

## C++ Przypomnienie

```
1      #include <iostream>
2
3      using namespace std;
4
5      int main()
6      {
7          cout << "Hello world!" << endl;
8          return 0;
9      }
10
```

Tak wygląda standardowy program “Hello World”.

**Linia 1:** `#include <iostream>` odpowiada za importowanie biblioteki. `<iostream>` odpowiada za import standardowej biblioteki *input/output stream*, która pozwala nam używać takich poleceń jak `cout`, `cin` oraz `endl`.

**Linia 2:** `using namespace std;` pozwala nam na ominięcie pisania `std::` przy każdym użyciu `cout`, `cin` lub `endl`.

**Linia 5:** jest to definicja podstawowej funkcji `main`, która obowiązkowa w każdym programie w C++. Funkcja `main` jest wykonywana jako pierwsza. **`int`** na początku definicji funkcji oznacza to, że funkcja musi zwrócić wartość typu **`int`**. Inne wartości, jakie może zwracać funkcja to: `float`, `double`, oraz `void`.

- **`void`** oznacza, że funkcja nie zwraca wartości – nie potrzebuje **`return`** na końcu.
- Zawartość funkcji znajduje się w klamrach `{}`.

Dla prostych programów, które nie posiadają innych funkcji, wszystko piszemy w funkcji **`main`**.

## Funkcja warunkowa if

```

5  int main()
6  {
7      int a = 4;
8
9      if(a==4) {
10         cout << "Zmienna a jest równa 4" << endl;
11     } else if(a==5) {
12         cout << "Zmienna a jest równa 5" << endl;
13     } else {
14         cout << "Zmienna a nie jest równa ani 4 ani 5" << endl;
15     }
16
17     return 0;
18 }
19

```

Funkcja warunkowa if, pozwala nam sprawdzić to, czy warunek jest prawdą.

W tym przykładzie najpierw sprawdzane jest czy zmienna **a** jest równa 4. Jeżeli jest równa 4 to odpowiedni tekst pojawi się w konsoli.

Jeżeli zmienna nie jest równa 4 to funkcja przechodzi do następnego warunku **else if**, który sprawdza czy zmienna **a** jest równa 5. Analogicznie jak przy sprawdzaniu czy zmienna równała się 4 – jeżeli prawda to zostaje wypisany odpowiedni tekst w konsoli, a jeżeli fałsz to funkcja warunkowa przechodzi do ostatniego warunku **else**.

Warunek else w tym przypadku zostanie użyty, gdy zmienna **a** nie równa się ani 4 ani 5. Else traktujemy zawsze jako ostatnią drogę, kiedy poprzednie warunki nie były prawdziwe.

Taką funkcję warunkową można zapisać również w inny sposób.

```

5  int main()
6  {
7      int a = 4;
8
9      if(a==4) {
10         cout << "Zmienna a jest równa 4" << endl;
11     }
12
13     if(a==5) {
14         cout << "Zmienna a jest równa 5" << endl;
15     }
16
17     if(a!=4 && a!=5)
18     {
19         cout << "Zmienna a nie jest równa ani 4 ani 5" << endl;
20     }
21

```

Ten kod wykonuje to samo co poprzedni, ale nie zostały użyte warunki **else** oraz **else if**.

Ostatnia funkcja warunkowa z tych trzech działa jak **else** z poprzedniego przykładu. W nawiasie mamy podany warunek (**a!=4 && a!=5**). Taki warunek mówi po prostu, **a** nie jest równe 4 ORAZ **a** nie jest równe 5.

W funkcji warunkowej możemy użyć następujących operatorów porównania.

Operator	Krótki opis
>	... jest większe od ...
>=	... jest większe lub równe niż ...
<	... jest mniejsze od ...
<=	... jest mniejsze lub równe niż ...
==	... jest równe ...
!=	... jest różne od ...

*cpp0x.pl*

W przykładzie powyżej został zastosowany operator !=

Mamy również do wykorzystania operatory logiczne

Nazwa	Język C++	Opis
i	&&	Iloczyn logiczny - wszystkie wartości muszą być prawdziwe, aby została zwrócona prawda.
lub		Suma logiczna - co najmniej jedna z wartości musi być prawdziwa, aby została zwrócona prawda.
negacja	!	Zanegowanie wartości - czyli zwrócenie wartości przeciwnej.

*cpp0x.pl*

## Pętle

Pętle działają w taki sposób, że to co się w nich znajduje jest wykonywane tyle razy, dopóki warunek pętli jest spełniony. W C++ (jak i w większości języków) wyróżniamy 3 pętle.

- for – przy podawaniu warunku z góry powinniśmy wiedzieć ile razy się wykona,
- do...while – ta pętla działa na zasadzie *pierw wykonaj a potem sprawdź*, innymi słowy pętla zawsze wykona się przynajmniej raz,
- while – można ją traktować jako odwrotność do...while – pierw sprawdza, a potem wykonuje.

### for

Pętla for daje nam największą elastyczność podczas jej tworzenia. Podajemy jej zmienną, która będzie służyć do iteracji pętli, podajemy jej warunek, który musi być spełniony, aby pętla się wykonała oraz podajemy jej w jaki sposób zmienna do iteracji ma się zmieniać po każdym wykonaniu pętli.

```

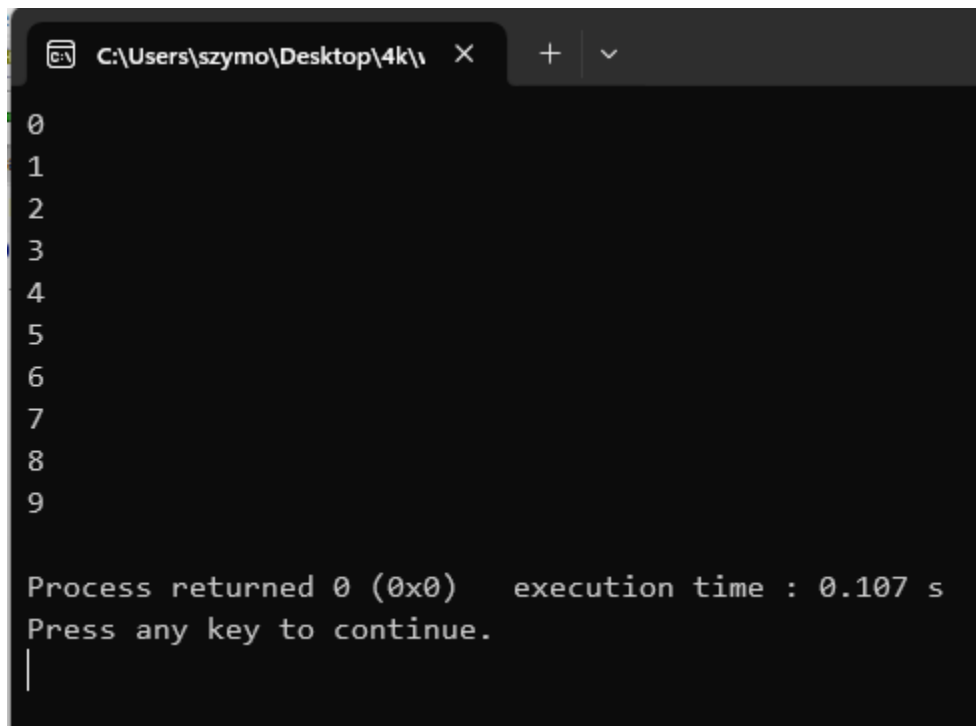
5  int main()
6  {
7      for(int i = 0; i < 10; i++){
8          cout << i << endl;
9      }
10
11     return 0;
12 }
13

```

Taka pętla `for` mówi nam, że definiuje zmienną (zmienna jest dostępna tylko z wnętrza pętli) i przypisuje do niej wartość 0, po średniku mamy warunek – pętla będzie się wykonywać dopóki `i` jest mniejsze od 10. **Jeżeli `i` będzie równe 10 to warunek już nie będzie prawdziwy!** Ostatnia informacja po średniku mówi nam w jaki sposób zmienna do iteracji ma się zmienić, w tym przypadku zwiększa się o 1.

Wewnątrz pętli znajduje się prosty `cout`, który wypisuje wartość zmiennej `i` przy każdym przejściu.

Wynik takiej pętli wygląda tak:



```

0
1
2
3
4
5
6
7
8
9

Process returned 0 (0x0)   execution time : 0.107 s
Press any key to continue.
|

```

Pętla `for` jest również przydatna do wypisywania wartości z tablicy, ponieważ w tej pętli definiujemy zmienną, której wartość się zmienia po każdym przejściu.

Dla przypomnienia tablice to inaczej swego rodzaju listy z wieloma wartościami. Każda wartość ma swoją pozycję – indeks. Indeksy w programowaniu zaczynają się od 0, a więc pierwsza wartość w tablicy ma indeks 0, a druga wartość ma indeks 1. Ostatnia wartość w tablicy będzie miała indeks `n - 1` gdzie `n` to rozmiar tablicy.

```

5  int main()
6  {
7      int tablica[10] = {10,9,8,7,6,5,4,3,2,1};
8
9      cout << "Indeks\tWartosc"<<endl;
10     for(int i = 0; i < 10; i++){
11         cout << i << ":\t" << tablica[i] << endl;
12     }
13
14     return 0;
15 }
16

```

Ten przykład pętli **for** wypisuje od nowej linii kolejne wartości tablicy *tablica* oraz ich indeks.

**Linia 7:** definiuje tablice *tablica* z liczbami całkowitymi (int), w kwadratowych nawiasach podawany jest rozmiar tablicy – 10 wartości. **C++ wymaga podawania rozmiaru tablicy przy jej definiowaniu!** W nawiasach klamrowych są podawane wartości, które oddzielane są przecinakami.

**Linia 9:** ten cout wypisuje tylko nazwę kolumn – po lewej będą indeksy, a po prawej wartości na tych pozycjach. `\t` pozwala nam na wpisanie tabulacji, dzięki czemu tekst będzie lepiej wyrównany.

**Linia 10:** definicja pętli for, identyczna do poprzedniego przykładu

**Linia 11:** Opisując od lewej do prawej:

1. wypisz wartość indeksu – zmienna *i*,
2. wypisz dwukropek oraz wstaw tabulację,
3. wypisz wartość z tablicy *tablica* o indeksie *i* – wartość zmiennej *i*,
4. zakończ linię

A oto wynik tej pętli:

```
Indeks  Wartosc
0:      10
1:       9
2:       8
3:       7
4:       6
5:       5
6:       4
7:       3
8:       2
9:       1

Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.
|
```

## do...while

```

5  int main()
6  {
7      int a = 1;
8
9      do
10     {
11         cout << "Wartosc a: " << a << endl;
12         a = a * 10;
13     }
14     while(a <= 100);
15
16     return 0;
17 }
18

```

Ta pętla jak i pętla **while** nie mają już zmiennej, którą można użyć jako iteratora czy indeks.

Na przykładzie powyżej pętla wykona się 3 razy, po każdym przejściu przez pętlę zmienna *a* zostanie pomnożona przez 10.

$a = a * 10$  można również zapisać w skróconej formie  $a *= 10$  – taki skrócony format zapisu działa dla każdego działania arytmetycznego: dodawanie, odejmowanie, mnożenie lub dzielenie.

Oto wynik tej pętli:

```

Wartosc a: 1
Wartosc a: 10
Wartosc a: 100

Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.

```

Ale co by się stało jeżeli byśmy zmienili nasz warunek  $a \leq 100$  na **false**? W taki sposób pętla od razu ma niespełniony warunek, ponieważ warunki logiczne działają tak, że zwracają zawsze prawdę lub fałsz.

```

13     }
14     while(false);

```

Zmieniliśmy tylko warunek na końcu pętli, a oto jej wynik działania:

```

Wartosc a: 1

Process returned 0 (0x0)   execution time : 0.027 s
Press any key to continue.
|

```

Jak widać pętla i tak wykonała się jeden raz, ponieważ do...while sprawdza warunek dopiero po wykonaniu zawartości pętli.

## Pętla while

```

5  int main()
6  {
7      int a = 1;
8
9      while(a <= 100)
10     {
11         cout<<"Wartosc a: " << a << endl;
12         a *= 10;
13     }
14
15     return 0;
16 }

```

Tak wygląda pętla **while**. Wygląda podobnie do **do...while** poza brakiem **do**, a **while** jest na początku pętli wraz z warunkiem logicznym. Sama definicja tej pętli w porównaniu do poprzedniej powinna nam sugerować, że warunek jest sprawdzany zawsze na początku pętli.

Wynik tego kodu wygląda tak:

```

Wartosc a: 1
Wartosc a: 10
Wartosc a: 100

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
|

```

Ale gdy zmienimy nasz warunek znowu na **false** to tym razem pętla w ogóle się nie wykona.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int a = 1;
8
9      while(false)
10     {
11         cout<<"Wartosc a: " << a << endl;
12         a *= 10;
13     }
14
15     return 0;
16 }
17
```

C:\Users\szymo\Desktop\4k\1 X + v

Process returned 0 (0x0) execution time : 0.026 s  
Press any key to continue.  
|



## Zadania

### **Zadanie 1** ( na ocenę 2 )

Napisz program, który będzie prosił Ciebie o podanie imienia i nazwiska, jeżeli podane imię i nazwisko będzie równe „Andrzej Duda” to program ma odpisać „Witam Panie Prezydencie”, w innym przypadku ma po prostu napisać „Witam imię i nazwisko”.

### **Zadanie 2** ( na ocenę 3 )

Dodaj do zadania 1, że jeżeli po podamy imię i nazwisko prezydenta to program nas poprosi o tajne hasło, jeżeli hasło będzie dobre to wyświetli się „Hasło poprawne”, a jeżeli złe to wyświetli się „Odmowa dostępu, wzywam ABW”.

### **Zadanie 3** ( na ocenę 4 )

Do poprzedniego zadania dodaj możliwość 3 prób przy podawaniu hasła i dopiero po 3 próbach odmów dostępu i zadzwoń po ABW.

Program ma informować na której próbie jest użytkownik.

Zadania od 1 do 3 wymagają użycia instrukcji **cin** do wprowadzania danych z klawiatury.

Więcej informacji:

<https://cpp0x.pl/kursy/Kurs-C++/Poziom-1/Obsluga-strumienia-wejsciwego/12>

### **Zadanie 4** ( na ocenę 5 )

Utwórz prosta grę, w której trzeba odgadnąć liczbę, program ma podpowiadać czy podana liczba przez użytkownika jest mniejsza bądź większa od wylosowanej przez program. Po odgadnięciu liczby program ma nam pogratulować i zakończyć działanie.

### **Zadanie 5** ( na ocenę 6 )

Do programu z zadania 4 dodaj, że po odgadnięciu liczby program wypisuje liczbę nieprawidłowych zgadnięć oraz po zakończonej grze, program ma się zapytać czy chcemy spróbować ponownie. Napisanie *tak* zacznie nową grę, a napisanie *nie zakończy program*.

Do zadań 4 oraz 5 będzie potrzeba funkcja `rand()`, która opisana jest [tutaj](#).

Pamiętaj o załączeniu biblioteki `cstdlib`.

Więcej informacji o użyciu funkcji `rand()` znajdziesz w akapicie o tytule **Losowanie liczb z określonego zakresu** w linku powyżej.

Dodatkowo w tych zadaniach potrzebna będzie pętla która wykonuje się w nieskończoność np. `while(true){/*Kod*/}`. Aby zakończyć taką pętlę np. po odgadnięciu liczby, użyj instrukcji **break**;