Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Programowanie mikrokontrolerów w języku wysokiego poziomu 1

Kod przedmiotu: **TS1F1008**

(studia stacjonarne)

ŚRODOWISKO VISUAL STUDIO CODE. JĘZYK C - OGÓLNA STRUKTURA PROGRAMU

Numer ćwiczenia

PMC_01

Autor:
dr inż. Jarosław Forenc

Białystok 2023

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 10/11.

1.2. Oprogramowanie

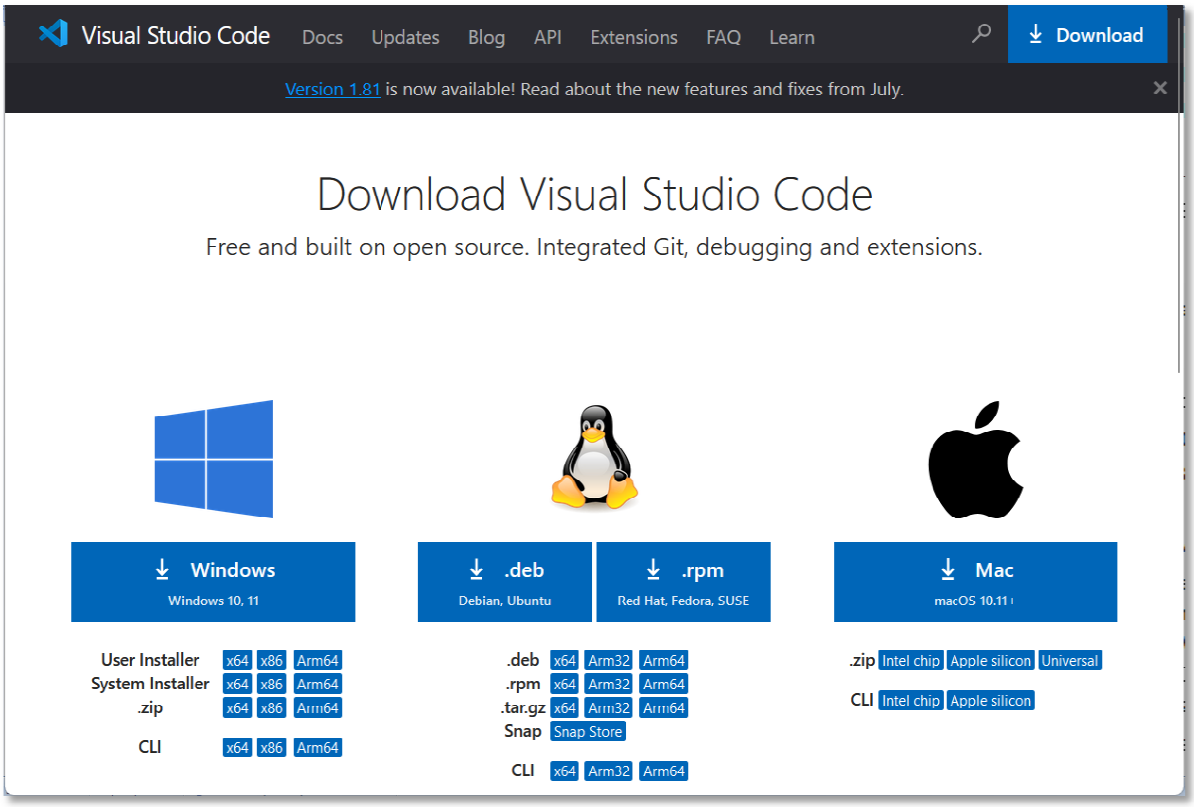
Na komputerach zainstalowany jest edytor kodu źródłowego Visual Studio Code 1.81 (lub nowszy) wraz z odpowiednimi rozszerzeniami (C/C++, Code Runner, Polish Language Pack for Visual Studio Code) oraz MinGW - zestaw kompilatorów różnych języków programowania (m.in. C, C++, Fortran, Java).

2. Visual Studio Code

2.1. Instalacja programu Visual Studio Code i jego rozszerzeń

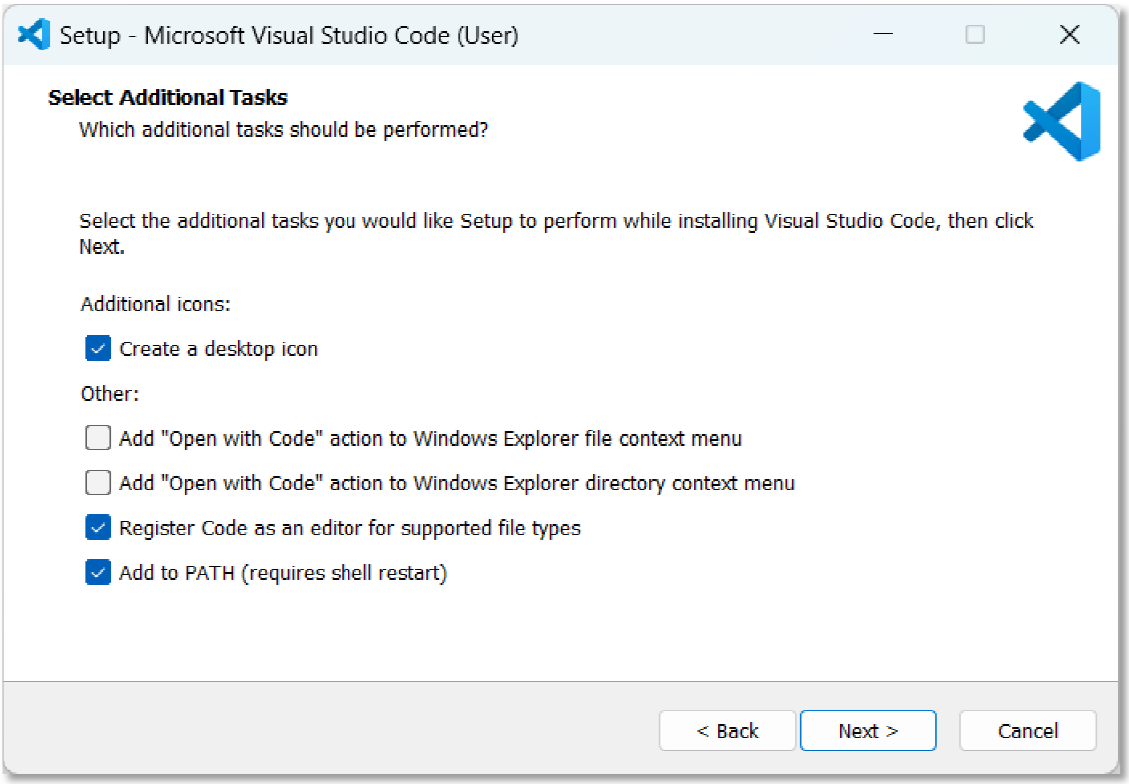
Visual Studio Code (VS Code) jest bezpłatnym edytorem kodu źródłowego opracowanym przez firmę Microsoft. Edytor ten dostępny jest dla systemów operacyjnych Microsoft Windows, macOS i Linux. Posiada wbudowaną obsługę JavaScript, TypeScript i Node.js oraz bogaty zestaw rozszerzeń dla innych języków programowania i środowisk uruchomieniowych, takich jak: C, C++, C#, Java, Python, PHP, Go, .NET.

Wersja instalacyjna edytora znajduje się na stronie internetowej https://code.visualstudio.com/download (Rys. 1). Plik instalacyjny zajmuje ok. 89 MB.



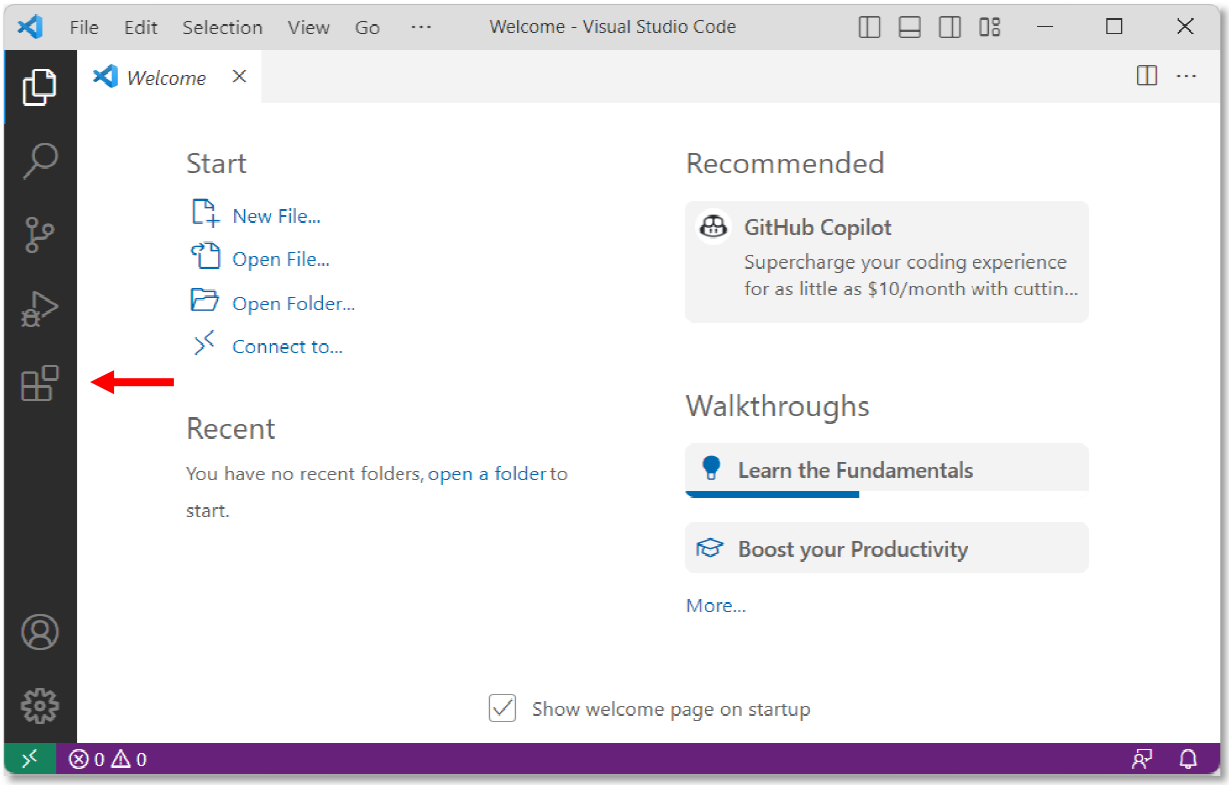
Rys. 1. Strona internetowa edytora Visual Studio Code

Po ściągnięciu pliku uruchamiamy instalację programu (Rys. 2).



Rys. 2. Rozpoczęcie instalacji VS Code

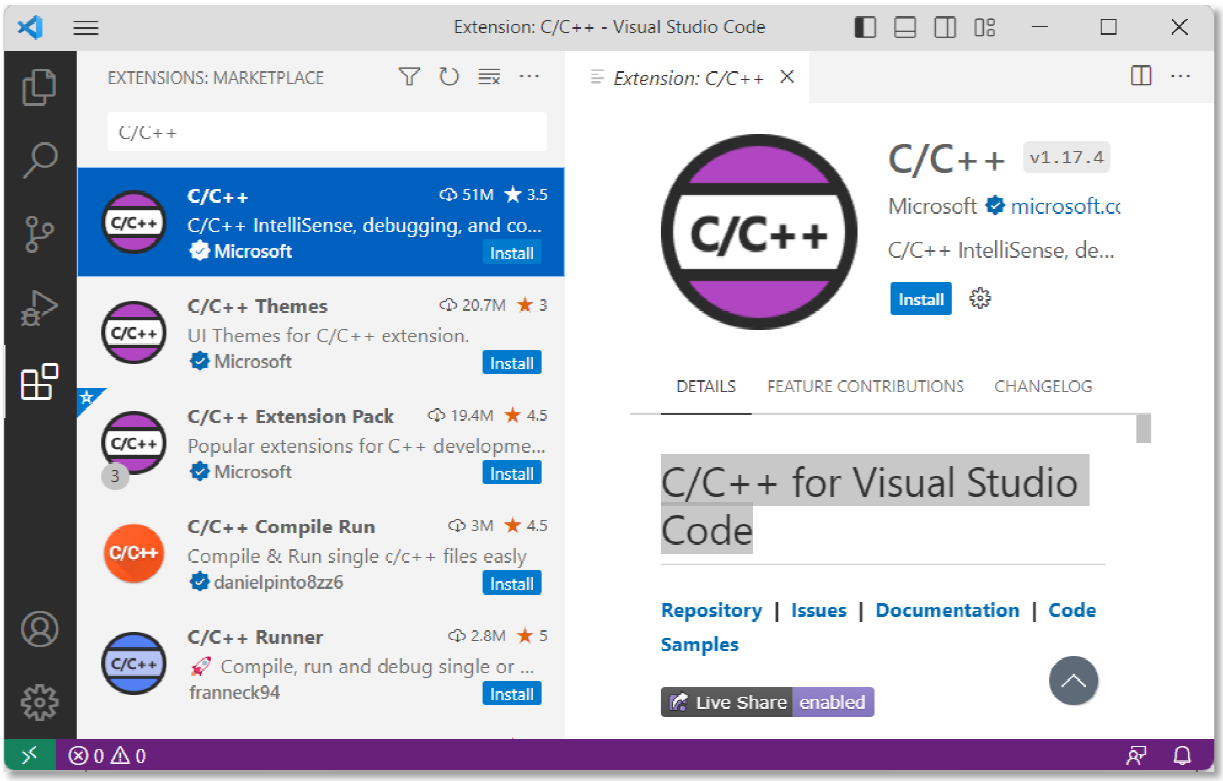
Główne okno programu Visual Studio Code po uruchomieniu przedstawia Rys. 3.



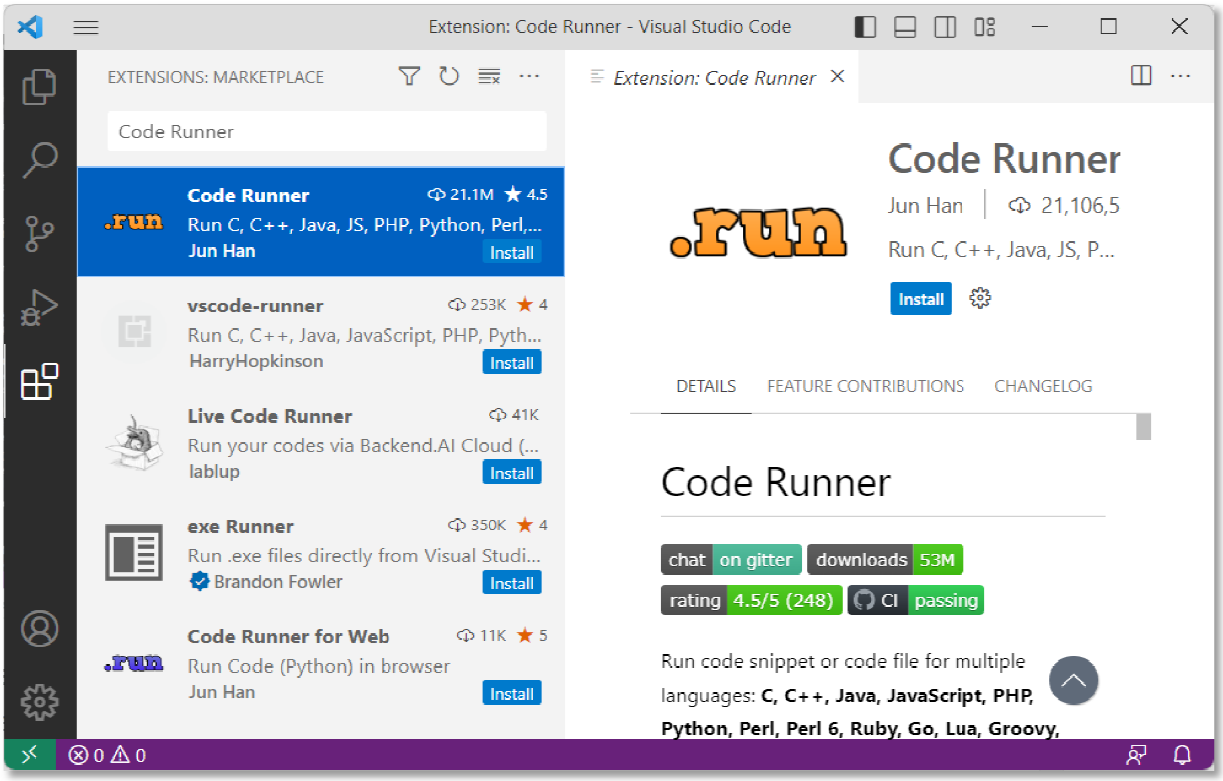
Rys. 3. Główne okno programu VS Code po pierwszym uruchomieniu

W kolejnym kroku instalujemy rozszerzenia umożliwiające pisanie programów w języku C oraz zmieniające język interfejsu użytkownika. W tym celu klikamy ikonkę, którą wskazuje czerwona strzałka na Rys. 3 lub wciskamy klawisz skrótu **Ctrl + Shift + X**. Wyszukujemy i instalujemy następujące rozszerzenia:

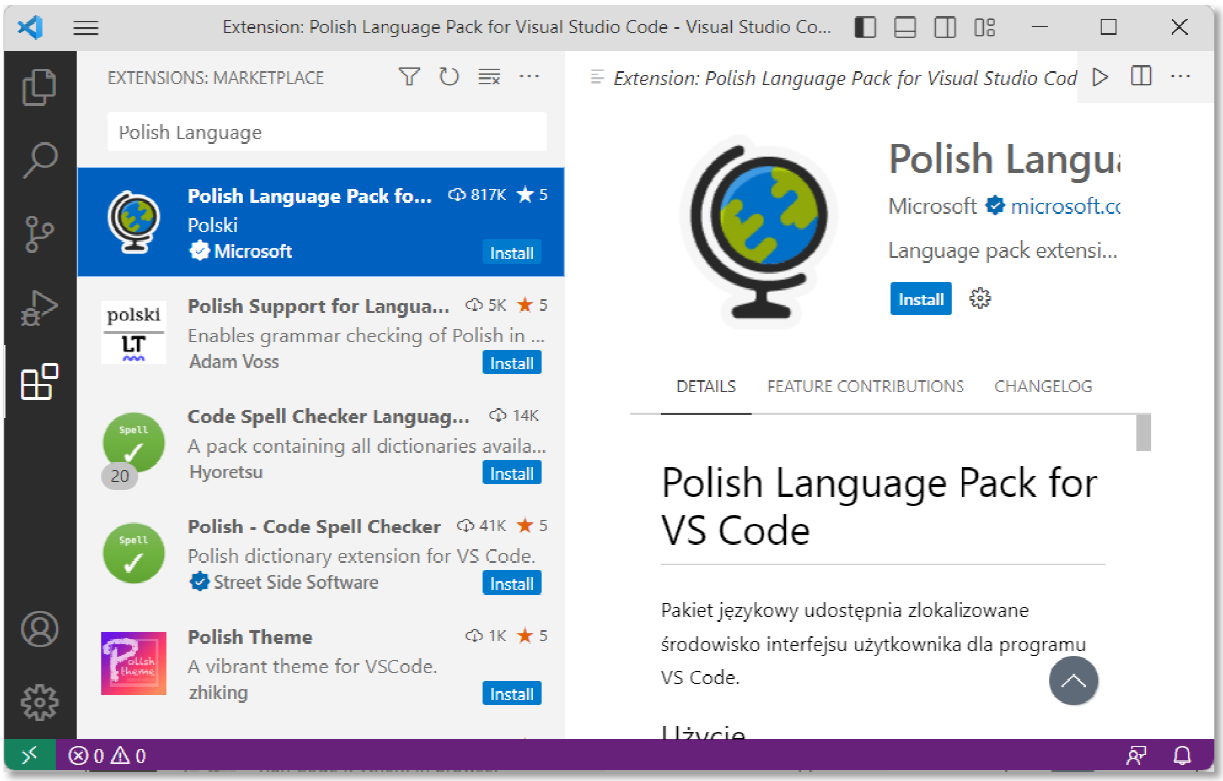
- **C/C++ for Visual Studio Code** (Microsoft) - rozszerzenie dodające obsługę języków C/C++ do Visual Studio Code, w tym funkcje edycji (**IntelliSense**) i debugowania (Rys. 4);
- **Code Runner** (Jun Han) - rozszerzenie umożliwiające uruchomienie pliku z kodem lub fragmentu kodu programu napisanego w różnych językach programowania, np. C, C++, Java, JavaScript, PHP, Python, Perl, Perl 6, Ruby, Go, Lua, Groovy, PowerShell, BAT/CMD, BASH/SH, F# Script, F# (.NET Core), C# Script, C# (.NET Core), VBScript (Rys. 5);
- **Polish Language Pack for Visual Studio Code** (Microsoft) - rozszerzenie umożliwiające zmianę języka interfejsu użytkownika na język polski (Rys. 6).



Rys. 4. Instalacja rozszerzenia **C/C++ for Visual Studio Code**

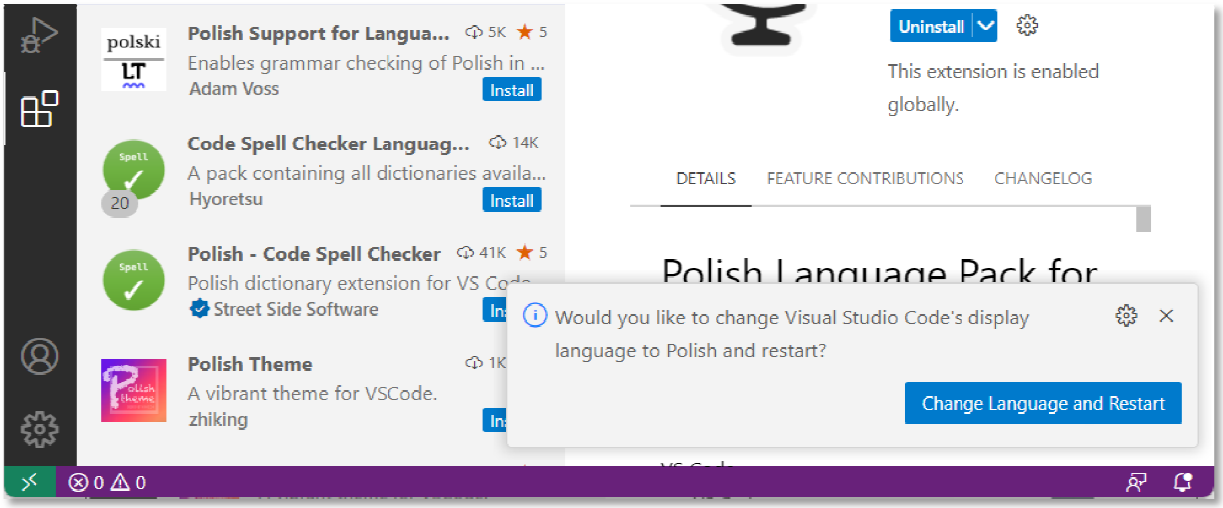


Rys. 5. Instalacja rozszerzenia **Code Runner**



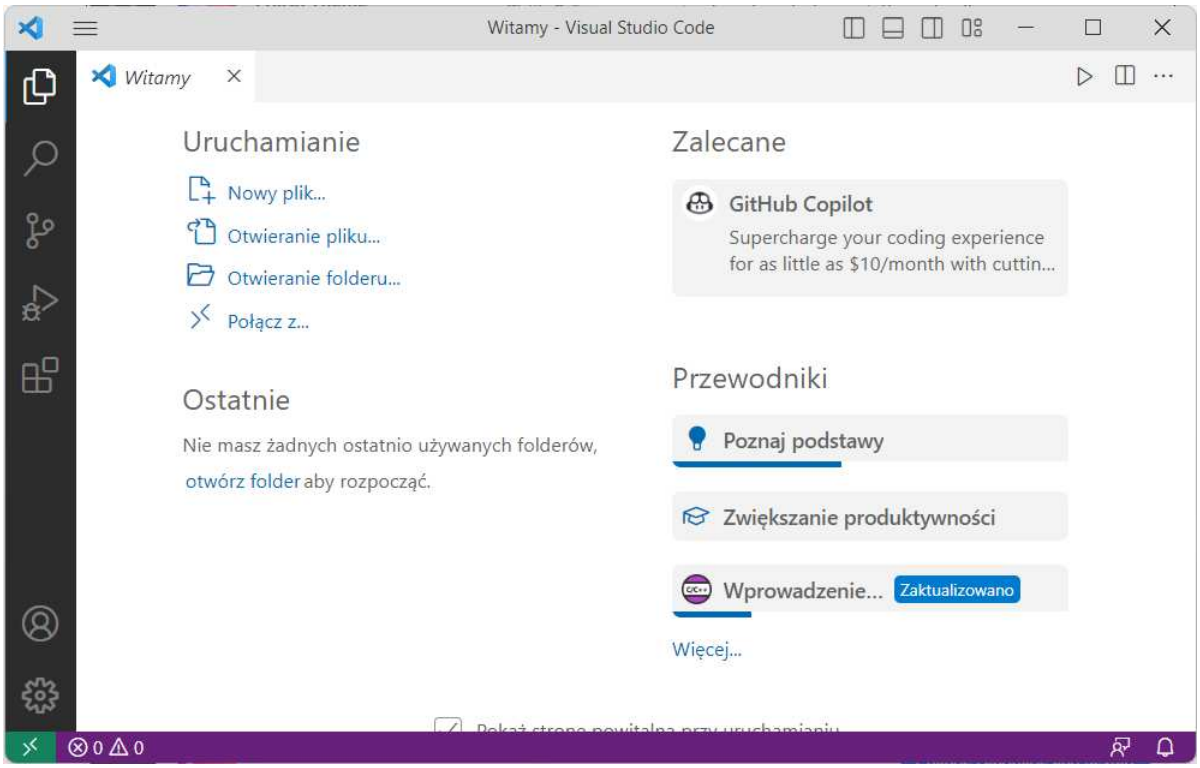
Rys. 6. Instalacja rozszerzenia **Polish Language Pack for Visual Studio Code**

Po zainstalowaniu rozszerzenia **Polish Language Pack for Visual Studio Code** na ekranie pojawi się komunikat (Rys. 7) z pytaniem, czy chcemy zmienić język interfejsu programu na język polski i ponownie uruchomić program.



Rys. 7. Pytanie o zmianę języka interfejsu i ponowne uruchomienie programu

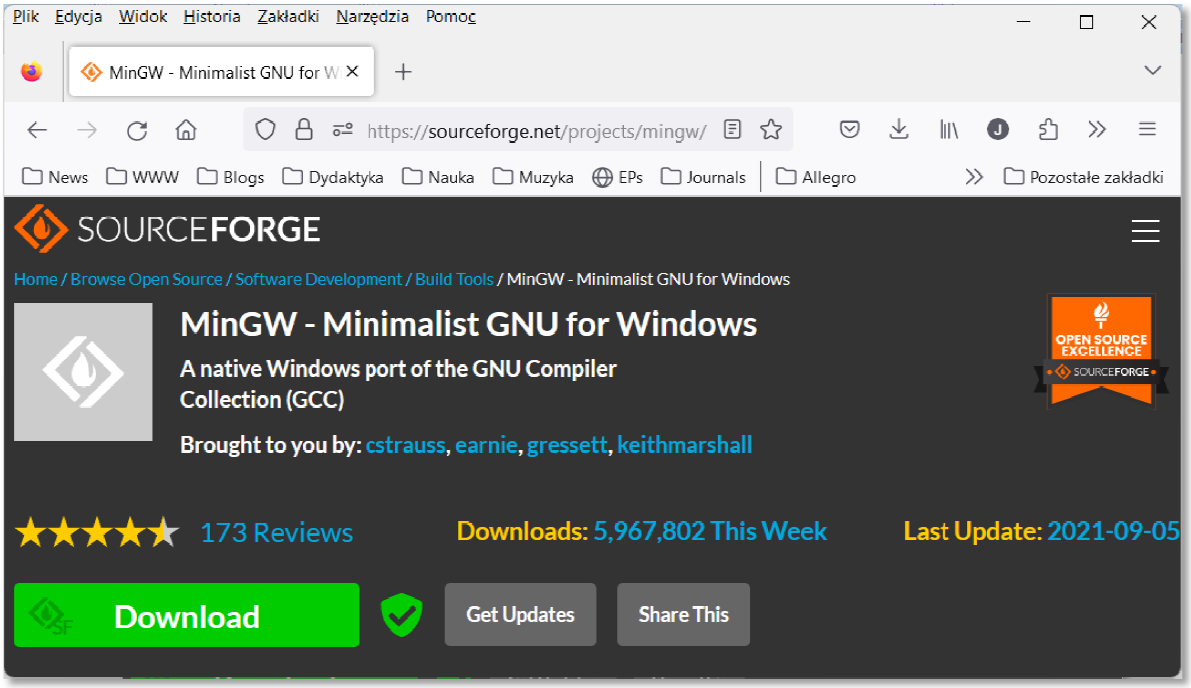
Po zatwierdzeniu zmian program uruchomi się ponownie z polskojęzycznym interfejsem (Rys. 8).



Rys. 8. Program Visual Studio Code z polskim interfejsem użytkownika

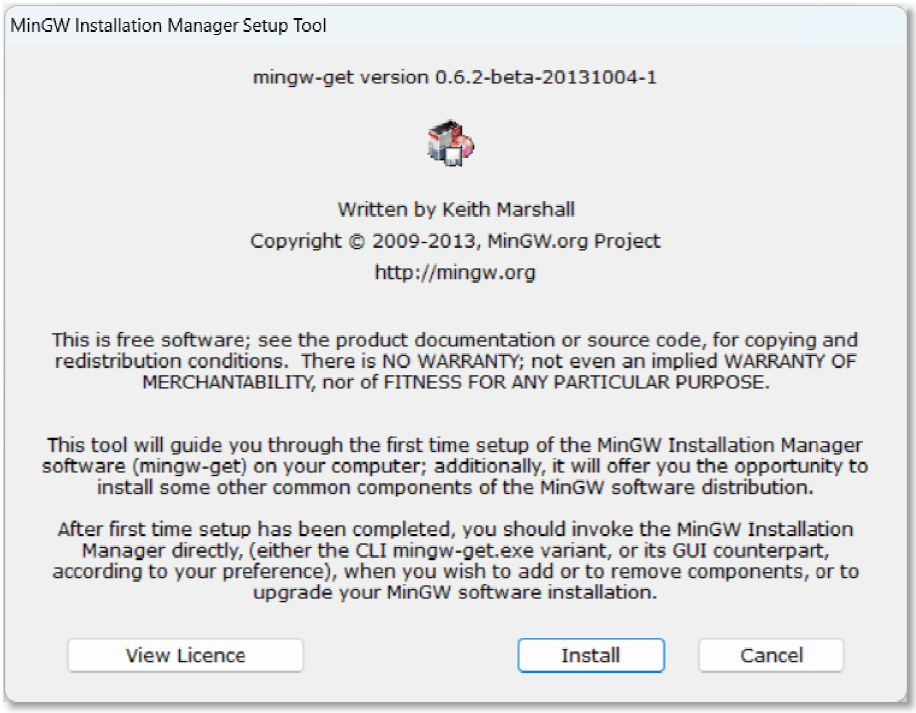
2.2. Instalacja kompilatora (MinGW)

W kolejnym kroku instalujemy **MinGW** zawierający kompilator języka C. Wersja instalacyjna znajduje się na stronie: https://sourceforge.net/projects/mingw/.



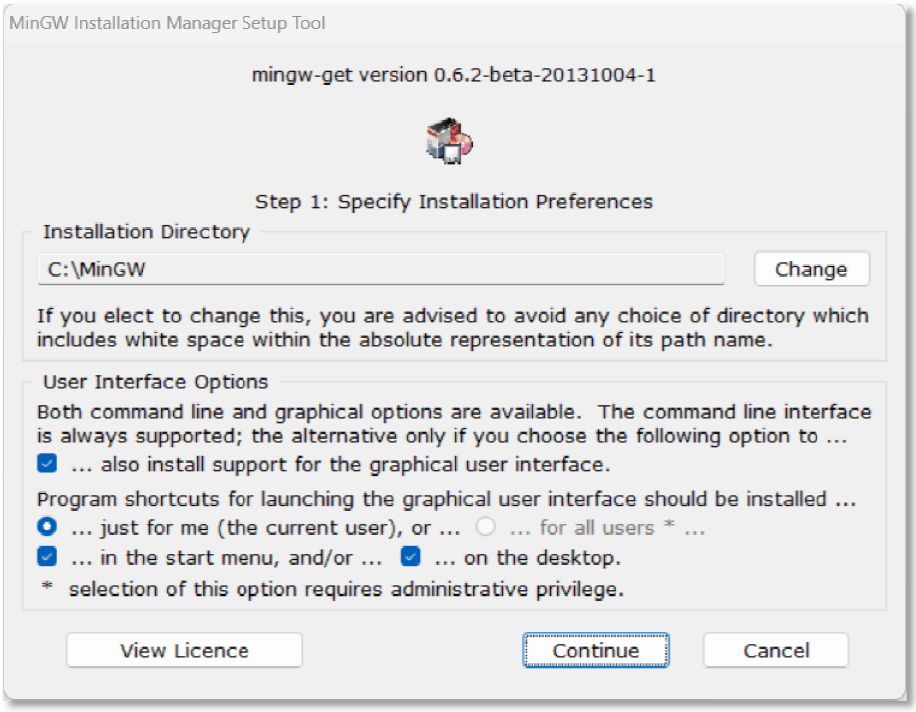
Rys. 9. Strona internetowa projektu MinGW

Na stroni internetowej klikamy zielony przycisk **Download** i ściągamy instalator. Po uruchomieniu instalatora na ekranie pojawi się poniższe okno:



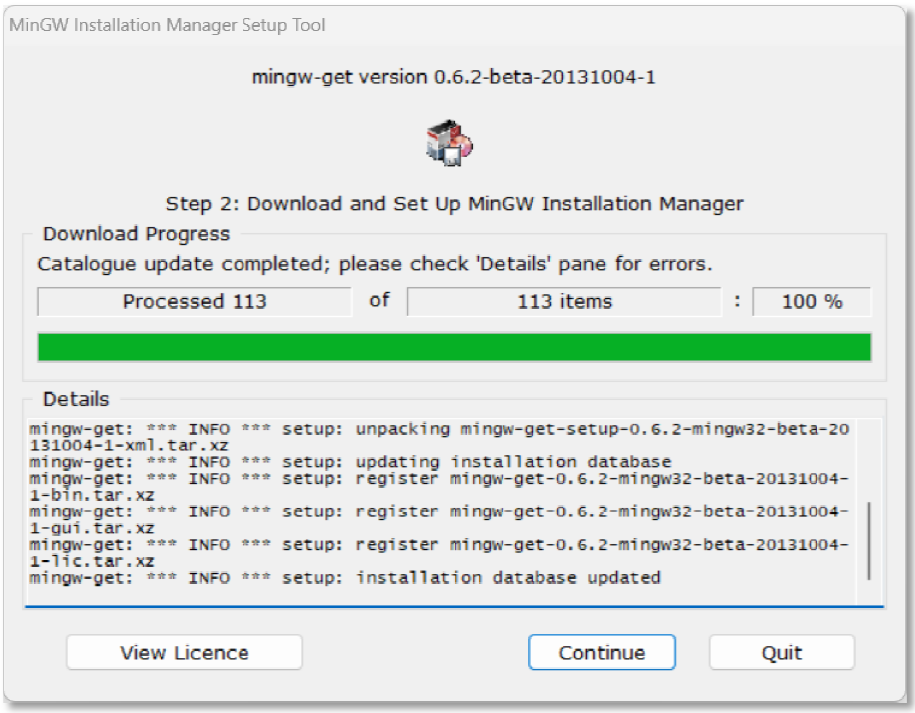
Rys. 10. Uruchomienie instalatora **MinGW**

Klikamy przycisk **Install** i wybieramy miejsce instalacji (folder na dysku) (Rys. 11).



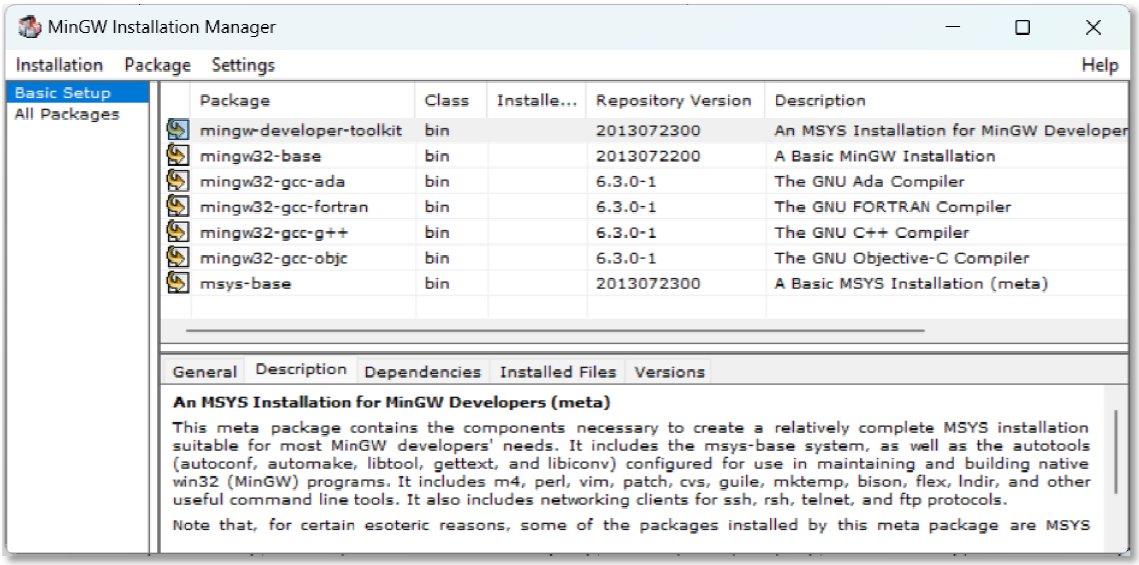
Rys. 11. Wybór folderu docelowego do instalacji **MinGW**

W tym momencie następuje ściągnięcie plików instalacyjnych na dysk. Po zakończeniu tego procesu (Rys. 12) klikamy przycisk **Continue**.



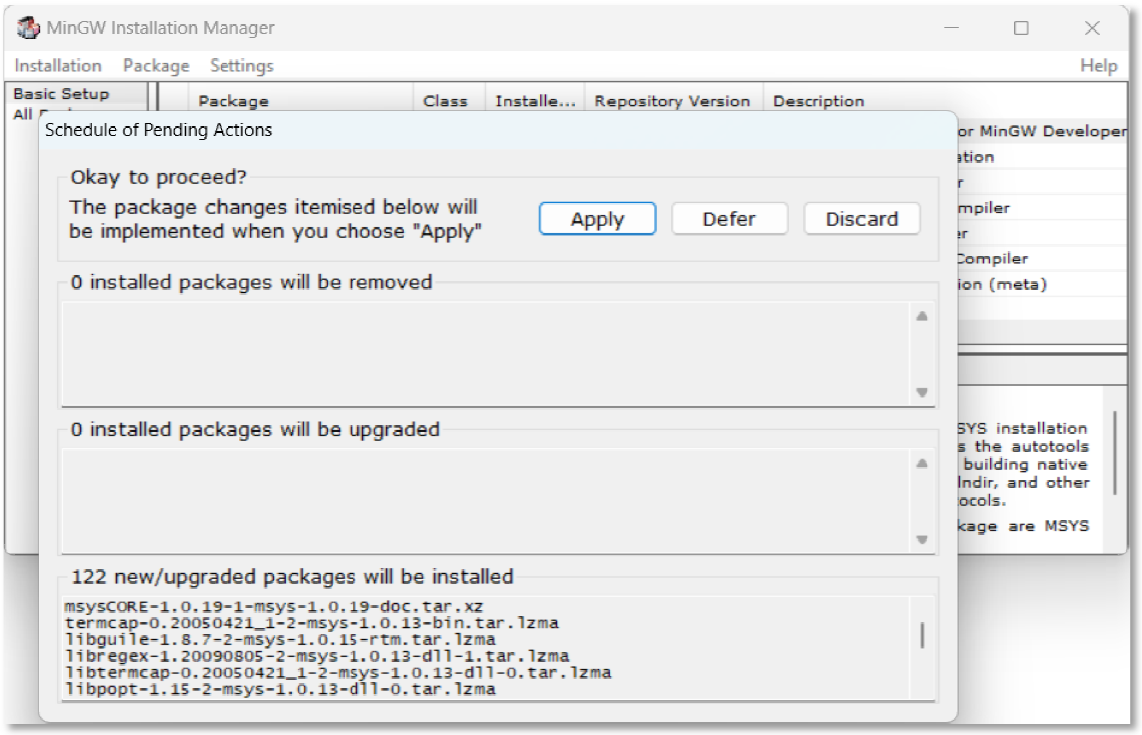
Rys. 12. Zakończenie procesu ściągania plików instalacyjnych **MinGW**

Zaznaczamy wszystkie pakiety do zainstalowania (Rys. 13).



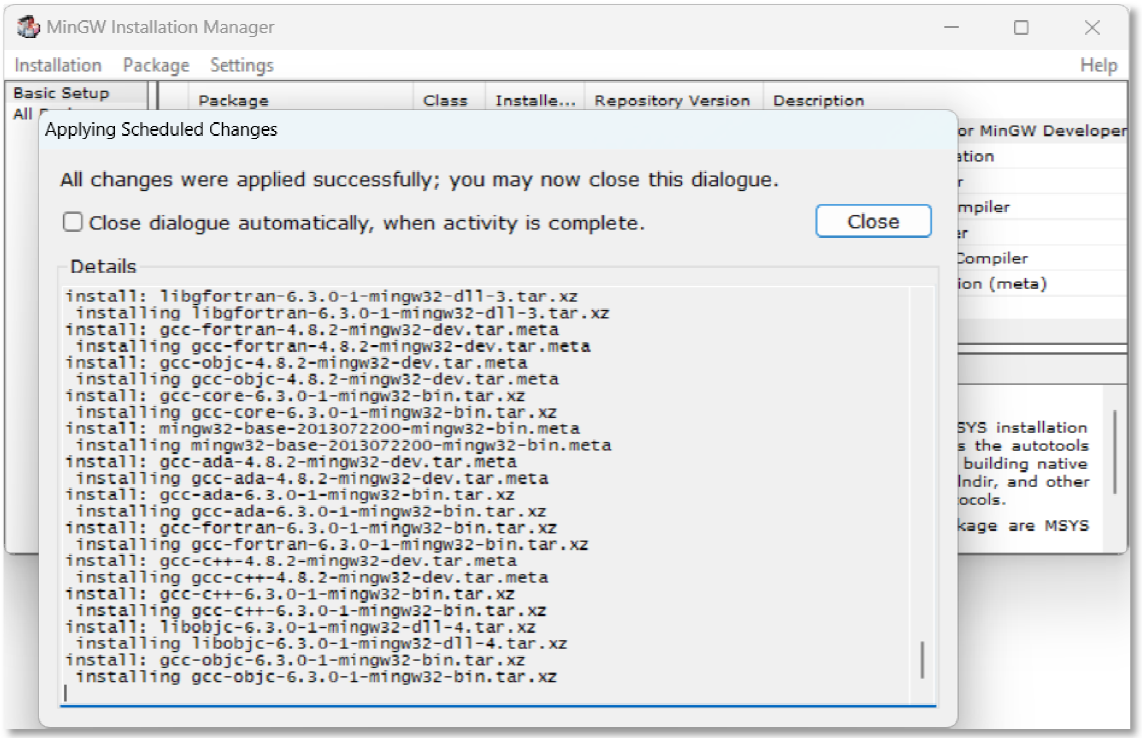
Rys. 13. Wybór pakietów **MinGW** do instalacji

Zatwierdzamy instalację wybierając z menu **Installation → Apply Changes**. W oknie, które zostanie wyświetlone, klikamy przycisk **Apply** (Rys. 14).



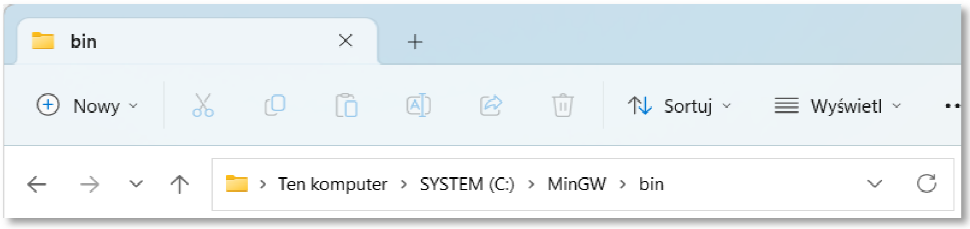
Rys. 14. Zatwierdzenie procesu instalacji pakietów **MinGW**

Proces instalacji kończymy poprzez kliknięcie przycisku **Close** w poniższym oknie.

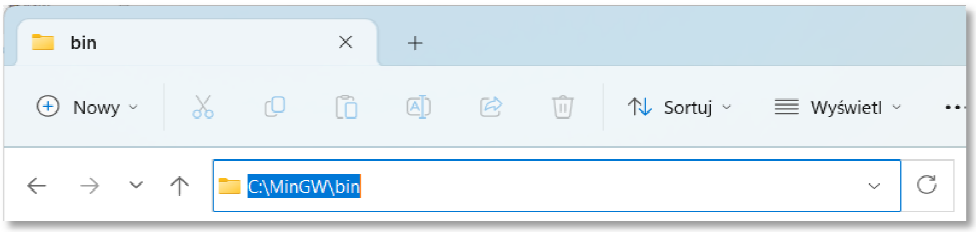


Rys. 15. Zakończenie procesu instalacji pakietów **MinGW**

Ostatnią czynnością jest dodanie ścieżki do folderu **bin** z instalacją **MinGW** do zmiennej środowiskowej o nazwie **Path**. W **Eksploratorze plików** wchodzimy do folderu **bin** z instalacją **MinGW** (Rys. 16) i kopiujemy ścieżkę dostępu (Rys. 17).

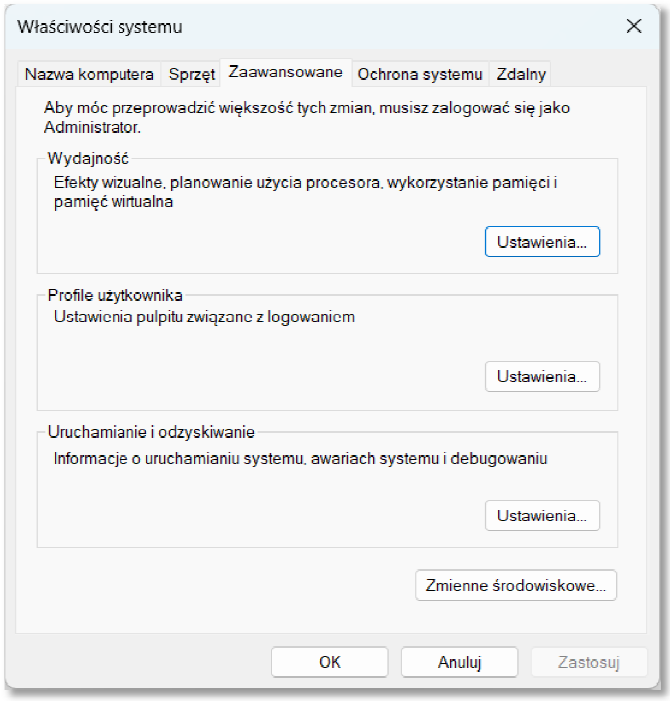


Rys. 16. Wejście do folderu **bin** z instalacją **MinGW**



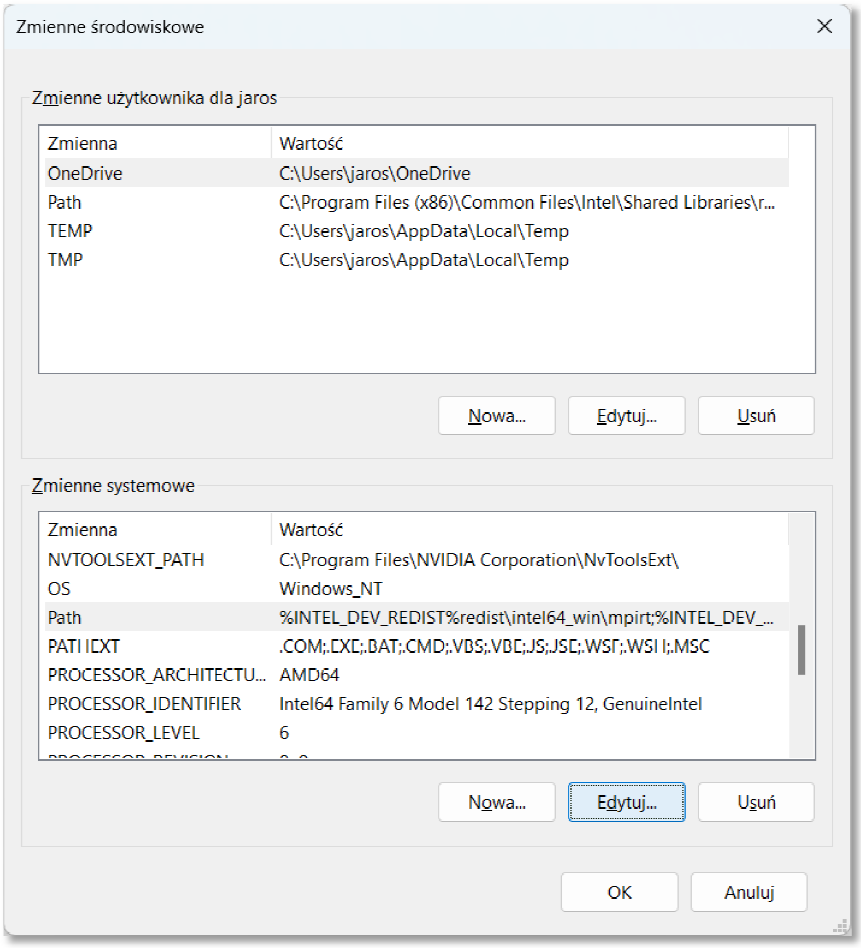
Rys. 17. Kopiowanie ścieżki dostępu do folderu **bin**

Następnie w systemie Windows wchodzimy do (Rys. 18): **Ustawienia → System → Informacje → Zaawansowane ustawienia systemu**.



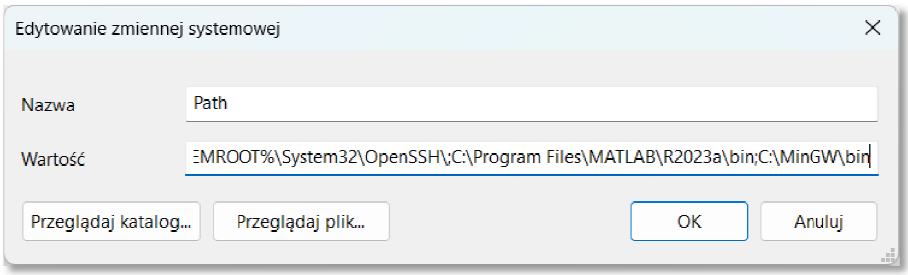
Rys. 18. Zaawansowane ustawienia systemu Windows

Klikamy przycisk **Zmienne środowiskowe**. W dolnym oknie **Zmienne systemowe** (Rys. 19) wyszukujemy i klikamy zmienną **Path**, a następnie klikamy przycisk **Edytuj**.



Rys. 19. Okno ze zmiennymi środowiskowymi

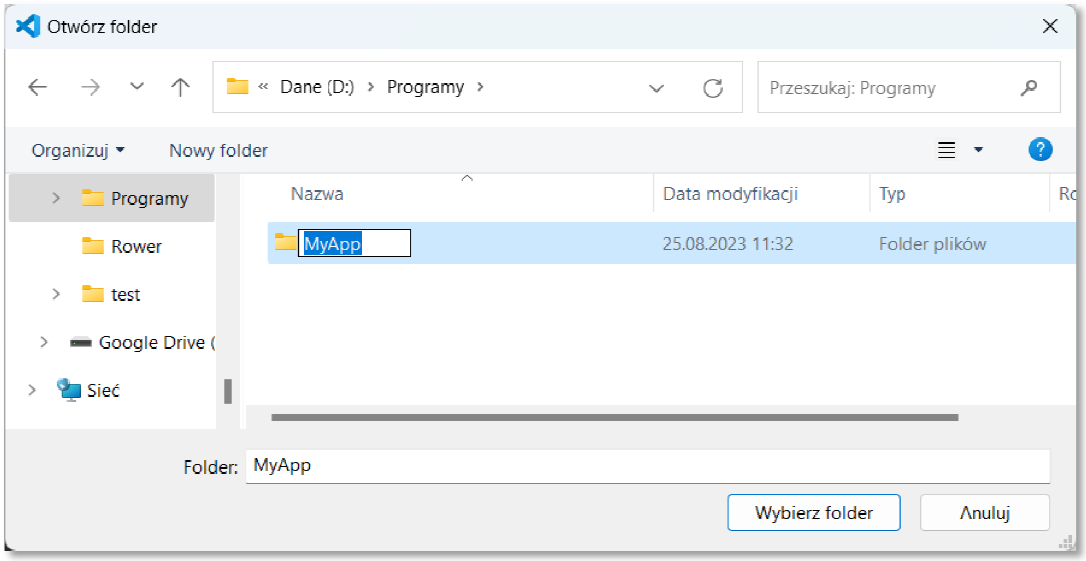
Przechodzimy na koniec pola **Wartość** i wklejamy ścieżkę dostępu do folderu **bin** (Rys. 20). Zatwierdzamy zmiany przyciskiem **OK**.



Rys. 20. Dodanie do zmiennej środowiskowej **Path** ścieżki dostępu do folderu **bin**

2.3. Utworzenie pliku z kodem źródłowym programu w języku C

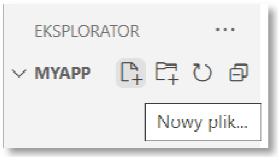
Napisanie programu w języku C przy wykorzystaniu edytora Visual Studio Code wymaga w pierwszej kolejności utworzenia i otwarcia folderu, w którym będzie znajdował się plik z kodem źródłowym. W tym celu z menu głównego wybieramy pozycję **Plik → Otwórz folder** (**Ctrl + K**, **Ctrl + O**). Tworzymy nowy folder i wybieramy go (Rys. 21). W przykładzie folder ten nazywa się **MyApp**.



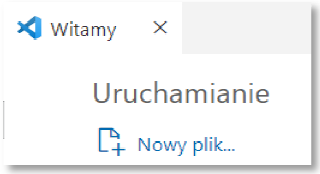
Rys. 21. Utworzenie nowego folderu

Następnie dodajemy nowy plik. Można to zrobić na kilka sposobów:

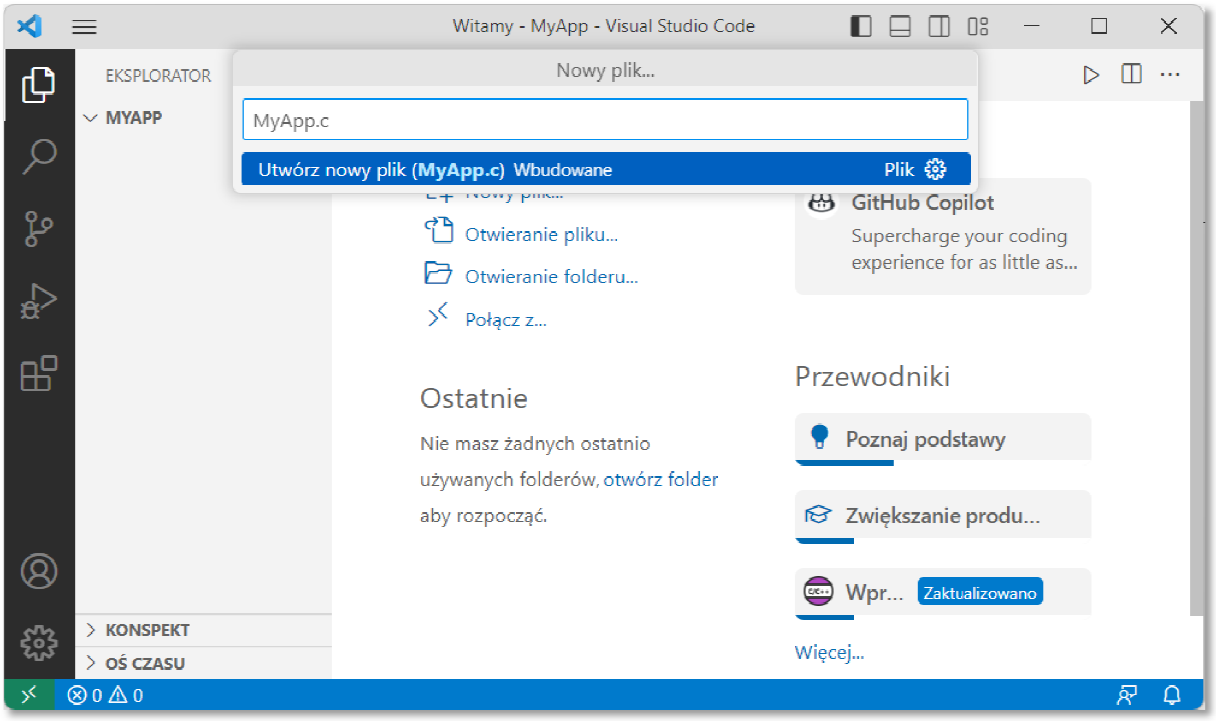
- wybierając w menu głównym **Plik → Nowy plik**;
- używając klawisza skrótu **Ctrl + Alt + Windows + N**;
- klikając w **Eksploratorze** ikonkę ze znakiem plus (obok nazwy projektu);



- klikając na karcie **Witamy** pozycję **Uruchamianie → Nowy plik**.

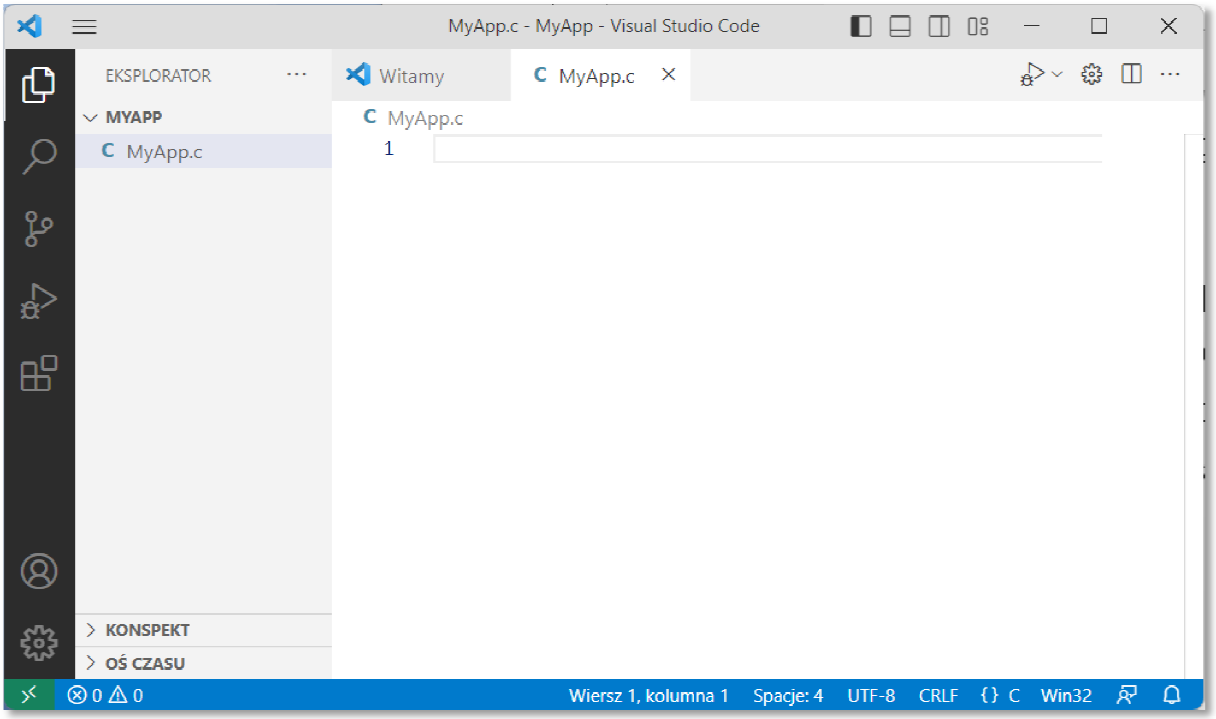


Wprowadzamy nazwę pliku z rozszerzeniem **.c** (np. **MyApp.c**) (Rys. 22).



Rys. 22. Dodanie pliku z kodem źródłowym

W ten sposób edytor gotowy jest do wprowadzenia kodu programu (Rys. 23).



Rys. 23. Edytor z otwartym plikiem do wprowadzania kodu 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:

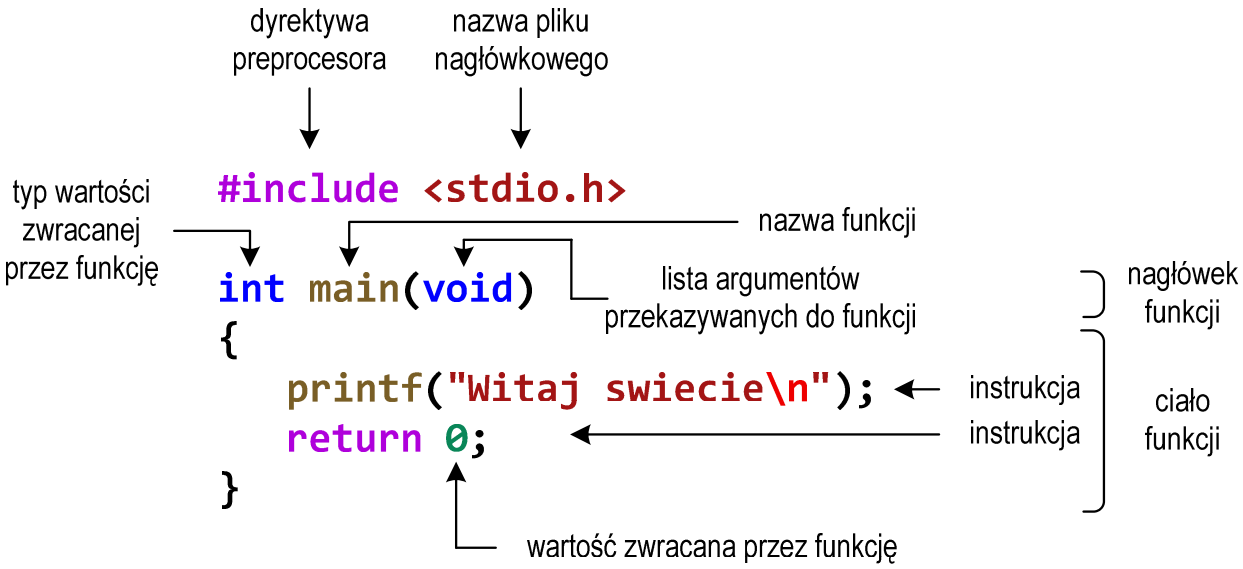
- ISO/IEC 9899:1999 - 1999 r., standard nazwany C99;
- ISO/IEC 9899:2011 - 2011 r., standard nazwany C11;
- ISO/IEC 9899:2018 - 2018 r., standard nazwany C18 lub C17.

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:

```c
#include <stdio.h>

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

Program w języku C (Rys. 24) składa się z funkcji i zmiennych. Funkcje zawierają instrukcje określające wykonywane operacje, zaś zmienne przechowują wartości wykorzystywane podczas tych operacji.



Rys. 24. Struktura programu w języku C

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.

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*).

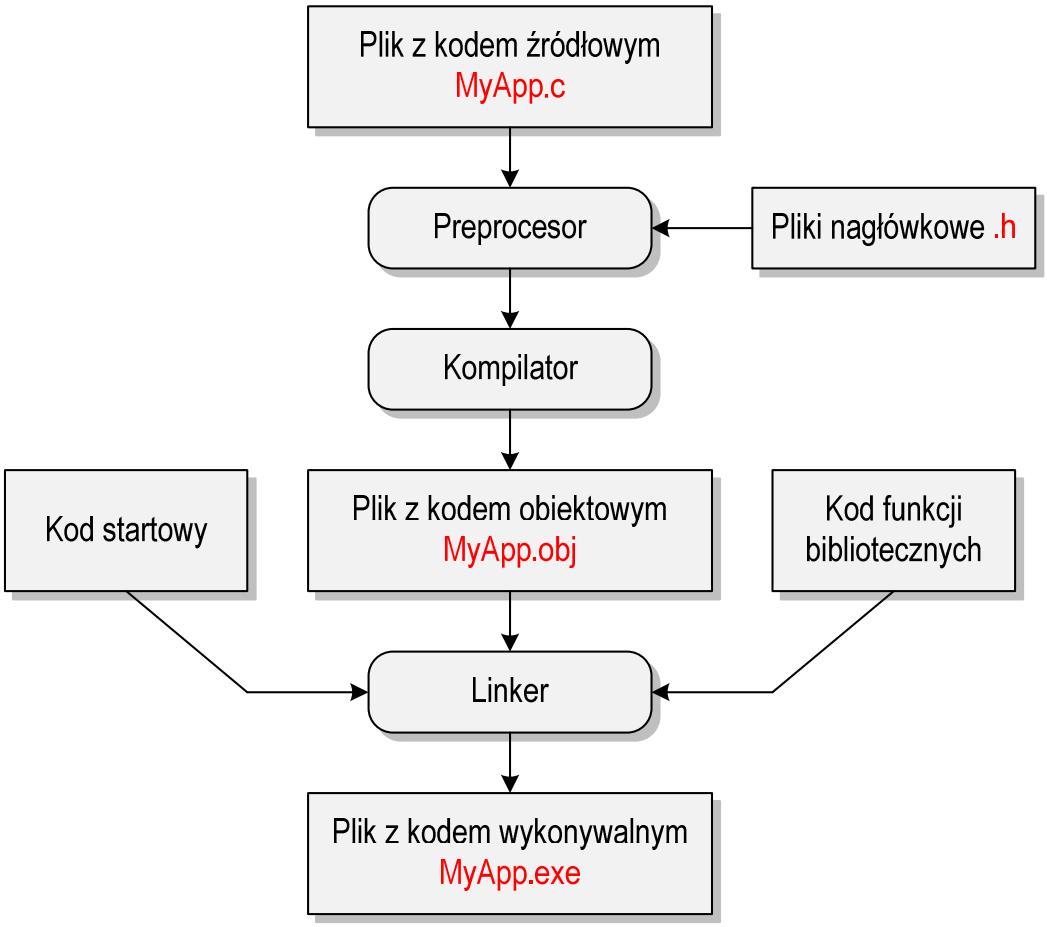
Funkcję wywołuje się podając jej nazwę (**printf()**) i (w nawiasach zwykłych) listę argumentów przekazywanych do funkcji (**"Witaj swiecie\n"**). W powyższym przykładzie do funkcji **main()** nie są przekazywane żadne argumenty, informuje o tym słowo **void** w jej nagłówku (słowo to może być pominięte).

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

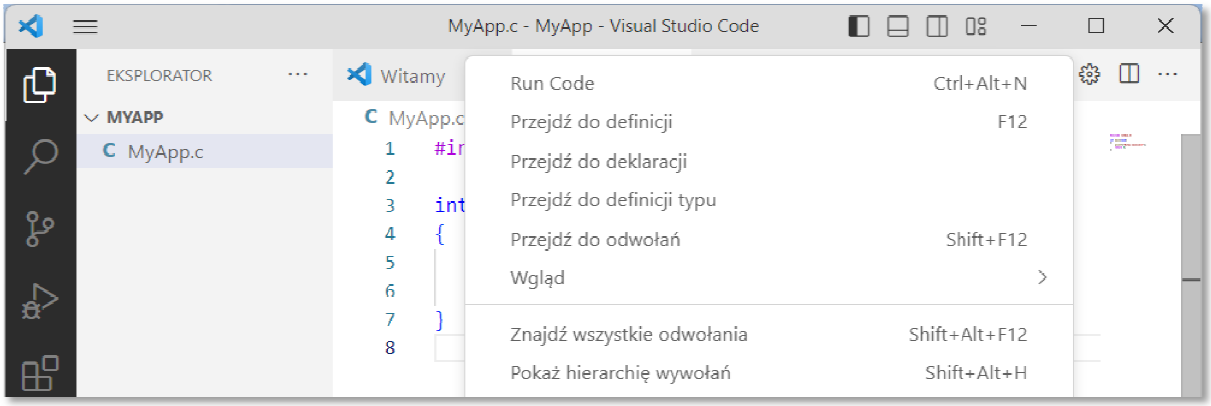
Uruchomienie programu wymaga przekształcenia pliku z kodem źródłowym **.c** na plik wykonywalny **.exe**. Operacja ta składa się z kilku kroków (Rys. 25).



Rys. 25. Etapy tworzenia pliku wykonywalnego z kodu źródłowego

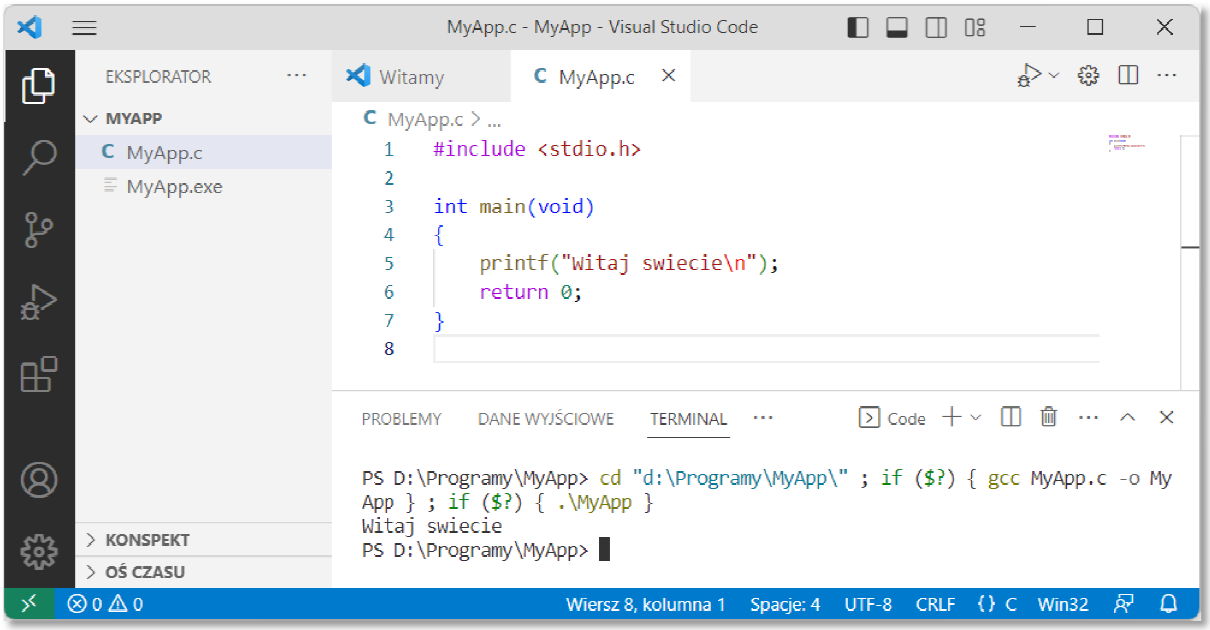
W pierwszym kroku specjalny program zwany preprocesorem analizuje kod źródłowy programu poszukując wyrażeń zaczynających się od znaku **#** (dyrektywy preprocesora). Na ich podstawie odpowiednio modyfikuje kod programu. Następnie kompilator (ang. *compiler*) kompiluje czyli przetwarza kod źródłowy programu 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 programie VS Code w celu kompilacji i uruchomienia programu należy kliknąć prawym klawiszem myszki na kodzie programu i z menu podręcznego wybrać pierwszą opcję **Run Code** lub użyć klawisza skrótu **Ctrl + Alt + N** (Rys. 26). Przed kompilacją i uruchomieniem programu należy pamiętać o konieczności zapisania na dysku zmian w kodzie programu (menu **Plik → Zapisz** lub klawisz skrótu **Ctrl + S**), gdyż VS Code nie zapisuje automatycznie pliku na dysku.



Rys. 26. Kompilacja i uruchomienie programu

Wynik działania programu możemy zobaczyć w oknie **TERMINAL** (Rys. 27).



Rys. 27. Wynik działania programu w oknie **TERMINAL**

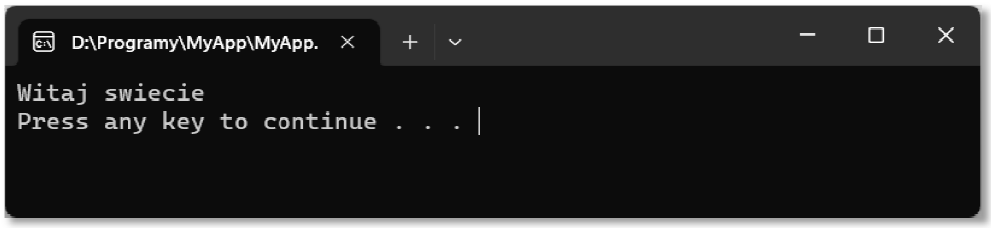
Jeśli w kodzie programu występują błędy, to kompilator wyświetla tylko odpowiednie komunikaty, natomiast plik wykonywalny **.exe** nie jest tworzony.

Jeśli uruchomimy skompilowany program z poziomu systemu operacyjnego, to nie zobaczymy efektów jego działania. System operacyjny utworzy okno **Wiersza polecenia**, uruchomi w nim program, program zakończy się i okno zostanie natychmiast zamknięte. Aby zaobserwować wyniki pracy programu z poziomu systemu operacyjnego 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 cudzysłowy) przekazanej do niej jako argument. Polecenie **pause** zawiesza wykonywanie programu i wyświetla komunikat pokazany na poniższym rysunku.



Rys. 28. Uruchomiony program w oknie **Wiersza polecenia**

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

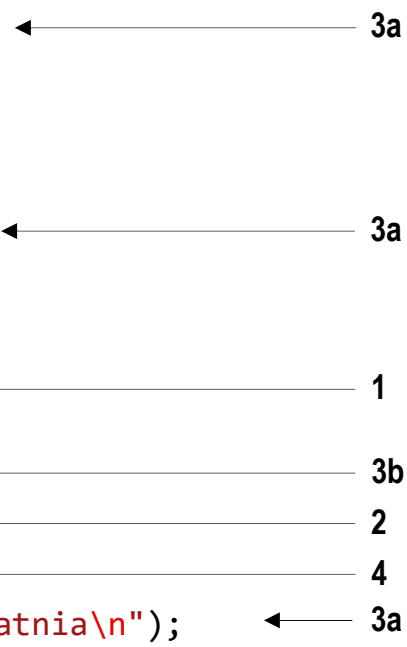
Sposób zapisu kodu programu wpływa tylko na jego przejrzystość, a nie na kompilację i wykonanie. W tym właśnie celu w przedstawionych powyżej programach występują dodatkowe spacje przed funkcją **printf()** i słowem kluczowym **return**. Program wyświetlający tekst **Witaj swiecie** można zapisać także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

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

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 tekstu:

```
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 zmiennej **x** wartości **5**);

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 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`

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 2.

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:  01-10-2023 08:00
  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).

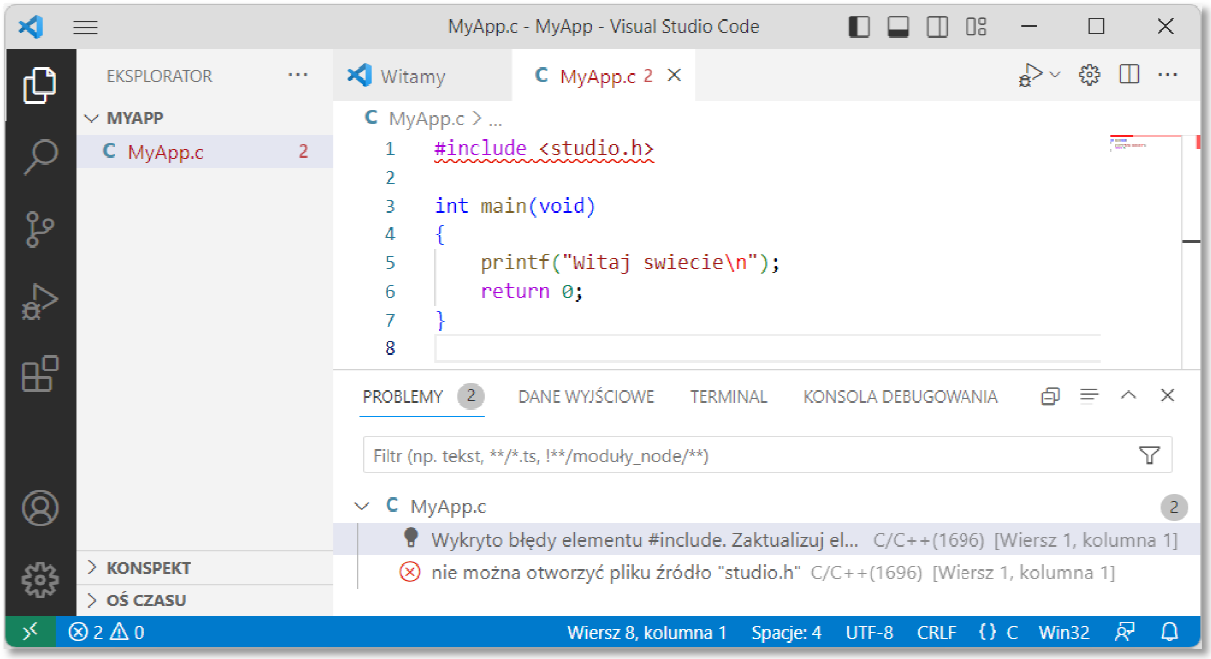
W początkowej fazie nauki programowania większość popełnianych błędów są to literówki oraz błędy składni. Pouczającym może być samodzielne zrobienie błędów i zaobserwowanie reakcji kompilatora na nie.

VS Code już podczas pisania kodu programu sprawdza jego poprawność i podkreśla fragmenty kodu, w których występują błędy. Przykładowo w poniższym programie błędnie podano nazwę pliku nagłówkowego - zamiast **stdio.h** jest **studio.h**.

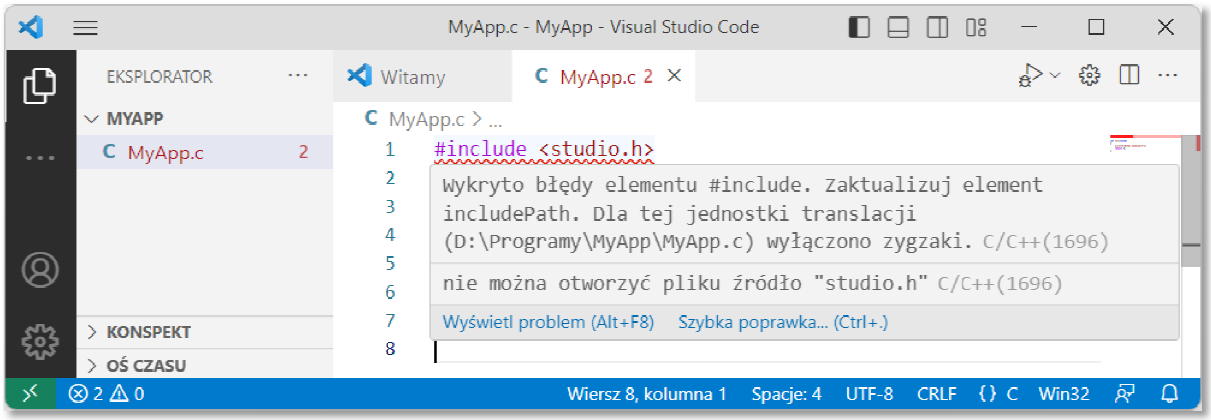
```c
#include <studio.h>

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

- błędna nazwa pliku nagłówkowego

We wprowadzonym kodzie programu pierwszy wiersz jest podkreślony na czerwono. Informacje o błędach można odczytać w zakładce **PPROBLEMY** (Rys. 29) lub wyświetlić po najechaniu kursorem myszki na podkreślenie (Rys. 30).



Rys. 29. Informacje o błędach w oknie **PROBLEMY**



Rys. 30. Informacje o błędach po najechaniu kursorem myszki na podkreślenie

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

- brak średnika na końcu wiersza

Nie wszystkie błędy wykrywane są już podczas pisania programu. W poniższym programie brakuje klamry kończącej program. Błąd ten zostanie wykryty dopiero podczas kompilacji programu.

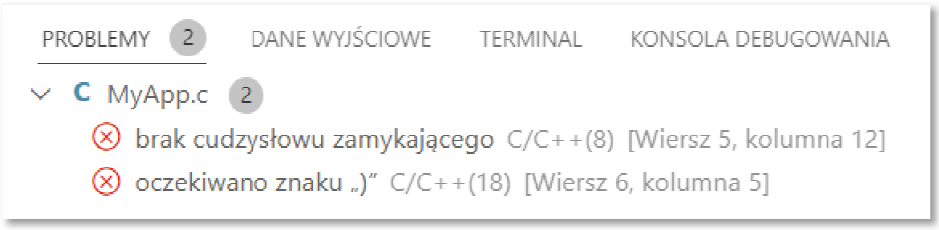
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      printf("Witaj swiecie\n");
6      return 0;
7
```

- brak nawiasu klamrowego kończącego program

```
MyApp.c: In function 'main':
MyApp.c:6:5: error: expected declaration or statement at end of input
     return 0;
```

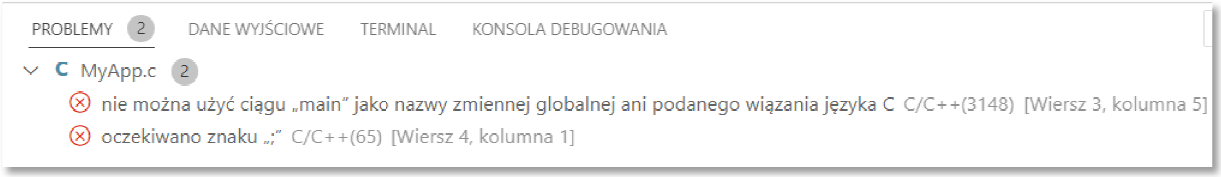
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      printf("Witaj swiecie\n);
6      return 0;
7  }
```

- brak cudzysłowu kończącego łańcuch



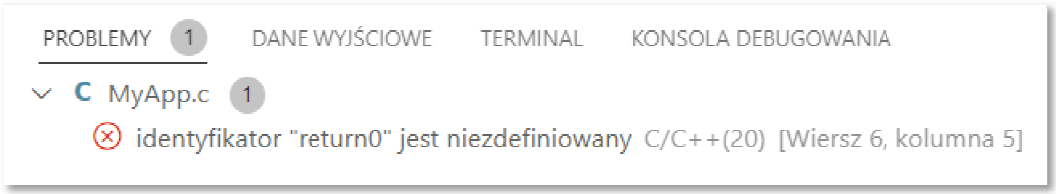

```c
#include <stdio.h>

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

- brak nawiasów po funkcji **main**




```c
#include <stdio.h>

int main(void)
{
    printf("Witaj swiecie\n");
    return0;
}
```

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




```c
#include <stdio.h>

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

- nazwa funkcji **printf** pisana wielką literą

```
MyApp.c: In function 'main':
MyApp.c:5:5: warning: implicit declaration of function 'Printf' [-Wimplicit-function-declaration]
    Printf("Witaj swiecie\n");
    ^~~~~~
C:\Users\AppData\Local\Temp\ccfU0Ipf.o:MyApp.c:(.text+0x16): undefined reference to `Printf'
collect2.exe: error: ld returned 1 exit status
```

```c
#include <stdio.h>

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

- nazwa funkcji **main** pisana wielką literą

```
c:/mingw/bin/../lib/gcc/mingw32/6.3.0/../../../libmingw32.a(main.o):(.text.startup+0xa0): undefined
reference to `WinMain@16'
collect2.exe: error: ld returned 1 exit status
```

```c
#include <stdio.h>

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

- średnik w nagłówku funkcji **main**

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) uruchom program Visual Studio Code, otwórz folder i dodaj plik z rozszerzeniem **.c**, w którym będzie znajdował się kod źródłowy programu;
   b) wprowadź kod źródłowy programu wyświetlającego tekst **„Witaj swiecie”**;
   c) skompiluj i uruchom program;
   d) odszukaj na dysku folder zawierający skompilowany plik wynikowy **.exe**; uruchom program z poziomu systemu operacyjnego; sprawdź, czy można zaobserwować wyniki jego działania;
   e) dodaj do programu instrukcję zatrzymującą program przed jego końcem (**system(”pause”)** lub **getch()**); sprawdź, czy uruchomienie programu z poziomu systemu operacyjnego pozwoli zobaczyć wyniki jego działania;
   f) przekształć kod źródłowy programu tak, aby zajmował jak najmniej wierszy; skompiluj i uruchom program.

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. Postaraj się, aby były to inne błędy 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://code.visualstudio.com/ - Visual Studio Code

[9] https://sourceforge.net/projects/mingw/ - MinGW

5. Pytania kontrolne

1. Omów sposób pisania, kompilacji oraz uruchamiania programu w edytorze Visual Studio Code.
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 z kodem źródłowym programu 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.