Operatory relacyjne

W tabeli 3.13 podano operatory relacyjne, z których można korzystać w języku C++.

Tabela 3.13. Zestawienie operatorów relacyjnych

Operator	Relacja
<	Mniejsze
<=	Mniejsze lub równe
==	Równe
!=	Różne (nieprawda, że równe)
>=	Większe lub równe
>	Większe

Operatory logiczne

Wartość wyrażenia logicznego, w którym korzystamy z operatorów logicznych (patrz tabela 3.14), jest zawsze typu całkowitego.

Tabela 3.14. Zestawienie operatorów logicznych

Operator	Działanie
!	Negacja
&&	Koniunkcja (iloczyn logiczny)
||	Alternatywa (suma logiczna)

W języku C++ dowolna niezerowa wartość interpretowana jest jako prawda (czyli `true`), natomiast wartość 0 jako fałsz (czyli `false`).

Przykład 3.12.

Przeanalizujmy następujący przykład (*zad3_8.cpp*):

```
int m=8, n=0, wynik;
wynik=m||n;
cout<<wynik<<"\t";
wynik=m&&n;
cout<<wynik<<"\t";
wynik=!m;
cout<<wynik<<"\t";
wynik=!n;
cout<<wynik<<endl;
```

Po wykonaniu podanego kodu wypisane zostaną następujące wartości:

1 0 0 1

Wynika to stąd, że m równe 8 interpretowane jest jako prawda, natomiast n równe 0 jako fałsz. Jeśli wynikiem wykonanej operacji jest prawda, to wypisywana jest wartość 1, natomiast jeśli fałsz — wartość 0.

Przykład 3.13.

Kolejny przykład pokazuje zastosowanie operatora logicznego `&&` w konstruowaniu warunku w instrukcji warunkowej `if` (*prog3_9.cpp*):

```
int m=-3, n=-7;
if (m<0&&n<0) cout<<"liczby są ujemne"<<endl;
else cout<<"co najmniej jedna liczba nie jest ujemna"<<endl;
```

Po uruchomieniu programu wypisany zostanie następujący komunikat:

liczby są ujemne

Liczby m i n są ujemne, spełnione są więc obydwa warunki, dlatego pojawia się ten komunikat. Gdyby chociaż jedna z tych liczb była nieujemna, wyrażenie byłoby fałszywe i wypisany zostałby komunikat „co najmniej jedna liczba nie jest ujemna".

Złożone operatory przypisania

Złożone operatory przypisania stosuje się do zapisywania wyrażeń `X = X · Y` w postaci `X ·= Y`, gdzie · to operator dwuargumentowy. Istnieje dziesięć złożonych operatorów przypisania, z których do najważniejszych należą:

`*=  /=  %=  +=  -=`

Przykład 3.14.

W tabeli 3.15 podano przykłady zastosowania złożonych operatorów przypisania w konstruowaniu wyrażeń.

Tabela 3.15. Przykłady zastosowania złożonych operatorów przypisania

Wyrażenie	Zapis wyrażenia bez złożonego operatora przypisania
`a *= 4*b-1;`	`a = a*(4*b-1);`
`m %= 3-n/2;`	`m = m%(3-n/2);`
`b -= 3+d*4;`	`b = b-(3+d*4);`
`k += (a*=2)-(b+=3);`	`k = k+((a=a*2)-(b=b+3));`

Zwróć uwagę na kolejność wykonywanych działań.

Operator warunkowy

Operator warunkowy jest jedynym w języku C++ operatorem trzyargumentowym. Jego składnia jest następująca:

$w_1 \ ? \ w_2 \ : \ w_3$

gdzie: w_1, w_2, w_3 — wyrażenia.

Wykonanie powyższej operacji przebiega następująco:

1. Obliczenie wartości wyrażenia w_1.
2. Jeśli wartość wyrażenia w_1 jest różna od 0 (czyli wyrażenie jest prawdziwe), rezultatem operacji jest wartość wyrażenia w_2. Wyrażenie w_3 nie jest wówczas obliczane.
3. Jeśli wartość wyrażenia w_1 jest równa 0 (czyli wyrażenie jest fałszywe), rezultatem operacji jest wartość wyrażenia w_3. Wyrażenie w_2 nie jest w tym przypadku obliczane.

Realizacja tego działania przypomina instrukcję warunkową `if`, która została omówiona w punkcie 3.3.3, „Instrukcje warunkowe".

Przykład 3.15.

Zastosujmy operator warunkowy do wyznaczenia najmniejszej wartości dla dwóch liczb całkowitych wprowadzanych z klawiatury (*prog3_10.cpp*):

```
int a, b, minimum;
cout<<"podaj dwie liczby całkowite:"<<endl;
cin>>a>>b;
minimum=a<b?a:b;
cout<<"minimum = "<<minimum<<endl;
```

Zadanie 3.2. Napisz program wyznaczający największą wartość dla trzech liczb rzeczywistych wprowadzanych z klawiatury. Zastosuj operator warunkowy.

Operator rozmiaru sizeof

Operator rozmiaru służy do wyznaczenia liczby bajtów, jaką zajmuje reprezentacja wartości określonego typu.

```
sizeof(wartość lub typ)
```

Przykład 3.16.

Przyjrzyjmy się przykładom zastosowania operatora `sizeof` (*prog3_11.cpp*):

```
int a;
double b;
a = sizeof(b);
cout<<a<<"\t"<<sizeof(char)<<endl;
```

Po wykonaniu programu wypisane zostaną następujące wartości:

8 1

3.2.5. Priorytety relacji i działań

Operatory występujące w języku C++, podobnie jak w matematyce, mają różne względem siebie priorytety ważności. W konsekwencji działania wykonywane przez operatory o wyższym priorytecie realizowane są w pierwszej kolejności. W tabeli 3.16 przedstawione zostały priorytety wybranych operatorów języka C++.

Tabela 3.16. Zestawienie priorytetów wybranych relacji i działań w języku C++

<table>
<tr><th>Priorytet</th><th>Operatory</th></tr>
<tr><td>1</td><td>() -> :: .</td></tr>
<tr><td>2</td><td>! + (znak) - (znak) ++ -- & (adres) * (wskaźnik) sizeof new delete</td></tr>
<tr><td>3</td><td>* / %</td></tr>
<tr><td>4</td><td>+ -</td></tr>
<tr><td>5</td><td>< <= > >=</td></tr>
<tr><td>6</td><td>== !=</td></tr>
<tr><td>7</td><td>&&</td></tr>
<tr><td>8</td><td>||</td></tr>
<tr><td>9</td><td>? :</td></tr>
<tr><td>10</td><td>*= /= %= += -=</td></tr>
</table>

W danym wyrażeniu operacje zawarte w jednym priorytecie realizowane są od lewej strony. Jeśli podczas pisania programu nie masz pewności, który z wykorzystanych przez Ciebie operatorów ma pierwszeństwo, **skorzystaj z nawiasów** (), które mają najwyższy priorytet.

Przykład 3.17.

Przeanalizuj obliczanie wartości następującego wyrażenia:

```
a = 7/3*2-7%9/2 = 2*2-7/2 = 4-3 = 1
```

Zwróć uwagę na kolejność wykonywanych działań zawartych w jednym priorytecie.

Dodajmy do tego wyrażenia nawiasy w taki sposób, aby kolejność działań nie zmieniła się:

```
a = (((7/3)*2)-((7%9)/2))
```

3.2.6. Funkcje matematyczne

W języku C++ mamy dostęp do wielu funkcji matematycznych, które są niezbędne przy wykonywaniu niektórych obliczeń. Zawarte są one w bibliotece `cmath`, którą należy dołączyć dyrektywą kompilatora:

```
#include <cmath>
```

W tabeli 3.17 przedstawiono zestawienie podstawowych funkcji matematycznych dostępnych w tej bibliotece.

Tabela 3.17. Zestawienie podstawowych funkcji matematycznych biblioteki cmath

Funkcje	Opis	Przykłady	Wyniki

`double pow(double x, double n)`	Potęgowanie: x^n	`double x=2, a, b;`
		`a=pow(x,3);`
		`b=pow(x,-1);`
			a = 8
			b = 0,5

`double sqrt(double x)`	Pierwiastkowanie: $\sqrt{x}$	`double x=9, a, b;`
		`a=sqrt(x);`
		`b=sqrt(16);`
			a = 3
			b = 4

`double sin(double x)`
`double cos(double x)`
`double tan(double x)`
	Funkcje trygonometryczne: *sinus*, *cosinus*, *tangens* (argumenty w radianach)	`double x=1, a, b, c;`
		`a=sin(x);`
		`b=cos(x);`
		`c=tan(x);`
			a = 0,841471
			b = 0,540302
			c = 1,55741

`int abs(int x)`
`float abs(float x)`
`double abs(double x)`
	Wartość bezwzględna: $\|x\|$	`double x=-10.5, a;`
		`a=abs(x);`
		`int y=-3, b;`
		`b=abs(y);`
			a = 10,5
			b = 3

Zadanie 3.3. Oblicz, jakie wartości w języku C++ mają podane wyrażenia:

a) `3*7/2-1`,
b) `3*7.0/2-1`,
c) `(3*7)/2-1`,
d) `3*(7/2.0-1)`,
e) `3*(7/2)-1`,
f) `3*7.0/(2-1)`,
g) `124%12-2*4`,
h) `4*105%10/2*108`,
i) `(-8-13*2)/2.5`,
j) `(pow(2.5,3)+17%5)-sqrt(4)`,
k) `12%5+2%5`,
l) `5+4*3.0/5`,
m) `2%5*3-4`,
n) `7/3%2+6`,
o) `10-4+2/2*3`,
p) `11/3%5+1`,
q) `4-5/4.0*2`.

Zadanie 3.4. Zapisz podane wyrażenia w języku C++:

a) $\sqrt{\dfrac{7}{a^3+\cos(b)}}$,

b) $\sin\left(\dfrac{(a+b)^4}{\sqrt{11}+1}\right)$,

c) $\left(\dfrac{\cos(a+1)}{\sqrt{5}+3}\right)^3$.

3.2.7. Liczby losowe

Funkcje przeznaczone do generowania liczb losowych znajdują się w bibliotekach `cstdlib` i `ctime`. W tabeli 3.18 przedstawiono wszystkie niezbędne do korzystania z liczb losowych elementy języka C++.

Tabela 3.18. Zestaw poleceń stosowanych przy generowaniu liczb losowych

Polecenie	Działanie
`rand()`	Funkcja generująca losową liczbę całkowitą zawartą między 0 i `RAND_MAX`
`RAND_MAX`	Predefiniowana stała, maksymalna wartość generowana przez funkcję `rand()`, może być różna w zależności od posiadanego kompilatora i bibliotek
`srand(time(NULL))`	Funkcja inicjalizująca funkcję `rand()`, przy każdym uruchomieniu programu uzyskujemy inną sekwencję liczb losowych
`time(NULL)`	Odczytany z zegara czas (w sekundach), jaki upłynął od 1970 roku, stanowi wartość bazową przy generowaniu liczb losowych

W tabeli 3.19 podano zastosowanie funkcji `rand()` do generowania liczb losowych oraz opis przedstawionych działań.

Tabela 3.19. Zastosowanie funkcji rand() do generowania liczb losowych

Zastosowanie funkcji `rand()` do wygenerowania liczby losowej `liczba`	Wykonane działanie
`liczba = p+rand()%(q-p+1);`	Generowanie liczby losowej całkowitej `liczba` z przedziału [p, q]
`liczba = rand()%(q+1);`	Generowanie liczby losowej naturalnej `liczba` z przedziału [0, q]
`liczba = p+(double)rand()/RAND_MAX*(q-p);`	Generowanie liczby losowej rzeczywistej `liczba` z przedziału [p, q]
`liczba = (double)rand()/RAND_MAX;`	Generowanie liczby losowej rzeczywistej `liczba` z przedziału [0, 1]
`liczba = (double)rand()/RAND_MAX*q;`	Generowanie liczby losowej rzeczywistej `liczba` z przedziału [0, q]

Przykład 3.18.

Wylosujmy liczbę całkowitą z przedziału [0, 10].

W części deklaracyjnej należy dodać dyrektywy preprocesora: `#include <cstdlib>` i `#include <ctime>`. W kodzie programu umieszczamy następujące polecenia (*prog3_12.cpp*):

```
int liczba;
srand(time(NULL));
liczba=rand()%11;
cout<<liczba<<endl;
```

Zadanie 3.5. Napisz program realizujący następujące operacje:

a) losowanie liczby całkowitej z przedziału [−15, 15],
b) losowanie liczby całkowitej z przedziału [3, 25],
c) losowanie liczby rzeczywistej z przedziału [1,5, 2,25],
d) losowanie liczby rzeczywistej z przedziału [0, 1],
e) losowanie liczby rzeczywistej z przedziału [0, 64,5].

3.2.8. Komentarze

Komentarz to tekst zawarty w kodzie programu, który nie jest analizowany przez kompilator. Wykorzystywany jest on do komentowania programu, w którym został umieszczony. Aby tekst został potraktowany jako komentarz, oznacza się go odpowiednimi znakami. Istnieją dwa rodzaje komentarzy, co zostało przedstawione w tabeli 3.20.

Tabela 3.20. Komentarze w języku C++

Rodzaj komentarza	Znaczenie
`/*` *komentarz* `*/`	Obejmuje tekst dowolnej długości; symbolem `/*` oznaczany jest początek, a `*/` koniec komentarza
`//` *komentarz*	Obejmuje jedną linijkę tekstu; symbolem `//` oznaczany jest początek wiersza, w którym zawarty jest komentarz

3.3. Podstawowe konstrukcje algorytmiczne

3.3.1. Instrukcja przypisania

Instrukcja przypisania stosowana jest, gdy chcemy zmienić wartość określonej zmiennej.

Przykład 3.19.

Poniżej podano przykłady instrukcji przypisania:

```
z=a*b-1;
n=n+2;
a=c%2+4;
```

Po lewej stronie operatora przypisania mogą pojawiać się zmienne lub elementy struktur danych, natomiast po prawej — wyrażenia.

3.3.2. Instrukcja złożona

Instrukcją złożoną (zwaną blokową) nazywamy dowolny ciąg instrukcji ograniczony nawiasami klamrowymi {}. Konstrukcję tę stosujemy w przypadku, gdy w miejscu, gdzie możemy wpisać tylko jedną instrukcję, musimy umieścić kilka instrukcji. Instrukcja złożona wykorzystywana jest więc do grupowania wielu instrukcji w jedną.

Przykład 3.20.

Przyjrzyj się przykładowi zastosowania instrukcji złożonej (*prog3_13.cpp*):

```
int a=1, b=2;
while (a<10)
{
 a+=3;
 b*=7+a;
}
```

3.3.3. Instrukcje warunkowe

Instrukcje warunkowe pojawiają się wówczas, gdy konstruowany algorytm zawiera warunki, od których spełnienia zależy kolejność wykonywanych działań.

Konstrukcje te mogą występować w dwóch postaciach. Składnia instrukcji warunkowej w **postaci pełnej** przedstawia się następująco:

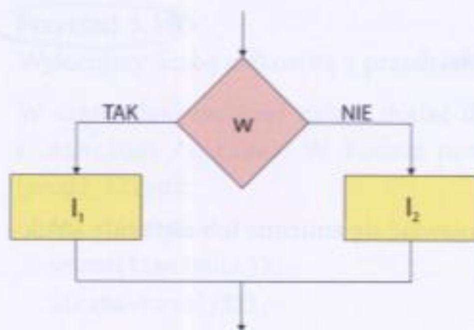
`if (w)` I_1`; else` I_2`;`

gdzie:

`w` — wyrażenie,

I_1, I_2 — instrukcje.

Na rysunku 3.3 pokazano schemat blokowy instrukcji warunkowej w postaci pełnej. Jeśli wyrażenie `w` jest prawdziwe, wykonywana jest instrukcja I_1. W przypadku gdy wyrażenie `w` jest fałszywe, przechodzimy do realizacji instrukcji I_2. W języku C++ przyjmuje się, że każda wartość różna od 0 jest prawdą, natomiast 0 to fałsz.



Rysunek 3.3. Schemat blokowy instrukcji warunkowej if w postaci pełnej

Składnia instrukcji warunkowej w **postaci skróconej** jest następująca:

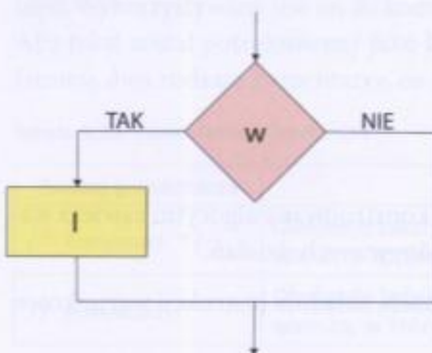
```
if (w) I;
```

gdzie:

`w` — wyrażenie,

`I` — instrukcja.

Na rysunku 3.4 przedstawiono schemat blokowy instrukcji warunkowej w postaci skróconej. Jeśli wyrażenie `w` jest prawdziwe, wykonywana jest instrukcja `I`. W przypadku gdy wyrażenie `w` jest fałszywe, kończymy instrukcję warunkową i przechodzimy do realizacji następnych instrukcji programu.



Rysunek 3.4. Schemat blokowy instrukcji warunkowej if w postaci skróconej

Przykład 3.21.

Przyjrzyjmy się podanym poniżej instrukcjom warunkowym. Dodany pod instrukcjami komentarz wyjaśnia ich działanie:

```
if (a>b) a=a-b; else b=b-a;
```

Jeśli `a>b`, wykonaj instrukcję `a=a-b`, w przeciwnym wypadku wykonaj `b=b-a`.

```
if (!z) z++;
```

Jeśli `z==0`, zwiększ z o 1.

```
if (z) z*=2;
```

Jeśli `z!=0`, przypisz `z=z*2`.

```
if (a%2) a=0; else a=1;
```

Jeśli a jest liczbą nieparzystą (wówczas a%2 jest różne od 0), przypisz a wartość 0, w przeciwnym wypadku przypisz a wartość 1.

Zadanie 3.6. Odszukaj instrukcje warunkowe zawarte w podanym kodzie. Połącz słowa `if` z odpowiednimi słowami `else`. Przedstaw podane konstrukcje algorytmiczne w postaci schematów blokowych:

a) `if (a<=0) if (a==0) m=5; else m=-5;`

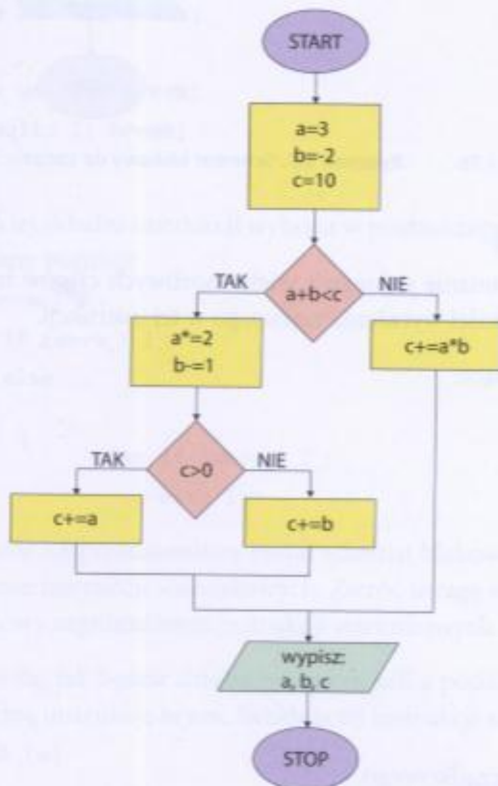
b) `if (a>0) m=0; else if (a<0) m=1; else m=2;`

c) `if (a>=0) if (a>0) m--; else m++; else m=0;`

d) `if (a>1) if (a<10) m-=2; else if (a==10); else m+=2; else if (!a); else m=0;`

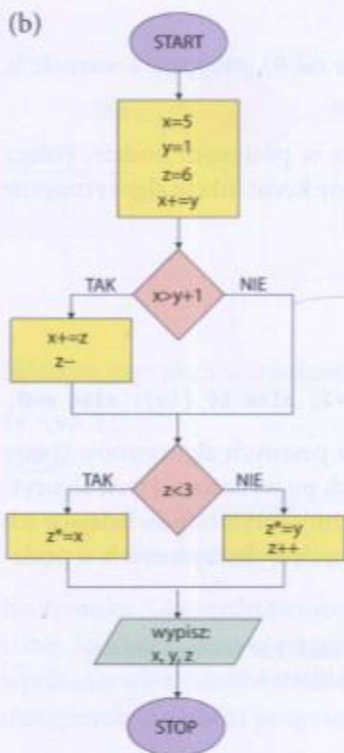
Zadanie 3.7. Poniżej przedstawiono schematy blokowe pewnych algorytmów (patrz rysunki 3.5, 3.6 i 3.7). Podaj, jaka będzie wartość zmiennych po wykonaniu tych algorytmów. Napisz programy realizujące przedstawione algorytmy (czyli odpowiadające ich schematom blokowym). Samodzielnie określ typy zmiennych wykorzystanych w podanych algorytmach.

(a)

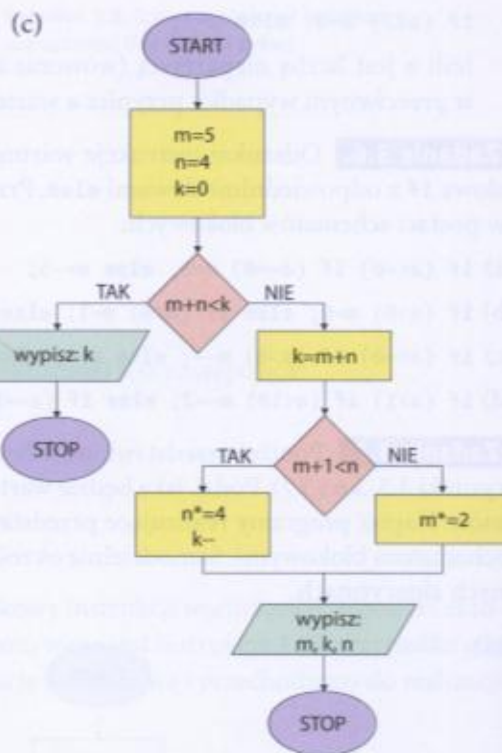


Rysunek 3.5. Schemat blokowy do zadania 3.7a

(b)



(c)



Rysunek 3.6. Schemat blokowy do zadania 3.7b

Rysunek 3.7. Schemat blokowy do zadania 3.7c

3.3.4. Instrukcja wyboru

Instrukcja wyboru pozwala na wykonanie jednego z wielu możliwych ciągów instrukcji. Wybór uzależniony jest tutaj od wartości wyrażenia zawartego w tej instrukcji.

Składnia tej instrukcji jest następująca:

```
switch (w)
{
 case w1: I1;
 case w2: I2;
 ...
 case wn: In;
 default: I;
}
```

gdzie:

`w` — wyrażenie typu porządkowego,

$w_1, w_2, ..., w_n$ — wartości wyrażenia `w`,

$I, I_1, ..., I_n$ — ciągi instrukcji.

Wyrażenie `w` musi być typu porządkowego, czyli należeć do grupy typów całkowitych, znakowych lub logicznych.

Wykonanie instrukcji wyboru polega na obliczeniu wartości wyrażenia `w`, odszukaniu wartości tego wyrażenia wśród $w_1, w_2, ..., w_n$ oraz zrealizowaniu ciągu instrukcji odpowiadającego znalezionej wartości w_i, dla `i=1, 2, ..., n` i każdej następnej $w_{i+1}, w_{i+2}, ..., w_n$ oraz etykiecie `default`. Jeśli żadna z wartości $w_1, w_2, ..., w_n$ nie jest równa wartości wyrażenia `w`, wykonywany jest ciąg instrukcji `I`, oznaczony etykietą `default`. Etykieta `default` może zostać pominięta w instrukcji wyboru. Wówczas, gdy wartość wyrażenia nie jest zgodna z żadną z wartości $w_1, ..., w_n$, instrukcja `switch` zostaje zakończona.

Instrukcja wyboru może stanowić prostszy sposób zapisu wielokrotnego zagnieżdżenia instrukcji warunkowej. W tym celu należy użyć polecenia `break`, które powoduje natychmiastowe przerwanie wykonywania instrukcji (patrz punkt 3.3.6, „Instrukcje sterujące"). Składnia instrukcji wyboru jest wówczas następująca:

```
switch (w)
{
 case w1: I1; break;
 case w2: I2; break;
 ...
 case wn: In; break;
 default: I; break;
}
```

Zapis takiej składni instrukcji wyboru w postaci zagnieżdżenia instrukcji warunkowych jest podany poniżej:

```
if (w==w1) I1;
else if (w==w2) I2;
     else ...
         ...
             else if (w==wn) In;
                  else I;
```

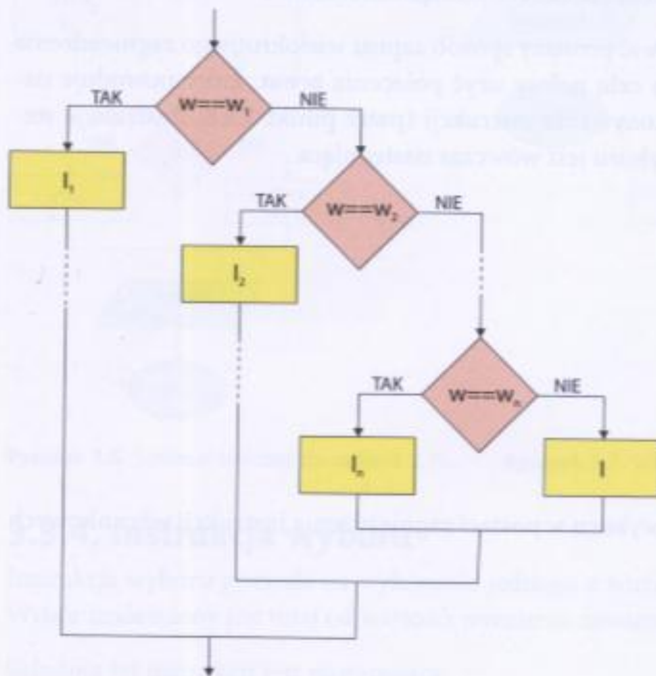
Na rysunku 3.8 przedstawiony został schemat blokowy instrukcji wyboru realizującej zagnieżdżenie instrukcji warunkowych. Zwróć uwagę na fakt, że jest on taki sam jak schemat blokowy zagnieżdżenia instrukcji warunkowych.

Zastanów się, jak będzie działać program, jeśli z podanej powyżej instrukcji wyboru usuniemy jedną instrukcję `break`. Składnia tej instrukcji `switch` może być wtedy następująca:

```
switch (w)
{
 case w1: I1;
 case w2: I2; break;
```

```
...
case wn: In; break;
default: I; break;
}
```

W podanym przypadku, jeśli w będzie równe w_1, wykonane zostaną ciągi instrukcji I_1 i I_2. Uogólniając, jeśli ciąg instrukcji I_j, dla j = 1, 2..., n, nie kończy się instrukcją break, to po nim będzie wykonywany następny ciąg instrukcji I_{j+1}. Operacje te będą realizowane aż do napotkania break albo końca instrukcji wyboru.



Rysunek 3.8. Schemat blokowy instrukcji wyboru realizującej zagnieżdżenie instrukcji warunkowych

Przykład 3.22.

Poniżej przedstawiono fragment programu, w którym zastosowano instrukcję switch. Algorytm realizuje wczytywanie numeru dnia i wypisywanie jego nazwy, na przykład 2 — wtorek, 3 — środa. W przypadku podania błędnej wartości pojawia się komunikat „podano błędną wartość" (*prog3_14.cpp*).

```
int n;
cout<<"podaj n: ";
cin>>n;
switch (n)
{
 case 1: cout<<"poniedziałek"<<endl; break;
 case 2: cout<<"wtorek"<<endl; break;
```

```
  case 3: cout<<"środa"<<endl; break;
  case 4: cout<<"czwartek"<<endl; break;
  case 5: cout<<"piątek"<<endl; break;
  case 6: cout<<"sobota"<<endl; break;
  case 7: cout<<"niedziela"<<endl; break;
  default: cout<<"podano błędną wartość"<<endl; break;
}
```

›rawdź, jak będzie działał program po usunięciu którejś z instrukcji `break`.

adanie 3.8. Samodzielnie zmodyfikuj program realizujący algorytm z przykładu 3.22, ımieniając instrukcję `switch` na zagnieżdżenie instrukcji warunkowych `if`.

adanie 3.9. Napisz program obliczający wartość podanej funkcji:

$$f(x)=\begin{cases}\sqrt{2x} & dla \quad n=1\\ x^3-5 & dla \quad n=2\\ \cos(x)+1 & dla \quad n=3\\ 1 & dla \quad innych \quad n\end{cases}$$

rgument rzeczywisty `x` i liczbę całkowitą `n` wprowadź z klawiatury. Zastosuj instrukę wyboru.

adanie 3.10. Napisz program wypisujący następujące menu:

. *Dane ucznia*

. *Aktualna data*

. *Samodzielnie wybrany algorytm*

. *Zakończenie programu*

raz wykonujący wybrane przez użytkownika polecenia 1. – 4.
› realizacji opcji 1. – 3. ponownie powinno wyświetlić się menu, natomiast po wyboæ polecenia 4. — program ma się zakończyć.

.3.5. Instrukcje iteracyjne

eracja to jedna z najważniejszych metod algorytmicznych. Została omówiona w podɔzdziale 1.5, „Iteracja". **Instrukcje iteracyjne** przeznaczone są do realizacji tej techniki, więc umożliwiają powtarzanie określonego ciągu operacji.

ıstrukcja while

kładnia tej instrukcji jest następująca:

```
while (w) I;
```

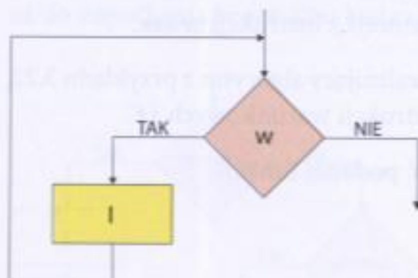
gdzie:

`w` — wyrażenie,

`I` — instrukcja.

Instrukcja `I` jest powtarzana dopóty, dopóki wyrażenie `w` jest prawdziwe, czyli jego wartość jest różna od 0.

Rysunek 3.9 przedstawia schemat blokowy instrukcji `while`.



Rysunek 3.9. Schemat blokowy instrukcji iteracyjnej while

Przykład 3.23.

Przeanalizujmy kod programu, w którym zastosowano instrukcję `while` (*prog3_15.cpp*):

```
int a=5;
while (a<10)
{
 cout<<setw(6)<<a;
 a++;
}
```

Po wykonaniu tego programu zostanie wypisany następujący ciąg liczb:

5 6 7 8 9

Instrukcja for

Podstawowa składnia instrukcji iteracyjnej `for` przedstawia się następująco:

```
for (wp; w; wz) I;
```

gdzie:

w_p — przypisanie wartości początkowej zmiennej sterującej pętlą,

`w` — wyrażenie,

w_z — wyrażenie zmieniające wartość zmiennej sterującej pętlą,

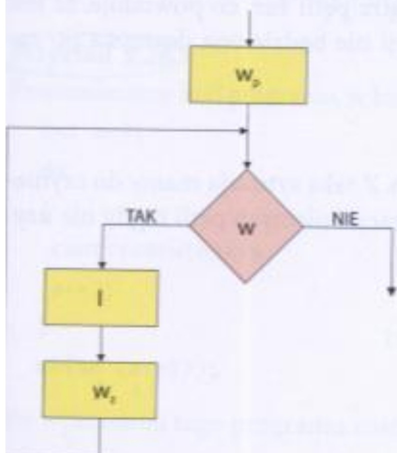
`I` — instrukcja.

Instrukcja `I` powtarzana jest dopóty, dopóki wyrażenie `w` jest prawdziwe, czyli jego wartość jest różna od 0.

Zmienna sterująca pętlą nazywana jest często licznikiem. Wartość wyrażenia `w` powinna być zależna od tej zmiennej.

Schemat blokowy instrukcji `for` przedstawiający jej zastosowanie w konkretnym algorytmie jest podobny do schematu blokowego instrukcji `while` realizującej te same działania. W obydwu przypadkach najpierw sprawdzana jest wartość logiczna wyrażenia `w`, a dopiero później wykonywana jest instrukcja `I`.

Rysunek 3.10 przedstawia schemat blokowy instrukcji `for`.



Rysunek 3.10. Schemat blokowy instrukcji iteracyjnej for

kawostka

Instrukcję `for` można stosować również w szerszym zakresie, wprowadzając do niej wiele wartości początkowych w_p oraz wiele wyrażeń zmieniających wartości zmiennych w_z. Poniżej przykład programu, który wypisuje wyrazy ciągu liczbowego (5, 7, 9, 11, 13):

```
for (int k=5, i=1; i<=5; k+=2, i++)
  cout<<k<<endl;
```

Jeśli w pętli `for` pozostawimy tylko wartość początkową oraz wyrażenie zmieniające wartość zmiennej sterującej pętlą `i`, to program będzie miał postać:

```
int k=5;
for (int i=1; i<=5; i++)
{
  cout<<k<<endl;
  k+=2;
}
```

rzykład 3.24.

oniżej przedstawiono kod programu, w którym zastosowano instrukcję `for` *rog3_16.cpp*):

```
int a;
for (a=22; a>=6; a-=3) cout<<setw(5)<<a;
```

Po wykonaniu tego programu zostanie wypisany następujący ciąg liczb:

22 19 16 13 10 7

Podany fragment programu można również zapisać następująco:

```
for (int a=22; a>=6; a-=3) cout<<setw(5)<<a;
```

W tym przypadku zmienną `a` zadeklarowano wewnątrz pętli `for`, co powoduje, że jest ona lokalna względem tej instrukcji. W konsekwencji nie będzie ona dostępna po zakończeniu polecenia `for`.

Przykład 3.25.

Przeanalizujmy teraz przykłady **pętli nieskończonych**. Z taką sytuacją mamy do czynienia wówczas, gdy wartość wyrażenia `w` podczas realizacji kolejnych pętli nigdy nie uzyskuje wartości równej `false`, czyli 0.

```
for (;;) { }
for (int a=10; a<16; a-=2) cout<<setw(5)<<a;
for (int a=5; ; a++) cout<<setw(5)<<a;
```

Instrukcja do-while

Składnia tej instrukcji jest następująca:

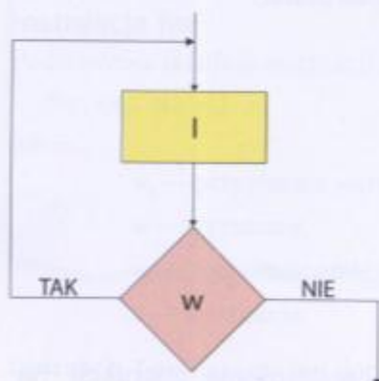
```
do I while (w);
```

gdzie:

`w` — wyrażenie,
`I` — instrukcja.

Instrukcja `I` jest powtarzana dopóty, dopóki wyrażenie `w` jest prawdziwe, czyli jego wartość jest różna od 0.

Rysunek 3.11 przedstawia schemat blokowy instrukcji `do-while`.



Rysunek 3.11. Schemat blokowy instrukcji iteracyjnej do-while

Schemat blokowy tej instrukcji iteracyjnej różni się od pozostałych przedstawionych wcześniej konstrukcji pętli. W tym przypadku w pierwszej kolejności wykonywana jest instrukcja `I`, a dopiero później sprawdzana jest wartość wyrażenia `w`. Taka kolejność gwarantuje, że bez względu na wartość wyrażenia `w` przynajmniej jeden raz instrukcja `I` zostanie wykonana. W przypadku instrukcji `while` i `for`, jeśli wyrażenie `w` jest fałszem przy pierwszym wejściu do pętli, instrukcja `I` nie zostanie wykonana.

Przykład 3.26.

Przeanalizujmy kod programu, w którym zastosowano instrukcję `do-while` (*prog3_17.cpp*):

```
int a=7;
do
{
 cout<<setw(6)<<a;
 a+=2;
}
while (a<=17);
```

Po wykonaniu tego programu zostanie wypisany następujący ciąg liczb:

7 9 11 13 15 17

Zadanie 3.11. Podaj, ile razy zostanie wykonana instrukcja zawarta w pętli:

a) `while (2) cout<<'a'<<endl;`

b) `do cout<<'b'<<endl; while (0);`

c) `while (0) cout<<'c'<<endl;`

Zadanie 3.12. Przypuśćmy, że dany mamy następujący kod programu:

```
double m=-3;
for (int i=0;i<3;i++)
 for (int j=6;j>=0;j-=2)
  m+=1.5;
```

Ile razy zostanie wykonana instrukcja `m+=1.5`? Jaka będzie wartość zmiennej `m` po wykonaniu podanych działań?

Zadanie 3.13. Na podstawie podanych fragmentów programów wykonaj następujące polecenia:

1. Narysuj schematy blokowe przedstawionych algorytmów.
2. Zamień instrukcje iteracyjne zgodnie z podaną poniżej listą i napisz programy z zastosowaniem tych instrukcji:

a) **for** na **do-while**,

b) **while** na **for**,

c) **do-while** na **for**.

3. Podaj, jakie wartości zostaną wypisane po wykonaniu programów.

a)

```
int k=-5, i;
for (i=1;i<=10;i+=2)
{
 cout<<k<<endl;
 cout<<i<<endl;
 k*=2;
}
cout<<k+i<<endl;
```

b)

```
int k=3, i=0;
while (i<4)
{
 cout<<k<<endl;
 k*=-3;
 cout<<i<<endl;
 i++;
}
cout<<2*k<<endl;
```

c)

```
int i=11, k=10;
do
{
 cout<<i<<endl;
 k-=4;
 cout<<k<<endl;
 i-=3;
}
while (i>=3);
cout<<k-2<<endl;
```

3.3.6. Instrukcje sterujące

Do tej grupy zaliczamy instrukcje, które wpływają na działanie określonych poleceń. Umieszczane są więc wewnątrz innych instrukcji i po uruchomieniu ingerują w ich działanie.

Instrukcja break

Instrukcja **break** powoduje natychmiastowe przerwanie wykonywania instrukcji, w której bezpośrednio została umieszczona. Dotyczy to zarówno przerwania pętli **while**, **for** i **do-while** (patrz punkt 3.3.5, „Instrukcje iteracyjne"), jak i instrukcji wyboru **switch** (patrz punkt 3.3.4, „Instrukcja wyboru"). W przypadku zagnieżdżenia instrukcji iteracyjnych **break** spowoduje przerwanie tylko tej pętli, w której bezpośrednio się znajduje.

Przykład 3.27.

Przeanalizujmy przykład zastosowania instrukcji **break** w pętli **for** (*prog3_18.cpp*):

```
for (int k=10; k<100; k+=5)
{
 cout<<setw(6)<<k;
 if (k==50) break;
}
```

W efekcie wykonania tego kodu wypisane zostaną następujące wartości:

10 15 20 25 30 35 40 45 50

Pętla została więc przerwana, gdy **k** osiągnęło wartość 50.

Instrukcja continue

Instrukcja **continue** stosowana jest do sterowania działaniem pętli. Wykorzystywana jest więc w instrukcjach **while**, **for** i **do-while**. Działanie tego polecenia polega na wymuszeniu natychmiastowego wykonania następnej iteracji z pominięciem pozostałych do wykonania instrukcji, które zawarte są w danej pętli.

Przykład 3.28.

Działanie instrukcji **continue** zostało pokazane w poniższym programie (*prog3_19.cpp*):

```
int k=0;
while (k<9)
{
 k++;
 if (k==4||k==6) continue;
 cout<<setw(5)<<k;
}
```

W efekcie wykonania powyższego kodu wypisany zostanie następujący ciąg liczb:

1 2 3 5 7 8 9

Dla **k**=4 i **k**=6 została wymuszona następna iteracja, co spowodowało, że wartości te nie zostały wypisane.