Instrukcje, definicje i bloki

Instrukcja to najmniejszy samodzielny element imperatywnego języka programowania. Instrukcja może być niskiego poziomu napisana w asemblerze, która po przetłumaczeniu na kod binarny (język maszynowy) nadaje się do uruchomienia przez procesor lub instrukcja wysokiego poziomu napisana np. w języku C: *int a = 5;*, która zostanie przetłumaczona na kilka instrukcji niskiego poziomu. Program komputerowy jest tworzony jako lista różnych instrukcji. Instrukcja może zawierać wewnętrzne komponenty (np. wyrażenia).

Wiele języków programowania (w tym język C) w swojej syntaktyce rozróżnia **instrukcje** i **definicje** – instrukcja zawiera kod wykonywalny (np.: *int a=5;*), a definicja deklarację identyfikatora (np.: *int a;*).

Instrukcje możemy podzielić na:

- Instrukcje proste, np.:
  - Instrukcja przypisania, np.:
    - *a=0;*
    - *int b=a+9.988;*
    - *char znak='a';*
  - Instrukcja wyjścia/powrotu, np.:
    - *return 0;*
  - Instrukcja pusta, np.:
    - *if(warunek);*
    - *if(warunek);*
      else instrukcja;
- Instrukcje złożone (mogą zawierać dowolne inne instrukcje jako komponenty), np.:
  - Instrukcja blokowa (grupująca), np.:
    - ```
      {
        int a, b, c;
        a=0;
        b=12;
        c=a+b;
      }
      ```
  - Instrukcja warunkowa, np.:
    - ```
      if (a==b) {
        printf ("a jest rowne b\n");
      } else {
        printf ("a nie jest rowne b\n");
      }
      ```
  - Instrukcja wyboru, np.:
    - ```
      switch (wyr) {
        case wart1: /* instrukcje, jeśli wyr == wart1 */
          break;
        case wart2: /* instrukcje, jeśli wyr == wart2 */
          break;
        default: /* instrukcje, jeśli wyr != wart1 i wyr != wart2 */
          break;
      }
      ```
  - Pętla, np.:
      ```

- *while (a <= 10) {*
  *printf ("%d\n", a*a);*
  a++;
  }

Bloki (instrukcje blokowe/grupujące – zob. wyżej) to wydzielone fragmenty kodu, które zawierają w sobie ciąg instrukcji. Każdy blok rozpoczyna się znakiem *{* i kończy znakiem *}*. Głównym zadaniem bloków jest właśnie grupowanie instrukcji. Ponieważ bloki to również instrukcje, dlatego mogą być one wzajemnie zagnieżdżane.

Przykłady

```c
#include<stdio.h>
int main()
{
    int a;
    printf("Wprowadz liczbe\n");
    scanf("%d", &a);
    printf("a= %d", a);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    { // poczatek bloku instrukcji
        int a;
        printf("Wprowadz liczbe\n");
        scanf("%d", &a);
        printf("a= %d", a);
    } // koniec bloku instrukcji
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    { // poczatek bloku instrukcji (blok b1)
        int a;
        { // poczatek bloku instrukcji (blok b2)
            printf("Wprowadz liczbe\n");
            scanf("%d", &a);
        } // koniec bloku instrukcji (blok b2)
        printf("a= %d", a);
    } // koniec bloku instrukcji (blok b1)
    return 0;
}
```

ZADANIE

Przetestuj działanie powyższych kodów programów. Jakie są różnice w działaniu tych programów?

Instrukcja warunkowa *if* oraz *if else*

Instrukcja warunkowa to element języka programowania, który pozwala na wykonanie różnych instrukcji w zależności od tego czy zdefiniowane przez programistę wyrażenie logiczne (warunek) jest prawdziwe, czy fałszywe. Zatem instrukcja ta daje możliwość warunkowego decydowania o tym, jaki krok zostanie wykonany w dalszej kolejności.

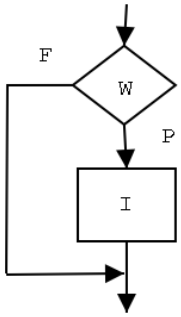
W języku C wyróżniamy 2 rodzaje instrukcji warunkowych:

- instrukcja *if*
- instrukcja *if else*

Postać ogólna instrukcji *if* wygląda następująco:

```c
if (wyrażenie) {
  /* blok wykonany, jeśli wyrażenie jest prawdziwe */
}
/* dalsze instrukcje */
```

Co zobrazować można rysunkiem:

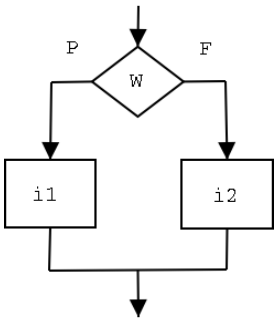


gdzie **W** to warunek logiczny, od którego zależy dalsze działanie programu. **P** i **F** oznaczają odpowiednio prawdę i fałsz – możliwe wyniki sprawdzenia określonego warunku, zaś **I** to instrukcja (blok instrukcji), która zostanie wykonana gdy wynik sprawdzenia warunku logicznego będzie prawdziwy.

Postać ogólna instrukcji *if else* jest następująca:

```
if (wyrażenie) {
  /* blok wykonany, jeśli wyrażenie jest prawdziwe */
} else {
  /* blok wykonany, jeśli wyrażenie jest nieprawdziwe */
}
/* dalsze instrukcje */
```

Co wyrazić można rysunkiem:



gdzie **P** i **F** oznaczają odpowiednio prawdę i fałsz – możliwe wyniki sprawdzenia warunku logicznego **W**, od którego zależy dalsze działanie programu. Instrukcja (blok instrukcji) **i1** zostanie wykonana gdy wynik sprawdzenia warunku logicznego będzie prawdziwy, zaś instrukcja (blok instrukcji) **i2** - gdy wynik sprawdzenia warunku logicznego będzie fałszywy.

Przykłady

```c
#include<stdio.h>
int main()
{
    if (10 > 5)
    {
        printf("warunek jest prawdziwy\n");
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    if (10 > 5)
    {
        printf("warunek jest prawdziwy\n");
    }
    else
    {
        printf("warunek jest falszywy\n");
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    if (10 > 5)
        printf("warunek jest prawdziwy\n");
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    if (10 > 5)
        printf("warunek jest prawdziwy\n");
    else
        printf("warunek jest falszywy\n");
    return 0;
}
```

W powyższych przykładach kody programów po lewej stronie działają identycznie jak kody po prawej, choć wyglądają nieco inaczej.

Jeśli w bloku *if( ){ }* znajduje się dokładnie 1 instrukcja, to możemy pominąć nawiasy klamrowe. Analogicznie w przypadku bloku *else{ }*.

Przykłady

```c
#include<stdio.h>
int main()
{
    int x = 4;
    if (x == 5)
        printf("Tak");
    else
        printf("Nie");
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int x;
    printf("Wprowadz liczbe calkowita:\n");
    scanf("%d", &x);
    if (x > 5)
        printf("x>5");
    else
        printf("x<=5");
    return 0;
}
```

ZADANIE

Przetestuj działanie powyższych kodów programów.

Instrukcje *if* i *if else* można zagnieżdżać.

Przykłady

```c
#include<stdio.h>
int main()
{
    int x;
    printf("Program sprawdza czy podana liczba jest dodatnia czy ujemna\n");
    printf("Wprowadz liczbe calkowita:\n");
    scanf("%d", &x);
    if (x < 0)
    {
        printf("%d < 0", x);
    }
    else
    {
        if (x > 0)
            printf("%d > 0", x);
        else
            printf("x = 0");
    }
    return 0;
}
```

ZADANIE

Przetestuj działanie powyższego przykładu.

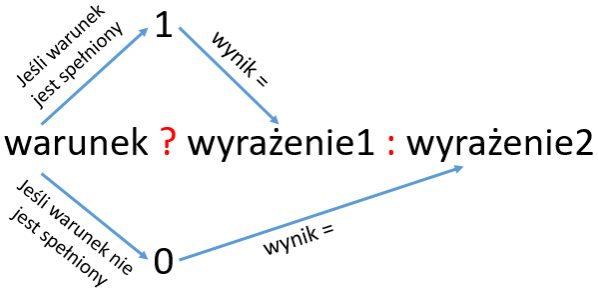
Operator warunkowy *? :*

Operator warunkowy to konstrukcja będąca formą instrukcji warunkowej wyrażona za pomocą operatora trójargumentowego. Umożliwia ona sprawdzenie warunku na poziomie wyrażenia, co w pewnym stopniu zaciera rozróżnienie między wyrażeniami a instrukcjami, dzięki czemu przy jej rozsądnym używaniu kod źródłowy może zyskać na zwięzłości i prostocie.

Postać ogólna operatora warunkowego jest następująca:

```
warunek ? wyrażenie1 : wyrażenie2
```

Zaś zasadę działania obrazuje rysunek:



Przykłady

```c
#include<stdio.h>
int main()
{
    int m, n, max;
    printf("Podaj m: ");
    scanf("%d", &m);
    printf("Podaj n: ");
    scanf("%d", &n);
    max = m < n ? n : m; // max = ( (m < n ) ? n : m );
    printf("max = %d\n", max);
    return 0 ;
}
```

```c
#include<stdio.h>
int main()
{
    int m, n, max;
    printf("Podaj m: ");
    scanf("%d", &m);
    printf("Podaj n: ");
    scanf("%d", &n);
    if(m<n)
        max = n;
    else
        max = m;
    printf("max = %d\n", max);
    return 0 ;
}
```

ZADANIE

Sprawdź działanie powyższych przykładów.

Operatory warunkowe również mogą być zagnieżdżane.

Przykład

```c
#include<stdio.h>
int main()
{
    int m, n, max;
    printf ("Podaj m: ");
    scanf ("%d", &m);
    printf ("Podaj n: ");
    scanf ("%d", &n);
    m < n ? printf("max = n = %d\n", n) : m > n ? printf("max = m = %d\n", m) : printf("max = m = n = %d\n", m);
//  (m<n) ? printf("max = n = %d\n", n) : ( (m>n) ? printf("max = m = %d\n", m) : printf("max = m = n = %d\n", m) );
    return 0 ;
}
```

Przeanalizujmy linię, w której wykorzystane zostały operatory warunkowe. W powyższym przykładzie w pierwszej kolejności sprawdzany jest warunek: `m < n`. Jeśli będzie on prawdziwy, to wykona się instrukcja: `printf("max = n = %d\n", n)`, w przeciwnym przypadku sprawdzany będzie kolejny warunek: `m > n`. Jeśli dla niego otrzymamy wynik 'prawda', to wykona się instrukcja: `printf("max = m = %d\n", m)`. Jeśli natomiast obydwa sprawdzane warunki będą fałszywe, to wykonana zostanie instrukcja: `printf("max = m = n = %d\n", m);`

ZADANIE

Przetestuj działanie powyższego przykładu. Zastąp linię z operatorami warunkowymi odpowiednimi instrukcjami warunkowymi *if* i/lub *if else*.

Instrukcja *switch*

Instrukcja **switch** pozwala na sprawdzenie wielu warunków w zależności od wartości zmiennej. Tym samym umożliwia ograniczenie wielokrotnego stosowania instrukcji if. Postać ogólna instrukcji switch wygląda następująco:

```
switch (wyrażenie) {
  case wartość1: /* instrukcje, jeśli wyrażenie == wartość1 */
    break;
  case wartość2: /* instrukcje, jeśli wyrażenie == wartość2 */
    break;
  /* ... */
  default: /* instrukcje, jeśli żaden z wcześniejszych warunków nie został spełniony */
    break;
}
```

Przykład

```
#include<stdio.h>
int main()
{
    int m;
    printf("Podaj numer opcji: ");
    scanf("%d", &m);
    switch(m)
    {
        case 1:
            printf("Wybrano 1\n");
            break;
        case 2:
            printf("Opcja\n");
            printf("druga\n");
            break;
        case 7: printf("Wybrano 7\n");
            break;
        case 3:
            printf("Opcja 3\n");
            break;
        default:
            printf("Opcja defaultowa\n");
            break;
    }
    return 0 ;
}
```

ZADANIE

Przetestuj działanie powyższego przykładu. Dodaj jeszcze kilka innych case'ów.

Przy wykorzystywaniu switcha należy pamiętać o użyciu instrukcji zatrzymania (przerwania) **break** po zakończeniu listy instrukcji następujących po **case**. Jeśli tego nie zrobimy, program przejdzie do wykonywania instrukcji z następnego case. Może mieć to fatalne skutki.

Przykład

```c
#include <stdio.h>
int main ()
{
    int a, b;
    printf("Podaj a: ");
    scanf("%d", &a);
    printf("Podaj b: ");
    scanf("%d", &b);
    switch(b)
    {
        case  0:
            printf ("Nie można dzielić przez 0!\n");
            /* tutaj zabrakło break! */
        default:
            printf ("a/b=%d\n", a/b);
    }
    return 0;
}
```

ZADANIE

Przetestuj działanie powyższego przykładu. Czy działa on prawidłowo? Co się stanie jeśli jako b podamy wartość 0? Czy program zadziała prawidłowo?

Podczas używania switcha czasami chcemy, aby dla kilku przypadków (case'ów) wykonywane były te same instrukcje. Np. dla wartości 1 i 2 ma zostać wypisane „OK", a dla pozostałych przypadków: „ERROR". Możemy to rozwiązać następująco:

```c
#include <stdio.h>
int main ()
{
    int a;
    printf("Podaj a: ");
    scanf("%d", &a);
    switch(a)
    {
        case  1:
            printf ("OK\n");
            break;
        case  2:
            printf ("OK\n");
            break;
        default:
            printf ("ERROR");
    }
    return 0;
}
```

lub krócej:

```c
#include <stdio.h>
int main ()
{
    int a;
    printf("Podaj a: ");
    scanf("%d", &a);
    switch(a)
    {
        case  1:
        case  2:
            printf ("OK\n");
            break;
        default:
            printf ("ERROR");
    }
    return 0;
}
```

ZADANIE

Przetestuj działanie dwóch powyższych kodów programów.

Zadania

1. Napisz program, który będzie pobierał od użytkownika numer miesiąca i wypisywał na konsolę pełną jego nazwę, np. dla numeru *11* otrzymamy wynik *listopad*. Program powinien obsługiwać sytuacje związane z błędnym numerem miesiąca. Zadanie rozwiąż z użyciem instrukcji warunkowych.
2. Napisz program, który będzie pobierał od użytkownika numer miesiąca i wypisywał na konsolę pełną jego nazwę, np. dla numeru *11* otrzymamy wynik *listopad*. Program powinien obsługiwać sytuacje związane z błędnym numerem miesiąca. Zadanie rozwiąż z użyciem operatorów warunkowych.
3. Napisz program, który będzie pobierał od użytkownika numer miesiąca i wypisywał na konsolę pełną jego nazwę, np. dla numeru *11* otrzymamy wynik *listopad*. Program powinien obsługiwać sytuacje związane z błędnym numerem miesiąca. Zadanie rozwiąż z użyciem instrukcji switch.
4. Napisz program, który będzie kalkulatorem. Aplikacja ma pobierać od użytkownika dwie liczby rzeczywiste oraz znak działania (+, -, *, /) i wypisywać na konsolę wynik w postaci: *liczba1 znak_działania liczba2 = wynik*. Program powinien obsługiwać sytuacje związane z błędnymi danymi – sytuacja niepoprawna: dzielenie przez 0. Zadanie rozwiąż z użyciem instrukcji *if* i/lub *if else*.
5. Napisz program, który będzie kalkulatorem. Aplikacja ma pobierać od użytkownika dwie liczby rzeczywiste oraz znak działania (+, -, *, /) i wypisywać na konsolę wynik w postaci: *liczba1 znak_działania liczba2 = wynik*. Program powinien obsługiwać sytuacje związane z błędnymi danymi – sytuacja niepoprawna: dzielenie przez 0. Zadanie rozwiąż z użyciem instrukcji switch.
6. Napisz program obliczający wysokość należnego podatku uwzględniając ulgę na dzieci. Użytkownik wprowadza liczbę posiadanych dzieci oraz wysokość podatku (bez ulgi), zaś program powinien wyświetlać wysokość podatku z ulgami.
   Ulgi obliczamy w następujący sposób:

Liczba dzieci	Ulga podatkowa
0	brak
1	2%
2	5%
3	8%
4 i więcej	15%

   Zadanie rozwiąż wykorzystując instrukcję switch.