

Sterowanie, operacje i operatory logiczne

Sterowanie, operacje i operatory logiczne

W rozdziale tym Czytelnik będzie miał okazję zapoznać się z operacjami sterowania. Należą do nich różnego rodzaju pętle i warunki, które złożone w odpowiednią całość stanowią o istocie działania kodu programu. Dzięki wykorzystaniu relacji, jakie zachodzą pomiędzy różnymi zbiorami komputer przy wykorzystaniu odpowiednich warunków logicznych jest w stanie rozstrzygać różnego rodzaju problemy. Przedstawione w tym rozdziale wiadomości pomogą w zrozumieniu mechanizmów, jakie kryją się w środku programów wspomagających podejmowanie decyzji.

Relacje i operatory logiczne

Na początku pragniemy przedstawić Czytelnikowi podstawowe relacje logiczne, jakie są najczęściej wykorzystywane przez programistów w rozwiązywaniu zadań. Ważnym jest, aby zapamiętać, iż w języku ANSI C nie istnieje typ logiczny danych. Oznacza to, że należy się posługiwać językiem numerycznym, który będzie w stanie zrozumiale opisać temat jednocześnie dla kompilatora i programisty. W języku ANSI C wynik operacji logicznej przechowywany jest w zmiennych typu `integer`. Oznacza to, że logiczna operacja będzie miała zapis tak/nie, przedstawiony w postaci bitowej. Jeżeli wynik operacji jest równy zero to jest przyjmowany za *false*, czyli fałsz. W przeciwnym wypadku wynik operacji jest przyjmowany za *true*, czyli prawdę. Możemy, zatem przyjąć dla uproszczenia, że wartościom prawdy i fałszu odpowiadają bitowe wartości typu `int` 1 i 0. W późniejszych wersjach języka ANSI C jak C++ wprowadzono typ *bool* przechowujący wartości *true* i *false*. Niemniej jednak zastosowanie typu *bool* jest ograniczone ze względu na brak standardu zapisu tej zmiennej w pamięci komputera. W konsekwencji operacja taka może prowadzić do braku możliwości przeniesienia programów pomiędzy różnymi systemami operacyjnymi. Zastanówmy się nad praktycznym zastosowaniem operacji logicznych w kodzie programu.

Przykład 4

Podajmy prosty przykład programu, który będzie w stanie sprawdzić czy wczytany znak z klawiatury jest małą literą.

<pre>#include<stdio.h> int main() { char c; int is; printf("Podaj znak:"); scanf("%c",&c); is= 'a' <= c && c <= 'z'; printf("Litera =%d\n",is); _getch(); return 0; }</pre>	<pre>// deklaracja zmiennych // pobranie zmiennych /* sprawdzenie czy wczytany znak jest małą literą */</pre>
--	---

Zgodnie z omawianą zasadą zapisu zmiennych logicznych w postaci numerycznej pokazany program wypisze 0 albo 1 w zależności od przeczytanego znaku. Interpreter programu wypisze 1 w przypadku przeczytania małej litery angielskiej albo 0 w innym przypadku. Czytelnik zwróci uwagę, że w kodzie programu użyto operatora logicznego `<=`. Jego zastosowanie umożliwia sprawdzenie czy kod pobranego znaku w ASCII jest większy lub równy literze 'a' oraz mniejszy lub równy literze 'z'. Do porównywania wartości kodu ASCII tych znaków używamy następujących operatorów logicznych: `<`, `<=`, `=`, `!=`, `>`, `>=`. Wśród najważniejszych operatorów logicznych znajdują się np. operator równości reprezentowany przez dwa znaki równości, co oznacza relację równości dwóch liczb. Natomiast relację różności dwóch liczb zapisujemy za pomocą wykrzyknika i równości. Działanie operatorów logicznych zostało przedstawione w poniższych tabelach.

Operatory logiczne

Negacja	
!=	
0	1
1	0

Koniunkcja		
&&	0	1
0	0	0
1	0	1

Alternatywa		
	0	1
0	0	1
1	1	1

Instrukcja warunkowa if()

W języku programowania dla kontroli wyników operacji wprowadzane są instrukcje warunkowe. Instrukcja warunkowa jest wykonywana, jeżeli wyrażenie logiczne przyjmuje wartość *true* w przeciwnym wypadku sterowanie zostaje przekazane do instrukcji umieszczonej po słowie kluczowym. W przypadku instrukcji warunkowej *if()* możemy budować kilkakrotnie złożone warunki oddzielone słowem kluczowym *else*. Postać instrukcji warunkowej *if()* jest następująca:

```
if(wyrażenie_0)
    instrukcja_0;
else if(wyrażenie_1)
    instrukcja_1;
else if(wyrażenie_2)
    instrukcja_2;
    .....
else
    instrukcja_n;
```

Instrukcja warunkowa może składać się z kilku warunków zapisanych w postaci różnych wyrażeń. Oznacza to, że komputer jest w stanie w sposób precyzyjny rozróżniać elementy badanej przestrzeni. Ostatnia z zadanych *instrukcja_n* zostanie wykonana, jeżeli żaden z warunków poprzedzających nie zostanie spełniony. Istnieje również możliwość uproszczonego zapisu warunku tylko dla jednego przypadku. Konstrukcja taka *if(wyrażenie) instrukcja*; nie musi wtedy zawierać słowa kluczowego *else*. Oznacza to, że wówczas w kompilatorze nie będzie rozpatrywany przypadek warunku przeciwnego. Poprawne stosowanie instrukcji warunkowych jest podstawą budowy efektywnych algorytmów. W kolejnych rozdziałach Czytelnik będzie mógł zobaczyć wiele przykładów multiwarunków, których poprawność działania zależy od odpowiedniego stosowania instrukcji warunkowej *if()*. Więcej przykładów można znaleźć w literaturze [5, 21, 28, 31, 33, 40, 43, 56, 63, 90, 91, 99]. Spróbujmy teraz prześledzić na przykładzie zasadę budowy i działania instrukcji warunkowej.

Przykład 5

Napisać program, który znajduje maksymalną liczbę z dwóch przeczytanych liczb.

<pre>#include<stdio.h> int main() { double a,b,max ; printf("Podaj a b:"); scanf("%lf %lf",&a,&b); if(a>b)max=a; else max=b; printf("max=%g\n",max); _getch(); return 0; }</pre>	<pre>// deklaracja biblioteki // deklaracja zmiennych // pobranie zmiennych /*instrukcja warunkowa, która poszukuje wartości maksymalnej spośród dwóch wczytanych liczb */</pre>
---	---

Uwaga: Przy wprowadzaniu liczb oddzielamy je spacją albo zakańczamy wpisywanie liczby klawiszem *Enter*. Oba sposoby wprowadzania są jednakowo rozróżniane przez konsolę obsługującą skompilowaną wersję kodu. Czytelnik zechce porównać opisaną powyżej instrukcję warunkową z jej krótszą wersją. Oba zastosowania instrukcji warunkowej są równoważne.

Instrukcja w postaci pełnego warunku If	Instrukcja w postaci skróconej
<code>if(a>b) max=a; else max=b;</code>	<code>max = (a>b) ? a : b;</code>

Postaramy się teraz omówić zastosowanie instrukcji warunkowych w przypadku złożonego warunku rozstrzygającego.

Przykład 6

Napisać program znajdujący rozwiązanie układu równań liniowych postaci

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

, gdzie współczynniki a, b, c, d, e, f są zadanymi liczbami rzeczywistymi. Natomiast zmienne x i y są szukаныmi niewiadomymi.

Do rozwiązania tego przykładu zastosujemy metodę wyznaczników znaną Czytelnikowi z kursu algebry. Metodę tę dla przypomnienia zdefiniujemy raz jeszcze. Na początku wyznaczamy następujące wyznaczniki układu równań liniowych.

$$W = \begin{vmatrix} a & b \\ d & e \end{vmatrix} = a \cdot e - b \cdot d$$

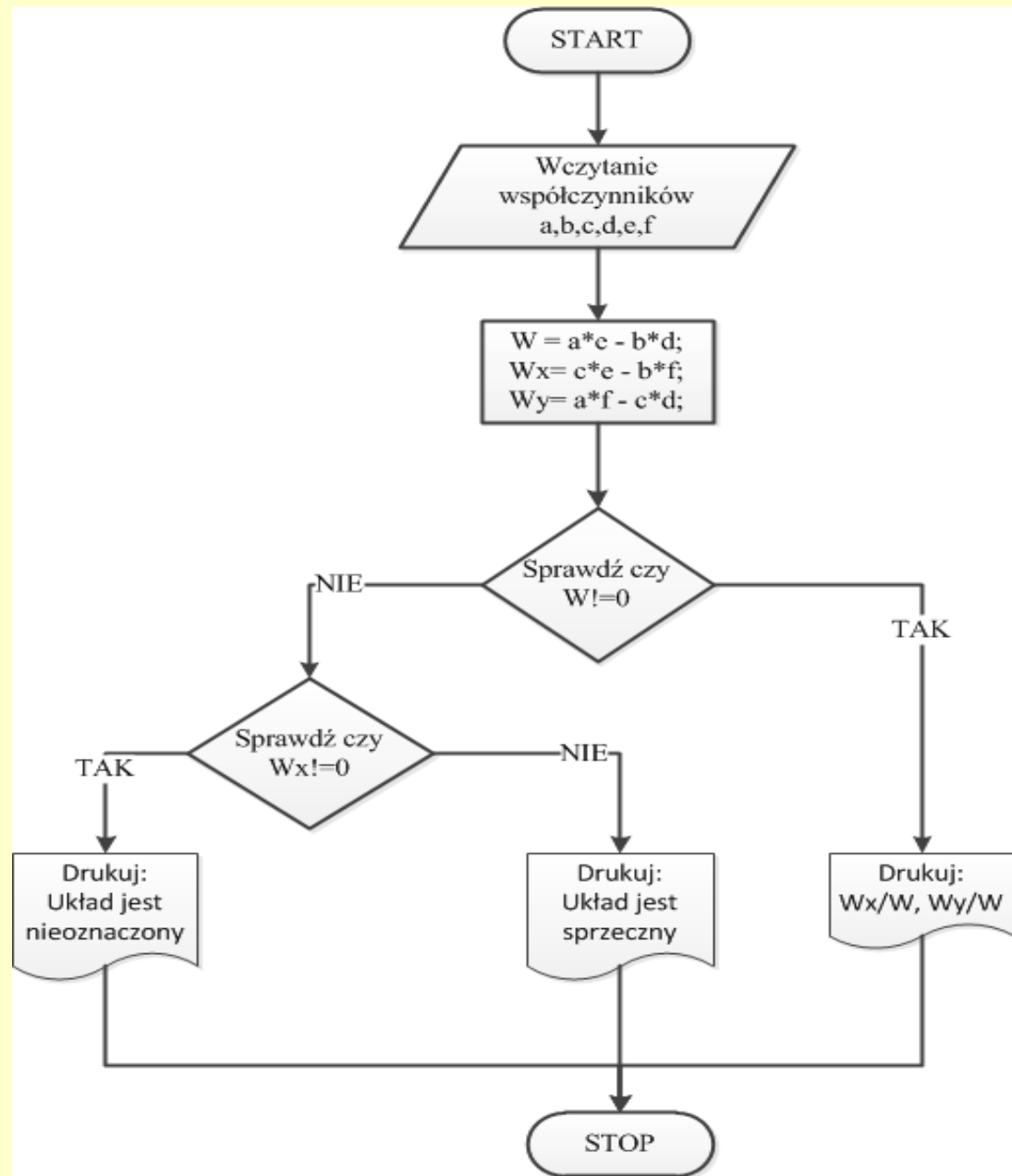
$$W_x = \begin{vmatrix} c & b \\ f & e \end{vmatrix} = c \cdot e - b \cdot f$$

$$W_y = \begin{vmatrix} a & c \\ d & f \end{vmatrix} = a \cdot f - c \cdot d$$

Mówimy wtedy o następujących możliwościach:

- 1) Jeżeli $W \neq 0$ to układ reprezentowany wyznacznikami układ równań liniowych jest oznaczony oraz istnieje jedna i tylko jedna para liczb $x = \frac{W_x}{W}$ oraz $y = \frac{W_y}{W}$ spełniająca ten układ równań,
- 2) Jeżeli $W = 0$ oraz $W_x \neq 0$, co pociąga za sobą $W_y \neq 0$, to układ reprezentowany wyznacznikami jest sprzeczny oraz żadna para liczb nie spełnia tego układu równań,
- 3) Jeżeli $W = 0$ oraz $W_x = 0$, co pociąga za sobą $W_y = 0$, to układ reprezentowany wyznacznikami jest nieoznaczony oraz istnieje nieskończenie wiele par liczb spełniających ten układ równań.

Zapiszmy
teraz
przedstawio
ną metodę
w postaci
schematu
blokowego,
a następnie
na jego
podstawie
w postaci
kodu
programu



```

#include<stdio.h>
int main()
{
    double a,b,c,d,e,f,W,Wx,Wy;

    printf("Podaj a b c:");
    scanf("%lf %lf %lf",&a,&b,&c);
    printf("Podaj d e f:");
    scanf("%lf %lf %lf",&d,&e,&f);

    W = a*e - b*d;
    Wx= c*e - b*f;
    Wy= a*f - c*d;

    if(W != 0)
    {
        printf("x1=%g\n",Wx/W);
        printf("x2=%g\n",Wy/W);
    }
    else if(Wx != 0)
        printf("Uklad rownan jest sprzeczny\n");
    else
        printf("Uklad rownan jest nieoznaczony\n");

    _getch();
    return 0;
}

```

```

// Deklaracja biblioteki

// Deklaracja zmiennych

/*      Wczytanie      zmiennych,      czyli
współczynników układu równań */

/*      Wyznaczenie      wyznaczników      układu
równań (4.2.1) zgodnie z przedstawionymi
wzorami */
/*      Instrukcja warunkowa if sprawdzająca
zachodzenie możliwości 1) */

// sprawdzenie możliwości 2)
/*      Ponieważ nie zaszła żadna z poprzednich
możliwości program zdecyduje o
zachodzeniu możliwości 3) */

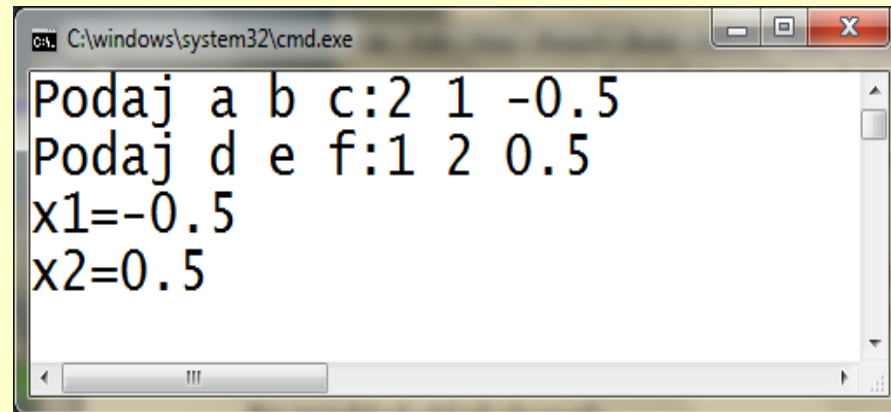
```

Programy pisane dla rozwiązywania różnych zagadnień i problemów najczęściej działają poprzez realizację procedur zapisanych w pewnych algorytmach czy schematach. Czytelnik zechce samodzielnie przeanalizować jeszcze raz przytoczony przykład i porównać go z wiedzą, jaką zaczerpnął z rozdziału poprzedniego.

Zobaczmy jak nasz program będzie się zachowywał w przypadku wprowadzenia różnych układów równań.

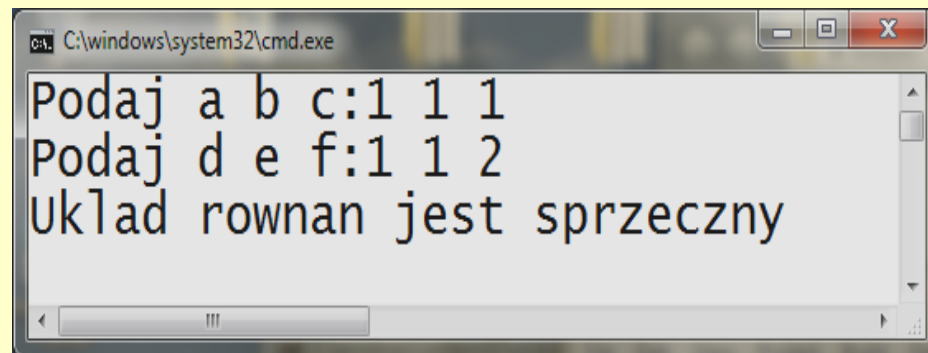
Na przykład układ równań
$$\begin{cases} 2x + y = -0.5 \\ x + 2y = 0.5 \end{cases}$$

ma rozwiązanie $x = -0.5$ oraz $y = 0.5$, co przedstawia poniższa ilustracja.



```
C:\windows\system32\cmd.exe
Podaj a b c:2 1 -0.5
Podaj d e f:1 2 0.5
x1=-0.5
x2=0.5
```

Natomiast układ równań $\begin{cases} x + y = 1 \\ x + 2y = 2 \end{cases}$ jest sprzeczny.



```
C:\windows\system32\cmd.exe
Podaj a b c:1 1 1
Podaj d e f:1 1 2
Układ rownan jest sprzeczny
```


Operatory bitowe

Język programowania, jako działający na bitach, oferuje różnego rodzaju działania bitowe. Zadaniem takich działań jest sformułowanie zasad operowania na informacjach zapisanych w postaci kodu bitów. Język C/C++ oferuje operatory pozwalające dotrzeć do pojedynczego bitu zmiennych typu *char* i *int*, a następnie wykonać żadaną operację. Mamy do dyspozycji następujące operatory:

Dopełnienie jedynkowe (operator jednoargumentowy)		
~	0	1
	1	0

Alternatywa bitowa (OR)		
	0	1
0	0	1
1	1	1

Koniunkcja bitowa (AND)		
&	0	1
0	0	0
1	0	1

Bitowa różnica symetryczna (XOR)		
^	0	1
0	0	1
1	1	0

Prześledźmy teraz jak będzie wyglądało działanie wykonane na podstawie opisanych operatorów bitowych. Załóżmy, że mamy do wykonania operacje bitowe na dwóch liczbach. Niech będą dane liczby 5 oraz 3. Komputer wykonując operacje bitowe będzie postępował następująco:

5 -> 0101

3 -> 0011

Najpierw nastąpi zamiana wartości
na ich zapis w systemie dwójkowym

Zobaczmy teraz wykonanie operacji bitowych

	Symbol	Wynik dwójkowo	Wynik dziesiętnie
Dopełnienie nie a	$\sim a$	1010	10
Dopełnienie nie b	$\sim b$	1100	12
AND	$a \& b$	0001	1
OR	$a b$	0111	7
XOR	$a \wedge b$	0110	6

Przykład 7

Napisać program wykonujący działanie przesunięcia dla char c1=0x4f.

<pre>#include<stdio.h> int main() { printf("dziesiętnie szesnastkowo\n"); printf("-----\n"); printf("%11d %x\n",c1,c1); c1=c1<<1; printf("%11d %x\n",c1,c1); c1=c1<<1; printf("%11d %x\n",c1,c1); c1=c1>>1; printf("%11d %x\n",c1,c1); _getch(); return 0; }</pre>	<pre>// Deklaracja biblioteki /* Wypisanie na ekran działania wykonanego przez operatory bitowe przesunięcia */ /* Wydruk poszczególnych wyników operacji przesunięcia bitowego */</pre>
---	--

A screenshot of a Windows command prompt window titled "C:\windows\system32\cmd.exe". Inside the window, there is a table with two columns: "dziesietnie" (decimal) and "szesnastkowo" (hexadecimal). The table is separated by a dashed line. The data rows are as follows:

dziesietnie	szesnastkowo
79	4f
-98	ffffff9e
60	3c
30	1e

W informatyce oprócz zapisu dziesiętnego i dwójkowego zastosowanie mają również inne systemy liczbowe. Każdą liczbę dziesiętną można zapisać w systemie szesnastkowym w następującej postaci:

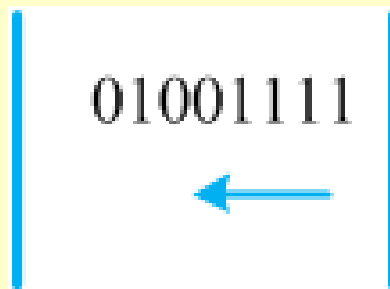
$$a_n \cdot 16^n + a_{n-1} \cdot 16^{n-1} + a_{n-2} \cdot 16^{n-2} + \dots + a_1 \cdot 16^1 + a_0 =$$

$$= \left(\dots \left((a_n \cdot 16 + a_{n-1}) \cdot 16 + a_{n-2} \right) \cdot 16 + \dots + \right) \cdot 16 + a_0$$

Liczby a_0, a_1, \dots, a_n są cyframi szesnastkowymi, co przedstawiono w poniższej tabeli:

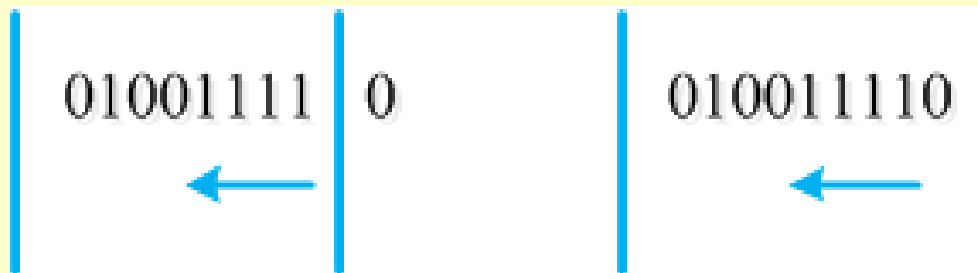
Dziesiętnie	Szesnastkowo	Binarnie
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111

Nietrudno zauważyć, że jedna liczba w układzie szesnastkowym jest zapisana dokładnie na czterech bitach. Jeden bajt jest reprezentowany jednoznacznie za pomocą dwóch cyfr szesnastkowych. Łatwo też zamienić zapisany bajt 0x4f na liczbę $4 \cdot 16 + 15 = 79$ dziesiętną. W takim zapisie 0x oznacza, że do zapisu liczby używamy małych liter. Gdybyśmy chcieli użyć dużych liter od A do F przed liczbą napisalibyśmy przedrostek 0X. W ten sposób zapisywane są liczby dodatnie. Liczby ujemne są zapisywane w kodzie uzupełnienia do 2.



01001111

Liczba 79 zapisana w systemie dwójkowym



Liczba 79 dwójkowo po przesunięciu o jeden to liczba 9e szesnastkowo.

W wyniku przesunięcia otrzymujemy liczbę 9e, która ma 1 na najbardziej znaczącej pozycji. Jest to liczba ujemna zapisana w kodzie uzupełnienia do liczby 2^8 . Aby znaleźć jej lustrzane odbicie tzn. liczbę dodatnią bez znaku dokonamy odejmowania $2^8 - 9e$. Otrzymany wynik łatwo można zinterpretować $6 \cdot 16 + 2 = 79$.

	100000000	2^8
-	10011110	9e
<hr/>		
	01100010	62

Odejmowanie liczb wykonane w systemie szesnastkowym.

Tak, więc otrzymaliśmy w wyniku przesunięcia o jeden bit, liczbę ujemną -98. Na przykładzie uwidoczniona jest również zamiana liczby typu *char* do liczby typu *int* przez powielenie najstarszego bitu 1 na trzech dodanych bajtach (ffffff9e). Ten sam wynik możemy otrzymać w sposób równoważny w systemie dwójkowym wykonując następujące działanie:

$$-2^7 + 2^4 + 2^3 + 2^2 + 2 = -98$$

Na komputerach wszystkie liczby całkowite są przechowywane w arytmetyce uzupełnieniowej ze względu na prostotę zapisu. Następnie w programie przesuwamy liczbę o jeden bit w lewo, na pozycję najbardziej znaczącą wpisywane jest zero i otrzymujemy liczbę: . Następnie przesuwamy o jeden bit w prawo i tracimy tym samym informację z przed poprzedniego przesunięcia, gdyż na pozycję najbardziej znaczącą wpisywane jest zero oraz otrzymujemy wynik: . Aby zapisać liczbę dziesiętną jako liczbę w systemie szesnastkowym należy podzielić ją przez 16. Kolejne wyniki dzielenia zapisujemy w postaci przedstawionego uprzednio sposobu zapisu liczby szesnastkowej.

$$\begin{aligned} 1867 &= 116 \cdot 16 + 11 = (7 \cdot 16 + 4) \cdot 16 + 11 = \\ &= 7 \cdot 16^2 + 4 \cdot 16 + 11 = 74b \end{aligned}$$

Istnieje również możliwość wygodnego zamieniania liczb pomiędzy różnymi systemami liczbowymi. Umieszczony kalkulator programisty pod systemem operacyjnym umożliwia dokonanie takiej konwersji w sposób bardzo prosty. Czytelnik zechce sprawdzić opisaną możliwość.

Postaramy się teraz przedstawić zastosowanie operatorów bitowych. Operatora bitowego XOR używamy często do szyfrowania informacji. Procedura szyfrowania polega na ustaleniu klucza, który będzie znany tylko osobie szyfrującej i osobie odczytującej informację. W naszym przypadku, ponieważ będziemy szyfrowali tylko jeden znak angielski, użyjemy klucza 0xc5. Czytelnik zechce prześledzić przedstawiony przykład.

Przykład 8

Napisać program szyfrujący z kluczem 0xc5.

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int c='c',c2=0xc5;
```

```
    printf("%x - kod litery 'c'\n",c);
```

```
    c^=c2;
```

```
    printf
```

```
        ("%x - litera po zakodowaniu kluczem 0xc5\n",c);
```

```
    _getch();
```

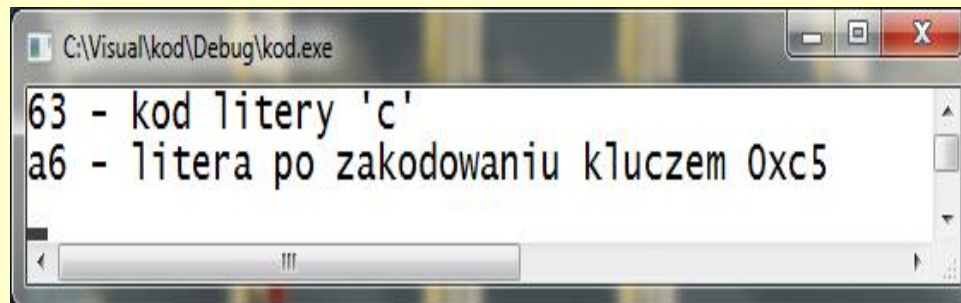
```
    return 0;
```

```
}
```

```
// Deklaracja biblioteki
```

```
/* Wypisanie na ekran  
działania wykonanego  
przez operatory bitowe  
przesunięcia. */
```

```
/*                               Wydruk  
poszczególnych  
wyników operacji  
przesunięcia bitowego.  
*/
```



```
C:\Visual\kod\Debug\kod.exe
63 - kod litery 'c'
a6 - litera po zakodowaniu kluczem 0xc5
```

Aby zrozumieć zasadę działania Czytelnik zechce spróbować teraz ręcznie odszyfrować informację zakodowaną w naszym programie używając do tego podanego klucza.

	10100110	a6
^	11000101	c5
<hr/>		
	01100011	63

Działanie kodowania liczb z kluczem 0xc5.

Widzimy, że klucz kodujący i odkodowujący informacje jest taki sam, dlatego istotny jest sposób przekazywania klucza tak, aby nie dostał się w niepowołane ręce.

Etykiety i instrukcja skoku

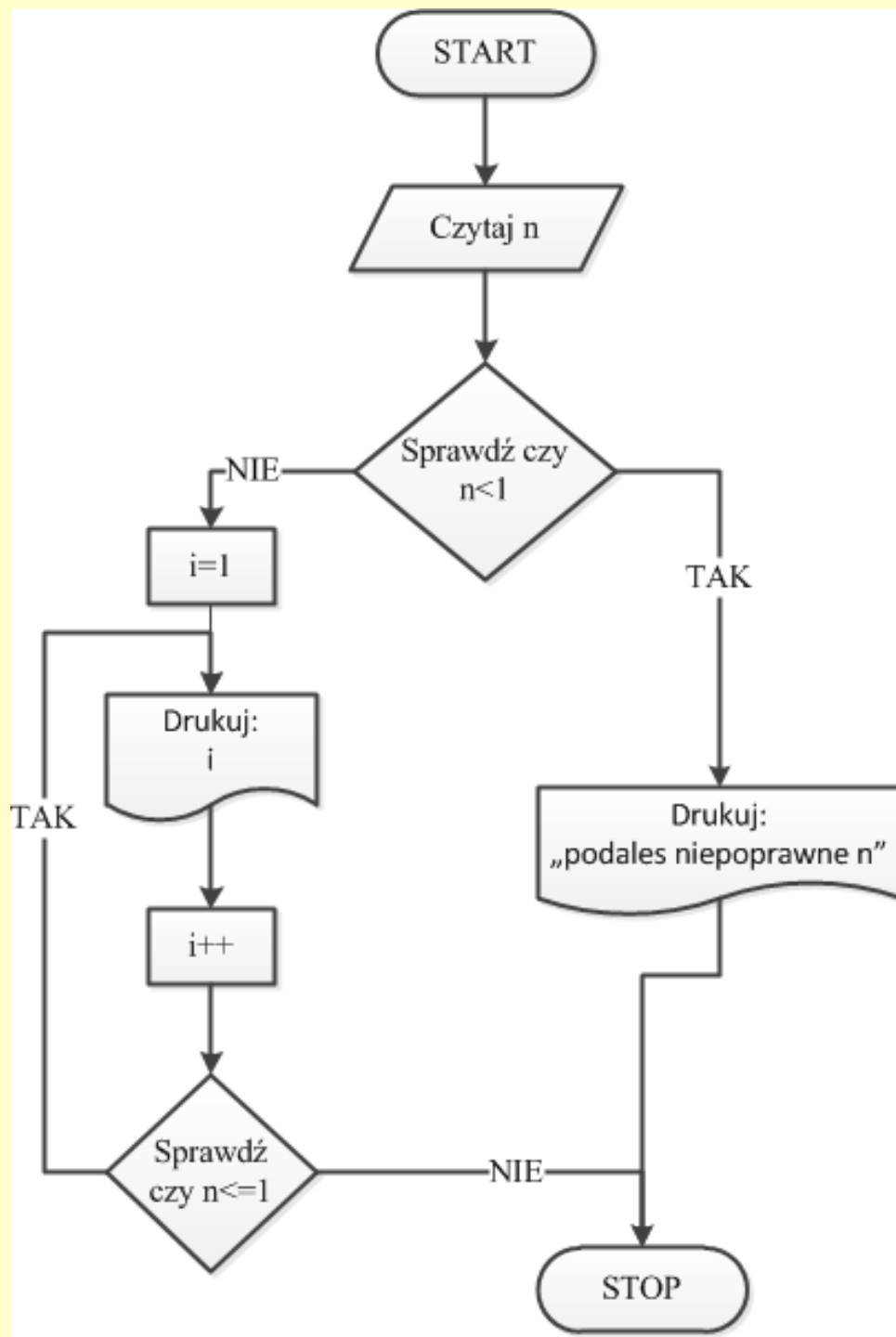
Etykietę definiujemy jako ciąg liter i cyfr zaczynających się od litery i zakończony znakiem ':'. Instrukcja skoku *goto* i etykieta wskazuje na miejsce gdzie zostanie przekazane sterowanie. Formalnie instrukcja skoku nie jest potrzebna i zawsze można napisać program bez jej użycia. Niemniej została zamieszczona w języku C/C++, dlatego jej użycie możemy zilustrować na następującym przykładzie.

Przykład 9

Napisać program drukujący liczby 1,2,...,n wykorzystując etykietę skoku.

Zapisany poniżej kod programu dla lepszego zrozumienia posiada również schemat blokowy, który został przytoczony poniżej. Czytelnik zechce przeanalizować zapisane w nim procedury.

Schemat
blokowy
realizujący
zadanie
opisane w
programie
przykład 9.



```
#include<stdio.h>
int main()
{
    int i,n;
    printf("Podaj n:");
    scanf("%d",&n);

    if(n<1)
    {
        printf("Podales niepoprawne n\n");
        return 0;
    }
    i=1;
et: printf("%d\n",i);
    i++;
    if(i<=n)
        goto et;

    _getch();
    return 0;
}
```

```
// Deklaracja biblioteki
```

```
/* Pobranie zmiennej ilości
znaków do wypisania */
```

```
// Sprawdzenie warunku
/* Ponieważ ilość liczb musi być
większa od 0 program wykonuje
analizę pobranej zmiennej w bloku
programowym. */
```

```
/* W przypadku, gdy n jest większe
od 0, można coś wydrukować i
następuje wykonanie operacji
drukowania. */
```


Przedstawiony program mógłby zawierać dwie instrukcje stopu. Nie jest to przyjęte jako dobra zasada programowania. Czytelny i jednoznaczny odsyłaniem jest użycie jednego końcowego stopu na samym końcu kodu programu.

Podobnie użycie instrukcji *goto* nie jest zgodne z przedstawianymi zasadami programowania strukturalnego i może prowadzić do błędów programowych. W programowaniu najlepiej unikać skoków bezwarunkowych przez zastępowanie ich odpowiednimi blokami programowymi. Blok instrukcji zawsze rozpoczynamy nawiasem klamrowym { i kończąc zamykamy nawiasem klamrowym }. Czytelnik zechce prześledzić takie postępowanie w kodzie Przykład 9.

Uwaga: Czytelnik zapewne już zauważył sposób komentowania w kodzie, jednak dla lepszego przyswojenia wiedzy opiszemy dokładnie tę procedurę. Komentarze w kodzie programu, czyli tekst, który nie jest brany pod uwagę przez kompilator rozpoczynamy znakami `/*` i kończymy znakami `*/`. Natomiast pojedynczą linię wyłączamy z kodu programu za pomocą dwóch ukośników `///`. Korzystając z komentowania często w łatwy sposób możemy znaleźć błędy w kodzie programu.