

Programowanie w C++

Opracował:
Henryk Szantula

1. Definicja problemu
2. Analiza wymagań i znalezienie odpowiedniej metody rozwiązania (stworzenie algorytmu)
3. Implementacja programu
4. Uruchomienie i testowanie programu
5. Konserwacja programu

***Proces wytwarzania programu**

Kod maszynowy

Kod maszynowy to jedyny język zrozumiały dla procesora. Polecenia i ich atrybuty są w tym kodzie reprezentowane poprzez sekwencje liczbowe. Każdy typ procesora ma swój własny kod maszynowy wraz z listą instrukcji, które rozumie, zatem programy napisane na jeden typ procesora nie mogą być w łatwy sposób przeniesione na inny.

Generacje języków programowania

- * I generacja: kod maszynowy
- * II generacja: język symboliczny niskiego poziomu (assembler)
- * III generacja: język wysokiego poziomu (ALGOL, BASIC, Pascal, C, C++, Java)
- * IV generacja: podejście obiektowe, interfejs bazodanowy, GUI
- * V generacja: języki sztucznej inteligencji

Im wyższa generacja tym język programowania jest bardziej podobny do języka naturalnego.

*Podstawowe typy danych

Każda zmienna ma swój typ, który musi być zdefiniowany przez programistę przed pierwszym użyciem.

Typ	Rozmiar w bajtach	Zakres
Char	1	-128...128
Unsigned char	1	0...256
Int	2 lub 4	-32 768 ... 32 767
Unsigned int	2 lub 4	0 .. 65 535
Short int	2	-32 768 ... 32 767
Unsigned short int	2	0 .. 65 535
Long int	4	-2 147 483 648 ... 2 147 483 647
Unsigned long int	4	0 ... 4 294 967 295
Float	4	Sześć cyfr znaczących w zapisie
Double	8	Dziesięć cyfr znaczących w zapisie
bool	1	True, false
String		Stała tekstowa

The background of the image features a series of concentric circles in shades of light blue and white, centered in the upper half of the frame. The circles are semi-transparent and fade out towards the edges of the image.

*ideone.com

Budowa programu

```
1. #include <iostream>
2. #include <cstdlib>
3. using namespace std;
4. main()
5. {
6.     cout<<"To jest moj pierwszy program";
7.     system("pause");
8. }
```

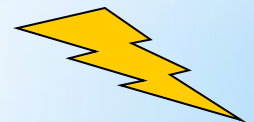
Plik biblioteczny

Dyrektywa
preprocesora

Przestrzeń nazw
jest zatem
zbiorem
obiektów, która
ogranicza dostęp
do nich

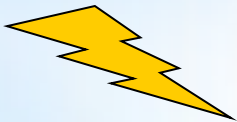
String- stała tekstowa-ciąg znaków ujęty w cudzysłów

Preprocessor- program dokonujący wstępnej obróbki kodu źródłowego przed kompilacją



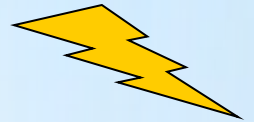
To samo inaczej

```
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
    cout<<"To jest ";
    cout<<"moj pierwszy";
    cout<<" program";
    system("pause");
}
```



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

main()
{
    cout<<"To jest moj"<<" pierwszy"<<" program";
    system("pause");
}
```



Manipulatory

Manipulatorami nazywamy specjalne funkcje, które włączone do wyrażeń związanych z operacjami wejścia/wyjścia zmieniają sposób formatowania tekstu

Manipulator	Opis
<i>endl</i>	Znak nowej linii
<i>\n</i>	Znak nowej linii
<i>\t</i>	Tabulator poziomy
<i>\r</i>	Powrót na początek wiersza
<i>\b</i>	Cofnięcie o jeden znak

*Zastosowanie manipulatorów

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

```
main()  
{  
    cout<<"To jest \n";  
    cout<<"moj pierwszy"<<endl;  
    cout<<" program";  
    system("pause");  
}
```



*Komentarze

```
#include <iostream>
```

```
#include <cstdlib>
```

```
/* to jest komentarz, mogę tu np opisać co dany  
fragment kodu wykonuje */
```

```
using namespace std;
```

```
main()
```

```
{
```

```
    cout<<"Witaj !!"<<endl;           // a tutaj pojawi się znak nowej linii
```

```
    cout<<"w programie";
```

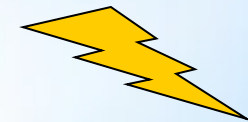
```
    system("pause");
```

```
}
```

Komentarze czyli opisy które kompilator ignoruje

`/* komentarz */` lub

`// komentarz`



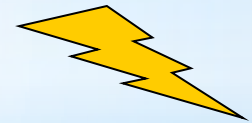
Deklaracja zmiennych

Deklaracja zmiennej to nadanie zmiennej nazwy, określenie typu wartości, jaka będzie w niej przechowywana, oraz rezerwacja w pamięci miejsca, w którym ta wartość będzie przechowywana.

Poprawne nazwy zmiennych	Błędne nazwy zmiennych
_7krasnali	7krasnali
Main	main
Ala	Ala!
Liczba_m	Liczba m

Zastosowanie zmiennych

```
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
    cout<<"ile masz lat ?";
    int lata;                //deklaracja zmiennej
    cin>>lata;               //pobranie z klawiatury
    cout<<"Już wiem: masz "<<lata<<" lat,,<<endl;
    system("pause");
}
```



Każda nazwa zmiennej użyta w C++ musi zostać zadeklarowana przed jej pierwszym użyciem.

Błędy syntaktyczne

Błąd syntaktyczny polega na naruszeniu reguł opisujących składnię języka. Zrobienie błędu syntaktycznego spowoduje, że kompilator nie wygeneruje pliku z rozszerzeniem .exe

Najczęstsze błędy:

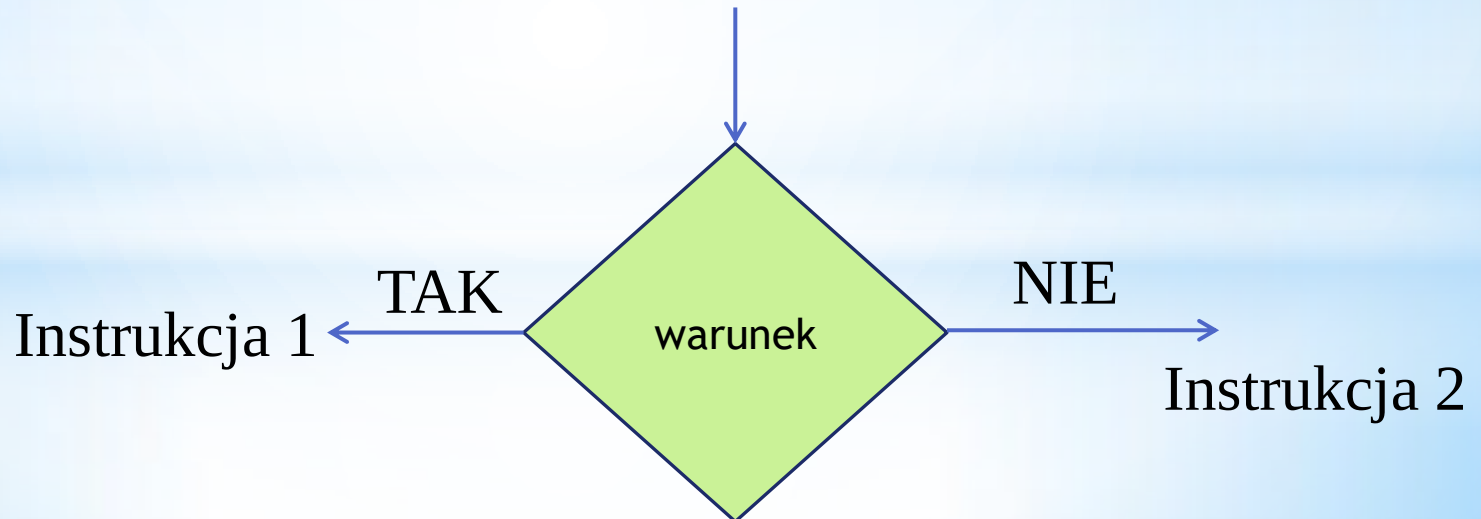
- *Pominięcie średnika
- *Użycie niezadeklarowanej zmiennej
- *Pominięcie klamry kończącej program
- *Zniekształcenie słowa kluczowego lub zmiennej
- *Pominięcie słowa kluczowego w instrukcji

Operatory logiczne i symbole porównawcze

Operator	Symbol porównawczy	Działanie
&&		Koniunkcja (I)
		Alternatywa (LUB)
!		negacja
	==	równy
	!=	różny
	<	mniej
	>	wiekszy
	>=	wiekszy lub równy
	<=	Mniejszy lub równy

Instrukcje sterujące

Instrukcje sterujące kierują przebiegiem wykonywania programu. Pozwalają na zmianę kolejności wykonywania innych instrukcji zależnie od otrzymanych wyników. Bez instrukcji sterujących moglibyśmy jedynie pisać programy liniowe.



Instrukcja warunkowa

if...else

Składnia instrukcji:

If (wyrażenie) instrukcja

Lub

If (wyrażenie) instrukcja 1

Else instrukcja 2

Wyrażenie może przyjąć jedną z dwóch wartości logicznych prawda lub fałsz (true, false) W miejscu wyrażenia może także znajdować się zmienna np.:

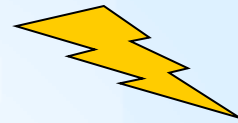
$A = 6$

$C + 12 * d < g$

$A \neq g$

Przykład instrukcji warunkowej

```
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
    int liczba;          //deklaracja zmiennej
    cout<<"podaj liczbę";
    cin>>liczba;         //pobranie z klawiatury
    if (liczba>0)
        cout<<"liczba jest dodatnia";
    else
        cout<<"Liczba jest ujemna";
    system("pause");
}
```

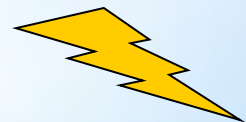


Blok instrukcji

Skończoną liczbę instrukcji pojedynczych ujętych w klamry { } nazywamy instrukcją złożoną lub inaczej blokiem instrukcji.

```
main()
{
    int dlugosc;      //deklaracja zmiennej
    cout<<"podaj liczbę";
    cin>>dlugosc;     //pobranie z klawiatury
    if (dlugosc>0)
    {
        cout<<endl;
        cout<<"podałeś dodatnią liczbę\n"<<endl;
        cout<<"dobrze bo to ma być długość odcinka\n";
    }
```

```
else
    cout<<"Liczba jest ujemna\n";
    cout<<"koniec działania";
    system("pause");
}
```

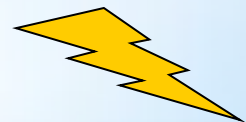


Blok instrukcji

W bloku *else* wprowadzmy zmianę dodając nawias klamrowy `{ }` i zaobserwuj zmiany

```
main()
{
    int dlugosc;      //deklaracja zmiennej
    cout<<"podaj liczbę";
    cin>>dlugosc;     //pobranie z klawiatury
    if (dlugosc>0)
    {
        cout<<endl;
        cout<<"podałeś dodatnią liczbę\n"<<endl;
        cout<<"dobrze bo to ma być długość odcinka\n";
    }
```

```
Else
{
    cout<<"Liczba jest ujemna\n";
    cout<<"koniec działania";
}
    system("pause");
}
```

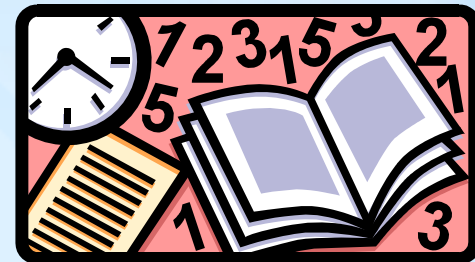


Składnia instrukcji
wzbogaconej o drugi
warunek (wyrażenie)

```
if (wyrażenie)
{
Instrukcja
}
else if(wyrażenie)
{
Instrukcja2
}
Else
{
Instrukcja 3
}
```

Rozbuduj poprzedni przykład, o możliwość rozpoznawania zera

*Zadanie



Napisz program wyprowadzający na ekran monitora
najmniejszą z trzech podanych liczb

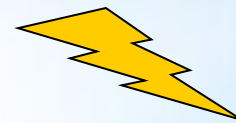
10 minut wystarczy ?

```
podaj pierwsza liczbe 5
podaj druga liczbe 3
podaj trzecia liczbe 4
najmniejsza z podanych liczb jest 3
-----
Process exited after 9.527 seconds with return value 0
Press any key to continue . . .
```

* Rozwiązanie

```
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
    int a,b,c;
    cout<<"podaj pierwszą liczbę";
    cin>>a;
    cout<<"podaj drugą liczbę";
    cin>>b;
    cout<<"podaj trzecią liczbę";
    cin>>c;
    if (a<b)
        if (a<c)
            cout<<"najmniejszą z podanych liczb jest"<<a;
        else
```

```
            cout<<"najmniejszą z podanych liczb jest"<<c;
        else
            if (c<b)
                cout<<"najmniejszą z podanych liczb
jest"<<c;
            else
                cout<<"najmniejszą z podanych liczb
jest"<<b;
            system("pause");
}
```



Instrukcja wyboru *switch*

W programie może się zdarzyć, że należy wybrać jeden z wielu sposobów postępowania, zależnych od wartości zmiennej

Składnia instrukcji *switch*

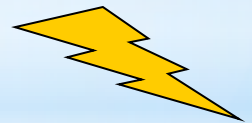
Switch (wyrażenie)

```
{  
  case wartość1:instrukcja1;break;  
  case wartość2:instrukcja2;break;  
  .  
  .  
  default:instrukcja_inna;break;  
}
```

Instrukcję *switch*
można stosować
również bez
instrukcji *break*

Przykład instrukcji *switch*

```
main()
{
    int lekcja;
    cout<<"Która godzina lekcyjna się zaczęła ?\t";
    cin>>lekcja;
    switch (lekcja)
    {
        case 1: cout<<"masz teraz 1 lekcje";break;
        case 2: cout<<"masz teraz 2 lekcje";break;
        case 3: cout<<"masz teraz 3 lekcje";break;
        case 4: cout<<"masz teraz 4 lekcje";break;
        case 5: cout<<"masz teraz 5 lekcje";break;
        default:cout<<"jesteś już po lekcjach";break;
    }
    system("pause");
}
```



Sprawdź jak działa bez *break*

*Instrukcja pętli *for*

Składnia pętli

for (instrukcja początkowa ; warunek sterujący; instrukcja kroku)

Instrukcja;

Gdzie:

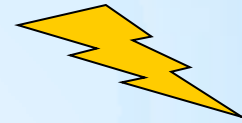
Instrukcja początkowa- instrukcja inicjalizująca

Warunek sterujący- wyrażenie którego logiczna wartość jest badana przed każdym obiegiem pętli

Instrukcja kroku- instrukcja wykonana po każdym przebiegu pętli

* Przykład pętli *for*

```
main()
{
    for (int i=0; i<10; i++)
    {
        cout<<i<<" ";
    }
    system("pause");
}
```



Instrukcja inkrementacji c++

i++ odpowiada *i=i+1*

Instrukcja dekrementacji c++

i-- odpowiada *i=i-1*

*Zadanie

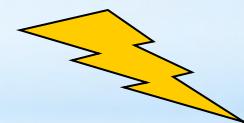
Napisz program który wyświetla w kolumnie liczby od 1 do 20, ale to jest zbyt proste. Uzupełnij program tak aby przy liczbach niepodzielnych przez 3 wstawił odpowiedni komentarz ;-)

Podpowiedź

Funkcja *mod* w c++ to %

*Rozwiązanie

```
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
    for (int i=0; i<20; i++)
    {
        cout<<i;
        if (i%3!=0)
            cout<<" - nie jest podzielne przez 3";
        cout<<endl;
    }
    system("pause");
}
```



*Instrukcja pętli *while*

Składnia

While(wyrażenie) instrukcja

Gdzie:

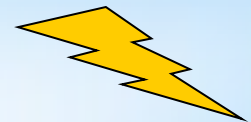
Wyrażenie- przyjmuje wartość prawdy lub fałszu

Instrukcja- instrukcja pojedyncza lub blok instrukcji

* Przykład instrukcji *while*

```
main()
{
    char znak;
    cout<<"podaj znak: ";
    cin>>znak;
    while (znak!='k')
    {
        cout<<"podaj kolejny znak: ";
        cin>>znak;
    }
    cout<<"Podales "<<znak<<" wiec koncze";
    system("pause");
}
```

Program działa
dopóki
użytkownik nie
wprowadzi „k”



Napisz program, który oblicza wynik dzielenia całkowitego dwóch liczb podanych z klawiatury. Zakładamy, że komputer nie zna działania dzielenia i odejmuje kolejno dzielną od dzielnika działając w pętli.

 **Zadanie**

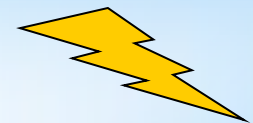
*Instrukcja pętli *do....while*

Składnia:

Do instrukcja *while* (wyrażenie)

Zacznijmy od prostego programu rysującego
rysującego cztery gwiazdki na ekranie

```
main()
{
    int i=4;
    do
    {
        cout<<"*";
        i--;
    }
    while (i);
    system("pause");
}
```



1. Narysuj w zeszycie algorytm dla fragmentu programu:

```
{a=0;  
do  
{  
cout<<„informatyka”;  
a=a+1;  
}  
while(a!=0);}
```

*Zadania

2. Ile razy w konsoli zostanie wydrukowany napis „informatyka”
3. Napisz program obliczający pole kwadratu o podanej długości boku. W przypadku podania 0 lub wartości ujemnej program prosi o wprowadzenie poprawnych danych.
4. Uzupełnij program o możliwość rysowania kwadratu z dowolnego znaku wprowadzonego przez użytkownika.

Powodzenia ;-)

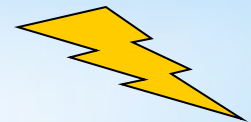
*Instrukcja *break*

```
main()
{
    int sum=0;
    for (int i=0;i<10;i++)
    {
        cout<<2*i;
        sum=sum+2*i;
        if (sum>20) break;
    }
    cout<<"jestem poza petla";
    system("pause");
}
```

Program wyświetla liczby parzyste dopóki ich suma nie przekroczy 20

// jeśli warunek spełniony

// wyskok z pętli



*Instrukcja *continue*

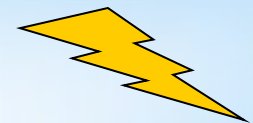
Po napotkaniu
instrukcji *continue*
część poleceń
zostaje pominięta

```
main()
{
    for(int i=0; i<5; i++)
    {
        cout<<i<< ", ";
        if(i%3==0) continue;
        cout<<"- nie jest podzielne przez 3, \n";
    }
    system("pause");
}
```

// przy spełnieniu warunku

// przechodzimy natychmiast do kolejnego przebiegu pętli

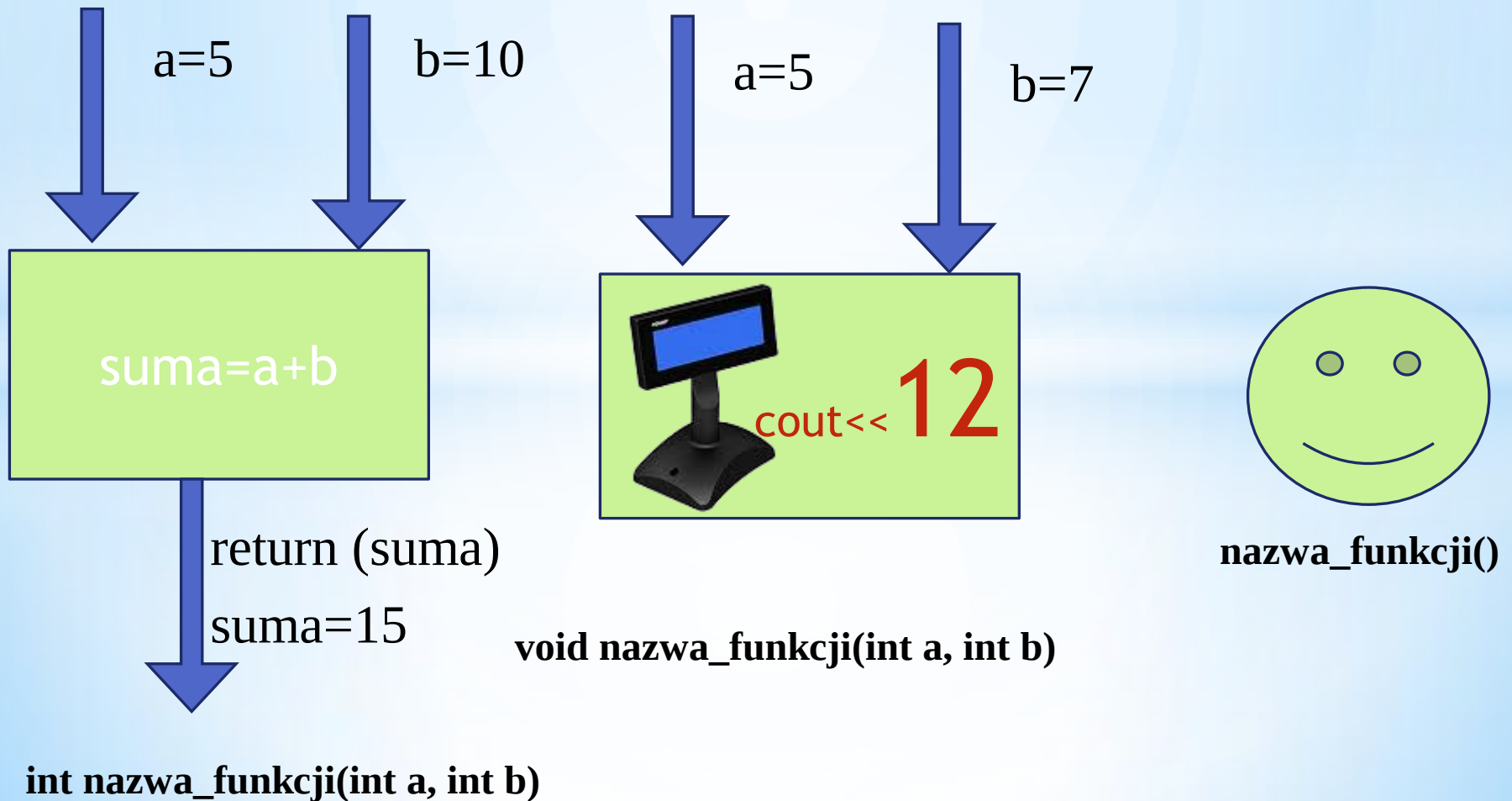
Program przy liczbach
niepodzielnych przez 3
wypisuje odpowiedni
komunikat



*Zadania kontrolne

- * Napisz program, który pobiera dane z klawiatury dopóki suma wprowadzonych liczb nie przekroczy 50
- * Napisz program, który na ekranie monitora wyświetli wszystkie liczby dwucyfrowe parzyste podzielne przez 3
- * Napisz program, który wyświetli kwadraty kolejnych liczb naturalnych skończywszy na liczbie podanej przez użytkownika
- * Napisz program, który obliczy pierwiastki równania kwadratowego
- * Napisz program, który wyświetli podany element ciągu Fibonachiego.
- * Napisz program, który narysuje koperte z „*” na ekranie

Funkcje w C++



* Definiowanie i wywołanie funkcji

- * Każdy program napisany w C++ musi mieć przynajmniej jedną funkcję *main()*
- * Każda funkcja zdefiniowana musi mieć nazwę
- * Funkcja może ale nie musi pobierać wartości z zewnątrz, podobnie nie musi zwracać wartości

*Definicji funkcji

Składnia

typ_zwracanego_wyniku nazwa funkcji (zestaw
parametrów)

{

wnętrze funkcji;

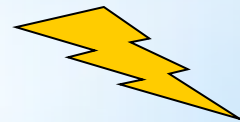
}

*Przykład funkcji

```
float srednia_aryt (float a, float b)
{
    return (a+b)/2;
}
```

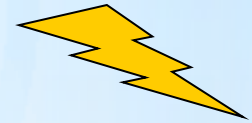
```
main()
{
    cout<<"srednia arytmetyczna liczb 2.7 i 5=" << srednia_aryt(2.7,5);
    system(„pause”);
}
```

Funkcja liczy
średnią
arytmetyczną
z 2,7 i 5



* Zakres zmiennych

```
float srednia_aryt (float a, float b)
{
    return (a+b)/2;
}
```



```
main()
{
    float x,y;
    cout<<"Podaj dwie liczby";
    cin>>x>>y;
    cout<<srednia_aryt(x,y);
    system("pause");
}
```

// zmienne lokalne funkcji



Zmienne, które są zadeklarowane we wnętrzu funkcji, to zmienne lokalne funkcji.

Nie są one znane poza wnętrzem funkcji. We wnętrzu funkcji zmienna jest znana od momentu jej deklaracji do klamry zamykającej blok, w którym zmienna została zadeklarowana.

*Przykład zmiennych

```
int a; //zmienna globalna
void wyswietl ()
{
    cout<<"bok kwadratu ma dlugosc :"<<a;
}
float pole_kwadratu()
{
    return a*a;
}
float b; // ta zmienna zna pole_prostokata() i main()
float pole_prostokata ()
{
    int pole=a*a;
    return pole; //zmienna lokalna funkcji
}
main()
{
    int c;
}
```

* Przykład 1

```
void szlaczek_dowolny(int i, char znak)
```

```
{
```

```
    for (int j=0; j<i; j++)
```

```
        cout<<znak;
```

```
}
```

```
main()
```

```
{
```

```
int a;
```

```
char b;
```

```
cout<<"podaj liczbe znaków\t";
```

```
cin>> a;
```

```
cout<<"podaj znak\t";
```

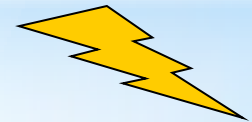
```
cin>> b;
```

```
szlaczek_dowolny(a,b);
```

```
system("pause");
```

```
}
```

**Funkcja rysuje
szlaczek znakiem
przekazanym
funkcji**



*Przykład 2

```
void szlaczek_10_gwiazdek (void)
{
    for (int p=0; p<10; p++)
        cout<<„*”;
}
```

Nie przekazujemy funkcji żadnych parametrów, i funkcja nie zwraca żadnych wartości- **funkcja bezparametrowa**

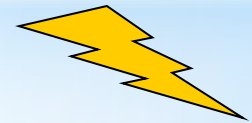
Równoważne są zapisy:

`nazwa_funkcji()` **oraz** `void nazwa_funkcji(void)`

*Przykład 3

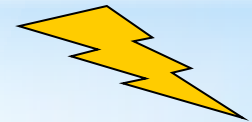
```
float maks (float a, float b, float c)
{
    float m=a;
    if (b>m)
        {
            m=b;
        }
    if (c>m)
        {
            m=c;
        }
    return m;
}
```

**Funkcja zwraca
największą z trzech
wprowadzonych liczb**



```
void zamien_miejscami(int&x, int&y)
{
    int temp=x;
    x=y;
    y=temp;
}
main()
{
    int a,b,c;
    cout<<"Podaj wartosci a,b,c";
    cin>>a>>b>>c;
    if (a>b) zamien_miejscami(a,b);
    if (a>c) zamien_miejscami(a,c);
    if (b>c) zamien_miejscami(b,c);
    cout<<"wartosci a,b,c"<<a<<b<<c;
    system("pause");
}
```

Program
porządkujący 3
liczby rosnąco



*Ćwiczenia

- * Napisz funkcję, która pobiera dwa znaki z klawiatury i na ich podstawie rysuje szlaczek z 30 elementów
- * Napisz program liczący objętość sześcianu, zabezpiecz się przed wartościami ujemnymi, wykorzystaj funkcje
- * Napisz program pobierający z klawiatury dwie liczby całkowite i pytający użytkownika o ich iloczyn, w przypadku podania złej wartości program pyta ponownie dopóki nie uzyska poprawnej odpowiedzi. Zastosuj funkcje.

Tablice zmiennych

Tablicą nazywamy ciąg ustalonej liczby elementów tego samego typu, do których odwołujemy się za pośrednictwem wspólnej nazwy. Dostęp do konkretnego elementu tablicy uzyskuje się za pomocą nazwy i indeksu tablicy.

Deklaracja tablicy jednowymiarowej

Typ_elementów nazwa_tablicy [liczba_elementów]

Przykład

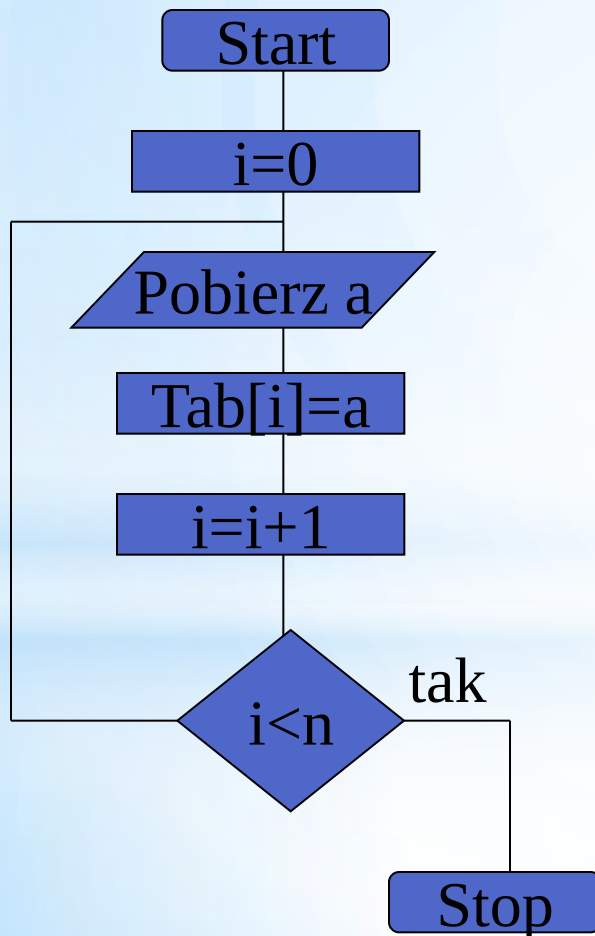
int oceny[8]

Tak zadeklarowana tablica wypełniona jest losowymi wartościami lub

int oceny[]={5,3,3,6,1,2,4,5}

Gdzie wprowadzamy zadane wartości (nie podajemy liczby elementów)

Wypełnianie tablicy



Odwołanie się do elementu o indeksie innym niż *liczba_elementów* nazywamy **przekroczeniem zakresu tablicy** (nie określone „n” w algorytmie)

* Wypełnianie tablicy jednowymiarowej

```
void wypelnij (int tab[8])
```

```
{  
    for (int i=0; i<8; i++)  
    {  
        cout<<"podaj wartosc elementu";  
        cin>>tab[i];  
    }  
}
```

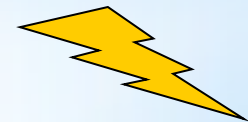
```
void wyswietl(int tab[8])
```

```
{  
    for (int i=0;i<8;i++)  
        cout<<tab[i]<<" ";  
    cout<<endl;  
}
```

Funkcja główna programu

```
main()
```

```
{  
    int tablica1[8],tablica2[8];  
    wypelnij(tablica1);  
    wyswietl(tablica1);  
    system("pause");  
}
```



* Wypełnianie tablicy przy pomocy funkcji

Przekazując funkcji tablicę jako parametr aktualny, podajemy tylko i wyłącznie jej nazwę bez nawiasów kwadratowych i bez podania rozmiaru

```
void wypelnij (int tab[],int rozmiar)
{
    for (int i=0; i<rozmiar; i++)
    {
        cout<<"podaj wartosc elementu";
        cin>>tab[i];
    }
}
```

```
void wyswietl(int tab[],int rozmiar)
{
    for (int i=0;i<rozmiar;i++)
        cout<<tab[i]<<" ";
        cout<<endl;
}

main()
{
    int tablica1[10];
    wypelnij(tablica1,10);
    wyswietl(tablica1,10);
    system("pause");
}
```

*Tablice wielowymiarowe

indeks	0	1	2
0	4	5	2
1	6	88	9
2	4	3	10

Deklaracja *tabl[w][k]*
oznacza deklarację
tablicy o nazwie *tabl*;
litera *w* oznacza liczbę
wierszy tablicy, litera *k*
- liczbę kolumn, gdzie *w*
i *k* są stałymi
całkowitymi.

Wypełnić taką tablicę możemy stosując
„pętlę w pętli” zwaną pętlą
zagnieżdżoną

Wypełnianie tablicy wielowymiarowej

```
void wypelnij(float tab[5][7])
```

```
{
    for (int i=0;i<5;i++)
    {
        for (int j=0;j<7;j++)
        {
            cout<<"podaj wartosc elementu:";
            cin>>tab[i][j];
        }
    }
}
```

```
main()
```

```
{
    float tab1[5][7];
    wypelnij(tab1);
    cout<<endl;
    wyswietl(tab1);
    system("pause");
}
```

```
void wyswietl(float tab[5][7])
```

```
{
    for (int i=0;i<5;i++)
    {
        for (int j=0;j<7;j++)
        {
            cout<<tab[i][j]<<",";
        }
        cout<<endl;
    }
}
```

Ćwiczenia

Dana jest tablica dwuwymiarowa oraz dwie liczby a oraz r .
Wypełnij kolejne elementy tablicy liczbami: pierwszy element liczbą a , każdy następny sumą wartości poprzedniego elementu i liczby r .

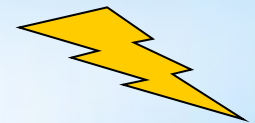
2	5	8	11
14	17	20	23
26	29	32	35
38	41	44	47

Dla $a=2$ i $r=3$

```
void wypelnij(float tab[4][4],float a,  
float r)  
{  
    for (int i=0;i<4;i++)  
    {  
        for (int j=0;j<4;j++)  
        {  
            tab[i][j]=a;  
            a=a+r;  
        }  
    }  
}
```

Ćwiczenia

Opracuj program, który znajduje średnią ocen, zapisanych w jednowymiarowej dwunastoelementowej tablicy, wyświetla tę średnią, a następnie podaje wszystkie oceny, z których obliczył średnią.



Klasyczne algorytmy działające na tablicach

- * Przeszukiwanie liniowe tablicy jednowymiarowej
- * Przeszukiwanie liniowe tablicy jednowymiarowej z wartownikiem
- * Poszukiwanie elementu maksymalnego (minimalnego) w tablicy
- * Zamiana liczb na inny system liczbowy
- * Sito Eratostenesa

*Przeszukiwanie liniowe tablicy jednowymiarowej

Problem

Szukamy wyróżnionego elementu w jednowymiarowej tablicy. Zakładamy, że szukany element może się w tablicy znajdować lub nie.

Lista kroków

1. Pobierz tablicę n -elementową, pobierz wartość elementu szukanego *szuk*
2. W miejsce zmiennej pomocniczej i podstaw 0
3. Jeśli $i < n$, przejdź do kroku 4, w przeciwnym wypadku wypisz „element nie został znaleziony” i zakończ.
4. Jeśli $tab[i]$ jest różne od *szuk*, zwiększ i o jeden i przejdź do kroku 3
5. Wypisz „element został znaleziony” i zakończ

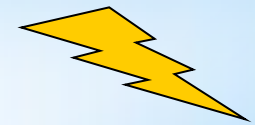
* Przeszukiwanie tablicy metodą element po elemencie nazywamy przeszukiwaniem liniowym

```
void wypelnij(int tab[10])
{
    for (int i=0;i<10;i++)
        tab[i]=rand()%21;
}

void szukaj (int tablica[10], int szukany)
{
    int i=0;
    while (i<10 && tablica[i]!=szukany)
    {
        i++;
    }
    if (i==10) cout <<"Element nie zostal
znaleziony";
    else cout<<"Element zostal znaleziony";
}
```

```
main(){
    int tab[10];
    int szuk;
    srand (time(NULL)); //inicjalizacja
generatora liczb losowych
    wypelnij(tab);
    wyswietl(tab);
    cout<<endl;
    cout<<"Podaj element szukany";
    cin>>szuk;
    szukaj(tab,szuk);
    system("pause");}
```

```
void wyswietl(int tab[10])
{
    for (int i=0;i<10;i++)
        cout<<tab[i]<<" ";
}
```



*Analiza

```
while (i<10 && tablica[i]!=szukany)
{
    i++;
}
```



Policzmy ile w tym algorytmie wykonuje się operacji **dominujących** czyli mających wpływ na szybkość działania. Dla n elementów tablicy wykonujemy w najgorszym przypadku $2n$ porównań. **Złożoność obliczeniowa** tego algorytmu jest więc **złożonością liniową**.

*Przeszukiwanie liniowe tablicy jednowymiarowej z wartownikiem

Dodajmy do tablicy element dodatkowy o poszukiwanej wartości i ustawmy go na końcu. Ten właśnie element nazywamy **wartownikiem**. W tak zmodyfikowanej tablicy zawsze znajdziemy element szukany, mamy więc pewność, że algorytm **nie przekroczy zakresu tablicy**.



```
void szukaj (int tablica[10], int szukany)
{
    int i=0;
    tablica[10]=szukany;

    while (tablica[i]!=szukany)
    {
        i++;
    }
    if (i<10) cout <<"Element został
znaleziony";
    else cout<<"Element nie został
znaleziony";
}
```

W ten sposób algorytm wykona n operacji porównań oraz jedno dodatkowe (sprawdzenie wartownika), lecz dalej jest to złożoność liniowa i klasa złożoności tego algorytmu nie zmieniła się.



Aby móc wykorzystać metodę z wartownikiem, musisz użyć tablicy o liczbie elementów o jeden większej niż liczba elementów do sprawdzenia.

* Poszukiwanie Elementu maksymalnego (minimalnego) w tablicy.

Problem

Dana jest tablica n-elementowa. Chcemy w niej znaleźć element maksymalny.

Specyfikacja problemu algorytmicznego

Problem: znalezienie elementu maksymalnego w tablicy

Dane wej.: tablica $tab[n]$, $tab[i] \in \mathbb{R}$; n to rozmiar tablicy, $n > 0$

Dane wyj.: max - element maksymalny tablicy

Zmienne pom. i zmienna licznikowa

*Rozwiązanie szukania (max)

Lista kroków

1. W miejsce zmiennej i podstaw 0
2. W miejsce max podstaw tab[i]
3. Zwiększ i o 1
4. Jeśli $i < n$, to krok 6
5. Wypisz max i zakończ
6. Jeśli $tab[i] > max$ to w miejsce max podstaw tab[i]
7. Przejdź do kroku 3

```
int maksymalny(int tab[10])  
{  
    int max=tab[0];  
    for (int i=1;i<10;i++)  
        if (tab[i]>max) max=tab[i];  
    return max;  
}
```

*Zamiana liczb na inny system liczbowy

Liczba dziesiętna to liczba o podstawie liczenia 10, liczba dwójkowa jest liczbą o podstawie liczenia dwa, a szesnastkowa postać to liczba o podstawie liczenia szesnaście.

Zamieńmy liczbę 47 z zapisu w systemie dziesiętnym na zapis w systemie o podstawie 3:

$$47:3=15, \text{ reszta } 2$$

$$15:3=5, \text{ reszta } 0$$

$$5:3=1 \text{ reszta } 2$$

$$1:3=0 \text{ reszta } 1$$

Liczba 47 w systemie trójkowym ma postać 1202

*Przykład

Dana jest liczba dziesiętna. Zmieńmy ją na liczbę zapisaną w systemie o podstawie n , gdzie $n \in \{2, \dots, 9\}$.



Zamiana na postać dwójkową

```
void zamiana_na_postac_dwojkowa(int x)
```

```
{  
  int t[8],i=0;
```

```
  do
```

```
  {
```

```
    t[i]=x%2;
```

```
    x=x/2;
```

```
    i++;
```

```
  }
```

```
  while(x!=0);
```

```
  // wyświetl w odwrotnej kolejności
```

```
  while (i>0)
```

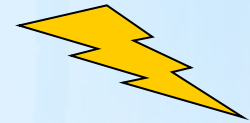
```
  {
```

```
    i --;
```

```
    cout<<t[i]<<" ";
```

```
  }
```

```
}
```



Funkcję masz gotową, zastosuj ją w programie ;-)

I dalejzmodyfikuj funkcję tak by liczyła w dowolnym systemie do dziesiętnego

(podpowiadam)

void zamiana (int x, int podstawa)

Z dowolnego systemu na dziesiętny

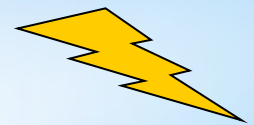
Każdą liczbę zapisaną w dowolnym systemie zamieniamy na system dziesiętny, stosując tę samą metodę, tylko cyfry informują nas wówczas o ilości kolejnych potęg podstawy liczenia, a nie liczby dziesięć.

Przykład:

$$101101_{(2)} = 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 45$$

$$210_{(3)} = 2*3^2 + 1*3^1 + 0*3^0 = 21$$

$$1A2_{(16)} = 1*16^2 + 10*16^1 + 2*16^0 = 418$$



Tak więc już wiesz jak napisać program w c++ ;-)

Rozwiązanie

```
int zamien_na_dziesietna (int tab[], int cyfry)
```

```
{  
    int i,j,k;  
    int liczba_dziesietna =0;  
        for (j=0; j<cyfry; j++)  
        {  
            i=1;  
                for(k=j+1; k<cyfry; k++) i=i*2;  
                    liczba_dziesietna=liczba_dziesietna+tab[j]*i;  
        }  
    return liczba_dziesietna;  
}
```

```
main ()  
{  
    int binarna[20];  
    int i,ilosc_cyfr;
```

```
    cout<<"Z ilu cyfr bedzie sie skladac liczba?: ";  
    cin>>ilosc_cyfr;  
    cout<<"Podaj liczbe w postaci binarnej; po kazdej cyfrze naciśnij Enter "<<endl;  
    for (i=0;i<ilosc_cyfr;i++) cin>>binarna[i];  
        cout<<"W systemie dziesiętnym to: "<<zamien_na_dziesietna(binarna, ilosc_cyfr);  
        system("pause");  
}
```

13 dziesiętnie to 1 1 0 1 binarnie
Czyli :cyfry=4

for(k=j+1; k<cyfry; k++) i=i*2;

- 1) for(k=1; k<4;k++) i=2*2*2
- 2)for(k=2; k<4;k++) i=2*2
- 3) for(k=3; k<4;k++) i=2
- 4) for(k=4; k<4;k++) i=1

8	4	2	1
1	1	0	1

Sito Eratostenesa

Problem liczb pierwszych jest nam znany. Dla przypomnienia to taka liczba naturalna różna od 1, która nie ma innych naturalnych dzielników prócz 1 i samej siebie.

Przykład:

Dany jest przedział liczbowy $\langle 2, b \rangle$. Znajdziemy i wypiszemy wszystkie liczby pierwsze z tego przedziału

Możemy badać każdą liczbę z podanego zakresu, ale jest to mało optymalne, np. z zakresu $\langle 4, 30 \rangle$ wartość 4 możemy skreślić i wszystkie wielokrotności 4, podobnie wyrzucimy wielokrotności liczb 3 i 5. Takie postępowanie przypomina sito i stąd metoda ta przyjęła nazwę sita Eratostenesa od nazwiska twórcy (grecki matematyk, astronom, fizyk)

Wystarczy badać liczby pierwsze
nie większe od $4\sqrt{N}$ czyli 4

Cd sita

Dla liczb z przedziału $<2, 19>$

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----

Skreślamy wielokrotności 2,3,4

2	3		5		7				11		13				17		19
---	---	--	---	--	---	--	--	--	----	--	----	--	--	--	----	--	----

Jak to zapisać w języku programowania !!!

Proponuję zadeklarować tablicę typu bool (prawda,fałsz), i dla kolejnych indeksów tablicy sprawdzać podzielność przez 2,3,4 wpisując odpowiednio „true” lub „false”.
Jako wynik wyświetlimy tylko elementy z wartością „true”. POWODZENIA

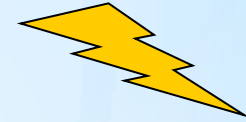
in d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
bo ol	false	false	true	true	false	true	false	true	false	false	false	true	false	true	false	false	false	true	false	true

Rozwiązanie

```
#include <iostream>
#include <cstdlib>
#include <math.h>
```

```
main ()
{
    int i,j,zakres;
    double b;
    bool tablica[100];
    cout<<"Podaj gorny zakres, max 99"<<endl;
    cin>>zakres;
    b = sqrt(zakres);
    for (i=2; i<zakres+1; i++)
        tablica[i]=true;           //czyszcze tablice
        for (i=2; i<=b; i++)       //od 2 do np. 5
            if (tablica[i]!=false) //true czyli pierwsza
                for (j=i+i; j<zakres+1; j=j+i)
                    tablica[j]=false; //ustaw wszystkie wielokrotności na false
```

Sprawdź z większymi
liczbami



```
        cout<<"Liczby pierwsze z zakresu od
        1 do "<<zakres<<" to:"<<endl;
        for (i=2; i<zakres+1; i++)
            if (tablica[i]!=false)
                cout<<i<<" ";
        system("pause");
    }
```

wyrażenie brzmiące tak samo
czytane od lewej do prawej i od
prawej do lewej. Przykładem
palindromu jest: Abba, Ada, Aga,
Ala, anilina, Anna, Ała, apokopa,
Ara, Aviva, Axa, Bob, bób, Elle, gag,
inni, kajak.

```
int main()
{
    string wyraz;
    cout << "Podaj wyraz" << endl;
    cin >> wyraz;
    cout << palindrom(wyraz) << endl << endl;
    system("PAUSE");
    return 0;
}
```

Zobacz więcej:

<http://www.algorytm.edu.pl/biblioteki/string.html>

Palindromy

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

string palindrom(string wyraz)
{
    int dlugosc = wyraz.length();
    for (int i = 0; i < (dlugosc/2); i++)
        if (wyraz[i] != wyraz[dlugosc-i-1])
            return "nie";
    return "tak";
}
```

*Sortowanie tablicy

Sortowaniem nazywamy ustawienie elementów ciągu w ustalonym porządku wg zadanego kryterium (klucza).

Klasyczne algorytmy sortowania

- Sortowanie bąbelkowe
- Sortowanie przez wstawianie
- Sortowanie przez wybór

*Sortowanie bąbelkowe

Nazwa wzięła się stąd, że dane podczas sortowania - tak jak bąbelki w napoju gazowanym - przemieszczają się ku prawej stronie i układają się w odpowiednim szyku.

<https://www.youtube.com/watch?v=4s44rXRdmhQ>

3 2 4 3 1 2 0

Pętla wewnętrzna porówna sąsiadujące ze sobą elementy. Dla

n

elementów zostanie wykonanych

$n - 1$

porównań:

2 3
3 2 4 3 1 2 0

3 4
2 3 4 3 1 2 0

3 4
2 3 4 3 1 2 0

1 4
2 3 3 4 1 2 0

2 4
2 3 3 1 4 2 0

0 4
2 3 3 1 2 4 0

sytuacja po pierwszym cyklu pętli wewnętrznej:

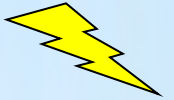
2 3 3 1 2 0 4

Przykładowe rozwiązanie

```
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <time.h>
using namespace std;
const int N = 20; // Liczebność zbioru.

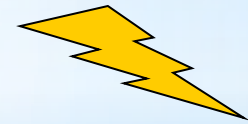
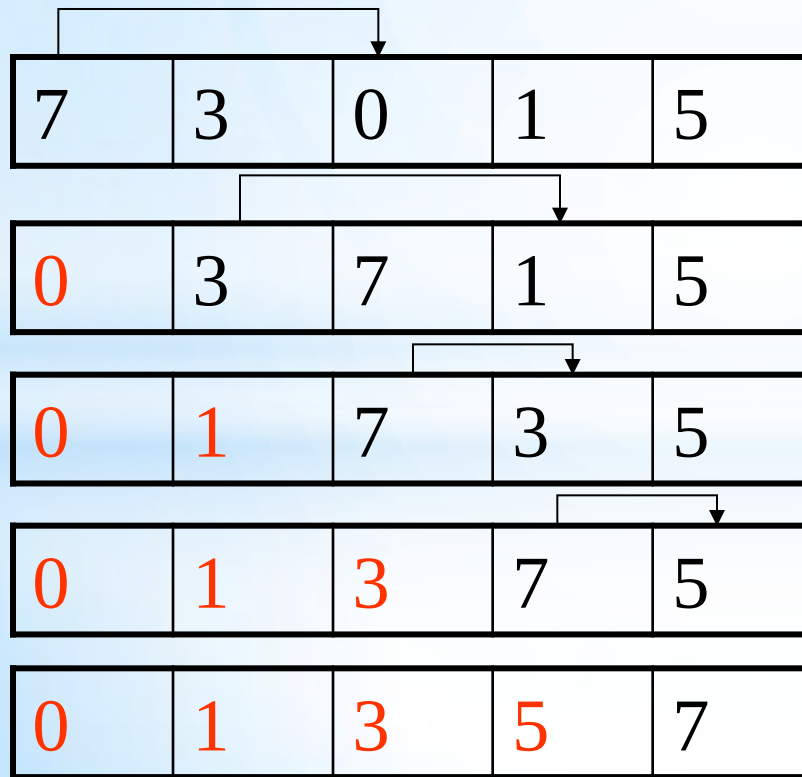
void sortowanie_babelkowe(int tab[],int N)
{
    for(int i=0;i<N;i++)
        for(int j=1;j<N-i;j++) //pętla wewnętrzna
            if(tab[j-1]>tab[j])
                //zamiana miejscami
                swap(tab[j-1], tab[j]);
}
```

```
main()
{
    int d[N],i,j;
    srand(time(NULL));
    for(i = 0; i < N; i++) d[i] = rand() % 100;
    for(i = 0; i < N; i++) cout << setw(4) << d[i];
    cout << endl;
    sortowanie_babelkowe(d,N);
    // Wyświetlamy wynik sortowania
    cout << "Po sortowaniu:\n\n";
    for(i = 0; i < N; i++) cout << setw(4) << d[i];
    cout << endl;
    system("PAUSE");
}
```



*Sortowanie przez wybór

Idea tej metody polega na znajdowaniu w tablicy najmniejszego elementu i zamianą z pierwszym elementem tablicy, Następnie w pozostałym fragmencie tablicy znowu szukamy minimum i zamieniamy z pierwszym elementem (nowego zakresu) i tak dalej do końca..



Przykładowe rozwiązanie (przez wybór)

```
void wybor (int tab[], int N)
{
    int k;
    for (int i=0; i<N; i++)
    {
        k = i;
        for (int j=i+1; j<N; j++) //w tej petli szukamy
                                   //indeksu najmniejszego elementu
            if (tab[j]<tab[k])
                k=j;
        swap(tab[k],tab[i]);
    }
}
```

Przykładowe rozwiązanie (przez wybór) cd..

```
void wypisz (int tab[], int n)
{
    for (int i=0; i < n; i++)
        cout<<setw (3)<<tab[i];
    cout<<endl;
}

main ()
{
    int tablica[20];
    int i;
    srand (time (NULL));
    for (i=0; i < 20; i++)
        tablica [i] = rand ()%100;
    cout<<"Tablica wypelniona losowo liczbami z przedzialu <0,99>:"<<endl;
    wypisz (tablica, 20);
    cout<<"Posortowana tablica:"<<endl;
    sortowanie_przez_wybor (tablica, 20);
    wypisz (tablica, 20);
    system("pause");
}
```

*Sortowanie przez wstawianie

Rozważmy jakiś przykład. Weźmy następujący ciąg liczb do posortowania:

309

Możemy sobie wyobrazić, że liczby po prawej stronie to nasze karty na stole, które będziemy po kolei brać, a te, które są po lewej (na razie nie mamy żadnej) to karty, które trzymamy w ręce. Przypominam, że każdą kartę, którą bierzemy ze stołu wstawiamy w odpowiednie miejsce wśród kart, które mamy w ręce tak, aby były one posortowane.

Tak więc bierzemy pierwszą liczbę (w tym wypadku jest to 3) i przenosimy ją na lewą stronę. Nie musimy nic poza tym robić, ponieważ po lewej stronie nie ma żadnych liczb. W ten sposób otrzymujemy:

3 09

Teraz bierzemy kolejną liczbę z lewej strony, czyli 0 i wstawiamy w odpowiednią pozycję po lewej stronie. Ponieważ 0 jest mniejsze od 3, więc wstawiamy je przed tą liczbę:

03 9

Podobnie postępujemy z liczbą 9. Ponieważ jest ona największa z tych po lewej, więc wstawiamy ją na końcu:

039

```
using namespace std;
const int N = 20; // Liczebność zbioru.
// Program główny
main()
{
    int d[N],i,j,x;
    // Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi
    // a następnie wyświetlamy jej zawartość
    srand((unsigned)time(NULL));
    for(i = 0; i < N; i++) d[i] = rand() % 100;
    for(i = 0; i < N; i++) cout << setw(4) << d[i];
    cout << endl;
    // Sortujemy
    for(j = N - 2; j >= 0; j--)
    {
        x = d[j];
        i = j + 1;
        while((i < N) && (x > d[i]))
        {
            d[i - 1] = d[i];
            i++;
        }
        d[i - 1] = x;
    }
}
```

```
#include <cmath>
#include <iostream>
#include <iomanip>

// Wyświetlamy wynik
// sortowania
cout << "Po sortowaniu:\n\n";
for(i = 0; i < N; i++) cout <<
setw(4) << d[i];
cout << endl;
}
```

Po co się męczyć ?!

Funkcja sort zdefiniowana w bibliotece <algorithm> porządkuje elementy (domyślnie niemalejąco)

sort([nazwa_tablicy], [nazwa_tablicy] + [liczba_elementów_do_posortowania]);

#include<algorithm>

#include <iomanip>

.....

const int N = 20; // Liczebność zbioru.

main()

{

int d[N],i;

for(i = 0; i < N; i++) d[i] = rand() % 100;

cout<<"Przed sortowaniem:\n";

for(i = 0; i < N; i++) cout << setw(4) << d[i];

sort(d, d+N);

cout<<"\nElementy uporządkowane:\n";

for(int i=0;i<N;i++)cout<<d[i]<<' ';

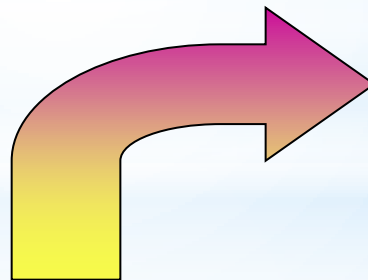
system("PAUSE");}

* Wykorzystanie tablicy dwuwymiarowej

Zadanie

Napiszmy program, który pozwoli na wprowadzenie ocen końcowych uczniów z kilku przedmiotów, a następnie obliczy i pokaże przedmiot o najwyższej średniej oraz ucznia który uzyskał najwyższą średnią.

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



Aby zaoszczędzić czasu
do wprowadzania danych
ograniczymy liczbę
uczniów i przedmiotów

```
const int w = 8; //liczba uczniów
const int k = 5; //liczba przedmiotów
```

Dla potrzeb programu opracujemy funkcję wprowadzania i wyprowadzania danych.

```
void wypelnij (int tab[w][k])
{
for (int i=0; i<w; i++)
{
cout<<"Podaj oceny ucznia: uczen"<<(i+1)<<endl;
cout<<"w kolejnosci: j.pol. j.ang. matem. biol. inf."<<endl;
for (int j=0; j<k; j++)
cin>>tab[i][j];
}
}
```

```
void wyswietl (int tab[w][k])
{
for (int i=0; i<w; i++)
{
cout<<"u"<<(i+1)<<" ";
for (int j=0; j<k; j++)
cout<<setw(7)<<tab[i][j];
cout<<endl;
}
}
```

Teraz zajmiemy się funkcją liczącą średnią z ocen ucznia i określimy, który uczeń ma ją największą.

```
void max_srednia_ucznia (int tab[w][k])
{
    float srednia_ucznia, max_srednia=0;
    int i,j,nr_ucznia;
    for (i=0; i<w; i++)
    {
        srednia_ucznia=0;
        for (j=0;j<k;j++)
        {
            srednia_ucznia=srednia_ucznia+tab[i][j];
        }
        srednia_ucznia=(float)srednia_ucznia/k;
        if (srednia_ucznia>max_srednia)
        {
```

```
            max_srednia = srednia_ucznia;
            nr_ucznia=(i+1);
        }
    }
    cout<<endl<<"Najwyzsza
srednia ucznia to:
"<<max_srednia;
    cout<<"Osiagnal ja uczen
u"<<nr_ucznia;
}
```

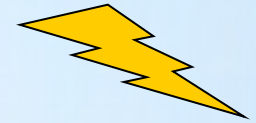
Podobnie musimy policzyć średnią z przedmiotów i określić maksymalną.

```
void max_srednia_przedmiot (int tab[w][k])
{
    float srednia_przedmiot, max_srednia=0;
    int i,j,nr_przedmiotu;
    for (j=0; j<k; j++)
    {
        srednia_przedmiot=0;
        for (i=0; i<w; i++)
            srednia_przedmiot=srednia_przedmiot+tab[i][j];
        srednia_przedmiot=(float)srednia_przedmiot/w;
        if (srednia_przedmiot>max_srednia)
        {
            max_srednia = srednia_przedmiot;
            nr_przedmiotu=j;
        }
    }
}
```

```
cout<<endl<<"Najwyzsza srednia z
przedmiotu: "<<max_srednia;
cout<<" Jest to ";
switch(nr_przedmiotu)
{
    case 0: cout<<"j.pol."; break;
    case 1: cout<<"j.ang."; break;
    case 2: cout<<"matem."; break;
    case 3: cout<<"biol."; break;
    case 4: cout<<"inf."; break;
}
}
```

Teraz wystarczy to poskładać ;-)

```
main ()  
{  
    int oceny[w][k];  
  
    wypelnij (oceny);  
    cout<<"    j.pol. j.ang. matem. biol. inf."<<endl;  
    wyswietl (oceny);  
    max_srednia_ucznia (oceny);  
    cout<<endl;  
    max_srednia_przedmiot(oceny);  
}
```



Mam nadzieję, że to było proste !!!

**Czas na utrwalenie, czyli
sprawdzian**

*Tablice tekstowe

W języku C++ teksty są przechowywane w tablicach z elementami typu *char*.

Deklaracja tablicy

char nazwa_tablicy[ilosc_elementow]

Można też inaczej

char przedmiot[]=„informatyka”

Ważne !!!

Ilość_elementów musi zawsze być o 1 większa, ponieważ jako ostatni znak jest zapisywany *\0* czyli *znacznik końca tekstu*.

Wyświetlanie zawartości tablicy

Za pomocą strumienia wejściowego *cin>>tekst* (gdzie tekstem jest zmienna będąca tablicą znaków) można pobrać tekst tylko do pierwszej spacji, reszta znaków jest ignorowana.

Można pobrać dowolny tekst zastępując *cin>>tekst* instrukcją
cin.getline(tekst,ilosc_znakow)

Np..

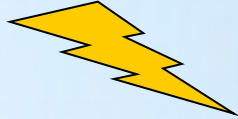
cin.getline(osoba,20)

```
main ()
{
char osoba[20];
cout<<"Przedstaw sie: ";
cin.getline (osoba,20);
cout<<"Juz wiem, jestes: "<<osoba << endl;
}
```

Napiszemy program, który w pobranym przez użytkownika równaniu matematycznym zamieni wszystkie znaki działań matematycznych na znaki zapytania i w tej postaci wypisze równanie na ekranie monitora np.:

2 * 12 + 200 - 60 = 264 zamieni na 2 ? 12 ? 200 ? 60 = 264

```
main (){
char dz[20];    //linia, do ktorej zostanie pobrane dzialanie
int i=0;
cout<<"Rownanie matematyczne do zamiany: ";
cin.getline(dz,20);
cout<<"Rownanie po ukryciu znakow dzialan arytmetycznych: ";
while (dz[i]!='\0')
{
if (dz[i]=='+'||dz[i]=='-' ||dz[i]=='*' ||dz[i]=='/')
    cout<<" ? ";    //znaki dzialan zostana zamienione na znak zapytania
else cout<<dz[i];    //inne pozostana niezmienione
    i++;
}
}
```



*Modyfikacje na tekście. Kody ASCII

Kod ASCII	Znak
31	☐
32	
33	!
34	"
35	#
36	\$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>

Kod ASCII	Znak
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^

Kod ASCII	Znak
95	_
96	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	
125	}
126	~

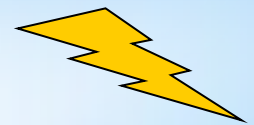
Z każdym znakiem klawiaturowym skojarzony jest jego kod **ASCII** np. A to 65

Oczywiście nie **musisz** znać wszystkich kodów **pod warunkiem**, że potrafisz programowo otrzymać informację o kodzie dla danego znaku.

* Już wiesz co masz robić !!!

Uczę się na pamięć 127 kodów || piszę programik

```
#include <iostream>
using namespace std;
main ()
{
    char znak[2];
    cout<<"podaj znak do zamiany:";
    cin>>znak;
    cout<<(int)znak[0];
}
```



*Szyfr Cezara

			a	b	c	d	e	f	g	h	i	j	k	l	.	.	.	.	.	.	w	x	y	z
a	b	c	d	e	f	g	h	i	j	k	l	.	.	.	.	.	.	.	.	x	y	z		

Liczbę pozycji o które się przesuwamy w celu znalezienia znaku po zaszyfrowaniu, nazywamy **kluczem**.

Funkcja szyfrująca

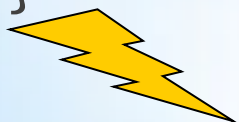
```
void szyfruj (char tekst[], int klucz)
{
    int i=0;
    klucz=klucz % 26;          //(1)
    while (tekst[i]!='\0')
    {
        if ((int)tekst[i]>122-klucz) //(2)
            tekst[i]=(char)((int)tekst[i]+klucz-26);
        else tekst[i] = (char)((int)tekst[i]+klucz);
        i++;
    }
}
```

Liczba znaków w
analizowanym
alfabecie

Zabezpieczenie przed
przekroczeniem zakresu
(122 ostatni kod ASCII)

Funkcja deszyfrująca

```
void deszyfruj(char tekst[], int klucz)
{
    int i=0;
    klucz=klucz % 26;
    while (tekst[i]!='\0')
    {
        if ((int)tekst[i]-klucz<97)
            tekst[i] = (char)((int)tekst[i]-klucz+26);
        else
            tekst[i] = (char)((int)tekst[i]-klucz);
        i++;
    }
```



Starym zwyczajem funkcję main()
robisz sam 😊

Wskaźniki-----

Wskaźnik (ang. Pointer) to specjalny rodzaj zmiennej, w której zapisany jest adres w pamięci komputera. Oznacza to, że wskaźnik wskazuje miejsce, gdzie zapisana jest jakaś informacja.

symbol	znaczenie	użycie
*	weź wartość x	*x
*	deklaracja wskaźnika do wartości	int *x;
&	weź adres	&x

main()

```
{  int i=13;
    int *wsk;           //deklaracja wskaźnika
    wsk=&i;              //od tej chwili wskaźnik będzie pokazywał na adres
                        //pod który znajduje się zmienna i
    cout<<"wskaznik wsk typu integer wskazuje na 'i', czyli(cout<<*wsk):"<<*wsk<<endl;
    cout<<"adres wskaznika w pamieci(cout<<wsk) :"<<wsk; }
```

Szyfrujemy lepiej od Cezara

Cezar bazował na jednej liczbie, kluczu wspólnym dla całego tekstu. My zastosujemy tablicę z której pobierać będziemy różne klucze.

Zaszyfrujemy tekst „Szkoła” przy pomocy zmiennych kluczy

Tablica klucz[]={12,13,5,7,2,9,10,17.....}

Litera	ASCII	Klucz	ASCII+klucz	Zaszyfrowany
S	83	12	95	-
z	122	13	135	‡
k	107	5	112	p
o	111	7	118	v
l	108	2	110	n
a	97	9	106	j

Szyfrujemy bezpieczniej

Nie będziemy się ograniczać do małych liter alfabetu czyli kodów ASCII od 97 do 122. Tabela ASCII zawiera 255 pozycji.

Unikniemy tzw. „zawijania tekstu”. Zmienne klucze szyfrujące zapiszemy w tablicy `klucz[]` z wartościami od 1 do 20.

wskaźnik

```
void szyfr (char *tekst, int przesuniecie[])
{
    int skok=0;
    for(int i=0;tekst[i]!='\0';i++)
    {
        tekst[i]=(tekst[i]+przesuniecie[skok]);
        skok++;
        if (skok==19) skok=0;
    }
}
```

```
void odszyfr (char *tekst, int przesuniecie[])
{
    int skok=0;
    for(int i=0;tekst[i]!='\0';i++)
    {
        tekst[i]=(tekst[i]-przesuniecie[skok]);
        skok++;
        if (skok==19) skok=0;
    }
}
```

Składamy

Składamy

main()

{

```
int klucz[]={12,13,5,6,17,19,4,8,17,19,4,11,5,7,6,9,16,14,13,8};
```

```
char tekstdoszyfrowania[100];
```

```
cout<<"tekst do zaszyfrowania: ";
```

```
cin.getline(tekstdoszyfrowania,100);
```

```
szyfr(tekstdoszyfrowania,klucz);
```

```
cout<<tekstdoszyfrowania<<endl;
```

```
cout<<endl<<"tekst odszyfrowany: ";
```

```
odszyfr(tekstdoszyfrowania,klucz);
```

```
cout<<endl<<tekstdoszyfrowania;
```

}

Automat wydaje resztę

Wydawanie reszty zaczynając od największej możliwej liczby nazywamy metodą **zachłanną**.

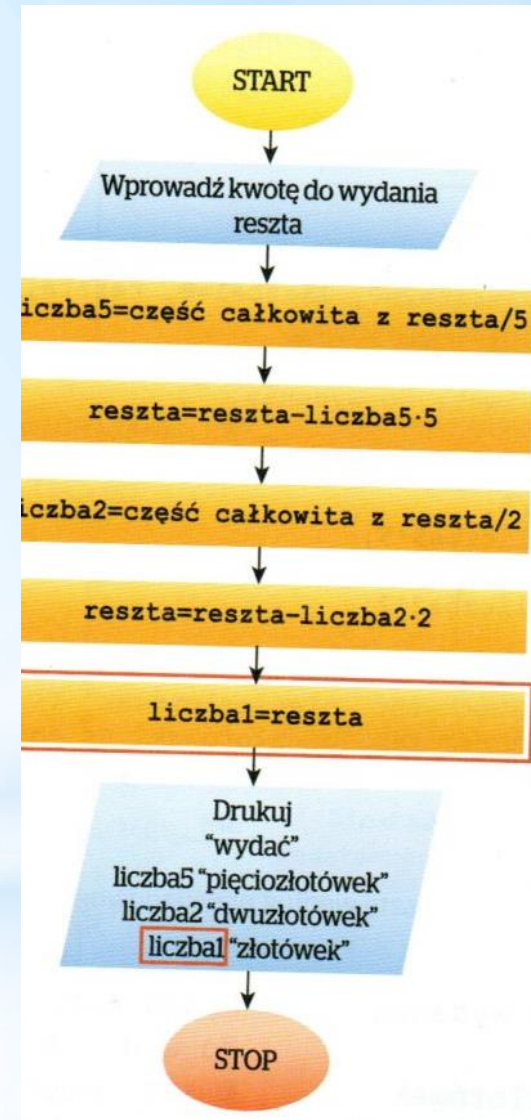
Nie sprawdzi się dla nietypowego zestawu nominałów np. mamy wydać 6 zł.

Metodą zachłanną to 1*5 zł i 1*1 zł.

Jeżeli w kasie brakuje monet 1 zł to automat ma problem, kasjerka natomiast wyda 3*2 zł.

Do opracowania algorytmu wydawania reszty metodą zachłanną musimy mieć do dyspozycji wszystkie dostępne nominały.

Dla uproszczenia ograniczymy się do pełnych złotych.



```

main()
{
    int l5=0,l2=0,l1=0,r=0;
    cout<<"Podaj kwote do wydania";
    cin>>r;
    l5=r/5;
    r=r-l5*5;
    l2=    r/2;
    r=r-l2*2;
    cout<<"Do wydania"<<endl;
    cout<<l5<<" szt. piecioletowek"<<endl;
    cout<<l2<<" szt. dwuzlotowek"<<endl;
    cout<<r<<" szt. zlotowek"<<endl;
}

```

```

Podaj kwote do wydania23
Do wydania
4 szt piecioletowek
1 szt dwuzlotowek
1 szt zlotowek
-----

```



```

Podaj kwote do wydania20
Do wydania
4 szt piecioletowek
0 szt dwuzlotowek
0 szt zlotowek
-----

```

Wydajemy co do grosza

W Polsce w obiegu mamy nominały 200 zł, 100 zł, 50 zł, 20 zł, 10 zł monety 5 zł, 2 zł, 1 zł, 0.50 zł, 0.20, 0.10, 0.05, 0.02 i 0.01 gr.

Definiujemy zmienne:

```
int l200=0, l100=0, l50=0, l20=0, l10=0, l5=0.....l001=0;
```

W tablicy prościej

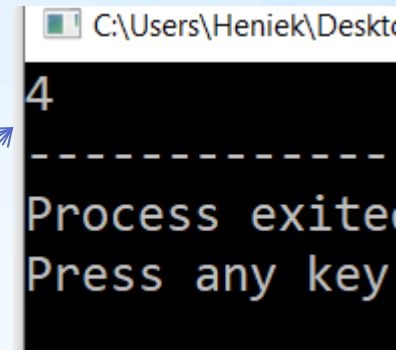
```
float nominaly[]={200,100,50,20,10,5,2,1,0.50,0.20,0.10,0.05,0.02,0.01}
```

oraz elastycznie, ponieważ w USA używa się monety 25 centów, a nie ma monet 2 i 20 centów, w „Bajkolandii” są monety 0.75 i 0.25 bajkonów ;)

Dzielimy ale musimy zaokrąglić, funkcja round()

.....

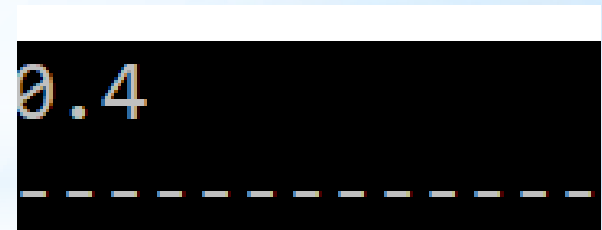
```
#include<math.h>
using namespace std;
main()
{
    float liczba=3.6754321;
    float wynik;
    wynik=round(liczba);
    cout<<wynik;
}
```



C:\Users\Heniek\Desktop
4

Process exited
Press any key

wynik=round(liczba)/10;



0.4

Co do grosza czyli 0,01

reszta=round((reszta-(nominaly[i]*liczbanominalow))*100)/100;

Rozwiązanie

```
#include<math.h>
using namespace std;
int ln, i=0; //ln-liczbanominalow, i-nominal z tablicy
float nominaly[]={200,100,50,20,10,5,2,1,0.50,0.20,0.10,0.05,0.02,0.01};
float reszta;
main()
{
    cout<<"Podaj liczbe do wyplacenia ";
    cin>>reszta;
    cout<<"Wydaj nominaly :"<<endl;
    while(reszta!=0)
    {
        if(reszta>=nominaly[i])
        {
            ln=reszta/nominaly[i];
            reszta=round((reszta-(nominaly[i]*ln))*100)/100;
            cout<<nominaly[i]<<" zl "<<ln<<" szt."<<endl;
            i++;
        }
    }
}
```

Ćwiczenia

- * Zmodyfikuj program tak by wydawanie reszty było funkcją
- * Przystosuj program do bankomatów w USA i „Bajklandii”
- * Zmodyfikuj program ze slajdu 104 i sprawdź dokładność wydawania 1 gr.

Wyszukiwanie wzorca w tekście metodą naiwną

Rozwiązaniem jest wskazanie w ciągu z wszystkich pozycji i takich, że zachodzi równość :

$$z[i:i+|q|]=q$$

Czyli wzorzec q jest fragmentem łańcucha z występującym na pozycji i -tej.

```
bool wzorzec(char* ciag_znakow, char* wzorzec)
{
    for(int i=0;ciag_znakow[i];i++)
    {
        int j=0;
        while(wzorzec[j]!='\0' && ciag_znakow[i+j]==wzorzec[j])
        {
            j++;
        }
        if(j>0&&wzorzec[j]=='\0')return true;
    }
    return false;
}
```

Biblioteka STRING, krótki opis

Biblioteka string odpowiedzialna jest za obiektowe podejście do ciągów znaków. Korzystanie z niej jest wygodniejsze niż ze standardowej obsługi tablicy znaków (**char ***).

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string a;
    cin>>a;           //wczytanie jednego wyrazu
    cout<<a<<endl;    //wypisanie go na ekranie
    cin.ignore();      //wyczyszczenie bufora
    getline(cin, a);   //wczytanie całego wiersza
    cout<<a;           //i wypisanie go na ekranie
    cin.get();
    return 0; }
```

Biblioteka STRING, krótki opis

Do odwoływania się do poszczególnych metod udostępnianych przez klasę string służy operator **kropki "."**. Na przykład, jeśli chcemy dowiedzieć się z ilu znaków składa się podany ciąg, użyjemy metody `size()` np. :

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string ciag;
    cout<<"Podaj wyraz: ";    //np. lokomotywa
    cin>>ciag;
    cout<<"Twój wyraz składa się z "<<ciag.size()<<" znaków"; cin.get();
    return 0;
}
```

Palindromy

.....

```
#include <string>
using namespace std;
string palindrom(string wyraz)
{
    int dlugosc = wyraz.length();
    for (int i = 0; i<=(dlugosc/2)-1; i++)
        if (wyraz[i]!=wyraz[dlugosc-1-i])
            return "nie";
    return "tak";
}

main()
{
    string wyraz;
    cout << "Podaj wyraz" << endl;
    cin >> wyraz;
    cout << palindrom(wyraz) << endl << endl;
}
```

Wyrażenie tak samo brzmiące czytane od lewej do prawej jak i prawej do lewej np. kajak, ala, abba.

Rekurencja

Definicja

Metodą rekurencyjną nazywamy taką metodę, w której stosujemy rekurencję, oznacza to odwołanie się funkcji (lub definicji) do samej siebie.

W matematyce funkcją rekurencyjną nazywamy funkcję, która jest zdefiniowana za pomocą samej siebie. W informatyce zaś **funkcją rekurencyjną** jest funkcja, która wywołuje samą siebie.

Silnia liczby naturalnej

Formalny zapis matematyczny funkcji silnia może wyglądać następująco:

$$n! = 1 \text{ dla } n \in \{0, 1\}$$

$$n! = 1 * \dots * n \text{ dla } N_+ - \{1\}$$

Rozwiązanie iteracyjne

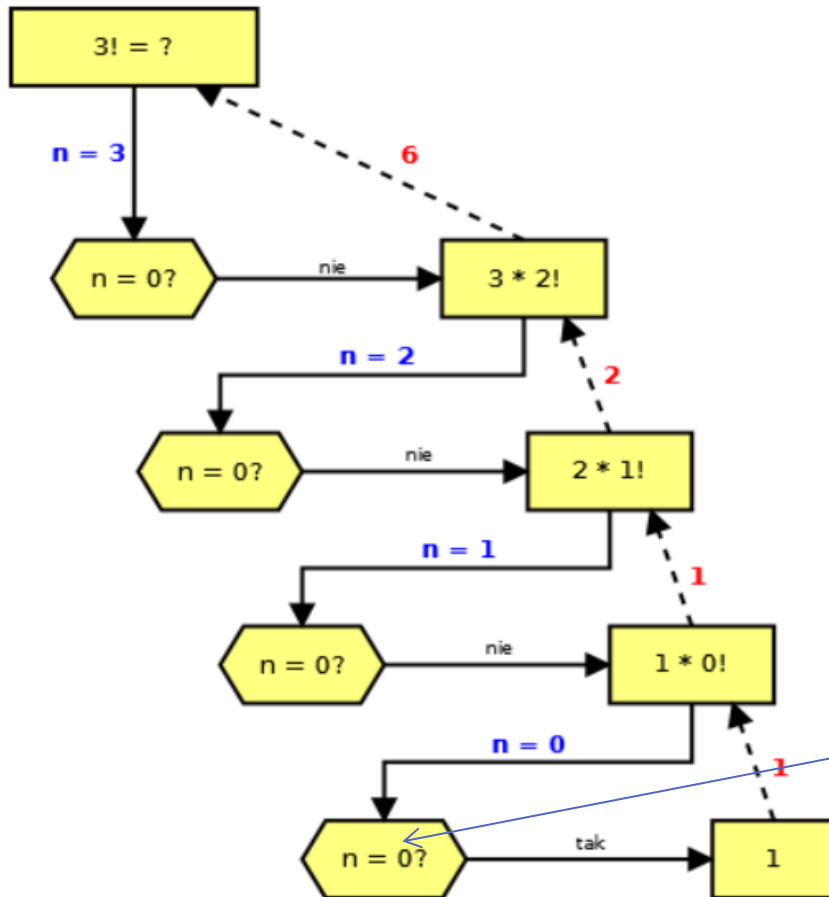
*Funkcja pobierze liczbę n
i zwróci otrzymany
iloczyn kolejnych licz
naturalnych, dla 0 nie
wykona pętli i zwróci 1*

```
double silnia_iteracyjnie (int n)
{
    double silnia=1;
    for (int i=2; i<n+1; i++) silnia=silnia*i;
    return silnia;
}
```

Oto rekurencyjna definicja silni zadanej liczby naturalnej n :

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n \cdot (n - 1)! & \text{dla } n \geq 1 \end{cases}$$

Schemat rekurencyjnych wywołań funkcji `silnia` dla argumentu 3 wygląda następująco:



Aby funkcja rekurencyjna była poprawna, musi mieć ściśle zdefiniowany warunek zakończenia wywołań rekurencyjnych.

Silnia rekurencyjnie

Jeżeli np. znamy $4!$ to $5! = 4! * 5$, a więc można to ująć w postaci ogólnej:

$$n! = (n-1)! * n$$

Aby funkcja rekurencyjna była poprawna, musi mieć ściśle zdefiniowany warunek zakończenia wywołań rekurencyjnych.

```
double silnia (int n)
{
if (n==0) return 1; //(1) warunek początkowy rekurencji
else return silnia(n-1)*n; //(2) rekurencyjne wywołanie funkcji
}
```

Potęga o wykładniku naturalnym liczby rzeczywistej

$$a^0=1$$

$$a^1=a$$

$$a^n=a*...*a$$

n czynników

$$A \geq 0$$

$$n \in \mathbb{N}^+ - \{1\}$$

Zapis rekurencyjny

$$a^0=1$$

$$a^n=a^{n-1}*a$$

```
float potega(float x,int n)
{
float wynik=1;
for(int
i=0;i<n;i++)wynik=wynik*x;
return wynik;
}
```

```
float potega_r(float x,int n)
{
    if(n==0)return 1;
    else return potega_r(x,n-1)*x;
}
```

Ciąg Fibonacciego

Ogólnie

$f(n)=f(n-1)+f(n-2)$ dla każdego argumentu większego od jeden.

Przykład:

1,1,2,3,5,8,13,21,34,55, itd

```
int fibonacci(int n)
```

```
{
```

```
    if (n<2) return 1;
```

```
    else return fibonacci(n-1)+fibonacci(n-2);
```

```
}
```

Teorię już poznałeś, więc czas na praktykę ;)

Napisz program który obliczy n-ty element ciągu

***To już było
!!!

Zamiana liczby na postać dwójkową -
rekurencyjnie

```
void dwojkowa(int liczba)
{
    if (liczba >= 2)
        dwojkowa(liczba/2);
    cout << liczba%2;
}
```

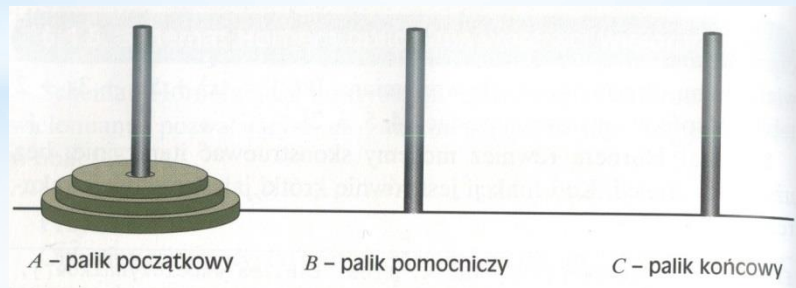
Zauważ, że nie operujemy na tablicach, więc nie ma problemu z wielkością przeliczanej liczby.

*Wieża Hanoi

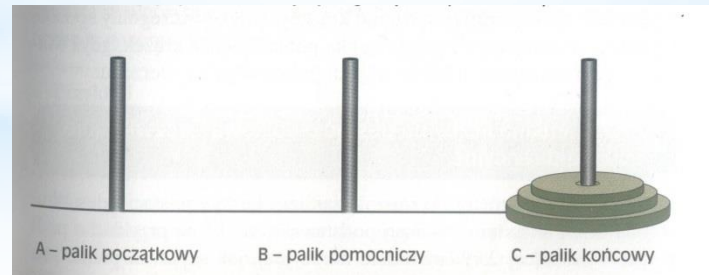
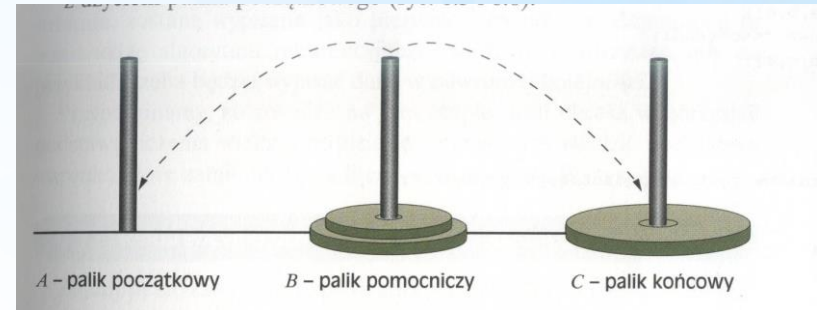
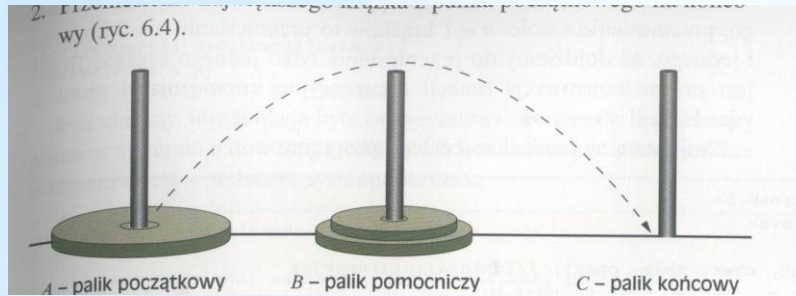
Wieża Hanoi oznacza zarówno klasyczny problem o rozwiązaniu rekurencyjnym, jak i grę dla starszych dzieci.

Przykład: Mamy trzy paliki. Na pierwszy z nich wsunięto n krążków od największego do najmniejszego. Zadanie polega na przeniesieniu wszystkich krążków z pierwszego palika na trzeci. Przestrzegamy przy tym zasad:

- Zawsze nakładamy mniejsze krążki na większe
- Z palika możemy zdjąć tylko jeden krążek



*Rozwiązanie



```
void hanoi (int, char, char, char); //funkcja  
    rekurencyjna
```

```
void hanoi(int n, char a, char c, char b)
```

```
{  
    if (n==1) cout<<a<<" -> "<<c<<endl;
```

```
    else
```

```
    {
```

```
        hanoi(n-1,a,b,c);
```

```
        cout<<a<<" -> "<<c<<endl;
```

```
        hanoi(n-1,b,c,a);
```

```
    }
```

```
}
```

```
main ()
```

```
{
```

```
    cout<<"ile krazkow jest na poczatkowym paliku? ";
```

```
    int ile;
```

```
    cin>>ile;
```

```
    hanoi(ile, 'A','C','B');
```

```
    System(„PAUSE”);
```

```
}
```

Rekurencyjnie

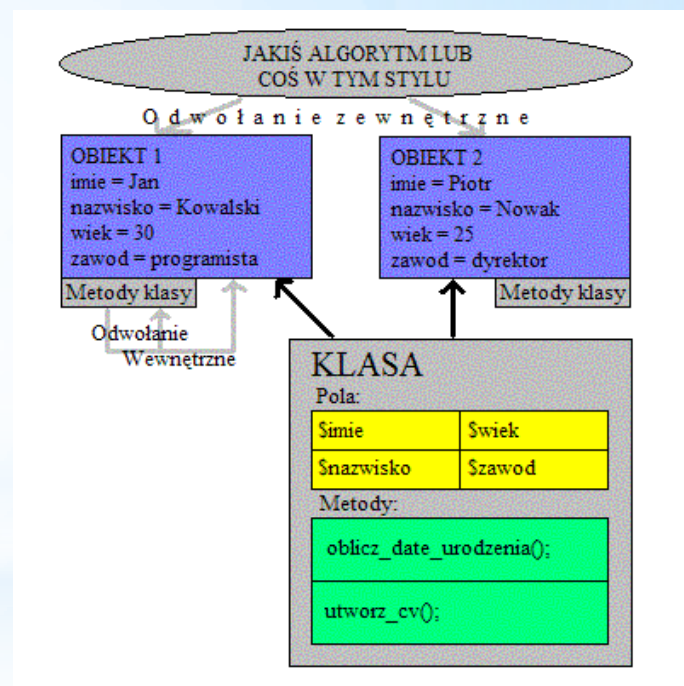
Poczytaj:

<http://kaj.uniwersytetradom.pl/cppf.html>

Struktury - typ definiowany przez użytkownika

Struktura (typ strukturalny) jest złożonym typem danych służącym do grupowania informacji opisujących jakiś obiekt.

Poszczególne dane **zgrupowane** w strukturze nazywamy **polami** lub **składowymi** struktury.



No to co to jest ?

W celu zapamiętania danych o osobie (nazwisko, imię, data urodzenia) musimy zdefiniować dane różnego typu i kontrolować te dane by dotyczyły tej samej opisywanej wielkości.

Można jednak zgrupować wszystkie informacje i umieścić je w jednej zmiennej.

Typ tej zmiennej to właśnie struktura.

No to może inaczej.....

Definiujemy strukturę *człowiek* a w niej

➤ *imię*[15] char

➤ *nazwisko*[20] char

➤ *wiek*[] integer

Mając tak zdefiniowany typ, możemy z niego korzystać, deklarując zmienne strukturalne. **Zmienne strukturalne są to zmienne, które są typu struktury przez nas zdefiniowanej.**

Dla typu *człowiek* mogą to być *zmienne mama, babcia, stryjek*

Wystarczy tylko nadać wartości poszczególnym polom np.:

```
mama.imie=„Anna”;
```

```
mama.nazwisko=„Kowalska”;
```

```
mama.wiek=40;
```

```
babcia.imie=„Maria”;
```

```
babcia.imie=Nowak”;
```

```
babcia.wiek=70;
```

Tego nie przepisuj, tylko zrozum ;)

```
struct czlowiek  
{  
    char imie[15];  
    char nazwisko[20];  
    int wiek;  
};
```

Definicja struktury, typu

//definicja typu musi kończyć się
średnikiem

Deklaracja zmiennych
strukturalnych

```
czlowiek mama,babcia;  
mama.imie=„Anna”;  
mama.nazwisko=„Kowalska”;  
mama.wiek=40;  
babcia.imie=„Maria”;  
babcia.imie=Nowak”;  
babcia.wiek=70;
```

Nadanie wartości
poszczególnym polom

I w ten sposób programista
stworzył człowieka ;)

*Praktyka

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct osoba
{
    char nazwisko[20];
    int wiek;
};
```

```
main()
{
    osoba uczen;
    cout<<"Podaj nazwisko ucznia";
    cin>>uczen.nazwisko;
    cout<<"Podaj wiek ucznia";
    cin>>uczen.wiek;
    cout<<"Informacje o uczniu"<<endl;
    cout<<uczen.nazwisko<<" wiek
"<<uczen.wiek<<" lat";
}
```

Dodaj osobie imię i płeć

Tak to powinno wyglądać

```
main()
{
    osoba uczen;
    cout<<"Podaj imię ucznia";
    cin>>uczen.imie;
    cout<<"Podaj nazwisko ucznia";
    cin>>uczen.nazwisko;
    cout<<"Podaj wiek ucznia";
    cin>>uczen.wiek;
    cout<<"Podaj plec ucznia";
    cin>>uczen.plec;
    cout<<"Informacje o uczniu"<<endl;
    cout<<uczen.imie<<" "<<uczen.nazwisko<<" wiek
    "<<uczen.wiek<<" lat "<<uczen.plec;
    getch();
}
```

```
struct osoba
{
    char nazwisko[20];
    char imie[15];
    int wiek;
    char plec[1];
};
```

Tablice o elementach typu strukturalnego

Zwykle grupujemy większą liczbę danych zdefiniowanych za pomocą tej samej struktury, przechowując je w tablicach.

```
#include<iostream>
#>
struct uczen
{
    char imie[20];
    char nazwisko[20];
    long numer_akt;
};

main()
{
    uczen grupa[10];
    cout<<"wypelniamy tablice danymi uczniow"
    <<endl;
    for(int i=0;i<10;i++)
    {
        cout<<"podaj imie ucznia: ";
        cin>>grupa[i].imie;
        cout<<"podaj nazwisko ucznia: ";
        cin>>grupa[i].nazwisko;
        cout<<"podaj nr akt ucznia: ";
        cin>>grupa[i].numer_akt;
    }
}
```

Struktury jako argumenty funkcji

Zadanie: Opracuj bazę danych na potrzeby szkoły dla obsługi wycieczki szkolnej na 20 osób. Należy zapisać

- *Imię:

- *Nazwisko:

- *Data urodzenia: rrrr-mm-dd

Przyjmij, że najstarsza osoba jest kierownikiem (opiekunem) wycieczki.

Zaczynamy oczywiście od zdefiniowania struktury, ale zamiast użyć w strukturze ucznia trzech pól związanych z rokiem urodzenia, zdefiniujemy strukturę definiującą datę.

Definiujemy struktury

```
struct data
{
    int rok;
    int miesiac;
    int dzien;
};
struct uczestnik
{
    char imie[20];
    char nazwisko[20];
    data data_urodzenia; //pole typu struktury data
                           zdefiniowanej wcześniej
};
```

Wprowadzamy dane

```
int wypelnij(uczestnik osoby[20])
{
    int t=0;
    char odpowiedz;
    do
    {
        cout<<"podaj imie uczestnika";
        cin>>osoby[t].imie;
        cout<<"podaj nazwisko
uczestnika";
        cin>>osoby[t].nazwisko;
        cout<<"podaj rok urodzenia";
        cin>>osoby[t].data_urodzenia.rok;
        cout<<"podaj miesiac urodzenia";

        cin>>osoby[t].data_urodzenia.miesiac
    ;
```

Argumentem funkcji
jest tablica zmiennych
strukturalnych

```
        cout<<"podaj dzien urodzenia";
        cin>>osoby[t].data_urodzenia.dzien;
        cout<<"Czy ktos jeszcze chce sie
zapisac T/N";
        cin>>odpowiedz;
        t++;
    }
    while(t<20 && odpowiedz!='N');
    return t;
}
```

Zwracana jest
informacja o liczbie
zajętych komórek

Wyświetlamy wprowadzone dane

```
void wypisz(uczestnik t[20], int liczba)
{
    for(int i=0;i<liczba;i++)
    {
        cout<<t[i].imie<<setw(20)<<t[i].nazwisko<<setw(5);
        cout<<t[i].data_urodzenia.rok<<"-
"<<t[i].data_urodzenia.miesiac;
        cout<<"-"<<t[i].data_urodzenia.dzien<<endl;
    }
}
```

Szukamy opiekuna wycieczki

W celu uproszczenia zakładamy, że jest znacząco starszy od uczestników i dlatego szukamy tylko roku.

```
uczestnik znajdz_opiekuna(uczestnik t[20],int liczba)
{
    uczestnik opiekun;
    opiekun=t[0];
    for(int i=0;i<liczba;i++)
        if(opiekun.data_urodzenia.rok>t[i].data_urodzenia.rok)
            opiekun=t[i];
    return opiekun;
}
```

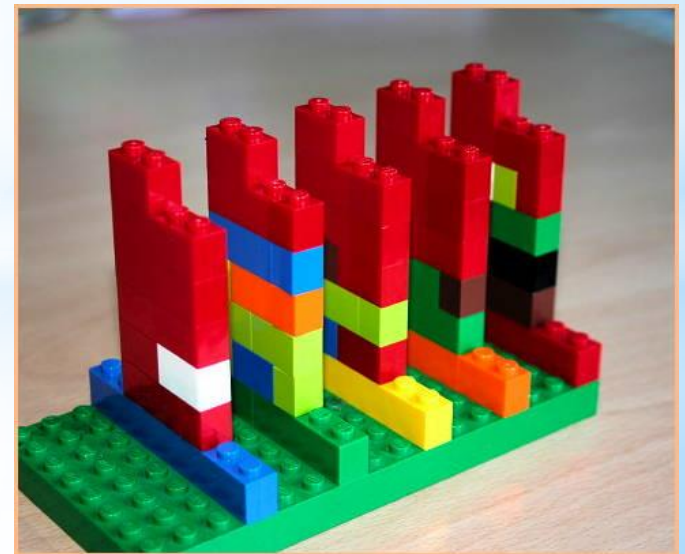
Wyświetlamy wskazany element tablicy struktury

```
void wypisz(uczestnik ktos)
{
    cout<<ktos.imie<<setw(10)<<ktos.nazwisko;
    cout<<ktos.data_urodzenia.rok<<"-"<<ktos.data_urodzenia.miesiac;
    cout<<"-"<<ktos.data_urodzenia.dzien<<endl;
}
```

no i wystarczy poskładać

Składamy

```
main()
{
    uczestnik osoby[20];
    int t=wypelnij(osoby);
    cout<<endl<<endl;
    wypisz(osoby,t);
    cout<<"***opiekun wycieczki***"<<endl;
    wypisz(znajdz_opiekuna(osoby,t));
    system(„PAUSE”);
}
```



Ćwiczenia

Znajdź i popraw błędy

```
struct kwiat
{
    int ilosc_platkow;
    int osiagana_wysokosc;
}
main()
{
    kwiat_lilia
    ilosc_platkow.lilia=4;
    osiagana_wysokosc.lilia=40;
}
```

Na podstawie poniższej funkcji zapisz najprostszą definicję struktury w niej wykorzystanej.

```
Void wyswietl_wartosc (osoba pracownik)
{
    cout<<"Staż pracy
    pana:"<<pracownik.nazwisko;
    cout<<" ,"<<pracownik.staż_pracy;
    cout<<"zarabia<<pracownik.zarobek;
}
```

Zadanie domowe

- *zdefiniuj strukturę opisującą filmy na płycie DVD. Film powinien być opisany przez tytuł, nazwisko reżysera, rok produkcji i czas trwania.
- *Zdefiniuj typ strukturalny opisujący kartę do gry oraz funkcję, która porównuje, czy dwie karty są tego samego koloru.



Typ wskaźnikowy

Wskaźnikami nazywamy zmienne, których wartością jest adres pewnego obszaru pamięci, a zadaniem - wskazywanie na inne zmienne.

Deklaracja zmiennej wskaźnikowej

typ *nazwa;

Może to pozwoli Ci zrozumieć

```
#include<iostream>
,
,
main()
{
    int i=13;
    int *wsk;    //deklaracja wskaźnika
    wsk=&i;      //od tej chwili wskaźnik będzie pokazywał na adres
                //pod który znajduje się zmienna i
    cout<<"wskaznik typu integer wskazuje na 'i', czyli:"<<*wsk<<endl;
    cout<<„adres wskaznika  :"<<wsk;
}
```

```
wskaznik typu integer wskazuje na 'i', czyli:13
adres wskaznika    :0x6ffe04
-----
Process exited after 0.1957 seconds with return value 0
Press any key to continue . . .
```

Zapamiętaj

**wsk* oznacza wartość, która znajduje się pod adresem będącym wartością wskaźnika *wsk*

wsk oznacza adres czyli wartość wskaźnika o nazwie *wsk*

&a oznacza adres komórki, w której znajduje się zmienna *a*

Możemy zmienić wartość wskaźnika

```
main()
{
    int i=13,j=16;
    int *wsk;
    wsk=&i;
    cout<<"wskaźnik typu integer wskazuje na 'i',  
czyli:"<<wsk<<endl;
    cout<<"wartosc wskaźnika  :"<<wsk;
    wsk=&j;
    cout<<"wskaźnik typu integer wskazuje na 'j',  
czyli:"<<wsk<<endl;
    cout<<"wartosc wskaźnika  :"<<wsk;
    system(„PAUSE”)
}
```

Zwróć uwagę na wartość adresów komórek, w których przechowywane są zmienne *i* i *j*.

Zadaniem wskaźnika jest nie tylko wskazywanie zmiennych i odczytywanie ich wartości, mamy także pełny dostęp do wskazywanych zmiennych np.:

```
*wsk=19;
```

Może zmienić zmienną *i* lub *j* w zależności na którą wskazuje

```
wsk=&i; lub wsk=&j;
```

* Inkrementacja i dekrementacja

Definiujemy wskaźnik

```
int *wsk;
```

Przypisujemy adres zmiennej całkowitej k

```
wsk=&k;
```

Instrukcja `wsk++;` zwiększy wartość wskaźnika o 4 bo tyle bajtów zajmuje integer w pamięci, natomiast `*wsk++;` modyfikuje wartość zmiennej na którą wskazuje.

Przekazywanie parametrów funkcji przez wskaźnik

```
void zmien(int *wsk)
{
    *wsk=*wsk+10;
}
main()
{
    int a=3;
    int *wsk;
    wsk=&a;
    cout<<"zmienna a ma wartosc: "<<a<<endl;
    zmien(wsk);
    cout<<" teraz zmienna a ma wartosc: "<<a<<endl;
    getch();
}
```

Wskaźniki w tablicach

```
main()
```

```
{
```

```
    int tab[5];
```

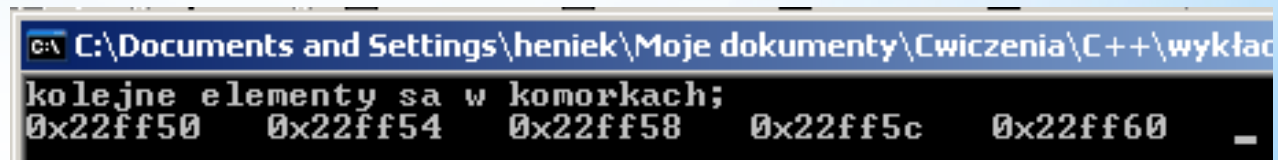
```
    cout<<"kolejne elementy sa w komorkach;"<<endl;
```

```
    for(int i=0;i<5;i++)
```

```
        cout<<&tab[i]<<"  ";
```

```
}
```

Sprawdzimy jak
ułożone są w
pamięci kolejne
elementy tablicy



```
C:\Documents and Settings\heniek\Moje dokumenty\Cwiczenia\C++\wykład
kolejne elementy sa w komorkach;
0x22ff50  0x22ff54  0x22ff58  0x22ff5c  0x22ff60
```

Praktyka

```
main()
```

```
{  
    char tab[40];  
    char *wsk;  
    int znaki=0;  
    wsk=tab;  
    cout<<"Podaj tekst;"<<endl;  
    cin.getline(tab,40);  
    cout<<"wskaźnik ustawiony na początku tablicy ";  
    cout<<&wsk<<endl;  
    while(*wsk!='\0')  
    {  
        znaki++;  
        wsk++;  
    }  
    cout<<"wprowadziles "<<znaki<<" znakow"<<endl;  
}
```

Program pobiera z klawiatury ciąg znaków, a na ekranie monitora wyświetla liczbę wprowadzonych znaków.

Na podstawie tego kodu opracuj program który na podstawie ostatniej litery imienia określi płeć.

```
Podaj tekst;  
stoi na stacji lokomotywa  
wskaźnik ustawiony na początku tablicy 0x6ffe10  
wprowadziles 25 znakow  
-----
```

Przykład z zastosowaniem struktur

Napiszemy program, w którym utworzymy listę trzech najlepszych uczniów, a następnie ich wyświetlimy, wykorzystując wskaźniki.

```
#include<iostream>
#include<cstdlib>
struct uczen
{
    char imie[10];
    char nazwisko[15];
    float srednia_ocen;
};
```

*cd....

```
main()
{
```

```
    uczen tab[3];
    uczen *wsk; // wsk na typ uczen
    wsk=tab; // wskaźnik na początek tablicy
    for(int i=0;i<3;i++)
    {
        cout<<"podaj imie ucznia: ";
        cin>>wsk->imie;
        cout<<"podaj nazwisko ucznia: ";
        cin>>wsk->nazwisko;
        cout<<"podaj srednia ucznia: ";
        cin>>wsk->srednia_ocen;
        wsk++;
    }
```

Wypełniamy tablicę
odwołując się do
struktury za pomocą
wskaźnika

```
    wsk=tab;
    cout<<"najlepsi uczniowie"<<endl;
    for( int i=0;i<3;i++)
    {
        cout<<wsk->imie<<" ";
        cout<<wsk->nazwisko<<" ";
        cout<<wsk->srednia_ocen<<endl;
        wsk++;
    }
}
```

Wykonaj samodzielne

Wypełnij tablicę 10 elementową losowymi wartościami z przedziału $\langle 5, 11 \rangle$. Z wykorzystaniem zdefiniowanego wskaźnika, znając rozmiar tablicy, odczytaj jej elementy w odwrotnej kolejności aniżeli zostały zapisane.

Zmienne dynamiczne są to zmienne, które tworzymy w trakcie działania programu za pomocą operatora new. Usuwa się je operatorem delete. Czas ich występowania w programie jest uzależniony od potrzeb programu.

*Zmienne i struktury
dynamiczne

*Tablica dynamiczna

Tablica dynamiczna to taka tablica, która zostaje utworzona w trakcie działania programu i może być usunięta przed jego zakończeniem

Aby w programie zadeklarować tablicę, musimy najpierw zadeklarować wskaźnik zdolny wskazać element tablicy, a następnie za jego pomocą **alokować pamięć**.

Deklaracja dynamicznej tablicy elementów typu *float*

*float *tab=new float[k]*

```
#include<iostream.h>  
#include<conio.h>  
#include<iomanip.h>  
#include<new.h>
```

Biblioteka w której
zdefiniowana jest funkcja
zgłaszająca błąd alokacji
pamięci: bad_alloc

* Tablica o liczbie
elementów określonych
przez użytkownika

```
main()
{
    float *tab=NULL;
    cout<<"Ile liczb wpiszesz do tablicy ";
    int ile;
    float liczba;
    cin>>ile;
    try
    {
        tab = new float[ile];
    }
    catch(bad_alloc)
    {
        cout<<"brak miejsca na utworzenie
        tablicy";
        return -1;
    }
}
```

Próba utworzenia
nowej tablicy za
pomocą operatora
new

Jeśli nie udało się
utworzyć tablicy

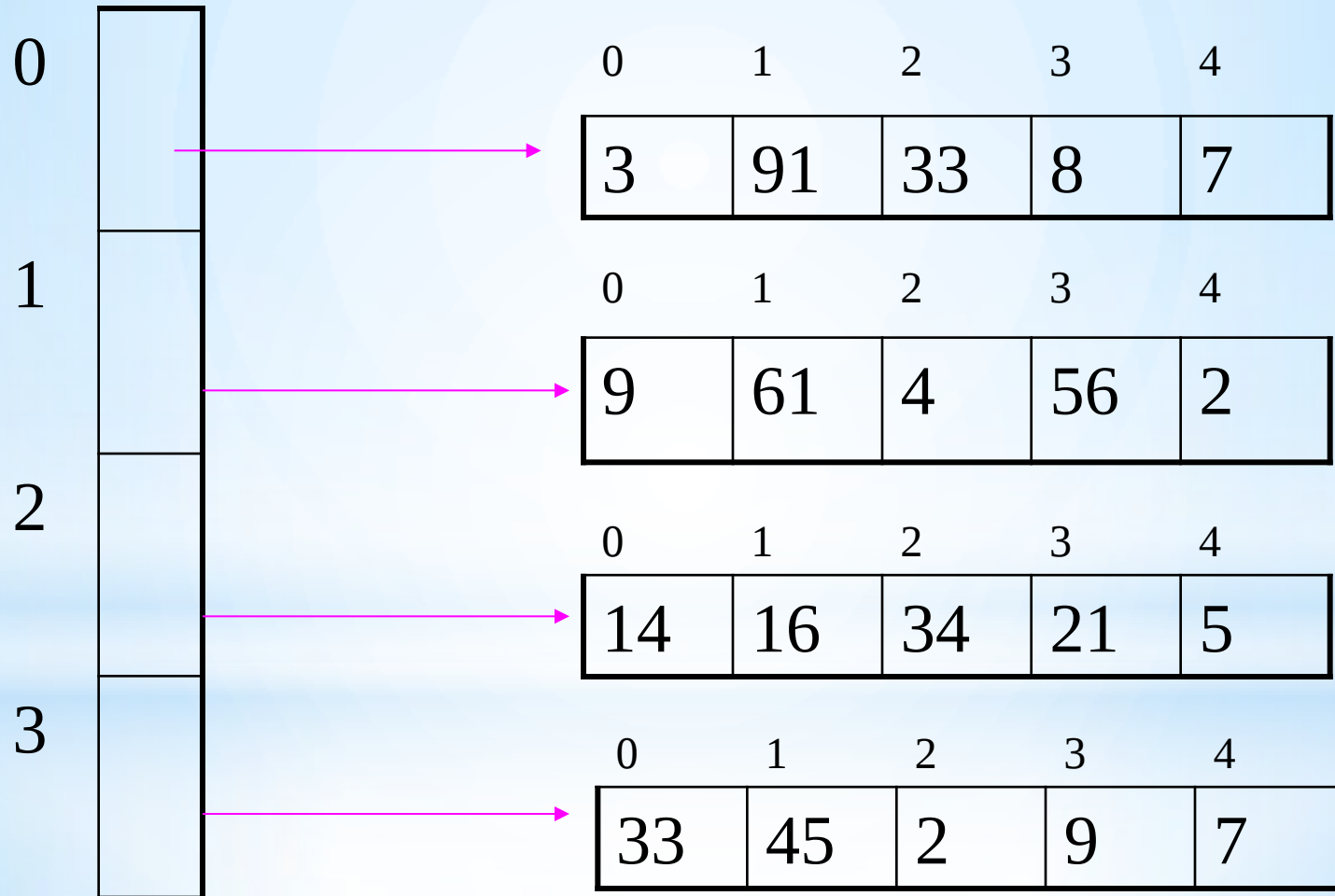
*Wypełniamy i wyświetlamy

```
for(int i=0;i<ile;i++)
{
    cout<<"podaj wartosc liczby:";
    cin>>liczba;
    *(tab+i)=liczba;
}
cout<<endl<<"Zawartosc tablicy:"<<endl;
for(int i=0;i<ile;i++)
{
    cout<<setw(6)<<*(tab+i);
}
delete[]tab;
getch();
}
```

Tablica dwuwymiarowa wymaga umiejętności zadeklarowania **wskaźnika do wskaźnika**. Musimy utworzyć tablicę tablic, co oznacza, że w naszej interpretacji **dwuwymiarowa tablica dynamiczna będzie jednowymiarową tablicą dynamiczną**, przechowującą **wskaźniki do tablic dynamicznych** jednowymiarowych w których umieszczone zostaną dane.

*** Tablica dynamiczna
dwuwymiarowa**

*Graficzna interpretacja



Deklaracja dwuwymiarowej tablicy dynamicznej [w] [k]

```
int **tab;
```

```
tab=new int*[w];
```

```
for(int i=0;i<w;i++)
```

```
    tab[i]=new int[k];
```

w- liczba wierszy
podanych przez
użytkownika

k - liczba kolumn
podanych przez
użytkownika

Zauważ, że najpierw tworzona jest jednowymiarowa tablica wskaźników, dopiero potem powstają tablice, na które te wskaźniki będą wskazywały.

*Przykład

Napiszemy program, który utworzy dwuwymiarową tablicę dynamiczną

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<iomanip.h>
#include<new.h>
main()
{
    int wie,kol,i,j;
    cout<<"podaj liczbe wierszy: ";
    cin>>wie;
    cout<<"podaj liczbe kolumn: ";
    cin>>kol;
```

```
srand(time(NULL));
```

```
int **tab;
```

```
tab=new int*[wie];
```

```
for(i=0;i<wie;i++)
```

```
try
```

```
{
```

```
    tab[i]=new int[kol];
```

```
}
```

```
catch (bad_alloc)
```

```
{
```

```
    cout<<"brak miejsca,  
    koncze";
```

```
    return -1;
```

```
}
```

Deklaracja
wskaźnika na
wskaźnik

Tworzona jest
tablica
wskaźników

Tworzone są
dynamiczne
tablice liczb

```

for(i=0;i<wie;i++)
{
    for(j=0;j<kol;j++)
    {
        tab[i][j]=rand()%101;
        cout<<setw(4)<<tab[i][j];
    }
    cout<<endl;
}
for(i=0;i<wie;i++)
delete []tab[i];
delete [] tab;
getch();
}

```

Usuwamy kolejne tablice z liczbami

Następnie usuwamy tablice wskaźników

tab=new int*[wie];

for(i=0;i<wie;i++)

tab[i]=new int[kol];

Zwróć uwagę, że kolejność jest odwrotna do tworzenia

*Praktycznie

Napiszemy program, który utworzy tablicę odległości pomiędzy miastami podanymi przez użytkownika.

	Kraków	Zakopane	Katowice	Poznań
Kraków	0			
Zakopane	100	0		
Katowice	75	156	0	
Poznań	300	411	270	0

W programie oprócz tablicy zamieszczającej odległości utworzymy również tablicę dynamiczną, do której wpisujemy nazwy miast w liczbie podanej przez użytkownika. Dla przejrzystości odczytu w odpowiednich miejscach wyświetlone zostaną nazwy miast.

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
#include<stdlib.h>
```

```
#include<iomanip.h>
```

```
#include<new.h>
```

```
main()  
{
```

```
    int liczba,i,j;
```

```
    cout<<"podaj liczbe miast: ";
```

```
    cin>>liczba;
```

```
    char **miasta;
```

```
    miasta=new char*[liczba];
```

```
    for(i=0;i<liczba;i++)
```

```
        miasta[i]=new char[20];
```

```
    for(i=0;i<liczba;i++)
```

```
    {
```

```
        cout<<"podaj nazwe miasta:
```

```
        ";
```

```
        cin>>miasta[i];
```



Wprowadzamy miasta

```

int **tab;
    tab=new int *[liczba];
    for(i=0;i<liczba;i++)
        try
        {
            tab[i]=new int[i+1];
        }
        catch(bad_alloc)
        {
            cout<<"brak miejsca
!";
            return -1;

```

```

        for (i=0;i<liczba;i++)
        {
            for(j=0;j<i+1;j++)
            {
                if(i==j) tab[i][j]=0;
                else {
                    cout<<"podaj odleglosc z miasta:
* <<miasta[i];
                    cout<<"do miasta "<<miasta[j]<<": ";
                    cin>>tab[i][j]; } } }
                cout<<" ";
            for (i=0;i<liczba;i++)
                cout<<setw(10)<<miasta[i];
            cout<<endl;

```

Wprowadzamy odległości

```

for(i=0;i<liczba;i++)
{
    for(j=0;j<i+1;j++)
    {
        if(j==0) cout<<setw(10)<<miasta[i];
        cout<<setw(10)<<tab[i][j];
    }
    cout<<endl;
}
for(i=0;i<liczba;i++)
    delete []tab[i];
delete []tab;
delete []miasta;
getch();

```



Wyświetlamy

Napisz program, który utworzy n - elementową tablicę dynamiczną gdzie n jest wartością podaną przez użytkownika, wypełnia ją liczbami rzeczywistymi, również podanymi przez użytkownika, a następnie skopiuje do nowo utworzonej tablicy dynamicznej tylko te liczby, które są większe od 0.

Zadanie

Zapis i odczyt z plików

Aby móc stosować pliki w programie musimy się nauczyć definiować strumienie płynące z pliku lub do pliku. Obsługa plików zawarta jest w dyrektywie preprocesora *#include<fstream>*

Możemy bez przeszkód zdefiniować obiekt, np. **zapis**, który pozwoli nam zapisać dane do pliku **wyjście.txt**

ofstream zapis(„wyjście.txt”);

```
#include<fstream>
main()
{
    ofstream zapis("wyjście.txt");
    zapis<<1<<" "<<9;
    zapis.close();
}
```

W celu stworzenia nowego pliku o nazwie **wyjście.txt** i zapisu do niego dwóch liczb całkowitych na przykład 3 i 9, oddzielonych spacją, wystarczy napisać.

A może sami nadamy nazwę pliku ?

```
#include<iostream>
#include<fstream>
main()
{
    char nazwa[20];
    cout<<"Podaj nazwę pliku";
    cin>>nazwa;
    ofstream zapis(nazwa);
    zapis<<1<<" "<<9;
    zapis.close();
}
```

„nazwa” może zawierać
ścieżkę bezwzględną
np.:c:\dane\wyniki\plik.txt
lub ścieżkę względną np.
..\dane\wyniki\plik.txt

Zabezpieczamy program

```
#include<iostream>
#include<fstream>
main()
{
    char nazwa[20];
    cout<<"Podaj nazwę pliku";
    cin>>nazwa;
    ofstream zapis(nazwa);
    if (!zapis)
    {
        cout<<"pliku nie można utworzyć";
        system(„Pause”);
        return 1;
    }
    else
    {
        zapis<<1<<" "<<9;
        zapis.close();
        system(„Pause”);
    }
}
```

Odczyt danych z pliku

Do definiowania strumieni wejściowych służy klasa *ifstream*, która jest zdefiniowana w bibliotece *fstream*

Napişemy program, który odczyta dwie liczby zapisane w pliku i obliczy pole prostokąta o długościach boków równych tym liczbom. Napişemy też obsługę błędów, gdyby nie powiodła się próba odczytu z pliku.

Kod programu

```
#include<iostream>
#include<fstream>
main()
{
    float a,b;
    ifstream wejscie(„liczby.txt");
    if(!wejscie)
    {
        cout<<"nie mozna otworzyc pliku";
        system(„PAUSE");
        return 1;
    }
```

Plik „liczby.txt”
przykładowe dane:

4.5

7

```
    wejscie>>a>>b;
    wejscie.close();
    cout<<"pole prostokata wynosi: "<<a*b;
    system(„PAUSE");}
```

Praktyka ;)

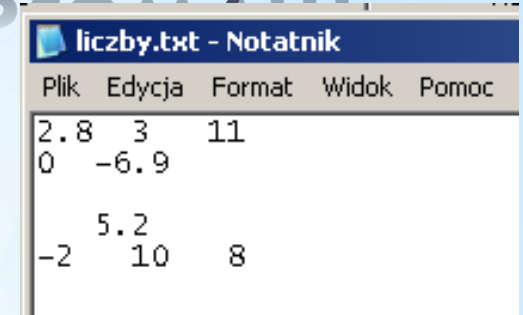
- * Napisz program obliczający miejsca zerowe funkcji kwadratowej,
- * Wartości a , b i c wpisz do pliku tekstowego, zabezpiecz zapisywane dane tak by były liczbami, współczynnik a musi być różny od zera,
- * Z pliku tekstowego pobierz dane i oblicz deltę oraz miejsca zerowe, a wyniki zapisz do pliku „wyniki.txt”.

Obliczymy sumę liczb typu *float* zapisanych w pliku tekstowym

```
main()
{   float a,suma=0;
    char nazwa[100];
    cout<<"Podaj nazwe pliku"<<endl;
    cin>>nazwa;
    ifstream we(nazwa);
    if (!we)
    {   cout<<"Nie można otworzyć pliku";

        return 1;    }
    while(!we.eof())
    {   we>>a;
        suma=suma+a;    }
    we.close();
    cout<<"suma liczb w pliku "<<nazwa<<" wynosi "<<suma;

}
```



Do wykrycia
końca pliku
wykorzystamy
funkcję *eof()*

Liczymy znaki w pliku tekstowym

```
#include<fstream.h>
```

```
main()
```

```
{   int n=0;
```

```
    char c;
```

```
    ifstream we("tekst.txt");
```

```
    if (!we)
```

```
    {       cout<<"Nie można otworzyć  
    pliku";
```

```
        return 1;    }
```

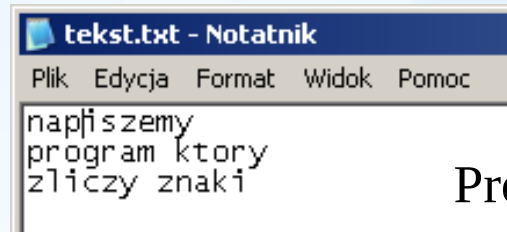
```
    while(!we.eof())
```

```
    {       we.get(c);  
           n=n+1;    }
```

```
    we.close();
```

```
    cout<<"ilosc znakow w pliku "<<n;
```

```
}
```



Program zliczy wszystkie znaki łącznie ze spacjami i znacznikami końca wiersza, oraz znacznikiem „eof”

if(isalnum(c))

n=n+1;

isalnum() zwraca wartość *prawda* jeśli jej argument podany w nawiasie jest literą, lub cyfrą

Funkcje często wykorzystywane w tekście

- * **isalnum()** - zwraca wartość *prawda*, jeżeli argumentem będzie litera, bądź cyfra
- * **isdigit()** - zwraca wartość *prawda*, jeżeli argument jest cyfrą.
- * **isalpha()** - zwraca wartość *prawda*, jeżeli argument jest literą.

No to kolega (koleżanka) nr 12

Napisze programik, który przepisze z naszego pliku *tekst.txt* do nowego pliku tylko litery (bez białych znaków).

Rozwiązanie

```
main()
{   int n=0;
    char c;
    ifstream we("tekst.txt");
    ofstream wy("literki.txt");
    while(!we.eof())
    {
        we.get(c);

        if(isalpha(c))
        {
            n=n+1;
            wy.put(c); }
    }
    we.close();
    wy.close();
    cout<<n;
    system(„PAUSE”);
}
```

W programie wykorzystamy funkcję **put()**, która zapisuje znak "c" do pliku powiązanego ze strumieniem wy.

Zabezpieczamy ostatni znak eof()

```
if(!we)
    break;
```

Liczymy słowa

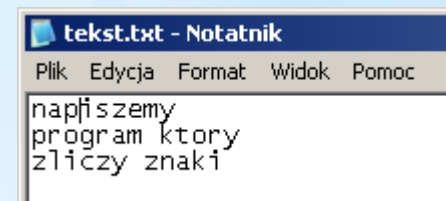
Napiszemy program, który policzy ilość słów w pliku.
Może Ci się to przydać w momencie kiedy dostaniesz listę osób a nie wiesz ile ich jest ;)

W programie tablica *plik[100]* służy do przechowywania nazwy pliku tekstowego, natomiast *slovo[100]* do przechowywania słowa odczytanego z pliku.

```
#include<iostream>
#include<fstream>
main()
{
    int n=0;
    char plik[100],slovo[100];
```



```
cout<<"Podaj nazwe pliku";
cin>>plik;
ifstream we(plik);
if(!we)
{   cout<<"nie mozna otworzyc pliku";
    return 1;
}
while(!we.eof())
{   we>>slowo;
    n=n+1;
}
we.close();
cout<<n;
system(„PAUSE”);
}
```



Wykorzystaj
plik tekst.txt z
poprzedniego
ćwiczenia

Czas na pytania

- * Jaką bibliotekę należy dołączyć do programu, aby mieć możliwość zapisu do pliku danych i odczytu danych z pliku.
- * Czym różni się plik od tablicy?
- * Jaką rolę zazwyczaj pełni plik w programie?
- * Jak skonstruować pętlę, która służy do odczytu całego pliku?

Praktycznie

- * Napisz program, który wpisze do pliku tekst pobrany z klawiatury, umieszczając każde zdanie w osobnej linii. Przez zdanie rozumiemy ciąg znaków zakończony kropką.



```
main()
{   char znak='.';
    ofstream wy("tekst1.txt");
    if(!wy)
    {       cout<<"Nie mozna utworzyc pliku";
            getchar();
            return 1;    }
    while(znak!='*')
    {
        cin>>znak;
        wy.put(znak);
        if(znak=='.')
            wy<<endl;
    }
    wy.close();
    getchar();
    return 0;
```

*Ad 1.....

Teraz policzymy wiersze w pliku.

Napiszemy program, który odczyta kolejne wiersze z pliku i obliczy ich ilość.

```
char wiersz[200]
```

```
while(!we.eof())
```

```
{
```

```
we.getline(wiersz, sizeof(wiersz));
```

```
    n=n+1;
```

```
}
```

```
we.close();
```

```
cout<<n;
```

```
getchar();
```

Nazwa tablicy
znaków

Zwraca rozmiar
tablicy

`getline(tablica, rozmiar, znak)`

Pominięcie trzeciego parametru powoduje przyjęcie wartości domniemanej `\n`

Inaczej:

`getline(wiersz, 10, 'k');` wczyta 10 znaków z wiersza, chyba, że wcześniej napotka literę **k**

Ćwiczenia- palindromy z pliku txt

```
include <iostream>
#include <string>
#include <fstream>
using namespace std;
int main()
{
    ifstream dane("dane_palindrom.txt");
    string palindrom;
    int i,j;
    bool test;
    while(!dane.eof())
    {
        getline(dane,palindrom);
        test = true;
        i = 0; j = palindrom.length() - 1;
        while(i < j)
            if(palindrom[i++] != palindrom[j--])
            {
                test = false;
                break;
            }
        if(test) cout << palindrom << endl;
    }
    dane.close();
    return 0;
}
```

W pliku dane_palindrom.txt umieszczono w kolejnych wierszach 1000 słów o długościach od 2 do 25 znaków, składających się z wielkich liter A, B, C, D, E, F, G, H, I, J. Napisz program, który przegląda słowa zapisane w pliku dane.txt i wypisuje te z nich, które są palindromami, po jednym w wierszu. Kolejność wypisywania palindromów powinna być taka sama jak w pliku z danymi.

http://www.nasza-szkola.pl/pomoce/informatyka/*klasy2c/

Programowanie obiektowe

Obiekt a Klasa

Definicja

Obiekt jest modułem, z którego programista buduje cały program. Obiekt w swej strukturze zawiera dane zwane **polami** i funkcje zwane **metodami**.

Aby w c++ móc w programie stworzyć obiekt, musisz zdefiniować klasę do której ten obiekt należy.

Definicja

Klasa jest to złożony typ będący opisem (definicją) pól i metod obiektów w programie.

* Składnikami klasy są pola i metody

Sposób definiowania klasy przeanalizujemy na przykładzie klasy ułamków zwykłych

class ulamek

{

public:

int licznik;

int mianownik;

};

Definicja klasy

PUBLIC oznacza składniki publiczne, czyli dostępne z wnętrza klasy, jak i spoza niej.

Mogą jeszcze wystąpić etykiety *private* oraz *protected* zwane **specyfikatorami dostępu**

Filmik

https://www.youtube.com/watch?v=aDXjubGK0jU&feature=emb_rel_pause

* Deklaracja

Deklaracja obiektu *ul1* klasy *ulamek* zdefiniowanej wcześniej może wyglądać następująco:

ulamek ul1;

Dostęp do poszczególnych składników obiektów uzyskujemy podobnie jak w składowych zmiennych strukturalnych

ul1.licznik;

ul1.mianownik;

*Hermetyzacja

Wymienione specyfikatory dostępu (*public*, *private*, *protected*) ściśle łączą się z cechami programowania obiektowego jakim jest **hermetyzacja**.

Definicja

Hermetyzacja pozwala na ukrycie pewnych danych i funkcji obiektu przed bezpośrednią ingerencją z zewnątrz.

Napişemy nową klasę ułamek w taki sposób, aby dostęp do danych *licznik* i *mianownik* był możliwy tylko przez funkcje, które „wiedzą” jak postępować z licznikiem i mianownikiem, czyli mianownik nie może być „0”.

Przykład

```
class ulamek
{   int licznik;
    int mianownik;
public:
    void zapisz (int l,int m)
    {
        licznik=l;
        if(m!=0)
            mianownik=m;
        else
        {
            cout<<"mianownik nie może mieć wartości 0";
            exit(1);
        }
    }
    void wypisz()
    {
        cout<<licznik<<"/"<<mianownik;
    }
};
```

```
main()
{
    ulamek ul1, ul2;
    ul1.zapisz(4,5);
    ul2.zapisz(1,7);
    cout<<"pierwszy ulamek: ";
    ul1.wypisz();
    cout<<"drugi ulamek: ";
    ul2.wypisz();
}
```

*Konstruktor

Definicja

Konstruktor nazywamy specjalną metodę automatycznie uruchamianą w trakcie definiowania (tworzenia) obiektu pozwalającą na zainicjalizowanie go określonymi danymi.

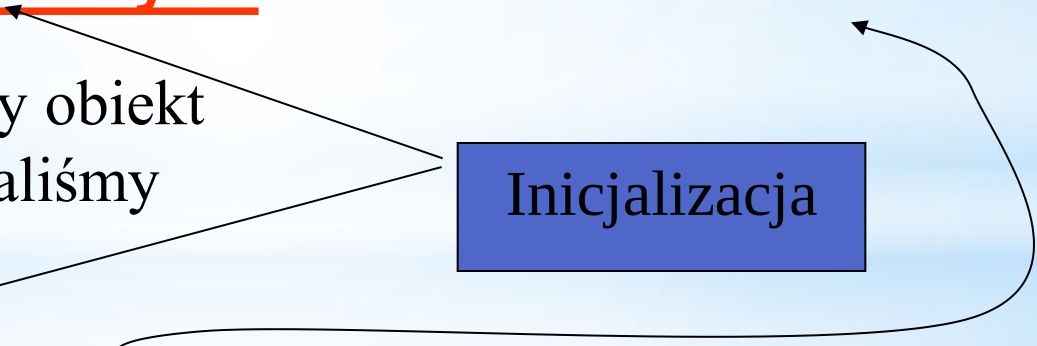
Tak deklarowaliśmy obiekt klasy ułamek i nadaliśmy wartość

ułamek ul1;

ul1.zapisz(4,5);

Inicjalizacja

Funkcja zapisywania danych
(nadania wartości)



* Czyli....

Deklarując zmienną typu `int` i nadając jej wartość możemy napisać

`int a;`

`a=5;`

Lub

`int a=5;`

```
class ulamek
{   int licznik;
    int mianownik;
public:
    ulamek(int l,int m);//deklaracja konstruktora
    //void zapisz (int l,int m);
    void wypisz()      {
        cout<<licznik<<"/"<<mianownik;    }
};
//void ulamek:: zapisz(int l,int m)
ulamek::ulamek(int l,int m)//definicja konstruktora
{   licznik=l;
    if(m!=0).....
```

*Destruktor

Definicja

Destruktozem nazywamy specjalną funkcję bezparametrową, która jest wywoływana zawsze w momencie usuwania obiektu.

W naszym programie wygląda to tak:

```
~ulamek()
```

```
{cout<<„usuwam obiekt”;}}
```

* Deklaracja przyjaźni

Definicja

Funkcja zaprzyjaźniona to taka funkcja zewnętrzna (niebędąca składnikiem klasy), dla której w definicji klasy umieszczona jest deklaracja przyjaźni za pomocą słowa kluczowego **friend**.

W przypadku klasy ułamek po wpisaniu:

```
friend ulamek pomnoz(ulamek,ulamek);
```

Funkcja `pomnoz` zwraca wyniki typu `ulamek` będzie zaprzyjaźniona z klasą `ulamek`.

```
class ułamek
{   int licznik;
    int mianownik;
public:
    void zapisz (int l,int m)
    {
        licznik=l;
        if(m!=0)
            mianownik=m;
        else
        {
            cout<<"mianownik nie może mieć wartości 0";
            exit(1);
        }
    }
    void wypisz()
    {
        cout<<licznik<<"/"<<mianownik;
    }
    friend ułamek pomnoz(ułamek,ułamek); //deklaracja przyjaźni
};
```

Napiszemy program,
który mnoży ułamki

Dodajemy funkcję bazującą na klasie

```
ulamek pomnoz(ulamek u1,ulamek u2)  
{  
    ulamek wynik;  
    wynik.licznik=u1.licznik*u2.licznik;  
    wynik.mianownik=u1.mianownik*u2.mianownik;  
    return wynik;  
}
```

```
main()
{
    ulamek ul1, ul2;
    int l,m;
    cout<<"podaj licznik i mianownik pierwszego ulamka";
    cin>>l>>m;
    ul1.zapisz(l,m);
    cout<<"podaj licznik i mianownik drugiego ulamka";
    cin>>l>>m;
    ul2.zapisz(l,m);
    cout<<"pierwszy ulamek: ";
    ul1.wypisz();
    cout<<"\ndrugi ulamek: ";
    ul2.wypisz();
    cout<<endl<<"-----"<<endl;
    cout<<endl<<"iloczyn ";
    ul1.wypisz();
    cout <<" i ";
    ul2.wypisz();
    cout<<"wynosi ";
    pomnoz(ul1, ul2).wypisz();
}
```

Pytania kontrolne

- * Czym się różni klasa od struktury?
- * W jaki sposób deklarujemy obiekty klasy?
- * Jakie są sposoby deklarowania funkcji w klasie?
- * Jak odwołujemy się do elementów klasy?
- * Jakie są typy składników klasy ze względu na dostęp do nich?
- * Co to jest konstruktor i do czego służy?
- * Co to jest funkcja zaprzyjaźniona i jakie ma właściwości?

Mirosław Zelent

<https://miroslawzelent.pl/kurs-objektowy-c++/podejscie-objektowe-objekt-klasa/>

<https://miroslawzelent.pl/kurs-objektowy-c++/struktura-projektu-objektowego/>

<https://miroslawzelent.pl/kurs-objektowy-c++/konstruktor-destruktor-public-private-protected/>

* Bibliografia

Materiały opracowane na podstawie podręcznika
„Informatyka”

Piotr Broda i Danuta Smołucha