4 Instrukcje

- *Instrukcja* (ang. *statement*) opisuje czynności wykonywane w programie. W języku naturalnym analogiczną konstrukcją jest zdanie.
- W języku C++ wyróżnia się instrukcje pojedyncze (proste) i złożone. Instrukcja *pojedyncza* jest zakończona średnikiem. Ciąg instrukcji umieszczony w nawiasach klamrowych tworzy instrukcję *złożoną* (ang. *compound statement*). Nazywana jest ona również *blokiem*. Instrukcji złożonej nie trzeba kończyć średnikiem.
- Jeżeli nie powiedziano inaczej, instrukcje są wykonywane *sekwencyjnie* czyli w takiej kolejności, w jakiej występują w programie.
- Kolejność wykonywania obliczeń można zmieniać za pomocą instrukcji *sterujących* (ang. *control statements*). W języku C++ są to instrukcje: *wyboru*, *powtarzania* i *skoku*.
- Instrukcje *wyboru* (ang. *decision*) służą do wybrania jednej z kilku ścieżek wykonania programu.
- Instrukcje *powtarzania* (*pętli*, ang. *loop*) służą do iteracyjnego wykonywania fragmentu programu.
- Instrukcje *skoku* służą do bezwarunkowego przekazania sterowania do innego miejsca w programie.
- Instrukcje języka C++ przedstawiono w poniższej tabeli.

Typ instrukcji	Składnia
pusta	`;`
wyrażeniowa (prosta)	`<wyrażenie>;` `<wywołanie_funkcji>();`

złożona	`{`
	`<instrukcja>`
	`<instrukcja>`
	`<instrukcja>`
	`}`

instrukcje wyboru	`if (<wyrażenie>) <instrukcja>`
	`if (<wyrażenie>) <instrukcja> else <instrukcja>`
	`switch (<wyrażenie>)`
	`{`
	`<instrukcja>`
	`}`

instrukcje powtarzania (pętli)	`while (<wyrażenie>) <instrukcja>`
	`do <instrukcja> while (<wyrażenie>);`
	`for (<wyrażenie>;<wyrażenie>;<wyrażenie>)`
	`<instrukcja>`

instrukcje skoku	`break;`
	`continue;`
	`return <wyrażenie>;`
	`goto <identyfikator>;`

- Najprostszą instrukcją jest instrukcja pusta. Ma ona postać: ;
  Instrukcja ta nie powoduje wykonania żadnej czynności i jest wprowadzona przede wszystkim do ujednolicenia opisu pewnych konstrukcji.
- W tabeli zapis <instrukcja> bez średnika oznacza, że może być w tym miejscu użyta dowolna instrukcja: pusta, prosta lub złożona.

4.1 Instrukcja prosta i złożona

- Instrukcja *prosta* (*wyrażeniowa*) to wyrażenie lub wywołanie funkcji zakończone średnikiem.
- Przykłady:

```
i++;
y=2*i;
drukuj("Pomiary",t); // funkcja drukuje wyniki pomiarów
```

(Uwaga: funkcja nie zwraca wartości lub pomijamy zwracaną wartość, ponieważ nas ona nie interesuje.)

```
a=sqrt(x); // funkcja zwraca wartość określonego typu
```

- Instrukcja *złożona* (*blok instrukcji*) to ciąg instrukcji umieszczonych w nawiasach klamrowych. Po instrukcji złożonej nie używa się średnika. Składniowo instrukcja złożona jest równoważna jednej instrukcji prostej - można ją umieścić wszędzie tam, gdzie może się pojawić prosta instrukcja.
- Przykład:

```
// zamiana wartości zmiennych a i b
if (a > b)
{
      int tymcz = b;
      b=a;
      a=tymcz;
}
```

4.2 Instrukcja przypisania

- Instrukcja przypisania (podstawienia) pozwala zmienić wartość zmiennej podczas wykonywania programu.
- Ogólna postać instrukcji przypisania (ang. assignment statement):

  <*nazwa_zmiennej*> = <*wyrażenie*>;

- Przykłady:

```
i=1;
i=(2+3)*5;
i=2+3*5;
k=i+j;
```

- Instrukcje przypisania pozwalają tylko na realizację algorytmów najprostszej postaci: algorytmów sekwencyjnych (liniowych), które polegają na wykonaniu ciągu podstawień w takiej kolejności, jaka wynika z następstwa zapisu.

- Przykład:

Dane są boki trójkąta. Oblicz jego pole ze wzoru Herona: $s=\sqrt{(p(p-a)(p-b)(p-c)}$, gdzie *a*,*b*,*c* to boki trójkąta, zaś *p* to połowa obwodu.

```
// Rozwiązanie 1
#include <iostream> // dla cin i cout
#include <cmath> // dla obliczania pierwiastka kwadratowego sqrt
using namespace std;
int main()
{
  double a,b,c;
  double p,s;
  cin >> a >> b >> c; // zakładamy, że długości a,b,c tak są dobrane,
                      // że można z nich zbudować trójkąt
  p=(a+b+c)/2;
  s=sqrt(p*(p-a)*(p-b)*(p-c));
  cout << "Pole: " << s << endl;
return 0;
}
// Rozwiązanie 2
#include <iostream >
#include <cmath>
using namespace std;
int main()
{
  double a,b,c;
  cin >> a >> b >> c; // zakładamy, że długości a,b,c tak są dobrane,
  // że można z nich zbudować trójkąt
  cout << "Pole: " << sqrt((a+b+c)*(-a+b+c)*(a-b+c)*(a+b-c))/4<< endl;
  return 0;
}
```

- *Komentarz*

Algorytm 1: zmienna pomocnicza *p* i *s*, 5 dodawań i odejmowań, 4 mnożenia i dzielenia, jedno pierwiastkowanie
Algorytm 2: nie ma zmiennych pomocniczych, 8 dodawań i odejmowań, 4 mnożenia i dzielenia, jedno pierwiastkowanie.
Algorytm 1 jest oszczędniejszy ze względu na liczbę wykonywanych operacji, algorytm 2 zaś ze względu na oszczędność pamięci.

4.3 Instrukcja wyboru - `if-else`

- Instrukcja **`if`** służy do podejmowania decyzji. Umożliwia wykonywanie instrukcji lub bloku instrukcji w zależności od tego, czy sprawdzany warunek jest prawdziwy.

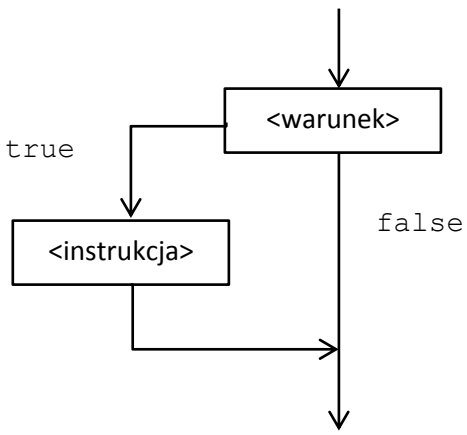
4.3.1 Pojedyńczy wybór - `if`

- Składnia:

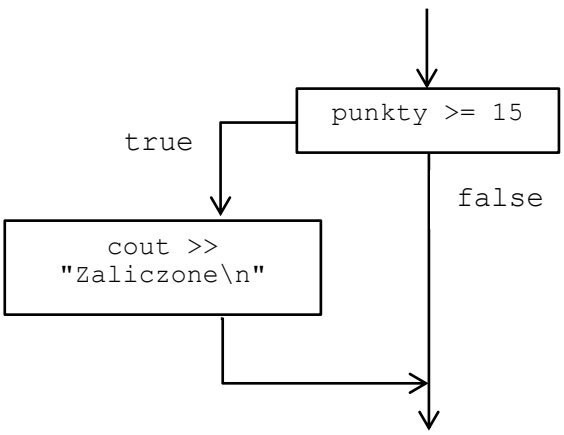
  `if` **(**<warunek>**)** <instrukcja>

- Działanie:

  <instrukcja> jest wykonywana tylko wtedy, gdy warunek jest prawdziwy (`true`).

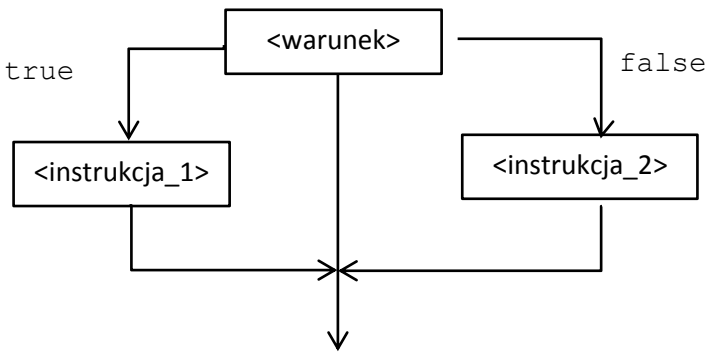



```
if (punkty >= 15 )  cout >> "Zaliczone\n";
```



4.3.2 Podwójny wybór `if-else`

```
if (<warunek)
      <instrukcja_1> // Wykonaj, gdy prawda
else
      <instrukcja_2> // Wykonaj, gdy fałsz
```



Przykład 3. Różne zapisy warunku w instrukcji `if`

```
if (x > 0) cout << "x jest dodatnie" << endl;

if (x != 0) cout << "x nie jest równe zeru" << endl;
if (x) cout << "x nie jest równe zeru" << endl; // krócej

if (x==0) cout << "x jest równe zero" << endl;
if (!x) cout << "x jest równe zero" << endl; // krócej

if (a>2 && a<5) cout << "a należy do przedziału" << endl;

if (!(a>2 && a <5)) cout << "a nie należy do przedziału" << endl;
// czytelniej
if (a <= 2 || a >=5) cout << "a nie należy do przedziału" << endl;
```

Przykład 4. Podwójny wybór `if-else` (w2p4)

```
// Program sprawdza, czy wpisano literę
#include <iostream>
#include <cctype> // dla isalpha()
using namespace std;
int main()
{
      char znak;
      cout << "Wpisz litere: ";
      // zamiast cin >> znak uzyjemy funkcji cin.get,
      // rezygnujemy z "inteligencji" operatora >>
      znak=cin.get();
      if (isalpha(znak))
            cout << "Wpisales litere: <" << znak << ">" << endl;
      else
            cout << "Znak <" << znak << "> o kodzie ASCII <"
                  << (int)znak <<"> to nie litera" << endl;
      return 0;
}
```

Komentarz:

- Funkcja `int isalpha(int c)` zwraca wartość różną od zera, jeżeli `c` jest literą. Nie są tu uwzględniane polskie litery.
- Poniższe zapisy są równoważne:

  ```
  if (isalpha(znak))
  if (isalpha(znak) != 0)
  ```
- Chcemy wczytać jeden dowolny znak, musimy więc zrezygnować z "inteligencji" operatora >> (na przykład pomijania spacji) i użyć odpowiedniej funkcji. Zamiast `cin >>` znak wywołujemy funkcję `cin.get()`.
- Operator << jest operatorem "inteligentnym" - rozpoznaje typ wyprowadzanej wartości. Chcąc wydrukować znak jako liczbę (czyli odpowiadający mu kod ASCII) musimy powiedzieć o tym operatorowi << za pomocą operatora rzutowania `(int)`.

Przykład 5. Użycie instrukcji złożonych

```
if (<warunek>)
{                    /* instrukcja złożona */
   <instrukcja_1>;
   <instrukcja_2>;
}
```

Poprawny fragment programu:

```
if (punkty < 15)
   cout << "Zaliczone\n";
else {
   cout << "Nie zaliczone\n";
   cout << "Termin egzaminu poprawkowego: \n";
}
```

Niepoprawny fragment programu:

```
if (punkty < 15)
   cout << "Zaliczone" << endl;
else
   cout << "Nie zaliczone" << endl;
   cout << "Termin egzaminu poprawkowego:" << endl;
```

Komentarz: składnia jest poprawna i kompilator skompiluje program. Jednakże niezależnie od liczby punktów wyświetlony zostanie komunikat: `Termin egzaminu poprawkowego:`, ponieważ z instrukcją `else` związano tylko jedną instrukcję: `cout << "Nie zaliczone" << endl;`

Przykład 6: Stopniowanie `if-else-if`

```
if (wyrażenie_1)
      instr_1;
else if (wyrażenie_2)
      instr_2;
...
else
      instr_n;
```

A. Zlicz liczby dodatnie, ujemne i zera:

```
if (x>0)
      l_dod++;
else if (x<0)
      l_uj++;
else // czyli x == 0
      l_zer++;
```

B. Wyświetl oceny

```
if ( punkty >= 28)
      cout << "bdb" << endl;
else if (punkty >= 25)
      cout << "db" << endl;
else if (punkty >= 15)
      cout << "dst" << endl;
else
      cout << "ndst" << endl;
```

Przykład 7: Zagnieżdżone struktury `if-else`

```
if (warunek_1)
{
      if (warunek_2)
            instrukcja;
}
else
      instrukcja;

/* drukuj 1 jeśli x ≠ 0 i y ≠ 0,
      drukuj 2, jeśli x = 0,
      w przeciwnym razie nic nie drukuj */
if (x)
{
      if (y)
            cout << "1" << endl;
}
else
      cout "2" << endl;//koniec wykładu 2
```

Przykład 8: Program zgadywanka

```
// Program zgadywanka wersja 1
#include <iostream>
#include <cstdlib> // dla rand()
using namespace std;
int main () {
      int liczba; // liczba z generatora
      int odp; // liczba podana przez zgadującego
   // generowanie liczby losowej
      liczba = rand()%10; // program ma generować liczby z zakresu 0-9
   // odgadywanie
      cout << "Podaj liczbę z zakresu od 0 do 9: ";
      cin >> odp;
      if (liczba == odp)
            cout << "* * Gratulacje. Odgadłeś * *" << endl;
      else
            cout << "* * Nie zgadłeś. Moja liczba to " << liczba <<" * *" <<endl;
      return 0;
}
```

Uwagi:

- Funkcja `rand()` generuje liczby pseudolosowe z zakresu 0 i RAND_MAX (stała zdefiniowana w `cstdlib`).
- Sekwencja liczb generowanych przez funkcję `rand()` jest zawsze taka sama. W przypadku naszego programu oznacza to, że za każdym razem użytkownik musiałby odgadnąć tę samą liczbę. Jeśli chcemy, żeby w każdym uruchomieniu programu generowana była inna liczba, trzeba dostarczyć jej innej wartości bazowej, na przykład za pomocą funkcji `srand()`.

```
/* program zgadywanka wersja 2 w2p5 */
#include <iostream>
#include <cstdlib>
#include <ctime> // dla time()
using namespace std;
int main ()
{
      int liczba; // liczba z generatora
      int odp; // liczba podana przez zgadującego
   // generowanie liczby losowej
      srand((unsigned) time(0));
      liczba = rand()%10;
   // odgadywanie
      cout << "Podaj liczbę z zakresu od 0 do 9: ";
      cin >> odp;
      if (liczba == odp)
            cout << "* * Gratulacje. Odgadłeś. * *" << endl;
      else {
            cout << "* * Nie zgadłeś. * *" << endl;
      if (odp > liczba) cout << "Twoja liczba jest zbyt duża\n" << endl;
      else cout << "Twoja liczba jest zbyt mała\n" << endl;
      }
      cout << "Moja liczba to " << liczba << endl;
      return 0;
}
```

Uwagi:

- Funkcja srand() pobiera argument całkowity typu unsigned i używa go do zainicjalizowania generatora liczb losowych.
- Jeśli nie chcemy wprowadzać za każdym razem liczby bazowej inicjalizującej generator liczb losowych, możemy wykorzystać funkcję `time()` z argumentem 0, która zwraca wtedy bieżący czas w sekundach.

4.4 Skrócony zapis instrukcji `if-else` czyli operator warunkowy `?`

Składnia:

```
<wyrażenie_1> ? <wyrażenie_2> : <wyrażenie_3>
```

- Jest to operator trójargumentowy. Zastępuje instrukcję `if-else` postaci:

```
if (<warunek>) <instrukcja>;
    else <instrukcja>;
```

- Działanie:
  1. Obliczana jest wartość <*wyrażenie_1*>;
  2. Jeśli tą wartością jest `true`, to obliczane jest <*wyrażenie_2*>, zaś <*wyrażenie_3*> jest ignorowane;
  3. W przeciwnym razie (wartością jest `false`) obliczane jest <*wyrażenie_3*>, zaś <*wyrażenie_2*> jest pomijane. Uzyskana wartość <*wyrażenie_2*> lub <*wyrażenie_3*> staje się wartością wyrażenia warunkowego
- Przykład 1

```
x=10;
znak = (x < 0) ? -1 : 1; /* zmiennej znak przypisane zostanie 1 */
```

To samo zapisane za pomocą instrukcji `if`:

```
x=10;
if (x < 0) znak =-1;
    else znak = 1;
```

- Przykład 2

```
cout << "Większa z liczb to: " << ( (a>b) ? a : b );
```

To samo zapisane za pomocą instrukcji `if`:

```
if (a>b)
    cout << "Większa z liczb to: " << a;
else
    cout << "Większa z liczb to: " << b;
```

- Przykład 3

```
int min (int a, int b)
    { return ( a<b) ? a : b; }
```

Zamiast

```
int min (int a, int b)
{
    if (a<b)
          return a;
    else
          return b;
}
```

- Przykład 4. Wartość bezwzględna

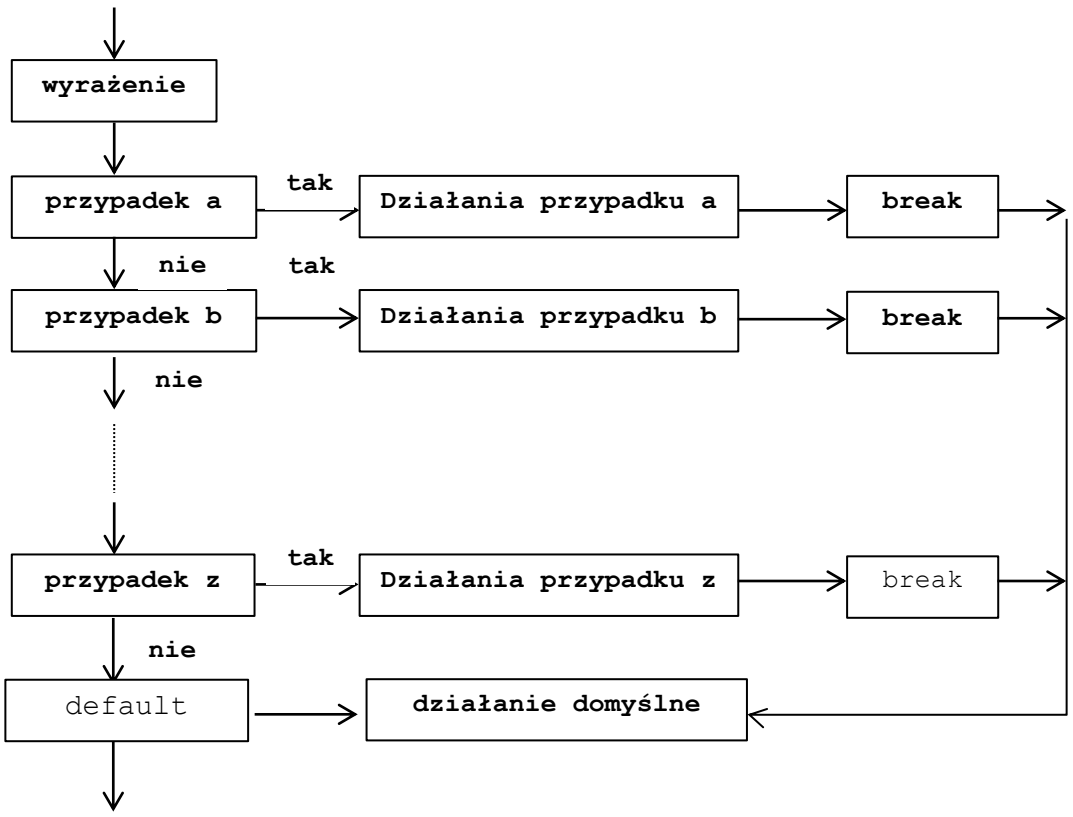
```
int abs(int x) {
 int ax;
 if (x >= 0)
      ax=x;
 else
      ax = -x;
 return ax;
}
```

```
int abs(int x) {
if (x < 0 )
   x = -x;
return x;
}
```

```
int abs(int x) {
return x>=0 ? x : -x;
}
```

4.5 Instrukcja wyboru `switch`

- Instrukcja **`switch`** służy do podejmowania decyzji wielowariantowych.



Składnia:

```
switch (wyrażenie_sterujące) {
      case wyrażenie_stałe: instrukcje
      ...
      case wyrażenie_stałe: instrukcje
      default: instrukcje
}
```

- Działanie:
  - Oblicz wartość wyrażenia sterującego, które występuje po słowie `switch`.
  - Porównaj kolejno tę wartość z wyrażeniami stałymi występującymi po słowie `case` (etykietami wariantów).
  - Jeżeli napotkana zostanie etykieta wariantu równa wartości wyrażenia, wykonaj instrukcje poczynając od pierwszej instrukcji występującej po tej etykiecie. Instrukcja `break` powoduje zakończenie instrukcji `switch` i przejście do pierwszej instrukcji za } zamykającym `switch`.
  - Jeżeli wartość wyrażenia `switch` nie odpowiada żadnej etykiecie `case`, a istnieje etykieta `default`, wykonaj instrukcje występujące po tej etykiecie. (Uwaga: nie musi to być ostatnia etykieta).
- *Uwaga*:
  - Wartość wyrażenia występującego po słowie `switch` musi być typu całkowitego
  - Instrukcje są wykonywane od pierwszej instrukcji występującej po wybranej etykiecie do końca instrukcji `switch` (nawias }), stąd potrzeba instrukcji `break`, która powoduje zakończenie wykonywania instrukcji wybranego przypadku i przejście do instrukcji umieszczonej po } zamykającym `switch`.
  - Przypadek `default` nie jest obowiązkowy. Jeśli nie występuje i wartość wyrażenia nie pasuje do innych przypadków, to nie podejmuje się żadnej akcji.

Przykład 1: Wyświetlenie opisu słownego otrzymanej oceny. Ocena wczytywana jest w postaci liczby.

```
cin >> ocena;
switch (ocena)
{
  case 5: cout << "bardzo dobry\n";
          break;
  case 4: cout << "dobry\n";
          break;
  case 3: cout << "dostateczny\n";
          break;
  case 2: cout << "mierny\n";
          break;
  case 1: cout << "niedostateczny\n";
          break;
  default:cout << "Ocena spoza zakresu\n";
          break; /* dobry zwyczaj programowania, to również break
          po ostatniej instrukcji case
          nie zapomni się wtedy o break w przypadku
          rozszerzenia o dodatkowy przypadek */
}
```

Przykład 2. Ustalenie liczby dni w wybranym miesiącu, miesiąc wybierany jest na podstawie numeru

```
switch (Miesiac)
{
// 30 dni ma Kwiecień, Czerwiec, Wrzesień, Listopad
case 4: case 6: case 9: case 11:
          Dni=30;
          break;
// pozostale miesiace mają 31 dni
case 1: case 3: case 5: case 7: case 8:
case 10: case 12:
          Dni=30;
          break;
// z wyjątkiem Lutego
case 2:
          if (RokPrzestepny)
               Dni=29;
          else
               Dni=28;
               break;
default:
          cout << "Nieprawidlowy numer miesiaca\n";
}
```

Przykład 3. Porównanie dwóch liczb i wyświetlenie odpowiedniego komunikatu

```
switch (x>y)
{
case 0: cout << "x nie jest większe od y" << endl;
          break;
case 1: cout << "x jest większe od y" << endl;
          break;
}
```

Przykład 4. Prosty kalkulator - funkcja

```
double oblicz (double arg1, double arg2, char op) {
switch (op) {
   case '+': return arg1 + arg2;
   case '-': return arg1 - arg2;
   case '*': return arg1 * arg2;
   case '/': return arg1 / arg2;
   default: cout << "Błąd: Nieznany operator" << endl;
            exit(1);
     }
}
```

Przykład 5. Prosty kalkulator - bez funkcji

```
#include <iostream>
using namespace std;
int main()
{
  char op; // znak dzialania
  double x, y; // argumenty
  cout << "Wpisz argument1 dzialanie (+ - * /) argument2\n"
       << " ( koniec: 0k) \n\n"; // aby zakończyć wpisz zero k
  for (;;) {
     cout << "Nastepne działanie: ";
     cin >> x;
     cin >> op;
     if (op == 'k')
        break; // break dla for(;;)
     cin >> y;
     switch (op) {
        case '+':
          cout << "= " << (x + y) << endl;
          break;
        case '-':
          cout << "= " << (x - y) << endl;
          break;
        case '*':
          cout << "= " << (x * y) << endl;
          break;
        case '/':
        case ':':
          if (y == 0.) {
          cout << "*** Dzielenie przez 0 ***\n";
          break; // break dla switch
          }
          cout << "= " << x / y << endl;
          break;
        default:
          cout << "*** Nieprawidlowe dzialanie ***\n";
          break;
     }
  }
  cout << "Dziekuje. Do widzenia" << endl;
  return 0;
}
```

Przykład 6: Proste menu.

```
int main() {
   char wybor;
   cout << "1. Dodaj rekord\n";
   cout << "2. Usuń rekord\n";
   cout << "3. Szukaj rekordu\n";
   cout << " Wpisz numer polecenia: ";
   cin >> wybor;

   switch (wybor)
   {
       case '1': dodaj_rekord();
                 break;
       case '2': usun_rekord();
                 break;
       case '3': szukaj_rekord();
                 break;
       default: cout << "Niewłaściwy numer polecenia\n";
                 break;
   }
   return 0;
}
```

Przykład 7: Program z menu.

```
#include <iostream>
#include <iomanip>
#include <cctype>       // dla toupper()
using namespace std;
const int MAXTYT = 10; // Max liczba tytułów
int liczbaEgz = 0; // Bieżąca liczba tytułów
int Tytuly[MAXTYT]; // Tablica tytułów
const int LEGZ = 1; // Liczba egz. tego samego tytułu
//************** Prototypy funkcji:
int ZnajdzTytul (int numer);
void PokazTytul (int numer);
void DodajTytul (int numer);
void UsunTytul (int numer);
void WykazTytulow ();
//*************** main ******************
int main() {
 char polecenie;
 int numer;
 cout << "\n Sklepik \n";
 cout << " System Magazynowy \n";
 for(;;) {
  cout << "\n";
  cout << "\t (P)okaz opis tytulu\n";
  cout << "\t (D)odaj tytul\n";
  cout << "\t (U)sun tytul\n";
  cout << "\t (W)yswietl aktualny stan\n";
  cout << "\t (Z)akoncz\n";
  cout << endl;
  cout << "Nastepne polecenie ==> ";
  cin >> polecenie;
  cin.ignore(80, '\n');
  polecenie = toupper(polecenie);
  if (polecenie == 'Z')
  break;
  cout << "\n\n**************************************\n";
  switch (polecenie) {
     case 'P':
      cout << "Numer tytulu: ";
      cin >> numer;
      PokazTytul(numer);
      break;
     case 'D':
      cout << "Numer tytulu: ";
      cin >> numer;
      DodajTytul(numer);
      break;
     case 'U':
      cout << "Numer tytulu: ";
      cin >> numer;
      UsunTytul(numer);
      break;
     case 'W':
      WykazTytulow();
      break;
     default:
      cout << "Niepoprawne polecenie.\n";
      break;
  }
  cout << "**************************************\n";
  }
}
```

```
//*************************************************************

//*************** Funkcje ******************
//*************************************************************
int ZnajdzTytul (int numer)
// Szuka tytułu o podanym numerze w tablicy Tytuly
// Zwraca jego indeks w tablicy, w przeciwnym wypadku zwraca -1
{
      int indeks=-1;
      // ...
      return indeks;
}
//*************************************************************
void PokazTytul(int numer)
// Wyświetla informacje o danym tytule
{
      int i;
      i = ZnajdzTytul(numer);
      if (i >= 0)
            cout << "Ilosc egzemplarzy: " << LEGZ << ".\n";
      else
            cout << "Brak .\n";
}
//*************************************************************
void DodajTytul(int numer)
// Dodaje nowy tytuł o podanym numerze do tablicy Tytuly.
// Sprawdza, czy tytuł ten jest już na stanie i czy tablica
// nie jest już pełna.
// Wyświetla "OK" jeśli operacja zakończona powodzeniem lub
// komunikat o błędzie
{
      // ...
}
//*************************************************************
void UsunTytul(int numer)
// Usuwa tytuł o podanym numerze z tablicy Tytuly, o ile tam się
// znajduje. Wyświetla odpowiedni komunikat, gdy tytułu nie ma na stanie.
{
      // ...
}
//*************************************************************
void WykazTytulow()
// Wyświetla listę tytułów na stanie
{
      int i;
      cout << "\t Nr tytulu L.egz.\n";
      cout << "\t---------------------\n";
      for (i = 0; i < liczbaEgz; i++) {
      // ...
      }
}
```

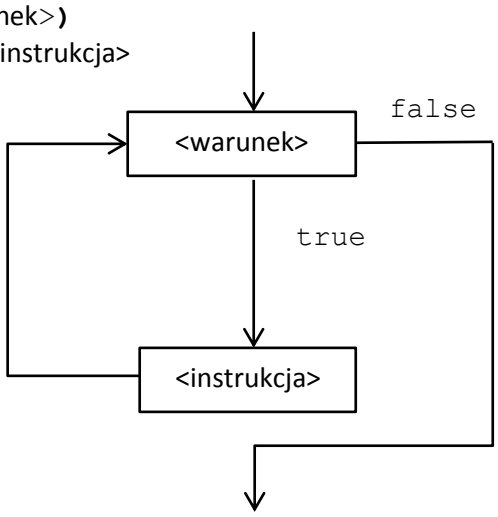
4.6 Instrukcje powtarzania

- Instrukcje powtarzania (iteracyjne) pozwalają wielokrotnie wykonać jeden ustalony ciąg instrukcji.
- Instrukcje te tworzą pętle (ang. *loop*).
- Pętle wykonują cyklicznie pewną część kodu programu dopóty, dopóki jakiś określony warunek jest prawdziwy.
- Taki warunek nazywany jest *warunkiem sterowania pętlą*.
- Instrukcje tworzenia pętli to:
  - `while` - warunek powtarzania jest sprawdzany *przed* wykonaniem treści pętli,
  - `do-while` - warunek powtarzania jest sprawdzany *po* wykonaniu treści pętli,
  - `for` - warunek powtarzania jest sprawdzany *przed* wykonaniem treści pętli.

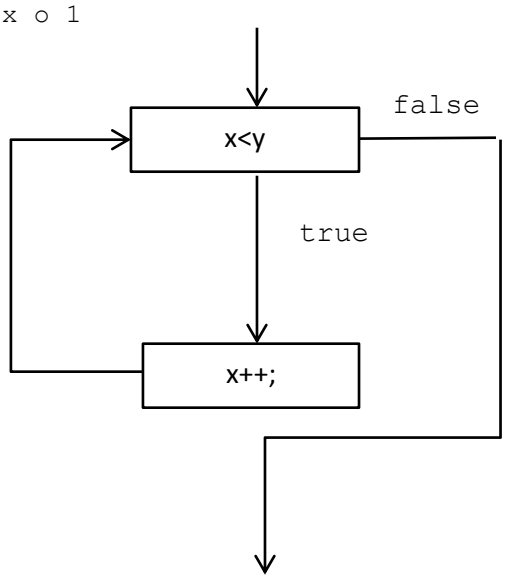
4.6.1 Pętla `while`

- Zasada działania:
  - Powtarzaj instrukcje, dopóki wyrażenie warunkowe jest prawdą.
  - Jeśli już za pierwszym razem warunek nie jest spełniony, to instrukcja związana z `while` nigdy nie będzie wykonana.

`while (`<warunek>**`)`**
<instrukcja>




```
// dopóki x jest mniejsze od y,
// zwiększaj x o 1
while (x<y)
      x++;
```



- Schemat działania:
  1. Oblicz wyrażenie warunkowe.
  2. Jeśli jego wartością jest `false`, przejdź do instrukcji następującej po pętli. Jeśli zaś `true`, wykonaj instrukcje objęte pętlą.
  3. Wróć na do kroku 1.

- Przykład: suma *n* liczb 1+2+3+...+*n*

```
int DodajLiczby(int n) {
int suma=0;
int i=1;      // inicjacja
while (i<=n) { // testowanie
  suma += i;
  i++;         // zmiana
}
return suma;
}
```

```
int DodajLiczby(int n) {
int suma=0;
while (n>0) { // testowanie
      suma += n--;
}
return suma;
}
```

- **Przykład 1.**

Pętla `while` powtarza związane z nią instrukcje tak długo, jak długo warunek kontrolujący wykonywanie pętli jest prawdziwy.

A. Dane są dwie liczby całkowite dodatnie *m* i *n*. Chcemy obliczyć ich największy wspólny dzielnik *k*.
Algorytm klasyczny Euklidesa:
Dane: dwie liczby naturalne *m* i *n*, *m* ≤ *n*.
Wynik: *NWD(m,n)* - największy wspólny dzielnik liczb *m* i *n*
Krok 1: Jeśli *m=0,* to *n* jest szukanym dzielnikiem. Zakończ algorytm.
Krok 2: *r:=n mod m*, *n=m*, *m=r*. Wróć do kroku 1.

```
#include <iostream>
using namespace std;
int main ()
{
    int m,n,r;
    cin >> m >> n; // zakładamy, że m i n > 0
    while (m) // skrót dla while( m != 0)
    {
      r = n % m;
      n = m;
      m = r;
    }
    cout << n << endl;
    return 0;
}
```

B. Chcemy obliczyć sumę ciągu liczb wprowadzanych z klawiatury, dodatkowo chcemy wiedzieć ile liczb zostało wprowadzonych. Koniec wprowadzania jest sygnalizowany znacznikiem końca pliku.

```
int x, licznik=0, suma=0;
while (cin >> x)
{
  ++licznik; // licznik określa ile liczb wprowadzono
  suma += x; // suma zawiera sumę wprowadzonych liczb
}
```

Komentarz:

- Pętla `while` będzie wykonywana dopóki warunek (`cin >> x`) będzie spełniony. Czytanie zakończy się niepowodzeniem wtedy kiedy:
  - napotkany zostanie znacznik końca pliku (można go wygenerować z klawiatury: w systemie Windows za pomocą klawiszy Ctrl-z, w Unixie klawisze Ctrl-d),
  - nastąpi próba wczytania danej, która jest niezgodna z typem zmiennej, w której ma być umieszczona (na przykład zamiast liczby typu `int` wpiszemy literę),
  - wystąpi uszkodzenie sprzętowe urządzenia wejścia.

  Stan strumienia wejściowego jest pamiętany dopóki program za pomocą odpowiedniej funkcji (`cin.clear()`) nie przywróci go do stanu umożliwiającego ponowne wczytywanie. Do tego momentu program nie będzie w stanie wczytywać nowych danych z tego strumienia. (Czyli każda nowa próba wczytania nawet poprawnych danych zakończona zostanie niepowodzeniem).

C. Chcemy obliczyć sumę ciągu liczb dodatnich wprowadzanych z klawiatury, dodatkowo chcemy wiedzieć ile liczb zostało wprowadzonych. Koniec wprowadzania jest sygnalizowany liczbą ujemną lub znakiem końca pliku.

```
int x, licznik=0, suma=0;
while ( (cin >> x) && (x > 0) )
{
 ++licznik; // licznik określa ile liczb wprowadzono
 suma += x; // suma zawiera sumę wprowadzonych liczb
}
```

D. Chcemy zliczyć samogłoski i spółgłoski we wprowadzanym tekście. Koniec wprowadzania jest sygnalizowany znacznikiem końca pliku. (Nie uwzględniamy polskich liter).

```
#include <iostream>
#include <cctype> // dla isalpha()
using namespace std;
int main() {
 char zn;
 int liczbaSamoglosek=0, liczbaSpolglosek=0;
 while (cin >> zn)
 {
 switch (zn)
 {
   case 'a' : case 'A' :
   case 'e' : case 'E' :
   case 'i' : case 'I' :
   case 'o' : case 'O' :
   case 'u' : case 'U' :
                        ++liczbaSamoglosek;
                        break;
   default :
             if (isalpha(zn)) ++liczbaSpolglosek;
             break;
 }
 }
 cout << "Liczba samoglosek: " << liczbaSamoglosek << endl;
 cout << "Liczba spolglosek: " << liczbaSpolglosek << endl;
 return 0;
}
```

Przykład 2.

Pętla `while` jest często używana do wczytywania danych wtedy, kiedy nie wiemy z góry ile razy będzie powtórzona. Można wtedy zbudować tzw. pętlę z wartownikiem - czyta ona i przetwarza dane aż do napotkania szczególnego niedozwolonego elementu powodującego zakończenie pętli. Element ten nazywany jest wartownikiem. Nie jest on przetwarzany.
Pętla z wartownikiem zazwyczaj wymaga wczytania danej przed pierwszym wykonaniem testu. Pseudokod pętli z wartownikiem ma postać:

```
wczytaj daną
while dana nie jest wartownikiem {
      przetwórz daną
      wczytaj daną
}
```

A. Chcemy obliczyć sumę ciągu liczb zakończonego liczbą 0. Wartownikiem w tym przypadku będzie liczba 0.

```
#include <iostream>
using namespace std;
int main() {
  int liczba, suma=0;
  cout << "Wpisz liczbe ('0' - koniec) ";
  cin >> liczba;
  while (liczba != 0)
  {
     suma += liczba;
     cout << "Wpisz liczbe ('0' - koniec) ";
     cin >> liczba;
  }
  cout << "Suma: " << suma << endl;
  return 0;
}
```

B. Chcemy wczytywać tekst z klawiatury znak po znaku i wyświetlać go na ekranie do momentu wprowadzania znaku końca pliku. Wartownikiem w tym przypadku będzie znacznik końca pliku EOF.

```
#include <iostream>
#include <cstdio> //dla EOFa
using namespace std;
int main() {
 int znak;
 znak=cin.get();
 while (znak != EOF) {
     cout.put(znak);
     znak=cin.get();
 }
 return 0;
}
```

Komentarz:

- Nie możemy zastosować konstrukcji `cin >> znak`, ponieważ operator `>>` pomija na przykład spacje. Musimy użyć funkcji, która wczytuje jeden znak. Taką funkcją jest `get()` bez argumentu. Pobiera ona jeden znak ze strumienia wejściowego i zwraca go jako wartość funkcji. Jeśli napotka w strumieniu koniec pliku, zwraca znacznik końca pliku EOF. Ponieważ funkcja ta dotyczy strumienia `cin`, który jest obiektem specjalnego typu, musimy użyć składni `obiekt.funkcja()`. Więcej informacji na ten temat - wykład "*Programowanie obiektowe*".
- Stała EOF symbolizuje znacznik końca pliku i jest zdefiniowana w pliku nagłówkowym `iostream`. Aby odróżnić znacznik od elementów zbioru znaków, często reprezentuje się go za pomocą liczby –1 (znak jest reprezentowany przez kod ASCII, a więc przez liczbę dodatnią). Zmienną, która przyjmuje wynik funkcji `get()` trzeba zatem zdeklarować jako `int`, tak aby mogła przyjmować zarówno wartości znaków jak i wartość stałej `EOF`.
- Odpowiednikiem funkcji `get()` dla strumienia wyjściowego `cout` jest funkcja `put()`. Wstawia ona jeden znak (swój argument) do strumienia wyjściowego.

C. Chcemy zliczyć spacje, tabulacje i znaki nowej linii w tekście.

```
#include <iostream>
using namespace std;
int main() {
  int zn;
  int l_sp=0, l_tab=0, l_nw=0;
  while ((zn=cin.get()) != EOF) {
     switch (zn)
     {
        case ' ' : l_sp++; break;
        case '\t' : l_tab++; break;
        case '\n' : l_nw++; break;
     }
  };
  cout << "Podsumowanie" << endl;
  cout << "Liczba spacji: " << l_sp << endl;
  cout << "Liczba tabulacji: " << l_tab << endl;
  cout << "Liczba nowych wierszy: " << l_nw << endl;
  return 0;
}
```

Przykład 3

Jeśli zawczasu wiemy, ile razy pętla będzie powtarzana możemy utworzyć pętlę z licznikiem - przetwarza ona dane tyle razy, ile wskazuje licznik.

```
// Załóżmy, że licznik i zmienia się od a do b
i=a;
while (i<=b) {
   // przetwórz i-ty element
   i++;
}
```

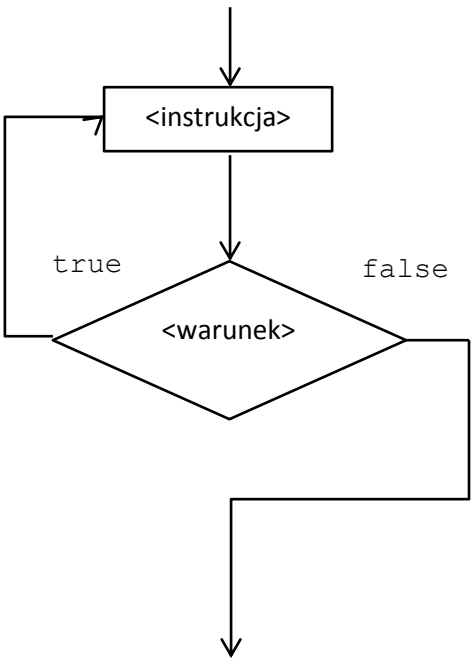
Komentarz:

- tego typu pętlę częściej buduje się z wykorzystaniem konstrukcji `for`.

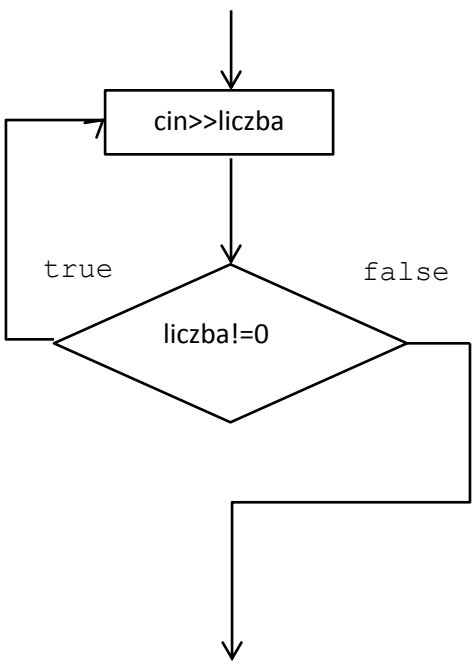
4.6.2 **Pętla do.. while**

- Zasada działania:
  - Wykonaj instrukcję. Sprawdź wartość wyrażenia warunkowego. Powtarzaj instrukcje, dopóki wyrażenie to jest prawdą.
  - Jeśli już za pierwszym razem warunek nie jest spełniony, to instrukcja związana z `do .. while` będzie wykonana dokładnie raz.

```
do
      <instrukcja>
while (<warunek>);
```




```
do
{
  cin >> liczba;
}
while (liczba != 0);
```



Schemat działania:

1. Wykonaj instrukcję pętli.

2. Oblicz wyrażenie warunkowe. Jeśli jego wartością jest `true`, wróć do kroku 1, w przeciwnym przypadku przejdź do instrukcji następującej po pętli.

Przykład 1. Obliczyć sumę liczb od 1 do 10.

```
licznik=1;
do
{
  cout << licznik;
  ++licznik;
}
while (licznik<=10);
```

Przykład 2. Wprowadzić liczbę z zakresu (2,5).

```
int min=2, max=5,x;
do
{
  cout << "Wpisz liczbę z zakresu od " << min << " do " << max <<":
";
  cin >> x;
}
while (x > min || x < max);
```

Przykład 3. Chcemy napisać program interaktywny, który po wykonaniu części obliczeń będzie pytał czy kontynuować pracę:

```
char z;
cout << "Kontynuowac (T/N): ";
do
{
  cin >> z;
  z=toupper(z);
}
while (z != 'T' && z != 'N');
```

Przykład 4. Pętlę `do..while` często wykorzystuje się wtedy, kiedy sterowanie pętlą jest ustalane wewnątrz treści pętli. Chcemy na przykład napisać program interaktywny, który po wykonaniu obliczeń będzie pytał czy powtórzyć je dla innych danych:

```
do
{
  // instrukcje
} while (czyPowtorzyc()) // czyPowtorzyc jest funkcją, która zwraca
true
                         // jeśli użytkownik zdecydował się
                           kontynuować,
                         // false w przeciwnym wypadku
```

To samo za pomocą pętli `while`:

```
// trzeba podać pierwszą wartość, aby rozpocząć pętlę
bool powtorzyc=true;
// teraz można rozpocząć obliczenia
while (powtorzyc) {
  // instrukcje
  powtórzyć=czyPowtorzyc();
}
```

Przykład 5. Sumowanie liczb. Liczby wczytywane są do wpisania 0.

```
#include <iostream>
using namespace std;
int main() {
int liczba, suma=0;
do
{
  cout << "Wpisz liczbe ('0' - koniec) ";
  cin >> liczba;
  suma += liczba;
}
while (liczba != 0);
cout << "Suma: " << suma << endl;
return 0;
}
```

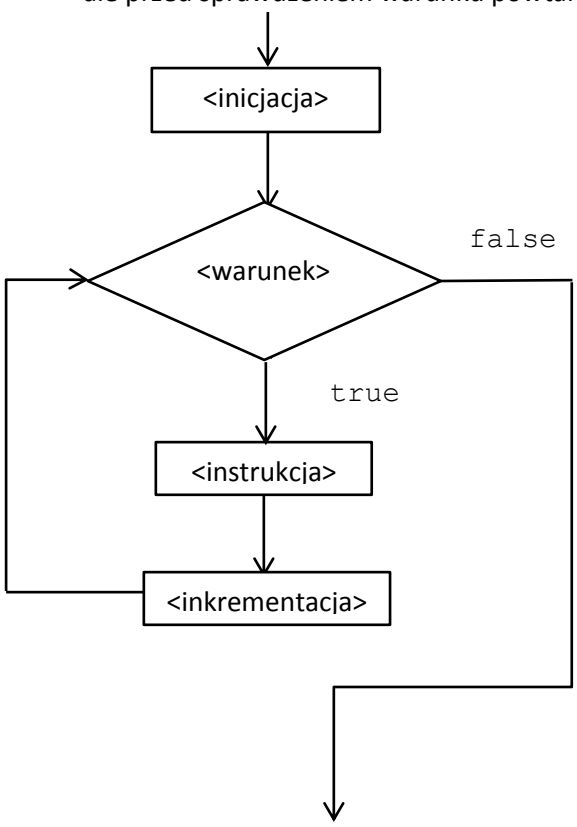
Przykład 6. Proste menu.

```
do
{
  wybor=cin.get();
  switch (wybor)
  {
     case '1': dodaj_rekord();
               break;
     case '2': usun_rekord();
               break;
     case '3': szukaj_rekord();
                break;
  }
}
while (wybor!='1' && wybor !='2' && wybor !='3');
```

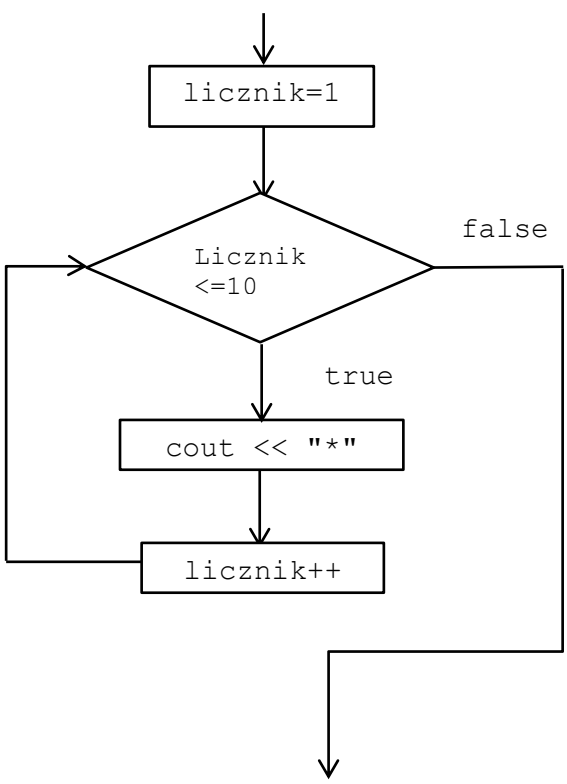
4.6.3 **Pętla `for`**

for (<*inicjacja*>; <*warunek*>; <*inkrementacja*>**)** <*instrukcja*>;
gdzie:

inicjacja	zwykle instrukcja przypisania wartości początkowej zmiennej sterującej pętli (licznika); jest wykonywana raz, przed rozpoczęciem powtarzania
warunek	wyrażenie określające warunek powtarzania; sprawdzane jest przed każdym kolejnym powtórzeniem pętli
inkrementacja	zwykle instrukcja modyfikująca zmienną sterującą pętli po zakończeniu każdego jej przebiegu (powtórzenia); jest wykonywana po każdym powtórzeniu pętli, ale przed sprawdzeniem warunku powtarzania




```
// wydrukuj 10 gwiazdek
for (licznik=1; licznik<=10; licznik++)
      cout << "*";
cout << endl;
```



Schemat działania:

1. Oblicz wyrażenie inicjujące pętli.
2. Oblicz wyrażenie warunkowe pętli (sterujące powtarzaniem).
3. Jeśli jego wartością jest `false`, przejdź do instrukcji następującej po pętli. Jeśli jego wartość jest `true`, wykonaj instrukcje pętli i następnie wyrażenie końcowe inkrementacji. Wróć do kroku 2 - sprawdzania wyrażenia warunkowego pętli.

• Przykład

```
int DodajLiczby(int n)
{
int suma=0;
int i;
for (i=1; i<=n; i++)
     suma += i;
return suma;
}
```

```
int DodajLiczby(int n)
{
int suma, i;
for (suma=0,i=1; i<=n;
           i++)
suma += i;
return suma;
}
```

```
// Uzupełnij
int DodajLiczby(int n)
{
int suma;
for (suma=0;
  ... ;
  ...);
return suma;
}
```

Przykład 1. Typowe użycie pętli `for` to pętla z licznikiem: przetwarzaj dane tyle razy, ile wskazuje licznik.

```
for (licznik=1; licznik < ILE; licznik++)
{
     // instrukcje
}
```

Równoważny zapis z użyciem `while`:

```
licznik=1;
while (licznik < ILE) {
     // instrukcje
     licznik++;
}
```

A. Chcemy obliczyć kwadraty liczb od 1 do 10

```
for (x=1; x<=10; x++)
{
     z = x*x;
     cout << "Kwadrat liczby " << x << " = " << z << endl;
}
```

B. Chcemy obliczyć sumę pierwszych 5 liczb naturalnych.

```
// Rozwiązanie 1: za pomocą pętli for
suma=0;
for (i=1;i<=5;i++)
      suma += i;

// Rozwiązanie 2: za pomocą pętli while
suma=0;
i=1;
while (i<=5)
{
      suma += i; /* zamiast suma = suma+i; */
      i++; /* zamiast i = i+1; */
}

// Rozwiązanie 3: za pomocą pętli do-while
suma=0;
i=1;
do
{
      suma += i;
      i++;
}
while (i<=5);
```

C. Licznik określający liczbę powtórzeń może być budowany na różne sposoby.

```
// zmienna znak będzie przyjmować wartości od 'a' do 'z'
for (char znak='a'; znak <= 'z' ; znak++)
   { // instrukcje }

// x przyjmuje wartości od 1.5 do 9.75 z krokiem 0.25
for (double x=1.5; x<10; x += 0.25)
   { // instrukcje }

// wykladnik przyjmuje wartości od 1 do n
for (potega=1.0, wykladnik=1; wykladnik<=n; ++wykladnik)
   potega *= n;
```

Przykład 2. Przy wyborze pętli warto jako jedno z kryteriów przyjąć czytelność programu. Przykład użycia pętli `for` w przypadku, gdy bardziej naturalne jest użycie pętli `while` :

```
for (bool znaleziono=false; !znaleziono;)
{
      // instrukcje
      znaleziono=szukaj();
      // instrukcje
}
```

Równoważny zapis z użyciem `while`:

```
bool znaleziono=false;
while (!znaleziono) {
      // instrukcje
      znaleziono=szukaj();
      // instrukcje
}
```

Przykład 3. Napisać program, który sprawdza wyniki testu. Wyniki te przechowywane są w postaci skompresowanej - wektora bitowego. W wektorze odpowiedzi 1 oznacza odpowiedź "tak", zaś 0 - "nie". Program ma utworzyć i wyświetlić wektor z niepoprawnymi odpowiedziami – w wektorze tym 1 oznacza, że nie odpowiedziano prawidłowo.

```
#include <iostream>
using namespace std;
/* prawidlowe odpowiedzi testu
      nntt nttt tttt nntt
      0011 0111 1111 0011
*/

const unsigned short int ODPOWIEDZ=0x37F3; //wzorzec prawidłowej odpowiedzi
int main() {
   unsigned short int i, wynik_testu=0, zle_odp;
   char c;
   cout << "Wpisz wyniki testu:" << endl;
   // ustaw bity odpowiedzi
   for (i=0; i<16; i++)
   {
       c=cin.get();
       if (c=='t' || c=='T')
       wynik_testu |= (1 << i); // skrót wynik_testu=wynik_testu | (1 << i);
   }

   // wyświetl wektor odpowiedzi
   cout << "Wpisano=" << endl;
   for (i=0x8000;i;i>>=1) // i=i>>1
   cout << ((i & wynik_testu) != 0); // skrót dla
                                     // if (i & wynik_testu) cout <<"1";
                                     // else cout << "0";
   cout << endl;

   // pozostaw bity z odpowiedziami niepoprawnymi
   zle_odp=wynik_testu^ODPOWIEDZ;
   // wyświetl wektor niepoprawnych odpowiedzi
   cout << "Zle odpowiedzi=" << endl;
   for (i=0x8000;i;i=i>>1)
   cout << ((i & zle_odp) != 0); // skrót dla
                                // if (i & zle_odp) cout <<"1 ";
                                // else cout << "0 ";
   cout << endl;
   return 0;
}
```

Przykład 4. Pętle zagnieżdżone

A. Wydrukować tabliczkę mnożenia dla liczb od 1 do 5

```
// wersja 1a
#include <iostream >
int main() {
 int i,j;
 const int max=5;
 for (i=1;i<=max;i++) {
   for (j=1;j<=max;j++)
      cout << i*j << ' ';
   cout << endl;
 }
 return 0;
}
```

Wynik działania programu:

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

```
// wersja 1a
#include <iostream>
int main() {
 int i,j;
 const int max=5;
 for (i=1;i<=max;i++) {
   for (j=1;j<=max;j++)
     cout << i*j << '\t';
   cout << endl;
 }
 return 0;
}
```

Wynik działania programu:

```
1   2   3   4   5
2   4   6   8   10
3   6   9   12  15
4   8   12  16  20
5   10  15  20  25
```

B. Chcemy wydrukować tabliczkę mnożenia, ale dodatkowo nadać wynikom postać:

```
1 2 3 4 5
---------------
1| 1 2 3 4 5
2| 2 4 6 8 10
3| 3 6 9 12 15
4| 4 8 12 16 20
5| 5 10 15 20 25
```

```
// wersja 2
#include <iostream>
#include <iomanip> // dla setw()
using namespace std;
int main() {
   int i,j;
   const int max=5; // tabliczka mnożenia do 5
   const int w=4; // wynik umieszczany jest w polu 4 znakowym

   cout << setw(w+1) << ' ';
   for (i=1;i<=max;i++)
       cout << setw(w) << i;
   cout << endl;
   cout << setw(w+1) << ' ';
   for (i=1;i<=max*w;i++)
       cout << '-';
   cout << endl;
   for (i=1;i<=max;i++) {
       cout << setw(w) << i <<'|';
       for (j=1;j<=max;j++)
           cout << setw(w) << i*j;
       cout << endl;
   }
   return 0;
}
```

Komentarz:

- Do zmiany formatowania strumienia wyjściowego służą manipulatory. Manipulator `setw()` pozwala ustalić szerokość wyjściowej reprezentacji liczby lub napisu. Jako argument podaje mu się wymaganą szerokość pola w znakach. Więcej informacji na temat formatowania można znaleźć w części materiałów do wykładu: *Biblioteka wejścia-wyjścia*.

4.7 Pętla nieskończona

- Pętla nieskończona nie zawiera warunku kończącego lub nigdy nie jest on osiągnięty.

- Przykłady:
  - Pętla `for` - brak warunku (wyrażenia sterującego) w instrukcji **`for`** oznacza, że zawsze jest on prawdziwy:

```
for (;;)
 cout << "ta pętla jest nieskończona" << endl;
```

  - Pętla `while` - wartość `true` lub stała 1 oznacza, że warunek zawsze jest prawdziwy.

```
while (true)
 cout << "ta pętla jest także nieskończona" << endl;
```

  - Pętla `do-while` - wartość `true` lub stała 1 oznacza, że warunek zawsze jest prawdziwy

```
do {
 cout << "i ta pętla także jest nieskończona" << endl;
} while (true);
```

- Jeśli stosujemy pętlę nieskończoną, musimy znaleźć inny sposób zakończenia pętli, niż przez sprawdzenie warunku powtarzania, czyli musimy użyć instrukcję skoku.

4.8 Instrukcje skoku `break` i `continue`

- Dają możliwość kontrolowanego zakończenia powtarzania pętli w inny sposób niż poprzez sprawdzanie warunku powtarzania pętli.

- **`break`** - może wystąpić instrukcjach powtarzania `for`, `while`, `do-while`. Powoduje przerwanie najciaśniej otaczającej ją instrukcji pętli (najbardziej zagnieżdżonej pętli) i przejście do instrukcji następnej po przerwanej pętli
- Instrukcje `break` występuje również w instrukcji `switch`. Powoduje tam zakończenie wykonywania instrukcji związanych z dopasowanym przypadkiem i przejście do instrukcji występującej po `}` zamykającym instrukcję `switch`.

- **`continue`** - może wystąpić tylko w instrukcjach powtarzania `for`, `while`, `do-while`; powoduje pominięcie wykonania instrukcji do końca najciaśniej otaczającej ją pętli i przejście do miejsca wznawiania tej pętli.

- **Przykład 1.** Pobieraj znaki z klawiatury do wprowadzenia litery a

```
for (;;) {
      znak=cin.get();
      if (znak=='a') break; //przejdź do pierwszej instrukcji za pętlą for
      cout << "Wpisałeś znak: " << znak << endl;
}
cout << "Koniec! Wpisałeś literę a." << endl;
```

Przykład 2. Wczytuj liczby i obliczaj ich pierwiastek kwadratowy, liczby ujemne pomijaj. Jeśli chcesz zakończyć, wprowadź 0.

```
x=-1;
while ( x ) // skrót while (x != 0)
{
      cin >> x;
      if (x<0) continue;
      if (x>0) cout << "Pierwiastek: " << sqrt(x) << endl;
}

do {
      cin >> x;
      if (x<0) continue
      if (x>0) cout << "Pierwiastek: " << sqrt(x) << endl;
} while (x);

for (i=0; i<10; ++i) {
      cin >> x;
      if (x<0) continue;
      if (x>0) cout << "Pierwiastek: " << sqrt(x) << endl;
}
```

Przykład 3. Obliczyć średnią ocenę. Wczytuj oceny do napotkania 0.

```
#include <iostream>
using namespace std;
int main() {
      int Ilosc=0;
      double Ocena, Suma=0;
      while (true) {
            cout << "Wpisz ocene. Konczy wpisanie l.ujemnej: ";
            cin >> Ocena;
            if (Ocena<0) break;
            ++Ilosc;
            Suma+=Ocena;
      }
      if (!Ilosc) cout << "Brak ocen" << endl;
      else cout << "Srednia ocena to: " << Suma/Ilosc << endl;
      return 0;
}
```

Przykład 4. Obliczanie pierwiastka kwadratowego

```
#include <iostream>
#include <cmath>
using namespace std;
int main () {
  double x;
  do
  {
     cout << "Podaj liczbę dodatnia: ";
     cin >> x;
     if (x < 0) {
      cout << "Prosiłem o liczbę dodatnią!" << endl;
      continue;
     }
     else if (x > 0)
      cout << "Pierwiastek kwadratowy wynosi: " << sqrt(x) << endl;
     else {
      cout << "Koniec." << endl;
      break;
      }
     }
  while (true);
return 0; }
```

4.9 Instrukcja skoku `goto`

- Jest to instrukcja, która nigdy nie jest konieczna i w praktyce zawsze można się bez niej obejść.
- Powszechnie uważa się, że instrukcja ta wprowadza bałagan do programu i czyni go nieczytelnym.
- Składnia:

```
goto etykieta
```

gdzie etykieta (ang. *label*) to identyfikator z dwukropkiem, który znajduje się w tej samej funkcji, w której odwołuje się do niej `goto`.

- Przykład :

```
x=1;
powtorz:
   x++;
   if (x<10) goto powtorz;
```

- Instrukcja `goto` jest stosowana do zaniechania przetwarzania w głęboko zagnieżdżonych strukturach programu, do jednoczesnego przerwania działania dwóch lub więcej pętli:

```
for (...)
for (...) {
...
if (niepowodzenie)
   goto stop;
...
}
...
stop:
cout << "Błąd w programie";
```

- Z instrukcji `goto` należy korzystać tylko w sytuacjach wyjątkowych.

4.10 Zadanie

Poniżej powtórzony jest przykład kalkulatora czterodziałaniowego z punktu 4.5. Przetestuj powyższy program. Znajdź sytuacje, w których program ten nie działa prawidłowo. Popraw program tak, aby był odporny na błędy popełniane przez użytkownika.

```
#include <iostream>
using namespace std;
int main()
{
   char op; // znak działania
   double x, y; // argumenty
   cout << "Wpisz argument-1 dzialanie (+ - * /) argument-2\n"
        << " ( koniec: 0k) \n\n";
   for (;;) {
     cout << "Nastepne: ";
     cin >> x;
     cin >> op;
     if (op == 'k')
       break; // break dla for(;;)
     cin >> y;
     switch (op) {
       case '+':
         cout << "= " << (x + y) << endl;
         break;
       case '-':
         cout << "= " << (x - y) << endl;
         break;
       case '*':
         cout << "= " << (x * y) << endl;
         break;
       case '/':
       case ':':
         if (y == 0.) {
         cout << "*** Dzielenie przez 0 ***\n";
         break; // break dla switch
         }
         cout << "= " << x / y << endl;
         break;
       default:
         cout << "*** Nieprawidlowe dzialanie ***\n";
         break;
     }
   }
}
```