1. Operator warunkowy

Jeśli nie ma potrzeby budowania rozległych instrukcji warunkowych możemy użyć prostszego operatora warunkowego w postaci:

warunek ? wyrażenie1 : wyrażenie2;

Powyższa konstrukcja zakłada, że jeśli warunek ma wartość *true* wypełniana jest instrukcja 1, jeśli *false* instrukcja 2.

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
    int a,b,c;

    a = 2;
    b = 7;
    c = (a > b) ? a : b;

   cout <<"Wynik warunku (a większe czy b?): " << c << '\n';
}
```

Przykład 2.

```
#include <iostream>
using namespace std;

int main()
{
    int x,y,z;
    x = 0;
    y = 1;
    z = (x > y) ? x : y;

//Jeśli wprowadzimy x = 4, a y = 8 to wynik teoretycznie będzie wynosił y (false) ale o wartości 1. Dlaczego?

   cout << "Podaj wartość x: ";
   cin >> x;
   cout << "Podaj wartość y: ";
   cin >> y;
   cout <<"Wynik: " << z << '\n';
}

//dzieje się tak dlatego, że operator wykonuje się przed realizacją podania nowych wartości x i y (następuje to później).
```

Przykład 3.

```
#include <iostream>
using namespace std;

int main()
{
    int x,y,z;
    x = 0;
    y = 1;

   cout << "Podaj wartość x: ";
   cin >> x;
   cout << "Podaj wartość y: ";
   cin >> y;

   z = (x > y) ? x : y;

   cout <<"Wartość warunku: " << z << '\n';
   cout <<"Wynik: " << z << '\n';
}

//Podobny przykład - tym razem przesunięte wykonanie warunku
```

Przykład 4.

```
#include <iostream>
using namespace std;

int main()
{
    string wynik;
    int x, y;

    x = 0;
    y = 0;

   cout << "Podaj wartość x: ";
   cin >> x;
   cout << "Podaj wartość y: ";
   cin >> y;

   (x > y) ? wynik = "prawda" : wynik = "fałsz";

   cout <<"x: " << x << '\n';
   cout <<"y: " << y << '\n';
   cout <<"x > y = " << wynik << '\n';
}
```

2. Pętle while oraz do-while

Pętla **WHILE** to jedna z najczęściej stosowanych pętli. Służą one do sprawdzenia warunku logicznego, następnie wykonania określonych instrukcji jeśli i dopóki warunek **ma wartość TRUE**. W pętli WHILE warunek jest sprawdzany na początku (w przeciwieństwie do instrukcji DO WHILE). Pętlę WHILE stosuje się tam, gdzie jest potrzeba przetworzenia nieokreślonej liczby danych.

Pętla wykonuje się tak długo jak warunek jest prawdziwy.

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
   int licznik = 10;

   while( licznik > 0 )
   {
     cout << licznik << '\n';
       --licznik;
   }
   cout << "Koniec odliczania!" << '\n';
}
```

Przykład 2.

```
#include <iostream>
using namespace std;

int main()
{
   int licznik;

   cout <<"Podaj licznik: ";
   cin >> licznik;

   while( licznik >= 0 )
   {
     cout << licznik << '\n';
       --licznik;
   }

   cout << "Koniec odliczania!" << '\n';
}
```

Przykład 3.

```
#include <iostream>
using namespace std;

int main()
{
 int i = 0,
 j;

 while (i <= 10)
 {
   cout <<"Wartość i: "<< i <<endl;
   j = 5;

   while (j == i) //zagnieżdżona pętla (inst. pętli wykonają się gdy j będzie = i)
   {
     cout <<"j = "<< j <<endl;
     j = --j;
   }
  ++i;
  cout <<"Wartość i: "<<i<<endl;
  getchar(); // czekanie na naciśniecie ENTER
 }
}
```

Pętla DO WHILE jest podobna do WHILE, jednak jej warunek sprawdzany jest na końcu. Ma to wpływ na działanie pętli, ponieważ instrukcje wykonają się przynajmniej RAZ.

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
 int a = 1;

  do {
     cout << "Podaj liczbę (różną od 0): ";
     cin >> a;
  } while(a == 0);
}
```

3. Pętla FOR

Pętla FOR jest używana gdy z góry wiemy ile przebiegów chcemy wykonać. Obojętnie czy jest to dana podana przez użytkownika, czy informacja wynikająca z obliczeń w programie.

for(int i = 1; i <= 10; i++)
{
[instrukcje, które będą się powtarzać dopóki „i” nie jest <= 10]
}

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
 int liczba = 10;

 for(int i = 1; i <= liczba; i++)
   {
    cout << "i: " << i << endl;
   }
}
```

Przykład 2.

```
#include <iostream>
using namespace std;

int main()
{
 int liczba;
 cout << "Podaj liczbe (1-10): ";
 cin >> liczba;

 for(int i = 1; i <= liczba; i++)
   {
    cout << "i: " << i << endl;
   }
}
```

Przykład 3.

```
#include <iostream>
using namespace std;

int main()
{
 for(int i = 5; i<= 100; i= i+5)
   cout << i << " ";
}
```

Przykład 4.

```
#include <iostream>
using namespace std;

int main()
{
   int x, y;

   cout << "Podaj x (Start): ";
   cin >> x;
   cout << "Podaj y (Meta): ";
   cin >> y;

   for(int i=x; i<=y; i++)
      cout << i << " ";
}
```

Przykład 5.

```
#include <iostream>
using namespace std;

int main()
{
   for(int i = 100; i >= 50; i--)
      cout << i << " ";
}
```

Przykład 6.

```
#include <iostream>
using namespace std;

int main()
{
   int y;
   cout << "Podaj y: ";
   cin >> y;

   for (int i = y; i!= -100; i--)
      cout << i << " ";
}
```

4. Instrukcja SWITCH

Instrukcja **SWITCH** to instrukcja warunkowa. Wykorzystujemy ją najczęściej gdy potrzebujemy wykonać różne instrukcje w zależności od wybranej opcji (zmiennej).

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
 cout << "Podaj swoją ocenę z informatyki: ";
 int ocena;
 cin >> ocena;

  switch(ocena)
    {
     case 1: cout << "Masz niedostateczny!"; break;
     case 2: cout << "Masz dwóję!"; break;
     case 3: cout << "Trója! - Mogło być lepiej ;)"; break;
     default: cout << "Masz wyżej niż 3! Gratuluję!";
     break;
    }
}
```

Przykład 2.

```
#include <iostream>
using namespace std;

int main()
{
  int i;
  cout << "Wybierz opcje:" << endl;
  cout << "1. Komputery" << endl;
  cout << "2. Aparaty fotograficzne" << endl;
  cout << "3. Konsole" << endl;
  cout << "Wybierz pozycję: " << endl;
  cin >> i;

  switch(i)
  {
    case 1: cout << "jesteś w menu komputery"; break;
    case 2: cout << "jesteś w menu aparaty fotograficzne"; break;
    case 3: cout << "jesteś w menu konsole"; break;
    default: cout << "niepoprawny wybor";
  }
}
```

Przykład 3.

```
#include <iostream>
using namespace std;

int main()
{
 float a,b;
 char wybor;

 cout << "Pierwsza liczba: " << endl;
 cin >> a;
 cout << "Podaj druga liczbę: " << endl;
 cin >> b;
 cout << "Wybierz '+' aby dodać " << endl;
 cout << "Wybierz '-' aby odjąć " << endl;
 cout << "Wybierz '*' aby pomnożyć "<< endl;
 cout << "Wybierz '/' aby podzielić "<< endl;
 cin >> wybor;

 switch(wybor)
 {
  case '+': cout<<"Suma wynosi: " << a+b; break;
  case '-': cout<<"Roznica wynosi: " << a-b; break;
  case '*': cout<<"Iloczyn wynosi: " << a*b; break;
  case '/': cout<<"Iloraz wynosi: " << a/b; break;
  default: cout<<"Wybrałeś niepoprawna opcje: ";
 }
}
```

5. Instrukcje break i continue

Instrukcje **BREAK** i **CONTINUE** nie powinny być często używane. Jedynie tam, gdzie jest to **konieczne.**

Instrukcja **continue** może być użyta tylko wewnątrz pętli. Powoduje ona:

- przerwanie wykonywania bieżącej iteracji pętli
- w przypadku pętli while i do-while powoduje skok do testowania warunku
- w przypadku pętli for powoduje skok do instrukcji inkrementacji, a następnie testowania warunku

Instrukcja **break** może być użyta wewnątrz pętli lub wewnątrz instrukcji switch. Powoduje ona przerwanie wykonywania pętli lub instrukcji switch i skok do pierwszej instrukcji za pętlą lub switch.

To instrukcja switch jest najczęściej używana w połączeniu z break.

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
 for( int x = 0; x < 10; x++ )
   {
     if( x == 7 )
     {
      cout << "Jestem w pętli!" << endl;
      cout << "Teraz 'x' = " << x << "." <<endl;
      cout << "Przerywam petle!" << endl;
      break; //instrukcja przerywająca pętlę
     }
     cout << "x = " << x << endl;
   }
   cout << "Jestem poza pętlą for! " << endl;
}
```

Przykład 2.

```
#include <iostream>
using namespace std;

int main()
{
 for( int x = 1; x < 7; x++ )
   {
     cout << "x = " << x << endl;
     if( x == 2 || x == 3 || x == 5 )
     {
       cout << "Teraz 'x' wynosi " << x;
       cout << " -> continue!" << endl;
       continue;
     }
     cout << "KONIEC kroku x = " << x << endl;
   }
}
```

6. Instrukcja GOTO

Instrukcja **goto** to instrukcja bezwarunkowego skoku. Powoduje ona przeniesienie wykonywania programu do miejsca oznaczonego etykietą. Nadużywanie instrukcji goto świadczy o bardzo złym stylu programowania.

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
 int i = 5;
 ++i;
 goto koniec;

 i = 0;
 cout <<"Ten tekst nie zostanie wypisany z powodu przejścia KONIEC"<<endl;

 koniec:
 cout <<"Zmienna i ma wartosc "<<i<<endl;

 cout <<endl<<"Nacisnij ENTER aby zakonczyc..."<<endl;
 getchar();
}
```

7. Tablice (jednowymiarowe)

Tablice służą do przechowywania wielu elementów takiego samego typu. Tablicę możemy wyobrazić sobie jako zbiór ponumerowanych elementów, np.: typu int. Ten „zbiór” ma jedną, wspólną nazwę. Każdy element natomiast ma swój unikalny numer („index”).

Przykład tablicy:

int tablicaLiczb[5];

Powyższy przykład mówi nam, że jest to zbiór o NAZWIE tablicaLiczb, zawierający 5 elementów typu int, numerowanych („indeksowanych”) od 0 do 4. Dlaczego od 0 do 4? Ponieważ elementy tablicy są numerowane od 0 (nie od 1). Powyższa tablica wygląda np. tak przy nadaniu danych początkowych:

int tablicaLiczb[5] = {1, 2, 5, 7, 9};

Jak widać w tablicy jest 5 elementów, a ich index (numeracja) to kolejno 0 (dla liczby 1), 1 (dla 2), i tak dalej.

Należy o tym pamiętać, ponieważ aby odwołać się do elementu 1 używamy indeksu 0, a do ostatniego n-1.

Przykład 1.

```
#include <iostream>
using namespace std;

int main()
{
  int tablicaLiczb[5] = {1, 2, 5, 7, 9};

cout<<"Pierwszy element tablicy o indexie 0 = "<<tablicaLiczb[0]<<endl;
cout<<"Drugi element tablicy o indexie 1 = " <<tablicaLiczb[1]<<endl;
cout<<"Trzeci element tablicy o indexie 2 = " <<tablicaLiczb[2]<<endl;
cout<<"Czwarty element tablicy o indexie 3 = " <<tablicaLiczb[3]<<endl;
cout<<"Piąty element tablicy o indexie 4 = " <<tablicaLiczb[4]<<endl;
//A co jeśli odwołamy się do elementu o indexie 5?
//cout<<"Element tablicy o indexie 5 ? " <<tablicaLiczb[5]<<endl;
}
```

Tablice jednowymiarowe deklarujemy następująco:

int tab[5] = {1, 3, 2, 1111, 9};

Operację nadania wartości można zrobić **JEDYNIE** przy **deklaracji (tworzeniu) tablicy**. Jeśli zostanie zadeklarowana mniejsza ilość np.: int tab[5] = {1, 2, 3} wówczas pozostałe miejsca zostaną wypełnione zerami. Przypisanie pustego nawiasu np.: int tab[5] = { }, sprawi, że wszystkie elementy będą zerami.

Przykład 2.

```
#include <iostream>
using namespace std;

int main()
{
  int tab[3];

  cout << "Podaj liczbe 1: ";
  cin >> tab[0];
  cout << "Podaj liczbe 2: ";
  cin >> tab[1];
  cout << "Podaj liczbe 3: ";
  cin >> tab[2];

  cout << "liczba [0] wynosi "<< tab[0] << endl;
  cout << "liczba [1] wynosi "<< tab[1] << endl;
  cout << "liczba [2] wynosi "<< tab[2] << endl;

  int sumaElementów = tab[ 0 ] + tab[ 1 ] + tab[ 2 ];
  cout << "Suma trzech liczb wynosi:" << sumaElementów <<endl;
}
```

Przykład 3.

```
#include <iostream>
using namespace std;

int main()
{

 int tab[5]; //deklaracja tablicy o 5 elementach od 0 - 4
 tab[0] = 43; //przypisanie do jej pierwszej komórki wartość 43
 tab[4] = 100; //przypisanie do jej ostatniej komórki wartość 100

 //tab[5] = 101 //taki zapis nie jest możliwy, nie ma indeksu nr 5, jesteś poza tablicą!!!

 cout<<"Pierwszy element: " <<tab[0]<<endl; //wyświetlenie zawartości pierwszej komórki tablicy (czyli 43)
 cout<<"Ostatni element: "<<tab[4]; //wyświetlenie zawartości ostatniej komórki, tj. 100
}
```

Wypełnianie tablicy znakami odbywa się nieco inaczej. Ostatnim elementem tablicy jest znak szczególny końca tablicy! „\0” Przy wyświetlaniu tablicy znaków podajemy tylko nazwę tablicy.

Przykład 4.

```
#include <iostream>
using namespace std;

int main()
{
 //pierwszy sposób
 char tab1[5] = {'a','b','c','\0'}; //tworzymy tablicę 5-elementową, która może przechować do 4 znaków, ostatni będzie znak specjalny \0
 cout<< tab1 << endl; //program wypisze ab

 //drugi sposób - nie podając wielkości tablicy - program sam dopasuje jej wielkość
 char tab2[] = {'a','b','\0'}; //tym razem tworzymy tablicę 3-elementową
 cout<< tab2 << endl;

 //trzeci sposób - podajemy ciąg znaków pamiętając o podwójnym cudzysłowie
 char tab3[] = "Kawa to paliwo bogów!";

 cout<< tab3 << endl;
}
```

Aby wczytać (podane przez użytkownika) dane do tablicy w postaci ciągu znaków, musimy pamiętać, że zwykły cin pobierze dane jedynie do pierwszej spacji (białego znaku). Aby tego uniknąć używamy metody np.: cin.getline

Przykład 5.

```
#include<iostream>
using namespace std;

int main()
{
  char tablica[50];
  cout<<"Podaj imie i nazwisko: ";
  cin.getline(tablica,50);
  cout<<"Twoje dane: "<<tablica<<endl;
}
```

7. Tablice (jednowymiarowe) - PĘTLE

Podstawowym narzędziem odczytywania elementów tablicy, a także wykonywania na niej operacji jest pętla for, jak i pozostałe pętle.

Przykład 6.

```
#include<iostream>
using namespace std;
int main()
{
  int tab[5] = {1,2,3,4,5};
  for(int i = 0; i<5; i++)
    cout <<"Indeks nr "<<i<<", element = "<<tab[i] <<endl;
}
```

Przykład 7.

```
#include<iostream>
using namespace std;

int main()
{
int tab[5];
//Podawanie elementów tablicy
for(int i = 0; i<5; i++)
  { cout<<"Podaj liczbę: ";
    cin >> tab[i]; //odczytaj liczbę z klawiatury i zapisz to tablicy o indeksie i
(tłumacząc – wpisana liczba zapisze się pod indeksem równym i z pętli)
  }
//Odczytywanie
cout<<"Elementy tablicy: ";
for(int i = 0; i<5; i++)
  {
    cout<<tab[i]<<", ";
  }}
```

Przykład 8.

```
#include <iostream>
using namespace std;

int main()
{
   int tab[5];
   int licznik = 0;

   //Użycie pętli do while do wypełnienia tablicy
   do
   {
      cout<<"Podaj liczbę: ";
      cin >> tab[licznik];
      licznik++; //w tej pętli musimy podnieść licznik "ręcznie" inkrementacją

   } while( licznik < 5 );

   cout << "Podales liczby: ";
   licznik = 0;

//przykładowe wyświetlanie

   do
   {
      cout << tab[licznik] << ", ";
      licznik++;
   } while( licznik < 5 );
}
```