

Wykład: 4

Instrukcje sterujące, operatorzy



Instrukcje sterujące



```
for ( instrukcja_ini ; wyrażenie_warunkowe ; instrukcja_krok )  
    tresc_petli ;
```

- **instrukcja_ini** - instrukcja wykonywana zanim pętla zostanie po raz pierwszy uruchomiona
- **wyrażenie_warunkowe** – wyrażenie obliczane przed każdym obiegiem pętli. Jeżeli jest ono różne od zera, to pętla będzie dalej wykonywana
- **instrukcja_krok** – instrukcja wykonywana po zakończeniu każdego obiegu pętli

Pętla for - przykład

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <cmath>
4
5  int main()
6  {
7      int n, k;
8      double s = 0;
9      printf("n = ");
10     scanf("%d", &n);
11     for (k=2; k<=n; k*=2)
12         s += 1.0/k;
13     printf("Suma = %lf\n", s);
14     getch();
15     return 0;
16 }
17
```

$$x_n = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n}$$



Przykład:

Program wypisuje kody ASCII

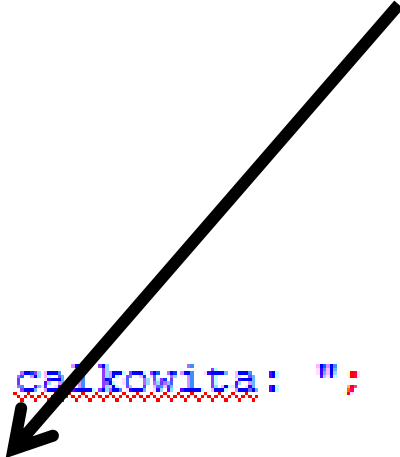
```
1  #include <iostream>
2  using namespace std;
3  |
4  int main()
5  {
6      int znak = 27;
7      for (; znak < 256 ; znak++)
8          cout << (char)znak << " ";
9      return 0;
10 }
11
```

Przykład:

Liczba cyfr liczby całkowitej

```
1  #include <iostream>
2
3  using namespace std;
4  int main()
5  {
6      long n;
7      int c;
8      cout << "Podaj liczbe calkowita: ";
9      cin >> n;
10     for (c=1; n/=10; c++) ;
11         cout << "Liczba cyfr wynosi: " << c << endl;
12     return 0;
13 }
14
```

Pusta pętla for
Pozornie „nic nie robi”





```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int n, a1=1, a2=1, temp;
6
7      cout << "Ile elementow wyliczyc? ";
8      cin >> n;
9      if(n<=1)
10         cout << a1 << " ";
11     else if (n<=2)
12         cout << a1 << " " << a2 << " ";
13     else
14     {
15         cout << a1 << " " << a2 << " ";
16         for (int i=3; i<=n; i++)
17         {
18             temp = a1 + a2;
19             a1 = a2;
20             a2=temp;
21             cout << a2 << " ";
22         }
23     }
24     return 0;
25 }
```

Przykład:
Ciąg Fibonacciego
Wersja iteracyjna
fib={1,1,2,3,5,8,13,21,.....}

Pętla for – pętla w pętli

Przykład: tabliczka mnożenia

```
1  #include <iostream>
2
3  using namespace std;
4  int main()
5  {
6      int i,j;
7      for(i=1;i<=10;i++)
8      {
9          for(j=1;j<=10;j++)
10         {
11             cout << (i*j) << "\t";
12         }
13         cout << endl;
14     }
15     return 0;
16 }
```



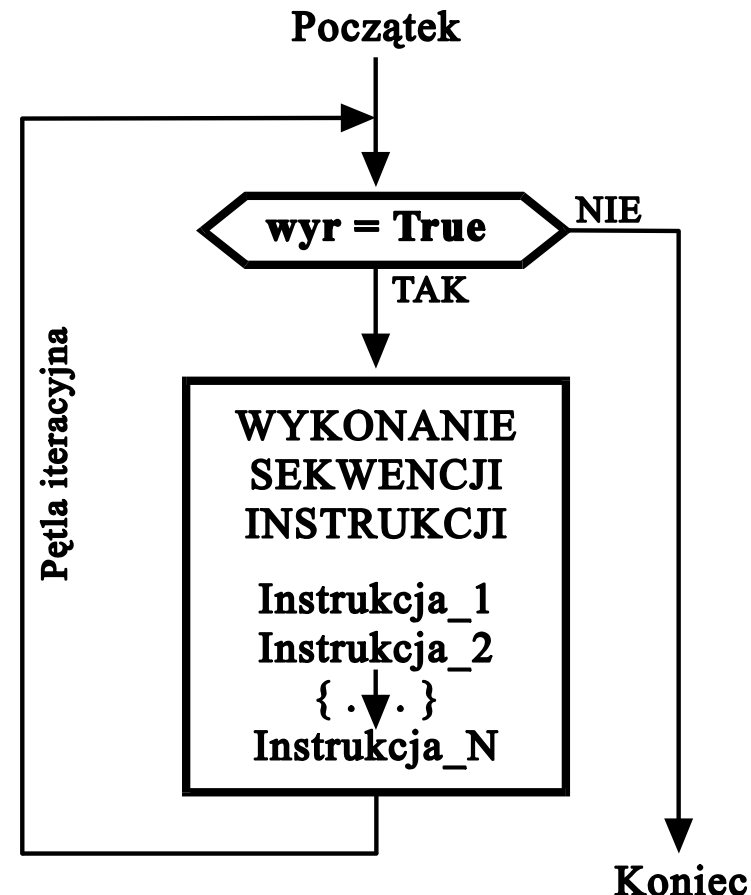
```
while (wyrazenie)
instrukcja;

.....

while (wyrazenie)
{
    instrukcja_1;
    instrukcja_2;
}
```

// pętla wyświetlająca liczby 1, 2, 3 ...

```
int i = 1;
while( i <=10 ) cout << i++ << " , ";
```



**Przykład:**

pętla wyświetlająca liczby 1, 2, 3 ...

```
int i = 1;  
while( i <=10 ) cout << i++ << ", ";
```



Przykład:

Algorytm Euklidesa

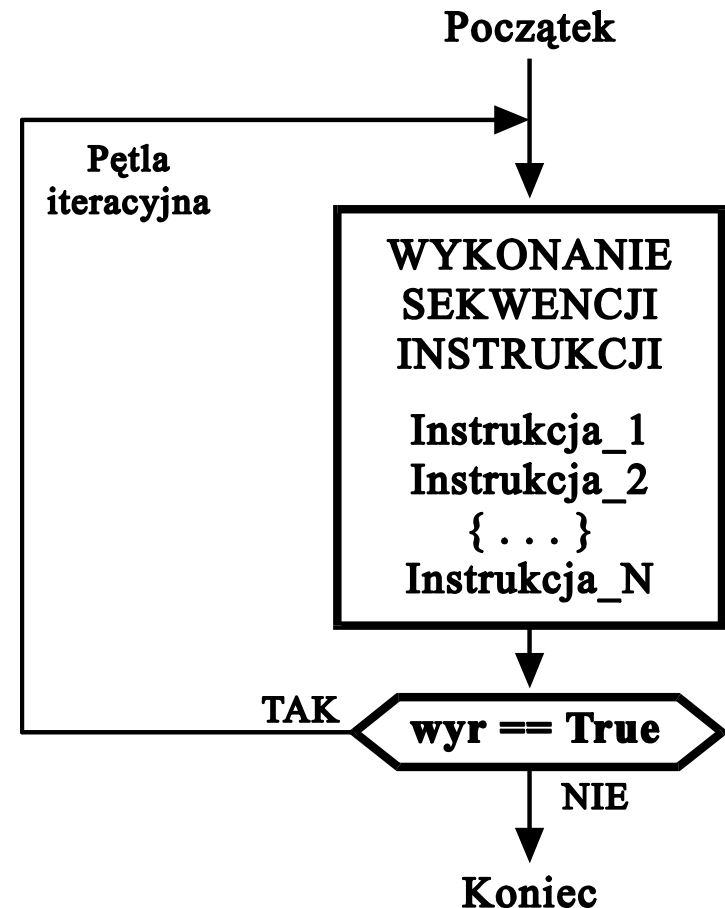
```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      unsigned int a, b;
6      cout << "a = ";    cin >> a;
7      cout << "b = ";    cin >> b;
8      while (a!=b)
9          if (a>b) a-=b; else b-=a;
10     cout << "NWP(a,b) = " << a << endl;
11     return 0;
12 }
```

Pętla do while



```
do
    instrukcja;
while ( wyrażenie );
```

```
do
{
    instrukcja_1;
    instrukcja_2;
}
while ( wyrażenie );
```



Pętla do while

Przykład:

pętla wyświetlająca liczby 1, 2, 3 ...

```
int i = 1;
do
{
    cout << i << ", ";
    i = i + 1;
}
while( i<=10);
```

Instrukcje break i continue

break - kończy wykonywanie najbliższej otaczającej pętli lub instrukcji warunkowej, w której występuje. Jeśli po końcu przerwanej instrukcji występuje kolejna, sterowanie przechodzi do niej.

Przykład:

Pętla kończy się po wypisaniu $i = 4$

```
for (int i = 1; i < 10; i++)  
{  
    cout << i << '\n';  
    if (i == 4)  
        break;  
}
```



Wyjście z pętli

Instrukcje break i continue

Instrukcja **break** jest używana z instrukcją warunkową **switch**, a także z instrukcjami pętli **do, for** i **while**.

W instrukcji **switch**, instrukcja **break** powoduje, że program wykonuje kolejną instrukcję, która występuje po instrukcji **switch**. Bez instrukcji **break**, wykonywane są wszystkie instrukcje od dopasowanej etykiety **case** do końca instrukcji **switch**, łącznie z klauzulą **default**.

W pętlach, instrukcja **break** kończy wykonywanie najbliższej otaczającej instrukcji **do, for** lub **while**. Sterowanie przechodzi do instrukcji następującej po zakończonej, jeśli taka istnieje.

Instrukcja break i co

continue - wymusza przekazanie kontroli do wyrażenia kontrolującego najmniejszej pętli **do**, **for**, lub **while**.

Przykład:

Pętla wypisze liczby od 1 do 10 pomijając liczbę 5

```
for (int i = 1; i <= 10; i++)  
{  
    if (i == 5) continue;  
    cout << i << '\n';  
}
```

Powrót na górę pętli
(z pominięciem instrukcji
poniżej continue)

Instrukcje break i continue

Przykład:

Algorytm Euklidesa – wersja 2

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      unsigned int a, b;
6      cout << "a = ";   cin >> a;
7      cout << "b = ";   cin >> b;
8      while (1)
9          if (a>b) a-=b;
10         else if (a<b) b-=a;
11         else break;
12     cout << "NWP(a,b) = " << a << endl;
13     return 0;
14 }
```

Pętla nieskończona

Wyjście z pętli



return - kończy wykonywanie funkcji i zwraca sterowanie do funkcji wywołującej (lub do systemu operacyjnego, jeśli kontrola zostanie przeniesiona z funkcji main). Wykonanie wznowia działanie w funkcji wywołującej w punkcie bezpośrednio po wywołaniu.

```
int normalizuj (int a)
{
    if (a>0) return(a);
    else if (a<0) return (-a);
    else return (0);
}
```



Operator



Operatory arytmetyczne:

$a = b + c$; // dodawanie
 $a = b - c$; // odejmowanie
 $a = b * c$; // mnożenie
 $a = b / c$; // dzielenie
 $a = b \% c$; // modulo – reszta z dzielenia

Priorytet operatorów arytmetycznych jest taki sam, jak w matematyce. Czyli zapis:

$$a + b \% c * d - f$$

oznacza to samo co

$$a + ((b \% c) * d) - f$$



Operatory arytmetyczne – zapis skrócony:

Dodawanie	$x = x + y;$	$x += y;$
Odejmowanie	$x = x - y;$	$x -= y;$
Mnożenie	$x = x * y;$	$x *= y;$
Dzielenie	$x = x / y;$	$x /= y;$
reszta z dzielenia	$x = x \% y;$	$x \% = y;$



Operatory inkrementacji i dekrementacji

i++ ; // inkrementacja - oznacza to: $i = i + 1$

i-- ; // dekrementacja – oznacza to: $i = i - 1$

Operatory te mogą mieć dwie formy:

- **Prefix: ++a** (przed argumentem) - najpierw zmienna jest zwiększana o 1, następnie ta zwiększona wartość staje się wartością wyrażenia.
- **Postfix: a++** (po argumentem) - najpierw brana jest stara wartość zmiennej i ona staje się wartością wyrażenia, a następnie zwiększany jest on o 1



Operator przypisania

Każde przypisanie samo w sobie jest także wyrażeniem mającym taką wartość, jaka jest przypisywana.

Zatem wartość wyrażenia:

$(x = 2)$

jako całości jest także 2



Operatory relacji

<	mniejszy
<=	mniejszy lub równy
>	większy
>=	większy lub równy
==	czy równy
!=	czy różny

Uwaga na błąd:

```
int a = 10;  
int b = 20;  
if (a=b) ..... // zamiast if (a==b)...
```

Nie jest błędem, tyle że oznacza

```
if (20) ..... // ogólnie if (b != 0) .....
```




Operatory sumy logicznej i iloczynu

- ||** - sumę logiczną - operację logiczną LUB (alternatywa)
- &&** - iloczyn logiczny - czyli operację I (koniunkcja)

„prawda” daje rezultat 1, a wynik „fałsz” daje rezultat 0

Wyrażenia logiczne tego typu obliczane są od lewej do prawej. Kompilator oblicza wartość wyrażenia dotąd, dopóki na pewno nie wie jaki będzie wynik.

Np.: jeżeli w wyrażeniu

`(a == 0) && (b != 10) && (c > 100)`

A będzie różne od zero, kolejne porównania nie będą wyliczane

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Grębosz J.: **Symfonia C++, Programowanie w języku C++ orientowane obiektowo**, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zająrzeć także do:

- Sokół R.: **Microsoft Visual Studio 2012 Programowanie w Ci C++**, Helion.
- Kernighan B.W., Ritchie D. M.: **język ANSI C**, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Grębosz J.: **Pasja C++**, Wydawnictwo Edition 2000.
- Meyers S.: **język C++ bardziej efektywnie**, Wydawnictwo Naukowo Techniczne