1. Informacje podstawowe

1.1. Funkcja Main

Każdy program w języku C zawiera funkcję o nazwie **main**. Funkcja ta (i jej ciało) jest zapisane zgodnie ze składnią języka C. Funkcja main zawiera kod „wykonywany" programu. Proszę spojrzeć na poniższy program.

```
#include <iostream>

int main() {
  std::cout << "Hello World!\n";
}
```

Cały efekt programu widoczny dla użytkownika – napis Hello World – jest konsekwencją wykonania instrukcji w nawiasach wąsiastych (czyli w „ciele") funkcji **main**.

1.2. Komentarze

Komentarze są to elementy kodu źródłowego **ignorowane** przez kompilator. To informacje dodatkowe, które programista pozostawia dla innych osób czytających kod. Dobrze dobrane komentarze ułatwiają odczytanie kodu i odnalezienie się w bardziej skomplikowanych programach.

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
int main() {
  std::cout << "Hello World!\n"; //to jest komentarz na jedną linijkę
  /* ten komentarz może zajmować dowolną ilość linijek.
  Ważne jest zamknięcie go znakami */
}
```

1.3. Podstawowe typy zmiennych

Podstawowe typy zmiennych:

- int - zmienne całkowite
- double - zmienne rzeczywiste podwójnej precyzji
- char - zmienne znakowe
- bool - zmienne logiczne

Zmienne „definiujemy" w programie, co oznacza, że informujemy kompilator jak nazywa się zmienna i jakiego będzie typu. **Np.: int liczba.**

Oto kilka definicji zmiennych:

```
int a;              // definiujemy zmienną całkowitą a
int b,c,d;          // definiujemy trzy zmienne całkowite b,c,d
double x1,x2;       // definiujemy dwie zmienne rzeczywiste x1 i x2
char znak;          // definiujemy zmienną znakową znak
bool p,w;           // definiujemy dwie zmienne logiczne p i w
```

Dokładniejszy wykaz zmiennych:

Typy całkowite

Nazwa	Wielkość (bajty)	Zakres
short	2	$-2^{15} \div 2^{15} - 1$, czyli przedział [-32768, 32767]
int	4	$-2^{31} \div 2^{31} - 1$, czyli przedział [-2147483648, 2147483647]
long	4	$-2^{31} \div 2^{31} - 1$, czyli przedział [-2147483648, 2147483647]
long long	8	$-2^{63} \div 2^{63} - 1$, czyli przedział [-9223372036854775808, 9223372036854775807]
unsigned short	2	$0 \div 2^{16} - 1$, czyli przedział [0, 65535]
unsigned int	4	$0 \div 2^{32} - 1$, czyli przedział [0, 4294967295]
unsigned long	4	$0 \div 2^{32} - 1$, czyli przedział [0, 4294967295]
unsigned long long	8	$0 \div 2^{64} - 1$, czyli przedział [0, 18446744073709551615]

Źródło: http://www.algorytm.edu.pl/wstp-do-c/typy-zmiennych.html

Typ rzeczywisty - przechowuje liczby zmiennoprzecinkowe (ułamki). Wyróżniamy następujące typy:

Nazwa	Wielkość (bajty)	Zakres
float	4	pojedyncza precyzja - dokładność 6 - 7 cyfr po przecinku
double	8	podwójna precyzja - dokładność 15 - 16 cyfr po przecinku
long double	12	liczby z ogromną dokładnością - 19 - 20 cyfr po przecinku

Źródło: http://www.algorytm.edu.pl/wstp-do-c/typy-zmiennych.html

Typ znakowy - przechowuje znaki, które są kodowane kodem ASCII. Tzn. znak w pamięci nie może być przechowany jako znak, tylko jako pewna liczba. Dlatego każdy znak ma swój odpowiednik liczbowy z zakresu [0, 255], który nazywamy kodem ASCII.

Nazwa	Wielkość (bajty)	Zakres
char	1	-128 ÷ 127
unsigned char	1	0 ÷ 255

Źródło: http://www.algorytm.edu.pl/wstp-do-c/typy-zmiennych.html

Typ logiczny - przechowuje jedną z dwóch wartości - true (prawda) oznaczone 1 albo false (fałsz) oznaczone 0.

Nazwa	Wielkość (bajty)	Zakres
bool	1	true (1) false (0)

Źródło: http://www.algorytm.edu.pl/wstp-do-c/typy-zmiennych.html

W C++ można również tworzyć **łańcuchy znaków (np. napisy)**. Aby to zrobić należy poinformować kompilator o przestrzeni nazw w jakiej zmienna się znajduje: **using namespace std;**

```
string napis = "To jest napis!"
```

Przed użyciem zmiennej należy ją **zadeklarować** - „poinformować" kompilator jak będzie się nazywać i jakiego będzie typu. Np.: **int liczba.**
Inicjalizacja zmiennej (inicjowanie) polega na przypisaniu jej wartości **w momencie** deklaracji (utworzenia). Przykład inicjalizacji: **int liczba=44;**

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
using namespace std;

int main()
{
   int zmiennaPierwsza; //definicja
   int zmiennaDruga = - 2; // definicja i inicjalizacja
   unsigned int zmiennaTylkoDodatnia; //definicja
   float ulamek;  //definicja
   ulamek = 12.45; //przypisanie wartości
   char a,b; //definicja dwóch zmiennych  chat
   a = 'a';  //przypisanie wartości
   b = 'b';  //przypisanie wartości
   string napis = "To jest napis!"; //definicja i inicjalizacja

   cout << "Zmienne o nazwie:" << endl;
   cout << "zmiennaPierwsza = " << zmiennaPierwsza << endl;
   cout << "zmiennaDruga = " << zmiennaDruga << endl;
   cout << "zmiennaTylkoDodatnia = " << zmiennaTylkoDodatnia << endl;
   cout << "ulamek = " << ulamek << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   cout << "napis = " << napis << endl;
   cout << "Można też wypisać tak. Zmienna ulamek = "<<ulamek<<", a zmienna zmiennaDruga =
"<<zmiennaDruga<<"." << endl;
}
```

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
using namespace std;

int main()
{
  string napis1;
  napis1 = "pierwszy napis";
  string napis2("drugi napis"); //inicjalizowanie łańcucha znaków w miejscu jego tworzenia – inna metoda
  string napis3 = "trzeci napis"; //definicja + inicjalizacja

  cout << napis1 << endl
    << napis2 << endl
    << napis3 << endl;

  string napis4(10,'X');
  cout << napis4;
}
```

1.4. Znaki specjalne

W języku C++ istnieją również znaki specjalne. Znaki te poprzedzone są ****. W powyższym programie przechodziliśmy do nowej linii za pomocą modyfikatora **endl**. Możemy to zrobić stosując również znak **\n.**

Lista znaków specjalinych:

'\n' - nowa linia
'\t' - tabulacja pozioma (czyli "normalny" tabulator)
'\v' - tabulacja pionowa
'\b' - cofnięcie (skasowanie ostatniego znaku)
'\r' - powrót karetki (przesunięcie do początku wiersza)
'\f' - nowa strona
'\a' - sygnał dźwiękowy
'\\' - ukośnik (backslash), czyli znak \
'\?' - znak zapytania
'\'' - pojedynczy cudzysłów
'\"' - podwójny cudzysłów

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
using namespace std;

int main()
{
   int zmiennaPierwsza = 1;
   int zmiennaDruga = 2;
   char a,b;
   a = 'x';
   b = 'y';
   string napis = "To jest napis!";

   cout << "Zmienne o nazwie zmiennaPierwsza = \n";
   cout << zmiennaPierwsza <<"\n";
   cout << "\tZmienne o nazwie zmiennaDruga = "<< zmiennaDruga <<"\n";
   cout << "\vZmienne o nazwie zmiennaPierwsza = ";
   cout << zmiennaPierwsza << "\n";
   cout << "Zmienne o nazwie a i b wypisane pojedynczo = ";
   cout <<a<<' '<<b << endl;
   cout << "Zmienne o nazwie a i b wypisane razem = \n";
   cout <<a <<b << endl;
   cout << "Zmienne o nazwie napis = ";
   cout <<napis;
}
```

1.4. Instrukcje – podstawy (wejście i wyjście)

Instrukcja to fragment programu powodujący akcję komputera w trakcie jej wykonywania. Przykładem instrukcji są instrukcje **wejścia i wyjścia**.

Wyjście (wyświetlamy coś na ekranie / konsoli):

cout <<

Wejście (pobieramy dane podane przez użytkownika):

cin <<

Aby użyć instrukcji wejścia należy dołączyć plik nagłówkowy `#include <cstring>`

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   string imie;

   cout << "Napisz, jak się nazywasz: ";
   cin >> imie;
   cout << "Cześć, " << imie << "! ";
}
```

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
  int wiek;
  string imie;
  string nazwisko;
  char plec;

  cout << "Podaj swoje dane\n";
  cout << "Imie: ";
  cin >> imie; //nastepuje przypisanie podanej wartosci zmiennej imie
  cout << "Nazwisko: ";
  cin >> nazwisko;
  cout << "Wiek: ";
```

```
 cin >> wiek;
 cout << "Kobieta = K, mezczyzna = M: "; //sposób na poczet przykładu. Później poznamy lepszy
 cin >> plec;

 cout << "Cześć, " << imie <<" "<< nazwisko <<"! "<<"Masz "<< wiek <<" lat"<< ", a twoja plec to: "<< plec<<
endl;
}
```

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
int i;
double d;

cout << "Liczba i: ";
cin >> i; // wczytuje liczbę typu int do i
cout << "Liczba d: ";
cin >> d; // wczytuje liczbę typu double do d
cout << "Podane liczby: i = "<<i <<", d = "<<d;
}
```

1.5. Operatory arytmetyczne

Proste zadania matematyczne, jakie wykonuje program komputerowy to **dodawanie, odejmowanie, mnożenie i dzielenie**. W C++ i w innych językach mamy do dyspozycji operatory arytmetyczne:

- "+"- dodawanie
- "-" - odejmowanie
- "*" - mnożenie
- "/" - dzielenie całkowite lub rzeczywiste. (jeśli argumentami są liczby całkowite, operator będzie wykonywał dzielenie całkowite, natomiast dla liczb rzeczywistych operator wykona dzielenie rzeczywiste. (przykład poniżej)
- "%" - reszta z dzielenia dwóch liczb całkowitych
- ++x pre-inkremetacja (zmiennej x)
- x++ post-inkrementacja (zmiennej x)
- --x pre-dekrementacja (zmiennej x)
- x-- post-dekrementacja (zmiennej x)

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
int x = 0;
int y = 0;

cout << "Podaj liczbę x: ";
cin >> x;
cout << "Podaj liczbę y (różną od 0): ";
cin >> y;

cout << "Wynik dodawania: " << x + y << endl; //dodawania
cout << "Wynik odejmowania: " << x - y << endl; //odejmowanie
cout << "Wynik dzielenia: " << x / y << endl; //dzielenie
cout << "Wynik dzielenia modulo: " << x % y << endl; //dzielenie modulo
cout << "Wynik mnożenia: " << x * y << endl; //mnożenie
}
```

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
int x = 10;
/*pre-inkrementacja (preinkrementacja powoduje zwiększenie wartości zmiennej o
jeden, ale jej zwiększenie następuje PRZED wykorzystaniem zmiennej)*/

int y = ++x; // NAJPIERW zostanie zwiększona wartość zmiennej x o jeden, a następnie
zostanie użyta wartość 11
cout << "x: " << x << endl;
cout << "y: " << y << endl;
}
```

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
int x = 10;
/*postinkrementacja powoduje zwiększenie wartości zmiennej o jeden, ale jej
zwiększenie następuje po wykorzystaniu zmiennej*/
int y = x++; //Najpierw zostanie użyta wartość 10, a następnie zmienna x zostanie
zwiększona o jeden

cout << "x: " << x << endl;
cout << "y: " << y << endl;
}
```

DEKREMENTACJE (POST I PRE) DZIAŁAJĄ NA PODOBNEJ ZASADZIE ZE ZMNIEJSZENIEM WARTOŚCI.
ZAPIS TO NP.: --x, lub x--

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
int zmienna;

cout << "Podaj liczbe: ";
cin >> zmienna;

cout << "Obliczam: 3 + zmienna * 4 = " << 3 + zmienna * 4 << endl; //Uwaga na kolejność działań!
cout << "Obliczam: (3 + zmienna) * 4 = " <<(3 + zmienna) * 4 << endl;
}
```

1.6. Operatory porównania

Do budowania wyrażeń logicznych używa się również operatorów porównania:

- operator > oznacza "większe"
- operator >= oznacza "większe lub równe"
- operator < oznacza "mniejsze"
- operator <= oznacza "mniejsze lub równe"
- operator == oznacza "równe"
- operator != oznacza "różne"

Operatory te zwracają **prawdę (1) lub fałsz (0).**

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
  int x = 5;
  int y = 2;

  cout << "x= " << x << ", y= " << y << endl;
  cout << endl;

  cout << "(x większe niż y) x > y = " << (x > y) << endl;
  cout << "(x większe lub równe y) x >= y = " << (x >= y) << endl;
  cout << "(x mniejsze niż y) x < y = " << (x < y) << endl;
  cout << "(x mniejsze lub równe y) x <= y = " << (x <= y) << endl;
  cout << "(x różne niż y) x != y = " << (x != y) << endl;
  cout << "(x równe y) x == y = " << (x == y) << endl;

  //wynik zapisany w zmiennych
  bool wynik1 = x < y;
  bool wynik2 = x > y;
  cout << "Wynik x < y: " <<wynik1 <<endl;
  cout << "Wynik x > y: " <<wynik2 <<endl;
}
```

1.7. Operatory logiczne

- operator && oznacza koniunkcję, w C++ można użyć słowa kluczowego and
- operator || oznacza alternatywę, w C++ można użyć słowa kluczowego or
- operator ! oznacza negację, w C++ można użyć słowa kluczowego not
- operator ^ oznacza alternatywę wykluczającą, w C++ można użyć słowa kluczowego xor

WARUNEK 1	WARUNEK 2	OR (||)	AND (&&)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

WARUNEK	NOT (!)
0	1
1	0

Źródło: https://miroslawzelent.pl/kurs-c++/operatory-w-c++/

Uwaga: w C++ obowiązuje konwencja, iż każda wartość liczbowa różna od zera odpowiada wartości true, a wartością liczbową odpowiadającą false jest tylko zero.

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
using namespace std;
int main()
{
  bool p=1;
  bool q=0;

  cout << "p = (true) " << p << ", q = (false) " << q << endl;
  cout << endl;
  cout << "p && q = " << (p && q) << endl;
  cout << "p || q = " << (p || q) << endl;
  cout << "p ^ q = " << (p ^ q) << endl; //jeśli dane bity są różne to jeden jeżeli są takie same to 0
   cout << "!q = " << !q << endl;
  cout << endl;
  cout << "p and q = " << (p and q) << endl;
  cout << "p or q = " << (p or q) << endl;
  cout << "p xor q = " << (p xor q) << endl; //jeśli dane bity są różne to jeden jeżeli są takie same to 0
  cout << "not q = " <<not q<< endl;
}
```

1.7. Instrukcja warunkowa if i else if

Aby program mógł **„podejmować decyzje”** istnieją tak zwane instrukcje warunkowe. Pozwalają one na wykonanie instrukcji w przypadku gdy jakiś warunek jest prawdą i innych gdy jest fałszem.

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
bool warunek = true;

if(warunek)
 { //początek bloku należącego do if
   cout <<"Jestem w środku IF!"<<endl;
   //instrukcje zostaną wykonane gdy warunek (warunki) jest prawdziwy
   //w przeciwnym wypadku ta część kodu zostanie pominięta
 } //koniec bloku

cout <<"Jestem poza blokiem instrukcji IF!";

}
```

Instrukcje warunkowe można uzupełnić o warunek gdzie instrukcja wypełni się w przypadku fałszu.

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
int wiek;
cout<<"Podaj swój wiek:  ";
cin >> wiek;

if(wiek >= 18)
 {
   cout <<"Masz co najmniej 18 lat!"<<endl;
 } else { //jeśli warunek jest fałszywy, wykona się JEDYNIE instrukcja poniżej
   cout <<"Nie jesteś pełnoletni!" <<endl;
 }
}
```

Instrukcje warunkowe można również zagnieżdżać. Warunków IF może być również kilka. Instrukcja wykona się, jeśli każdy jest prawdziwy.

Proszę przetestować poniższy kod w repl.it lub VS Code:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
 cout<<"Podaj liczbę posiadanych dzieci: ";
 int liczba;
 cin >> liczba;

 if (liczba == 1)
  cout<<"Masz jedno dziecko.";
        else if (liczba > 1) //zagnieżdżona instrukcja if
         cout<<"Masz więcej niż jedno dziecko!";
              else
              cout<<"Nie masz dzieci.";
}
```