Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 1

Kod przedmiotu: **ES1E2009**

(studia stacjonarne)

ŚRODOWISKO CODE::BLOCKS.
JĘZYK C - OGÓLNA STRUKTURA PROGRAMU

Numer ćwiczenia

INF01

Autor:
dr inż. Jarosław Forenc

Białystok 2021

Spis treści

1. Opis stanowiska ... 3
1.1. Stosowana aparatura ... 3
1.2. Oprogramowanie ... 3
2. Wiadomości teoretyczne ... 3
2.1. Środowisko Code::Blocks ... 3
2.2. Utworzenie nowego projektu w Code::Blocks ... 5
2.3. Język C ... 9
2.4. Ogólna struktura programu w języku C ... 9
2.5. Tworzenie pliku wykonywalnego i uruchomienie programu ... 11
2.6. Sposób zapisu kodu programu ... 16
2.7. Struktura programu z kilkoma funkcjami, typy instrukcji w języku C ... 17
2.8. Wyświetlanie tekstu funkcją printf() ... 18
2.9. Komentarze ... 19
2.10. Najczęściej popełniane błędy podczas pisania programów ... 20
3. Przebieg ćwiczenia ... 24
4. Literatura ... 25
5. Pytania kontrolne ... 26
6. Wymagania BHP ... 26

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

© Wydział Elektryczny, Politechnika Białostocka, 2021 (wersja 5.0)

Wszelkie prawa zastrzeżone. Żadna część tej publikacji nie może być kopiowana i odtwarzana w jakiejkolwiek formie i przy użyciu jakichkolwiek środków bez zgody posiadacza praw autorskich.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows (XP/7/10).

1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Code::Blocks.

2. Wiadomości teoretyczne

2.1. Środowisko Code::Blocks

Code::Blocks jest bezpłatnym, wieloplatformowym, zintegrowanym środowiskiem programistycznym (ang. *IDE - Integrated Development Environment*) na licencji GNU General Public License version 3. Środowisko to umożliwia pisanie programów w językach C, C++ i Fortran. Dzięki zastosowaniu wieloplatformowej biblioteki wxWidgets programy mogą być uruchamiane na systemach operacyjnych Windows, Linux oraz MacOS.

Pierwsza wersja środowiska Code:Blocks, tzw. RC (ang. *Release Candidate*) została wydana w 2005 roku, natomiast pierwsza stabilna wersja (Code::Blocks 8.02) pojawiła 28 lutego 2008 roku. Numer wersji oznacza rok i miesiąc wydania. Aktualna stabilna wersja programu to Code::Blocks 20.03, która została wydana 29 marca 2020 roku.

Główna strona internetowa programu to: https://www.codeblocks.org/, zaś najnowsza wersja instalacyjna środowiska znajduje się pod adresem: https://www.codeblocks.org/downloads/binaries/. W przypadku systemu Windows, spośród dostępnych wersji instalacyjnych (Rys. 1), należy wybrać plik oznaczony jako codeblocks-20.03mingw-setup.exe, gdyż dodatkowo zawiera on kompilatory GCC/G++/GFortran z projektu MinGW-W64.

Microsoft Windows

File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

NOTE: The codeblocks-20.03-setup.exe file includes Code::Blocks with all plugins. The codeblocks-20.03-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s).

NOTE: The codeblocks-20.03mingw-setup.exe file includes additionally the GCC/G++/GFortran compiler and GDB debugger from **MinGW-W64 project** (version 8.1.0, 32/64 bit, SEH).

Rys. 1 Strona internetowa z wersjami instalacyjnymi środowiska Code:Blocks 20.03

Dużą zaletą środowiska Code:Blocks jest występowanie w nim tzw. pluginów, które są zintegrowane z programem i rozszerzają jego możliwości. Część pluginów jest od razu instalowana ze środowiskiem, a dodatkowe mogą być doinstalowywane, zależnie od potrzeb.

Po zainstalowaniu i uruchomieniu środowiska Code:Blocks wyświetlane jest główne okno programu (Rys. 2).

Elementy głównego okna programu:

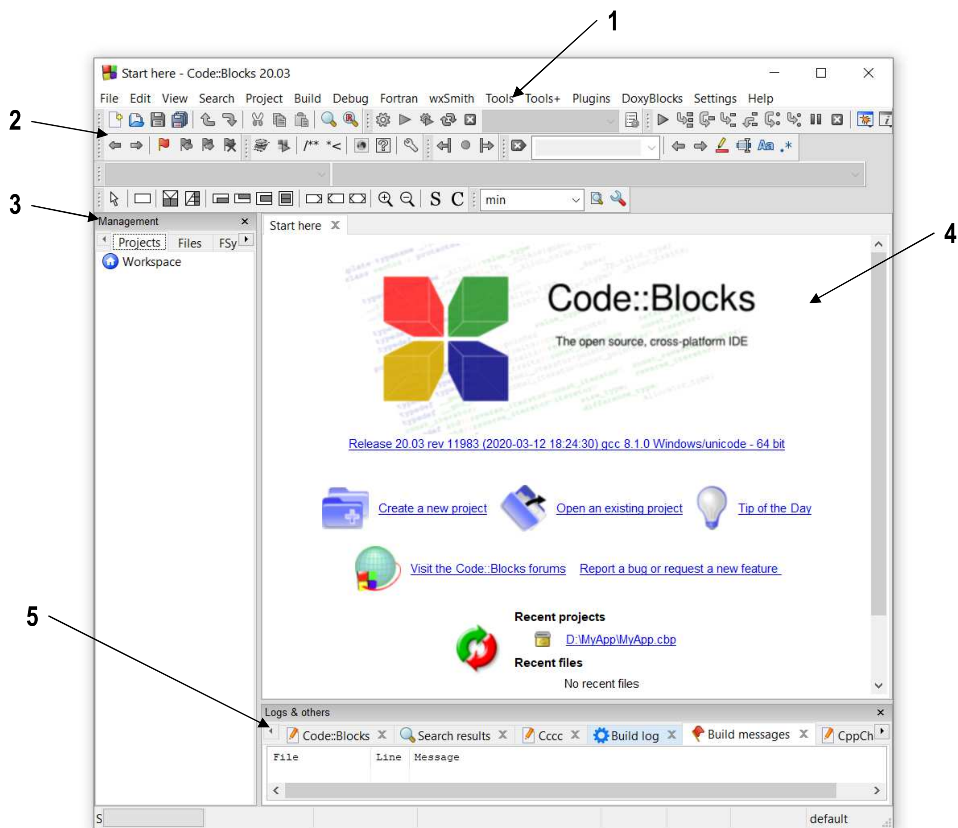
1 - Menu główne;

2 - Paski narzędziowe;

3 - *Management* - lista projektów otwartych w programie;

4 - *Start here* - okno wyświetlające podstawowe informacje o programie oraz umożliwiające m.in. stworzenie nowego projektu (*Create a new project*), otwarcie istniejącego projektu (*Open an existing project*), otwarcie jednego z ostatnio otwieranych projektów (*Recent projects*) lub plików (*Recent Files*);

5 - *Logs & others* - okno używane do logów kompilacji, wyników wyszukiwań.



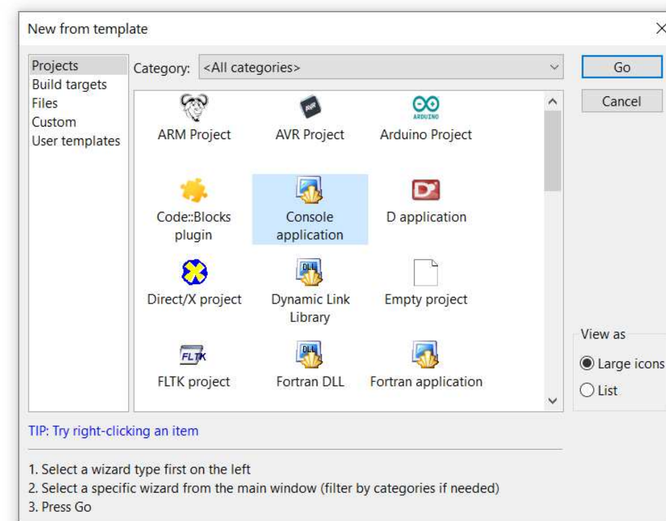
Rys. 2 Główne okno środowiska Code::Blocks 20.03

2.2. Utworzenie nowego projektu w Code::Blocks

Rozpoczęcie pracy ze środowiskiem Code::Blocks wymaga utworzenia nowego projektu. Nowy projekt można utworzyć na dwa sposoby:

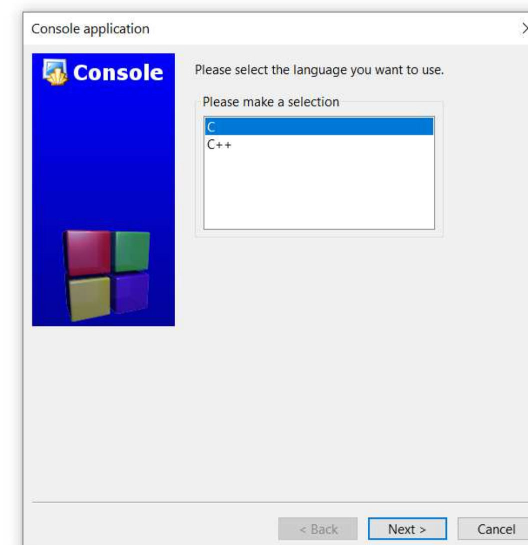
- klikając w oknie **Start here** opcję **Create a new project**;
- wybierając w menu głównym **File → New → Project.**

Wybranie nowego projektu powoduje wyświetlenie okna (Rys. 3) przeznaczonego do określenia szablonu tworzonej aplikacji (*New from template*). W przypadku programu w języku C należy wybrać **Console Application** i kliknąć przycisk **Go**.



Rys. 3 Tworzenie nowego projektu

W kolejnym oknie (Rys. 4) wybieramy język, w którym będziemy pisali program.

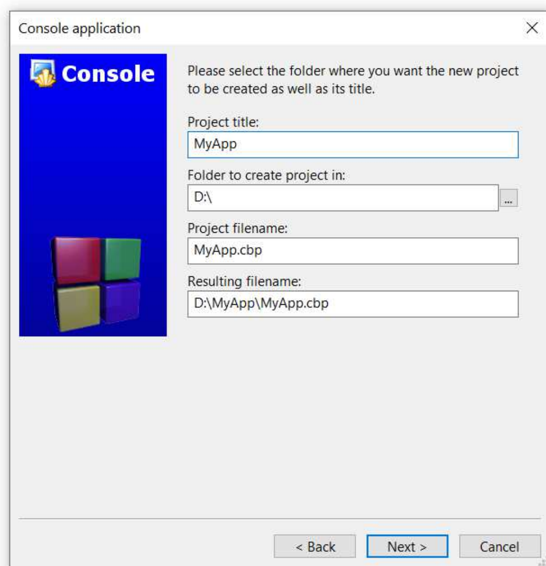


Rys. 4 Wybór języka programowania

Następnie wprowadzamy (Rys. 5):

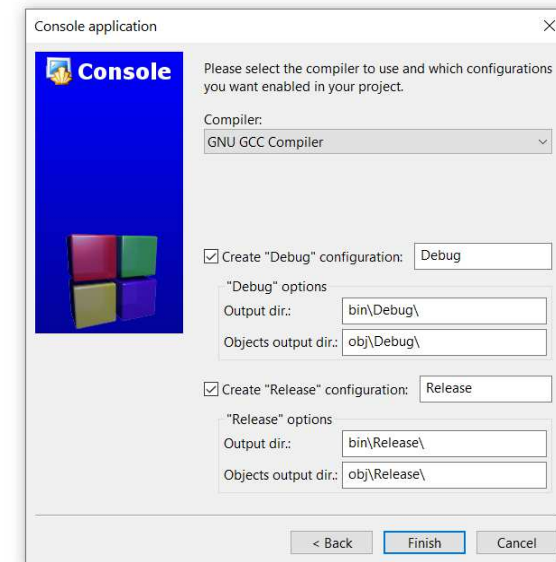
- tytuł projektu (**Project title:**), który będzie także nazwą pliku wynikowego **exe**;
- katalog, w którym będą znajdowały się pliki projektu (**Folder to create project in:**) - najprościej jest wskazać wybrany katalog na dysku poprzez kliknięcie trzech kropek (**...**).

Pozostałe dane, czyli nazwa głównego pliku projektu (**Project filename:**) oraz pełna ścieżka dostępu do tego pliku (**Resulting filename:**) zostaną automatycznie uzupełnione.



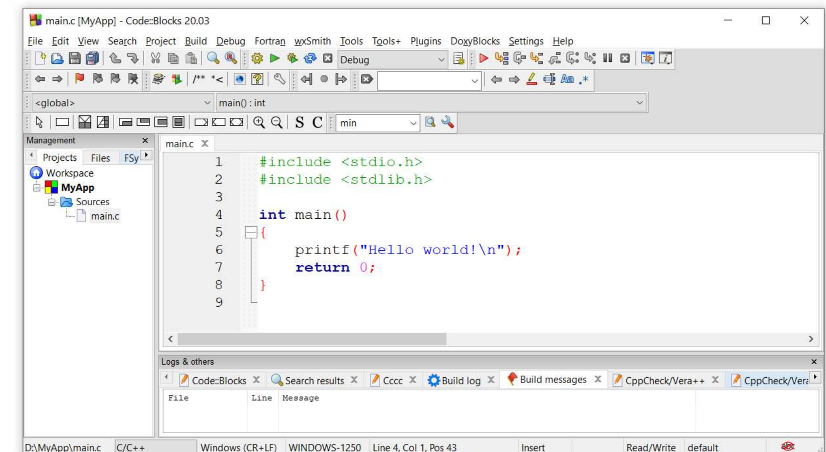
Rys. 5 Wprowadzenie tytułu projektu i wybranie katalogu z plikami projektu

W kolejnym kroku (Rys. 6) wybieramy kompilator języka C, który zostanie wykorzystany przy kompilacji programu. Domyślnie jest to **GNU GCC Compiler**. Możemy rozwinąć listę i wybrać inny kompilator, ale musi on być zainstalowany na komputerze.



Rys. 6 Wybór kompilatora języka C

Kliknięcie przycisku **Finish** kończy tworzenie projektu. Do projektu zostanie dodany plik **main.c**, którego zawartość możemy zobaczyć w edytorze (Rys. 7).



Rys. 7 Edytor z kodem domyślnego programu

2.3. 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).

2.4. Ogólna struktura programu w języku C

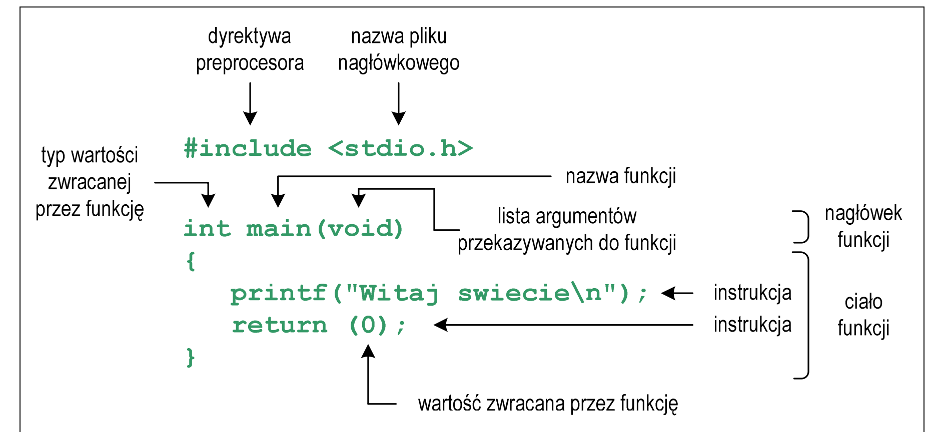
Program w języku C jest to niesformatowany plik tekstowy o odpowiedniej składni mający rozszerzenie **.c**. Najprostszy program ma następującą postać:


```
#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 szczególną 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. 2.5). 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*).

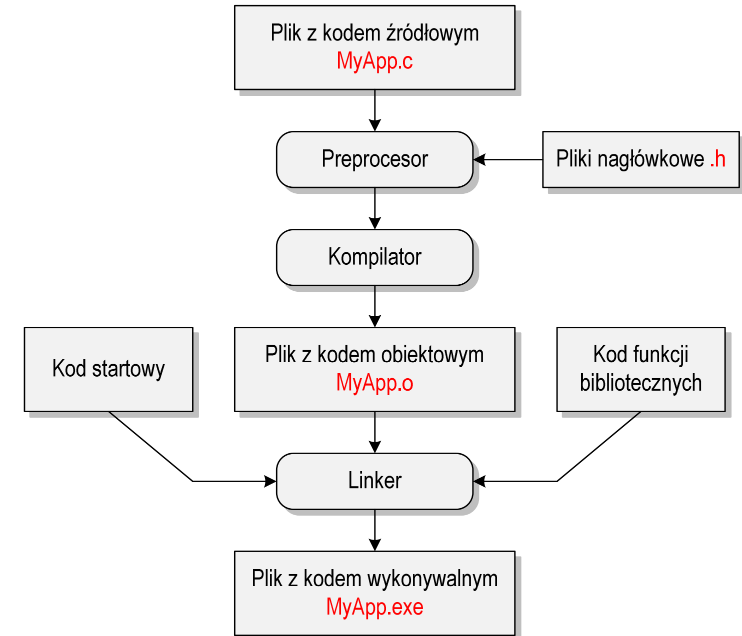
Funkcję wywołuje się podając jej nazwę (**printf()**) i (w nawiasach zwykłych) listę argumentów przekazywanych do funkcji (**"Witaj swiecie\n"**). W powyższym przykładzie do funkcji **main()** nie są przekazywane żadne argumenty, informuje o tym słowo **void** w jej nagłówku (słowo to może być pominięte).

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.

2.5. Tworzenie pliku wykonywalnego i uruchomienie programu

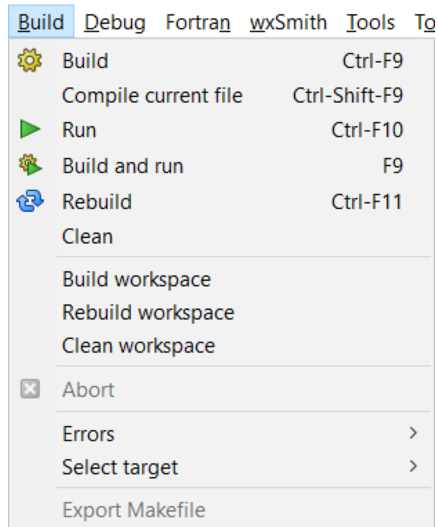
Uruchomienie programu wymaga przekształcenia pliku z kodem źródłowym na plik wykonywalny **exe**. Operacja ta składa się z kilku kroków (Rys. 9).



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 **.o**. 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 Code::Blocks do utworzenia pliku wykonywalnego i jego uruchomienia można wybrać jedną z pozycji znajdujących się w menu głównym **Build** (Rys. 10).



Rys. 10. Menu Build do kompilacji i uruchomienia programu

Poszczególne pozycje wykonują następujące operacje:

- **Build (Ctrl-F9) -** buduje projekt znajdujący się w przestrzeni roboczej (kompilowane są tylko te pliki, które uległy zmianie od czasu ostatniej kompilacji);
- **Compile current file (Ctrl-Shift-F9)** - kompiluje tylko edytowany plik z kodem źródłowym;
- **Run (Ctrl-F10)** - uruchamia aktualny projekt;
- **Build and run (F9)** - buduje aktualny projekt i uruchamia go;
- **Rebuild (Ctrl-F11) -** przebudowuje aktualny projekty znajdujący się w przestrzeni roboczej (kompilowane są wszystkie pliki niezależnie od tego czy uległy zmianie od czasu ostatniej kompilacji);
- **Clean** - usuwa wszystkie pliki będące wynikiem budowania projektu wchodzącego w skład przestrzeni roboczej.

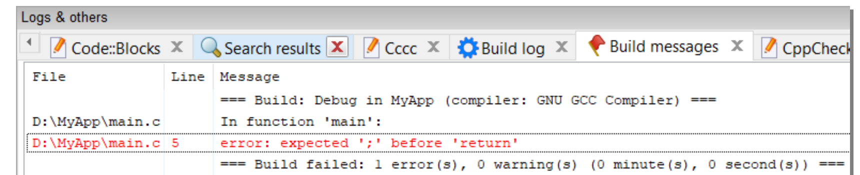
Do kompilacji i uruchomienia programu można wykorzystać także ikonki na pasku narzędziowym (Rys. 11).

Rys. 11 Ikonki do kompilacji i uruchamiania programu

Przebieg kompilacji wyświetlany jest w zakładce **Build log** okna **Logs & others**. Przykładowy przebieg kompilacji przedstawiony jest poniżej.

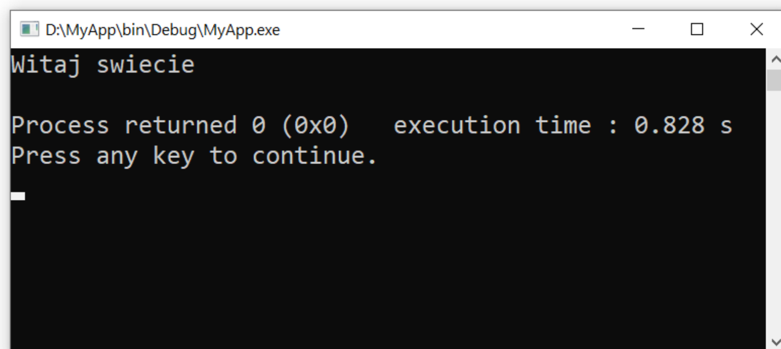
```
-------------- Build: Debug in MyApp (compiler: GNU GCC Compiler)---------------

gcc.exe -Wall -g  -c D:\MyApp\main.c -o obj\Debug\main.o
gcc.exe  -o bin\Debug\MyApp.exe obj\Debug\main.o
Output file is bin\Debug\MyApp.exe with size 53.43 KB
Process terminated with status 0 (0 minute(s), 0 second(s))
0 error(s), 0 warning(s) (0 minute(s), 0 second(s))
```

Jeśli w kodzie programu występują błędy, to w zakładce **Build messages** okna **Logs & others** wyświetlane są odpowiednie komunikaty (Rys. 12), natomiast plik wykonywalny **exe** nie jest tworzony.



Rys. 12 Okno z błędami kompilacji

Uruchomienie programu (**Run** lub **Build and run**) odbywa się automatycznie w oknie **Wiersza polecenia** (Rys. 13). Przed uruchomieniem programu system operacyjny tworzy takie okno, a następnie uruchamia w nim program. Program wyświetla tekst „**Witaj swiecie**". Dodatkowo środowisko Code::Blocks zatrzymuje na koniec program i wyświetla komunikat: „**Press any key to continue.**". Po naciśnięciu dowolnego klawisza okno **Wiersza polecenia** jest zamykane.



Rys. 13 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 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:

Press any key to continue ...

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**.

2.6. 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;}
```

2.7. 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.

2.8. 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.

Kod	Wynik
`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 mogą być wyświetlone 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	%	%%

2.9. 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:  22-02-2021 08:30
  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;
}
```

2.10. 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

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c	1	fatal error: studio.h: No such file or directory
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

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

- brak średnika na końcu wiersza

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c		In function 'main':
D:\MyApp\main.c	5	error: expected ';' before 'return'
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

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

- brak nawiasu klamrowego kończącego program

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c		In function 'main':
D:\MyApp\main.c	6	error: expected declaration or statement at end of input
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===


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

- brak cudzysłowu kończącego łańcuch

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c		In function 'main':
D:\MyApp\main.c	5	warning: missing terminating " character
D:\MyApp\main.c	5	error: missing terminating " character
D:\MyApp\main.c	6	error: expected expression before 'return'
D:\MyApp\main.c	6	error: expected ';' before '}' token
		=== Build failed: 3 error(s), 1 warning(s) (0 minute(s), 0 second(s)) ===

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

- brak nawiasów po funkcji **main**

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c	4	error: expected '=', ',', ';', 'asm' or '__attribute__' before '{' token
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

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

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

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c		In function 'main':
D:\MyApp\main.c	6	error: 'return0' undeclared (first use in this function)
D:\MyApp\main.c	6	note: each undeclared identifier is reported only once for each functi...
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

```
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

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c		In function 'main':
D:\MyApp\main.c	5	warning: implicit declaration of function 'Printf'; did you mean 'printf'?...
obj\Debug\main.o		In function `main':
D:\MyApp\main.c	5	undefined reference to `Printf'
		error: ld returned 1 exit status
		=== Build failed: 2 error(s), 1 warning(s) (0 minute(s), 0 second(s)) ===

```
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

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
C:\Program Fi...		undefined reference to `WinMain'
		error: ld returned 1 exit status
		=== Build failed: 2 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

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

- średnik w nagłówku funkcji **main**

File	Line	Message
		=== Build: Debug in MyApp (compiler: GNU GCC Compiler) ===
D:\MyApp\main.c	4	error: expected identifier or '(' before '{' token
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

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 Code::Blocks - jako typ projektu (szablonu) wybierz **Console Application**;
   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 zakończeniem jego działania (**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;
   g) 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.

```
**********************************
*          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 błędy kompilacji. Dla każdego błędu podaj: kod źródłowy zwierający błąd, otrzymany komunikat kompilatora, tłumaczenie komunikatu na język polski, wyjaśnienie na czym polegał błąd.

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://www.codeblocks.org/ - Code::Blocks

5. Pytania kontrolne

1. Omów sposób tworzenia projektu, kompilacji oraz uruchamiania programu w środowisku Code::Blocks.
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.