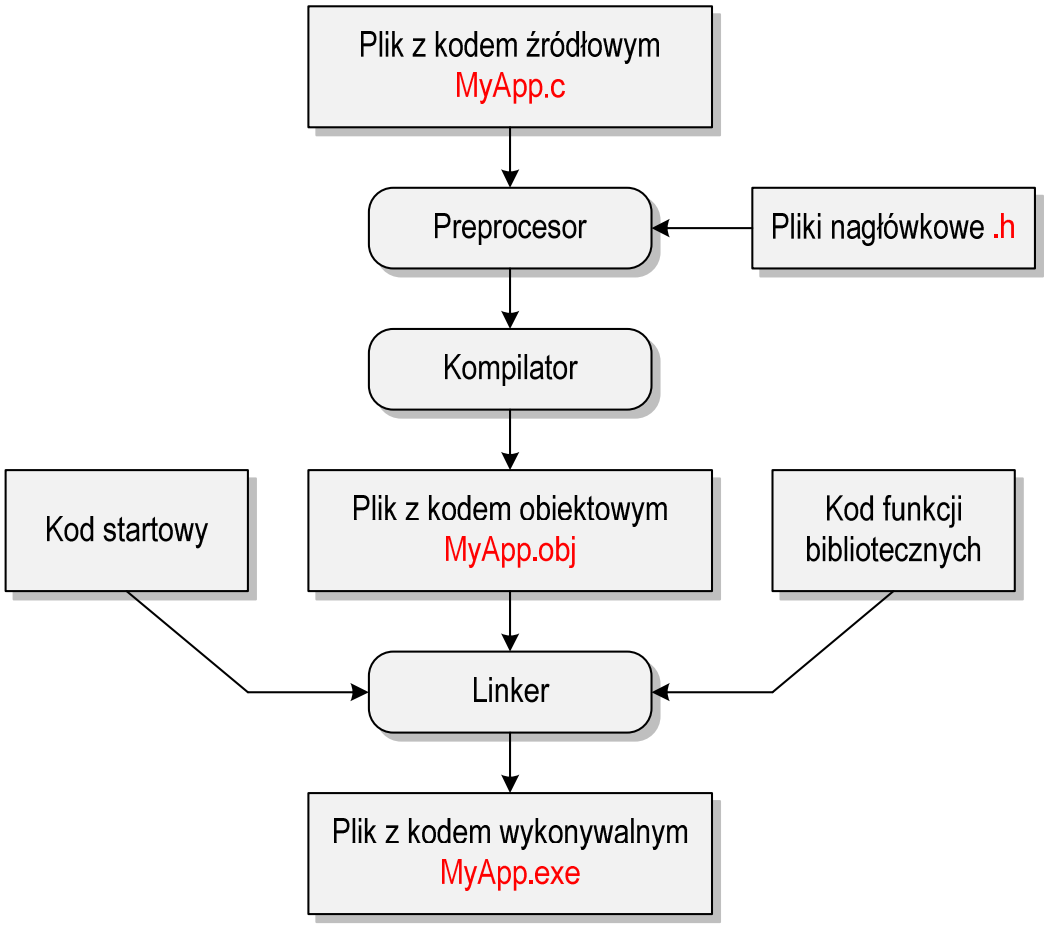
Funkcję wywołuje się podając jej nazwę (**printf()**) i (w nawiasach zwykłych) listę argumentów przekazywanych do funkcji (**"Witaj swiecie\n"**). W powyższym przykładzie do funkcji **main()** nie są przekazywane żadne argumenty, informuje o tym słowo **void** w jej nagłówku (słowo to może być pominięte).

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 kilku kroków (Rys. 9).



Rys. 9. 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 Microsoft Visual Studio 2019 do utworzenia pliku wykonywalnego można wybrać jedną z pozycji znajdujących się w menu głównym **Kompilowanie** (Rys. 10).

Rys. 10. Menu główne **Kompilowanie**

Najważniejsze pozycje z menu **Kompilowanie** wykonują następujące operacje:

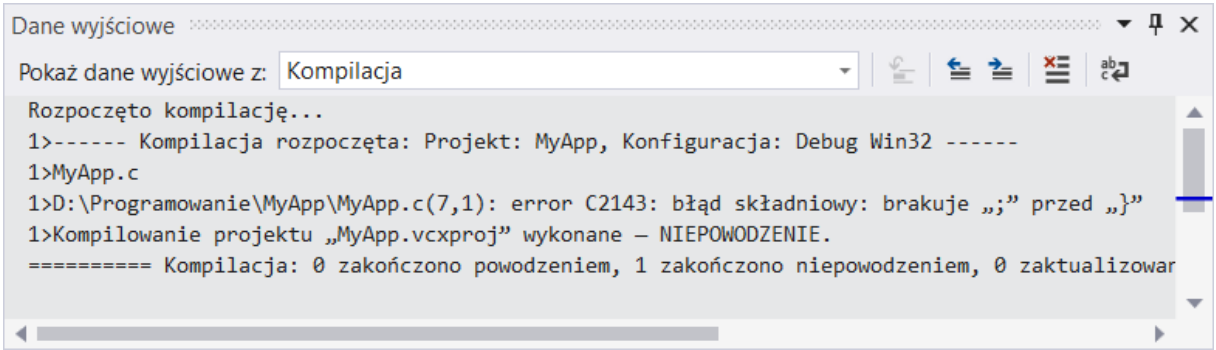
- **Kompiluj rozwiązanie (F7) -** buduje wszystkie projekty znajdujące się w rozwiązaniu (kompilowane są tylko te pliki, które uległy zmianie od czasu ostatniej kompilacji);
- **Kompiluj ponownie rozwiązanie (Ctrl + Alt + F7) -** przebudowuje wszystkie projekty znajdujące się w rozwiązaniu (kompilowane są wszystkie pliki niezależnie od tego czy uległy zmianie od czasu ostatniej kompilacji);
- **Wyczyść rozwiązanie** - usuwa wszystkie pliki będące wynikiem budowania projektów wchodzących w skład rozwiązania;
- **Kompiluj *MyApp* (Ctrl + B)** - buduje aktywny projekt (o nazwie ***MyApp***);
- **Kompiluj ponownie *MyApp*** - przebudowuje aktywny projekt;

- **Wyczyść *MyApp*** - usuwa wszystkie pliki będące wynikiem budowania aktywnego projektu;
- **Tylko projekt** - podmenu zawierające pozycje przeznaczone do budowania (**Tylko kompilacja *MyApp***), przebudowania (**Tylko rekompilacja *MyApp***), czyszczenia (**Tylko oczyszczenie *MyApp***) i linkowania (**Tylko konsolidacja *MyApp***) tylko aktualnego projektu;
- **Kompilacja wsadowa…** - otwiera okno budowania wsadowego;
- **Menedżer konfiguracji…** - otwiera okno menadżera konfiguracji projektu;
- **Kompiluj (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 **Kompiluj rozwiązanie**. Można wtedy użyć klawisza skrótu **F7**. Przykładowy przebieg kompilacji przedstawiony jest poniżej.

```
Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>MyApp.vcxproj -> D:\Programowanie\MyApp\Debug\MyApp.exe
========== Kompilacja: 1 zakończono powodzeniem, 0 zakończono niepowodzeniem,
                        0 zaktualizowano, 0 pominięto ==========
```

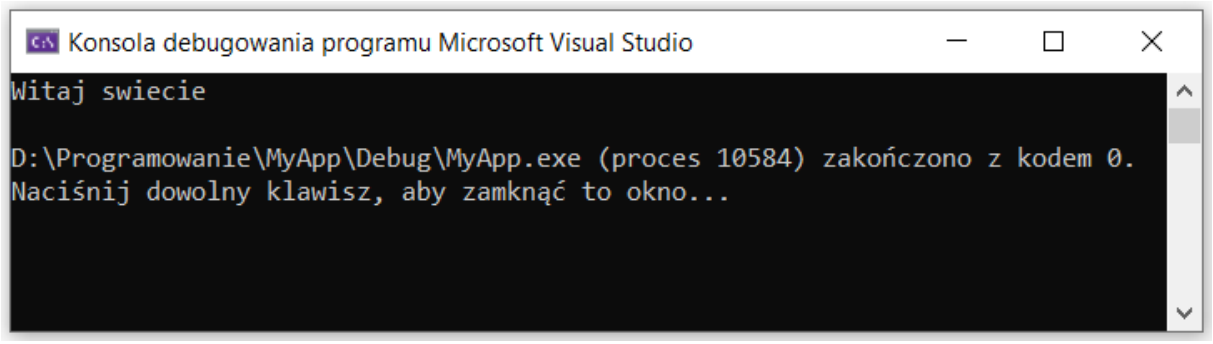
Jeśli w kodzie programu występują błędy, to w oknie **Dane wyjściowe** wyświetlane są odpowiednie komunikaty (Rys. 11), natomiast plik wykonywalny **exe** nie jest tworzony.



Rys. 11 Okno z błędami kompilacji

Skompilowany program uruchomia się poprzez wybranie pozycji **Uruchom bez debugowania** (**Ctrl + F5**) z menu głównego **Debugowanie**.

Uruchomienie programu odbywa się automatycznie w oknie **Wiersza polecenia** (Rys. 12). Przed uruchomieniem programu system operacyjny tworzy takie okno, a następnie uruchamia w nim program. Program wyświetla tekst „**Witaj swiecie**”. Dodatkowo środowisko Visual Studio zatrzymuje na koniec program i wyświetla komunikat: „**Naciśnij dowolny klawisz, aby zamknąć to okno…**”. Po naciśnięciu dowolnego klawisza okno **Wiersza polecenia** jest zamykane.



Rys. 12 Okno wiersza polecenia z uruchomionym programem

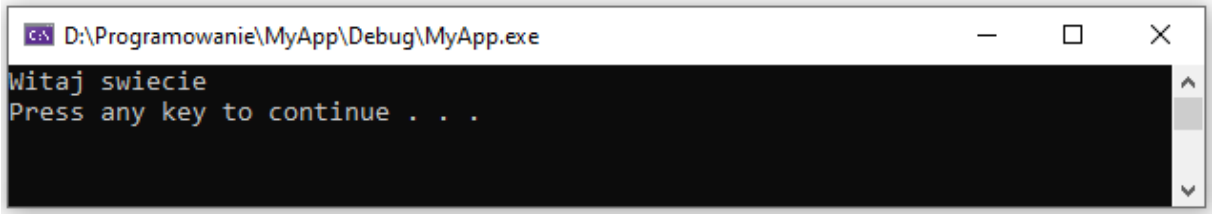
Jeśli uruchomimy skompilowany program z poziomu systemu operacyjnego, to nie zobaczymy efektów jego działania. System operacyjny utworzy okno, uruchomi w nim program, program zakończy się i okno zostanie natychmiast zamknięte. Aby zaobserwować wyniki pracy programu należy zatrzymać go przed zakończeniem jego działania. Można to zrobić wywołując, przed instrukcją **return**, 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 **Press any key to continue…** (Rys. 13).



Rys. 13 Zatrzymanie wykonywania programu poleceniem systemowym **pause**

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

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()**.

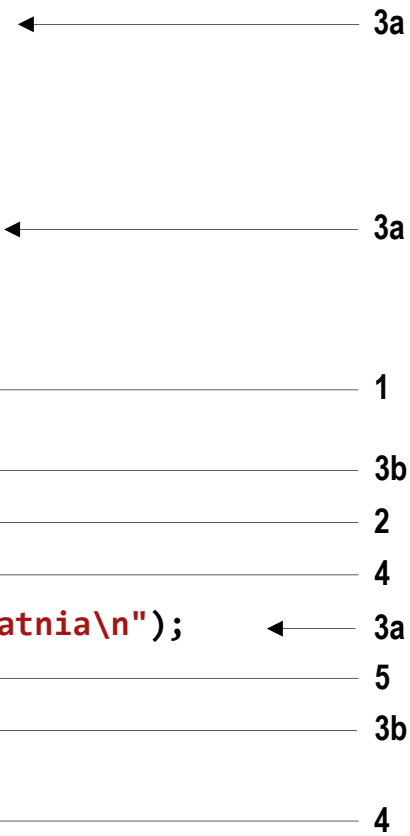
```c
#include <stdio.h>

void komunikat1(void)
{
    printf("Zaczynamy...\n");
}

void komunikat2(void)
{
    printf("Konczymy...\n");
}
int main(void)
{
    int x;

    komunikat1();
    x = 5;
    if (x > 0)
        printf("x to liczba dodatnia\n");
    ;
    komunikat2();

    return 0;
}
```



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()

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 1.

Tabela 1. 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`

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

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 2. 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  {
```

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

```
5       printf("Witaj swiecie\n");
6       return 0;
7   }
```

Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(1,10): fatal error C1083: Nie można otworzyć pliku dołącz: 'studio.h': No such file or directory
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto ==========

Liczby w nawiasie po nazwie pliku: **D:\Programowanie\Myapp\Myapp.c(1,10)** oznaczają miejsce w pliku (numer wiersza i kolumny), 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

Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(6,2): error C2143: błąd składniowy: brakuje „;” przed „return”
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto ==========

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

- brak nawiasu klamrowego kończącego program

```
Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(4): fatal error C1075: „{”: nie znaleziono zgodnego
                                                          tokenu
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem,
                       0 zaktualizowano, 0 pominięto ==========
```

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

- brak cudzysłowu kończącego łańcuch

```
Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(5,9): error C2001: w stałej występuje symbol przejścia
                                                    do następnego wiersza
1>D:\Programowanie\MyApp\MyApp.c(6,2): error C2143: błąd składniowy: brakuje „)”
                                                    przed „return”
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem,
                       0 zaktualizowano, 0 pominięto ==========
```

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

- brak nawiasów po funkcji **main**

```
Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(4,1): error C2054: oczekiwano "(" postępuj zgodnie
                                                    z "main"
```

1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem,
0 zaktualizowano, 0 pominięto ==========

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

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

Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(6,9): error C2065: "return0": niezadeklarowany identyfikator
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem,
0 zaktualizowano, 0 pominięto ==========

```
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

Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(5,8): warning C4013: niezdefiniowany "Printf"; przy założeniu, że extern zwraca int
1>MyApp.obj : error LNK2019: nierozpoznany symbol zewnętrzny _Printf przywołany w funkcji _main
1>D:\Programowanie\MyApp\Debug\MyApp.exe : fatal error LNK1120: liczba nierozpoznanych elementów zewnętrznych: 1
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem,
0 zaktualizowano, 0 pominięto ==========

```
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

Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>MSVCRTD.lib(exe_main.obj) : error LNK2019: nierozpoznany symbol zewnętrzny _main przywołany w funkcji "int __cdecl invoke_main(void)" (?invoke_main@@YAHXZ)
1>D:\Programowanie\MyApp\Debug\MyApp.exe : fatal error LNK1120: liczba nierozpoznanych elementów zewnętrznych: 1
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto ==========

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

- średnik w nagłówku funkcji **main**

Rozpoczęto kompilację...
1>------ Kompilacja rozpoczęta: Projekt: MyApp, Konfiguracja: Debug Win32 ------
1>MyApp.c
1>D:\Programowanie\MyApp\MyApp.c(4,1): error C2449: znaleziono „{” w zakresie pliku (brak nagłówka funkcji?)
1>D:\Programowanie\MyApp\MyApp.c(7,1): error C2059: błąd składniowy: „}”
1>Kompilowanie projektu „MyApp.vcxproj” wykonane — NIEPOWODZENIE.
========== Kompilacja: 0 zakończono powodzeniem, 1 zakończono niepowodzeniem, 0 zaktualizowano, 0 pominięto ==========

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 Studio - jako typ projektu (szablonu) wybierz **Pusty projekt**;
   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”)**); 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, 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] http://www.cplusplus.com/reference/clibrary - C library - C++ Reference

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

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

5. Pytania kontrolne

1. Omów sposób tworzenia projektu, kompilacji oraz uruchamiania programu w środowisku Visual Studio.
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.