INFORMATYKA

C

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
ELEKTROTECHNIKA
DR INŻ. MICHAŁ ŁANCZONT

III

Wprowadzenie

C – imperatywny, strukturalny język programowania wysokiego poziomu stworzony na początku lat siedemdziesiątych XX w. przez Dennisa Ritchiego do programowania systemów operacyjnych i innych zadań niskiego poziomu.

imperatywny - paradygmat programowania, który opisuje proces wykonywania jako sekwencję instrukcji zmieniających stan programu, wyraża żądania jakichś czynności do wykonania.

Wprowadzenie

Język C był dominującym narzędziem w którym tworzono systemy operacyjne i aplikacje

W 1983 roku ANSI powołało komitet `X3J11` w celu ustanowienia standardu języka C.

Standard został zatwierdzony w 1989 roku jako ANSI X3.159-1989 *"Programming Language C"* , standardowe C lub C89.

Wprowadzenie

W 1990 roku standard ANSI C został zaadoptowany przez ISO jako norma ISO/IEC 9899:1990 – wersja C90.

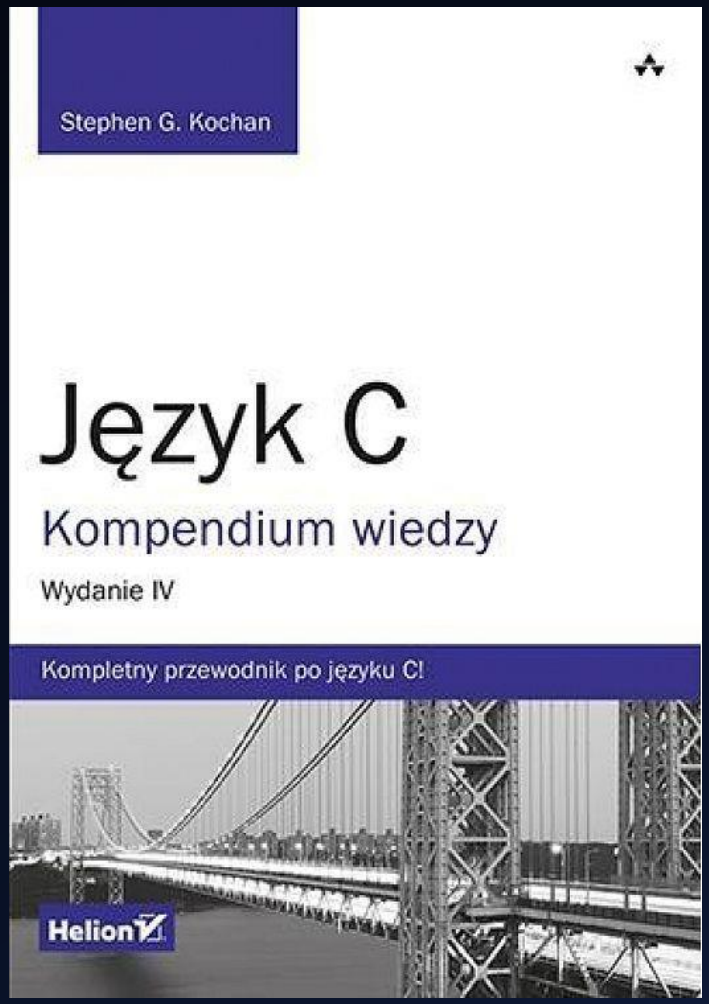
W 1999 roku ISO opublikowało normę ISO/IEC 9899:1999 – wersja C99.

Ostatnia norma została opublikowana w 2011 roku pod nazwą ISO/IEC 9899:2011 – wersja C11

Trwają prace nad nową normą, roboczo oznaczona jako C2a

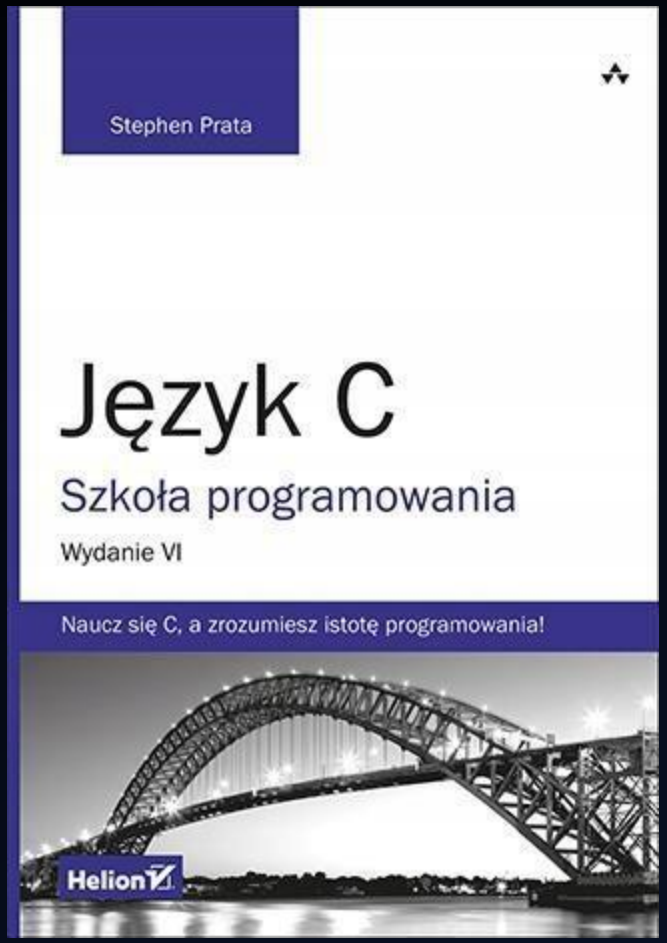
Literatura

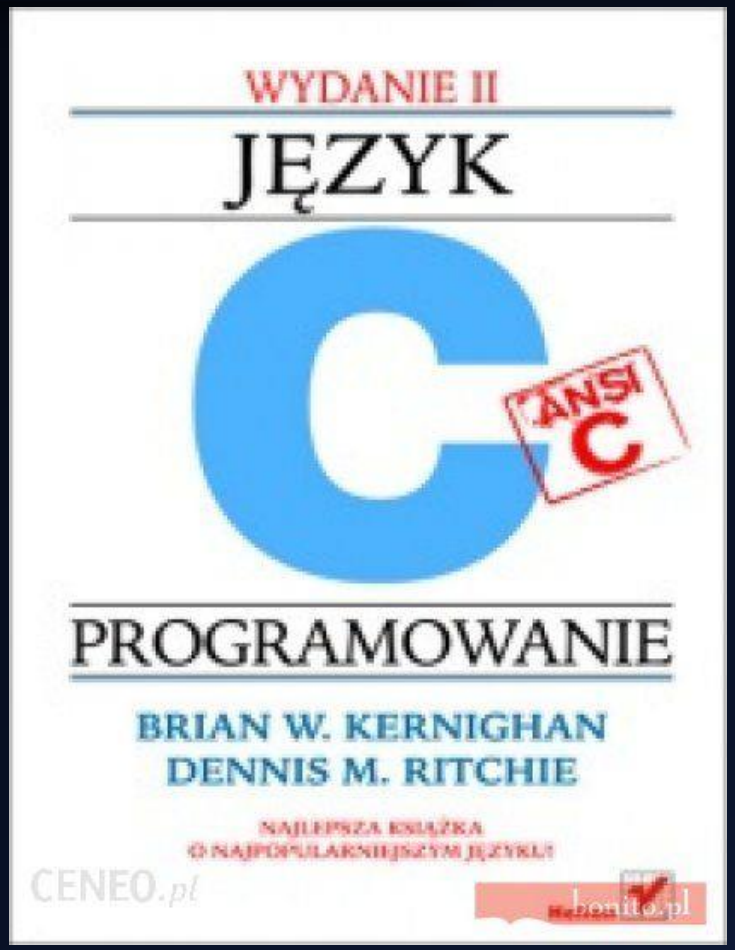
1. S. G. Kochan, *Język C Kompedium wiedzy. Wydanie IV*, Helion 2015



Literatura

2. B. W. Kernighan, D. M. Ritchie, ***Język ANSI C. Programowanie. Wydanie II***, Helion 2010





3. S. Prata, Język C. Szkoła programowania, Wydanie VI, Helion 2016

Zakres wykładu

- Kompilatory
- Edytory i środowiska dedykowane
- Systemy liczbowe
- Zmienne i stałe
- Procedury We/Wy
  - `printf(), puts(), putchar()`
  - `scanf(), gets(), getchar()`
- Liczby zespolone
- Działania i funkcje matematyczne
- Liczby losowe
- Komentarze

Kompilatory

Podstawowy kompilator dla języka C jest dostępny w GCC, który jest zestawem kompilatorów dla systemów linux'owych (C, C++, Objective-C, Fortran, Ada, i Go)

Dla systemów Windows można zainstalować kompilator za pomocą narzędzi MinGW, MinGW64 lub CygWin

Wraz z dedykowanymi środowiskami programistycznymi instalowany jest dedykowany im kompilator (Microsoft Visual Studio, Borland C++, Dev C++, QtCreator

Kompilator i środowisko programowania

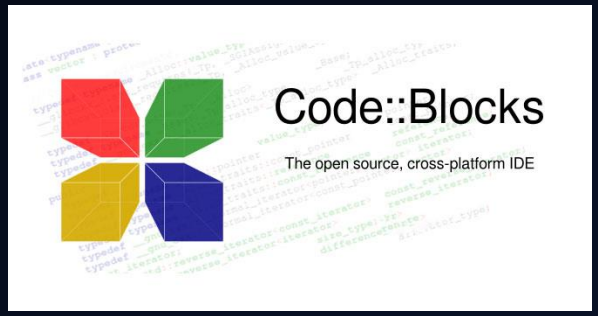
Wykład i laboratorium oparte są o zestaw:

1. Kompilator **MinGW**
   http://www.mingw.org
2. Zalecany edytor **Code::Blocks**
   http://www.codeblocks.org/

Edytor zapewnia pełne wsparcie przy programowaniu (system podpowiedzi składni), oraz zapewnia bardzo dobre wsparcie przy analizie kodu w czasie procesu debugowania.

Code::Bloks

- http://www.codeblocks.org/
- Aktualna wersja 20.03
- Wieloplatformowe, zintegrowane środowisko programistyczne IDE, na licencji GNU
- Wspiera programowanie w:
  - C/C++
  - Fortran
- Rozszerzanie funkcjonalności poprzez pluginy
- Wspiera rysowanie schematów blokowych Nassi-Shneiderman'a
- Dobry debuger



Alternatywne IDE

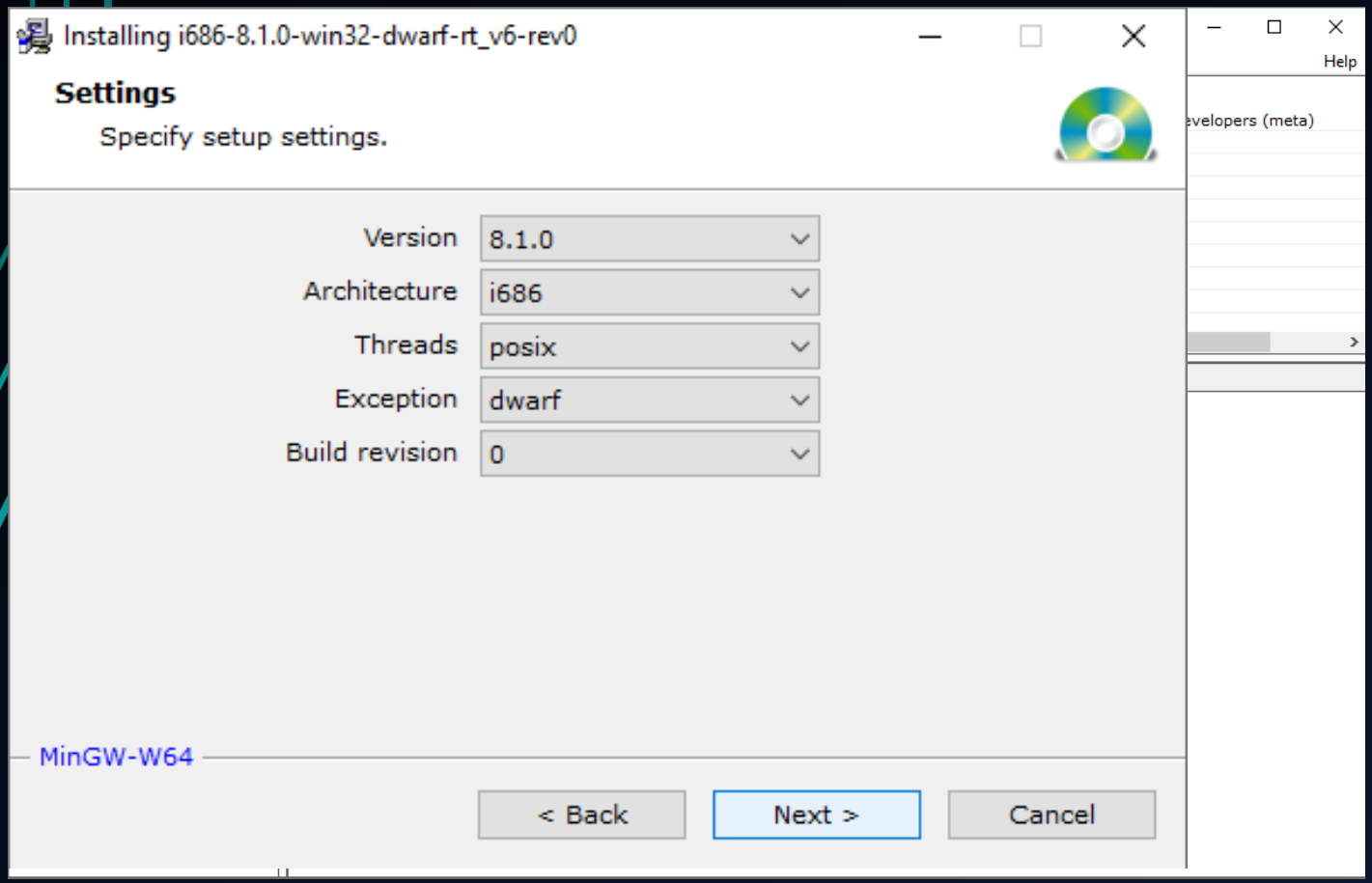
- **Dev-C++** - Zintegrowane środowisko programistyczne IDE wpierające programowanie w C i C++, zintegrowane z kompilatorem. Rozwijana przez firmę **Embarcadero**

- **Atom** – Rozbudowany edytor tekstowy wspierający programowanie w większości popularnych języków programowania, możliwość rozszerzania funkcjonalności poprzez plugin'y

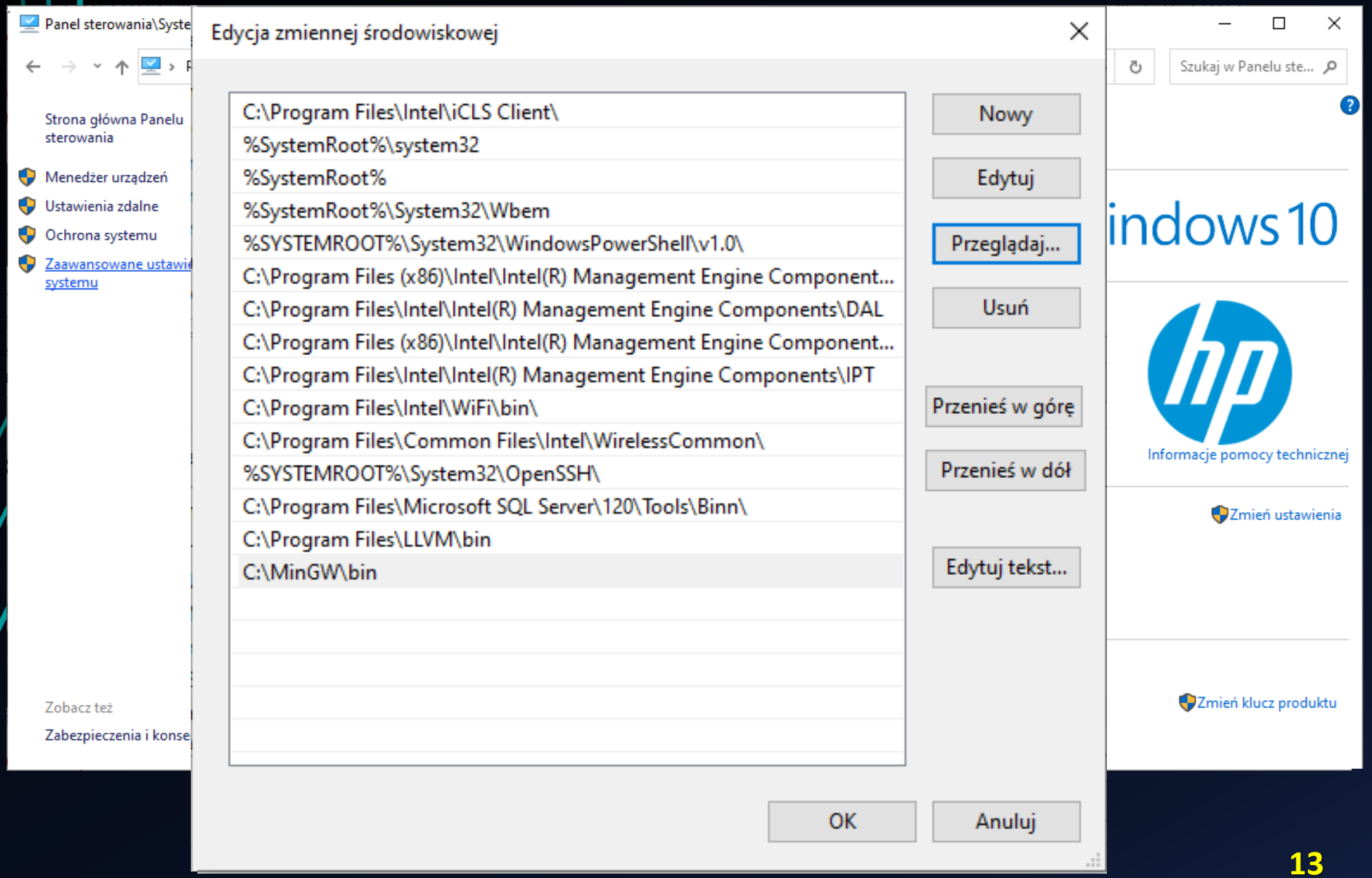
- **Visual Studio Code** – Rozbudowany edytor tekstowy wspierający programowanie w większości współczesnych językach programowania, najpopularniejszy obecnie edytor kodu

Kompilator

- Dla środowiska Windows zalecany zestaw kompilatorów to
  - MinGW - http://www.mingw.org/
  - minGW64 - http://mingw-w64.org/doku.php

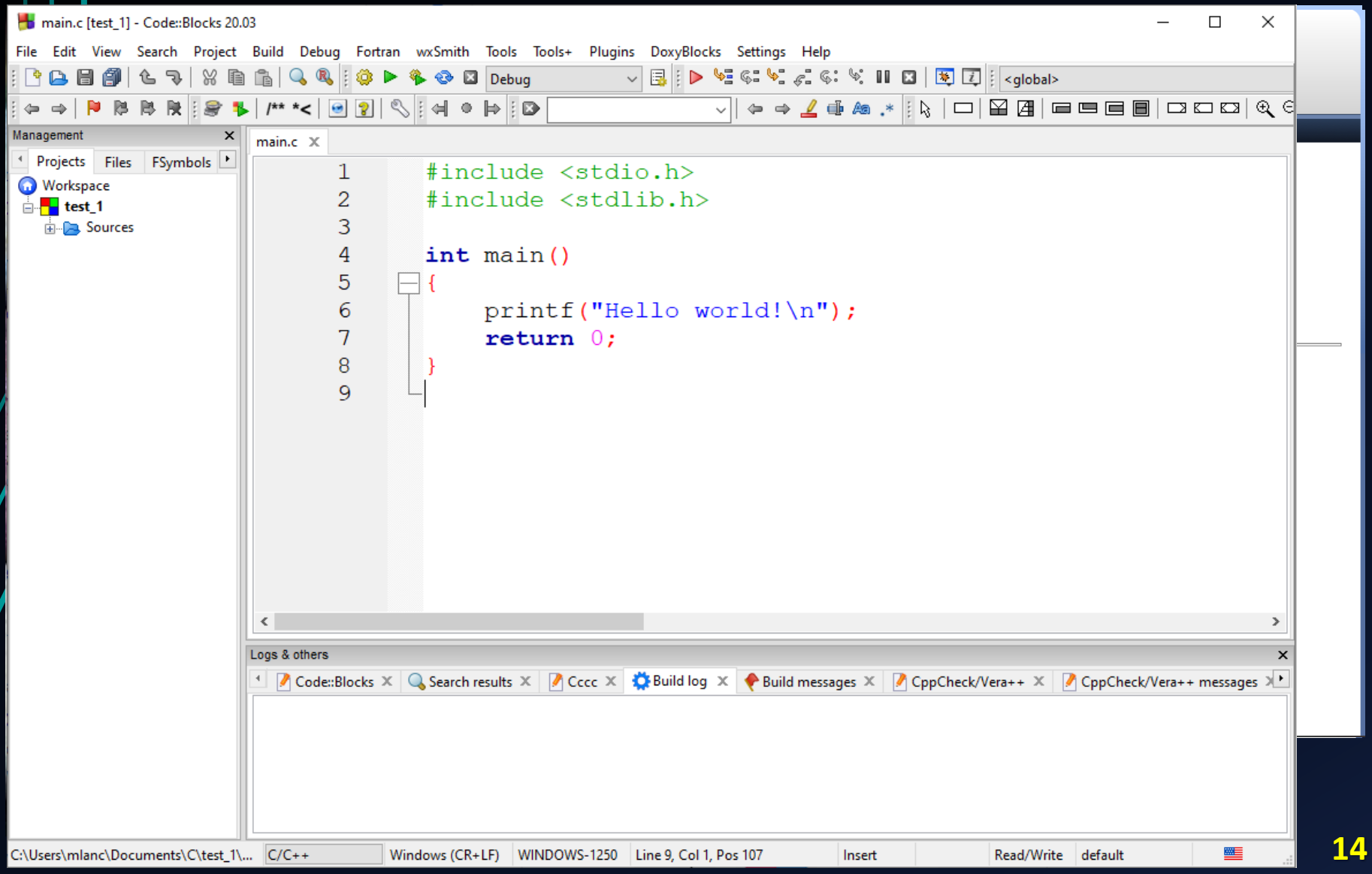


Konfiguracja minGW



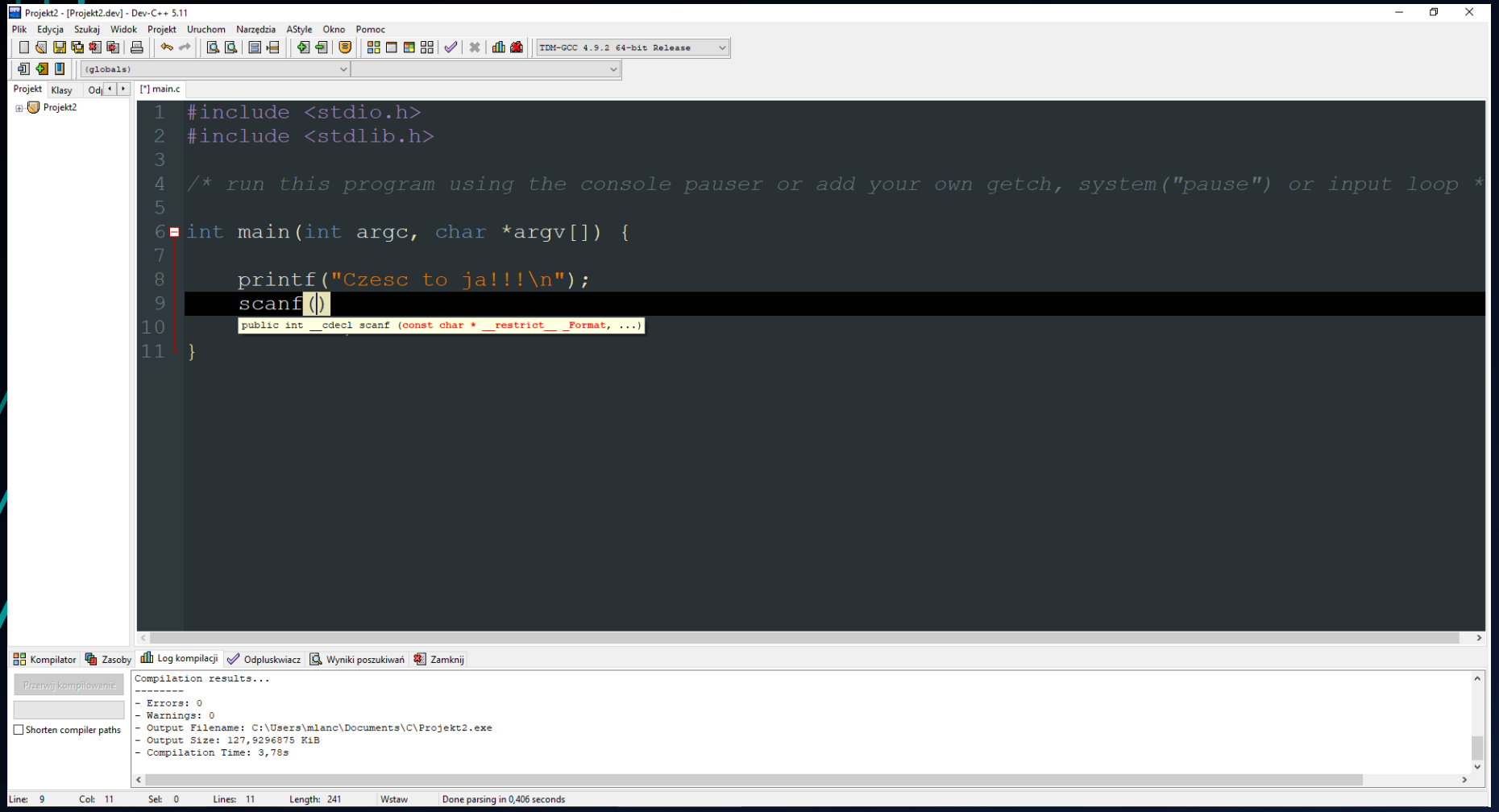
Code::Blocks

http://www.codeblocks.org/



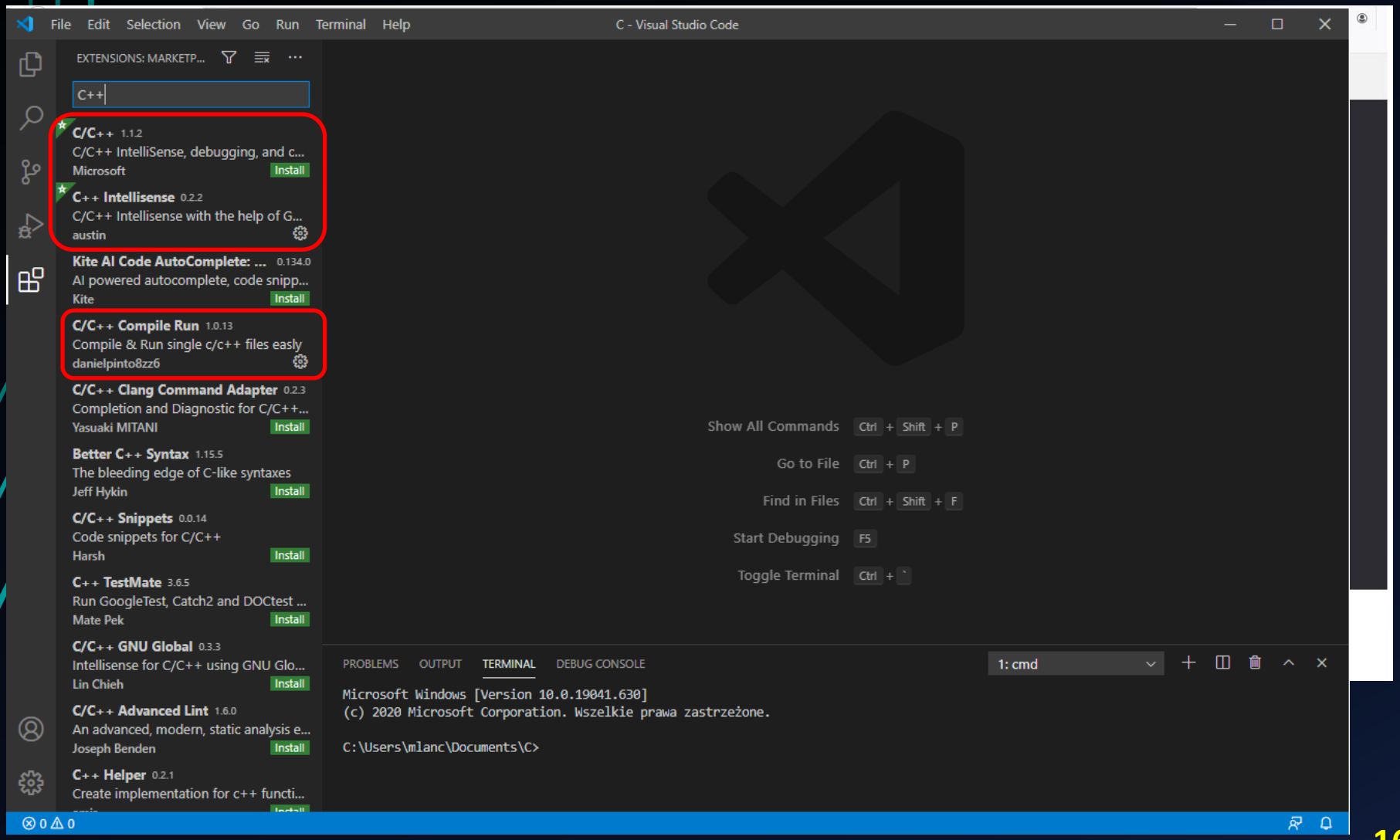
DEV-C++

https://sourceforge.net/projects/orwelldevcpp/files/latest/download



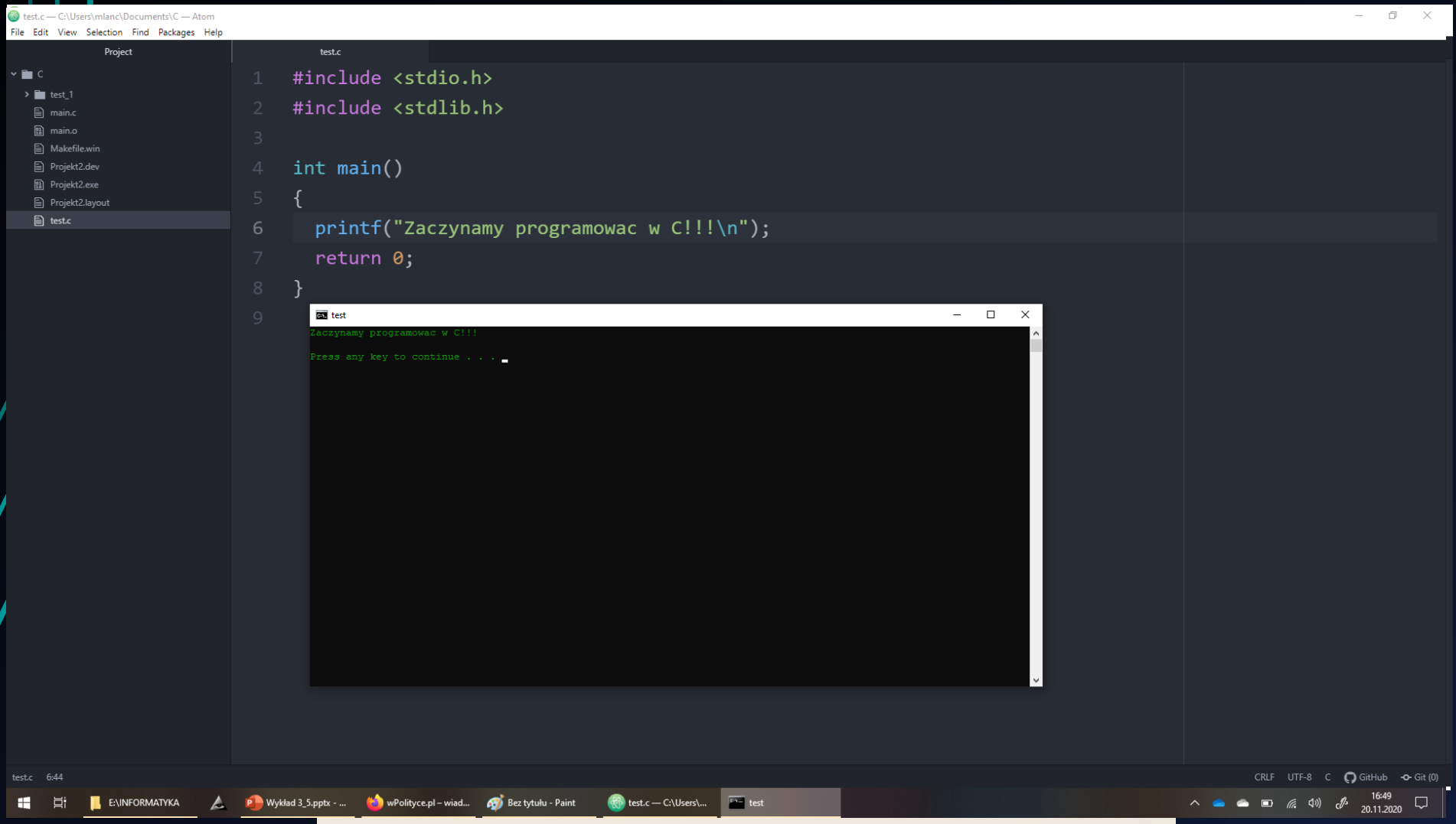
Visual Studio Code

https://code.visualstudio.com/

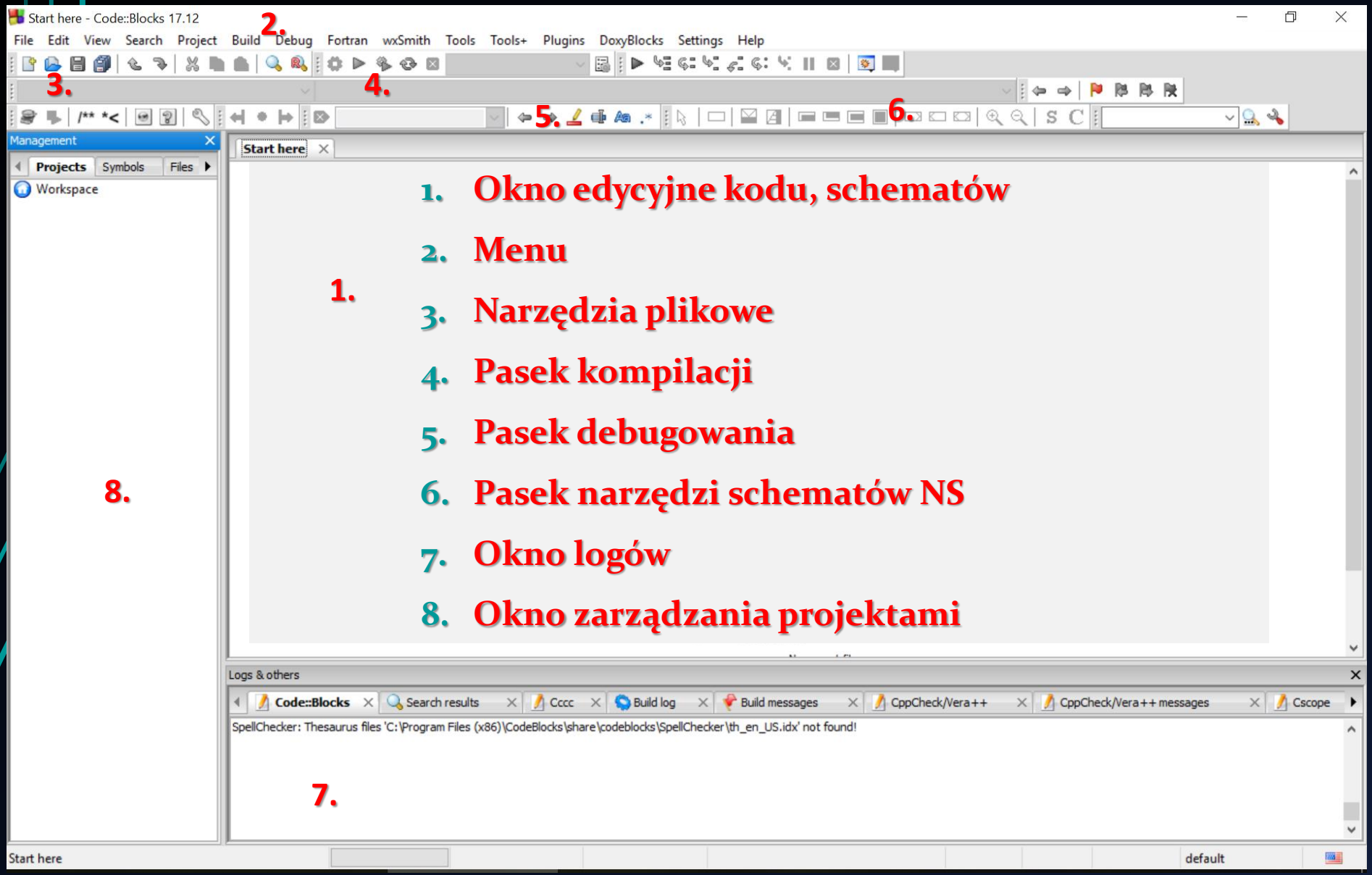


ATOM

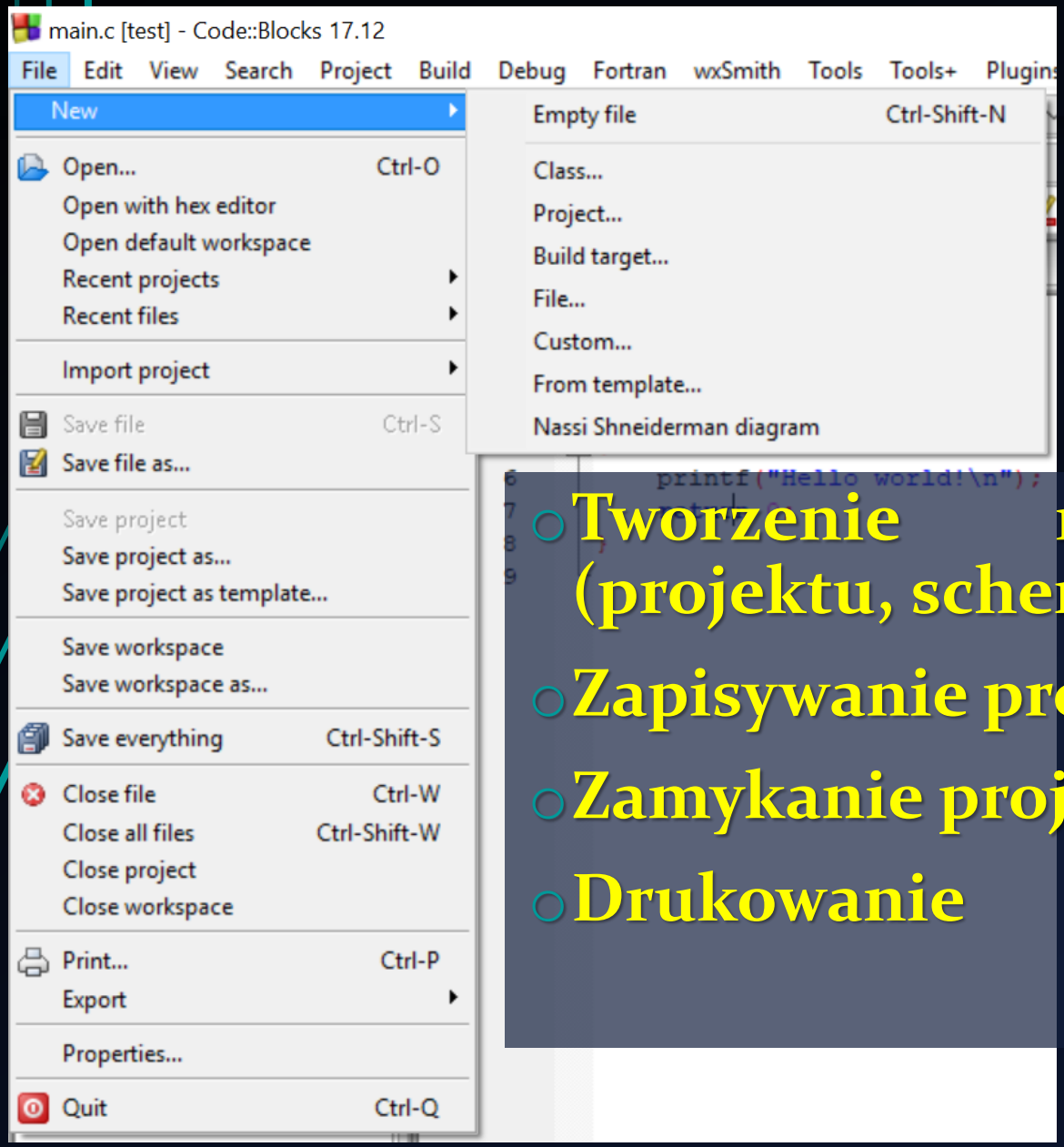
https://atom.io/



Środowisko Code::Blocks

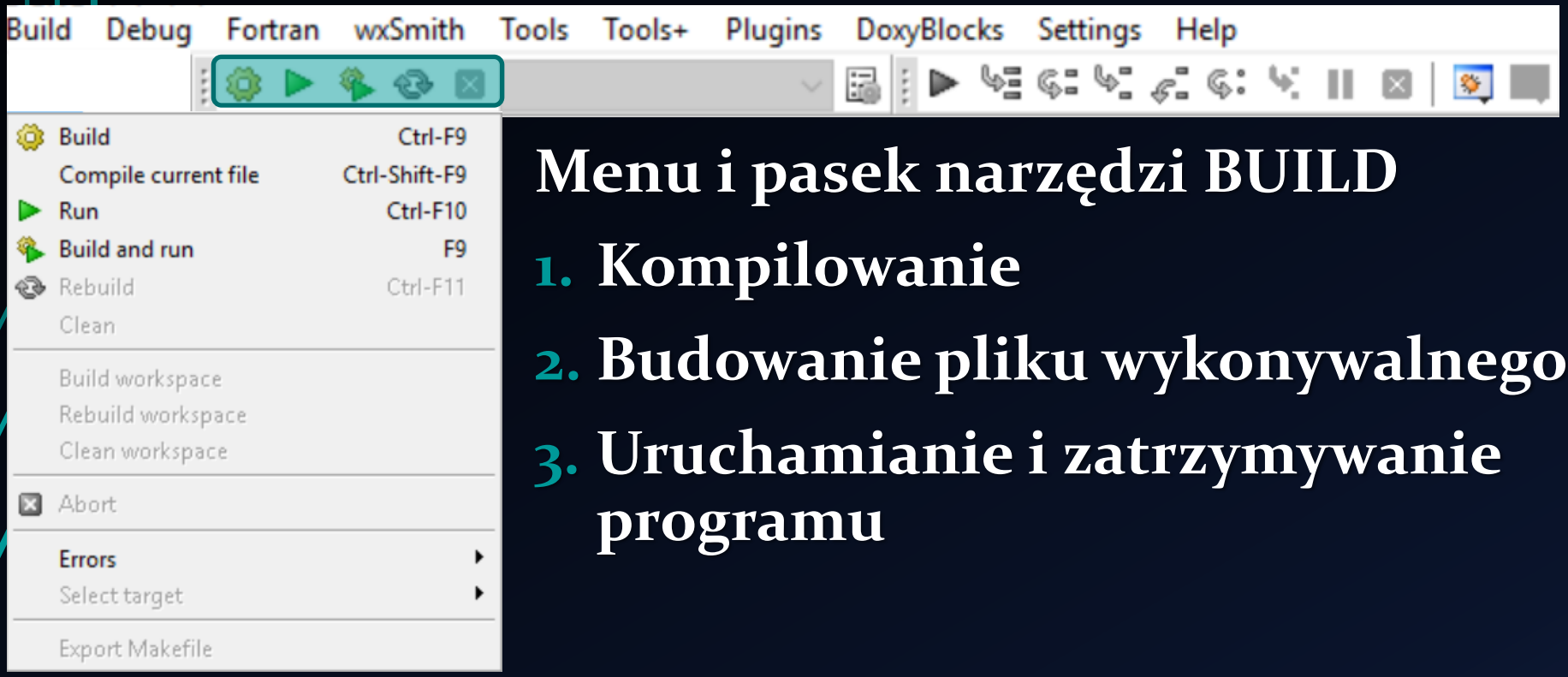


Środowisko Code::Blocks



- Tworzenie nowego pliku (projektu, schematu NS itp.)
- Zapisywanie projektu
- Zamykanie projektu
- Drukowanie

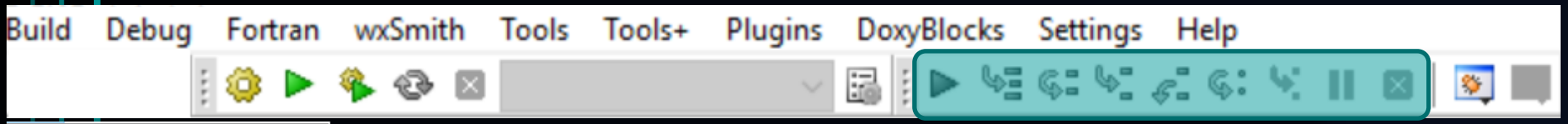
Środowisko Code::Blocks

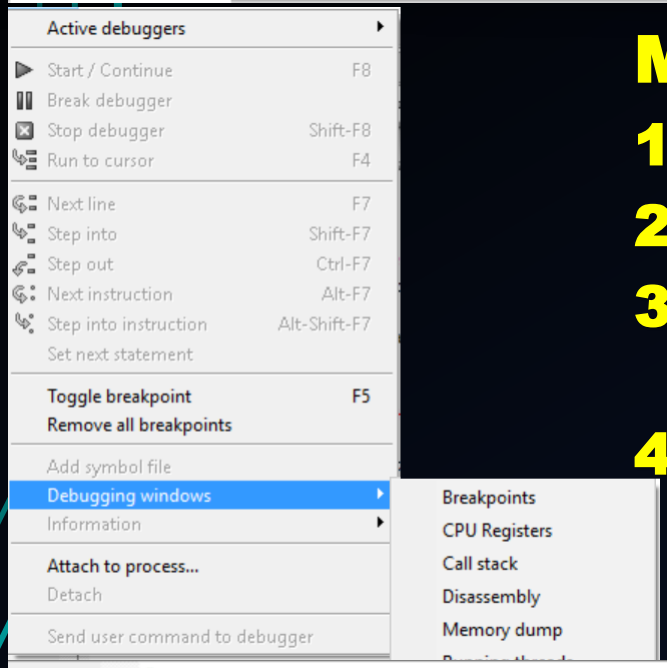


Menu i pasek narzędzi BUILD

1. Kompilowanie
2. Budowanie pliku wykonywalnego
3. Uruchamianie i zatrzymywanie programu

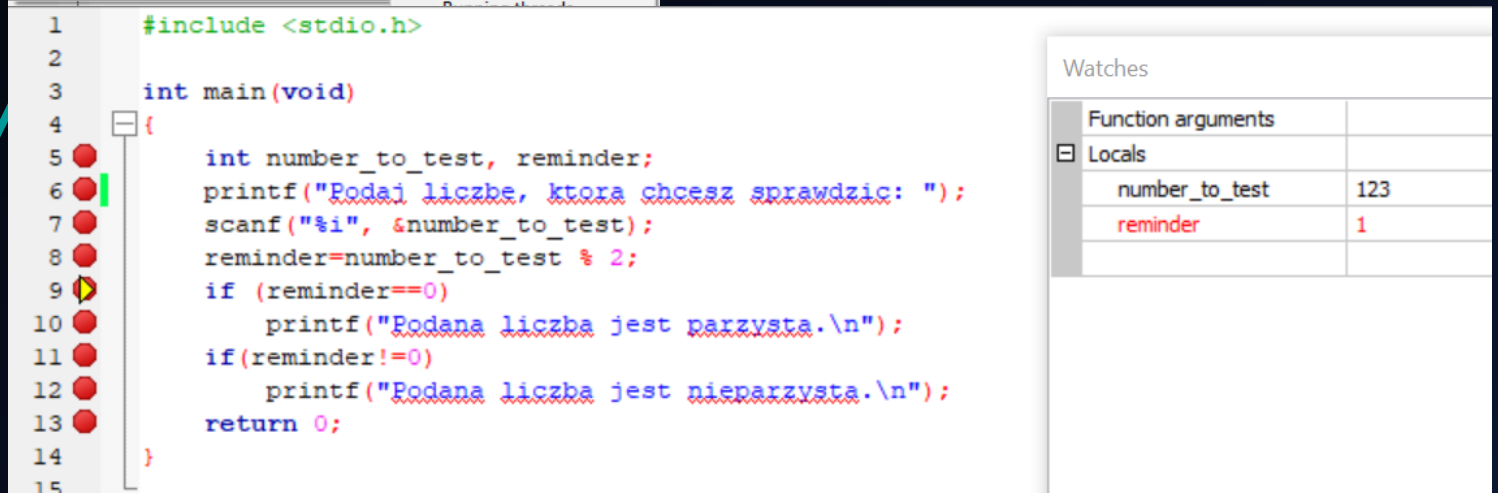
Środowisko Code::Blocks





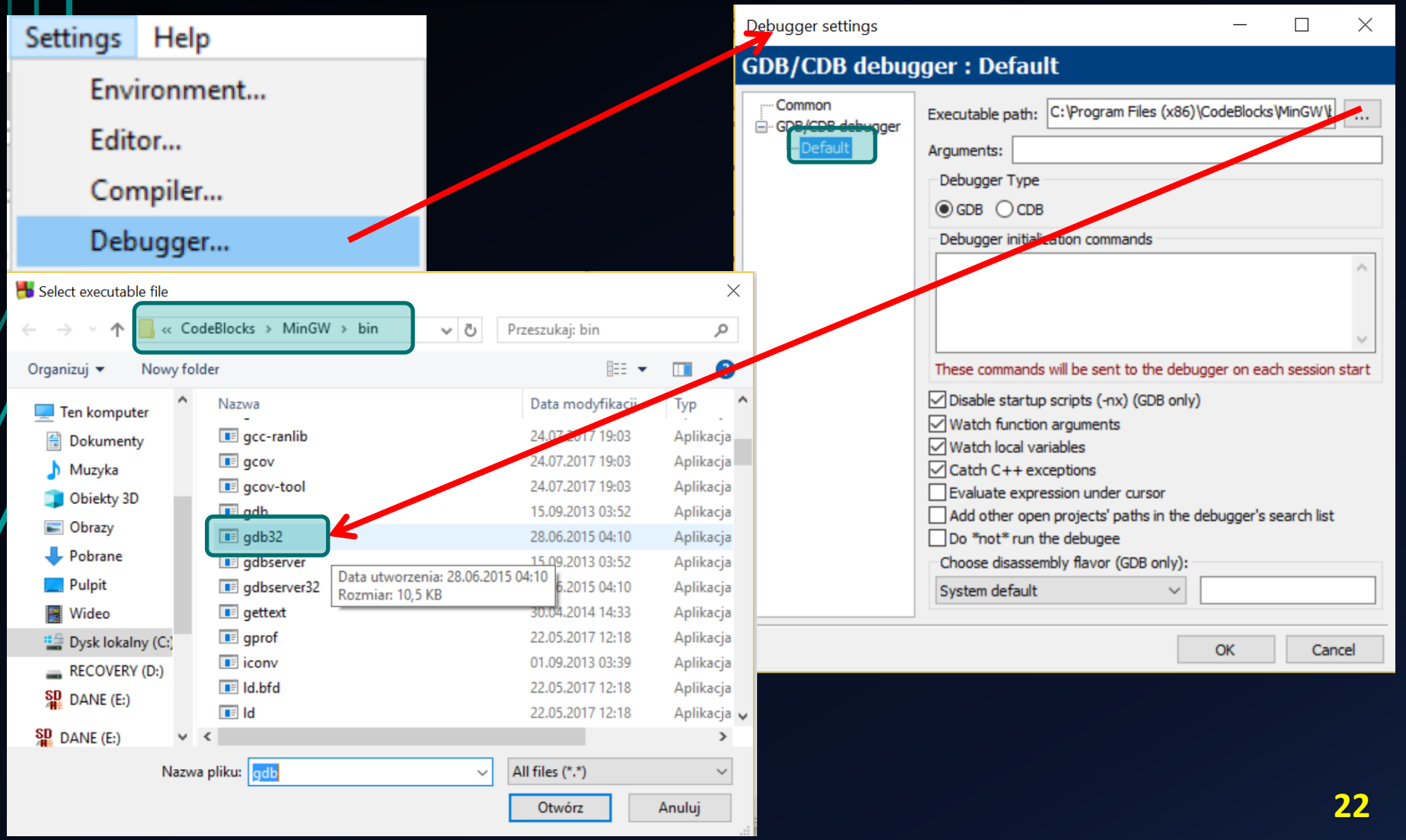
Menu i pasek DEBUGERA

1. **Analiza działania kodu**
2. **Oznaczanie breakpoint'ów**
3. **Okna z danymi wspomagającymi analizę algorytmu**
4. **Sterowanie analizą**
   - A. **Wchodzenie do funkcji**
   - B. **Omijanie funkcji**

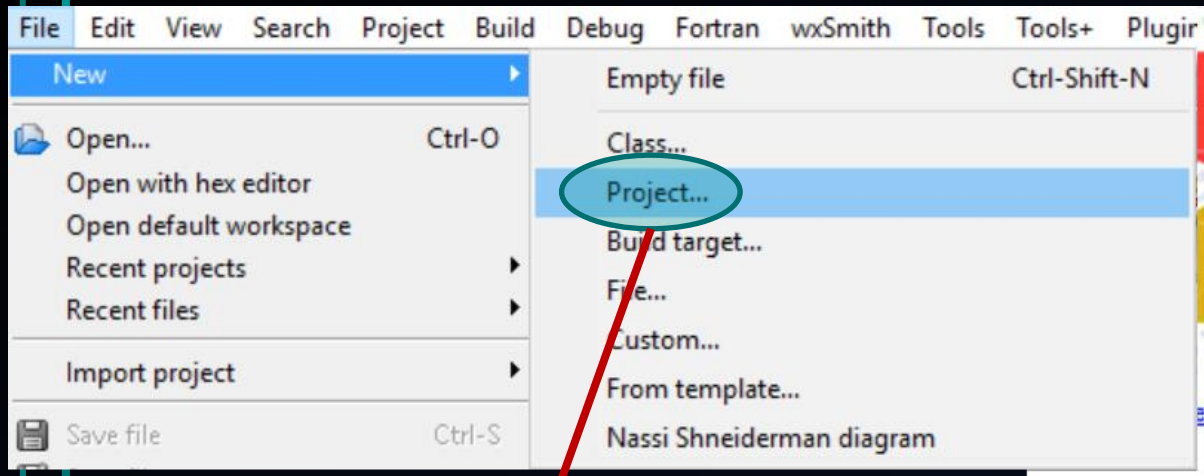


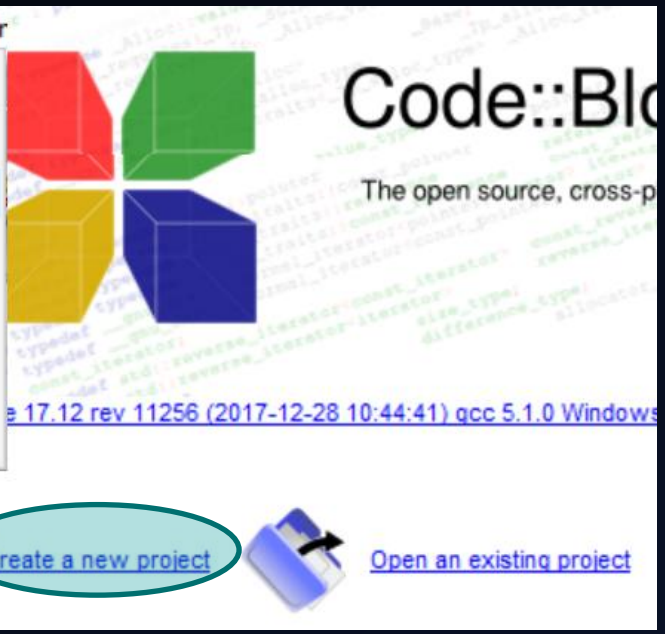
Debuger

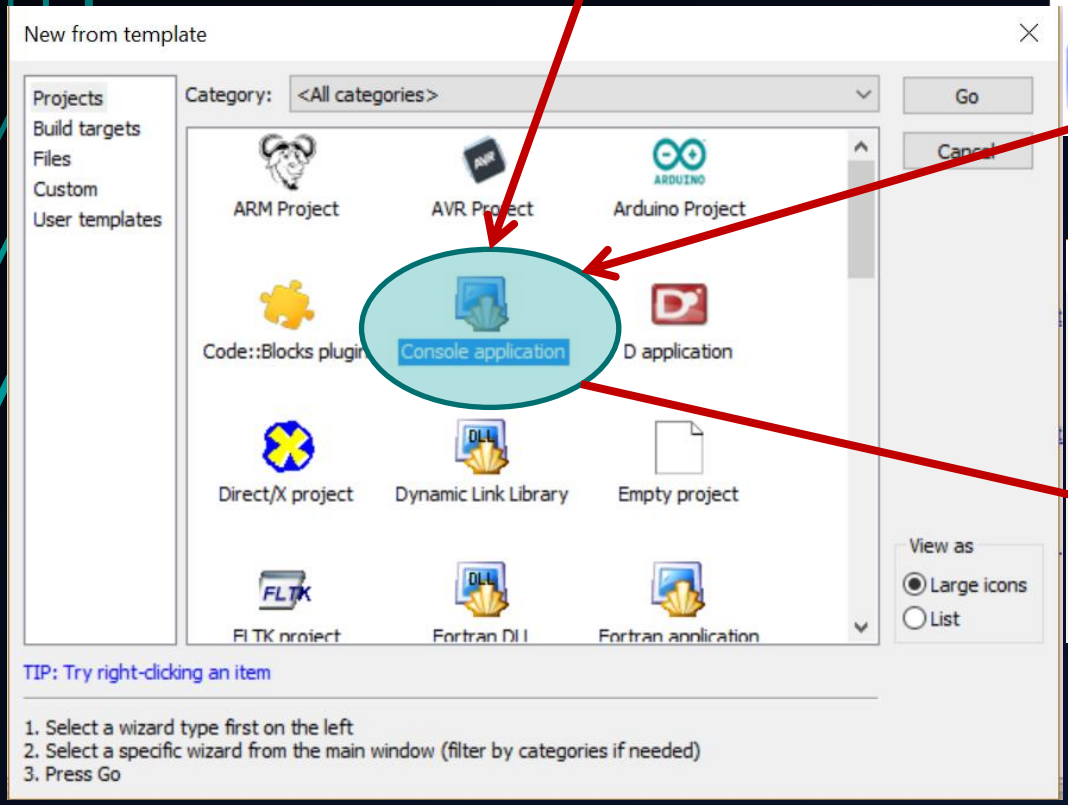
Przy instalacji z dołączonym kompilatorem MinGW konieczne jest ręczne wskazanie pliku debugera.

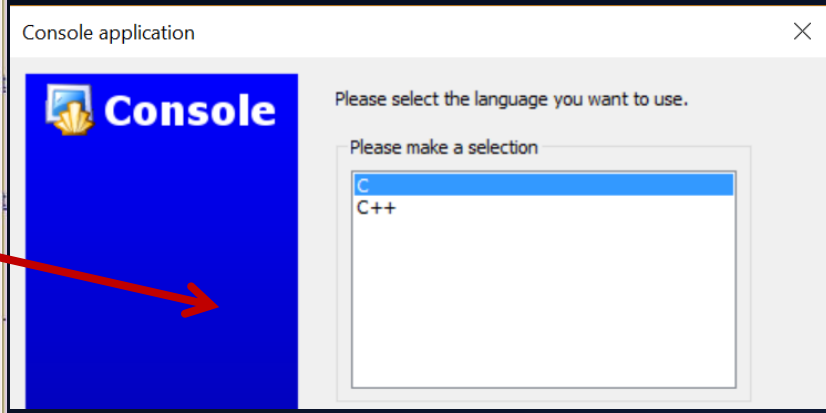


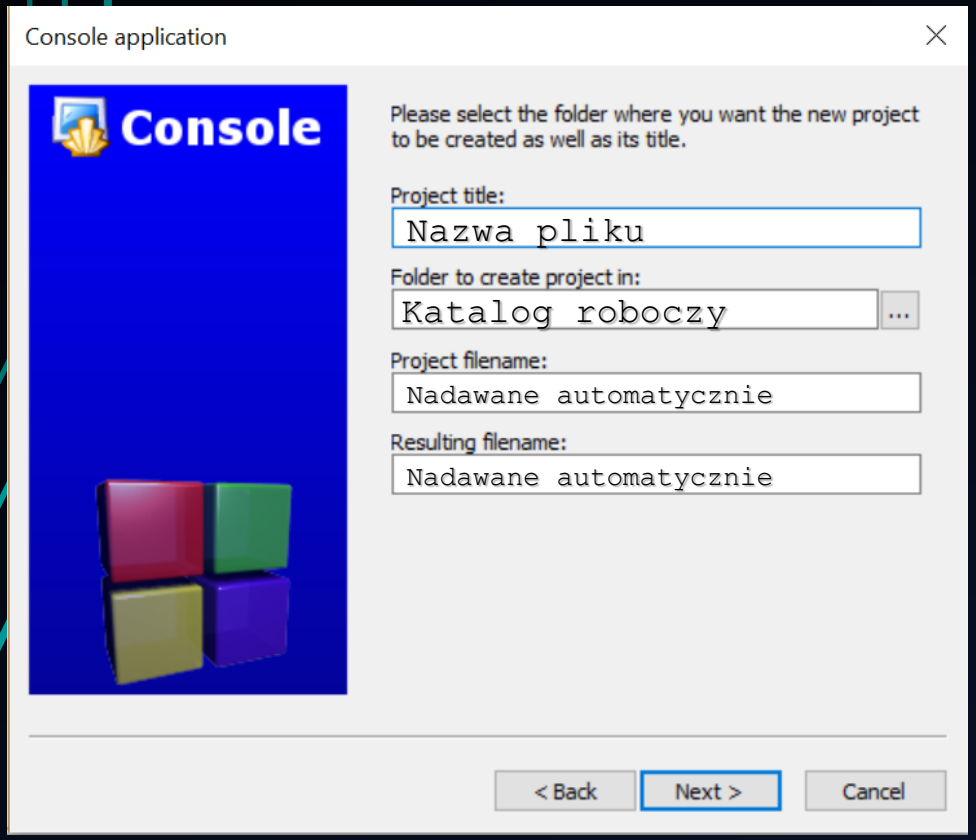
Code::blocks

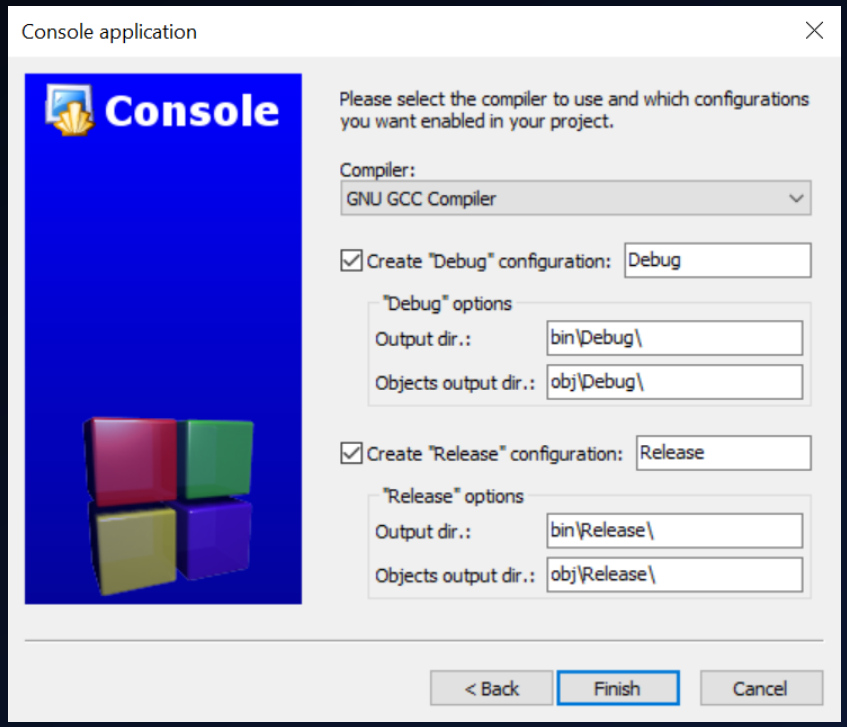

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugir
New
Open... Ctrl-O
Open with hex editor
Open default workspace
Recent projects
Recent files
Import project
Save file Ctrl-S
Empty file Ctrl-Shift-N
Class...
Project...
Build target...
File...
Custom...
From template...
Nassi Shneiderman diagram


Code::Blo
The open source, cross-p
17.12 rev 11256 (2017-12-28 10:44:41) gcc 5.1.0 Windows
Create a new project
Open an existing project


New from template
Projects
Build targets
Files
Custom
User templates
Category: <All categories>
Go
Cancel
ARM Project
AVR Project
Arduino Project
Code::Blocks plugin
Console application
D application
Direct/X project
Dynamic Link Library
Empty project
FLTK project
Fortran DLL
Fortran application
View as
Large icons
List
TIP: Try right-clicking an item
1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go


Console application
Console
Please select the language you want to use.
Please make a selection
C
C++


Console application
Console
Please select the folder where you want the new project to be created as well as its title.
Project title:
Nazwa pliku
Folder to create project in:
Katalog roboczy
...
Project filename:
Nadawane automatycznie
Resulting filename:
Nadawane automatycznie
< Back
Next >
Cancel


Console application
Console
Please select the compiler to use and which configurations you want enabled in your project.
Compiler:
GNU GCC Compiler
Create "Debug" configuration:
Debug
"Debug" options
Output dir.:
bin\Debug\
Objects output dir.:
obj\Debug\
Create "Release" configuration:
Release
"Release" options
Output dir.:
bin\Release\
Objects output dir.:
obj\Release\
< Back
Finish
Cancel

Linux

- W wielu dystrybucjach kompilator języka C/C++ jest już preinstalowany

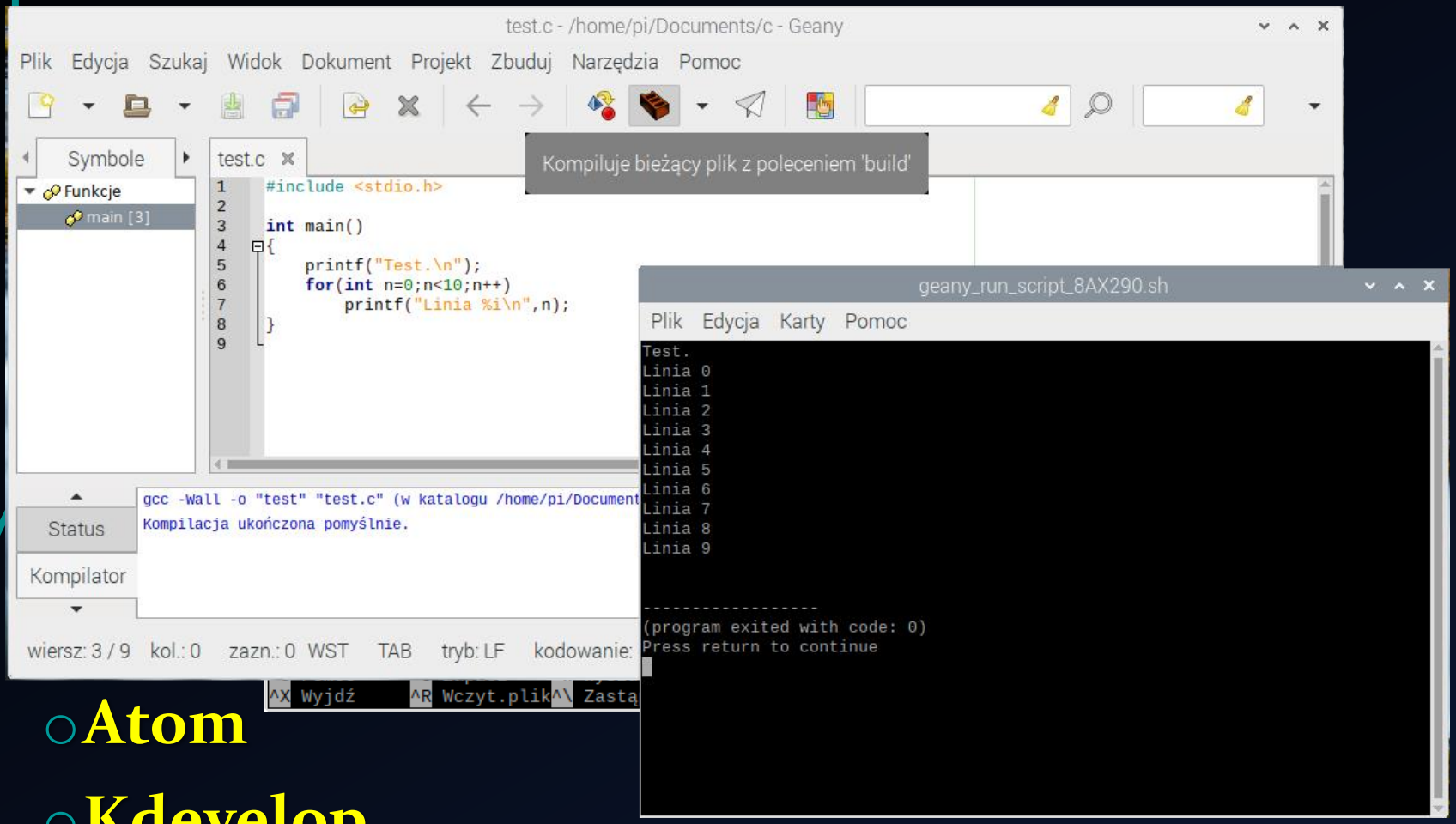
```
gcc --version
```

- W dystrybucjach opartych na Debianie (Ubuntu, Kubuntu, Raspbian itd.)

```
sudo apt install build-essential
sudo apt-get install manpages-dev
```

- W większości dystrybucji za pomocą graficznego instalatora można doinstalować wymagane pakiety

Środowiska programistyczne



- Atom
- Kdevelop

AI w programowaniu

Download it here

Languages

Free

- Java
- HTML/CSS
- C Based
- Go
- Typescript
- Scala
- Kotlin
- Javascript
- Less
- PHP
- Ruby
- Bash

Freemium

- Python

Editors

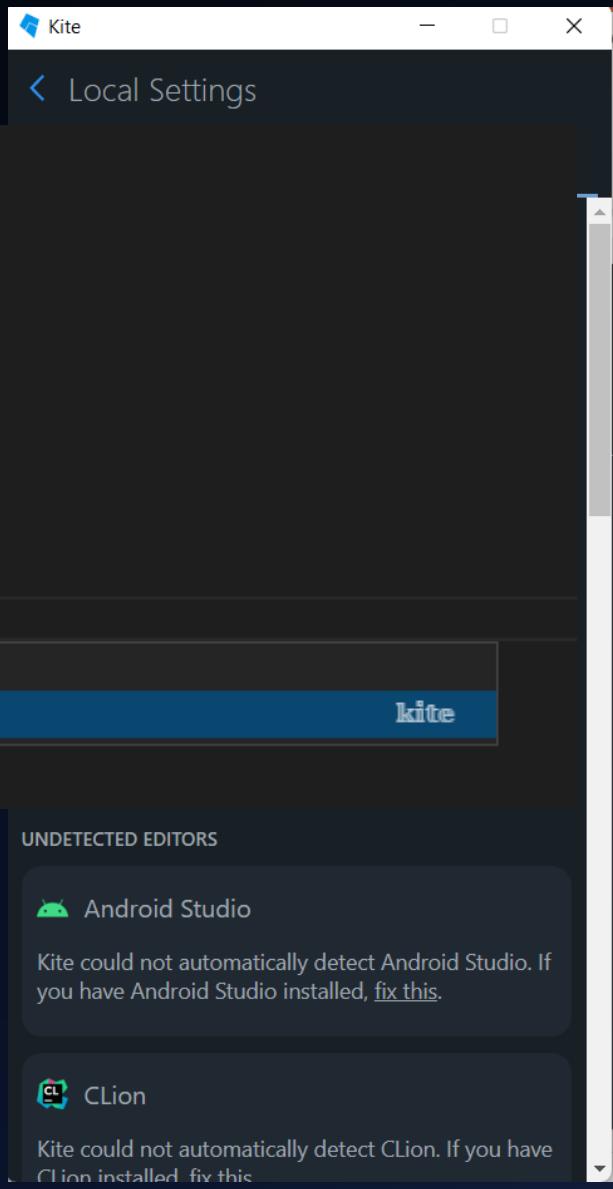
- Jupyter Lab
- VS Code
- IntelliJ
- PyCharm
- Sublime
- Spyder
- Webstorm
- Vim
- Atom
- CLion
- PhpStorm
- Rider
- RubyMine
- AppCode
- GoLand
- Android Studio

kite

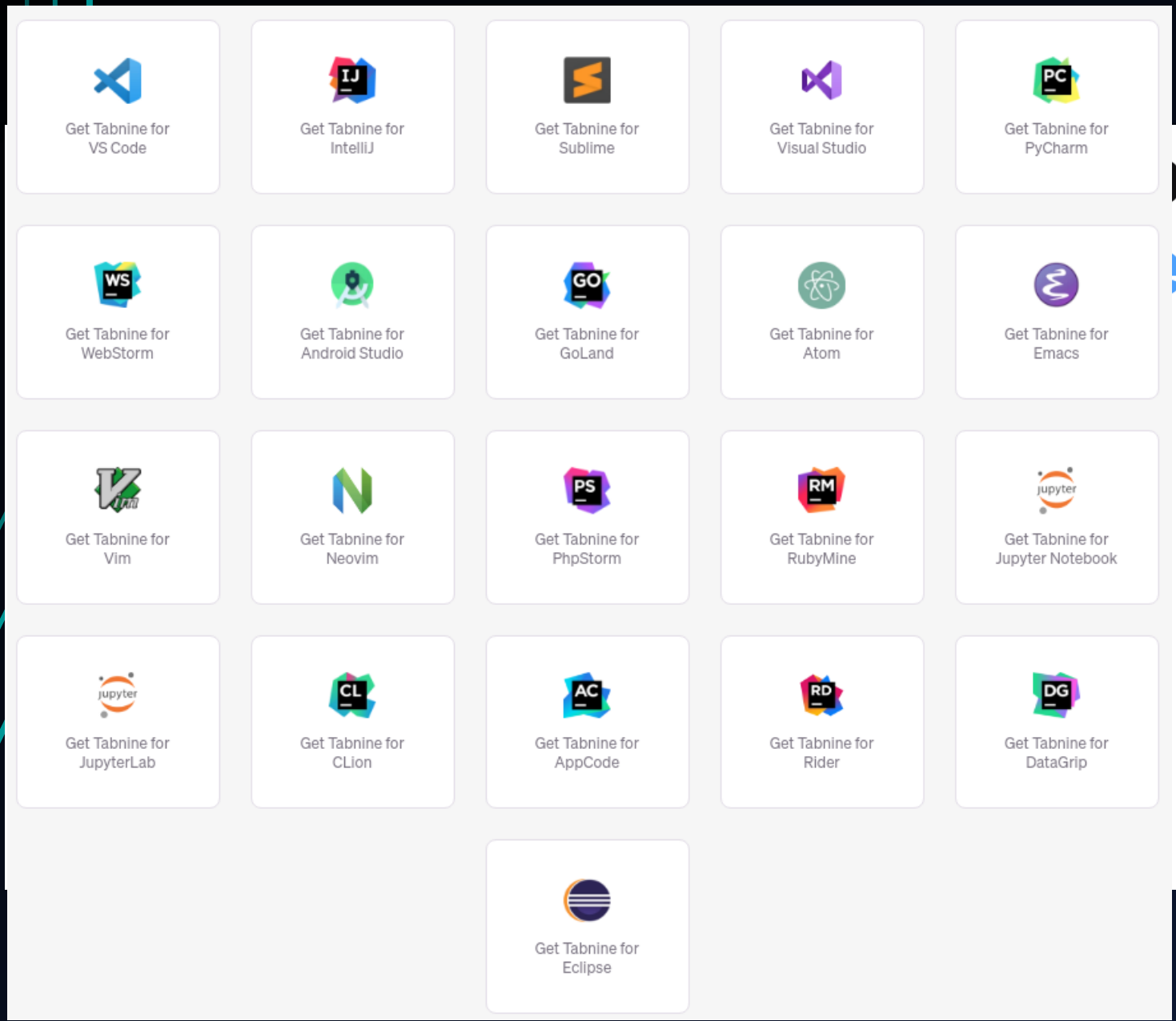
testKite.c

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

int main()
{
    int i;
    int *tabela;
    printf("Podaj rozmiar tablicy: ");
    scanf("%i",&i)
    tabela = (int*) calloc(i, sizeof(int));
}
```



Tabnine



opular

eworks

CSS

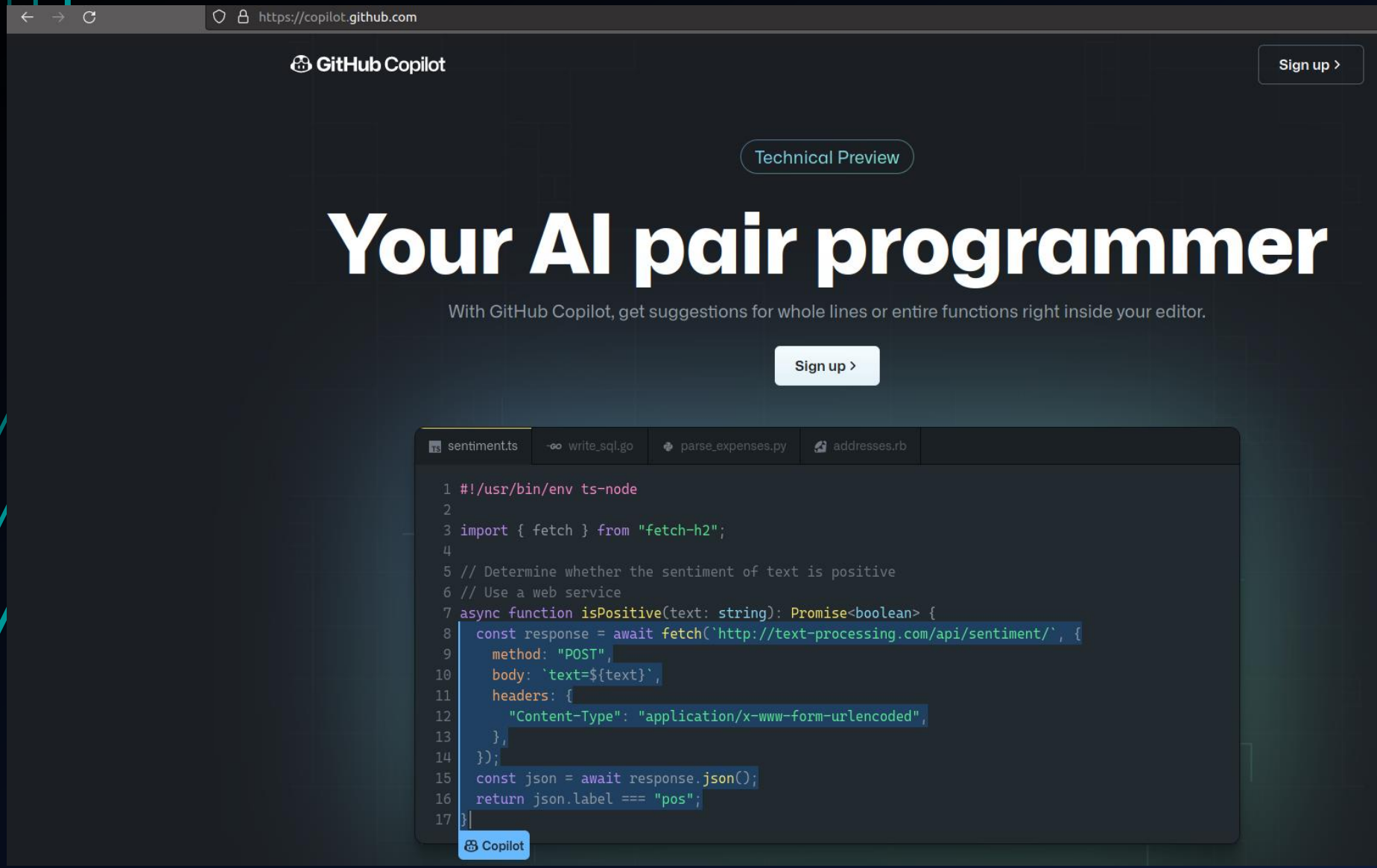
Java

Objective C

Ruby

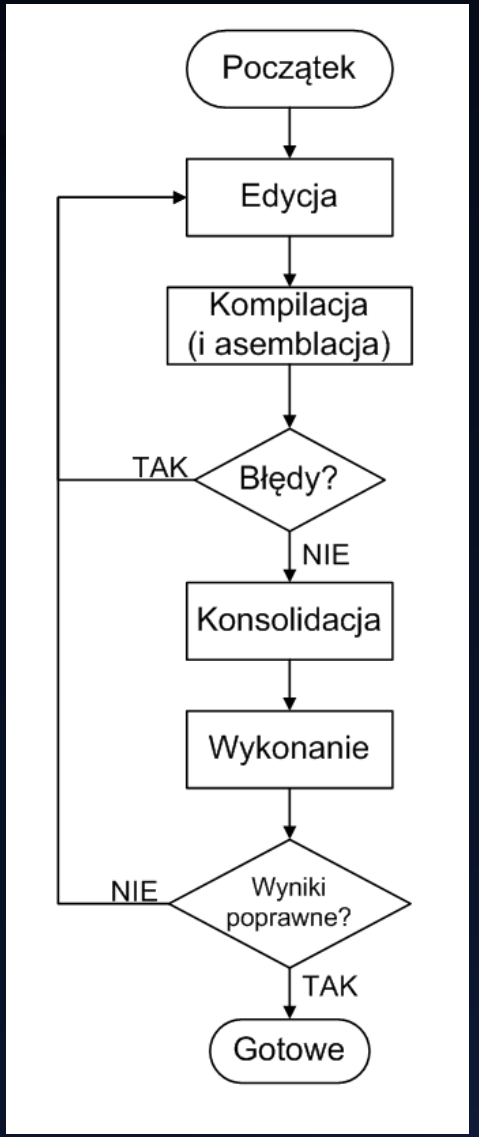
Typescript

Copilot


https://copilot.github.com
GitHub Copilot
Sign up >
Technical Preview
Your AI pair programmer
With GitHub Copilot, get suggestions for whole lines or entire functions right inside your editor.
Sign up >
sentiment.ts
write_sql.go
parse_expenses.py
addresses.rb
#!/usr/bin/env ts-node
import { fetch } from "fetch-h2";
// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
const response = await fetch(`http://text-processing.com/api/sentiment/`, {
method: "POST",
body: `text=${text}`,
headers: {
"Content-Type": "application/x-www-form-urlencoded",
},
});
const json = await response.json();
return json.label === "pos";
}
Copilot

Proces tworzenie

1. Tworzenie – pisanie kodu źródłowego
2. Kompilacja – sprawdzenie poprawności kodu
3. Konsolidacja – tworzenie pliku wynikowego
4. Wykonanie i testowanie programu

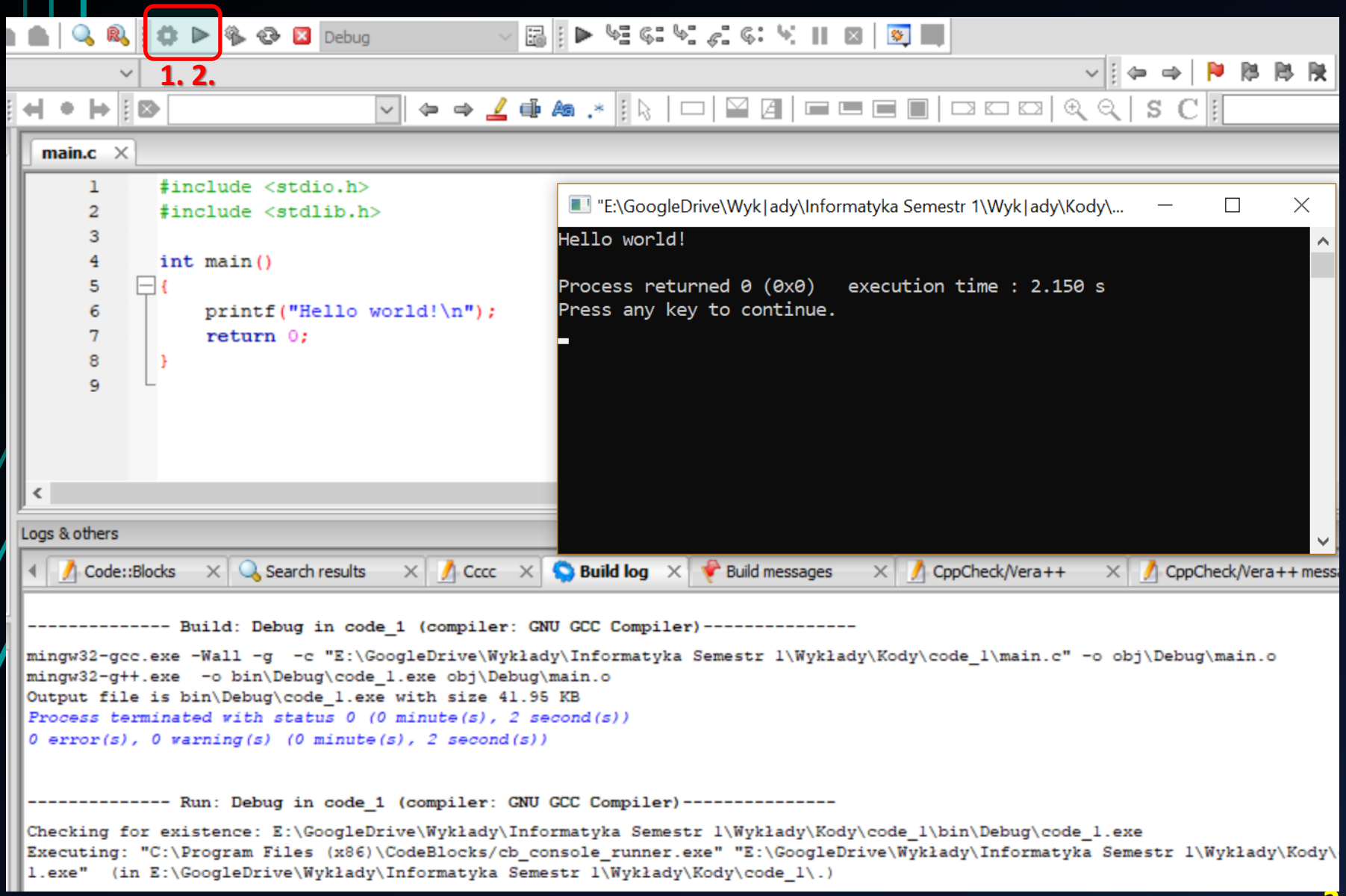


Struktura ogólna kodu w C

1. `#include <******>`- włączenia bibliotek
2. `#define ******` - stałe, makro instrukcje
3. Zmienne globalne
4. Prototypy funkcji
5. Funkcja główna - `main()`
6. Pozostałe funkcje

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

int main()
{
    printf("Hello world!\n");
    return 0;
}
```


1. 2.
Debug
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6 printf("Hello world!\n");
7 return 0;
8 }
9
"E:\GoogleDrive\Wykłady\Informatyka Semestr 1\Wykłady\Kody\...
Hello world!
Process returned 0 (0x0) execution time : 2.150 s
Press any key to continue.
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ mess
---------- Build: Debug in code_1 (compiler: GNU GCC Compiler)----------
mingw32-gcc.exe -Wall -g -c "E:\GoogleDrive\Wykłady\Informatyka Semestr 1\Wykłady\Kody\code_1\main.c" -o obj\Debug\main.o
mingw32-g++.exe -o bin\Debug\code_1.exe obj\Debug\main.o
Output file is bin\Debug\code_1.exe with size 41.95 KB
Process terminated with status 0 (0 minute(s), 2 second(s))
0 error(s), 0 warning(s) (0 minute(s), 2 second(s))
---------- Run: Debug in code_1 (compiler: GNU GCC Compiler)----------
Checking for existence: E:\GoogleDrive\Wykłady\Informatyka Semestr 1\Wykłady\Kody\code_1\bin\Debug\code_1.exe
Executing: "C:\Program Files (x86)\CodeBlocks/cb_console_runner.exe" "E:\GoogleDrive\Wykłady\Informatyka Semestr 1\Wykłady\Kody\
1.exe" (in E:\GoogleDrive\Wykłady\Informatyka Semestr 1\Wykłady\Kody\code_1\.)

Systemy liczbowe

- System dwójkowy – (system binarny, pozycyjny) liczby zapisywane za pomocą dwóch cyfr 0 i 1. Podstawą systemu jest liczba 2.

- System dziesiętny – system pozycyjny zapisywany za pomocą cyfr od 0 do 9, podstawą systemu jest cyfra 10

- System szesnastkowy – system hexadecymalny, o podstawie 16, do zapisu wykorzystuje się cyfry 0 do 9 i litery A, B, C, D, E i F

Binarny na dziesiętny

<table>
<tr><td rowspan="3">53</td><td colspan="6">110101</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>53:2=26 reszty 1
26:2=13 reszty 0
13:2=6 reszty 1
3:2=3 reszty 0
3:2=1 reszty 1
1:2=0 reszty 1</td><td colspan="6">$(1*2^5)+(1*2^4)+(0*2^3)+(1*2^2)+(0*2^1)+(1*2^0)=$
$32+16+4+1=$</td></tr>
<tr><td>110101</td><td colspan="6">53</td></tr>
</table>

Hexadecymalny na ...

<table>
<tr><td>0</td><td>0</td><td>0000</td><td colspan="4">12188</td><td colspan="3">101110101101</td></tr>
<tr><td>1</td><td>1</td><td>0001</td><td colspan="4" rowspan="5">12188:16=761 reszty 12 → C
761:16=47 reszty 9
47:16:2 reszty 15 → F
2:16=0 reszty 2</td><td>1011</td><td>1010</td><td>1101</td></tr>
<tr><td>2</td><td>2</td><td>0010</td><td>B</td><td>A</td><td>D</td></tr>
<tr><td>3</td><td>3</td><td>0011</td><td colspan="3">BAD</td></tr>
<tr><td>4</td><td>4</td><td>0100</td><td></td><td></td><td></td></tr>
<tr><td>5</td><td>5</td><td>0101</td><td></td><td></td><td></td></tr>
<tr><td>6</td><td>6</td><td>0110</td><td colspan="4">2F9C</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>7</td><td>7</td><td>0111</td><td>2</td><td>F</td><td>9</td><td>C</td><td colspan="3" rowspan="8">$(11*16^2)+(10*16^1)+(13*16^0)=$
$(11*256)+(10*16)+(13*1)=$
2816+160+13=
2989</td></tr>
<tr><td>8</td><td>8</td><td>1000</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>9</td><td>9</td><td>1001</td><td colspan="4" rowspan="6">$(2*16^3)+(F*16^2)+(9*16^1)+(C*16^0)=$
$(2*16^3)+(15*16^2)+(9*16^1)+(12*16^0)=$
$(2*4096)+(15*256)+(9*16)+(12*0)=$
8192+3840+144+12=
12188</td></tr>
<tr><td>10</td><td>A</td><td>1010</td></tr>
<tr><td>11</td><td>B</td><td>1011</td></tr>
<tr><td>12</td><td>C</td><td>1100</td></tr>
<tr><td>13</td><td>D</td><td>1101</td></tr>
<tr><td>14</td><td>E</td><td>1110</td></tr>
<tr><td>15</td><td>F</td><td>1111</td><td colspan="4">12188</td><td colspan="3">2989</td></tr>
</table>

Typy danych w C

Podstawowe typy danych (dla systemu 32bit)

Typ	Zawartość	w bitach	Zakres wartości
`char`	Znak	8	-128 ÷ 127
`int`	Liczba całkowita	32	-2 147 483 648 2 147 483 647
`float`	Liczba rzeczywista	32	$\pm 1.2 * 10^{-38} \div 3.4 * 10^{38}$
`double`	Liczba rzeczywista	64	$\pm 2.2 * 10^{-308} \div 1.8 * 10^{308}$
`bool`	Zmienna logiczna	8	true (1), false (0)

Typ zmiennej logicznej wymaga dodania biblioteki <stdbool.h>

Modyfikatory typów danych

1. `signed`
2. `unsigned`

Ustala czy dany typ danych obejmuje zakres liczb dodatnich i ujemnych czy tylko dodatnich. Domyślnie przyjmowany jest typ `signed` (±), łączy się ze zmienną typu `int` i `char`.

3. `short`

Zmniejsza o połowę rozmiar pamięci przydzielanej dla zmiennej, łączy się ze zmienną `int`.

4. `long`

Zwiększa rozmiar pamięci przydzielanej dla zmiennej, łączy się z `int` i `double`.

Zmodyfikowane typy danych

Lp	Typ	Wielkość w bitach	Zakres
1	`signed char`	8	-128 ÷ 127
2	`unsigned char`	8	0 ÷ 255
3	`short int`	16	-32 768 ÷ 32 767
4	`unsigned short int`	16	0 ÷ 65 535
5	`signed int`	32	-2 147 483 648 ÷ 2 147 483 647
6	`unsigned int`	32	0 ÷ 4 294 967 295
7	`signed long int`	32	-2 147 483 648 ÷ 2 147 483 647
8	`unsigned long int`	32	0 ÷ 4 294 967 295
9	`long double`	80	$3.4 * (10^{-4932}) \div 1.2 * (10^{4932})$

Deklaracja zmiennych

- Zmienne w języku C mogą być deklarowane jako globalne lub lokalne
- Zmienne globalne definiuje się poza obszarem funkcji, na początku kodu
- Dostęp do zmiennej globalnej jest z poziomu każdej funkcji, można jej używać i zmieniać zawartość

Deklaracja zmiennych

- Zmienne lokalne definiuje się wewnątrz funkcji (bloków kodu)
- Zmienna lokalna jest „widoczna” tylko wewnątrz funkcji (bloku) w której została zdefiniowana
- Zmienna zadeklarowana bez wartości początkowej w momencie definiowania przyjmuje wartość „losową”

Deklaracja zmiennych

- Deklaracja zmiennej

```
typ_zmiennej nazwa zmiennej;
```

```
int x;
```

```
double y,z;
```

- Deklaracja zmiennej z przypisaniem wartości początkowej

```
int x,y=10;
```

```
char tekst[]=”Alfa”, T=‘W’;
```

- Deklaracja stałej

```
conts float PI=3.1415;
```

```
#DEFINE PI 3.1415
```

Definicja typu

- W języku C można zdefiniować własny typ danych
- Określa się go definiując nazwę dla określonego typu
- Definicję typu rozpoczyna się słowem kluczowym `typedef`
- Po nim określa się typ danych za pomocą dostępnych typów i modyfikatorów
- Ostatnim elementem jest nazwa wskazanego typu

```
typedef short unsigned int sint;
sint x;
```

Słowa kluczowe

Nazwy użyte dla słów kluczowych są zastrzeżone i nie mogą być stosowane do nazw zmiennych.

auto	double	inline	static
break	else	int	struct
case	enum	long	switch
char	extern	register	typedef
complex	float	restrict	union
cont	for	return	unsigned
continue	goto	short	void
default	if	signed	volatile
do	imaginary	sizeof	while

Operatory w C

1. Operatory arytmetyczne
   - `+, -, *, /` - podstawowe działania
   - `%` - dzielenie z resztą
2. Operatory przypisania
   - `=` - `x=2;`
   - `+=` - `x+=2;` - `x=x+2;`
   - `-=` - `x-=2;` - `x=x-2;`
   - `*=` - `x*=2;` - `x=x*2;`
   - `/=` - `x/=2;` - `x=x/2;`
   - `%=` - `x%=2;` - `x=x%2;`

Operatory w C

Opera

- **x**
  w
- **+**
  w
- **x**
  w
- **–**
  w

yliczeniu

iczeniem

yliczeniu

iczeniem

```
int
```

a) x=

b) x=

c) x+

```c
#include <stdio.h>

int main()
{
    int x=1;
    int y=1;
    printf("\t\tx=%i\ty=%i\n",x,y);
    y=x++;
    printf("y=x++;\t\tx=%i\ty=%i\n",x,y);
    y=++x;
    printf("y=++x;\t\tx=%i\ty=%i\n",x,y);
}

```

```
C:\WINDOWS\system32\cmd.exe
                        x=1     y=1
y=x++;                  x=2     y=1
y=++x;                  x=3     y=3

Press any key to continue . . .
```

Operatory w C

1. Operatory relacji
   - `==` - równe
   - `!=` - różne
   - `<`, `>` - mniejsze, większe
   - `<=`, `>=` - mniejsze lub równe, ...
2. Operatory logiczne
   - `&&` - koniunkcja (**AND**)
   - `||` - alternatywa (**OR**)
   - `!` - negacja (**NOT**)

Wyjścia-Wyjścia

- Program komputerowy komunikuje się z użytkownikiem za pomocą standardowych portów wejścia-wyjścia
- Wejścia:
  - **Klawiatura**
  - Dysk (plik)
  - Myszka (inne urządzenie wskazujące)
- Wyjścia:
  - **Ekran monitora**
  - Dysk (plik)
  - Drukarka

Operacje wyjścia - printf

- Standardowa procedura wyjściowa wysyłająca sformatowane dane w standardowe wyjście
- Funkcja wymaga biblioteki stdio.h

```
include <stdio.h>
printf("tekst", arg_1, arg_2,…);
```

- tekst – stała łańcuchowa umieszczona w " " zawierająca:
  - Znaki – wysyłane na ekran
  - Kody formatujące – sterujące wyświetlaniem kolejnych argumentów
  - Znaki kodowe sterujące przepływem wyświetlanego tekstu
- Argumenty – zmienne i stałe przekazywane do wyświetlanego tekstu

Operacje wyjścia - printf

- Tekst wysyłany na port wyjściowy (ekran) jest łańcuchem tekstowym
- Do funkcji `printf()` może być wprowadzony jako tekst w ” ” lub jako tablica znakowy typu char
- Umieszczanie zawartości zmiennych w wysyłanym tekście realizowane jest przez kody formatujące

Operacje wyjścia - printf

- Kody formatujące:
- %c- znak z zmiennej (char)
- %s- ciąg znakowy (char[])
- %i- liczba całkowita (int)
- %hi - liczba całkowita (short int)
- %hu - liczba całkowita (unsigned short int)
- %li - liczba całkowita (long, unsigned long)
- %u- dodatnia liczba całkowita (unsigned int)
- %f- liczba rzeczywista (float)
- %lf – liczba rzeczywista podwójnej precyzji (double)
- %Ld - rozszerzona liczba rzeczywista (long double)
- %e – liczba zmiennoprzecinkowa x.xxxxxxe±xxx
- %g – skrócona forma liczby rzeczywistej

Operacje wyjścia - printf

Znaki specjalne

- Niektóre znaki nie mogą być wyświetlone w sposób standardowy
  - '`\a`' – alarm (dźwięk)
  - '`\b`' – cofa kursor o jeden znak
  - '`\r`' – kursor na początek wiersza
  - '`\n`' – przejście do nowego wiersza
  - '`\t`' – tabulacja
- Znaki specjalne umożliwiają sterowanie położeniem kursora w konsoli
  - '`\"`', '`\'`', '`\\`', '`\?`' – znaki specjalne

Przykład 1

```c
#include <stdio.h>
short int x=10;
char litera='W';
int main()
{
    printf("Znak: %c%c%c\b\b \n",litera,litera,litera);
    printf("Tekst:\r%s\n","Tekst do wyslania na ekran");
    printf("Liczba: x=\t%hi \n",x);
    return 0;
}
```

```
Znak: W W
Tekst do wyslania na ekran
Liczba: x=      10

Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.
```

Operacje wyjścia - printf

Sposób wyświetlania liczby rzeczywistej można formatować za pomocą odpowiednio ustawionych parametrów

```
%[flagi][szerokosc][.precyzja][modyfikator]typ
```

flaga	znaczenie
-	wyrównanie do lewej (do prawej jest domyślne),
+	wymusza znak + przed liczbami dodatnimi,
spacja	wstawia spację przed liczbami dodatnimi,
0	jeżeli liczba jest, krótsza niż podana długość to uzupełnia zerami (domyślnie spacjami)

Operacje wyjścia - printf

```
%[flagi][szerokość][.precyzja][modyfikator]typ
```

- **Szerokość** – minimalna liczba znaków zastosowana do wyświetlenia liczby
- Jeżeli liczba jest krótsza od zadeklarowanej, wyświetlana postać uzupełniona jest odpowiednią liczbą spacji
- **Precyzja** określa liczbę znaków po przecinku
- **Modyfikator** umożliwia zmianę zakresu liczby (h - short, l-long, L – long double)

Operacje wyjścia - printf

Jeżeli liczbę określającą szerokość i precyzję zastąpi się '*' to konieczne jest podanie na liście parametrów przed formatowanym argumentem wartości liczbowych (zmiennych) które określą te parametry.

```
int a=5,b=2;
float x=345.12456;
printf(„Liczba x=%f\n”,x);
printf(„Liczba x=%+.2f\n”,x);
printf(„Liczba x=%6.2f\n”,x);
printf(„Liczba x=%*.*f\n”,4,2,x/y);
printf(„Liczba x=%*.*f\n”,a,b,x);
```

Kodowanie znaków ASCII

```
1   #include <stdio.h
2
3   int main()
4   {
5       printf("Znak $
6   }
```

C:\WINDOWS\system32\cmd.exe

```
Znak $: $ lub $
Press any key t
```

znak: ☺	1	0x1	znak: ☻	2	0x2		
znak: ♥	3	0x3	znak: ♦	4	0x4		
znak: ♣	5	0x5	znak: ♠	6	0x6		
znak:	7	0x7	znak:	8	0x8		
znak: ♂	11	0xb	znak: ♫	14	0xe		
znak: ☼	15	0xf	znak: ►	16	0x10		
znak: ◄	17	0x11	znak: ↕	18	0x12		
znak: ‼	19	0x13	znak: ¶	20	0x14		
znak: §	21	0x15	znak: ▬	22	0x16		
znak: ↨	23	0x17	znak: ↑	24	0x18		
znak: ↓	25	0x19	znak: →	26	0x1a		
znak: ←	27	0x1b	znak: ∟	28	0x1c		
znak: ↔	29	0x1d	znak: ▲	30	0x1e		
znak: ▼	31	0x1f	znak:	32	0x20		
znak: !	33	0x21	znak: "	34	0x22		
znak: #	35	0x23	znak: $	36	0x24		
znak: %	37	0x25	znak: &	38	0x26		
znak: '	39	0x27	znak: (	40	0x28		
znak:)	41	0x29	znak: *	42	0x2a		

Operacja wyjścia: `puts()`, `putchar()`

- Język C dostarcza narzędzi umożliwiających wysyłanie tekstu i znaków na ekran alternatywnych do printf
- `puts()` – wysyła na ekran ciąg tekstowy zakończony domyślne znakiem nowej linii
- `putchar()` – wysyła na ekran pojedynczy znak

Operacje wejścia - scanf

- Standardową procedurą wejściową, pobierającą dane od użytkownika jest `scanf()`

```
scanf("kody zmiennych", &zm1,&zm2 …);
```

- Lista kodowych znaków formatujących definiująca typy oczekiwanych do wprowadzenia zmiennych

- Dla każdej zmiennej pobieranej przez `scanf` musi zostać podany kod formatujący, podobnie jak w procedurze wyjściowej `printf`

Operacje wejścia - scanf

- **&zm*** - lista zmiennych pobierających wartości z procedury **scanf**

- Znak **&** jest znakiem wymaganym przed nazwą zmiennej, wskaźnika do zmiennej

```
int x;
float y;
char d;
scanf(”%i”, &x);
scanf(”%i %f”,&x,&y);
scanf(”%i %c %f”,&x,&d,&y);
```

Przykład 2

```c
#include <stdio.h>
int a, b, c;

int main(void)
{
   printf("Program liczy sume trzech liczb calkowitych: a+b+c. ");
   printf("Podaj liczby:\n");
   printf("Podaj wartosc liczby a=");
   scanf("%i",&a);
   printf("Podaj wartosc liczby b=");
   scanf("%i",&b);
   printf("Podaj wartosc liczby c=");
   scanf("%i",&c);
   printf("Suma liczb %i+%i+%i=%i\n",a,b,c,a+b+c);
  return 0;
}
```

```
Program liczy sume trzech liczb calkowitych: a+b+c. Podaj liczby:
Podaj wartosc liczby a=10
Podaj wartosc liczby b=23
Podaj wartosc liczby c=9
Suma liczb 10+23+9=42

Process returned 0 (0x0)   execution time : 23.050 s
Press any key to continue.
```

Operacje wejścia: `gets()`, `getchar()`

- Język C dostarcza alternatywne funkcje do `scanf()` umożliwiające pobranie danych od użytkownika
- Funkcja `getchar()` umożliwia pobranie pojedynczego znaku z wejścia i przypisanie go do zmiennej typu `int`
- Funkcja `getchar()` jest wykorzystywany jako zatrzymanie działania programu w oczekiwaniu na naciśnięcie dowolnego klawisza
- Funkcja `gets()` pobiera od użytkownika ciąg znaków i przypisuje ją do tablicy znakowej

Arytmetyka w C

- Istotne jest stosowanie właściwych typów danych do stosowanych w programowanej aplikacji kodzie

- Język C dostarcza dwóch podstawowych typów danych liczbowych:
  - Liczby całkowity – `int`
  - Liczby rzeczywiste – `float`

Arytmetyka w C

Typ całkowity `int`	Typ rzeczywisty `float`

`int a=25;`
`int b=2;`
`int c;`
`c=a/b*b;`
	`float a=25;`
	`float b=2;`
	`float c;`
	`c=a/b*d;`

Wynik działania: `25/2*2=24`	Wynik działania `25/2*2=25`

Konwersja między typami

- Język C dostarcza narzędzi pozwalających na dokonanie konwersji pomiędzy typami `int` a `float`
- Działania takie są czasem konieczne przy wykonywaniu działań arytmetycznych zgodnie z oczekiwaniem programisty lub przy stosowaniu niektórych funkcji i procedur wymagających sprecyzowanego typu danych na wejściu
- Konwersja realizowana jest poprzez umieszczenie przed wybraną zmienną lub działaniem wybranego typu danych w nawiasie `(int)` lub `(float)`

Konwersja pomiędzy typami

Typ całkowity `int`	Typ rzeczywisty `float`

`int a=25;`
`int b=2;`
`int c;`
`c=(float)a/b*b;`
	`float a=25;`
	`float b=2;`
	`float c;`
	`c=(int)(a/b)*d;`

Wynik działania: `25/2*2=25`	Wynik działania `25/2*2=24`

Pokazaną konwersję typów można z powodzeniem stosować wewnątrz funkcji wyjścia `printf()`

Przykład 4

Przykład obliczeń na liczbach rzeczywistych i całkowitych

```
12.340000 przypisane zmiennej int 12
-10 przypisane do zmiennej float -10.000000
-10 podzielone przez 3 daje int -3
-10 dzielone przez 3.0 daje float -3.333333
(float) -10 dzielone przez 3 daje float -3.333333
```

```c
#include <stdio.h>
int main()
{
    float x,y=12.34;
    int a,b=-10;
    a=y;
    printf("%f przypisane zmiennej int %i\n",y,a);
    x=b;
    printf("%i przypisane do zmiennej float %f\n",b,x);
    a=b/3;
    printf("%i podzielone przez 3 daje int %i\n",b,a);
    x=b/3.0;
    printf("%i dzielone przez 3.0 daje float %f\n",b,x);
    x=(float)b/3;
    printf("(float) %i dzielone przez 3 daje float %f\n",b,x);
    return 0;
}
```

printf – liczby rzeczywiste

- Procedura `printf()` umożliwia sterowanie formą wyświetlania liczb rzeczywistych

- Pomiędzy znakiem % a kodem formatujący można zamieści definicję wyświetlania liczby rzeczywistej

- Definiuje się liczbę znaków oraz liczbę znaków po przecieku w postaci `%X.Xf`

```
printf("Wynik to x=%6.3f\n",x);
```

```
Wynik to x=123.456
```

Przykład 4

- Napisz program przeliczający temperaturę podaną w stopniach Fahrenheita (F) na skalę Celcjusza (C)

- Przeliczanie realizowane jest zgodnie z równaniem: C=(F-32)/1.8

Przykład 4

```
Program przelicza podana w stopniach Fahrenheita
temperature na stopnie Celcjusza.
Podaj temperature F=197.35
Temperaturze F=197.35 odpowiada C=91.86
```

```c
#include <stdio.h>
int main()
{
    float f,c;
    printf("Program przelicza podana w stopniach Fahrenheita\n");
    printf("temperature na stopnie Celcjusza.\n");
    printf("Podaj temperature F=");
    scanf("%f",&f);
    c=(f-32)/1.8;
    printf("Temperaturze F=%4.2f odpowiada C=%4.2f\n",f,c);
    return 0;
}
```

Komentarze

- Podczas pisania kodu programu warto w kluczowych miejscach uzupełniać kod o komentarz
- Komentarz pozwala samemu zorientować się po czasie w kodzie
- Dzięki komentarzom inne osoby korzystające z kodu będą mogły łatwiej zorientować się w ewentualnych zawiłościach kodu

Komentarze

- Komentarz umieszczony w jednej linii kodu umieszcza się na końcu linii po //

```
int    pi=3.1415;    //deklaracja liczby pi
```

- Komentarz obejmujący kilka linii kodu umieszcza się pomiędzy znacznikami /* */

```
int    pi=3.1415;    /*deklaracja liczby pi*/
```

Polskie znaki w konsoli

- Standardowo polskie znaki dialektyczne nie są wyświetlane prawidłowo
- Rozwiązaniem może być kodowanie znaków na polski system dialektyczny
- Biblioteka:

```
#include <locale.h>
```

- Ustawienie

```
setlocale(LC_CTYPE,"Polish");
```

Przykład

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