Języki programowania

Instrukcje sterujące

Wstęp

- Instrukcje sterujące:

  są to polecenia służące do sterowania przebiegiem programu.

- W instrukcjach sterujących podejmowane są decyzje o wykonaniu tych czy innych instrukcji programu.

- Decyzje te podejmowane są w zależności od spełnienia lub niespełnienia jakiegoś warunku (czyli od prawdziwości lub fałszywości jakiegoś wyrażenia).

Reprezentacja PRAWDY i FAŁSZU

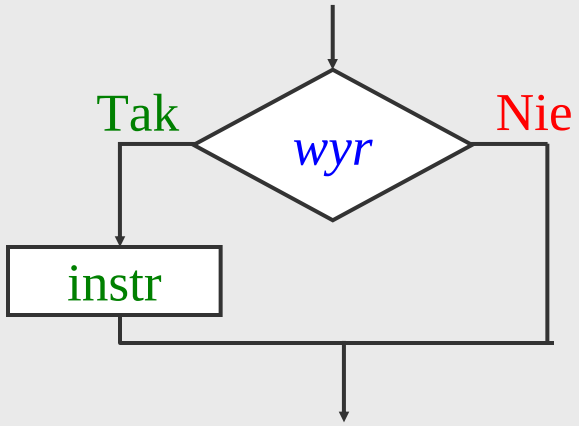
- W języku C++ nie ma specjalnego typu określającego zmienne logiczne (przyjmujące wartości prawda lub fałsz).
- Do przechowywania tej informacji nadaje się każdy typ!

wartość zero →	FAŁSZ
każda inna wartość →	PRAWDA

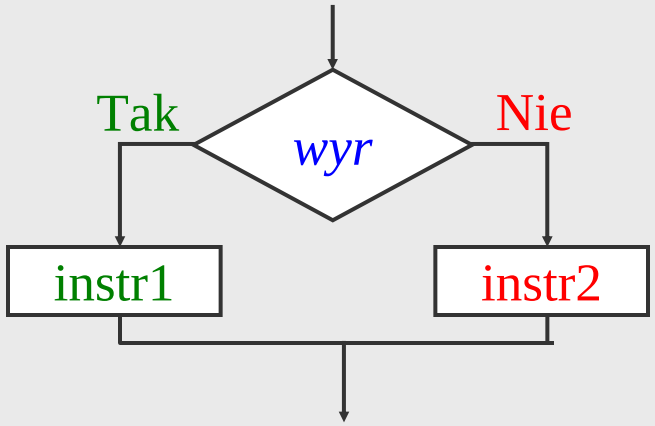
- Wartość PRAWDA lub FAŁSZ ma każde, dowolnie skomplikowane wyrażenie.
- Jego wartość nie musi być nawet liczbą.

Instrukcja warunkowa `if` (1)

```
if(wyr)
    instr;
```




```
if(wyr)
    instr1;
else
    instr2;
```



- Wyrażenie to coś, co ma jakąś wartość:
  - obiekt lub
  - rzeczywiste wyrażenie.
- Zamiast jednej instrukcji może wystąpić blok instrukcji.

Instrukcja warunkowa `if` (2)

- Przykład użycia instrukcji `if` do wyświetlenia modułu liczby rzeczywistej:

```
#include <iostream.h>

void main() {
   double x;
   cout << "Podaj liczbe rzeczywista: ";
   cin >> x;
   if(x > 0.0)        // obliczamy wartość bezwzględną z 'x'
      cout << x;
   else
      cout << (-x);
}
```

Blok instrukcji

- Często się zdarza, że chcemy wykonać warunkowo nie jedną, lecz kilka instrukcji. Stosujemy wówczas instrukcję składaną zwaną blokiem:

```
{
    instr1;
    instr2;
    ...
}
```

- Po klamrze nie trzeba stawiać średnika!
- Zróżnicowane wcięcia nie mają znaczenia dla kompilatora, ale ogromnie zwiększają czytelność kodu.

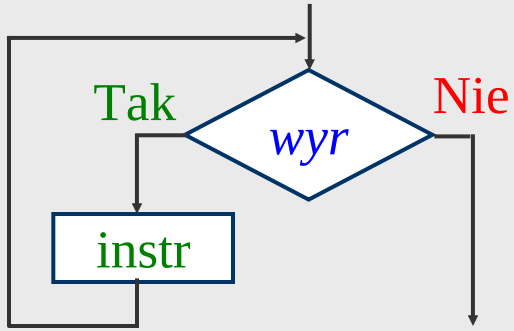
Wybór wielowariantowy

- Instrukcje warunkowe **`if..else`** możemy ze sobą łączyć – uzyskamy wtedy możliwość wyboru wielowariantowego.

```
if(wyrażenie1)
   instrukcja1;
else if(wyrażenie2)
   instrukcja2;
else if(wyrażenie3)
   instrukcja3;
else if(wyrażenie4)
   instrukcja4;
```

Pętla `while` (1)

```
while (wyr)
    instr;
```



Wartość obliczana przed wykonaniem instrukcji!

- Najpierw obliczana jest wartość wyrażenia ***wyr***.
- Jeśli wynik jest zerowy, wychodzimy z pętli.
- Jeśli wartość wyrażenia jest niezerowa, wykonujemy instrukcję **`instr`**.
- Instrukcję wykonujemy dopóki (*while*) wyrażenie ma wartość niezerową.
- Uwaga: pętla może w ogóle nie wykonać się ani razu!

Pętla `while` (2)

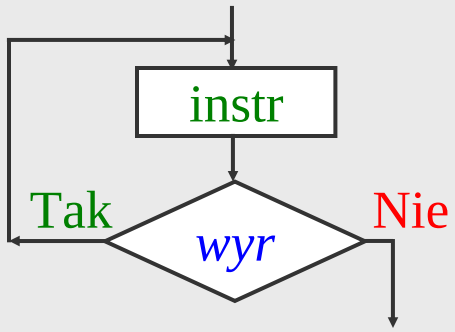
```
#include <iostream.h>
void main() {
   int ile;
   cout << "Ile gwiazdek ma miec kapitan? : ";
   cin >> ile;
   cout << "Wlasnie " << ile << " : ";
   while(ile) {                  // pętla rysująca gwiazdki
      cout << "*";
      ile = ile - 1;
   }
   cout << "\nTeraz zmienna ile ma wartosc " << ile << endl;
}
```

- Wygląd ekranu po wykonaniu programu:

```
Ile gwiazdek ma miec kapitan? : 4
Wlasnie 4 : ****
Teraz zmienna ile ma wartosc 0
```

Pętla `do..while` (1)

```
do
    instr;
while(wyr);
```



Wartość obliczana po wykonaniu instrukcji!

- Najpierw wykonywana jest instrukcja **`instr`**.
- Następnie obliczana jest wartość wyrażenia ***`wyr`***.
- Jeżeli jest ono niezerowe, to zostanie powtórzone wykonanie tej instrukcji.
- Jeśli wynik jest zerowy, wychodzimy z pętli.

- Uwaga: pętla wykona się co najmniej jeden raz!

Pętla do..while (2)

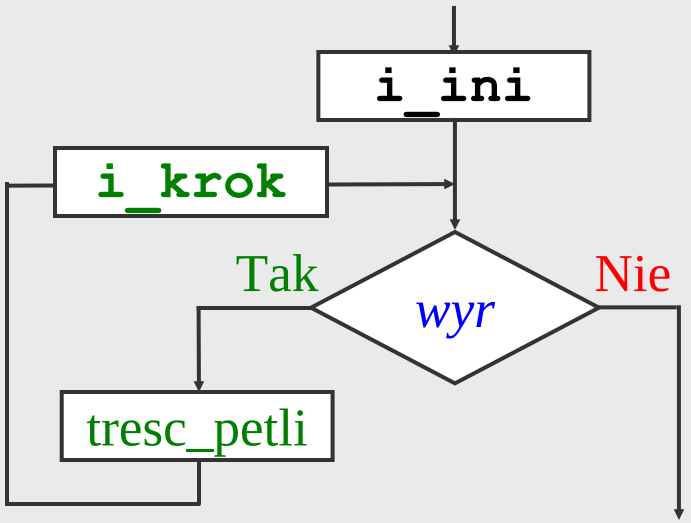
```
#include <iostream.h>
void main() {
   char litera;
   do {
      cout << "Napisz dowolna litere: ";
      cin >> litera;
      cout << "Napisano: " << litera << endl;
   } while(litera !='K');
   cout << "Napisano K, program zakonczono!\n";
}
```

- Wygląd ekranu po wykonaniu programu:

```
Napisz dowolna litere: a
Napisano: a
Napisz dowolna litere: k
Napisano: k
Napisz dowolna litere: K
Napisano: K
Napisano K, program zakonczono!
```

Pętla `for` (1)

```
for(i_ini;wyr;i_krok)
  tresc_petli;
```



Wartość obliczana przed wykonaniem treści pętli!

- Na początku, jednorazowo, wykonywana jest **`i_ini`**.
- Następnie obliczana jest wartość wyrażenia *wyr*.
- Jeśli wynik jest zerowy, wychodzimy z pętli.
- Jeśli wartość wyrażenia jest niezerowa, wykonujemy treść pętli, a potem **`i_krok`**.
- Instrukcję wykonujemy dopóki wyrażenie *wyr* ma wartość niezerową.
- Uwaga: pętla może w ogóle nie wykonać się ani razu, choć wykona się wtedy 1 raz **`i_ini`**.

Pętla `for` (2)

- **`i_ini`** nie musi być jedną instrukcją – może być ich kilka, wtedy oddzielone są przecinkami.
- Podobnie w wypadku **`i_krok`**.
- W **`i_ini`** może pojawić się definicja obiektu.
- Wyszczególnione elementy pętli: **`i_ini`**, *wyr* i **`i_krok`** – nie muszą wystąpić explicite.

  Dowolny z nich można opuścić.
- Zapis:

```
for( ; ; ) {
   .....
}
```

lub

```
while(1) {
   .....
}
```

oznacza pętlę nieskończoną.

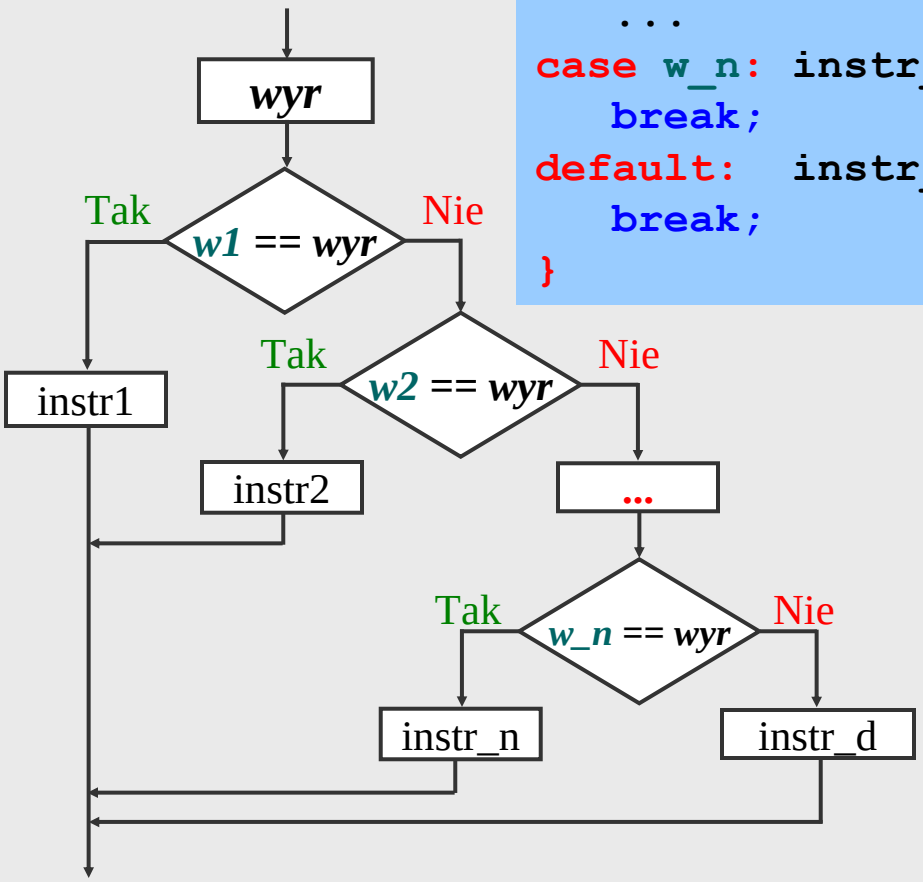
Pętla `for` (3)

```
#include <iostream.h>
void main() {
   int ile;
   cout << "Ile razy ma wykonac sie petla for? : ";
   cin >> ile;
   for(int i = 1; i <= ile; i = i + 1) {
      cout << "To jest " << i;
      cout << " wykonanie petli for!\n";
   }
}
```

- Wygląd ekranu po wykonaniu programu:

```
Ile razy ma wykonac sie petla for? : 4
To jest 1 wykonanie petli for!
To jest 2 wykonanie petli for!
To jest 3 wykonanie petli for!
To jest 4 wykonanie petli for!
```

Instrukcja wyboru `switch` (1)




```
switch(wyr) {
case w1:   instr1;
   break;
case w2:   instr2;
   break;
   ...
case w_n:  instr_n;
   break;
default:   instr_d;
   break;
}
```

- Instrukcja wyboru służy do podejmowania wielowariantowych decyzji.
- Najpierw obliczane jest wyrażenie *wyr*.
- Jeśli jego wartość odpowiada którejś z wartości podanej w jednej z etykiet **`case`**, to wykonywane są instrukcje po tej etykiecie, aż do napotkania instrukcji **`break`**.
- Jeśli wartość wyrażenia nie zgadza się z żadną z wartości przy etykietach **`case`**, wówczas wykonywane są instruckcje po etykiecie **`default`**.

Instrukcja wyboru `switch` (2)

- Instrukcja **`break`**:
  - powoduje wyskok z instrukcji **`switch`**.
- Etykieta **`default`**:
  - może znajdować się w dowolnym miejscu,
  - może w ogóle nie występować w instrukcji **`switch`** – wtedy, jeśli wartość wyrażenia nie zgadza się z żadną z wartości przy etykietach **`case`**, instrukcja **`switch`** nie wykonuje niczego.

- Instrukcji występujących po etykiecie **`case`** nie musi kończyć instrukcja **`break`**.

  Jeśli jej nie umieścimy, to zaczną wykonywać się instrukcje umieszczone pod następną etykietą **`case`**.

Instrukcja wyboru `switch` (3)

```
#include <iostream.h>
void main() {
   int nr;
   cin >> nr;
   switch(nr) {
      case 1:  cout << "Dla nr = " << nr << "\t%\n";
               break;
      case 2:  cout << "Dla nr = " << nr <<"\t&\n";
      case 3:  cout << "Dla nr = " << nr <<"\t*\n";
               break;
      default: cout << "Dla nr != 1,2,3" << "\t!!!!\n";
               break;
   }
}
```

- Wygląd ekranu po wykonaniu programu:

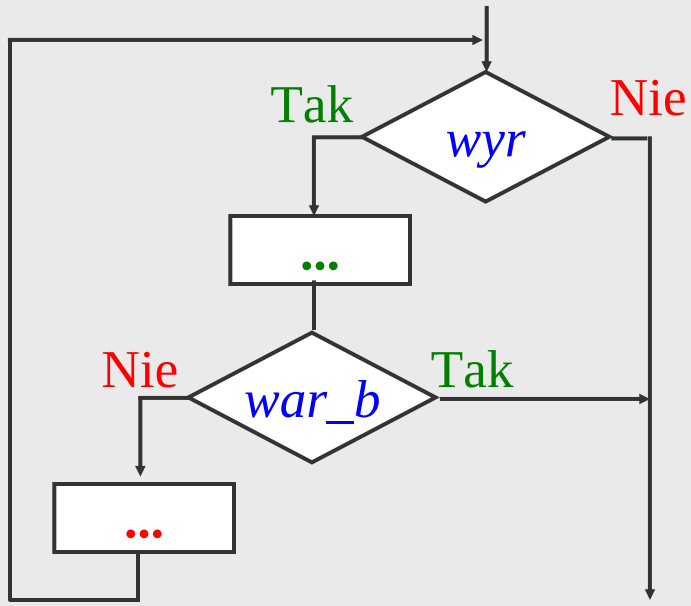
Dla różnych wartości zmiennej nr:

```
1
Dla nr = 1      %
4
Dla nr != 1,2,3 !!!!
```

```
2
Dla nr = 2      &
Dla nr = 3      *
3
Dla nr = 3      *
```

Instrukcja sterująca `break` (1)

```
while(wyr) {
    ...
    if(war_b) break;
    ...
}
```



- Instrukcja sterująca **`break`** przerywa natychmiast działanie innych instrukcji sterujących:
  - **`switch`**,
  - **`for`**,
  - **`while`**,
  - **`do..while`**.
- Jeśli instrukcja **`break`** występuje wewnątrz kilku zagnieżdżonych pętli, to przerywa działanie tylko tej pętli, w której bezpośrednio tkwi (jest to tak jakby wyjście „o jeden poziom wyżej”).

Instrukcja sterująca `break` (2)

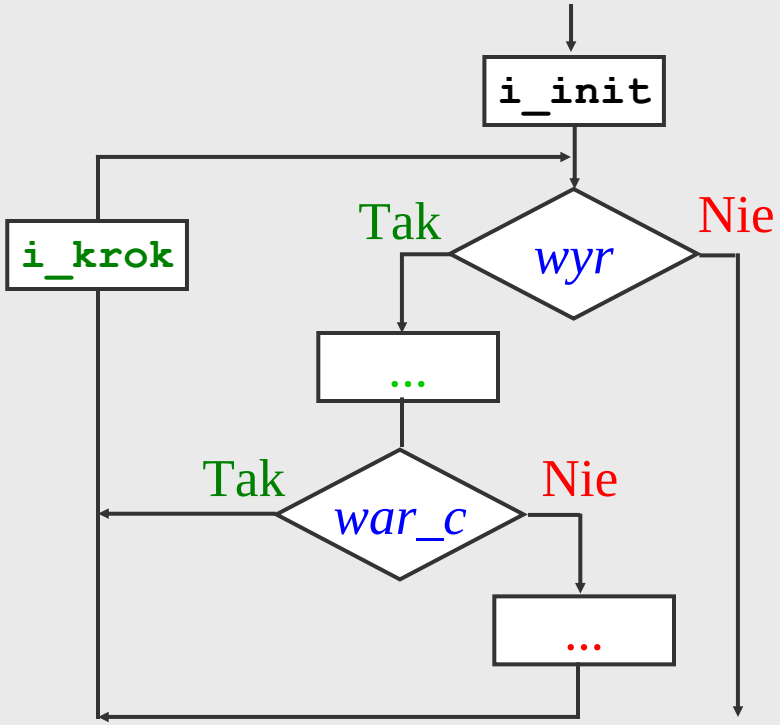
```
#include <iostream.h>
void main() {
        int i, j;
        int max_dl = 5;
        for(i=0; i < 3; i = i + 1) {
            for(j = 0; j < 100; j = j + 1) {
                cout << "*";
                if(j > max_dl)
                    break;
            }
            cout << "\nWywolanie zew. petli dla i = " << i << endl;
        }
}
```

- Wygląd ekranu po wykonaniu programu:

```
*******
Wywolanie zew. petli dla i = 0
*******
Wywolanie zew. petli dla i = 1
*******
Wywolanie zew. petli dla i = 2
```

Instrukcja sterująca `continue` (1)

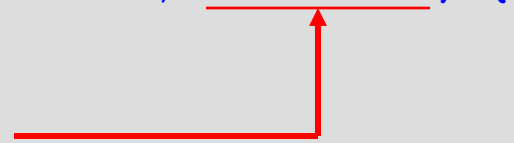
```
for(i_init; wyr; i_krok){
    ...
    if(war_c) continue;
    ...
}
```



- Instrukcja sterująca **`continue`** stosowana jest w pętlach.
- Powoduje ona zaniechanie realizacji instrukcji będących treścią pętli, jednak (w przeciwieństwie do instrukcji **`break`**) sama pętla nie zostaje przerwana.
- Instrukcja **`continue`** przerywa tylko aktualny obieg pętli i zaczyna następny, kontynuując pracę pętli.
- Po wykonaniu instrukcji **`continue`** w pętli **`for`** wykonywana jest **`i_krok`**.

Instrukcja sterująca `continue` (2)

```
#include <iostream.h>
void main() {
   for(int i = 0; i < 10; i = i + 1) {
      cout << "a";
      if(i > 2)
         continue;
      cout << "b\n";
   }
}
```



- Wygląd ekranu po wykonaniu programu:

```
ab
ab
ab
aaaaaaa
```

Instrukcja skoku `goto` (1)

```
    ...
    if(war)
        goto et1;
    instr1;
    instr2;
et1:
    instr3;
```

- Używanie instrukcji goto zdradza (zazwyczaj), że jest się złym programistą.
- Instrukcji goto zawsze da się uniknąć (choć nie zawsze jest to efektywne).

- Po napotkaniu instrukcji skoku `goto` wykonywanie programu przenosi się do miejsca oznaczonego etykietą.
- Z instrukcją `goto` wiąże się zawsze etykieta, do której należy przeskoczyć.
- Etykieta musi znajdować się w aktualnym zakresie ważności.
- Etykieta to nazwa, po której następuje dwukropek.

Instrukcja skoku `goto` (2)

- Jednym z kilku sensownym zastosowaniem **`goto`** w zwykłym kodzie jest umożliwienie wyjścia z zagnieżdżonej pętli lub instrukcji **`switch`** – daje to większą efektywność kodu.

```
#include <iostream.h>
void main() {
   while(1) {
      cout << "W petli while nr 1\n";
      while(2) {
         cout << "W petli while nr 2\n";
         for(int i = 0; i < 100; i++) {
            cout<<"W petli for\n";
            goto koniec;
         }
      }
   }
koniec:
   cout << "Przerwano dzialanie petli\n";
}
```

Wygląd ekranu po wykonaniu programu:

```
W petli while nr 1
W petli while nr 2
W petli for
Przerwano dzialanie
petli
```