Instrukcje sterujące

Ostatnia aktualizacja: 29 września 2026

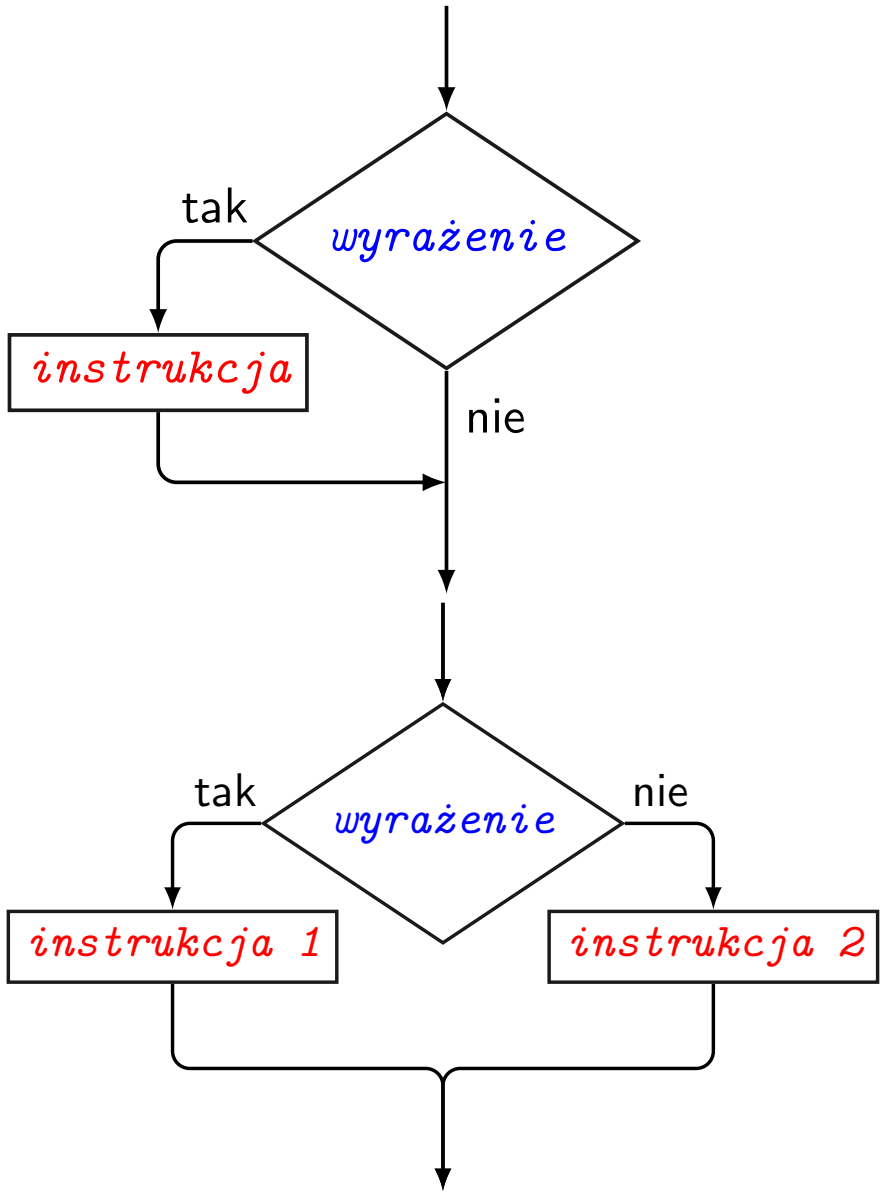
Przypomnienie: operatory

Operator przypisania			
=	przypisanie	`x = y`	$x \leftarrow y$
Operatory arytmetyczne			
*	mnożenie	`x * y`	$x \cdot y$
/	dzielenie	`x / y`	$\frac{x}{y}$
+	dodawanie	`x + y`	$x + y$
-	odejmowanie	`x - y`	$x - y$
%	reszta z dzielenia (modulo)	`x % y`	$x \mod 2$
++	inkrementacja	`x++`	$x \leftarrow x + 1$
--	dekrementacja	`x--`	$x \leftarrow x - 1$
Operatory relacji			
<	mniejszy niż	`x < y`	$x < y$
>	większy niż	`x > y`	$x > y$
<=	mniejszy lub równy	`x <= y`	$x \leqslant y$
>=	większy lub równy	`x >= y`	$x \geqslant y$
==	równy	`x == y`	$x = y$
!=	różny	`x != y`	$x \neq y$
Operatory logiczne			
!	negacja (NOT)	`!x`	$\neg x$
&&	koniunkcja (AND)	`x>1 && y<2`	$x > 1 \wedge y < 2$
||	alternatywa (OR)	`x<1 || y>2`	$x < 1 \vee y > 2$

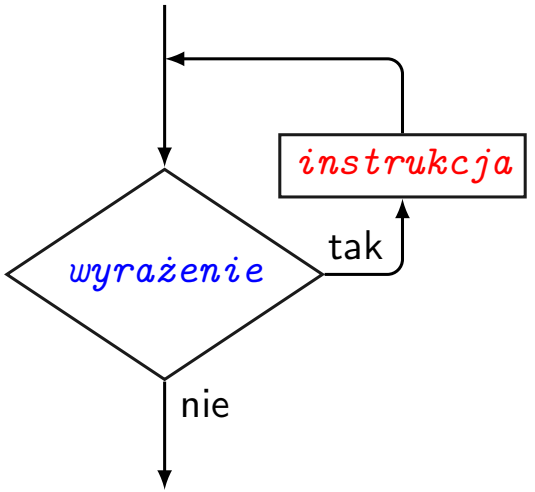
Instrukcja warunkowa if else

```
if ( wyrażenie )
    instrukcja
```

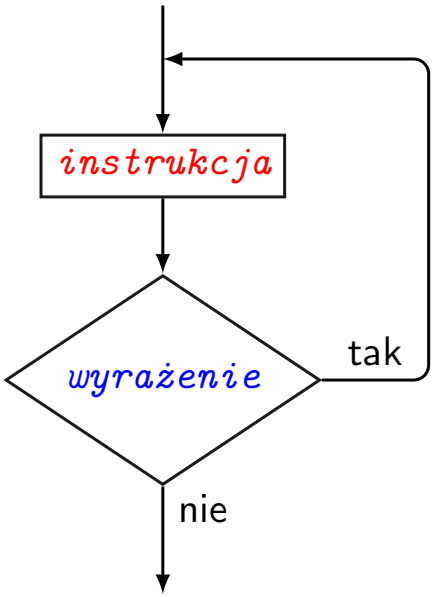
```
if ( wyrażenie )
    instrukcja 1
else
    instrukcja 2
```



```
while ( wyrażenie )
   instrukcja
```




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



Problem: wyznaczenie wartości pierwiastka $\sqrt{a}$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \xrightarrow{n\to\infty} \sqrt{a}$$

Algorithm Algorytm Herona

Dane wejściowe: liczba $a \geqslant 0$, wartość początkowa $x_0 > 0$, dokładność obliczeń $\epsilon > 0$

Wynik: wartość przybliżona $x \approx \sqrt{a}$ z dokładnością ϵ

1: $x \leftarrow x_0$
2: **wykonuj**
3: $\quad x_0 \leftarrow x$
4: $\quad x \leftarrow \frac{1}{2}\left(x_0 + \frac{a}{x_0}\right)$
5: **dopóki** $|x - x_0| > \epsilon$
6: **wypisz** x

```c
#include<stdio.h>

int main()
{
    const float eps = 1e-4;
    float x, a, x0;

    printf("a = ");
    scanf("%f", &a);

    if (a < 0) printf("Zle dane: a < 0\n");
    else
    {
        x0 = 1;
        x = x0;
        do
        {
            x0 = x;
            x = (x0 + a/x0)/2;
        } while (x - x0 > eps || x - x0 < -eps);

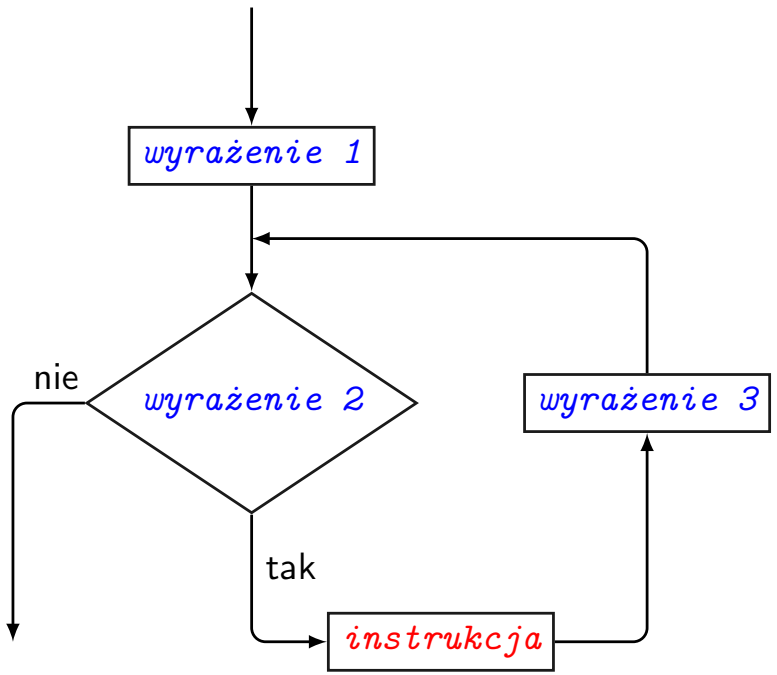
        printf("pierwiastek z %f wynosi %f (eps= %f)\n", a, x, eps);
    }
    return 0;
}
```

☞ heron.c

```
for ( wyrażenie 1 ; wyrażenie 2 ; wyrażenie 3 )
    instrukcja
```

```
int s=1, n=100, i;

for (i=1; i<n; i++)
{
    s = s * i;
}
```



```
A
while ( B )
{
    instrukcja
    C
}
```

```
for ( A ; B ; C )
{
    instrukcja
}
```

```
i=0;
while (i < n)
{
    printf("%d\n", i);
    i = i + 1;
}
```

```
for (i=0; i<n; i++)
    printf("%d\n", i);
```

break przerywa działanie pętli (while, do while, for).

```
1  #include<stdio.h>
2
3  int main()
4  {
5      char c;
6
7      while (1)
8      {
9          printf("Czy przerwac [t/n]? ");
10         scanf(" %c", &c);
11         if (c == 't' || c == 'T') break;
12         printf("Przmysl to!\n");
13     }
14
15     printf("Koniec.\n");
16 }
```

```
Czy przerwac [t/n]? n
Przmysl to!
Czy przerwac [t/n]? N
Przmysl to!
Czy przerwac [t/n]? y
Przmysl to!
Czy przerwac [t/n]? t
Koniec.
```

☞ break.c

Algorithm Metoda siłowa sprawdzania czy liczba jest liczbą pierwszą

Dane wejściowe: liczba całkowita $n > 0$

Wynik: odpowiedź: liczba n jest (lub nie jest) liczbą pierwszą

1: **dla** $i = 2, 3, \ldots, n-1$ **wykonuj**
2: **jeżeli** i jest dzielnikiem n **wykonaj**
3: **wypisz** `nie jest liczbą pierwszą`
4: **przerwij**
5: **jeżeli** nie znaleziono dzielnika **wykonaj**
6: **wypisz** `jest liczbą pierwszą`

```c
#include <stdio.h>

int main()
{
    int n, i = 2;

    printf("n = ");
    scanf("%d", &n);

    while (i < n)
    {
        if (n % i == 0) break;
        i = i + 1;
    }

    if (i == n || n == 1)
        printf("Liczba %d jest liczba pierwsza\n", n);
    else
        printf("Liczba %d nie jest liczba pierwsza\n", n);

    return 0;
}
```

☞ czy–pierwsza.c

Kolejna iteracja: continue

Polecenie `continue` przechodzi do następnej iteracji (pomija wszystkie pozostałe instrukcje do końca bloku pętli.

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int i, n=16;
6
7      for (i=1; i<=n; i++)
8      {
9          if (i==13) continue;
10         printf("%d\n",i);
11     }
12
13     return 0;
14 }
```

```
1
2
3
4
5
6
7
8
9
10
11
12
14
15
16
```

☞ continue.c

Instrukcja skoku `goto` przenosi sterowanie w miejsce kodu oznaczone etykietą.

```
etykieta:
instrukcje
...
goto etykieta;
```

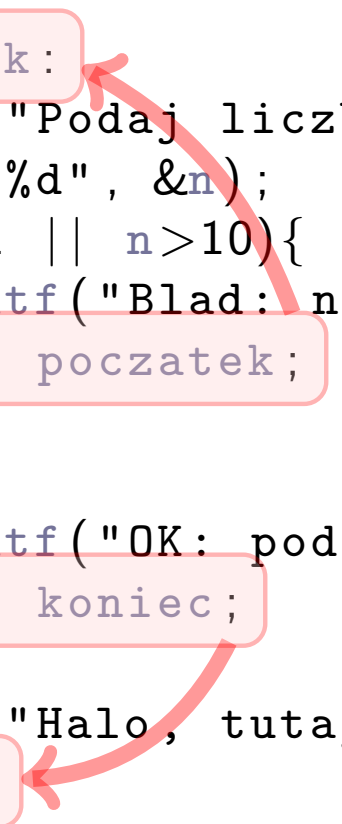
Główna zasada **programowania strukturalnego**:

Nie używaj instrukcji skoku!

```c
#include <stdio.h>

int main()
{
    int n;
    poczatek:
    printf("Podaj liczbe z zakresu od 1 do 10\n");
    scanf("%d", &n);
    if (n<1 || n>10){
        printf("Blad: niepoprawna wartosc\n");
        goto poczatek;
    }
    else{
        printf("OK: podales liczbe %d\n", n);
        goto koniec;
    }
    printf("Halo, tutaj jestem!\n");
    koniec:

    return 0;
}
```



☞ goto.c

Czemu skok to zazwyczaj zły pomysł?

- instrukcja skoku przerywa **liniowy przepływ programu**, kod staje się **nieczytelny i trudny do zrozumienia** (tzw. *spaghetti code*)
- kod ze skokami jest **trudniejszy do utrzymania i debugowania**, przepływ programu jest **mniej przewidywalny** a ewentualne modyfikacje kodu częściej prowadzą do powstawania błędów
- trudności techniczne, niektóre skoki są niewykonalne:

  skoki do wnętrza pętli, z pętli do pętli, do wnętrza funkcji, itd. ...
- skok omija strukturalne elementy kodu (pętle, bloki warunkowe), jest to sprzeczne z praktykami **programowania strukturalnego**
- kompilatory mogą mieć **trudności z optymalizacją** kodu zawierającego skoki, co może prowadzić do mniej efektywnego działania programu
- `break` i `continue` - podobne wątpliwości, ale to skoki lokalne
- kod ze skokami zawsze można przepisać do postaci bez skoków
- używanie instrukcji skoku świadczy o złym stylu programowania

Poprzedni przykład bez skoku

```c
#include <stdio.h>

int main()
{
    int n;
    int stop = 0;

    while (stop == 0)
    {
        printf("Podaj liczbe z zakresu od 1 do 10\n");
        scanf("%d", &n);
        if (n >= 1 && n <= 10) stop = 1;
        else printf("Blad: niepoprawna wartosc\n");
    }

    printf("OK: podales liczbe %d\n", n);
    return 0;
}
```

☞ goto2.c

Pętle zagnieżdżone

```c
#include <stdio.h>

int main ()
{
    int i, j, n;

    printf("Podaj zakres: ");
    scanf("%d", &n);

    printf("Liczby pierwsze w zakresie od 1 do %d\n", n);

    for (i=2; i<n; i++) {
        for (j=2; j <= (i/j); j++)
            if (!(i%j)) break;          /* nie jest l. pierwsza */
        if (j > (i/j)) printf("%d\n", i);
    }

    return 0;
}
```

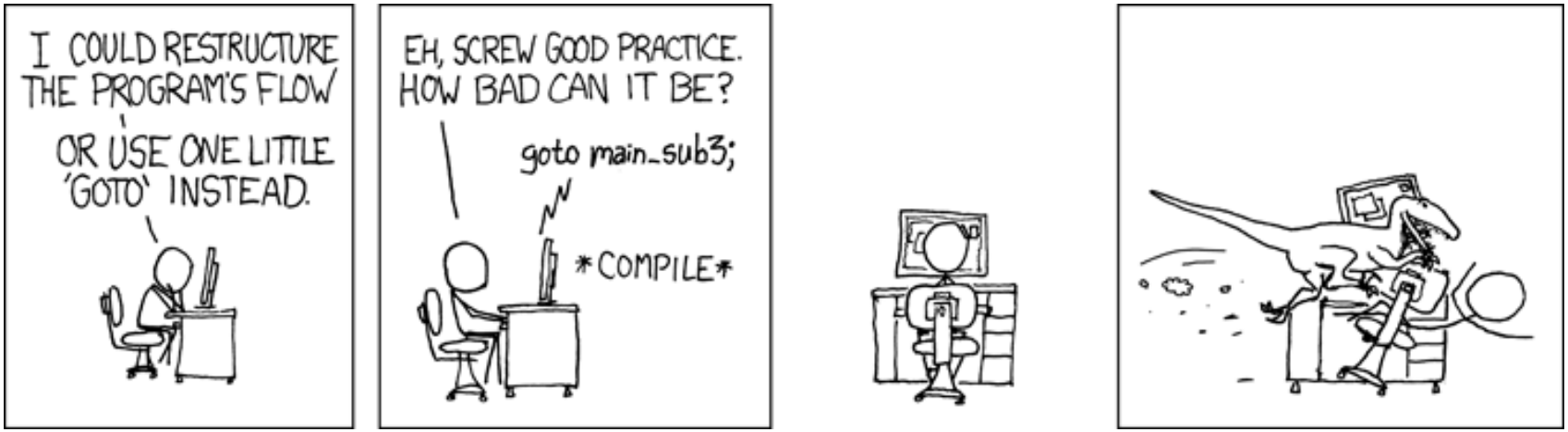
☞ prime.c

Pętle zagnieżdżone i instrukcja skoku

```
1   int i, j, k;
2
3   for (i =0; i < 100; i = i + 1)
4   {
5      for (j =0; j < i; j = j + 1)
6      {
7         for (k=0; k < j; k = k + 1)
8         {
9            if (i + j + k > 42) goto koniec;
10        }
11     }
12  }
13
14  koniec:
```

☞ goto4.c

- przykładowe uzasadnienie użycia instrukcji skoku - zakończenie zagnieżdżonej pętli
- ale zagnieżdżone pętle to też oznaka złych praktyk



http://xkcd.com/292/

Instrukcja warunkowa `switch` porównuje wyrażenie z listą wartości i wykonuje instrukcje pasującego przypadku (`case`).

```
switch ( wyrażenie )
{
    case wartość 1:
        instrukcja 1
        break;
    case wartość 2:
        instrukcja 2
        break;
    ...
    default:
        instrukcja n
}
```

```c
#include<stdio.h>

int main()
{
    float x, y;
    char op;

    scanf("%f %c%f", &x, &op, &y);

    switch (op)
    {
        case '+' :
            printf("%f\n", x + y);
            break;
        case '-' :
            printf("%f\n", x - y);
            break;
        case '*' :
            printf("%f\n", x * y);
 //           break;
        case '/' :
            printf("%f\n", x / y);
//          break;
        default:
            printf("Nieznana operacja: %c\n", op);
    }
    return 0;
}
```

```
3.14 + 5
8.140000

3.14+5
8.140000

3.14
+
5
8.140000
```

☞ switch.c

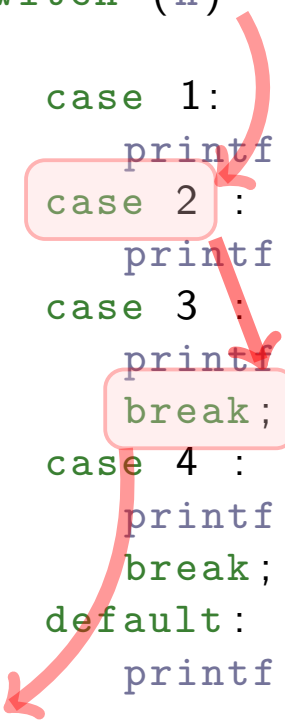
- Dopasowanie przypadków możliwej jest tylko dla wartości całkowitych (`int`, `char`, `enum`)
- `break` kończy działanie instrukcji `switch`
- instrukcje `break` i `default` są opcjonalne
- Po dopasowaniu właściwego przypadku wykonywane są wszystkie instrukcje aż do napotkania pierwszego wystąpienia `brake` (lub do końca bloku `switch`). Pominięcie instrukcji `break` spowoduje więc wykonanie instrukcji dla kolejengo przypadku.

```c
#include<stdio.h>

int main()
{
    int n;

    printf("Podaj liczbe od 1 do 4\n");
    scanf("%d", &n);

    switch (n)
    {
        case 1:
            printf("Przypadek 1\n");
        case 2 :
            printf("Przypadek 2\n");
        case 3 :
            printf("Przypadek 3\n");
            break;
        case 4 :
            printf("Przypadek 4\n");
            break;
        default:
            printf("Nieznana operacja.\n");
    }
    return 0;
}
```



```
Podaj liczbe od 1 do 4
2
Przypadek 2
Przypadek 3
```

Instrukcje sterujące przepływem programu w języku C:

- Instrukcje warunkowe: `if`, `else`, `switch`
- Pętle: `while`, `do while`, `for`
- Instrukcja wielokrotnego wyboru `switch`
- Instrukcja skoku `goto` (lepiej nie używać, łamie zasady programowania strukturalnego)
- Instrukcja przerwania bieżącej pętli `break`
- Instrukcja kontynuacji bieżącej pętli `continue`

Literatura dodatkowa


David Griffiths, Dawn Griffiths *„Rusz głową! C.”*, Helion, Gliwice, 2013.


„Kurs programowania w C”, WikiBooks, `http://pl.wikibooks.org/wiki/C`