

Wstęp do informatyki- wykład 7

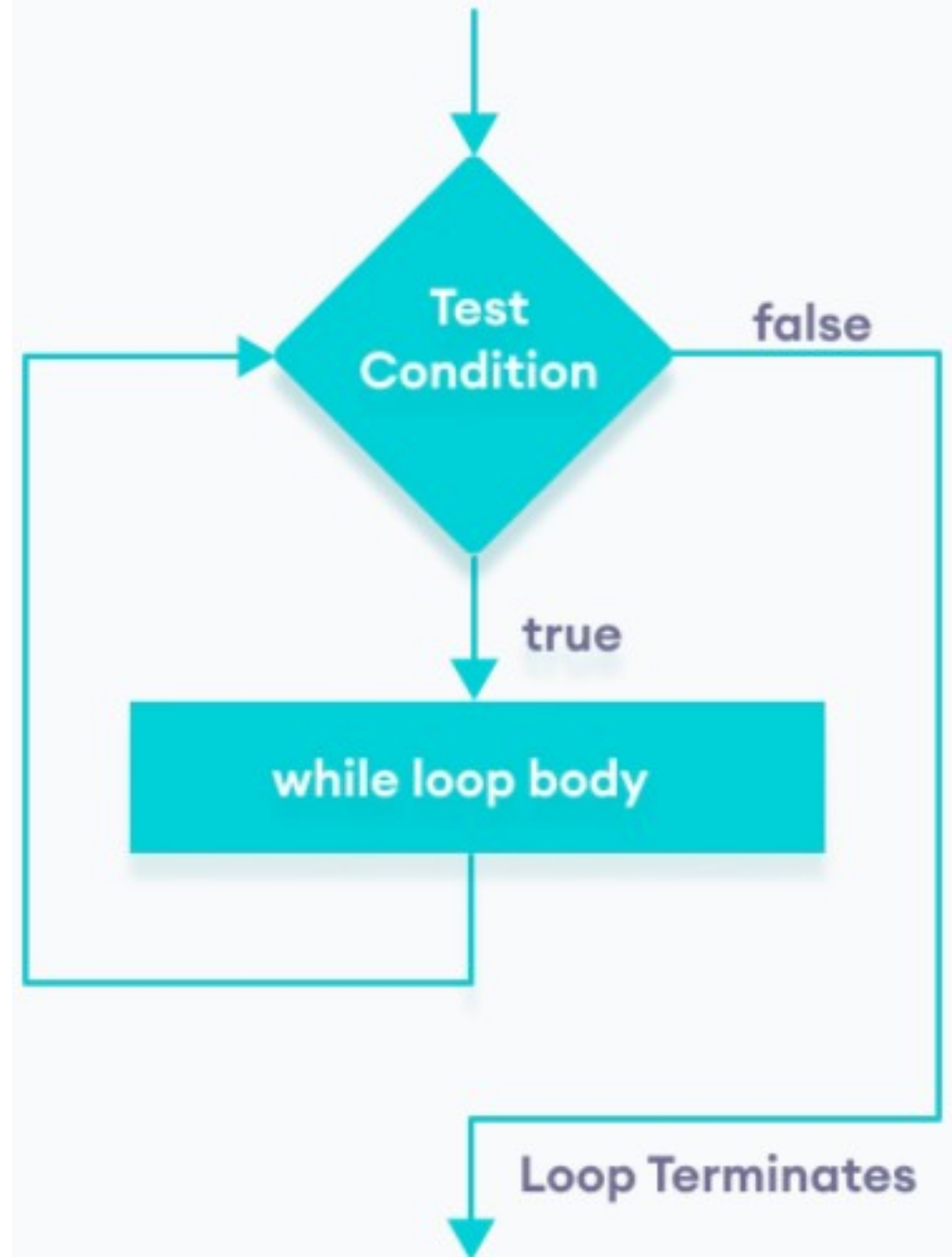
- **Pętla for cd.**
- **Instrukcje: break i continue**
- **Pętla while, do while, for -pętla w pętli- przykłady**
- **Operator rzutowania**
- **Formatowanie –setw**

Treści prezentowane w wykładzie zostały oparte o:

- S. Prata, Język C++. Szkoła programowania. Wydanie VI, Helion, 2012
- *www.cplusplus.com*
- Jerzy Grębosz, Opus magnum C++11, Helion, 2017
- B. Stroustrup, Język C++. Kompendium wiedzy. Wydanie IV, Helion, 2014
- S. B. Lippman, J. Lajoie, Podstawy języka C++, WNT, Warszawa 2003.

Instrukcje iteracyjne - pętla while

```
while (condition) {  
    // body of the loop  
}
```



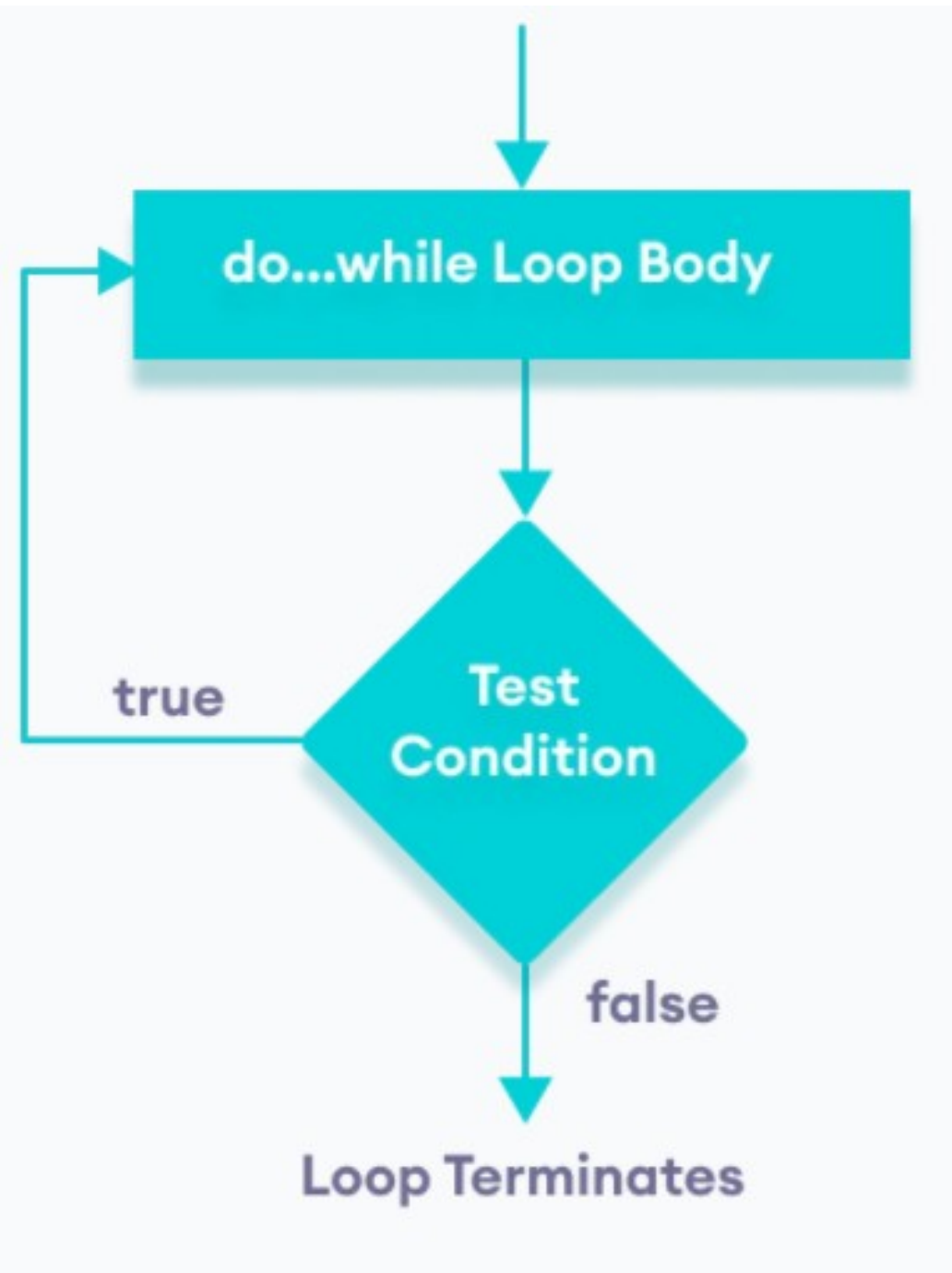
Instrukcje iteracyjne - pętla while

/ program oblicza ile liczb dodatnich podał użytkownik, podanie liczby ujemnej lub 0 kończy pętlę */*

```
int main() {  
    int n;  
    int ile = 0;  
  
    // pobieramy liczbę od użytkownika  
    cout << "Podaj liczbę: ";  
    cin >> n;  
  
    while (n > 0) {  
        //zliczamy liczbę dodatnia  
        ile++;  
  
        // pobieramy kolejną liczbę  
        cout << "Podaj kolejna liczba: ";  
        cin >> n;  
    }  
  
    // wyświetlamy wynik  
    cout << "\nLiczba dodatnich " << ile << endl;  
  
    return 0;  
}
```

Pętla do...while...

```
do {  
    // body of loop;  
}  
while (condition);
```



do while przykład

Użytkownik ma zgadnąć losowo wybraną liczbę z przedziału 1..100.

```
int main()
{
    int magic; // magic - losowo wybrana liczba
    int guess; // liczba podawana przez użytkownika

    magic = rand() % 100 + 1; // liczba losowa
                                // z przedziału od 1 do 100;
    do {
        cout << "Podaj twoja liczba: ";
        cin >> guess;

        if(guess == magic) {
            cout << "** Trafiles ** ";
            cout << magic << " to szukana liczba.\n";
        }
        else {
            cout << "...Niestety, nie trafiles.";
            if(guess > magic)
                cout << " Podales za duza liczba.\n";
            else
                cout << " Podales za mala liczba.\n";
        }
    } while(guess != magic);
}
```

Pętla zaporowa – wymuszanie poprawności danych

Poniższe pętle zapewniają, że wiek wprowadzony przez użytkownika jest z zakresu 13...99 .

```
//while
int main() {
    int age;
    cout << "Enter your age (valid range 13...99)" << endl;
    cin >> age;
    while (age < 13 || age > 99) {
        cout << "Invalid age, re-enter: ";
        cin >> age;
    }
}
```

```
//do-while
int main() {
    int age;
    do {
        cout << "Enter your age (valid range 13...99)" <<
endl;
        cin >> age;
    } while (!(age >= 13 && age <= 99));
}
```

Pętla for

```
for(instr_inicjaliz; wyr_warunkowe; instr_kroku)  
    treść_pętli ;
```

Praca pętli **for**:

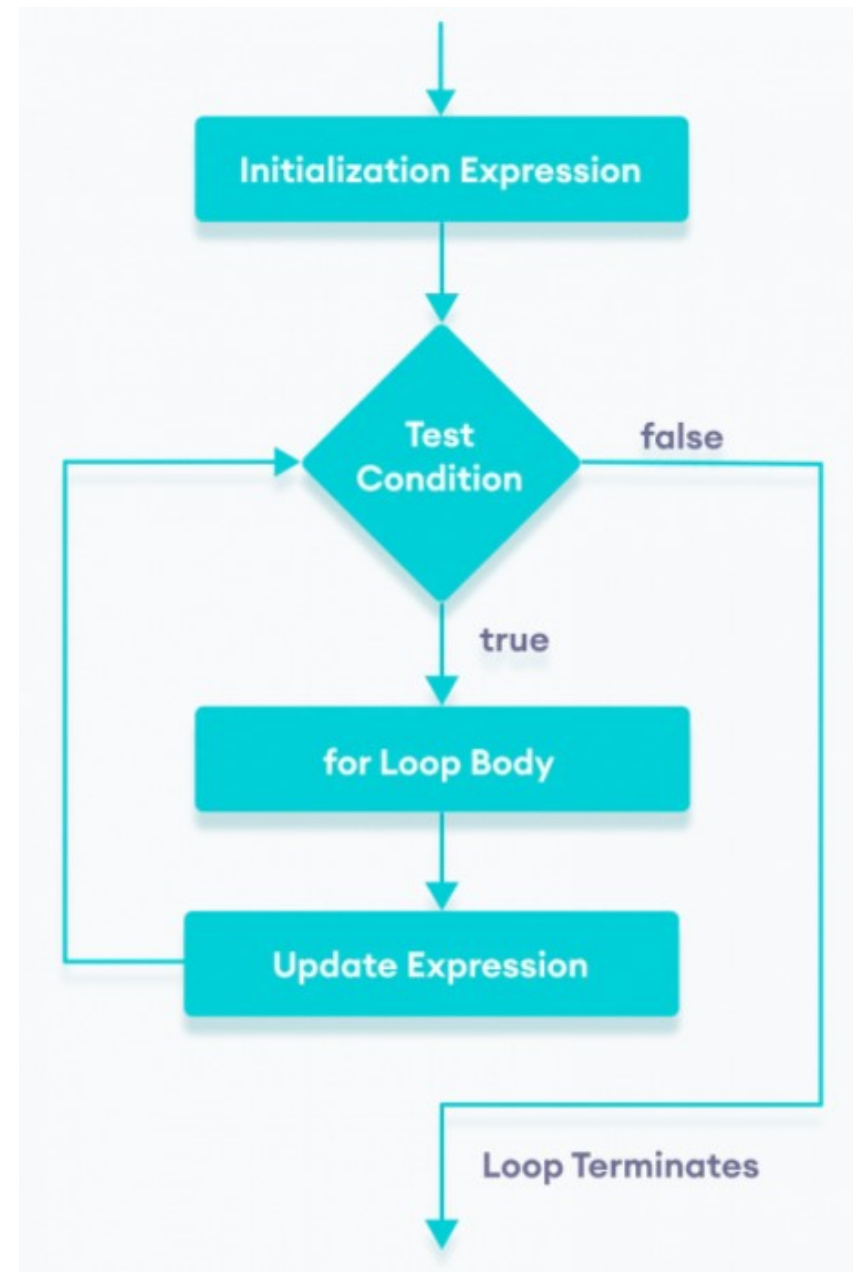
- (1) Najpierw wykonuje się instrukcje **instr_inicjaliz** inicjalizujące pracę pętli.
- (2) Obliczana jest wartość wyrażenia warunkowego **wyr_warunkowe**:
 - Jeżeli ma wartość **false** praca pętli jest przerywana.
 - Jeżeli zaś ma wartość **true**, wówczas wykonywane zostają instrukcje będące treścią pętli.
- (3) Po wykonaniu treści pętli wykonana zostaje instrukcja kroku pętli (**instr_kroku**), a następnie wracamy do (2).

Pętla for

```
for (initialization; condition; update) {  
    // body of-loop  
}
```

// Wyświetlamy litery od A do Z

```
for(char zn='A'; zn<='Z'; zn++)  
    cout<<zn<<endl;
```



Pętle – przykład - for– pętla z licznikiem

*// program obliczający sumę liczb 1+2+3+...+n
//n>0 pobrane od użytkownika(pętla zaporowa)*

```
int main() {  
    int n, sum;  
    sum = 0;  
  
    do {  
        cout << "Podaj liczba naturalna dodatnia: ";  
        cin >> n;  
    }while(n<=0);  
  
    for (int i = 1; i <= n; ++i) {  
        sum += i;  
    }  
  
    cout << "Suma = " << sum << endl;  
  
    return 0;  
}
```

Pętla - for – uwagi

- Poszczególne elementy: `instr_inicjaliz`, `wyr_warunkowe`, `instr_kroku` nie muszą wystąpić. Dowolny z nich można opuścić, zachowując jednak średnik oddzielający go od sąsiada.
- Opuszczenie wyrażenia warunkowego traktowane jest tak, jakby stało tam wyrażenie zawsze prawdziwe (warunek spełniony `true`).
- Tak więc pętla

```
for( ; ; )  
{  
    // treść pętli  
}
```


jest pętlą nieskończoną.

Pętle – break


- Zapoznaliśmy się już z działaniem instrukcji **break**, polegającym na przerywaniu wykonywania instrukcji **switch**.
- Podobne działanie ma **break** w stosunku do instrukcji pętli: **for**, **while**, **do...while**.
- Instrukcja **break** powoduje natychmiastowe przerwanie wykonywania tych pętli.
- Jeśli mamy do czynienia z kilkoma pętlami, zagnieżdżonymi jedna wewnątrz drugiej, to instrukcja **break** powoduje przerwanie tylko tej pętli, w której bezpośrednio tkwi. Jest to więc jakby przerwanie z wyjściem tylko o jeden poziom wyżej.

Pętle – break

```
for (init; condition; update) {  
    // code  
    if (condition to break) {  
        break;  
    }  
    // code  
}
```



```
while (condition) {  
    // code  
    if (condition to break) {  
        break;  
    }  
    // code  
}
```



Pętla – break - przykład

```
for (int i = 1; i < 10; i++)  
{  
    cout << i << '\n';  
    if (i == 4)  
        break;  
}
```

W rezultacie wykonania tego fragmentu programu na ekranie pojawi się:

1
2
3
4

Pętla – break - przykład

```
// program oblicza sumę liczb dodatnich pobieranych od użytkownika
// jeśli użytkownik poda liczbę ujemną, wychodzimy z pętli (break)
// wprowadzona liczba ujemna nie jest dodawana do sumy
int main() {
    int number;
    int sum = 0;

    while (true) {
        // pobieramy liczbę
        cout << "Podaj liczbę: ";
        cin >> number;

        // sprawdzamy czy liczba ujemna - warunek break
        if (number < 0) {
            break;
        }


        //dodajemy liczby dodatnie
        sum += number;
    }

    // wyświetlamy sumę
    cout << "Suma " << sum << endl;
    return 0;
}
```

Pętle – `continue`

- Instrukcja **`continue`** przydaje się wewnątrz pętli `for`, `while`, `do...while`.
- Powoduje ona zaniechanie wykonywania instrukcji będących dalszą treścią danego obiegu pętli, jednak (w przeciwieństwie do instrukcji **`break`**) sama pętla nie zostaje przerwana.
- **`continue`** przerywa tylko ten obieg pętli i przygotowuje do rozpoczęcia następnego, kontynuując pracę pętli.

Pętle – continue



```
for (init; condition; update) {  
    // code  
    if (condition to break) {  
        continue;  
    }  
    // code  
}
```



```
while (condition) {  
    // code  
    if (condition to break) {  
        continue;  
    }  
    // code  
}
```


Pętle – continue -przykład

```
for(int k = 0 ; k < 12 ; k = k + 1)
{
    cout << "A";
    if(k > 1)
        continue;
    cout << "b\n"; //ta instrukcja będzie zaniechana
                   //jeśli wykona się continue
}
```

W rezultacie wykonania tego fragmentu programu na ekranie pojawi się:

Ab

Ab

AAAAAAAAAA

Pętle – break – przykład 2 – czy jest 1 wśród cyfr liczby

```
#include <iostream>
using namespace std;
int main()
{//wczytuje z klawiatury liczbę całkowitą i sprawdza,
  // czy wśród cyfr tej liczby jest 1.
  int n;
  cout<<"podaj liczba całkowita ";
  cin>>n;
  int rob=(n<0 ? -n : n);//rob=abs(n);
  bool jest1 = false;
  //tu może być while bo szukamy cyfry 1!=0
  //jeśli szukalibyśmy 0 to do while
  while(rob>0 && !jest1)//dopóki liczba się nie skończyła
  {
    //i nie znaleźliśmy 1
    if(rob%10==1) //sprawdzamy czy ostatnia cyfra to 1,
      jest1=true; //jeśli tak to true
    else rob/=10;//jeśli nie to pozbywamy się ostatniej
    cyfry
  }
}
```

Pętle – break – przykład 2 – czy jest 1 wśród cyfr liczby

//wersja z break

```
rob=abs(n);
```

```
bool jest1 = false;
```

```
while(rob>0) //dopóki są cyfry w liczbie
```

```
{
```

```
    if(rob%10==1) //sprawdzamy ostatnią cyfrę czy == 1
```

```
{
```

```
        jest1 = true; //jeśli tak to jest1 na true
```

```
        break; //znaleźliśmy 1, przerywamy pętlę
```

```
}
```

```
    rob = rob/10; //skracamy liczbę przez 10
```

```
}
```

```
if(jest1)
```

```
    cout<<"jest 1 wsrod cyfr liczby "<<n<<endl;
```

```
else
```

```
    cout<<"nie ma 1 wsrod cyfr liczby "<<n<<endl;
```

```
return 0;
```

```
}
```

Pętle – przykład – pętla w pętli – czynniki pierwsze

*// Rozkład liczby naturalnej na czynniki pierwsze
// np. $12 = 2 \cdot 2 \cdot 3$ $40 = 2 \cdot 2 \cdot 2 \cdot 5$ $90 = 2 \cdot 3 \cdot 3 \cdot 5$
/* Elementarnym sposobem rozkładu liczb na czynniki
pierwsze jest wykonywanie kolejnych dzielen, np.:*

*56/2
28/2
14/2
7/7
1/ */*

```
#include <iostream>
#include <afxres.h>
using namespace std;

int main()
{
    SetConsoleOutputCP(CP_UTF8);
    //system("chcp 65001"); //zmiana strony kodowej
    unsigned int n;
    cout << "podaj liczbę";
    cin >> n;
```

Pętle – przykład – pętla w pętli – czynniki pierwsze

```
//rozkład liczby naturalnej n na czynniki pierwsze
cout<<"Rozkład liczby "<< n <<
    " na dzielniki pierwsze: \n";
int k = 2; //kolejne cyfry rozkładu
while (n>1)//dopóki są jeszcze jakieś dzielniki
{
    //dzielimy dopóki się da przez k
    while (n%k == 0)//dopóki n się dzieli przez k
    {
        cout << k << " "; //wypisujemy dzielnik
        n /= k; //skracamy liczbę
    }
    //n już nie dzieli się przez k,
    k++; //następny potencjalny dzielnik
}
```

podaj liczbę

56

Rozkład liczby 56 na dzielniki pierwsze:

2 2 2 7

Operator rzutowania

Operator postaci

(nazwa typu) wyrażenie

to **operator rzutowania** (jawnego przekształcania typu).

Jest on operatorem jednoargumentowym: wynikiem działania tego operatora na *wyrażenie* pewnego typu jest odpowiadająca tej wartości wartość innego typu - tego wymienionego w nawiasie.

Z operatora tego należy korzystać tylko w przypadku, gdy naprawdę wiemy, co chcemy zrobić. Nie należy go nadużywać. Czasem jest jednak przydatny. Na przykład w drugiej instrukcji fragmentu

```
double x = 7;
```

```
int k = (int)x;
```

Rzutowanie wartości zmiennej **x** jest wskazane, gdyż wartość ta, jako wartość szerszego typu, może być wpisana do zmiennej typu węższego, jakim jest typ **int**, tylko ze stratą informacji (precyzji). Choć nie jest to błąd, to kompilator zwykle wypisuje ostrzeżenia; jeśli zastosujemy jawne rzutowanie, ostrzeżeń nie będzie.

Pętle – przykład – pętla w pętli - liczby pierwsze

```
//Program wyświetla wszystkie liczby pierwsze
//z danego zakresu
int main()
{
    unsigned int n = 500;
    if (n >= 2)
        cout<< 2 <<" "; // 2 wyświetlamy osobno
    //sprawdzamy wszystkie liczby nieparzyste od 3 do n
    for (int i = 3; i <= n; i += 2)
    {
        bool isPrime = true; // zakładamy że jest
                             // pierwsza
        //aby pokazać, że liczba jest pierwsza wystarczy
        //sprawdzić czy ma jakieś dzielniki od 3 do sqrt(n)
        //obliczamy pierwiastek przed petla
        int p = (int) (sqrt(i)); //rzutowanie na int
        // dla sqrt trzeba: #include<cmath>
```

Pętle – przykład – pętla w pętli - liczby pierwsze

```
//testujemy czy i jest pierwsza:dzielimy i przez
// liczby nieparzyste od 3 do sqrt(i)
    for (int j = 3; j <= p; j += 2)
    {
        if (i % j == 0) //jeśli i ma dzielnik
        {
            isPrime = false;//to nie jest pierwsza
            break; //wychodzimy z pętli
        }
    }
    //albo doszliśmy do sqrt(i) i nie było dzielnika
    //albo (był dzielnik)opuściliśmy pętle z break
    if (isPrime)    cout<< i << ",";
} //end for po nieparzystych z zakresu [3,n]

return 0;
} //main
```


- Sposób formatowania danych w operacjach wyjścia można zmienić. Służą do tego **flagi (znaczniki) formatowania** (*format flags*) i **manipulatory**.
- **Manipulatory** są to funkcje zdefiniowane w klasie `ios` i wywoływane poprzez podanie ich nazw jako elementów wstawianych do lub wyjmowanych ze strumienia.

Manipulatory z argumentami

- **setw(int szer)** — ustawia szerokość pola dla najbliższej operacji na strumieniu, przy czym określana jest minimalna szerokość pola: jeśli dana zajmuje więcej znaków, to odpowiednie pole zostanie zwiększone. Domyślną wartością szerokości pola jest zero, czyli każda wypisywana dana zajmie tyle znaków, ile jest potrzebne, ale nie więcej

```
int e = 1, f = 10;
cout<< "-----" << endl
    << setw(4) << e
    << setw(4) << f << endl;
cout<< left << setw(4) << e
    << setw(4) << f << endl << "-----";
```

```
-----
      1   10
1      10
-----
```

Pętle for- przykład- tabliczka mnożenia

*/*Wyświetlić na ekranie tabliczkę mnożenia wymiaru $n \times n$. Pobrać od użytkownika n , $0 < n < 15$. */*

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main () {
```

```
    int n;
```

```
    //pobieramy n w pętli zaporowej
```

```
    do{
```

```
        cout<<"podaj liczbę ";
```

```
        cin>>n;
```

```
    }while(n<=0 || n>=15); //(!(n>0 && n<15))
```

Pętle for- przykład- tabliczka mnożenia

*/*tabliczka mnożenia, dla n=3*

1 2 3

2 4 6

*3 6 9 */*

//mamy wiersze od 1 do n

for(**int** i =1; i<=n; i++)

{ *//jesteśmy w i tym wierszu*

//w każdym wierszu mamy n elementów

for(**int** j=1; j<=n; j++)

 cout << setw(4) << i*j;

//po wydrukowaniu wiersza znak nowej linii

 cout<<endl;

}

//wstawiamy puste linie przed kolejną częścią

cout<<endl<<endl;

Pętle for- przykład- trójkąt – pętle zagnieżdżone

*/*wyświetlić trójkąt prostokątny o wysokości n postaci:*

dla n=4

** dla i=1 1 **

*** dla i=2 2 **

**** dla i=3 3 **

***** dla i=4 4 **

*w wierszu i drukujemy i * */*

```
for(int i = 1; i <= n; i++) // mamy n wierszy  
{
```

//jesteśmy w i tym wierszu, drukujemy i gwiazdek

```
    for(int j = 1; j <= i; j++)
```

```
        cout << '*';
```

//po wyświetleniu wiersza znak nowej linii

```
    cout << endl;
```

```
}
```

```
cout << endl << endl; //dodatkowe puste linie
```