

**7.04.2015**

**Programowanie** to proces składający się z etapów:

- a) tworzenia programu,
- b) testowanie programu.

**Język programowania** musi mieć określone:

- a) Reguły syntaktyczne → reguły **składni** wyrażeń i funkcji użytych w programie,
- b) Reguły semantyczne → semantyka języka programowania definiuje precyzyjnie **znaczenie poszczególnych symboli** (instrukcje, operatory itp.) oraz ich funkcję w programie.

**Definicja 1.**

**Algorytm** → jest to pewien ciąg czynności, który prowadzi do rozwiązania danego problemu w skończonej ilości kroków.

**Definicja 2.**

**Algorytm** → to jednoznaczny przepis, opisujący krok po kroku sposób postępowania w celu rozwiązania pewnego problemu lub sposobu osiągnięcia jakiegoś celu.

Ilość kroków algorytmu zależy od tego, jak złożony jest problem, którego on dotyczy. Zawsze jednak liczba tych kroków będzie **liczbą skończoną**.

**Cechy charakterystyczne poprawnego algorytmu:**

1. **Poprawność** - dla każdego przypisanego zestawu danych, po wykonaniu skończonej liczby czynności, algorytm prowadzi do poprawnych wyników.
2. **Jednoznaczność** - w każdym przypadku zastosowania algorytmu dla tych samych danych otrzymamy ten sam wynik.
3. **Szczegółowość** - wykonawca algorytmu musi rozumieć opisane czynności i potrafić je wykonywać.
4. **Uniwersalność** - algorytm ma służyć rozwiązywaniu pewnej grupy zadań, a nie tylko jednego zadania. Przykładowo algorytm na rozwiązywanie równań w postaci  $ax + b = 0$  ma je rozwiązać dla dowolnych współczynników  $a$  i  $b$ , a nie tylko dla jednego konkretnego zadania, np.  $2x + 6 = 0$
5. **Skończoność** – dla każdego zbioru poprawnych danych wejściowych algorytm powinien zwracać wyniki w skończonej liczbie kroków.
6. **Efektywność** – algorytm powinien rozwiązywać problem w jak najmniejszej liczbie kroków.

**Etapy konstruowania algorytmu(programu):**

1. Sformułowanie zadania.
2. Określenie danych wejściowych.
3. Określenie wyniku oraz sposobu jego prezentacji.
4. Ustalenie metody wykonania zadania.
5. Przy użyciu wybranej metody następuje zapisanie algorytmu.
6. Dokonujemy analizy poprawności rozwiązania.
7. Testowanie rozwiązania dla różnych danych.
8. Ocena skuteczności tegoż algorytmu.

**Algorytmy można przedstawiać m.in. następującymi sposobami:**

- słowny opis
- schemat blokowy
- lista kroków
- drzewo algorytmu
- drzewo wyrażeń
- w pseudokodzie
- w język programowania.

### Różne sposoby przedstawiania algorytmów

#### **a) opis słowny**

Jest na ogół pierwszym, mało ścisłym opisem sposobem rozwiązywania problemu. Rozpoczyna się często dyskusją, w jaki sposób można rozwiązać postawione zadanie. Dyskusja służy do rozważań nad sposobem i technikami przydatnymi w rozwiązaniu problemu.

*np. Opis słowny do algorytmu opisującego funkcję modułu (wartość bezwzględna).*

Dla wartości dodatnich argumentu  $x$  funkcja przyjmuje wartość  $x$ , dla wartości ujemnych argumentu  $x$  funkcja przyjmuje wartość  $-x$ .

#### **b) schemat blokowy**

#### **c) lista kroków**

Poszczególne kroki zawierają opis operacji, które mają być wykonane przez algorytm. Mogą w nich również wystąpić polecenia związane ze zmianą kolejności wykonywanych kroków.

Kolejność kroków jest wykonywana w kolejności ich opisu z wyjątkiem sytuacji gdy jedno z poleceń w kroku jest przejściem do kroku o podanym numerze. Budowa opisu algorytmu w postaci listy kroków jest następująca:

- ♦ tytuł algorytmu
- ♦ specyfikacja problemu
- ♦ lista kroków
- ♦ komentarze ujęte w nawiasy klamrowe {komentarz}

uwaga: Krok 0 może być opuszczony

<i>np. Lista kroków dla funkcji <math>SGN(x)</math> czytaj <math>sgn</math>.</i>	$SGN(x) = \begin{cases} -1 & \text{dla } x < 0 \\ 0 & \text{dla } x = 0 \\ 1 & \text{dla } x > 0 \end{cases}$
--	---

#### **Algorytm obliczania wartości funkcji $SGN(x)$**

*Dane:* Dowolna liczba rzeczywista  $x$ .

*Wynik:* Wartość funkcji

**Krok 0.** Wczytaj wartość danej  $x$

**Krok 1.** Jeśli  $x > 0$ , to  $f(x) = 1$ . Zakończ algorytm.

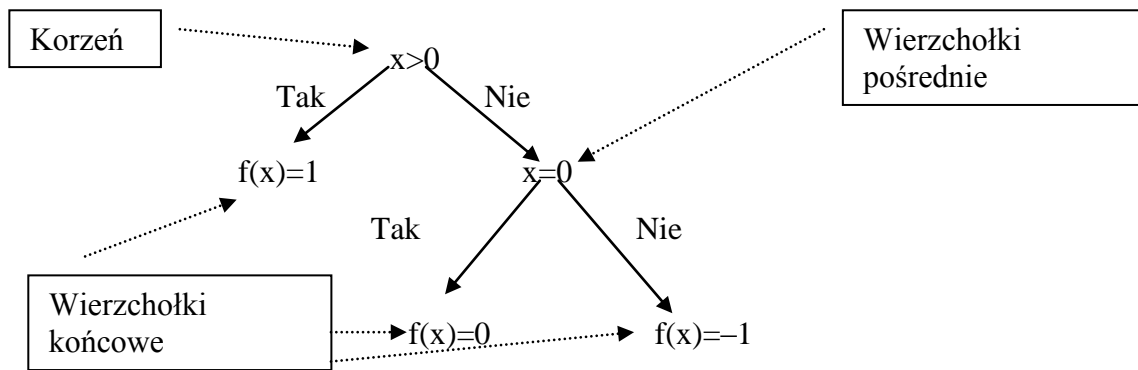
**Krok 2.** {W tym przypadku  $x \leq 0$ .} Jeśli  $x = 0$ , to  $f(x) = 0$ . Zakończ algorytm.

**Krok 3.** {W tym przypadku  $x < 0$ .} Mamy  $f(x) = -1$ . Zakończ algorytm.

#### **d) drzewo algorytmu**

Nazywany jest również drzewem obliczeń. Każde dwie drogi obliczeń mogą mieć tylko początkowe fragmenty wspólne, ale po rozejściu już się nie spotkają.

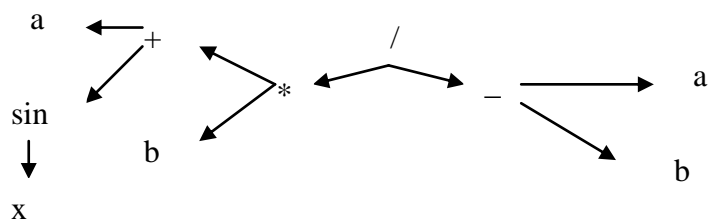
*np. Drzewo algorytmu dla funkcji  $SGN(x)$ .*



#### e) drzewo wyrażeń

Stosowane do obliczeń wyrażeń arytmetycznych.

np. Wyrażenie  $(a + \sin(x)) * b / (a - b)$



#### f) program w języku programowania np. C++, Pascal

#### g) pseudojęzyk

```


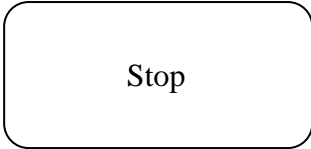

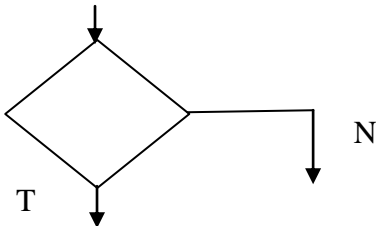
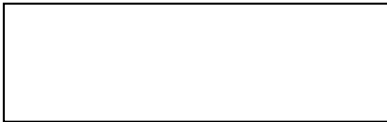
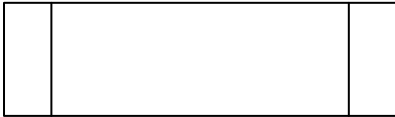
PROGRAM Wycieczka;
ZMIENNE punkty:naturalne;
        koszty, dofinansowanie:rzeczywiste;
ZACZNIJ;
        WPROWADŹ(PUNKTY,KOSZTY);
        JEŚLI punkty >=100 i punkty <= TO dofinansowanie :=1/3*koszty+0.2*koszty
                W PRZECIWNYM WYPADKU
dofinansowanie:=0.2*koszty;
        WYPROWADŹ('Dofinansowanie wynosi:'dofinansowanie);
ZAKOŃCZ.
  
```




### **Specyfikacja problemu**

Jest to dokładny opis problemu, który chcemy rozwiązać.

Specyfikacja składa się z:

- ◆ **danych wejściowych,**
- ◆ **dane wyjściowe** oraz warunki jakie muszą spełniać (czyli związek pomiędzy danymi a wynikami).
- ◆ **warunki** jakie muszą spełniać dane wejściowe
- ◆ **rysunki** (jeśli są konieczne), **wzory** obliczeniowe

Symbole stosowane w schematach blokowych.		
Początek algorytmu		W każdym algorytmie musi się znaleźć dokładnie jedna taka figura z napisem "Start" oznaczająca początek algorytmu. Blok symbolizujący początek algorytmu ma dokładnie jedną strzałkę wychodzącą.
Koniec algorytmu		W każdym algorytmie musi się znaleźć dokładnie jedna figura z napisem "Stop" oznaczająca koniec algorytmu. Najczęściej popełnianym błędem w schematach blokowych jest umieszczanie kilku stanów końcowych, zależnych od sposobu zakończenia programu. Blok symbolizujący koniec ma co najmniej jedną strzałkę wchodzącą.
blok wyjścia  wejścia		Równoległobok jest stosowany do odczytu lub zapisu danych. W jego obrębie należy umieścić stosowną instrukcję np. Read(x) lub Write(x) (można też stosować opis słowny np. "Drukuj x na ekran"). Figura ta ma dokładnie jedną strzałkę wchodzącą i jedną wychodzącą. Jest to blok wyjścia/wejścia wy/we I/O.
blok decyzyjny		Romb symbolizuje blok decyzyjny. Umieszcza się w nim jakiś warunek (np. "x>2"). Z dwóch wybranych wierzchołków rombu wyprowadzamy dwie możliwe drogi: gdy warunek jest spełniony (strzałkę wychodzącą z tego wierzchołka należy opatrzyć etykietą "Tak") oraz gdy warunek nie jest spełniony „Nie”. Każdy romb ma dokładnie jedną strzałkę wchodzącą oraz dokładnie dwie strzałki wychodzące.
blok przetwarzania		Jest to figura oznaczająca proces. W jej obrębie umieszczamy wszelkie <b>obliczenia lub podstawienia</b> . Proces ma dokładnie jedną strzałkę wchodzącą i dokładnie jedną strzałkę wychodzącą.
blok podprogramu		Ta figura symbolizuje proces, który został już kiedyś zdefiniowany. Można ją porównać do procedury, którą definiuje się raz w programie, by następnie móc ją wielokrotnie wywoływać. Warunkiem użycia jest więc wcześniejsze zdefiniowanie procesu. Podobnie jak w przypadku zwykłego procesu i tu mamy jedno wejście i jedno wyjście.

łącznik stronicowy		Koło symbolizuje tzw. łącznik stronicowy. Może się zdarzyć, że chcemy "przeskoczyć" z jednego miejsca na kartce na inne. Możemy w takim wypadku posłużyć się łącznikiem. Umieszczamy w jednym miejscu łącznik z określonym symbolem w środku (np. cyfrą, literą) i doprowadzamy do niego strzałkę. Następnie w innym miejscu kartki umieszczamy drugi łącznik z takim samym symbolem w środku i wyprowadzamy z niego strzałkę. Łączniki występują więc w parach, jeden ma tylko wejście a drugi wyjście.
łącznik międzystronicowy		Ten symbol to łącznik międzystronicowy. Działa analogicznie jak pierwszy, lecz nie w obrębie strony. Przydatne w złożonych algorytmach, które nie mieszczą się na jednej kartce.
element łączący		Poszczególne elementy schematu łączy się za pomocą strzałek. W większości przypadków blok ma jedną strzałkę wchodzącą i jedną wychodzącą

<a href="http://www.rafalbaran.net/mb">www.rafalbaran.net/mb</a>	Strona z magicznymi blokowymi.
--	--------------------------------

### Reguły rysowania schematów blokowych

- I. Po zbudowaniu schematu blokowego nie powinno być takich strzałek, które z nikąd nie wychodzą, lub do nikąd nie dochodzą.
- II. Każdy schemat blokowy musi mieć tylko jeden element startowy oraz co najmniej jeden element końca algorytmu.
- III. Element łączący(strzałki łączące) powinien być rysowany w poziomie i pionie, załamania pod kątem prostym.

### Podział algorytmów.

#### Definicja algorytmu liniowego

Algorytmem liniowym nazywamy taki algorytm, który ma postać listy kroków wykonywanych **zgodnie z ich kolejnością**.

Algorytmy liniowe są zapisem obliczeń, które mają postać ciągu operacji rachunkowych wykonywanych bez sprawdzania jakichkolwiek warunków.

#### Algorytm z warunkami (rozgałęzieniami)

Ten typ algorytmu musi mieć bloki decyzyjne czyli bloki sprawdzania warunków.

#### Algorytm numeryczny

Algorytmy, które wykonują działania matematyczne na danych liczbowych, nazywamy algorytmami numerycznymi.

#### Algorytm typu dziel i zwyciężaj

Dzielimy problem na kilka mniejszych, a te znowu dzielimy, aż ich rozwiązania staną się oczywiste,

#### Algorytmy iteracyjne

Iteracja jest to zapętlenie algorytmu, czyli wykonywania danych działań, dopóki warunek iteracji nie zostanie spełniony. Jest ona podstawą wszystkich choć troszkę bardziej złożonych algorytmów. Zazwyczaj ma ona składnię wykonuj "jakaś czynność" dopóki "jakiś wyrażenie logiczne".

#### Algorytmy rekurencyjne

Rekurencje wykorzystuje się do rozwiązywania problemów gdzie powtarza się czynność aby do niego dojść. Swoim działaniem przypomina iterację. Jednak w tym przypadku **funkcja**

**sama siebie wywołuje, dopóki nie otrzyma rozwiązania**, natomiast tam mieliśmy powtórzenie pewnej czynności określoną ilość razy.

**Złożoność algorytmu**- ilość zasobów potrzebnych do poprawnego działania danego algorytmu

**Złożoności obliczeniowa**-Algorytm wykonujący najmniejszą ilość operacji podstawowych w celu rozwiązania problemu.

**Złożoność czasowa**- Określa ilość operacji podstawowych potrzebnych do wykonania algorytmu o danej wielkości wejściowej.

**Złożoność pamięciowa**- Określa ilość przestrzeni pamięci wirtualnej potrzebnej do wykonania algorytmu z określonym zestawem danych wejściowych.

# Programowanie w języku C++ (dotycząca Dev-C++)

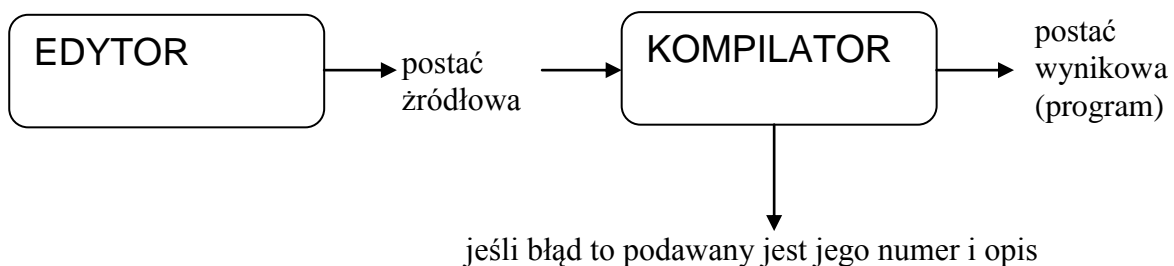
## Część 1

### Edytor, Kompilator. Tryb tekstowy

System DEV-C++ składa się z:

1. **EDYTORA** – służy do tworzenia tekstu programu w języku C++, traktowanej jako ciąg znaków (litera+liczby+znaki specjalne). Tekst tego programu nazywany jest programem w postaci źródłowej.

2. **KOMPILATORA** – program tłumaczący program w postaci źródłowej na program w postaci kodu asemblera (jest to ciąg operacji maszynowych zakodowanych w postaci liczb dwójkowych). Otrzymujemy plik w pośredni **OBJ**. W celu otrzymania postaci wynikowej konieczne jest użycie **linkera(konsolidatora)**. Linker: łączy pliki \*.obj, \*.lib, \*.dll, generuje plik wykonywalny \*.EXE. Postać wynikowa jest gotowa do wykonania jest zapisywany na nośniku w postaci EXE. Gdy program źródłowy ma błędy kompilator sygnalizuje błąd poprzez wskazanie jego miejsca oraz podanie jego numeru.



#### Tworzenie pierwszego projektu dla trybu DOS.

Plik→Nowy→Projekt→Console Application→Ok

Pojawi się okno gdzie będziemy przechowywać nasz projekt→wybierz miejsce przechowywania i Zapisz

W okienku edycji kodu źródłowego automatycznie jest generowany szkielet aplikacji:

```
#include <cstdlib>    //włączenie modułu użytecznych funkcji
#include <iostream>    //włączenie modułu wejścia-wyjścia

using namespace std; //zapewnienie użycia funkcji standardowych z biblioteki std

int main(int argc, char *argv[ ]) // początek funkcji głównej
{
    system("PAUSE");           // zatrzymanie programu w okienku DOS
    return EXIT_SUCCESS;      // zwrot wartości funkcji gdy jest błąd
}
```

Pliki generowane przez kompilator:

- plik projektu \*.dev
- plik kodu źródłowego \*.cpp
- plik wykonywalny \*.exe
- plik binarny \*.o
- plik reguł kompilatora Makefile. win

**Kompilowanie programu** czyli tłumaczenie z języka CPP na język maszynowy wraz z wykrywaniem błędów.

Uruchom→Kompiluj lub Ctrl+F9

**Uruchomienie programu** gdy kompilacja nie miała błędów to Uruchom→Uruchom lub Ctrl+F10



## **Polskie znaki na konsoli:**

- W treści programu wpisz instrukcję `system("chcp 1250");`
- uruchom Właściwości okna CMD (poprzez kliknięcie granatowego tytułu okna) i w zakładce Czcionki należy wybrać czcionkę Lucida Console i zatwierdzić dolną opcję.

### **Praca domowa pierwsza**

Zapisz numer pytania ( przed pytaniem np. Pytanie 1) poniżej zapisz treść pytania. Treść pytania podkreśl na zielono a pod treścią pytania zapisz odpowiedź.

#### **Pytanie 1**

Narysuj oraz opisz schemat współpracy edytora z kompilatora.

#### **Pytanie 2**

Co to jest postać źródłowa programu.

#### **Pytanie 3**

Co to jest kompilator? Jakie wykonuje czynności.

#### **Pytanie 4**

Co to jest kod asemblera?

#### **Pytanie 5**

Jakie pliki mają rozszerzenia OBJ.?

#### **Pytanie 6**

Jakie czynności wykonuje linker(konsolidator)?

#### **Pytanie 7**

Jakie rozszerzenie ma plik wykonywalny?

#### **Pytanie 8**

b)Zapisz w jaki sposób tworzymy nowy projekt dla trybu DOS.

#### **Pytanie 9**

c)Zapisz szkielet aplikacji automatycznie generowanej przez system.

#### **Pytanie 10**

Wypisz oraz opisz pliki generowane przez system CPP Dev.

#### **Pytanie 11**

Zapisz w jaki sposób kompilujemy programy w DEV

#### **Pytanie 12**

Zapisz w jaki sposób uruchamiamy programy w DEV

#### **Pytanie 13**

Co to jest komentarz, rodzaje komentarzy z przykładem, jakim kolorem zaznaczany jest komentarz w Dev.

#### **Pytanie 14**

Co to są pliki nagłówkowe. Po jakim słowie kluczowy są pisane pliki nagłówkowe i jak się kończy deklaracja. Znaczenie znaku #. Jakim kolorem oznaczane są dyrektywy preprocesora.

#### **Pytanie 15**

Co to jest funkcja główna. Przepisz przykład. Co oznaczają puste nawiasy okrągłe? Co oznaczają nawiasy klamrowe? Znaczenie `int`.

#### **Pytanie 16**

Opisz instrukcję `cout`

#### **Pytanie 17**

w jaki sposób realizujemy przejście do nowej linii z użyciem instrukcji `cout`

#### **Pytanie 18**

omów użycie znaków: `# { } ;`

#### **Pytanie 19**

Opisz polecenie `using namespace std;`

#### **Pytanie 20**

Opisz w jaki sposób używamy instrukcji dosowych. Zapisz cztery przykłady takich funkcji.

#### **Pytanie 21**

zapisz uwagi na temat stosowania dużych i małych liter.

#### **Pytanie 22**

zapisz uwagi na temat stylu programowania.

#### **Pytanie 23→pytanie praktyczne.**

Na domowym komputerze zainstaluj C++ DEV możliwie najnowszą wersję. Uruchom program. Wstaw szkielet aplikacji automatycznie generowanej przez system. Wykonaj rzut ekranu z wstawionym i widocznym szkieletem

aplikacji z Twoim Nazwiskiem. Zapisz rzut ekranu na pendrive oraz wyślij na pocztę w formacie GIF lub JPG. Zrzut ten będzie sprawdzany łącznie z pytaniami z pracy domowej.

## Komentarz

Komentarze są to napisy, których kompilator nie bierze pod uwagę podczas kompilacji programu.

### a)Komentarz wielolinijkowy.

/\* treść komentarza \*/

### b)Komentarz jednoliniowy

Komentarz zaczynający się od // treści komentarza.

## Plik nagłówkowy

Każdy program w C++ musi najpierw załączać odpowiednie pliki nagłówkowe z definicjami interesujących nas funkcji (można pisać własne pliki nagłówkowe). Po tych liniijkach ze słowem #include (możemy załączać kilka plików, ale wtedy każdemu poświęcamy oddzielną instrukcję #include). Zauważcie, że po instrukcji #include nie ma średnika. Jest ich kilkadziesiąt, a w każdym jest zdefiniowane ok. 50 funkcji.

Opis plików nagłówkowych.

### CONIO.H - CONsole Input/Output.

Plik nagłówkowy zawierający prototypy funkcji potrzebnych do obsługi standardowego Wejścia/Wyjścia na/z konsoli (CONSOLE). Plik zawiera między innymi prototyp funkcji clrscr(), potrzebnej nam do czyszczenia ekranu.

### STDIO.H - STanDard Input/Output

Plik nagłówkowy zawierający prototypy funkcji potrzebnych do obsługi standardowego Wejścia/Wyjścia na/z konsoli (Input - Wejście, Output - Wyjście). Plik zawiera między innymi prototyp funkcji printf(), potrzebnej nam do drukowania wyników na ekranie.

### IOSTREAM.H

Dołącza plik nagłówkowy udostępniający operacje we/wy w stylu C++

Instrukcja cout wyprowadza danych (wartości zmiennych, tekstów) z programu z użycie << na ekran (może być też do pliku). Jeśli jest to tekst to ujęty jest on w znaki " ".

cout<<endl;                      oznacza przejście do nowej linii i może być używane do wykonania pustej linii.

## Funkcja główna

Każdy program w C++ musi posiadać funkcję o nazwie **main**. Po słowie main występują nawiasy okrągłe. Puste nawiasy oznaczają, że funkcja nie przyjmuje żadnych argumentów. A sama funkcja nazywa się main (z angielskiego: główna), ponieważ to właśnie instrukcje umieszczone wewnątrz niej (w jej ciele) będą wykonywane przez program. Początek i koniec określania ciała funkcji oznaczamy odpowiednio: otworzonym i zamkniętym nawiasem klamrowym. int oznacza typ wyniku zwracanego przez funkcję główną w tym przypadku jest to liczba całkowita. Dev generuje automatycznie funkcję główną w postaci int main(int argc, char \*argv[]). W nawiasie okrągłym są argumenty wejściowe funkcji.

np.

```
int main( )  
{
```

treść programu

```
}
```

## Polecenie using namespace std;

Polecenie rozwiązuje problem dublowania się nazw różnych funkcji i poleceń. Std oznacza bibliotekę standardową, w której znajdują się definicje wszystkich najważniejszych symboli oraz poleceń i funkcji.

## Znaki specjalne

znak #

występuje przed dyrektywami preprocesora.

średnik ;

stawiany jest na zakończenie polecenia(instrukcji)

nawiasy klamrowe { }

umożliwiają tworzenie bloku funkcji, czyli łączyć wiele instrukcji prostych tak, aby były wykonywane razem.

## Użycie instrukcji dosowych

a)dołącz

```
#include <iostream>
```

```
#include <cstdlib>
```

b)wpisz instrukcję →pamiętaj, że instrukcja dosowa jest w cudzysłowie

```
system("dir");  lub
```

```
system("cls");      →czyszczenie ekranu
```

```
system("pause");    →zatrzymanie programu
```

która zatrzyma program i wyświetli komunikat mniej więcej w tym stylu: **Aby kontynuować. naciśnij dowolny klawisz...**

```
system("color 5");  →ustawienie koloru tła
```

## Ustawia domyślne kolory tła i pierwszego planu.

COLOR [atr]

atr     Określa atrybut koloru dla wyjścia konsoli

Atrybuty kolorów są określone przez DWIE cyfry heksadecymalne

-- pierwsza oznacza tło,

--druga pierwszy plan.

Każda cyfra może być jedną z wartości:

0 = Czarny

1 = Niebieski

2 = Zielony

3 = Błękitny

4 = Czerwony

5 = Purpurowy

6 = żółty

7 = Biały

8 = Szary

9 = Jasnoniebieski

A = Jasnozielony

B = Jasnobłękitny

C = Jasnoczerwony

D = Jasnopurpurowy

E = Jasnożółty

F = Jaskrawobiały

Jeśli nie podano argumentu, używany jest kolor odpowiadający chwili uruchomienia CMD.EXE. Wartość ta jest brana z bieżącego okna konsoli, z opcji /T wiersza polecenia lub z wartości rejestru DefaultColor.

Polecenie COLOR ustawia ERRORLEVEL na 1, jeśli podjęto próbę określenia tej samej wartości dla tła i dla pierwszego planu w poleceniu COLOR.

**Przykład:** "COLOR f0" daje kolor jasnoczerwony na jaskrawobiałym tle.

### Uwagi:

- Pamiętaj, że dla języka C++ PRINTF i printf to nie to samo! Słowa kluczowe i nazwy standardowych funkcji **muszą być pisane małymi literami !!!**

### Styl programowania

- Można funkcję main zapisać w takiej postaci, jak to zrobiliśmy w naszym programie (czyli w czterech liniach: nazwa main, nawias, jedna pusta linijka i nawias). Ale można też to wszystko zapisać w jednej linijce, np. `main(){ }`. Taki sposób nie jest zalecany ze względu na czytelność programu.
- Poza nielicznymi wyjątkami (nie można przedzielać tekstów ujętych w cudzysłów) w języku C++ możemy zrobić przerwę gdzie tylko chcemy.
- void - pusty, wolny

### Strumienie

Strumień to przepływ informacji od jednego urządzenia do innego. Strumieniem może być przepływ danych np. tekstu lub wartości zmiennych z pamięci komputera na ekran monitora. Trójkątne nawiasy (<< lub >>) wskazują

kierunek przepływu informacji.

### **Obiekty**

Obiekt podobnie jak program komputerowy jest to grupa danych i funkcji działających wspólnie i przeznaczonych razem do wykonania jakichś zadań. Dla przykładu obiekt **cout** służy do obsługi przesyłania danych na ekran monitora.

### **Przykład 1**

#### **Temat:**

Przykład demonstruj:

- dołączanie plików nagłówkowych,
- wyprowadzanie tekstów na ekran,
- użycie komentarz,
- wprowadzenie pustej linii ekranu,
- zatrzymanie programu.

#### **Wykonaj:**

- Wpisz temat do zeszytu
- Wpisać Przykład 1 (łącznie z komentarzem) do komputera. Nazwa pliku na dysku p1\_cztery\_pierwsze\_litery\_nazwiska
- Przepisać treści programu do zeszytu.

//Początek przykładu1

/\* Oto przykład program pokazujący wyprowadzanie tekstu na ekranu sposób1 z użyciem obiektu cout oraz zatrzymanie działania programu \*/

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

```
int main(int argc, char *argv[])
{
    cout<<"dzień dobry, tutaj Twój komputer";
    cout<<endl;
    cout<<endl;
    system("PAUSE");
    return 0;
}
```

### **Zadanie 1a**

Wykonaj program piszący 6 emotikonów wraz z wytłaczeniem, każdy w nowej linii.

### **Przykład 2**

#### **Temat:**

Przykład demonstruj:

- wyprowadzanie tekstów na ekran z użyciem instrukcji Printf,

#### **Wykonaj:**

- Wpisz temat do zeszytu
- Wpisać Przykład 2 do komputera. Nazwa pliku na dysku p2\_cztery\_pierwsze\_litery\_nazwiska
- Przepisać treści programu do zeszytu.

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

```
int main(int argc, char *argv[])
{
```

```

printf("Autor: .....");
cout<<endl;
printf("TO JA, TWOJ PROGRAM-PIERWSZY.CPP");
cout<<endl;
system("PAUSE");
return 0;
}

```

### **Zadanie 1**

#### **Temat: Użycie tabulatorów i przejścia do nowej linii.**

##### **Przeczytaj o znakach sterujących.**

Napisz program opisujący zakresy wartości przedstawionych w tabeli umieszczonej w instrukcji (patrz tabel na stronie około 19) w formie tabelarycznej (bez ramek). Tabela powinna mieć główkę z tytułami kolumn. Każda linia powinna mieć numer linii (oprócz główki).

Użyj instrukcji printf. W celu zachowania równych kolumn użyj znaków tabulatora \t. W celu przejścia do nowej linii użyj \n. Linie pionowe wykonaj ze znaków |.

Wpisujesz z tabeli numery wierszy nr\_z\_dziennika+4. Uczeń o numerze 7 w dzienniku wpisuje 7,11, 15, 19, 23, 27, 3.

### **Zadanie 2**

#### **Temat: Napisz program, który wywoła pięć dowolnych funkcji DOS.**

Zachowaj następujący schemat:

a)Przed wywołaniem funkcji na monitorze pojawi się napis( dwa opisy instrukcja cout i trzy printf )

Np.

Teraz kasuję ekran i dalsza część programu po naciśnięciu klawisza Enter.

b)wywoła się funkcja DOS

c)zatrzymanie ekranu okna konsoli DOS (Pause)

d)czyszczenie ekranu konsoli DOS (Cls)

.....powtórzenie dla pozostałych instrukcji

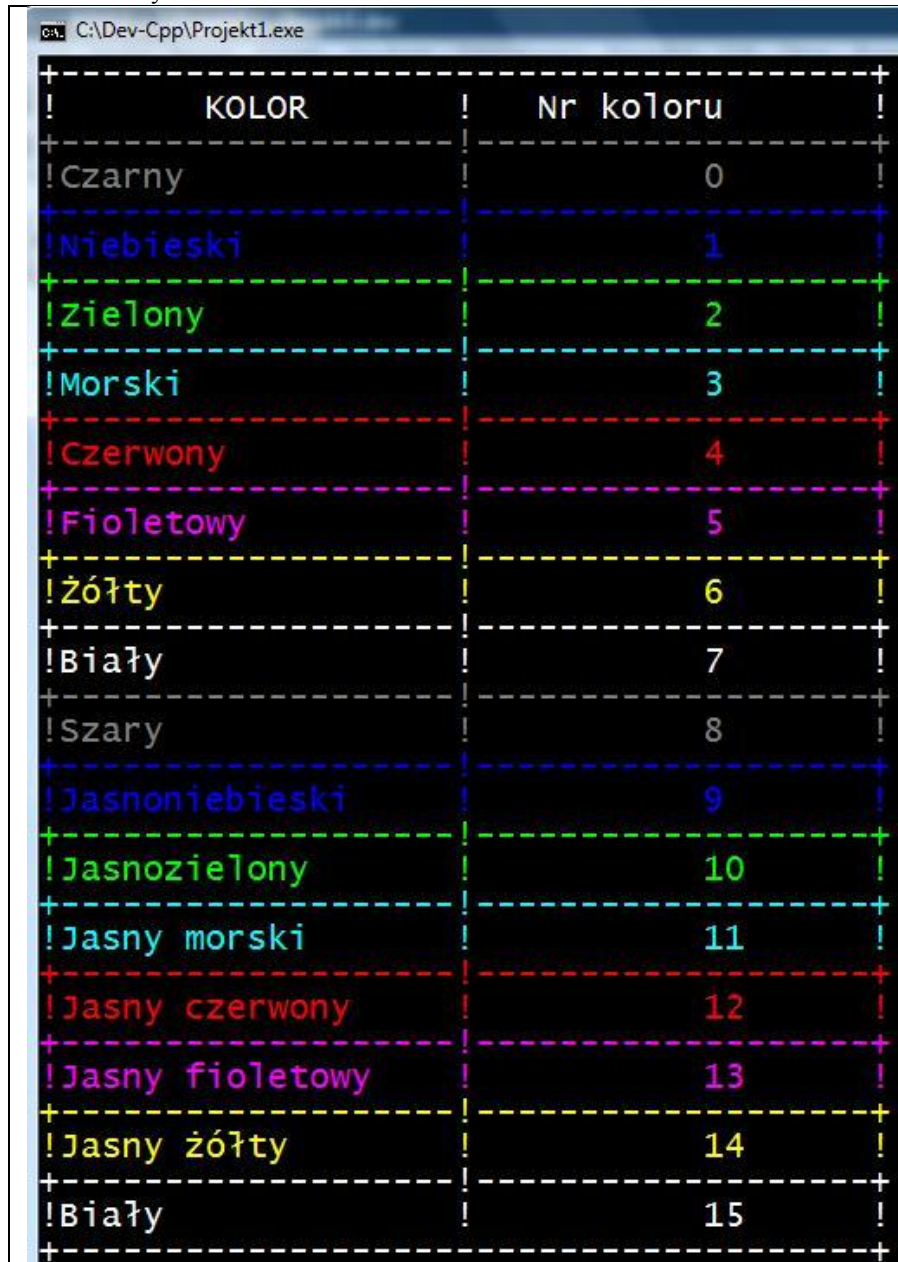
### Zadanie 3

Wykorzystując **przykład 3 i 4** napisz program, który narysuje następującą tabelę:

!	kolor	!	numer koloru	!
+	-----	!	-----	!
!	wszyst kolory	!	.....	!
+	-----	!	-----	+
!	biały	!	15	!
+	-----	!	-----	+

Każda linia (od drugiej) pisana takim kolorem, jaki opisuje. Kolory od 0 do 15.

Pause→białym



!	KOLOR	!	Nr koloru	!
+	-----	!	-----	+
!	Czarny	!	0	!
+	-----	!	-----	+
!	Niebieski	!	1	!
+	-----	!	-----	+
!	Zielony	!	2	!
+	-----	!	-----	+
!	Morski	!	3	!
+	-----	!	-----	+
!	Czerwony	!	4	!
+	-----	!	-----	+
!	Fioletowy	!	5	!
+	-----	!	-----	+
!	Żółty	!	6	!
+	-----	!	-----	+
!	Biały	!	7	!
+	-----	!	-----	+
!	Szary	!	8	!
+	-----	!	-----	+
!	Jasnoniebieski	!	9	!
+	-----	!	-----	+
!	Jasnozielony	!	10	!
+	-----	!	-----	+
!	Jasny morski	!	11	!
+	-----	!	-----	+
!	Jasny czerwony	!	12	!
+	-----	!	-----	+
!	Jasny fioletowy	!	13	!
+	-----	!	-----	+
!	Jasny żółty	!	14	!
+	-----	!	-----	+
!	Biały	!	15	!
+	-----	!	-----	+

### Zadanie 4

Wykonaj rysunek choinki noworocznej(gałęzie, stojak, pień, bombki , świeczka) . Z użyciem kolorów oraz funkcji gotoxy(int x, int y). Wpisz funkcję gotoxy nad main().

Uwaga:

Gdy chcesz wprowadzić znak \ na ekran konsoli w CPP musisz zapisać dwa razy \\ ponieważ znak \ jest znakiem specjalnym.

Czyli napisz  
Cout<<"\\";

### **Przykład 3**

Temat: Określanie kolorów tła i liter z użyciem <windows.h>

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

using namespace std;

void gotoxy(int x, int y)
{
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main()
{
    cout << "Tekst nie kolorowany\n\n";
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_GREEN |
    FOREGROUND_INTENSITY);
    cout << "Tekst pokolorowany\n\n";
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 15 | FOREGROUND_INTENSITY);
    // 15 to kolor biały
    cout << "Tekst nie kolorowany\n\n";

    system("pause");
}
```

### **Przykład 3a**

Temat: Określanie kolorów tła i liter z użyciem <windows.h>

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

using namespace std;
void gotoxy(int x, int y)
{
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main()
{
    gotoxy(10,10);
    cout<<"***";
    gotoxy(20,20);
    cout<<"***";
    system("pause");
}
```

### **Przykład 4**

Temat: Kolory na konsoli.

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

using namespace std;

int main()
{
    HANDLE hOut;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    cout << "Standart" << endl << endl;

    SetConsoleTextAttribute(hOut,BACKGROUND_RED);
    cout << "czerwony." << flush << endl << endl;

    SetConsoleTextAttribute(hOut,FOREGROUND_GREEN);
    cout << "zielony." << endl << endl;

    SetConsoleTextAttribute(hOut,FOREGROUND_BLUE);
    cout << "niebieski." << endl << endl;

    system("PAUSE");
    return 0;
}
```



## Część 2

### Zmienne, stałe, obliczenia, formatowanie wydruków.

#### Zadanie → domowe

Zapisz numer pytania ( przed pytaniem np. Pytanie 1) poniżej zapisz treść pytania. Treść pytania podkreśl na zielono a pod treścią pytania zapisz odpowiedź.

#### Pytanie 1

Omów, jakie nazwy mogą ( nie mogą ) mieć zmienne i stałe.

#### Pytanie 2

wymień oraz, krótko opisz sześć podstawowych typów danych wraz z rozmiarem w bajtach..

#### Pytanie 3

wymień uwagi na temat stosowania nazw,(wypisz 10 słów kluczowych, które nie mogą być nazwami)

#### Pytanie 4

Omów:

- deklarację,
- inicjację,
- definicje zmiennych

#### Pytanie 5

narysuj schemat ogólnego podziału danych

#### Pytanie 6

omów podstawowe modyfikacje do podstawowych danych,

#### Pytanie 7

omów deklarację stałych na przykładzie

#### Pytanie 8

- Omów sposoby zapisu liczb zmiennoprzecinkowych wraz z przydałem.
- Zamień liczbę z postaci wykładniczej na numer\_w\_dzienniku•miesiąc\_urodzeniae4 na liczbę bez potęgi.( tam jest kropka)

#### Pytanie 9

W jakim miejscu programu można dokonywać deklaracji zmiennych w C++..

#### Pytanie 10

Omów instrukcję cin. Podaj przykład jednej instrukcji cin do wczytania dwóch zmiennych.

#### Pytanie 11

Omów instrukcję przypisania w C++.

#### Pytanie 12

Zapisz co oznacza zapis w treści programu \n

#### Pytanie 13

Opisz Instrukcje **cin** wraz z przykładem oraz przykład wczytania dwóch zmiennych o nazwach **dwie\_pierwsze\_litery\_nazwiska** i **dwie\_pierwsze\_litery\_imienia**.

#### Pytanie 14

Opisz inkrementację oraz dekrementację wraz z przykładem.

#### Pytanie 15

Przerysuj tabelę operatorów arytmetycznych języka C++.

#### Pytanie 16

Przerysuj tabelę Operatorów relacji języka C++.

#### Pytanie 17

Zapisz Operatory logiczne języka C++ wraz z przykładem.

#### Pytanie 18

Omów funkcję **scanf( )**(Znaczenie, składnia bez opisu, przykład1 z opisem, przykład 2 z opisem)

#### Pytanie 19

Omów funkcję **printf( )** (Znaczenie, składnia bez opisu, przykład1 z opisem, przykład 2 z opisem)

#### Pytanie 20

Zanotuj trzy wzorce konwersji dla postać %s, %d, %f

#### Pytanie 21

Opisz trzy zakresy ważności zmiennych.

#### Pytanie 22

Opisz przykrywanie zmiennych.

### Nazwy zmiennych i stałych

Nazwy zmiennych i stałych mogą składać się z liter, cyfr i podkreślenia \_.

**Nazwa nie może się zaczynać od cyfry. np. 1promien**

Uwagi dotyczące stosowania nazw:

- nazwa powinna być znacząca np. **promien** a nie nie znacząca nazwa **mien**,
- nazwa może się składać z dużych i małych liter lecz musimy pamiętać, że kompilator rozróżnia duże i małe litery, czyli zmienne o nazwach R i r są innymi zmiennymi,
- przyjęto, że nazwy zmiennych piszemy małymi literami a stałych dużymi,
- nie można stosować nazw, które są zarezerwowane dla słów kluczowych:  
asm, auto, break, case, char, class, const, continue, default, delete, do, double, else, enum, extern, float, for, friend, goto, huge, if, inline, int, interrupt, long, near, new, operator - operator, pascal, private, protected, public, register, return, short, signed, unsigned, sizeof, static, struct, switch, this, typedef, union, virtual, void, volatile, while

### Deklaracja, inicjacja, definiowanie zmiennych.

#### DEKLARACJA

np. int a oznacza mniej więcej, jakbyśmy powiedzieli kompilatorowi: "Jakbyś gdzieś spotkał nazwę a to wiedz, że jest to zmienna typu int". Określa typ wartości, jakie może przechowywać dana stała lub zmienna.

#### INICJACJA

to pierwsze przypisanie stałej lub zmiennej. Powoduje przydzielenie pamięci.

#### DEFINICJI

to jednocześnie deklaracja i inicjacja.

Język C/C++ operuje sześcioma podstawowymi typami danych:

- **char** (znak, numer znaku w kodzie ASCII) - 1 bajt;
- **int** (liczba całkowita) - 2 bajty;
- **float** (liczba z pływającym przecinkiem) - 4 bajty;
- **double** (podwójna ilość cyfr znaczących) - 8 bajtów;
- **bool** (wartość logiczna, prawda lub fałsz)
- **void** (nieokreślona) 0 bajtów.

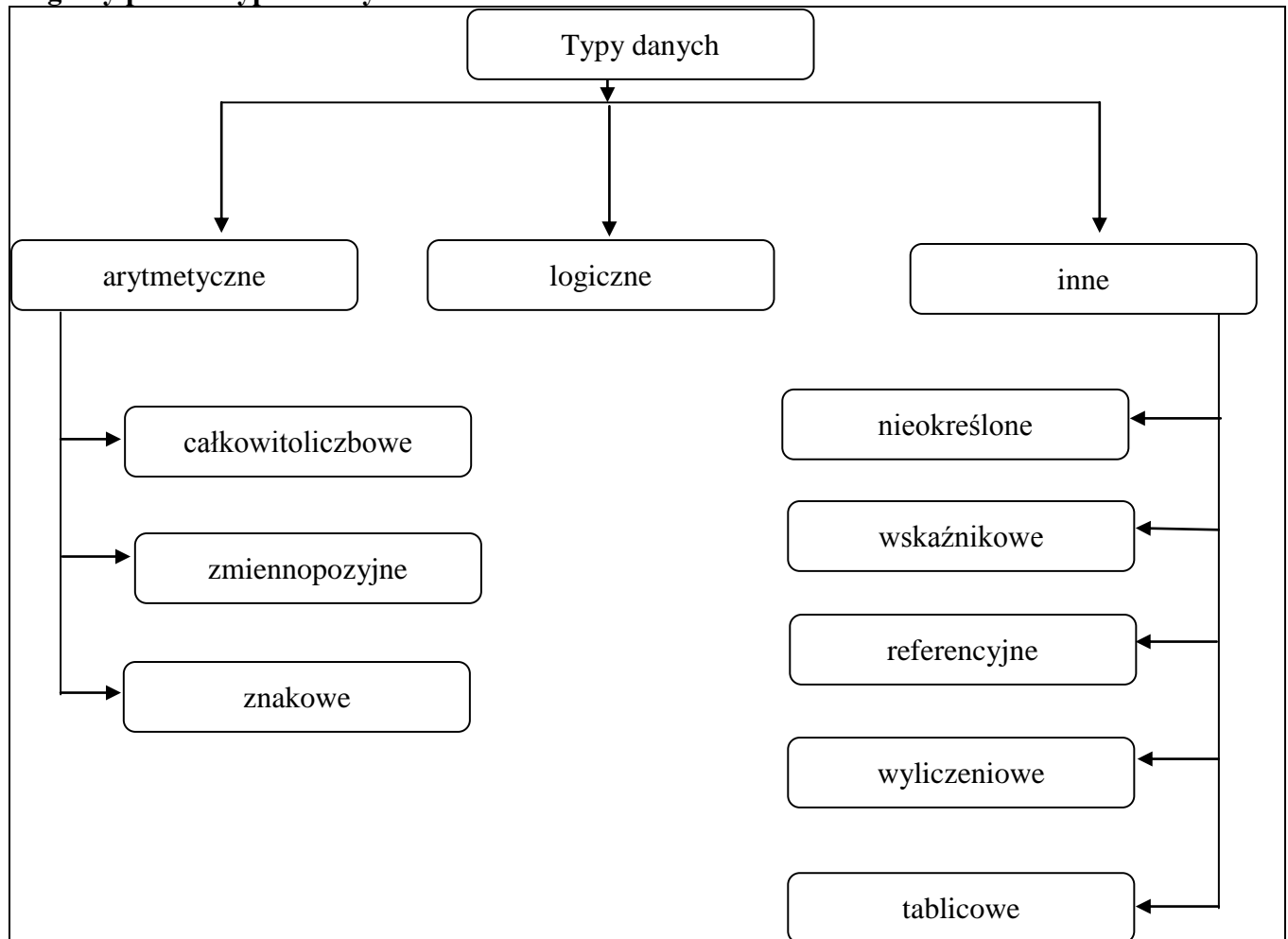
Podstawowe typy danych mogą być stosowane z jednym z czterech modyfikatorów:

- **signed** - ze znakiem
- **unsigned** - bez znaku
- **long** - długi
- **short** - krótki

Zakres wartości przedstawiono w Tabeli poniżej.

	Typ	Typowy rozmiar w bajtach	rozmiar
1	char signed	1 bajty	-128 do 127
2	unsigned char	1 bajty	0 do 255
3	unsigned short	2 bajty	0 do 65535
4	Int	2 bajty	-32768 do 32767
5	unsigned int	4 bajty	0 do 65535
6	long int	4 bajty	-2147483648 do 2147483647
7	signed short	2 bajty	od -32768 do 32767
8	signed int	4 bajty	Od -2147483648 do 2147483647
9	unsigned long int	4 bajty	0 do 4294967295
10	signed long int	4 bajty	-2147483648 do 2147483647
11	float	4 bajty	3.4 E- 38 do 3.4e+38
12	double	8 bajtów	1.7 E-308 do 1.7 E+308
13	long double	10 bajtów	1.2 E-4932 do 1.2E+4932
14	bool	1	true, false
15	unsigned short int	2 bajty	0 do 65535
16	unsigned long int	4 bajty	Od 0 do 4294967295
18	unsigned long long int	8 bajtów	Od 0 do 18446744073709551615
19	unsigned long long	8 bajtów	Od 0 do 18446744073709551615
20	short	2 bajty	Od -32768 do 32767
21	short int	2 bajty	Od -32768 do 32767
22	signed	4 bajty	Od -2147483648 do 2147483647
23	long	4 bajty	-2147483648 do 2147483647
24	signed long	4 bajty	-2147483648 do 2147483647
25	long long	8 bajtów	Od -9223372036854775808 do 9223372036854775807
26	long long int	8 bajtów	Od -9223372036854775808 do 9223372036854775807
27	signed long long	8 bajtów	Od -9223372036854775808 do 9223372036854775807
28	signed long long int	8 bajtów	Od -9223372036854775808 do 9223372036854775807

## Ogólny podział typów danych:



### Stale

Stała to taka zmienna, której wartość można przypisać tylko raz.

```
const float PI = 3.14159;
```

nie można przypisać w programie żadnej innej wartości, innymi słowy zapis: jest jednocześnie DEKLARACJĄ, DEFINICJĄ i ZAINICJOWANIEM stałej PI.

Przykład :

float a,b,c;	(DEKLARACJA)
const float euler = 2.7	(DEFINICJA)
y = 441;	(ZAINICJOWANIE zmiennej)

---

## Sposoby zapisu liczb zmiennoprzecinkowych

### Sposób 1

zapis części ułamkowej pisanej po kropce np. 13.345

### Sposób 2

sposób wykładniczy  $34.6e4=34.6*10^4=34.6*10000=346000$

## Przykład 5

### Temat:

Przykład demonstruje definiowanie zmiennych, wczytywanie ich wartości z klawiatury oraz operacje na nich. Całość z użyciem strumieni.

#### Wykonaj:

- Wpisz temat do zeszytu
- Wpisać Przykła5 (łącznie z komentarzem) do komputera. Nazwa pliku na dysku p5\_cztery\_pierwsze\_litery\_nazwiska
- Przepisać treści programu do zeszytu.

/\* Oto program pokazujący operacje wczytywania z klawiatury i obliczania wartości wyrażenia \*/

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

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    float metry;
    float kilometry;
    float k=1000;
```

```
    cout<<"Podaj liczbe metrow:";
```

```
    cin>>metry;                //wczytanie z klawiatury
```

```
    kilometry=metry/k;
```

```
    cout<<"Oto wynik przekształcenia:\n"<<metry<<"  metrow to  "<<kilometry<<"  kilometrow.";
```

```
    cout<<endl;
```

```
    system("PAUSE");
```

```
    return EXIT_SUCCESS;
```

```
}
```

---

Wy tłumaczenie instrukcji użytych w przykładzie powyżej.

#### Definicje zmiennych

##### *Pierwsza zmienna:*

Instrukcja **float metry** ; powoduje, że tworzymy zmienną o nazwie **metry**, która jest typu **float**. Typ ten służy do przechowywania liczb zmiennoprzecinkowych czyli takich, które mają przecinki.

##### *Druga zmienna:*

Tworzymy zmienną **kilometry**, która jest typu **float**.

##### *Trzecia zmienna:*

Definiujemy w niej zmienną k typu float i **od razu wpisujemy do niej wartość 1000**. Uwaga: W momencie utworzenia zmiennych tzw. lokalnych znajduje się w nich wartość przypadkowa.

W C++ można definiować zmienne wtedy, kiedy się zorientujemy, że są nam potrzebne.

---

#### Opis instrukcji

**cin >> cośtam** ; oznacza wczytanie z klawiatury do zmiennej **cośtam**. Jak widać, jest to ruch tak jakby odwrotny do wypisywania – tak nam sugerują strzałki.

Przeliczenia na kilometry. Dzielenie to /.

Po przeliczeniu wypisujemy na ekran wynik.

'\n' – nowa linia

---

Instrukcja cin → wczytywanie (pobierania) danych do programu z klawiatury lub pliku zewnętrznego.

np. cin>>promien;

Wczytanie z klawiatury liczby do zmiennej o nazwie **promien**.

np. cin>>x>>y

Wczytanie z użyciem jednej instrukcji cin dwóch zmiennych o nazwach **x** i **y**.

#### Operacja przypisania

do przypisania używa się znaku = a nie := jak w pascalu.

np.

p = (a+b)/2\*h;

x = y = 10;                      przypisanie zmiennym x i y wartości 10

#### Inkrementacja

Pojęcie to oznacza zwiększanie o jeden wartości zmiennej

np.

x++

wartość zmiennej x została zwiększona o jeden

#### Dekrementacja

Pojęcie to oznacza zmniejszanie o jeden wartości zmiennej

np.

y--

wartość zmiennej y została zmniejszona o jeden

---

### **OPERATORY ARYTMETYCZNE języka C++.**

<b>Operator</b>	<b>Nazwa</b>	<b>Tłumaczenie</b>	<b>Działanie</b>
+	ADDition	Dodawanie	Suma liczb
-	SUBstraction	Odejmowanie	Różnica liczb
*	MULTiplication	Mnożenie	Iloczyn liczb
/	DIVision	Dzielenie	Iloraz liczb
%	Moduluj	Dziel Modulo	Reszta z dzielenia

---

### **Operatory relacji języka C++.**

<b>Operator</b>	<b>Nazwa</b>	<b>przykład</b>	<b>wynik</b>
==	Równe	34==(37-3)	prawda
!=	różne od	9!=(15-6)	Fałsz
<	Mniejsze	10<11	prawda
>	Większe	9>8	fałsz
<=	mniejsze lub równe	7<=7	prawda
>=	większe lub równe	3>=4	fałsz

### **Operatory logiczne języka C++.**

AND    →    &&    koniunkcja    **i**  
OR     →     ||     alternatywa    **lub**  
NOT    →     !     zaprzeczenie    **nie**

np.

if (a>0 && a<100) printf("Dwucyfrowa"); else printf("100+");

#### **Zadanie 5**

Wykonaj program działania kalkulatora, który po wczytaniu dwóch liczb naturalnych dodatnich w jednej instrukcji poda wynik:

- Suma liczb
- Różnica liczb
- Iloczyn liczb
- Iloraz liczb
- Reszta z dzielenia

Zmienne z trzema literami nazwiska. Zmienne typu Int oprócz iloraz, ta zmienna typu float.

Zapisz wzór na iloraz  
Iloraz=1.0\*a\_kow/b\_kow;

Wygląd ekranu:

Np.

Podaj a=

Podaj b=

Wynik działania programu(wygląd ekranu):

suma=5

różnica=1

iloczyn=6

iloraz=0,66

reszta z dzielenia=1

### **Zadanie 6**

Wykonaj program zamiany radianów na stopnie po wczytaniu z klawiatury radianów.

Użyj zdefiniowanej stałej PI w programie jako 3.14159. Wypisz jednostki przy wyprowadzaniu wyników np. stopnie lub radiany.

Zmienne z trzema literami nazwiska.

Dane do sprawdzenia

Radiany=3.14159 otrzymasz wynik stopnie=180

### **Zadanie 7**

Wykonaj program zamiany stopni na radiany po wczytaniu z klawiatury stopni.

Użyj zdefiniowanej stałej PI w programie jako 3.14159.. Wypisz jednostki przy wyprowadzaniu wyników np. stopnie lub radiany.

Zmienne z trzema literami nazwiska.

Dane do sprawdzenia

stopnie=180 otrzymasz wynik Radiany=3.14159

---

## **funkcja scanf( )**

Znaczenie:

Funkcja scanf( ) wczytuje dane dowolnego typu i dowolnego rozmiaru (także łańcuchy znakowe).

Składnia funkcji scanf( ):

```
#include<stdio.h> // taki plik nagłówkowy musisz dołączyć  
int scanf(const char *format, [adres, ...]);
```

Opis:

- Zmienna **format** reprezentuje łańcuch formatujący (format string), w którym mogą być użyte, różne specyfikatory formatu (do wczytywania danych różnych typów).
- Jeśli działanie funkcji zakończy się poprawnie, funkcja zwraca **liczbę wczytanych danych** w przeciwnym razie zwraca **EOF**.
- Wielokropek** oznacza, że funkcja scanf( ) jest funkcją o zmiennej liczbie argumentów, tzn. można nią wczytać dowolną ilość danych.
- Funkcja **wczytuje dane** do pojawienia się jednego ze znaków: **spacja, ENTER, TAB, tabulacja pionowa, FF (znak końca stronicy)**.

Przykład z omówieniem→Wywołanie funkcji scanf( ):

Przykład 1

```
scanf("%d %f ",&x, &y);
```

Opis:

Funkcja scanf( ) wczytuje dwie wartości typu int i float podstawiane pod adres zmiennych x i y (przy użyciu operatora adresowego &).

Przykład 2

```
scanf("%s",str);
```

### Opis:

Wczytuje łańcuch znaków (specyfikator %s) i zapisuje go pod adres tablicy str (nie ma operatora &, bo nazwa tablicy jest wskaźnikiem - wskazuje adres pierwszego elementu) i dodaje na końcu wczytanego łańcucha znak '\0'. Wszystkie zmienne trzeba najpierw zadeklarować.

## **funkcja printf( )**

### Znaczenie:

Służy do wyprowadzania danych różnych typów na ekran.

### Składnia funkcji scanf( ):

Składnia funkcji printf( ):

```
#include<stdio.h>
```

```
int printf(const char *format_string, ...);
```

### Opis:

- Pierwszy argument format\_string to łańcuch znaków zawierający specyfikatory formatu (%d, %c itp. - patrz specyfikatory formatu).
- Wielokropek oznacza, że jest to podobnie jak scanf( ) funkcja o zmiennej liczbie argumentów, w tym miejscu znajduje się dowolna ilość danych wyprowadzanych na ekran. Ważne, aby ich liczba zgadzała się z liczbą specyfikatorów formatu. Jeśli działanie funkcji zakończy się poprawnie funkcja zwraca liczbę wyprowadzanych danych, w przeciwnym razie zwraca EOF.

### Przykład z omówieniem→Wywołanie funkcji printf( ):

#### Przykład 1

```
printf("suma dwóch liczb %d + %d = %d\n",x,y,suma);
```

### Opis

Funkcja printf( ) drukując na ekranie łańcuch znakowy (ten pomiędzy cudzysłowami (" ")) w miejsce specyfikatorów %d wstawia odpowiednio dane znajdujące się po przecinku: x, y, suma.

#### Przykład 2

```
printf("pierwiastek z liczby %f = %f\n",a,sqrt(a));
```

### Opis

Funkcja natomiast w miejsce drugiego specyfikator %f wstawia wartość zwracaną przez funkcję sqrt( ) (funkcje matematyczne). Tak więc argumentem funkcji printf( ) mogą być nie tylko zmienne ale także dowolne funkcje, czy wyrażenia.

W łańcuchu formatującym funkcji printf( ) znalazł się także znak specjalny '\n', oznaczający przejście do następnej linii, więc funkcja ta może zawierać dowolne znaki specjalne (ich spis znajdziesz tutaj).

## **Wzorce konwersji**

mają postać %s, %d, %f

### **a)**

%s - wyprowadź łańcuch znaków (s - String - łańcuch)

#### Przykład:

```
printf("%s","jakiś tekst");
```

format "%s" jest formatem domyślnym dla funkcji printf().

#### Przykład:

```
printf("%35s","jakiś tekst");
```

spowoduje uzupełnienie napisu spacjami do zadanej długości 35 znaków Funkcja printf() operuje tzw. POLEM WYJŚCIOWYM zwanym inaczej matrycą. Długość pola wyjściowego możemy określić przy pomocy liczb wpisanych pomiędzy znaki % oraz typ - np. s.

### **b)**

%c - wyprowadź pojedynczy znak (c - Character - znak)

Przykład:

```
printf("%c",'X');
```

spowoduje wydrukowanie litery X)

### **c)**

%d - wyprowadź liczbę całkowitą typu int w postaci dziesiętnej →d - Decimal - dziesiętny.



Przykład:

```
printf("%d", 1994);
```

d)

%f - wyprowadź liczbę rzeczywistą typu float w postaci.

Możemy także określić ilość cyfr przed i po przecinku (f - Floating point - zmienny przecinek).

Przykład:

```
printf("%f", 3.1416);
```

```
printf("%3.2f", 3.14159);
```

e)

%o - wyprowadź liczbę całkowitą typu int w postaci ósemkowej o - Octal - ósemkowa.

Przykład:

```
printf("%o", 255);
```

f)

%x - wyprowadź liczbę całkowitą typu int w postaci szesnastkowej x - hexadecimal - szesnastkowa.

%x lub %X - cyfry szesnastkowe a,b,c,d,e,f lub A,B,C,D,E,F.

g)

%ld - liczba całkowita "długa" - long int.

h)

%Lf - liczba rzeczywista poczwórnej precyzji typu long double float.

i)

%e - liczba w formacie wykładniczym typu 1.23e-05 (0.0000123)

j)

%g - automatyczny wybór formatu %f albo %e.

**Uogólnijmy sposób zastosowania wzorca**

formatu:                    %[przełączniki][szerokość\_pola][.precyzja][rozmiar]Typ

Przykład wyprowadzania złożonych napisów.

```
printf("Iloczyn 3 %c 5 %8s %d", '*', "wynosi ", 15);
```

**Wytłumaczenie**

"Iloczyn\_3\_" - wyprowadź jako łańcuch znaków.

%c - tu wyprowadź pojedynczy znak - '\*'.

\_5\_ - wyprowadź jako łańcuch znaków.

%8s - wyprowadź łańcuch "wynosi " uzupełniając go z przodu spacjami do długości 8 znaków.

%d - wyprowadź 15 jako liczbę dziesiętną.

Język C pozwala na wstawienie w specyfikatorze formatu liczby pomiędzy znak % z literę.

I tak:

pojedyncza liczba oznacza minimalną szerokość pola wyjściowego (ilość znaków), dzięki czemu wyprowadzone znaki mogą zajmować jak najmniej miejsca, np.:

```
printf("%6d \n", 12);
```

funkcja printf( ) wyprowadzi liczbę 12 w polu o szerokości 6 znaków z wyrównaniem do prawej.

umieszczenie przed liczbą znaku '-' spowoduje wyrównanie wyprowadzonych znaków do prawej, np.:

```
printf("%-7d \n", 12);
```

liczba 12 wyprowadzona zostanie w 7-io znakowym polu z wyrównaniem do lewej (domyślnie wyrównanie jest do prawej).

po liczbie (lub bez niej) można dodać znak kropki '.' i następną liczbę. Połączenie znak + kropka daje specyfikator precyzji:

dla liczb zmiennoprzecinkowych:

```
printf("%-10.3f \n", 12.34567);
```

wyprowadzona zostanie liczba 12.346 na 10-io znakowym polu z 3 liczbami po przecinku (zaokrąglonymi) z wyrównaniem do lewej.

dla liczb całkowitych i łańcuchów tekstowych:

```
printf("%10.6d \n", 123);
```

wyprowadzona zostanie liczba 000123 w 10-io znakowym polu na 6-iu pozycjach (jeśli dana liczba zajmuje mniejszą ilość znaków niż 6 wolne miejsca są wypełniane zerami). Liczba po kropce to maksymalna szerokość pola wyjściowego.

#### Podziały typów zmiennych:

a)typy na fundamentalne i pochodne.

b)typy wbudowane (czyli takie, w które C++ jest na starcie wyposażony) oraz definiowane przez użytkownika, czyli takie, które sobie sami wymyślimy.

#### Typy pochodne

Typami pochodnymi są np. tablica, wskaźnik, funkcja czy referencja.

#### Typy fundamentalne

Oto one:

*Typy całkowite:*

short int (lub short)

long int (lub long)

int

enum (typ wyliczeniowy)

*Typy rzeczywiste:*

float

double

long double

Trzy typy rzeczywiste są po to, żeby programista mógł wybrać dokładność swoich obliczeń. Najmniej w pamięci zajmuje typ float, ale jest przez to najmniej dokładny. Long double zajmuje najwięcej pamięci, ale jest najdokładniejszy.

*Typ znakowy: char.*

*Uwaga:*

Typy całkowite i char mogą występować w dwóch odmianach: z lub bez słowa unsigned. Postawienie unsigned przed np int oznacza, że typ ten może reprezentować liczby tylko dodatnie (bez znaku, czyli unsigned). Natomiast w wypadku typu char to jak zostanie zrozumiane słowo unsigned (lub jego brak), zależy od implementacji (czyli od kompilatora).

Do zmiennej typu char możemy podstawić dowolną literę (bądź cyfrę) ujętą w apostrofy, np:

char znakowa = 'a' ;

znakowa = '5' ; // przypisanie cyfry, a nie liczby

Do zmiennej znakowej można też przypisać jeden z tzw. znaków sterowanych. Jeśli do zmiennej znakowej przypiszemy np. '\n', to zmienna ta będzie zawierała znak nowej linii. Oto lista wszystkich znaków sterowanych:

'\b' – cofnięcie o jedną pozycję

'\f' – nowa strona

'\n' – nowa linia

'\r' – powrót karetki

'\t' – tabulator poziomy

'\v' – tabulator pionowy

'\a' – sygnał dźwiękowy

'\\' – oznaczenie backslasha

'\'' – apostrof

'\"' – cudzysłów

'\0' – NULL, znak o kodzie zero

'\?' – znak zapytania

Ostatnie pięć znaków służy do wypisywania specjalnych znaków, które są już używane w C++. Przykładowo, żeby wypisać cudzysłów, nie możemy napisać po prostu “, trzeba napisać \“.

Do zmiennych znakowych można też przypisać wartość danego znaku z tablicy ASCII, ale musi to być liczba podana w zapisie ósemkowym bądź szesnastkowym.

*Typ void,*

który oznacza mniej więcej type: nic, pustka.

*Typ enum.*

Typ wyliczeniowy. Zmienna typu wyliczeniowego może przyjmować tylko takie wartości, jakie wyliczyliśmy przy definicji. Np.

```
enum dzien {pon, wto, sro, czw=20, pia, sob, nie} ;  
 dzien zmienna = pon ;  
 zmienna = 21 ; // nielegalne!
```

Trzeba zaznaczyć, że każdy element w liście wyliczeniowej ma jakąś wartość liczbową. W naszym przykładzie: pon=0, wto=1, sro=2, czw=20, pia=21, sob=22, nie=23. Widać, że numeracja zaczyna się od zera (jeśli nie określiliśmy inaczej), natomiast jeśli jakiemuś elementowi nadamy inną wartość (niewynikającą z kolejności), to następne elementy listy będą numerowane zaczynając od tej zmienionej wartości.

### Wypisywania i podawanie liczb.

Jeśli podamy liczbę normalnie to kompilator zrozumie to jako liczbę dziesiętną.

Jeśli liczbę poprzedzimy zerem, np 034, to jest to zapis liczby w systemie ósemkowym (oktalnym).

Jeśli poprzedzimy liczbę znakami 0x (zero i litera x), to będzie to zinterpretowane jako liczba zapisana w systemie szesnastkowym. Oczywiście można używać występujących w systemie szesnastkowym liter a,b,c,d,e,f. Jeśli PO liczbie dopiszemy L to liczba ta będzie traktowana jako liczba typu long. To samo z typem unsigned, tyle że tutaj dopisujemy literę U (bądź małe u).

### String

String ciąg znaków ujęty w cudzysłów, np. "Taki sobie napis". Trzeba wiedzieć, że komputer przechowuje taki string jako tablicę zmiennych znakowych, czyli w tym przypadku tablica będzie się składać ze zmiennych, które będą miały wartość odpowiednio: T, a, k, itp.

### Zakresy ważności nazw i czas życia obiektów

Czas życia obiektu to okres pomiędzy definicją obiektu (czyli przydzieleniem pamięci dla niego), a destrukcją obiektu (czyli zwolnieniem pamięci). Natomiast zakres ważności nazwy obiektu to fragment (czasami całość) programu, w którym dana nazwa obiektu jest znana kompilatorowi. Jaka jest między nimi różnica?! Ano taka, że w pewnym momencie w programie obiekt może istnieć (czyli żyje, jego czas życia się jeszcze nie skończył), ale może nie być dostępny.

### wyróżniamy trzy rodzaje zakresów.

**Zakres lokalny** to fragment programu objęty parą nawiasów {}.

```
main()  
{  
    ... // jakieś tam instrukcje  
    { // otwarcie bloku lokalnego  
        int a ; // definiujemy sobie zmienną a  
        ... // tutaj nazwa a jest znana, możemy sobie na zmiennej a pracować  
    }  
    ... // tutaj nazwa a nie jest znana
```

Takim blokiem jest też funkcja, zatem we wnętrzu funkcji zdefiniowane nazwy nie są znane na zewnątrz.

**Zakres globalny** (pliku) oznacza, że jeśli zdefiniujemy jakąś zmienną na zewnątrz wszystkich funkcji, to jest ona znana w każdej funkcji, czyli: w całym obszarze pliku dana nazwa jest znana. Jeśli chcemy, aby nazwa była znana we wszystkich plikach, czyli w całym programie, to przed deklaracją należy postawić słówko extern, czyli z angielskiego: na zewnątrz.

**Zakres obszaru klasy** Nazwa użyta wewnątrz definicji klasy jest znana tylko wewnątrz klasy (i dostępna dla jej funkcji składowych)

### Przykrywanie zmiennych.

Może się zdarzyć sytuacja, że będzie istniała globalna zmienna o nazwie x, a my w jakimś lokalnym bloku definiujemy drugą też o nazwie x. Taka sytuacja może mieć miejsce (pomimo tego, że są dwie zmienne o tych samych nazwach). W takim razie, skoro funkcja zna wszystkie zmienne globalne, jak ma ona rozróżnić, do której się właśnie zwracamy?? Otóż, jeśli w wyrażeniach będziemy używali x normalnie, tak jakby była jedna zmienna o nazwie x, to funkcja zrozumie, że chodzi nam o zmienną LOKALNĄ. Jeśli natomiast przed nazwą x postawimy dwa dwukropki, czyli będziemy mieć ::x, to funkcja będzie pracowała na zmiennej GLOBALNEJ.

Tak się uтарыło w C++, że operator `::` oznacza mniej więcej “przejście na wyższy poziom” (tak przynajmniej ja to odbieram).

---

### **Przykład 6**

#### **Temat:**

Przykład na formatowanie i wyprowadzanie danych.

#### **Wykonaj:**

- Wpisz temat do zeszytu
- Wpisać Przykład 6 (łącznie z komentarzem) do komputera. Nazwa pliku na dysku p6\_cztery\_pierwsze\_litery\_nazwiska
- Przepisać treści programu do zeszytu.

```
#include <cstdlib>
#include <iostream>
#include <conio.h>
#include <stdio.h>

using namespace std;

int main(int argc, char *argv[])
{
    float x,y;
    float wynik;

    printf("Zamieniam ułamki zwykłe na dziesiętne\n");
    printf("\nPodaj licznik ułamka:");
    scanf("%f",&x); /*pobiera liczbę z klawiatury*/
    printf("\nPodaj mianownik ułamka:");
    scanf("%f",&y);

    wynik=x/y; /*tu wykonuje się dzielenie*/

    printf("\n%f: %f=%f", x,y, wynik);
    cout<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### **Przykład 7**

#### **Temat:**

Przykład na skomplikowane formatowanie i wyprowadzanie danych.

#### **Wykonaj:**

- Wpisz temat do zeszytu
- Wpisać Przykład 7 (łącznie z komentarzem) do komputera. Nazwa pliku na dysku p7\_cztery\_pierwsze\_litery\_nazwiska
- Przepisać treści programu do zeszytu.
- Wykonaj i uzupełnij tabelę w zeszycie

W programie	Na ekranie
<code>printf("Iloczyn 3 %c 5 %8s %d", '*', "wynosi ", 15);</code>	Wpisz wygląd ekranu
<b>Wytlumaczenie</b> "Iloczyn_3_" - wyprowadź jako łańcuch znaków. %c - tu wyprowadź pojedynczy znak - '*'. _5_ - wyprowadź jako łańcuch znaków. %8s - wyprowadź łańcuch "wynosi_" uzupełniając go z przodu spacjami do długości 8 znaków. %d - wyprowadź 15 jako liczbę dziesiętną.	Zostaw puste
Zostaw puste	Zostaw puste
<code>printf("Iloczyn 3 %c 5 %9s %d", 'x', "wynosi ", 3*5);</code>	Wpisz wygląd ekranu
<b>Wytlumaczenie→napisz sam</b>	Zostaw puste

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

```
int main(int argc, char *argv[])
{
    printf("Skomplikowany napis:\n");
    printf("Iloczyn 3 %c 5 %8s %d", '*', "wynosi ", 15);
    printf("\n");
    system("PAUSE");
    printf("\nWyrażenie jako argument:\n");
    printf("Iloczyn 3 %c 5 %9s %d", 'x', "wynosi ", 3*5);
    printf("\n\n\n");
    printf("Przyjrzyj się i naciśnij klawisz...");
    printf("\n");
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### Zadanie 8

Napisz program, który po wczytaniu liczby x poda tę liczbę w formie szesnastkowej w następującej formie:  
np. . dla wczytanej zmiennej x równej tysiąc otrzymamy wygląd ekranu →

wczytana liczba x=1000  
liczba w formie szesnastkowej x=3e8

Użyj funkcji printf oraz scanf.

Pamiętaj, że zmienna „x” musi być całkowita (taki format wczytywania danych %d).

Nazwy zmiennych tak, aby zawierały trzy pierwsze litery nazwiska.

Jak wykonać szukaj w tej instrukcji np., jaka zmienić na ósemkowy?

### Zadanie 9

Uwagi:

- Użyj funkcji printf oraz scanf
- Nazwy zmiennych tak, aby zawierały trzy pierwsze litery nazwiska.

Napisz program, który po wczytaniu liczby x (naturalnej) poda wynik działania programu w następującej formie:  
np. dla wczytanej zmiennej x równej trzy otrzymamy wygląd ekranu →

wczytana liczba x=3  
kwadrat liczby x=3  $3^2=9$   
sześciąt liczby x=3  $3^3=27$

### **Przykład 8**

#### **Temat:**

Program oblicza pole oraz obwód trójkąta po podaniu kąta ( w stopniach) i przeciwprostokątnej.

#### **Wykonaj:**

- Wpisz temat do zeszytu
- Wpisać Przykład 8 (łącznie z komentarzem) do komputera. Nazwa pliku na dysku p8\_cztery\_pierwsze\_litery\_nazwiska
- Przepisać treści programu do zeszytu.
- Narysuj rysunek trójkąta prostokątnego w zeszycie oznacz boki a b c oraz alfa. Innym kolorem zaznacz te zmienne na rysunku, które są danymi a w zadaniu. Pod rysunkiem zapisz wzory na zamianę stopni na radiany, a i b.

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

using namespace std;

int main(int argc, char *argv[])
{
    float const PI=3.14159;//definicja stalej
    float a,b,c;
    float pole,obwod,alfa_x,alfa_s;
    printf("Umiem obliczac pole oraz obwod trojkata\n");
    printf("po podaniu kata i przeciwprostokatnej\n");
    printf("podaj przeciwprostokatna\n");
    scanf("%f",&c);
    printf("podaj kat w stopniach\n");
    scanf("%f",&alfa_s);
    printf("wczytane dane");
    printf("\nc= %.2f",c);
    // w linii wyżej definiowane są dwa miejsca po przecinku
    printf(" cm");
    printf("\nalfa= %f",alfa_s,"st");
    printf(" st\n");
    alfa_x=alfa_s*PI/180;
    a=c*cos(alfa_x);
    b=c*sin(alfa_x);
    pole=a*b/2;
    obwod=a+b+c;
    printf("wyniki");
    printf("\na= %f",a);
    printf(" cm");
    printf("\nb= %f",b);
    printf(" cm");
    printf("\npole= %f",pole);
    printf(" cm^2");
    printf("\nobwod= %f",obwod);
    printf(" cm");
    cout<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

---

### ZADANIE 10

Nazwy zmiennych tak, aby zawierały trzy pierwsze litery nazwiska np. a\_kow typu float.

Napisz program obliczający przy stałej przeciwprostokątnej C=30 cm (wpisz jako stałą do programu )

float const C=30;

i wczytywanej z klawiatury przyprostokątnej A. Do wczytania użyj instrukcji scanf

- Narysuj rysunek trójkąta prostokątnego w zeszycie oznacz boki a b c oraz alfa. Innym kolorem zaznacz te zmienne na rysunku, które są danymi a w zadaniu. Pod rysunkiem zapisz wzory na b, pole, cos\_alfa, sin\_alfa, tg\_alfa.
- nieznany bok trójkąta
- pole trójkąta
- cosinus, sinus, tangens dowolnego kąta ostrego
- kąty trójkąta wyrażone w stopniach

Przy wynikach pisz jednostki, formatuj wydruki do trzech miejsc po przecinku. Trzy miejsca po przecinku to "%.3f"

Do obliczenia kątów użyj instrukcji atan(x) np. ALFA = atan(1) to ALFA=PI/4 czyli ALFA:=45 stopni.

Testuj program wykonując po jednym punkcie.

Dane do testowania:

A=15 to:

B=

### ZADANIE 11

Temat: Obliczenia na wektorach → kąt między wektorami.

Nazwy zmiennych tak, aby zawierały trzy pierwsze litery nazwiska.

Wykonaj czynności oraz obliczenia

- Wczytać punkty A B C D.
- Wypisz te punkty po wczytaniu np. A(2,-3).
- Obliczyć współrzędne wektorów AB i CD
- Wypisz te wektory po wczytaniu np. AB=[2,-3].
- Oblicz kąt między wektorami AB, CD. Podaj kąt w stopniach w formatach dogodnych dla użytkownika np. do jednego miejsca po przecinku.
- Użyj komentarzy. Zmień kolor tła tego okna oraz kolor pisania.

Wskazówka:

Kąt między wektorami obliczmy:

$v1=[a1,b1]$      $v2=[a2,b2]$      $\cos\_alfa=(a1*a2+b1*b2)/(dv1*dv2)$

dv1,dv2–długości wektorów. Długość wektora jest to pierwiastek z sumy kwadratów ich współrzędnych, czyli:

$$dv1 = \sqrt{a1^2 + b1^2}$$

$$dv2 = \sqrt{a2^2 + b2^2}$$

Następnie sin\_alfa z jedynki trygonometrycznej potem tangens.

$$\sin\_alfa = \sqrt{1 - (\cos\_alfa)^2}$$

$$\tan\_alfa = \frac{\sin\_alfa}{\cos\_alfa}$$

Do obliczenia kata użyj atan(tangens).

$$alfa\_x = a \tan(\tan\_alfa)$$

Z radianów zamień na stopnie.

$$alfa\_s = \frac{alfa\_x * 180}{\Pi}$$

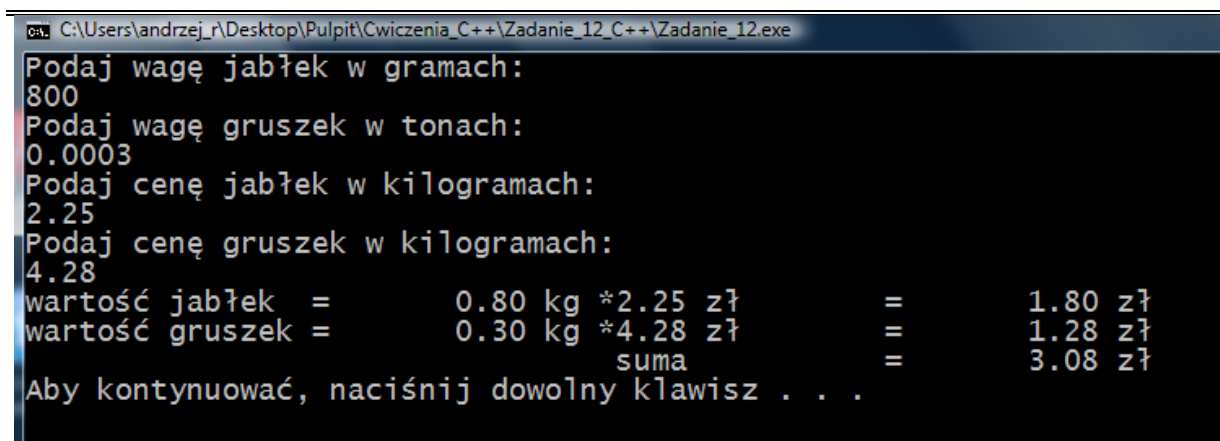


### ZADANIE 11a

Zenek kupił w warzywniaku jabłuszka i gruszki. Podaj ile zapłacił za jabłka ile za gruszki a ile razem. Zeniu lubi żarciki, więc wagę jabłek podał w gramach 800 a gruszek w tonach 0,0003. Sklepowa zdębiała, ponieważ ceny które znała wyliczone były w kilogramach, ale dała radę bo skończyła ZSE w Gdańsku i napisała krótki programik w C++. Program napisany został w sposób umożliwiający wprowadzanie różnych wartości wejściowych. Wynik podała wg poniższego wzoru:

wartość\_jabłek=800\*2,25zł=... zł  
wartość\_gruszek=0.0003\*4,28=...zł  
suma=...zł

przypilnuj aby wyświetlane znaki = znalazły się jeden pod drugim



```
C:\Users\andrzej_r\Desktop\Pulpit\Cwiczenia_C++\Zadanie_12_C++\Zadanie_12.exe
Podaj wagę jabłek w gramach:
800
Podaj wagę gruszek w tonach:
0.0003
Podaj cenę jabłek w kilogramach:
2.25
Podaj cenę gruszek w kilogramach:
4.28
wartość jabłek =      0.80 kg *2.25 zł      =      1.80 zł
wartość gruszek =      0.30 kg *4.28 zł      =      1.28 zł
                        suma                =      3.08 zł
Aby kontynuować, naciśnij dowolny klawisz . . .
```

## Sprawdzian przykładowy z drugiej części CPP

### Zadanie 1 (na ocenę dopuszczającą/dostateczną)

Napisz program obliczający natężenie prądu elektrycznego  $I[A]$ , po podaniu napięcia  $U[V]$  oraz oporu elektrycznego  $R[ohm]$ . Wypisz wczytane dane oraz wyniki z jednostkami. Użyj formatowania danych dwa miejsca po przecinku. Użyj komentarzy. Wzór:  $I=U/R$ .

Przykładowy wydruk:

Dane:

$U=5.00 [V]$

$R=2.00 [ohm]$

Obliczone:

$I=2.50 [A]$

### Zadanie 2 (na ocenę dopuszczającą/dostateczną)

Użyj instrukcje printf oraz scanf. Napisz program, który po podaniu trzech liczb naturalnych:

a) Wyświetli liczbę w postaci 
$$\frac{B}{A \cdot C + B}$$

b) zamieni liczbę na postać ułamka niewłaściwego (Ułamek niewłaściwy – jest to ułamek, w którym licznik jest większy lub równy mianownikowi). Skorzystaj ze wzoru:

$$\frac{A \cdot C + B}{C}$$

c) zamień ułamek na postać dziesiętną trzy miejsca po przecinku. W celu zamiany podziel  $(A \cdot C + B)/C$  i wypisz na ekran a uzyskasz zamianę.

### Wygląd ekranu

Podaj całości

A=

Podaj licznik ułamka

B=

Podaj mianownik ułamka

C=

wczytanymi liczby są:

A=1 B=2 C=3

uwaga: znaki równości są w jednej kolumnie

$$\frac{B}{A \cdot C + B} = \frac{2}{1 \cdot 3 + 2} = \frac{2}{5} = \text{wynik w postaci ułamka dziesiętnego trzy miejsca po przecinku}$$

np. po wczytaniu A=1 B=2 C=3 uzyskasz ekran:

$$\frac{2}{5} = 1.667$$

uwaga: kreska ułamkowa w postaci minusów

### Zadanie 3 (na ocenę dobrą)

Napisz program obliczający  $V$  (objętość) oraz  $P_c$  (pole powierzchni całkowitej) dla prostopadłościanu prawidłowego czworokątnego po podaniu krawędzi podstawy  $A[cm]$  oraz przekątnej ściany bocznej  $D[cm]$ . Wypisz wczytane dane oraz wyniki z jednostkami. Użyj formatowania danych dwa miejsca po przecinku. Użyj komentarzy.

Przykładowy wydruk:

Dane:

$A=5.00 [cm]$

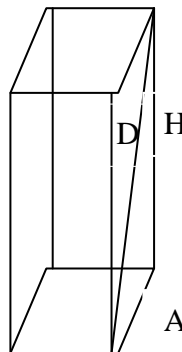
$D=13.00 [cm]$

Obliczone:

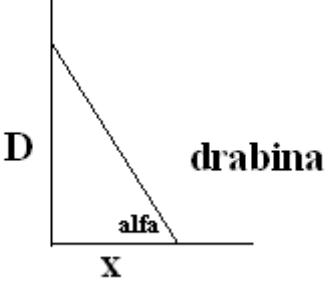
$V=300.00 [cm^3]$

$P_c=290.00 [cm^2]$

$$V=A^2 \cdot H \quad P_c=2A^2+4AH$$



**Zadanie 5 (na ocenę bardzo dobrą)**

	<p><u>Wczytaj:</u> D-wysokość oparcia drabiny X- odległość oparcia drabiny</p> <p><u>Oblicz:</u> długość drabiny [m] kąt nachylenia drabiny w [stopniach]</p>
---	---

**Zadanie 6 (na ocenę celującą)**

Trzy punkty  $A(x_1, y_1)$   $B(x_2, y_2)$   $C(x_3, y_3)$  tworzą trójkąt. Wczytać do komputera punkty A B C. Oblicz oraz wypisz na ekranie monitora równanie prostej przechodzącej przez punkty AB w postaci  $y=ax+b$  oraz równanie dowolnej wysokości. Użyj komentarze, wykonaj formaty wydruków oraz zapisz jednostki. W celu urozmaicenia prezentacji graficznej wyników programu użyj definiowania okien zmień kolor tła tego okna na fioletowy oraz kolor pisania na zielony.

Dane:

A(1,3)

B(3,1)

C(3,3)

Obliczone:

prosta AB  $y = -x + 4$

wysokość z punktu C  $y = x + 0$

## Część 3

# Programowanie obiektowe.

### **ZADANIE →Praca domowa**

Zapisz numer pytania ( przed pytaniem np. Pytanie 1) poniżej zapisz treść pytania. Treść pytania podkreśl na zielono a pod treścią pytania zapisz odpowiedź.

#### **Pytanie 1**

Podaj definicję programowania obiektowego (obiekt, stan, pole, zachowanie)

#### **Pytanie 2**

Dlaczego stosujemy programowanie obiektowe ( 3 przyczyny?).

#### **Pytanie 3**

Jakie są powiązania pomiędzy funkcjami i zmiennymi opisujące dany przedmiot w programowaniu\_strukturalnym?

#### **Pytanie 4**

Jakie są powiązania pomiędzy funkcjami i zmiennymi opisujące dany przedmiot w programowaniu\_obiektowym?

#### **Pytanie 5**

Co to jest klasa?

#### **Pytanie 6**

Jak jest zbudowana klasa oraz podaj przykład definiowania klasy? Podaj uwagę.

#### **Pytanie 7**

Podaj definicję obiektu.

#### **Pytanie 8**

Podaj przykład kodu programu jak tworzony jest obiekt.

#### **Pytanie 9**

Co to jest instancja w programowaniu obiektowym?

#### **Pytanie 10**

Jak rozumiesz interfejs w programowaniu obiektowym?

#### **Pytanie 11**

Omów na przykładzie, w jaki sposób dołączamy do klasy pola.

#### **Pytanie 12**

Omów na przykładzie, w jaki sposób dołączamy do klasy metody.

#### **Pytanie 13**

Omów na przykładzie, w jaki sposób dołączamy do klasy metody z użyciem prototypu.

#### **Pytanie 14**

Co to jest identyfikator dostępu (zasięgu) na podstawie schematu?

#### **Pytanie 15**

Na podstawie przykładu podaj jak tworzone są obiekty.

#### **Pytanie 16**

Na podstawie przykładu podaj jak tworzone są odwołania do obiektu.

#### **Pytanie 17**

Omów konstruktor korzystając z trzech punktów zapisanych w treści instrukcji. Co się dzieje, gdy nie zdefiniowaliśmy destruktora?

#### **Pytanie 18**

Co to jest przeciążenie konstruktora?

#### **Pytanie 19**

Omów destruktor korzystając z czterech punktów zapisanych w treści instrukcji.

#### **Pytanie 20**

Co to jest hermetyzacja danych?

**Pytanie 21**

Co oznaczają dyrektywy: public, protected, private używane w ciele klasy.

**Pytanie 22**

Omów pola statyczne korzystając z czterech punktów zapisanych w treści instrukcji.

**Pytanie 23**

Na podstawie przykładu omów zagnieżdżenie klas

**Pytanie 24**

Omów dziedziczenie z uwzględnieniem:

-jak się je wykonuje

-co ono umożliwia

**Pytanie 25**

Omów pojęcie klasy bazowej i pochodnej na podstawie przykładu

**Pytanie 26**

Omów pojęcie polimorfizmu.

**Pytanie 27**

Omów funkcje wirtualne.

## Koncepcja programowania obiektowego

### Programowanie obiektowe definicja:

Programowanie obiektowe (ang. object-oriented programming) — sposób programowania, w którym programy definiuje się za pomocą:

- **obektów** (obiekty powoływane są do życia wirtualnego przez programistę, mogą być kasowane przez programistę) np. okienka w windows, okienko jest obiektem,
- obiekt ma jakiś **stan** (czyli dane, w programowaniu obiektowym nazywane najczęściej **polami**) np. wielość okienka w Windows,
- obiekt wykazuje pewne **zachowanie** (czyli posiada pewne procedury w programowaniu obiektowym nazywane metody). Np. jest zdefiniowana metoda do zmiany wielkości okienka.

Czyli **obiektyowy program komputerowy** wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

### **Dlaczego stosujemy programowanie obiektowe:**

- Jest to bardziej naturalne podejście przez programistę podczas tworzenia programów ponieważ otaczający nas świat to różnego rodzaju przedmioty. Tworząc program komputerowy dokonujemy pewnego ich odwzorowania rzeczywistości np. definiujemy obiekt samochód. Definiujemy pola czyli jego stan np. rok produkcji, marka, kolor. Definiujemy również metody czyli co umie np. hamowanie, przyspieszanie. Definicja obiektu jego metod i pól znajduje się w klasie. Następnie na podstawie zdefiniowanego obiektu (może mieć pola oraz metody) powołujemy obiekt w pamięci wirtualnej
- Programowanie obiektowe pozwala na lepsze gospodarowanie pamięcią komputera, ponieważ gdy obiekt staje się zbędny możemy go usunąć co zwolni pamięć zajmowaną przez ten obiekt. W programowaniu strukturalnym aby zwalniać pamięć konieczne było stosowanie dynamicznych struktur danych np. stos czy kolejka.
- W programowaniu obiektowym dobrze napisana i przetestowana klasa pozwala na stosowanie przez innych programistów. Zaoszczędza to czas i środki pieniężne.

W programowaniu proceduralnym (strukturalne) funkcje i zmienne opisujące dany przedmiot nie były ze sobą powiązane.

W przypadku programowania zorientowanego na obiekt (programowania obiektowego) dokonuje się powiązania metod (funkcji programu) z danymi (zmiennymi) definiującymi przedmiot. Wszystko to grupuje się w tzw. klasie zawierającej zarówno dane (pola) definiowanego przedmiotu, jak i funkcje (metody). W ten sposób definicje przedmiotu i jego właściwości znajdują się w jednym miejscu programu. W języku C++, opis klasy jest dokonywany za pomocą pól (zmienne) i metod(funkcje).

### **Definicja klasy**

Klasa jest to złożony typ będący opisem (definicją) obiektu/obiektów wraz z definicją pól i metod obiektów.

### **Definiowanie klasy CPP**

Definicja klasy zawiera dwie części: nagłówek składającego się ze słowa kluczowego class, po którym następuje nazwa klasy oraz z ciała klasy ograniczonego parą nawiasów klamrowych i zakończonego średnikiem.

np.

```
class TPunkt
{

};
```

Uwaga! Definicja klasy musi zawsze kończyć się średnikiem.

### **Definicja obiektu**

Obiekt jest praktyczną realizacją (wirtualną) na podstawie klasy (pola i metody zdefiniowane w tej klasie) wirtualnej struktury.

### **Instancja**

Instancja to w programowaniu obiektowym możliwość powołania do „życia” obiektu określonej klasy. Instancja oznacza konkretny obiekt istniejący w pamięci operacyjnej.

### **Interfejs w programowaniu obiektowym**

Program musi mieć możliwość komunikowania się z obiektami, zmieniać stan obiektu czy też obiekt powinien wykonać, jaką czynność. Połączenie programu z obiektem nazywane jest interfejsem. Interfejs pozwala wpływać na stan obiektu oraz czynności, które wykonuje obiekt.

### **Dołączenie do klasy pól**

np.

```
class TPunkt
{
public:           //sekcja public zostanie wyjaśniona później.
int x;          //pole typu int o nazwie x
int y;          //pole typu int o nazwie y
};
```

## **Dołączenie do klasy metod(funkcji)**

np.

```
class TPunkt
{
public:
    int x;
    int y;
    void fun(void){ }           //definicja funkcji fun() w klasie
};
```

## **Dołączenie do klasy metod(funkcji), która znajduje się poza ciałem klasy.**

W przypadku, gdy definicja funkcji jest obszerna można ją przenieść poza klasę.

W ciele klasy musimy jednak podać uprzedzenie, że metoda(funkcja) będzie definiowana poza klasą. Takie uprzedzenie nazywane jest **prototypem**.

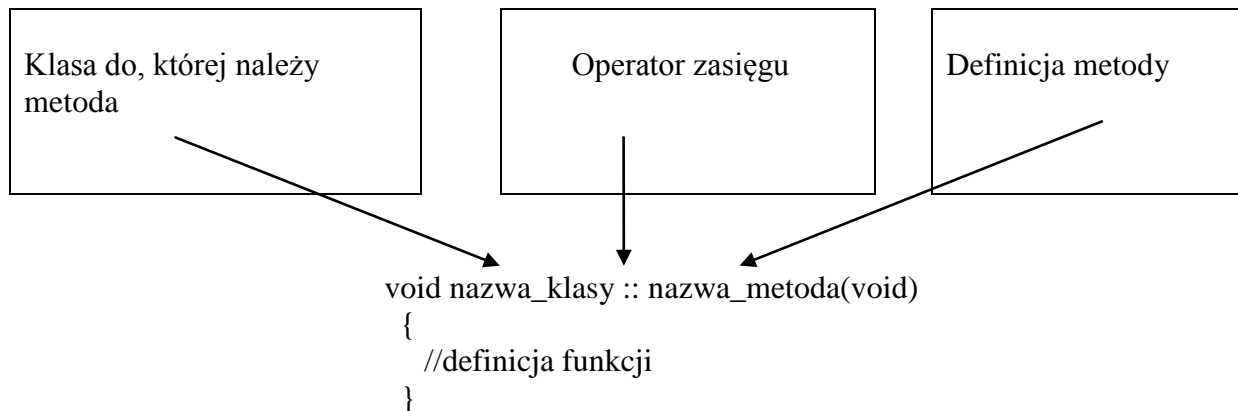
np.

```
class TPunkt
{
public:
    int x;
    int y;
    void fun(void); //deklaracja funkcji fun() jako prototyp
};

void TPunkt::fun()
{
    //definicja funkcji fun() poza klasą
}
```

W klasie zadeklarowano metodę (funkcję) o nazwie fun. Definicja funkcji znajduje się pomiędzy nawiasami klamrowymi.

W ten oto sposób możemy sprawić, że ciało klasy będzie zawierało tylko deklaracje funkcji, a ich definicje będą znajdowały się w innym miejscu (lub nawet w innym pliku źródłowym). Aby jednoznacznie określić, która funkcja należy do określonej klasy stosuje się identyfikator dostępu :: podając. Nazywany jest on również jako **operator zasięgu**.



### Tworzenie obiektów

Proces tworzenia obiektu nazywamy instancjonowaniem lub konkretyzacją klasy. Tworzenie obiektów dokonuje się tak samo (podobnie) jak tworzenie zmiennych poprzez:  
nazwa\_klasy nazwa\_instancji

np.

```
int i;                //zmienna typu int o nazwie i
Tsamochod fiat;      //obiekt klasy Tsamochod o nazwie fiat
```

### Odwołanie się do obiektu

Do obiektów odwołujemy się poprzez:

- a) „ . ” kropkę
- b) lub poprzez „ -> ”

np.:

```
class TPunkt
{
    int x;
    int y;
public:
    int kolor;
    void putx(int xx){x=xx;}
    void puty(int y){y=yy;}
    void kolor(int kk){kolor=kk;}
};
```



```

main(void)
{
    TPunkt p1;           //obiekt stworzony statycznie
    p1.kolor=10;         //odwołanie do pola przez kropkę, najpierw podajemy nazwę obiektu
                        // „p1”i kropce nazwę pola „kolor”
    p1.putx(20);         //odwołanie do funkcji przez kropkę najpierw podajemy nazwę obiektu
                        // „p1”i kropce nazwę metody „putx”

    TPunkt *wsk;         //stworzenie skaźnika „wsk” - wskaźnik na obiekt klasy TPunkt

    wsk = new TPunkt;    // stworzenie obiektu z użyciem operatora new

    wsk->kolor=10;        //odwołanie do pola przez -> (zamiast kropki)
    wsk->putx(20);        //odwołanie do funkcji przez -> (zamiast kropki)

    delete wsk;          //operator delete skasowanie wskaźnika
}

```

### Konstruktor

- Konstruktor nazywamy specjalną funkcję (metodę) automatycznie uruchamianą w trakcie definiowania(tworzenia) oraz jego kopiowania obiektu, pozwalającą na nadanie początkowych wartości danym obiektu.
- To specjalna funkcja jest o tej samej nazwie jak klasa.
- Konstruktor deklaruje się (i definiuje) podobnie do zwykłej funkcji

### Przeciążenie konstruktora

Przeciążenie konstruktora następuje wtedy gdy istnienie wielu konstruktorów o tych samych nazwach. Konstruktory te muszą się jednak różniących ilością argumentów.

### Destruktor

- Destruktorem nazywamy specjalną metodę bezparametrową, która jest wywoływana zawsze automatycznie w momencie usuwania obiektu.
- Destruktor ma taką samą nazwę jak klasa ale nazwa destruktora jest poprzedzona znakiem "~".
- Jeśli w swojej klasie nie zdefiniujesz destruktora, zostanie on wygenerowany przez kompilator.
- Wywołanie destruktora zwalnia pamięć zajmowaną przez obiekt.

### Hermetyzacja (enkapsulacji lub kapsułkowania)

Wewnątrz ciała klasy znajdują się pola i metody. Część pól i metod można odpowiednio ukryć przed "zewnętrznym światem" klasy tak jak to ma miejsce z przedmiotami ze świata rzeczywistego. Czyli np. ukrywasz dla obiektów zdefiniowane i stworzonych przez inne klasy. W ciele klasy deklaruje się odpowiednie sekcje:

public	→	widoczną dla obiektów klasy, podklas oraz globalnie w programie,
protected	→	widoczną dla obiektach klasy i podklas,
private	→	widoczną tylko dla obiektów jednej klasy.

Uwaga: Gdy **nie** użyjesz słowa private to składowe klasy **przez domniemanie** będą private.

np.

```

class TPunkt
{
    public:
        metody i pola
    private:
        metody i pola
    protected:
        metody i pola
};

```

#### Uwagi

- Liczba deklarowanych sekcji jest dowolna.
- W przypadku braku deklaracji sekcji domyślnie jest przypisana sekcja prywatna.

np.

```

class TPunkt
{
    fun1(){}
    public:
        fun2(){}
        fun3(){}
    private:
        fun4(){}
        int x;
        int y;
    public:
        fun5(){}
};

```

W tym przykładzie metody: fun2(), fun3() i fun5() są publiczne, a metody fun1(), fun4() oraz pola x i y są prywatne

#### **Przykład**

Można tak zaprojektować klasę, że dostęp do pól obiektu będzie realizowany tylko przez odpowiednie funkcje.

```

class TPunkt
{
    private:
        int x;
        int y;
    public:
        int getx()      {return x;}
        int gety()      {return x;}
        void putx(int xx) {x=xx;}
        void puty(int yy) {y=yy;}
};

```

Funkcje `getx()`, `gety()`, `putx()` i `puty()` znajdują się w sekcji `public` i są widoczne poza obiektem klasy.

Pola `x` i `y` **nie** są widoczne poza obiektem klasy, ale jedynie wewnątrz obiektu (mają do niej dostęp metody klasy, czyli nasze funkcje `getx()`, `gety()`, `putx()` i `puty()`). Z tego względu te funkcje stanowią **"interfejs"** dostępu do pól obiektu klasy `TPunkt`. Daje to nam kontrolę nad wprowadzaniem i odczytem pól `x` i `y` poprzez odpowiednie definicje metod.

### Pola statyczne

Pola klasy mogą być zadeklarowane jako statyczne wewnątrz deklaracji klasy. Do tego celu stosuje się **atrybut `static`**.

- Tak zadeklarowane pole klasy jest współdzielone przez wszystkie obiekty danej klasy.
- Zmienne statyczne w języku C pełnią rolę zmiennych globalnych z ograniczonym zakresem dostępu tylko do poziomu funkcji, w której są deklarowane.
- Element `static` nie jest częścią obiektów danej klasy i deklaracja elementu statycznego w deklaracji klasy nie jest definicją. Taką definicję wprowadzana się poza klasą.
- Zmienna statyczna zadeklarowana w bloku funkcji, zgodnie z definicją pamięta swoją wartość pomiędzy wywołaniami funkcji, co nie jest możliwe przy użyciu zwykłych zmiennych.

### Przykład

```
class TPunkt
{
    private:
        int x;
        int y;
    public:
```

```
        static int a;           //definicja pola statycznego
}
```

```
int TPunkt::a = 1;           //deklaracja pola statycznego poza klasą TPunkt
```

```
// Odwołanie do takiego pola jest możliwe przez podanie nazwy klasy z operatorem zakresu ::
// lub przez operator . lub ->.
```

```
TPunkt::a = 2;               //odwołanie do pola statycznego przez ::
```

```
TPunkt p1;
p1.a = 2;                    //odwołanie do pola statycznego przez . (kropkę)
```

```
TPunkt *wsk;
wsk = new TPunkt;
wsk->a=10;                   //odwołanie do pola statycznego przez ->
delete wsk;
```

Dwa ostatnie sposoby (operator `.` lub `->`) są takie same jak odwoływanie do zwykłych pól niestatycznych.

### **Zmienna statyczna w bloku klasy.**

Zmienna statyczna zadeklarowana w bloku klasy jest jedyną wspólną instancją dla wszystkich obiektów danej klasy. Jest tworzona już na początku działania programu.

### **Przykład**

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

class MojaKlasa
{
public:
    static int x;
    MojaKlasa() { x++; } // konstruktor - zwiększa "x"
    ~MojaKlasa() { x--; } // destruktor - zmniejsza "x"
};

int MojaKlasa::x = 0; // definicja i zainicjowanie zmiennej statycznej "x";

int main()
{
    MojaKlasa obiekt1;
    cout << obiekt1.x << endl; //wyświetli 1

    MojaKlasa obiekt2;
    cout << obiekt2.x << endl; //wyświetli 2

    cout << MojaKlasa::x << endl; // nie jest błędem: wyświetli 2

    return 0;
}
```

### Opis programu

#### a) Jak wykonać licznik istniejących obiektów w programie?

Widzimy zadeklarowaną zmienną statyczną w klasie MojaKlasa. Konstruktor w klasie zwiększa jej wartość, natomiast destruktor zmniejsza. Dzięki temu uzyskaliśmy licznik obiektów klasy MojaKlasa (pominięta została kwestia współbieżności).

#### b) Jak korzystać ze zmiennych statycznych?

Każda zmienna statyczna w klasie jest jedynie deklarowana. Z powodu tego, że posiada ona jedną instancję w programie, należy podać również jej definicję, którą realizuje się na zewnątrz deklaracji klasy w zakresie globalnym (należy pamiętać, że nie w każdym języku programowania jest to konieczne). W tym miejscu również następuje zainicjowanie zmiennej. W funkcji main() tworzymy obiekt klasy MojaKlasa. Wyświetlamy aktualną wartość zmiennej statycznej i widzimy wartość 1. Następnie tworzymy drugi obiekt i widzimy, że wartość zmiennej statycznej wynosi już dwa. Zwróćmy uwagę, że za drugim razem odwołujemy się do zmiennej statycznej korzystając z obiektu obiekt2, dzięki temu widzimy, że wszystkie obiekty (w przykładzie są ich 2) współdzielą zmienną statyczną. W kolejnym kroku przedstawiona została technika odwołania się do zmiennej statycznej nie poprzez obiekt, a poprzez nazwę klasy. Jest to spowodowane tym, że zmienne statyczne w klasie istnieją nawet gdy nie został stworzony żaden obiekt danej klasy. Różnica w C++ polega

jedynie na **użyciu operatora zakresu "::" zamiast "."** (w innych językach programowania jak Java czy C# do zmiennych statycznych w klasie odwołujemy się poprzez ".").

Zmienne statyczne w bloku klasy nazywane są często:

- polami statycznymi
- zmiennymi klasowymi.

### Przeciążenie operatora

Przeciążenie operatorów to cecha języka C++, która pozwala definiować operatory współpracujące ze zdefiniowanymi przez siebie typami. Czyli możesz definiować jak działać będzie operator np. „+” czy „==” ale na nowych typach danych zdefiniowanych przez siebie.

Class budynek

```
{
    Public:
        budynek operator*(fasada dach)
        {
            //ciało nowego operatora „*”
        }
};
```

Wy tłumaczenie;

Dla obiektów klasy budynek został zdefiniowany nowy operator „\*”.

### Zagnieżdżenie klas

Możliwe jest definiowanie klas wewnątrz definicji klasy. Takie klasy, których definicja jest zanurzona w definicji innej klasy, nazywamy klasami **zagnieżdżonymi lub klasami wewnętrznymi**. Klasę, we wnętrzu której podana jest definicja klasy zagnieżdżonej, nazywamy **klasą otaczającą**. Zagnieżdżenie definicji oznacza, że nazwa tej klasy leży w zakresie klasy otaczającej, a nie w zakresie globalnym. W zasadzie nie znaczy nic więcej. Klasy otaczająca i zagnieżdżona nie są specjalnie ze sobą związane.

Class lampa

```
{
    Private:
        // składnik klasy lampa
    Public:
        // składowe klasy lampa
    Class zarowka
    {
        Private:
            // składnik klasy zarowka
        Public:
            // składowe klasy zarowka
    };
};
```

### Dziedziczenie

W przypadku tworzenia nowej klasy nie musisz wszystkiego budować od początku. Jeżeli istnieje obiekt podobny do tego, który zamierzasz stworzyć, to możesz:

- dziedziczyć z danego obiektu
- dodać kilka własnych właściwości,
- zmienić już istniejące.

Dziedziczeniu podlegają metody oraz pola klasy podstawowej(nadrzędnej). Jeśli klasa pochodna zawiera już te same elementy, to powtarzające się elementy klasy podstawowej zostaną pominięte na rzecz takich samych, nowo utworzonych elementów klasy pochodnej.

Co umożliwia dziedziczenie:

- wprowadzenie hierarchii klas,
- wprowadzenie klasyfikacji obiektów,
- zmniejszenie kodu programu.

Na bazowym (podstawowym) poziomie hierarchii mamy bardzo ogólne definicje, na kolejnych poziomach, znajdują się natomiast coraz precyzyjniejsze sformułowania cech obiektów.

### Klasy bazowe i pochodne

W podanym przykładzie klasa B dziedziczy z klasy A. Klasę A nazywamy **klasą bazową** (rodzic), a klasę B **klasą pochodną** (dzieckiem, klasa potomna).

Przykład

```
class A
{
    private:
    int a;

    protected:
    int b;

    public:
    int c;

};

class B: private A
{
    protected:
    A::b;

    public:
    A::c;

};
```

Klasa B dziedziczy prywatnie z klasy bazowej A zmienne chronione b i c. Dla klasy B te zmienne stają się prywatnymi.

Słowa kluczowe public, protected i private, które oznaczają sposób dziedziczenia uprawnień przed klasami bazowymi są opcjonalne.

### Dziedziczenie wielokrotne

Klasa **potomna** może dziedziczyć z kilku klas. Uzyskuje się w ten sposób wielokrotne dziedziczenie.

### **Przykład**

```
class A
{
    ...
};

class B: public A                //dziedziczy z A
{
    ...
};

class C: public A, public B      //C dziedziczy z A i B
{
    ...
};

class D: public B, public C      //D dziedziczy z B i C
{
    ...
};
```

### **Uwaga:**

Klasa C otrzyma dwukrotnie klasę A.

Klasa D otrzyma dwukrotnie klasę B oraz trzykrotnie klasę A.

### Ekspansja cech

Przy tworzeniu nowej klas nie musimy stosować dziedziczenia. Istnieje możliwość wykorzystać obiekty już istniejącej klas w nowej klasie i ich wzbogacenie o nowe cechy. W takim przypadku mamy do czynienia z **ekspansją cech**.

### **Poliformizm**

Polimorfizm czyli wielopostaciowość opisuje zdolność kodu C++ do różnych zachowań zależnie od sytuacji w trakcie wykonywania programu.

Poliformizm nie jest cechą charakterystyką obiektów, jest cechą charakterystyką funkcji składowych klasy.

Implementuje się go poprzez architekturę klasy, ale tylko funkcje składowe klasy mogą być poliformiczne, nie zaś cała klasa.

Polimorfizm jest realizowany przez **funkcje wirtualne**.

### **Funkcje virtual**

Funkcje te są zdefiniowane dla poszczególnych klas z danej hierarchii, jako posiadające:

- tą samą nazwę,
- zestaw parametrów,
- realizujące ten sam cel,
- dla każdej klasy mogą różnić się sposobem realizacji.
- określenie, która funkcja będzie wywołana na rzecz pewnego obiektu jest przesunięte do czasu wykonania programu.

Czyli funkcje wirtualne to nic innego jak metody, których definicja może ulec zmianie w podklasach.

### **Wczesne i późne wiązanie**

W C++ używa on zarówno wczesnego, jak i późnego wiązania.

**Proces wczesnego wiązania** polega na tym, że kompilator wywołuje identyfikatory funkcji na podstawie kodu źródłowego. Następnie linker pobiera te identyfikatory i zamienia je na adres fizyczny. W ten sposób identyfikatory funkcji łączone są z adresami fizycznymi przed wykonaniem programu w procesie kompilacji i konsolidacji programu. Problem z wczesnym wiązaniem polega na tym, że programista musi przewidzieć, jakie obiekty będą używane we wszystkich wywołaniach funkcji w każdej sytuacji. Daje to w wyniku dużą szybkość, ale brak elastyczności.

**Proces późnego wiązania** jest bardziej złożony. Kod programu sam musi decydować w czasie swojego wykonania, którą funkcję należy wywołać. Wymusza to, aby kod wykonawczy sortował powiązania identyfikatorów i adresów funkcji. Daje to w wyniku skuteczny język, lecz stosowanie późnego wiązania spowalnia działanie programu.

### Uwagi:

- Do programisty należy decyzja, kiedy należy użyć wczesnego, a kiedy późnego wiązania.
- Późne wiązanie dla funkcji (metod) określa się przez zadeklarowanie jej jako virtual.
- Późne wiązanie ma sens tylko dla obiektów, które są częścią hierarchii klas. Z tego względu deklaracja funkcji wirtualnych w klasach, które nie będą używane jako klasy bazowe, jest składniowo poprawne, ale niepotrzebnie wydłuża czas wykonywania programu.

### **Przykład**

#### Temat:

Polimorfizm oraz użycie funkcji wirtualnych.

```
class A
{
public:
virtual void wyswietl(void) { cout << "Klasa A"; } //pierwsza definicja
};
```

```
class B : public A
{
public:
virtual void wyswietl(void) { cout << "Klasa B"; } //druga definicja
};
```

//Klasa B dziedziczy z klasy A. Obie klasy posiadają własną definicję metody wyswietl().  
//Dodatkowo zdefiniowano funkcję pokaz().



```
void pokaz(A* a)
{
    a->wyswietl();
}
```

//Stworzono również przykładowe obiekty obu klas w funkcji main().

```
void main()
{
    A* a = new A;
    B* b = new B;

    a->wyswietl();//użyta zostanie funkcja A::wyswietl()
    b->wyswietl();    //użyta zostanie funkcja B::wyswietl()

    pokaz(a);    //użyta zostanie funkcja A::wyswietl()
    pokaz(b);    //użyta zostanie funkcja B::wyswietl()
}
```

/\*

Podczas wywołania funkcji wyswietl() program sam decyduje, której funkcji należy użyć. Poliformiczne zachowanie funkcji wyswietl() w klasach A i B nie jest oczywiste. Takie zachowanie uwidacznia się w przypadku funkcji pokaz(), w której trudniej jest przewidzieć, czy zostanie wywołana funkcja A::wyswietl(), czy też B::wyswietl(). W przypadku usunięcia obu słów kluczowych virtual z klasy A i B po wywołaniu funkcji pokaz() w obu przypadkach zostanie użyta funkcja A::wyswietl().

\*/

```
A* a = new A;
B* b = new B;

pokaz(a);    //użyta zostanie funkcja A::wyswietl()
pokaz(b);    //użyta zostanie funkcja A::wyswietl()
```

/\*

Drugie wywołanie funkcji spowoduje wywołanie A::wyswietl(), ponieważ argument B\* zostanie skonwertowany na A\*, a następnie przekazany do funkcji pokaz(A\*). Wywołanie a->wyswietl() ma wczesne wiązanie z funkcją A::wyswietl(), co daje jej stałe zachowanie. Deklaracja funkcji jako virtual nie oznacza, że jest ona nakładana w klasie pochodnej, ale może tak być.

Jest możliwe zdefiniowanie metod wirtualnych, których definicja może ulec zmianie w podklasach. Metoda wirtualna jest więc zdefiniowana w klasie początkowej, a później przeddefiniowywana w jej podklasach. Słowo kluczowe virtual nie musi być konieczne używane w danej klasie pochodnej, ale użycie jego jest zalecane, dla czystości i przejrzystości kodu źródłowego programu.

\*/

### **Przykład 9**

Temat: Definiowanie klasy Tnapis z dwoma metodami. Jedna metoda zapisana w całości ciele klasy, druga metoda na zewnątrz ciała klasy. Utworzony jest obiekt oraz wywołane metody.

Wykonaj:

Zapisz w zeszycie, przepisz treść podpunktów i pod nimi odpowiedzi:

- a) nazwę klasy,
- b) ile jest metod w tym zadaniu,
- c) W jakim miejscu w stosunku do treści klasy definiowane są metody,
- d) jak nazywa się utworzony obiekt,
- e) zapisz linijkę która tworzy obiekt,
- f) zapisz linijkę, która wywołuje metodę,
- g) zapisz w zeszycie napis, który pokazał się na ekranie w wyniku działania programu.
- h) utwórz nowy obiekt **napis\_zew** i wywołaj metodę dla tego obiektu **pisz\_nazewnatrz()**, program wykonaj tak aby efekty wywołania metody *pisz\_wewnatrz()* oraz metody *pisz\_nazewnatrz()* wykonały się w osobnych wierszach.

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Tnapis
```

```
{  
    public: // początek metod publicznych  
    void pisz_wewnatrz()  
    {  
        cout << "Metoda zadeklarowana wewnatrz klasy";  
    }  
    void pisz_nazewnatrz();  
};
```

```
void Tnapis::pisz_nazewnatrz()  
{  
    cout << "Metoda zadeklarowana na zewnatrz klasy";  
};
```

```
int main (int argc, char *argv[])
```

```
{  
    Tnapis napis_wew; // deklarowanie obiektu napis_wew  
    // wywołanie metody pisz_wewnatrz() dla obiektu napis_wew  
    // zwróć uwagę na kropkę w linii poniżej  
    napis_wew.pisz_wewnatrz();  
    cout<<endl;  
    system("PAUSE");  
    return EXIT_SUCCESS;  
}
```

## ZADANIE 12

- a) Utwórz klasę Tnazwisko\_ucznia (bez polskich liter).
- b) Utwórz dwie metody

-pierwsza metoda → nazwisko\_ucznia\_definicja\_wewnatrz, metoda ta wyświetla nazwisko ucznia. Cała metoda zdefiniowana w ciele klasy.

-druga metoda → nazwisko\_ucznia\_definicja\_zewnatrz, metoda ta wyświetla imię ucznia. Metoda zdefiniowana poza ciałem klasy.

c) obiekt napis\_nazwisko\_ucznia.

### **Przykład 10**

Użycie obiektów do obliczenia pola i obwodu kwadratu.

Pisz przykład zmień w nim:

- Nazwa klasy na Tnazwisko\_ucznia
- metoda obliczająca obwód to nazwisko\_ucznia\_obwod
- metoda obliczająca pole to nazwisko\_ucznia\_pole
- Obiekty to nazwisko\_ucznia\_kwadrat1 oraz nazwisko\_ucznia\_kwadrat2

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Tkwadrat
```

```
{
```

```
    public: // początek metod publicznych
```

```
    void pole(int a)
```

```
    {
```

```
        cout << "P=" << a*a << " cm^2" << endl;
```

```
    }
```

```
    void obwod(int a)
```

```
    {
```

```
        cout << "OBW=" << 4*a << " cm" << endl;
```

```
    }
```

```
};
```

```
int main (int argc, char *argv[])
```

```
{
```

```
    Tkwadrat kw1, kw2; // deklarowanie obiektów kw1 kw2
```

```
    // wywołanie metody pole i obwod dla obiektu kw1 kw2
```

```
    // zwróć uwagę na kropkę w liniach poniżej
```

```
    cout << "Obliczenia dla kwadratu 1" << endl;
```

```
    kw1.pole(3);
```

```
    kw1.obwod(3);
```

```
    cout << "Obliczenia dla kwadratu 2" << endl;
```

```
    kw2.pole(4);
```

```
    kw2.obwod(4);
```

```
    cout << endl;
```

```
    system("PAUSE");
```

```
    return EXIT_SUCCESS;
```

}

### ZADANIE 13

Z użyciem programowania obiektowego Nazwa klasy na Tnazwisko\_ucznia

- metoda obliczająca obwód(objętość) to nazwisko\_ucznia\_obwod/objetosc
- metoda obliczająca pole to nazwisko\_ucznia\_pole
- Obiekty to nazwisko\_ucznia..... oraz nazwisko\_ucznia.....

oblicz: (wybierasz numer zadania takie jakie masz w dzienniku(grupie))

Uwaga: gdy deklarujesz stałą PI to deklaracja tej stałej musi być umieszczona wewnątrz metody.

1. pole oraz obwód koła po podaniu promienia.

$$R=10 \quad P=314,15 \text{ cm}^2 \quad \text{Obw}=62,8 \text{ cm}$$

2. pole oraz obwód trójkąta równobocznego po podaniu boku a.

$$a=3 \quad P= \text{ cm}^2 \quad \text{Obw}=6 \text{ cm}$$

3. pole oraz obwód rombu po podaniu przekątnych,

$$d1=12 \quad d2=16 \quad P=96 \text{ cm}^2 \quad \text{Obw}= 40 \text{ cm}$$

4. pole oraz obwód prostokąta po podaniu boków,

5. pole oraz obwód trójkąta po podaniu dwóch boków oraz kąta między nimi podany w stopniach,

6. pole oraz obwód równoległoboku po podaniu dwóch boków oraz kąta między nimi podany w stopniach,

$$a=5 \quad b=4 \quad \alpha= 30 \text{ st} \quad \text{Obw}= 18 \text{ cm} \quad P= 10 \text{ cm}^2$$

7. pole oraz obwód kwadratu po podaniu jego przekątnej

8. pole oraz objętość walca po podaniu wysokości oraz promienia podstawy

$$r=10, \quad h=15, \quad V=4710,39 \text{ cm}^3 \quad PC=1570,79 \text{ cm}^2$$

9. pole oraz objętość stożka po podaniu wysokości oraz promienia podstawy

$$r=10, \quad h=15, \quad V= \quad PC=$$

10. pole oraz objętość ostrosłupa prawidłowego czworokątnego po podaniu wysokości i krawędzi podstawy

$$h=8, \quad a=12, \quad V= \quad Pc=384 \text{ cm}^2, \quad V=384 \text{ cm}^3$$

$$a=12 \quad h=8 \quad V=384 \text{ cm}^3 \quad Pc=384 \text{ cm}^2$$

11. pole oraz objętość prostopadłościanu po podaniu jego trzech wymiarów.

12. pole oraz objętość prostopadłościanu prawidłowego po podaniu wysokości oraz krawędzi podstawy

13. pole oraz objętość prostopadłościanu prawidłowego sześciokątnego po podaniu krawędzi podstawy oraz krawędzi bocznej.

14. pole oraz objętość czworościanu foremnego po podaniu krawędzi bocznej  
 $a=15$   $P_c=339 \text{ cm}^2$   $V=323 \text{ cm}^3$
15. pędu i energii kinetycznej po podaniu prędkości i masy ciał
16. oporu zastępczego po połączeniu dwóch oporników w połączeniu szeregowym i równoległych.

### **Przykład 11**

Temat: Demonstracja definiowania: klasy, konstruktora, destruktor.

Wykonaj:

- 1)Zapisz w zeszycie temat.
- 2)Uruchom przykład.
- 3)Zmień (dopisz) tak program, aby stworzyć człowieka o Twoim nazwisku (bez polskich liter) o wadze, jakiej masz i wieku. Spowoduj starzenie wirtualnej postaci prezentując Ciebie oraz Twoje przytycie o 10%.
- 4)Przepisz przykład łącznie z komentarzami.
- 5)Zapisz pod przykładem
  - a)jaką nazwę ma konstruktor→ .....
  - b)jaką nazwę ma destruktor→ .....
  - c)linijkę wywołującą metodę starzenia→ .....
  - d)linijkę wywołującą metodę tycia→ .....
  - e)linijka powoływania wirtualnego człowieka→ .....

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Tczlowiek //poczatek klasy
```

```
{
```

```
private:
```

```
float wiek;
```

```
float waga;
```

```
public:
```

```
Tczlowiek() //definicja konstruktora1 łącznie z jego działaniem
```

```
// dane statystycznego człowieka po urodzeniu
```

```
{
```

```
wiek=0;
```

```
waga=3.5;
```

```
}
```

```
Tczlowiek(float wi, float wa);
```

```
//definicja konstruktora2 bez definicji →definicja później
```

```
~Tczlowiek() // destruktor
```

```
{
```

```
cout<<"człowiek umiera"<<endl;
```

```
}
```

```
void podaj_dane()
```

```
{
```

```

        cout<<"wiek="<<wiek<<" lat"<<endl;
        cout<<"waga="<<waga<<" kg"<<endl;
    }
    void tycie(float procent)
    {
        waga=waga+waga*procent/100;
    }
    void starzenie_rok()
    {
        wiek++;
    }
}; // koniec klasy

//definicja konstruktora2 służy do zmiany pól klasy
Tczlowiek::Tczlowiek(float wi, float wa)
{
    wiek=wi;
    waga=wa;
}

int main(int argc, char *argv[])
{
    Tczlowiek Kowalski; // narodziny człowieka Kowalski o danych statystycznych
        //utworzenie obiektu
    cout<<"Kowalski się urodził i ma parametry"<<endl;
    Kowalski.podaj_dane();
    Tczlowiek Malinowski(0,4.5); // narodziny człowieka Malinowski o wadze 4.5 kg
        //utworzenie obiektu o zadanych parametrach
    cout<<"Malinowski się urodził i ma parametry"<<endl;
    Malinowski.podaj_dane();
    Malinowski.tycie(20);
    cout<<"Malinowski przytył o 20 %i ma parametry"<<endl;
    Malinowski.podaj_dane();
    cout<<"Malinowski postarzał się o rok i ma parametry"<<endl;
    Malinowski.starzenie_rok();
    Malinowski.podaj_dane();
    Kowalski.~Tczlowiek(); // człowiek umiera
    Malinowski.~Tczlowiek(); // człowiek umiera
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

## ZADANIE 14

Uwaga:

Przy rozwiązywaniu tego zadania stosuj komentarze jak w przykładzie poprzednim. Przy umieszczaniu listingu komentarze też muszą być w zeszycie.

Stwórz:

**klasa**

klasę Tsamochod\_dwie\_litery\_nazwiska np. Tsamochod\_ko

klasa powinna mieć prywatne pola

rok\_produkcji\_dwie\_litery\_nazwiska np. rok\_produkcji\_ko

kolor\_dwie\_litery\_nazwiska np. kolor\_ko → typu string

aby zapisać wartość do pola to:

nazwa\_obiektu.nazwa\_pola="wartość\_pola"

np. samochod.kolor\_ko="czerwony"; // gdy pole jest typu string

np. samochod.rok\_produkcji\_ko=1; // gdy pole jest typu int

predkosc\_dwie\_litery\_nazwiska predkosc\_ko

ilosc\_paliwa\_dwie\_litery\_nazwiska np. ilosc\_paliwa\_ko

### **konstruktor 1**

określający stan początkowy samochodu

rok\_produkcji\_ko=1990+numer\_w\_dzienniku

kolor\_ko=czerwony (pamiętaj, że jest to pole typu znakowego[string], czyli w cudzysłowie)

predkosc\_ko=50+numer\_w\_dzienniku

ilosc\_paliwa\_ko=10+numer\_w\_dzienniku [w litrach]

### **konstruktor 2** (poza ciałem klasy)

dający możliwość powołania obiektu o zadanych parametrach:

rok\_produkcji\_ko

kolor\_ko

predkosc\_ko

ilosc\_paliwa\_ko

**metody:** (będę sprawdzał czy jest sześć metod)

wypisująca stan obiektu → nazwę zaproponuj sam z dwiema literami

starzenie się samochodu

zmianę koloru samochodu

zmianę prędkości samochodu

ilość paliwa w baku w wyniku jazdy samochodem

tankowanie paliwa o określonej ilości paliwa

### **powołania obiektów**

powołaj **trzy** samochody do życia:

samochód pierwszy → z parametrami statystycznym (czyli zdefiniowanymi w konstruktorze1)

samochód drugi, samochód trzeci → z odpowiednimi parametrami zaproponowanymi przez Ciebie (czyli użycie konstruktora2 z parametrami)

Po każdym powołaniu samochodu do życia wyświetl na ekranie jakie parametry ma samochód.

### **Operacje**

Wykonaj 10 operacji na obiektach np. tankowanie, zmiana prędkości, zmiana parametrów powinna być opisywana przez odpowiednie komunikaty na ekranie

### **Kasowanie obiektów**

Dokonaj kasowania trzech samochodów i poinformuj o tym. (Czyli musisz zdefiniować destruktory w każdej klasie)

### **Przykład 12**

Temat: Demonstracja definiowania dwóch klas: motocykl oraz samochod. Demonstracja hermetyzacji danych.

Wykonaj:

- 1)Przepisz temat
- 2)Uruchom przykład. Przeczytaj komunikaty ze zrozumieniem na ekranie CMD.
- 3)a)Zmień sekcję w przykładzie tak, aby **nie** można było zmienić ilości kół motocykla (jaką sekcję zmienić patrz opis o Hermetyzacji).
- b)Zapisz w zeszycie jak zareagował kompilator na zmianę sekcji, czyli zapisz w zeszycie, jakie błędy wykazał kompilator (po angielsku). Przepisz tylko kolumnę Komunikat (około 5 wierszy).
- c)Dokonaj kasowania tych linii programu, które żądają dostępu do pola **kola**. Program będzie się teraz kompilował. Zapisz w zeszycie skasowane linijki.

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
class motocykl //poczatek klasy
```

```
{  
    public:  
    int kola;
```

```
public:
```

```
    motocykl() //definicja konstruktora łącznie z jego działaniem  
                // motocykl tradycyny o 2 kołach
```

```
    {  
        kola=2;  
    }
```

```
    motocykl(int kola1) //definicja konstruktora przeciązonego  
                // gdy chcesz utworzyc motocykl o wiekszej liczbie kol
```

```
    {  
        kola=kola1;  
    }
```

```
    ~motocykl() // destruktor
```

```
    {  
        cout<<"motocykl do kasacji"<<endl;  
    }
```

```
    void podaj_dane()
```

```
    {  
        cout<<"ilosc kol="<<kola<<endl;  
    }
```

```
    void zmiana_ilosc_kol(int kola1)
```

```
    {  
        kola=kola1;
```



```

    }
}; // koniec klasy

class samochod //poczatek klasy
{
private:
    string nadwozie;

public:
    samochod() //definicja konstruktora łącznie z jego działaniem
                // o typowym nadwoziu sedan, inne: coupe, kombi, hatchback
    {
        nadwozie="sedan";
    }
    samochod(string nadwozie1) //definicja konstruktora gdy chcesz
                                // wyprodukowac samochod o innym nadwoziu
    {
        nadwozie=nadwozie1;
    }
    ~samochod() // destruktork
    {
        cout<<"samochod do kasacji"<<endl;
    }
    void podaj_dane()
    {
        cout<<"rodzaj nadwozia="<<nadwozie<<endl;
    }
}; // koniec klasy

```

```

int main(int argc, char *argv[])
{
    motocykl honda; // wyprodukowaliśmy motocykl honda i ma on dwa kola
                    //utworzenie obiektu wraz z uruchomieniem konstruktora
                    //bezparametrowego, taki konstruktor daje 2 kola
    cout<<"wyprodukowano motor honda"<<endl;
    honda.podaj_dane();
    honda.kola=5; // teraz honda bedzie miała 5 kol
                // gdy pole kola jest public
    cout<<"podaj ile kol ma motor honda=";
    cin>>honda.kola; //mozesz wczytac ilosc kol do pola kola
                    // gdy pole kola jest public
    cout<<"zmieniono ilosc kolo dla honda"<<endl;
    honda.podaj_dane();
    honda.zmiana_ilosc_kol(2);
                //tak uzyskujesz dostep do zmiany pola kola gdy
                // ono private w klasie musi byc metoda "zmiana_ilosc_kol"
    motocykl kawasaki(3); // wyprodukowaliśmy motocykl kawasaki i ma on trzy kola
                    //wywołano przeciążony konstruktor z parametrem 3
                    //utworzenie obiektu

```

```

cout<<"wyprodukowano motor kawasaki"<<endl;
kawasaki.podaj_dane();
samochod fiat; // wyprodukowaliśmy samochod fiat i ma on nadwozie sedan
cout<<"wyprodukowano samochod fiat"<<endl;
fiat.podaj_dane();
samochod kia("kombi"); // wyprodukowaliśmy samochod kia i ma on nadwozie kombi
cout<<"wyprodukowano samochod kia"<<endl;
kia.podaj_dane();
system("PAUSE");
return EXIT_SUCCESS;
}

```

### **Przykład 13**

Temat: Demonstracja dziedziczenia.

Wykonaj:

- 1)Przepisz temat
- 2)Uruchom przykład.
- 3)Wykonaj oraz uzupełnij w zeszycie tabelę

Nazwa klasy pierwszej→	
Nazwa klasy drugiej→	
Nazwa pola dla klasy 1→	
Nazwa konstruktora dla klasy 1→	
Nazwa konstruktora przeciążonego dla klasy 1→	
Nazwa destruktora dla klasy 1→	
Nazwy metod dla klasy 1 (dwie) →	
Nazwa pola dla klasy 2→	
Nazwa konstruktora dla klasy 2→	
Nazwa konstruktora przeciążonego dla klasy 2→	
Nazwa destruktora dla klasy 2→	
Nazwy metod dla klasy 2 (dwie) →	
Linia programu, która powołała klasę pojazd na podstawie klas 1 i 2 ↓	
Nazwa konstruktora dla klasy pojazd→	
Nazwa konstruktora przeciążonego dla klasy pojazd→	
Nazwa destruktora dla klasy pojazd→	
Nazwy metod dla klasy pojazd (jedna) →	
Nazwa pola dla klasy pojazd→	
Powołanie obiektu traktor→	
Wywołanie dowolnej metody dla obiektu traktor→	

```

#include <cstdlib>
#include <iostream>

using namespace std;

class motocykl //poczatek klasy
{
private:
    int kola;

public:
    motocykl() //definicja konstruktora łącznie z jego działaniem
                // motocykl tradycyjny o 2 kolach
    {
        kola=2;
    }
    motocykl(int kola1) //definicja konstruktora przeciązonego
                // gdy chcesz utworzyć motocykl o większej liczbie kol
    {
        kola=kola1;
    }
    ~motocykl() // destruktork
    {
        cout<<"motocykl do kasacji"<<endl;
    }
    void podaj_dane_m()
    {
        cout<<"ilosc kol="<<kola<<endl;
    }
    void zmiana_ilosc_kol(int kola1)
    {
        kola=kola1;
    }
}; // koniec klasy

class samochod //poczatek klasy
{
private:
    string nadwozie;

public:
    samochod() //definicja konstruktora łącznie z jego działaniem
                // o typowym nadwoziu sedan, inne: coupe, kombi, hatchback
    {
        nadwozie="sedan";
    }
    samochod(string nadwozie1) //definicja konstruktora gdy chcesz
                // wyprodukować samochod o innym nadwoziu
    {
        nadwozie=nadwozie1;
    }
};

```

```

    }
    ~samochod() // destruktor
    {
        cout<<"samochod do kasacji"<<endl;
    }
    void karoseria(string nadwozie1) //definicja metody zmiany karoseri
    {
        nadwozie=nadwozie1;
    }
    void podaj_dane_s()
    {
        cout<<"rodzaj nadwozia="<<nadwozie<<endl;
    }
}

```

```

}; // koniec klasy

```

```

class pojazd: public motocykl, public samochod
{
    private:
        int rocznik;
        string kolor;
    public:
        pojazd(string kolor1,int rocznik1) //definicja konstruktora gdy chcesz
            // wyprodukowac samochod o innym nadwoziu
        {
            rocznik=rocznik1;
            kolor=kolor1;
        }

        ~pojazd() // destruktor
        {
            cout<<"pojazd do kasacji"<<endl;
        }

        void podaj_dane_p()
        {
            cout<<"kolor="<<kolor<<endl;
            cout<<"rocznik="<<rocznik<<endl;
        }
};

```

```

int main(int argc, char *argv[])
{
    motocykl honda; // wyprodukowaliśmy motocykl honda i ma on dwa kola
        //utworzenie obiektu wraz z uruchomieniem konstruktora
        //bezparametrowego, taki konstruktor daje 2 kola
    cout<<"wyprodukowano motor honda"<<endl;
    honda.podaj_dane_m();
    motocykl kawasaki(3); // wyprodukowaliśmy motocykl kawasaki i ma on trzy kola
}

```

```

        //wywołano przeciążony konstruktor z parametrem 3
        //utworzenie obiektu
cout<<"wyprodukowano motor kawasaki"<<endl;
kawasaki.podaj_dane_m();
samochod fiat; // wyprodukowaliśmy samochód fiat i ma on nadwozie sedan
cout<<"wyprodukowano samochód fiat"<<endl;
fiat.podaj_dane_s();
samochod kia("kombi"); // wyprodukowaliśmy samochód kia i ma on nadwozie kombi
cout<<"wyprodukowano samochód kia"<<endl;
kia.podaj_dane_s();
pojazd traktor("czerwony",2010);          // powołanie obiektu traktor
                                           // o kolorze czerwonym i roczniku 2010
traktor.zmiana_ilosc_kol(4);                //traktor ma 4 koła
traktor.karoseria("kombi");                // karoseria typu kombi
cout<<"traktor ma dane"<<endl;
traktor.podaj_dane_m(); //
traktor.podaj_dane_s();
traktor.podaj_dane_p();
system("PAUSE");
return EXIT_SUCCESS;
}

```

## Zadanie 15

Uwaga:

Przy rozwiązywaniu tego zadania stosuj komentarze jak w przykładzie poprzednim. Przy umieszczaniu listingu komentarze też muszą być w zeszycie.

Stwórz:

**klasę**

klasę Tzwierz\_dwie\_litery\_nazwiska np. Tzwierz\_ko

klasa powinna mieć prywatne pola

wiek\_dwie\_litery\_nazwiska np. wiek\_ko

waga\_dwie\_litery\_nazwiska np. waga\_ko

### konstruktor 1

określający stan początkowy zwierza

wiek\_ko=10+numer\_w\_dzienniku

waga\_ko=50+numer\_w\_dzienniku

### konstruktor 2

dający możliwość powołania obiektu o zadanych parametrach:

wiek\_ko=10+numer\_w\_dzienniku

waga\_ko=50+numer\_w\_dzienniku

**metody:** (będę sprawdzał czy jest sześć metod)

wypisująca stan obiektu → nazwę zaproponuj sam z dwiema literami

starzenie się zwierza

## przybierania na wadze

**klasy potomne:**

stwórz klasę Tssak\_dwie\_litery na podstawie klasy zwierzak

pole:

siersc\_dwie\_litery

np. biały, rudy itp

metoda

warcz\_dwie\_litery → wynik działania tej metody to napis na ekranie jaki dźwięk wyda obiekt

stwórz klasę Tptak\_dwie\_litery na podstawie klasy zwierzak

pole:

upierzenie\_dwie\_litery

np. lotki, puch itp

metoda

lataj\_dwie\_litery →wynik działania tej metody to napis na ekranie właśnie latam

## powołania obiektów

powołaj **dwa** zwierzaki do życia (owady):

zwierzak pierwszy → z parametrami statystycznym (czyli zdefiniowanymi w konstruktorze1)

zwierzak drugi → z odpowiednimi parametrami zaproponowanymi przez Ciebie (czyli użycie konstruktora2 z parametrami)

Po każdym powołaniu zwierzaka do życia wyświetl na ekranie jakie parametry ma zwierzak.

powołaj **ssaka** do życia:

ssak → z parametrami statystycznym (czyli zdefiniowanymi w konstruktorze1)

ptak → z odpowiednimi parametrami zaproponowanymi przez Ciebie (czyli użycie konstruktora2 z parametrami)

Po każdym powołaniu zwierzaka do życia wyświetl na ekranie jakie parametry ma zwierzak.

## Operacje

Wykonaj kilka operacji na obiektach opisywana przez odpowiednie komunikaty na ekranie

## Kasowanie obiektów

Dokonaj kasowania wszystkich obiektów i poinformuj o tym. (Czyli musisz zdefiniować destruktory w każdej klasie)

## Część 4

### While, Do while, repeat, IF, Continue oraz Break

**Zadanie** (praca domowa)

**Pytanie 1**

Opisz konstrukcję pętli while, jaka to pętla, narysuj schemat, przepisz program jako przykład.

**Pytanie 2**

Opisz instrukcję kbhit()

**Pytanie 3**

Opisz konstrukcję pętli do while, jaka to pętla, narysuj schemat, przepisz program jako przykład.

**Pytanie 4**

Opisz konstrukcję instrukcji warunkowa if z własnym przykładem innym niż w instrukcji, narysuj schemat.

**Pytanie 5**

Opisz konstrukcję instrukcji warunkowa if...else z własnym przykładem innym niż w instrukcji, narysuj schemat.

**Pytanie 6**

Na podstawie przykładu zapisz co oznacza zapis `SUMA+=(++n);`

**Pytanie 7**

Napisz przykład zagnieżdżonej instrukcji if z własnym przykładem innym niż w instrukcji.

**Pytanie 8**

Opisz instrukcję Continue oraz Break

## Pętle → definicja i rodzaje

Pętla jest to struktura informatyczna, która wykonuje powtarzanie pewnych operacji w programie np. wyświetlaj napis „Programowanie jest piękne” 10 razy.

Rodzaje pętli:

- a) pętla **while**
- b) pętla **do while**
- c) pętla **for**

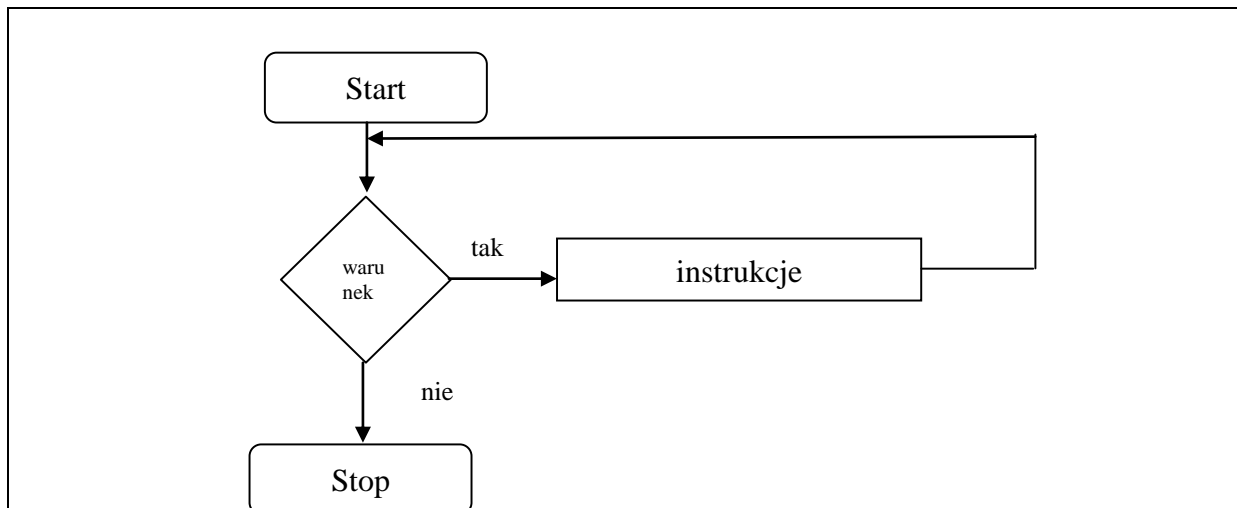
### PĘTLA while.

Pętlę typu **while** jest pętlą z **kontrolowanym wejściem** tzn. najpierw jest obliczany warunek a po jego spełnieniu wchodzimy do pętli i wykonujemy instrukcje z niej.

Konstrukcja pętli while wygląda następująco:

```
while (Wyrażenie_logiczne)
{
    Instrukcja1;
    .....
    InstrukcjaN;
}
```

Jeśli Wyrażenie\_logiczne ma wartość logiczną zera, to nie zostaną wykonane Instrukcje czyli nie nastąpi wejście do pętli..





### **Przykład 14**

Temat: Wykorzystanie pętli while do programu piszącego wykrzykniki aż do wciśnięcia dowolnego klawisza.

Funkcja kbhit() oznacza wciśnięcie dowolnego klawisza.

Wykonaj:

- 1)Przepisz temat.
- 2)Zapisz teorię+schemat blokowy dotyczącą pętli while.
- 3)Zapisz, co oznacza funkcja kbhit() , !kbhit(), printf("\n"),
- 4)Uruchom przykład,
- 5)Zapisz przykład do zeszytu (pamiętaj o wcięciach).

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <conio.h>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    while(!kbhit())
    {
        printf("!");
    }
    printf("\n");
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

**Zadanie 15a**

Uzupełnij tabelkę operacji obliczeniowych.

Zapis tradycyjny	Zapis skrócony pierwszy	Zapis skrócony drugi
zm = zm + zm1;	zm+=zm1;	
?	?	p++;
?	zmie+= 5;	
?	ujka - = 1;	?
jj = jj % 2;	?	
?	u*=10;	
hitrus= hitrus/4;	?	

**Inkrementacja**

Operator inkrementacji (++), czyli zwiększenie wartości zmiennej o stałą wartość najczęściej o jeden (1). Np.

i=i+5;

i=i+1; lub w formie skróconej i++;

**Dekrementacja**

Operator dekrementacji (--), czyli zmniejszenie wartości zmiennej o stałą wartość najczęściej o jeden (1). Np.

x=x+5;

x=x-1; lub w formie skróconej x--;

**Formy zapisu:**

**Przedrostkowa** np. (++i --x) najpierw zmienna jest zwiększana o jeden, a następnie ta zwiększona wartość jest brana do obliczeń.

**Przyrostkowa** (końcówkowa) (i++ x--) najpierw brana jest stara wartość zmiennej do obliczeń a dopiero później jest ona zwiększana o jeden.

### Przykład 15

Temat: Stosujemy pętlę while w programie obliczającym sumę kolejnych liczb naturalnych dopóki SUMA nie przekroczy 1000.

Wykonaj:

- 1)Przepisz temat,
- 2)Zapisz, co oznacza instrukcja SUMA+=(++n);,
- 3)Uruchom przykład,
- 4)Zapisz w zeszycie daną wyjścia dla programu,
- 5)Zapisz przykład do zeszytu (pamiętaj o wcięciach).
- 6)Wykonaj schematy blokowy.

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    float SUMA=0, n=0;
    while (SUMA<1000)
    {
        SUMA+=(++n);          // SUMA = SUMA + n;
                              // ++n zwiększaj zmienna a o jeden
                              //w formie przedrostkowej
                              // inny zapis ++n to    n=n+1;
    }
    printf("SUMA: %.1f ostatnia liczba: %.2f", SUMA, n);
    cout<<"\n"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### Uwaga:

Podczas procesu **sumowa** w dolnym języku programowania zmienna sumująca musi być zerowana przed przystąpieniem do czynności sumowania np. **suma=0;** .

Podczas procesu **mnożenia** w dolnym języku programowania zmienna w którym zapisujemy aktualny iloczyn musi mieć wartość 1 (jeden) przed przystąpieniem do czynności obliczania iloczynu np. **iloczyn=1;** .

### **Zadanie 15b**

Temat:

Oblicz z użyciem pętli while **iloczyn** liczb:

Od  $\rightarrow [(liczba\_liter\_imienia) \bmod 4] + 1$

Do  $\rightarrow [(liczba\_liter\_imienia) \bmod 4] + 6$

Dokonaj obliczenia i zapisz w zeszycie:

Od  $\rightarrow$  *tutaj wpisz w zeszycie obliczoną liczbę*

Do  $\rightarrow$  *tutaj wpisz w zeszycie obliczoną liczbę*

Uwaga1:

Program może sam obliczać mod lub sam możesz obliczyć w zeszycie wartość **Od** oraz **Do**

Uwaga2:

Zmienna w, której zapisujesz wartość iloczynu to ILOCZYN\_trzy\_pierwsze\_litery\_nazwiska  
np. ILOCZYN\_kow

Wykonaj schematy blokowy.

### **PĘTLA do...while.**

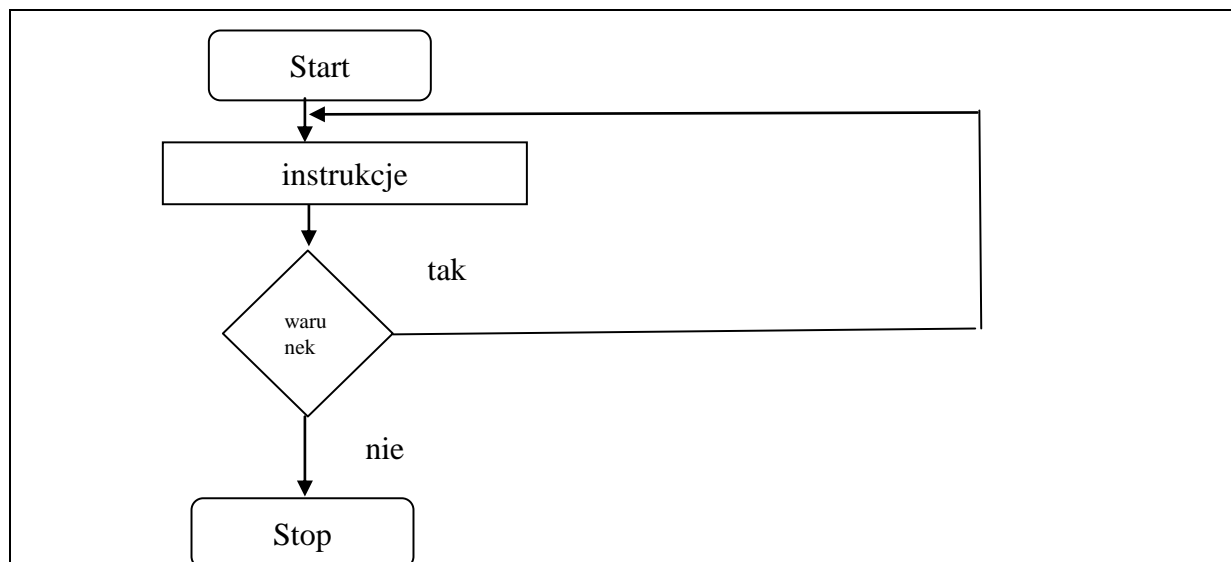
Pętlę typu do while jest pętlą z **kontrolowanym wyjściem** tzn. warunek obliczany po wykonaniu pętli.

Przerwanie pętli powodowane jest przez **NIESPEŁNIENIE WARUNKU**.

Konstrukcja pętli do while wygląda następująco:

```
do
{
    Instrukcja1
    Instrukcja2
    .....
    InstrukcjaN
}
```

while (Wyrażenie\_logiczne);



### Przykład 16

Temat: Program obliczający pole trójkąta ze sprawdzeniem poprawności wczytywania danych z użyciem pętli **do...while**.

Wykonaj:

- 1)Przepisz temat,
- 2)Uruchom przykład,
- 3)Zapisz teorię +schemat blokowy dotyczącą pętli **do .....while**.
- 4)Zapisz w zeszytie, jakich wartości **nie** przyjmuje przykład jako danych wejściowych,
- 5)Zapisz do zeszytu (pamiętaj o wcięciach) tę część programu, która sprawdza poprawność wczytywanej podstawy (pamiętaj o wcięciach).
- 6) zapisz do zeszytu
  - pętla **do...while**→ to pętla z .....
  - pętla **while**→ to pętla z .....
- 7)Wykonaj schemat blokowy.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    float bok_a, wysokosc, pole;
    do
    {
        cout<<"podaj dlugosc podstawy a=";
        cin>> bok_a;
    }
    while (bok_a<=0);
    do
    {
        cout<<"podaj dlugosc wysokosci=";
        cin>> wysokosc;
    }
    while (wysokosc<=0);
    pole = bok_a*wysokosc/2;
    cout<<"pole="<<pole<<" m^2";
    cout<<"\n"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Koniec przykladu 16

## Przykład 16a

Temat: Demonstracja pętli **do...while** do zapewnienia powtarzalności programu.

Wykonaj:

- 1)Przepisz temat,
- 2)Uruchom przykład,
- 3)Zapisz przykład do zeszytu (pamiętaj o wcięciach).
- 4)Dopisz, aby powtarzanie było również na klawisz „T” użyj operatora OR ( sprawdź jak go zapisujemy w CPP) .
- 5)Wykonaj schemat blokowy.

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <conio.h>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    char znak;
    do
    {
        printf("Umiem powtarzać program\n");
        printf("\nczy powtórzyć t(tak)/n(nie) wciśnij klawisz\n");
        cin>>znak;
    }
    while (znak!='t');

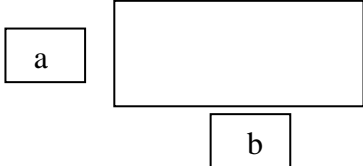
    return EXIT_SUCCESS;
}
```

**Specyfikacja problemu to:**

- 1) dane wejściowe
- 2) dane wyjściowe
- 3) ograniczenia na dane wejściowe
- 4) rysunki i wzory

**Przykład 16a**

Temat: Obliczanie pola i obwodu prostokąta po podaniu boków.

1) dane wejściowe: <b>a</b> -bok <b>b</b> -bok	
2) dane wyjściowe <b>Pole- pole prostokąta</b> <b>Obw-obwód prostokąta</b>	
3) ograniczenia na dane wejściowe <b>a&gt;0</b> <b>b&gt;0</b>	
4) rysunki i wzory <b>Pole=<math>a*b</math></b> <b>Obw=<math>2*(a+b)</math></b>	

**Sposoby przedstawiania algorytmów:**

- 1) Opis słowny (np. książka kucharska),
- 2) Schemat blokowy,
- 3) Lista kroków,
- 4) program w dowolnym języku programowania
- 5) Pseudokod.

**Zadanie 15c**

- 1) Wykonaj specyfikację problemu dla problemu obliczania pola trójkąta, po podaniu wysokości i podstawy ( 4 punkty wykonywania specyfikacji problemu)
- 2) Znajdź w Internecie inny sposób przedstawiania algorytmów niż zaprezentowany w instrukcji i zapisz go w zeszycie wraz z krótkim opisem.

**Zadanie 16**

Temat: Napisz program znajdujący największy wspólny dzielnik dla nieujemnych liczb całkowitych  $n$  i  $m$  algorytmem Euklidesa  $n \geq m$ . Największy wspólny dzielnik oznaczamy symbolem  $NWD(m,n)$ .

**Wykonaj:**

- 1) Zapisz temat zadania.
- 2) Zapisz jakie elementy zawiera specyfikacja problemu.
- 3) Jako ćwiczenie zapisz specyfikację problemu dla: obliczanie pola trójkąt po podaniu podstawy wysokości ( wszystkie cztery punkty).
- 4) Zapisz sposoby przedstawiania algorytmów.
- 5) Zapisz w zeszycie specyfikację dla problemu znajdowania NWD algorytmem Euklidesa.



6) Zapisz w zeszycie listę kroków dla problemu znajdowania NWD algorytmem Euklidesa.

### Specyfikacja dla algorytmu Euklidesa.

**Dane:** Dwie liczby naturalne  $m$  i  $n$ ,  $m \leq n$ .

**Wynik:**  $NWD(m,n)$  – największy wspólny dzielnik  $m$  i  $n$

### Lista kroków dla algorytmu Euklidesa.

**Krok 1** Jeśli  $m=0$ , to  $n$  jest szukany dzielnikiem. Zakończ algorytm.

**Krok 2**  $r=(n \bmod m)$ ,  $n=m$ ,  $m=r$ . Wróć do kroku 1

7) Wykonaj symulację znajdowania  $NWD(21,14)$  w postaci uzupełnionej tabeli.

przebieg algorytmu	n	m	r
1	?	?	?
.....	.....	.....	.....
.....	.....	.....	.....

8) Wykonaj symulację znajdowania  $NWD(1517,1073)$  w postaci uzupełnionej tabeli.

przebieg algorytmu	n	m	r
1	?	?	?
.....	.....	.....	.....
.....	.....	.....	.....

9) Zapisz algorytm w postaci schematu blokowego.

10) Wykonaj program.

---

## **Zadanie 16a**

Temat; Rozwiązanie zadania maturalnego. Użycie pętli while do rozwiązywania problemu.

Wykonaj:

1) Przepisz temat,

2) Przepisz specyfikację oraz listę kroków dla tego problemu dla zadania 16a)

### Specyfikacja algorytmu:

**Dane:**  $N$  – liczba całkowita większa od 0.

**Wynik:** wartość zmiennej **wyn**

### Algorytm w postaci listy kroków:

Krok 0: wczytaj  $N$

Krok 1:  $wyn := 0$ ;  $d := 2$ ;

Krok 2: Dopóki  $d \leq (N \text{ div } 2)$  wykonuj kroki 2.1 i 2.2;

    Krok 2.1: Jeżeli  $N \bmod d = 0$ , to  $wyn := wyn + 1$ ;

    Krok 2.2:  $d := d + 1$ ;

Krok 3: wypisz na ekranie  $wyn$

*Koniec listy kroków.*

**UWAGA1:** „ $N \bmod d$ ” – jest równe reszcie z dzielenia całkowitego liczby  $N$  przez  $d$ ,  
np.  $10 \bmod 5 = 0$ ,  $10 \bmod 3 = 1$ .

„ $N \text{ div } 2$ ” – jest równe wynikowi dzielenia całkowitego liczby  $N$  przez  $d$ ,  
np.  $10 \text{ div } 5 = 2$ ,  $10 \text{ div } 3 = 3$ .

„ $:=$ ” – oznacza instrukcję przypisania.

### Uwaga2:

Instrukcja **div** w cpp jest realizowana jako zwykłe dzielenie, ale zmienna, do której zapisujemy wynik musi być **int**. Instrukcja **mod** w cpp jest realizowana jako %

**Wykonaj ciąg dalszy:**

- 3) Zapisz w zeszycie jak wykonywana jest instrukcja `div` i `mod` w CPP,  
 4) Uruchom program i dokonaj kompilacji wg listy kroków podanej powyżej.

Użyj:

- `do .....while`,  
 → wszystkie zmienne naturalne,  
 → wyrażenia `wyn := wyn + 1` `d := d + 1` zapisz z użyciem **inkrementacji**, takiej jak używa się w CPP  
 → instrukcja `if` pisze się z małej litery i porównanie jest z użyciem `==` (dwa znaki =)

5) Przy użyciu programu uzupełnij tabelę, zapisz ją w zeszycie:

N → dana wejściowa	Wyn → dana wyjściowa
10	2
90	10
17	0
(Numer w dzienniku)*10	Uzupełnij sam
Dowolna Liczba pierwsza większa od (100+nr_dziennika)	Uzupełnij sam

6) Co jest wynikiem działania powyższego algorytmu? Czyli co on oblicza. Jak podpowieść traktuj użycie instrukcji **mod** (odpowieść w tabeli → jest poniżej, zapisz ją w zeszycie? (Odpowiedź udziel w tabeli)

7) Jaki jest sens stosowania zmiennej **d** czyli jaka liczba jest trzymana w tej zmiennej. (Odpowiedź udziel w tabeli).

8) Dlaczego koniec pętli `while` ma następujący warunek  **$d \leq (N \text{ div } 2)$**  (odpowieść udziel w tabeli).

9) Podaj wszystkie wartości N, dla których kroki 2.1 i 2.2 nie zostaną wykonane ani razu. (Odpowiedź udziel w tabeli)

Podpunkt	Odpowiedź
6)	
7)	
8)	
9)	

10) program w zeszycie

Koniec zadania

## INSTRUKCJA WARUNKOWA if

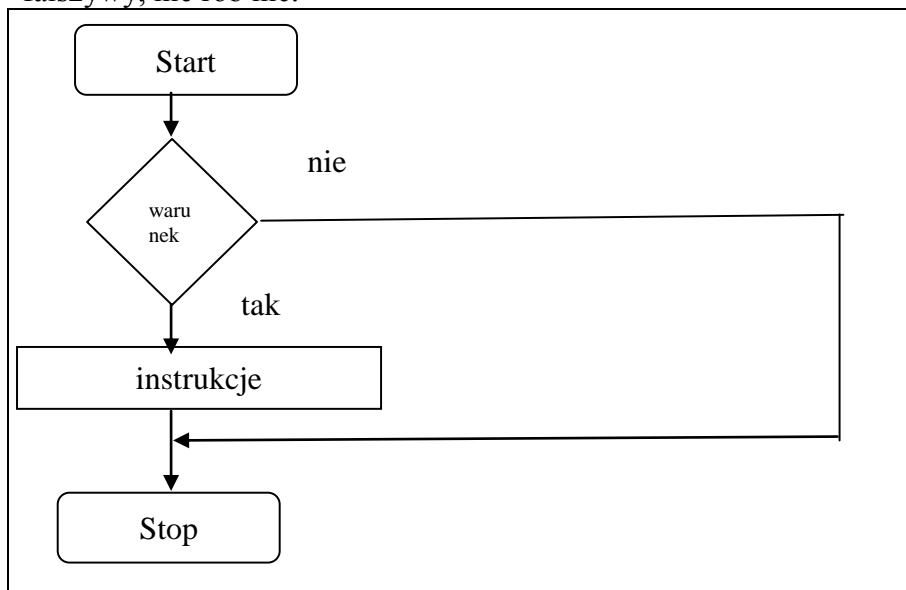
Instrukcja warunkowa ma postać:

```
if (Wyrażenie)
{
    Instrukcja1;
    .....
    InstrukcjaN;
}
```

np.

```
if (a<b) { cout<<"Liczba a jest mniejsza od b"<<endl; }
```

Jeśli warunek jest prawdziwy, wykonaj instrukcję lub grupę instrukcji. Gdy warunek jest fałszywy, nie rób nic.



## INSTRUKCJA WARUNKOWA if else

Instrukcja warunkowa ma postać:

```
if (Wyrażenie)
{
    Instrukcja1;
    .....
    InstrukcjaN;
}
else
{
    Instrukcja1;
    .....
    InstrukcjaN;
}
```

np.

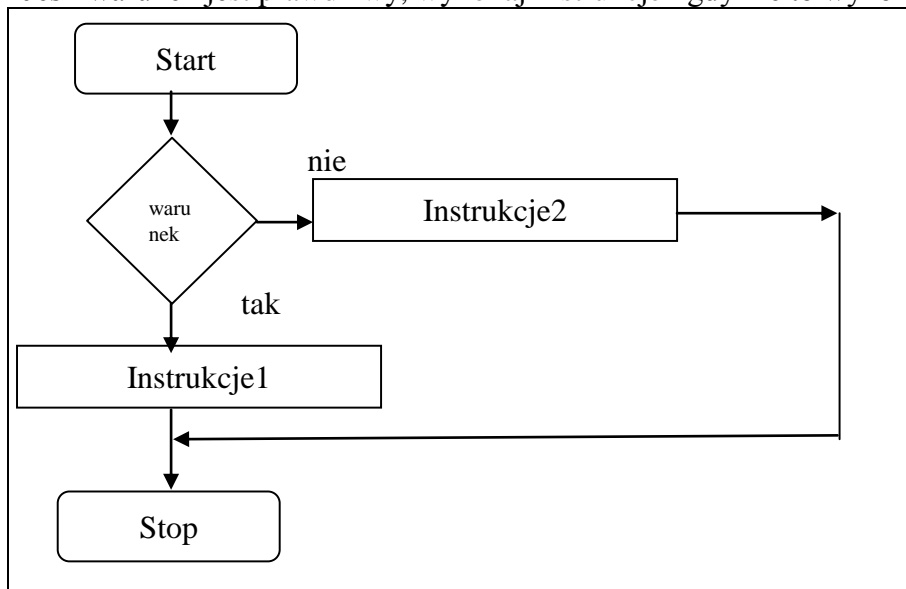
```
if (a<b)
```

```
{cout<<"Liczba a jest mniejsza od b"<<endl;}
```

```
else
```

```
{cout<<"Liczba a jest większa lub równa od b"<<endl;}
```

Jeśli warunek jest prawdziwy, wykonaj Instrukcje1 gdy nie to wykonaj Instrukcję2. Gdy



### Przykład 16b

Temat: Prezentacja instrukcji **if** dla podania komunikatu o nieprawidłowego wczytania danych.

Wykonaj:

- 1)Przepisz temat.
- 2)Zapisz teorię +schemat blokowy dotyczącą instrukcji if.
- 3)Uruchom przykład
- 4)Wpisz przykład do zeszytu.

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    system("chcp 1250");
```

```

system("cls");
int waga;
do
{
    printf("Podaj Twoja w [kg] wagę=");
    scanf("%d",&waga);
    if (waga<=0)
        { printf("waga musi być większa od zera\n");    }
    }
while(!(waga>0));
printf("waga wczytana poprawnie\n");
system("PAUSE");
return EXIT_SUCCESS;
}

```

### Zadanie 17

Temat: Użycie instrukcji **if .....else.....** oraz operatorów **AND** oraz **OR**.

Wykonaj:

- 1)Przepisz temat,
- 2)Zapisz jak tworzone są w CPP operatory **AND** oraz **OR**
- 3)Zapisz teorię +schemat blokowy dotyczącą instrukcji **if.....else.....**.
- 4)Zapisz zadanie do zeszytu (pamiętaj o wcięciach).

#### Treść zadania

Napisz program z użyciem **if ....else...** ( jednego) oraz operatorów **AND** lub **OR**

- a)Napisze „Będę sprawdzał czy liczba należy do przedziału <-5,6)
- b)Będzie się pytał o liczbę wczytaną do zmiennej z trzema literami nazwiska np. **x\_kow**.
- c)następnie wypisywał jeden z komunikatów:  
liczba należy do przedziału <-5,6) lub liczba nie należy do przedziału <-5,6)

### Zadanie 18

Temat: Program rozpoznaje znak z klawiatury z zastosowaniem **if...else** zagnieżdżonej.

Treść zadania: Komputer pyta się o liczbę i rozpoznaje znak liczby możliwe odpowiedzi programu:

- Liczba większa od zera
- Liczba mniejsza od zera
- Liczba równa zera

Użyj dwóch instrukcji **if else**.

### Zadanie 18a

Temat: Napisać program obliczający pole kwadratu.

Sprawdź czy bok kwadratu jest większy od zera. Jeżeli jest większy oblicz pole jeśli nie to podaj komunikat "bok musi być większy od zera". W programie użyj **tylko jednej** instrukcji **IF...THEN...ELSE**. Zapewnij powtarzalność programu jak zapewnić powtarzalność patrz przykład poprzedni..

Wykonaj w zeszycie schemat blokowy dla zadania.

### Zadanie 19

Temat: Program obliczający wartość funkcji danej wzorem w postaci klamkowej.

Wykonaj:

1)Przepisz temat,

2)Rozwiąż zadanie

Zmienne z trzema literami nazwiska.

Napisz program obliczający wartość funkcji danej wzorem

$$f(x) = \begin{cases} 1 & \text{dla } x \geq 1 \\ x^2 & \text{dla } -1 < x < 1 \\ 1 & \text{dla } x \leq -1 \end{cases}$$

Dopisz do programu jego powtarzanie czyli po podaniu  $x$  i obliczeniu  $f(x)$  komputer spyta się czy ma powtórzyć program (jak to zrobić patrz przykład poprzedni) i na wczytaną literę  $\rightarrow$  pierwsza litera Twojego imienia np. Jan będzie to litera „J” lub „j” nastąpi ponowne pytanie o  $x$  i obliczenie  $f(x)$ .

3) Wykonaj w zeszycie tabelę i uzupełnij ja.

X					0					
Y										

Uzupełnij pierwszy wiersz liczbami dwie  $> 1$ , dwie  $< -1$ , trzy z przedziału  $(-1,0)$ , trzy z przedziału  $(0,1)$

Pisz  $x$  a komputer obliczy  $y$ . Zapisz  $x$  i  $y$  tabeli. Uzupełnij całą tabelę.

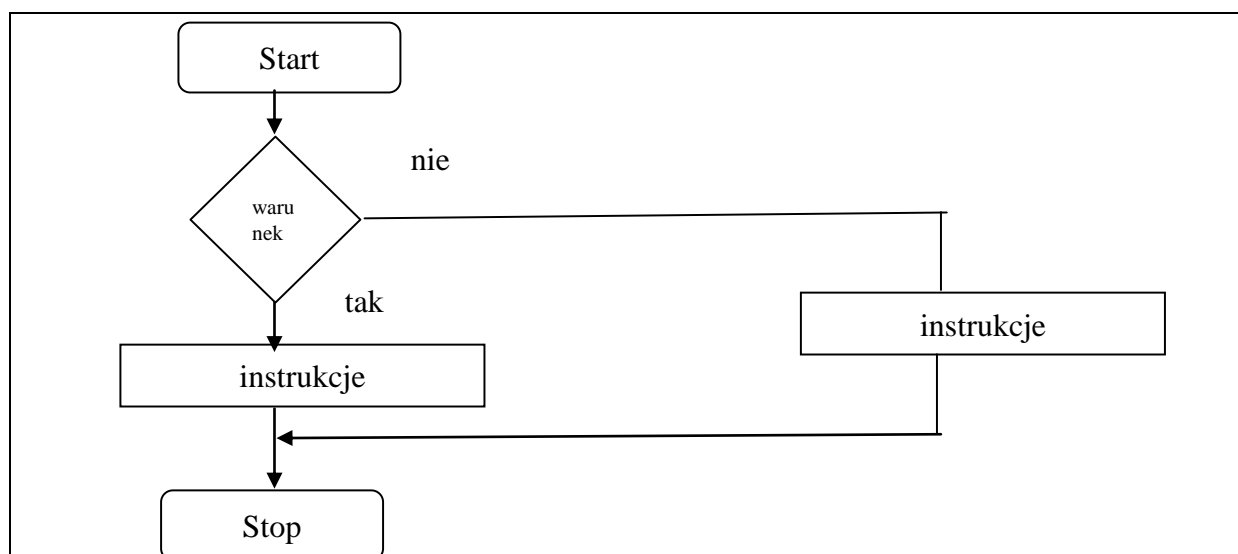
4)Posługując się programem wykonaj wykres funkcji  $f(x)$  w zeszycie, 5 cm na osi  $X$  i  $Y$  to jeden( czyli duża skala).

Punkty z tabeli zaznacz w układzie  $XY$ . Połącz punkty a otrzymasz wykres.

5)Przepisz zadanie do zeszytu.

## INSTRUKCJA WARUNKOWA if...else

```
if (Wyrażenie)
{
    Instrukcja1
    .....
    InstrukcjaN;
}
else
{
    Instrukcja1
    .....
    InstrukcjaN;
}
```



Jeśli Wyrażenie jest prawdziwy to zostanie wykonana Instrukcja1, w przeciwnym razie wykonana zostanie Instrukcja2. Instrukcje mogą być instrukcjami grupującymi. Słowa kluczowe if i else mogą być stosowane wielokrotnie. Pozwala to tworzyć np. tzw. zagnieżdżenia.

### Przykład instrukcji zagnieżdżonych:

```
if (a>0) if (a<100) printf("Dwucyfrowa"); else printf("100+");
```

inaczej:

```
if(a>0) {if(a<100) printf("Dwucyfrowa"); else printf("100+");}
```

Zapis 100+ oznacza "sto i więcej".

### **Zadanie 20**

Temat: Użycie instrukcji if else do sprawdzania hasła w postaci kodu dwu-literowego.



### Treść

Napisać program, który będzie można uruchomić przez podanie kodu dwu-literowego. Komputer pyta się o pierwszą literę kodu, a następnie o drugą literę kodu następnie, jeśli jest prawidłowy szyfr, to komputer drukuje "DZIEŃ DOBRY", jeśli jest zły "ŻEGNAJ". Przy podawaniu szyfru wielkość liter nie będzie miała znaczenia. Użyj jednej instrukcji if else. Wielkość liter nie będzie miała znaczenia. Gdy program działa, zmień instrukcję if else na wersję skróconą patrz przykład poniżej. Gdy wpisujesz treść programu do zeszytu zapisz tak, aby można było zobaczyć obie wersje(normalną i skróconą if else).

### Uwaga1:

Zmienne z **trzema literami nazwiska** np. znak\_kow

### Uwaga2:

Zmienne do wczytania kodu to **char**.

### Uwaga3:

Do wczytywania zmiennej użyj instrukcji **cin**

Koniec zadania 21

---

### Przykład→ciekawostka:

C++ pozwala na krótszy zapis instrukcji warunkowej:

```
(a>b)? MAX=a : MAX=b;
inaczej:
if (a>b) MAX=a; else MAX=b;
```

---

### Zadanie 21

Temat: Użycie pętli do { } while(????????????); do powtarzania programu na zadany warunek określony przedziałem.

Program wykona:

### Pierwsza czynność:

Napisze: Będę powtarzał program, aż do wczytania liczby z przedziału <-3,2)

### Druga czynność:

program, taki, że Komputer będzie wykonywał program (powtarzał napis DZIEŃ DOBRY) aż do wczytania liczby z przedziału <-3,2) użyj pętli `do { } while(????????????);`

Uzupełnij tabelę w zeszycie po wykonaniu programu:

Wartość wczytanej liczby	<u>Reakcja komputera:</u> Możliwe wpisy do tabeli: <ul style="list-style-type: none"> <li>• Powtórzył</li> <li>• Nie powtórzył</li> </ul>
-30	
-4	
-3	
0	
2	
10	
15	

### **Przykład 18**

Temat: Program oblicza pole oraz obwód okręgu po podaniu promienia. Sprawdza wczytywane dane oraz wybierana jest opcja obliczeń.

Wykonaj:

- 1)Przepisz temat,
- 2)Uruchom przykład
- 3)Uzupełni tabelę w zeszycie:

Wielkość wczytywana	Odpowiedź programu, ( co pokazało się na ekranie).
Opcja=3	
Opcja=1	
R= - 1	
R=2	
Opcja=2	
R= - 3	
R=3	

4)Wpisz przykład do zeszytu

5)W zeszycie otocz ramką tą linijkę, która tworzy stałą w przykładzie

```

#include <cstdlib>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <conio.H>
using namespace std;
int main(int argc, char *argv[])
{
    float const PI=3.14159; short int opcja;
    float pole,obwod,r;
    printf("Umiem obliczac pole oraz obwod okregu\n");
    printf("po podaniu promienia\n");
    printf("umiem sprawdzac wczytywane dane\n");
    printf("musisz wybrac opcje\n");
    printf("*****\n");
    printf("1-pole okregu\n");
    printf("2-obwod okregu\n");
    printf("*****\n");
    do
    {
        printf("podaj opcje\n");
        scanf("%d",&opcja);
    }
    while((opcja!=1)&&(opcja!=2));
    do
    {
        printf("podaj promien okregu\n");
        scanf("%f",&r);
    }
    while (r<=0);
    printf("wczytane dane");
    printf("\nr=%.2f",r);
    printf(" cm");
    printf("\nwyniki");
    if (opcja==1)
    {
        pole=PI*r*r;
        printf("\npole=%.2f",pole);
        printf(" cm2");
    }
    if (opcja==2)
    {
        obwod=2*PI*r;
        printf("\nobwod=%.2f",obwod);
        printf(" cm");
    }
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

### **Zadanie 22**

Zmienne z trzema literami nazwiska.

Napisz program, który obliczać będzie

- 1.pole trójkąta
- 2.promień koła opisanego
- 3.promień koła wpisanego

po podaniu długości boków.

Program sprawdza poprawność wprowadzenia danych tzn. czy  $A, B, C > 0$  oraz czy boki tworzą trójkąt. {jaki to warunek? }

Dane do sprawdzenia:

A=3 B=4 C=5

Wyniki:

S=6.00 cm<sup>2</sup>

R= 2.50 cm

r=1 cm

**Wzory:**

p– połowa obwodu

S– pole trójkąta

r– promień wpisany

R– promień opisany

A, B, C boki trójkąta

$$p = \frac{(A + B + C)}{2}$$

$$S = \sqrt{p(p - A)(p - B)(p - C)}$$

$$r = \frac{S}{p}$$

$$R = \frac{(ABC)}{(4S)}$$

---

### **Zadanie 23**

Zmienne z trzema literami nazwiska.

Napisz program, który będzie rozwiązywał równanie kwadratowe w postaci

$A \cdot x^2 + B \cdot x + C = 0$ .

Sprawdzone będzie czy  $A \neq 0$ . Rozwiązania podawane będą w zależności od wartości delty.

<u>Dane do sprawdzenia:</u> A=1 B=1 C=1 <u>Wyniki:</u> Delta= -3 Brak rozwiązań	<u>Dane do sprawdzenia:</u> A=1 B=2 C=1 <u>Wyniki:</u> Delta= 0 Jedno rozwiązanie X0= - 1	<u>Dane do sprawdzenia:</u> A=1 B=2 C= -3 <u>Wyniki:</u> Delta = 16 Dwa rozwiązania X1= -3 X2= 1
---	--	--

## Algorytm

Wczytaj A, B , C.

```
gdy A=0
{
    pisz: to nie jest równie kwadratowe
}
w przeciwnym przypadku
{
    pisz: to jest równanie kwadratowe
    delta=B^2-4AC
    gdy delta<0
    {
        pisz: brak rozwiązań
    }
    gdy delta==0
    {
        pisz: jedno rozwiązanie
        x0= -b/(2A)
        pisz x0
    }
    gdy delta>0
    {
        pisz: dwa rozwiązania
        x1=(-b-sqrt(delta))/(2A)
        x2=(-b+sqrt(delta))/(2A)
        pisz x1
        pisz x2
    }
}
```

## **Zadanie 24**

Zmienne z trzema literami nazwiska.

Napisz program, który będzie rozstrzygał po wczytaniu trzech różnych liczb A B C , która jest największa , średnia i najmniejsza np.

A- średnie

B- największe

C- najmniejsze

Sprawdzone będzie również czy liczby; są od siebie różne. W przypadku podania liczb o tej samej wartości to komputer wyda stosowny komunikat np. A=B lub A=B=C itp. i poprosi o ponowne wczytanie liczb.

## **INSTRUKCJE break i continue.**

Instrukcja **break** powoduje natychmiastowe bezwarunkowe opuszczenie pętli dowolnego typu i przejście do najbliższej instrukcji po zakończeniu pętli. Jeśli w pętli for opuścimy wyrażenie logiczne, to zostanie automatycznie przyjęte 1. Pętla będzie, zatem wykonywana bezwarunkowo w nieskończoność

Instrukcja **continue** powoduje przedwczesne, bezwarunkowe zakończenie wykonania wewnętrznej instrukcji pętli i podjęcie próby realizacji następnego cyklu pętli. Próby, ponieważ najpierw zostanie sprawdzony warunek kontynuacji pętli.

## **INSTRUKCJE switch i case.**

Instrukcja switch dokonuje WYBORU w zależności od stanu wyrażenia przełączającego (selector) jednego z możliwych przypadków - wariantów (case). Każdy wariant jest oznaczony przy pomocy stałej - tzw. ETYKIETY WYBORU. Wyrażenie przełączające może przyjmować wartości typu int. Ogólna postać instrukcji jest następująca:

```
switch (selector)
{
    case STAŁA1: Ciąg_instrukcji-wariant 1;
    case STAŁA2: Ciąg_instrukcji-wariant 2;
    .....
    case STAŁAn: Ciąg_instrukcji-wariant n;
    default : Ostatni_ciąg_instrukcji;
}
```

Należy podkreślić, że po dokonaniu wyboru i skoku do etykiety wykonane zostaną również WSZYSTKIE INSTRUKCJE PONIŻEJ DANEJ ETYKIETY. Jeśli chcemy tego uniknąć, musimy dodać rozkaz break.

## **#define.**

Użycie #define polega na zastąpieniu w tekście programu jednych łańcuchów znaków przez inne.  
np.

```
# define Program main()
# define Begin {
# define Writeln printf
# define Readln getch()
# define End }
```

Takie zdefiniowanie pozwala zastąpić instrukcje CPP instrukcjami Pascala.

### Przykład 19

Zapisz w zeszycie składnię instrukcji

- INSTRUKCJE switch i case (znaczenie break).
- #define.

#### Treść programu

Program po podaniu numeru dnia tygodnia odpowie, jaki to dzień tygodnia. W przykładzie zastosowano #define.

Wykonaj:

- Wpisz przykład do komputera oraz przetestuj go.
- Wykonaj schemat blokowy przykładu.

```
#define pisz printf //dla przypomnienia
#include <cstdlib>
#include <iostream>
#include <stdio.h>
using namespace std;

int main(int argc, char *argv[])
{
    int Numer_Dnia;
    pisz("\nPodaj numer dnia tygodnia\n");
    scanf("%d",&Numer_Dnia);
    switch(Numer_Dnia)
    {
        case 1: {pisz("PONIEDZIALEK");break;}
        case 2: {pisz("WTOREK");break;}
        case 3: {pisz("SRODA");break;}
        case 4: {pisz("CZWARTEK");break;}
        case 5: {pisz("PIATEK");break;}
        case 6: {pisz("SOBOTA");break;}
        case 7: {pisz("NIEDZIELA");break;}
        default: pisz("\nniema takiego dnia tygodnia!!!");
    }
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Zwróć uwagę, że w przykładzie wariant default zostanie wykonany ZAWSZE nawet, jeśli podasz liczbę większą niż 7.

### Zadanie 25

Temat: Użycie instrukcji case, swich, oraz #define

Wykonaj:

a) Program z przykładu poprzedniego zmodyfikuj w taki sposób, aby: po podaniu pierwszej litery kraju, z którym graniczy Polska odpowie, jaki to kraj.

Np. po podaniu „c” lub „C” wyświetlone zostanie „Czechy”. Wielkość liter nie będzie miała znaczenia.

b)W przykładzie zastosowano #define do przedefiniowania Ty zastosuj:

**case**           →     na     **przypadek**  
**switch**        →     na     **przełącz**

c)Zmienną selektora zadeklaruj jako char.

Zmień literkę d na taką aby wczytywać znaki(stringi)

```
scanf("%d",&);
```

Do sprawdzania użyj znaku apostrofu np. 'C'

d)Zmienne z trzema literami nazwiska np. kraj\_kow gdy uczeń nazywa się Kowalski.

*Koniec zadania 24*



## Skok bezwarunkowy goto (idź do...),

Gdy chcesz, aby program przeszedł do określonego miejsca w programie możesz użyć tak zwanej **etykiety**.

Etykieta składa się z:

- polecenia **goto** nazwa; np. goto skocz;
- oraz nazwy zakończonej dwukropkiem np. **dawaj:** gdzie trzeba skoczyć.

UWAGA: Po każdej etykiecie musi wystąpić co najmniej jedna instrukcja. Jeśli etykieta oznacza koniec programu, to musi po niej wystąpić instrukcja pusta. Należy tu zaznaczyć, że etykieta nie wymaga deklaracji.

np.

```
goto dawaj;
```

```
// część programu
```

```
:dawaj
```

## Przykład 20

Temat: Po wczytaniu liczby program poda jak jest litera w kodzie ASCII o liczbę miejsc dalej od litery A

Wykonaj:

a) Zapisz w zeszycie co to jest etykieta, jak jest zbudowana, jakie jest znaczenie znaku „:” , przepisuj również uwagę.

b) odczytaj z tabeli kodów ASCII numer kodu dla litery A oraz K o ile numerów się różnią

c) Wpisz przykład i uruchom.

d) W zeszycie zanotuj, co oznacza:

- `printf("%c", 'A');`
- `printf("\t\t\t%d", 'A');`
- `printf("\n%c\t\t\t%d\n", 'A'+liczba, 'A'+liczba);`

Przepisz linijkę programu i dokładnie opisz znaczenie zastosowanych wszystkich znaków.

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    short int liczba;
```

```
    printf("Wydrukuj A jako \nLiteral znakowy:\tKod ASCII:\n");
```

```
    printf("%c", 'A');
```

```
    printf("\t\t\t%d", 'A');
```

```
    skocz:                                     // miejsce skoku
```

```
    printf("\nmiem podac kod ASCII o n wieksza od kody A ");
```

```
    printf("\npodaj mi liczbe ? ");
```

```
    scanf("%d", &liczba);
```

```
    if (liczba<0) goto koniec;                 //skok do etykiety koniec
```

```
    printf("\n%c\t\t\t%d\n", 'A'+liczba, 'A'+liczba);
```

```
    goto skocz;                                //skok do etykiety skocz
```

```
    koniec:                                    // miejsce skoku
```

```
    system("PAUSE");
```

```
    return EXIT_SUCCESS;
```

```
}
```

**Zadanie 26**

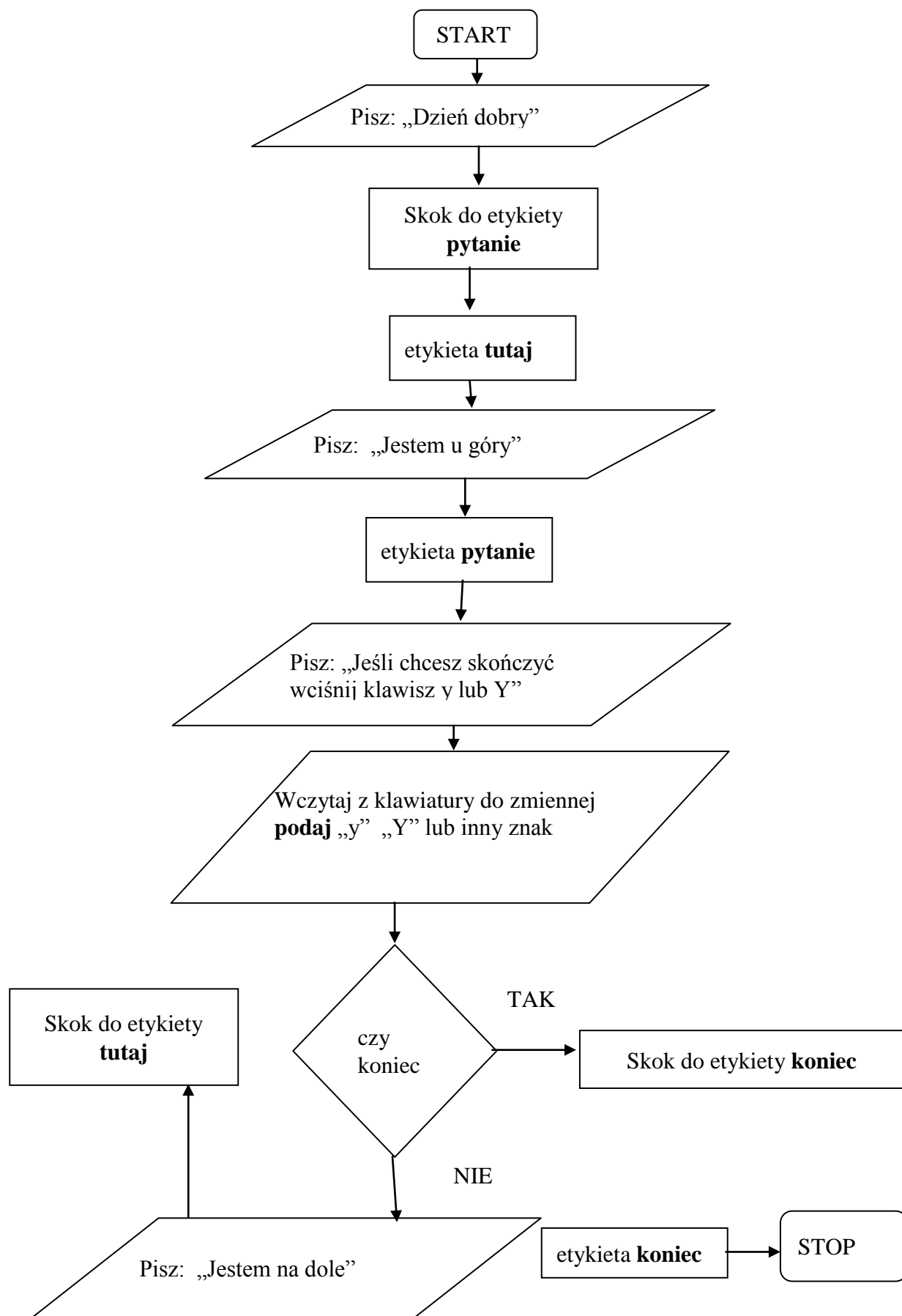
Wykonaj program na podstawie schematu blokowego z użyciem etykiet.

Nazwy etykiet:

koniec\_numer\_w\_dzienniku np. koniec\_17

tutaj\_numer\_w\_dzienniku np. tutaj\_17

pytanie\_numer\_w\_dzienniku np. pytanie\_17



---

---

## Różnica między instrukcją getch() a kbhit()

### **a) kbhit()**

Aż do naciśnięcia dowolnego klawisza.

### **b) !kbhit()**

Zapis !kbhit() oznacza "nie naciśnięto klawisza", czyli w buforze klawiatury nie oczekuje znak.

### **c) getch()**

Zwróć uwagę, że funkcja getch() może oczekiwać na klawisz w nieskończoność. Aby uniknąć kłopotliwych sytuacji, czasem znacznie wygodniej jest zastosować kbhit(), szczególnie, jeśli czekamy na dowolny klawisz.

### **Zadanie przykładowe na sprawdzian czas 85 minut**

Po wykonaniu każdego zadania zgłaszać nauczycielowi. Kolejność dowolna.

ocena: punkty      ocena

0–1 ndst	2 dopuszczający	3 dostateczny	4 dobry	5 bardzo dobry	6 celujący
----------	-----------------	---------------	---------	----------------	------------

#### **Zadanie1** (jeden punkt)

Program będzie uruchamiany przez podanie kolejno : numeru dnia urodzenia+10, pierwszej litery imienia oraz pierwszej litery nazwiska. Wielkość liter nie będzie miała znaczenia

– Gdy podany jest prawidłowo szyfr komputer napisze "Brawo znasz moje parametry"

– Gdy szyfr jest nieprawidłowy komputer napisze "Nie znasz szyfru żegnaj".

#### **Zadanie2** (jeden punkt)

Komputer napisze:

będę powtarzał program czyli napis gdy podasz liczbę z przedziału  $(-1,3>$

następnie zapyta:

czy chcesz powtarzać część programu piszącą "SPRAWDZIAN ZADANIE2".

Jeśli tak to podaj liczbę z przedziału  $(-1,3>$  jeśli nie to pozostałe liczby

Następnie nastąpi wczytanie liczby.

Następnie nastąpi powtórzenie napisu "SPRAWDZIAN ZADANIE2" lub wyjście z programu.

#### **Zadanie3** (jeden punkt)

Zostanie wyświetlone menu:

1– obliczanie objętości walca

2– obliczanie powierzchni całkowitej walca

Wybór wariantu 1 lub 2 wykonanie obliczeń oraz wyprowadzenie wyników na monitor z podaniem jednostek czyli  $\text{cm}$  i  $\text{cm}^2$ .

Sprawdzanie poprawności wczytanych danych:

♠ numeru wariantu (powtórzenie MENU → gdy numer jest nieprawidłowy)

♠ danych liczbowych

#### **Zadanie4** (jeden punkt)

Po wczytaniu wartości dwóch kątów program poda wartość trzeciego kąta i odpowie jaki to jest trójkąt (prostokątny, równoboczny, równoramienny, różnokątny).

Zapewnij powtarzalność programu na literę „j” lub „J”.

Czyli jeśli chcesz powtórzyć podaj literę „j” lub „J”.

#### **Zadanie5** (jeden punkt)

Napisz program rozpoznający, jaką cyfrę wczytał użytkownik. Użyj instrukcji swich case. Cyfra wczytywana, jako liczba naturalna. Gdy wczytana jest nieistniejąca cyfra to komputera poinformuje poprzez komunikat „Taka cyfra nie istnieje”. Gdy wczytana jest istniejąca cyfra to komputera poinformuje poprzez komunikat „Taka cyfra istnieje i jest nią cyfra np. zero”.

Zapewnij powtarzalność programu. Do powtarzalności użyj instrukcję gotoxy. Zmień instrukcje **goto** na skocz przy użyciu #define.

#### **Zadanie6** (jeden punkt)

Z użyciem pętli „do while” lub „while” napisz program wykonujący rozkład liczby na czynniki pierwsze.

Podajemy liczbę naturalną i ukazują się rozkład w formi jak poniżej:

Dla liczby  $N=8778$

Na ekranie uzyskasz

$8778=2*3*7*11*19$

## Część 5 pętla FOR.

### Instrukcja pętli iteracyjnej FOR

Postać pętli for jest następująca:

a) dla jednej instrukcji wykonywanej przez pętlę  
for (wyrażenie\_inicjujące; wyrażenie\_logiczne; wyrażenie\_kroku)  
Instrukcja;

b) dla wielu instrukcji wykonywanej przez pętlę  
for (wyrażenie\_inicjujące; wyrażenie\_logiczne; wyrażenie\_kroku)  
{  
Instrukcja1;  
Instrukcja2;  
.....  
Instrukcja;  
}

### Wykonanie pętli for przebiega następująco:

1. wykonanie jeden raz wyrażenia inicjującego.
2. obliczenie wartości logicznej wyrażenia logicznego.
3. jeśli w\_logiczne ma wartość prawda (true) nastąpi wykonanie instrukcji.
4. obliczenie wyrażenia kroku.
5. powtórne sprawdzenie warunku - czy wyrażenie logiczne ma wartość różną od zera. jeśli wyrażenie logiczne ma wartość zero, nastąpi zakończenie pętli.

Przykłady

a) pełna instrukcja for

```
for ( int n=0; n<=100; n++)  
{  
    printf("%d\t", n);  
}
```

b) z wyrażeniem inicjującym obliczonym poza pętlą

```
.....  
{  
    float n,p=2;  
    n=(p+1)/(p+1);  
    for (; n<=100; n++)  
    {  
        printf("%.2f\t", n);  
    }
```

c) z obliczanym warunkiem przerywania pętlę

```
{  
    float y=0, n=0;  
    for (; (sqrt(n)-y)<=3.0; n++)  
    {  
        y=sqrt(n); printf("%.3f\t", y);  
    }
```

### Uwagi:

#### Uwaga1

Każde z tych wyrażeń może zostać pominięte patrz → for(;;).

#### Uwaga2

warunek jest testowany przed wykonaniem instrukcji. Jeśli zatem nie zostanie spełniony warunek, instrukcja może nie wykonać się ani raz.

## **PRZYKŁAD 21**

Temat: Program piszący na ekranie 100 wykrzykników w jednym ciągu.

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <conio.h>

using namespace std;

int main(int argc, char *argv[])
{
    for ( int i=0;i<=99;i++)
    {
        printf("!");
    }
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

**Uwaga:** Przy rozwiązywaniu następnego zadania wykorzystaj treść tego przykładu.

## **ZADANIE 27**

Temat: Napisać program drukujący 20 gwiazdek. Każda w nowej linii, wykorzystaj instrukcję FOR.

**Uwaga:** Przy rozwiązywaniu następnego zadania wykorzystaj treść tego przykładu.

## **ZADANIE 28**

Temat: Napisz program z użyciem pętli for wypisujący liczby parzyste od numeru\_miesiąca\_urodzenia np. od 12 do numeru\_miesiąca\_urodzenia +30 np. do 42

Wykonaj schemat blokowy

**Wskazówka:** w trzeciej sekcji instrukcji for zamiast i++ w pisz warunek tak, aby zmienna i przyjmowała wartości parzyste, czyli liczby zwiększające się o dwa ( pełna lub skrócona wersja) . Start pętli od: zaczynające się liczby parzystej a koniec na kończącej liczbie parzystej zawartej w przedziale ucznia.

Zmienne z trzema literami nazwiska.

**Zapisz** w zeszycie pod listingiem czcionką 1,5cm, trzecią sekcję pętli ( czyli tą, w której gdzie było zwiększanie o dwa pętli).

## Przykład 22

### Temat:

Program sumuje liczby aż do wciśnięcia liczby 0.

W przykładzie poniżej nieskończoną pętlę przerywa po podaniu z klawiatury wartości zera instrukcja break.

### Wykonaj:

- Wpisz przykład do komputera oraz przetestuj go.
- Wykonaj schemat blokowy przykładu.
- Zapisz w zeszycie pod listingiem czionka 1,5cm, **nieskończoną pętlę**

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
float a, sigma=0;
```

```
for (;;)          // nieskończona pętla
```

```
{
```

```
    printf("\n Podaj liczbe do sumowania\n");
```

```
    scanf("%f", &a);
```

```
    if (a==0) break;
```

```
    sigma+=a;      // inaczej sigma = sigma + a;
```

```
    printf("\n SUMA CZESCIOWA: %f",sigma);
```

```
}
```

```
printf("Nastapil BREAK \n");
```

```
system("PAUSE");
```

```
return EXIT_SUCCESS;
```

```
}
```

## Zadanie 29

Zmienne z trzema literami nazwiska.

Program z przykładu poprzedniego zmodyfikujemy w taki sposób, aby:

- jeśli liczba jest dodatnia - dodawał ją do sumy sigma;
- jeśli liczba jest ujemna - nie robił nic, pomijał bieżącą pętlę przy pomocy rozkazu **continue**; (Ponieważ warunek wejściowy pętli jest zawsze spełniony, to pętlę zawsze uda się kontynuować.)
- jeśli liczba równa się zero - przerywał pętlę instrukcją **break**.

## ZADANIE 30a

**Temat:** Program wypisujący kody ASCII oraz liczby odpowiadające tym kodom.

### **ETAP1**

Napisać program drukujący na ekranie kody ASCII dla znaków o numerach od 33 do 255.

Wydruk powinien mieć postać(przykład jednej linii) np.

kod znaku 33 !

.....



.....

Użyj pętli. W celu wydrukowania znaku o odpowiadającym mu kodzie użyj rzutowania zmiennych (czyli zmiany typu int → char)

np. cout<<(char)i<<"\n";                      gdy i=65 wyświetli się litera A.

### **ETAP2**

Zmodyfikuj program tak aby zatrzymał się jeśli zapisze cały ekran ( np. 20 wierszy) i czekał, aż wciśniemy dowolny klawisz. Po wciśnięciu klawisza będzie drukował nowy ekran kodów. Do zmiany ekranów użyj instrukcji IF oraz MOD → % (sprawdzaj podzielność zmiennej sterującej I użytej w pętli FOR przez 20 z użyciem MOD (%) → ponieważ użyjemy dwadzieścia wierszy na ekranie) gdy zmienna I jest podzielna przez 20 to zatrzymanie ekranu i czekanie na naciśnięciu dowolnego klawisza (np. poprzez **system("PAUSE")**; czyszczenie ekranu (np. poprzez **system("cls")**); i nowe 20 kodów ASCII.

### **ETAP3**

Zmodyfikuj program tak aby drukował na ekranie trzy słupki kodów i odpowiadających mu znaków.

### **ZADANIE 30**

Napisać program drukujący liczby od 180+nr\_w\_dzienniku do 200+nr\_w\_dzienniku oraz ich pierwiastki. Wykorzystaj treść poprzedniego programu. Zastanów się jak liczba będzie początkiem pętli a jaka końcem oraz w jaki sposób uzyskać wydruk zadania jak poniżej. Zwróć uwagę, że wydruk ma dwa miejsca po przecinku.

```
printf("y=%6.2f",y);
```

Jest to przykład matrycy 6 znaków dwa miejsca po przecinku

Gdy chcesz użyć do wyświetlania cout dwa miejsca po przecinku możesz uzyskać  
cout.precision(2);

Przykład wydruku zadania dla ucznia, który ma numer w dzienniku 20.

SQRT(200)=14.14

SQRT(201)=14.18

.....

.....

### ZADANIE 31

**Temat:** Napisać program obliczający sumę szeregu harmonicznego od wyrazu pierwszego do milionowego.

Ćwiczenia wstępne:

oblicz:

$$suma = \sum_2^4 i^2$$
$$suma = \sum_1^3 \frac{i}{i+2}$$

zapisz w postaci sumy:

$$suma = 1 + 8 + 27 + 64$$

Wyrazu szeregu harmonicznego powstają następująco:

$$a_n = \frac{1}{n}$$

sumę wyrazów od pierwszego do milionowego można zapisać z użyciem znaku sigmy

$$suma = \sum_{i=1}^{1000000} \frac{1}{i}$$

Wskazówka: Należy w każdym kroku pętli obliczyć wartość wyrazu  $a_n$  i dodać do aktualnie obliczanej sumy. Pamiętaj o wyzerowaniu zmiennej zawierającej sumę przed przystąpieniem do sumowania. We wzorze na  $n$ -ty wyraz ciągu wpisz zamiast 1 wpisz 1.0 aby dokonała się konwersja.

### **PRZYKŁAD 22a**

Temat: Wpisz program rysujący linię pionową na środku ekranu składającą się z gwizdek.

Następnie dopisz pętlę rysującą nową linię poziomą na wysokości 10.

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

using namespace std;

void gotoxy(int x, int y)
{
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main(int argc, char *argv[])
{
    for (int i=0;i<=22;i++)
    {
        gotoxy(40,i);
        cout<<"*";
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

---

### **ZADANIE 32a**

Zmodyfikuj przykład poprzedni tak aby gwiazdki rysowały się po przekątnej kwadratu o boku 20 linii ekranowych rozpoczynając od lewego górnego punktu ekranu.

### **ZADANIE 32b**

Zmodyfikuj przykład poprzedni tak aby gwiazdki rysowały się po przekątnej kwadratu o boku 20 linii ekranowych rozpoczynając od prawego górnego punktu ekranu.

```
      *
     *
    *
```

.....

### **ZADANIE 32c**

Napisać program, który z użyciem czterech pętli ( instrukcji iteracyjnej FOR) narysuje prostokąt z gwiazdek o następujących wymiarach:

ilość wierszy= 5+(reszta z dzielenia numer\_z\_dziennika przez 3),

ilość kolumn=11+(reszta z dzielenia numer\_z\_dziennika przez 2).

### **ZADANIE 32d**

Napisać program, który po wczytaniu środka linii poziomej ( podajesz x i y ) oraz długości linii (podajesz D) program rysuje ze znaków minus „-„ linię o określony środek (x,y) oraz długości D.

Wygląd ekranu programu:

Podaj zmienną (kolumnę ekranu) x=40

Podaj zmienną (wiersz ekranu) y=10

Podaj długość linii D=11

*I teraz program rysuje tę linię.*

---

## Przykład 23(24)

Temat: Program do sprawdzania czy liczba wczytana z klawiatury jest liczbą pierwszą.

Wykonaj:

- 1)Wpisz w zeszycie temat programu.
- 2)Przepisz opis problemu.
- 3)Przepisz opis algorytmu.
- 4)Po rozwiązaniu zapisz listing.
- 5)Z użyciem tego programu znajdź i zapisz w zeszycie liczbę pierwszą, która jest większa lub równa **liczbie=1000+nr\_z\_dziennika\*100+15**

Opis problemu:

**Liczba pierwsza**, liczba naturalna  $n > 1$ , dla której istnieją tylko dwa dzielniki naturalne: 1 i  $n$ . Największą znaną liczbą pierwszą jest  $2^{69725932}-1$  (7 VII 1999), liczba ta zapisana w systemie dziesiętnym składa się z ponad 2 mln cyfr.

Opis algorytmu sprawdzania czy liczba jest liczbą pierwszą.

Sprawdzamy czy kolejne liczby naturalne od 2 do pierwiastek( $n$ ) są podzielnikami liczby  $n$ . Sprawdzenie podzielności odbywa się poprzez użycie funkcji % (MOD→reszta z dzielenia), jeśli reszta z dzielenia jest zero to oznacza to, że liczba  $n$  jest podzielna przez liczbę mniejszą od  $n$  i  $n$  nie jest liczbą pierwszą. Należy zauważyć, że nie jest konieczne sprawdzanie kolejnych liczb naturalnych do 2 do  $n$  a wystarczy do **pierwiastek( $n$ )**. Zmienna **k** ma wartość domyślnie zero. Po zakończeniu programu sprawdzamy wartość zmiennej **k**, jeśli jest równa zero oznacza to, że liczba  $n$  jest pierwsza, jeśli nie (czyli jeden) to liczba nie jest pierwsza. Zmiana  $k$  zmieniana jest na wartość jeden gdy liczba  $n$  jest podzielna przez kolejną liczbę naturalną mniejszą od **pierwiasta( $n$ )**.

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

using namespace std;

int main(int argc, char *argv[])
{
    cout<<"podaj liczbe naturalna, ktora sprawdze czy jest pierwsza=";
    int pierwsza;
    int k=0;
    cin>>pierwsza;
    for(int i=2;i<=sqrt(pierwsza);++i)
    {
        if ((pierwsza) % i==0)
        {
            k=1;
            cout<<"liczba "<<pierwsza<<" to nie jest liczba pierwsza jest podzielna przez "
            <<i<<"\n";
        }
    }
    if ( k==0)
    {
        cout<<"liczba "<<pierwsza<<" to jest liczba pierwsza"<<"\n";
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### **ZADANIE 32e) (36)**

Napisać program, który po wczytaniu dwóch liczb naturalnych większych od 1, czyli liczba **dol** oraz liczba **gor** gdzie  $dol < gor$ . Wypisze sumę liczb pierwszych z tego przedziału.

Wskazówka:

Wykorzystaj poprzedni przykład. Utwórz pętlę dla zmiennej **j\_naz** ( z trzema literami nazwiska) od liczby **dol** do liczby **gor**. Utwórz zmienną **suma**. Pamiętaj o wyzerowaniu tej zmiennej. Użyj instrukcji **suma=suma + j\_naz**. Po zakończeniu pętli wyprowadź zmienna **suma**. Przy wczytywaniu danych pamiętaj, że najmniejsza liczba pierwsza to 2.

Np.

Podaj dol=2

Podaj gor=10

Suma=17

### **ZADANIE 32f**

Napisz program z użyciem pętli for rozkładający dowolną liczbę naturalną wczytaną z klawiatury na iloczyn czynników pierwszych.

Np.

Podaj liczbę naturalną=.....

18=2\*3\*3

---

---

### Przykładowy sprawdzian

#### Zadanie1(1,5 punkt)

Napisać program, który po wczytaniu dwóch liczb naturalnych większych od 1, czyli liczba **dol** oraz liczba **gor** gdzie  $dol < gor$ . Wypisze wszystkie liczby pierwsze bliźniacze. Liczby pierwsze bliźniacze to dwie liczby pierwsze różniące się między sobą o dwa np. 3 i 5.

Wygląd ekranu:

Podaj dolną liczbę przedziału przeszukiwania=1

Podaj górną liczbę przedziału przeszukiwania=20

Znalezione liczby bliźniacze to :

```
3      5
5      7
11     13
17     19
```

#### Zadanie2 (1,5 punkt)

Napisać program obliczając:

- sumę dwucyfrowych liczb naturalnych, czyli :

$$suma = \sum_{i=10}^{99} i$$

- średnią dwucyfrowych liczb naturalnych (podziel sumę przez odpowiednią wartość)
- ilość liczb dwucyfrowych podzielnych przez liczbę wczytaną z klawiatury.

Program powinien wypisać:

Podaj liczbę, której podzielność będziesz badał=

Suma to:.....

Średnia to.....

Ilość liczb podzielnych przez..... to .....

Użyj pętli.

#### Zadanie3(1,5 punkt)

Wykonaj program rysujący linię pionową składającą się z podwójnej ilości gwiazdek na środku ekranu o wczytanej długości. Długość linii od 2 do 20.

np. dla  $dlu=5$

```
**
**
**
**
**
```

#### Zadanie4(1,5 punkt)

Narysuj figurę jak poniżej z gwiazdek o długości i wysokości  $liczba\_liter\_imienia+3$  użyciem dwóch pętli. Od miejsca ekranu (x,y) x i y wczytane z klawiatury jest to położenie pierwszej gwiazdki u góry po lewej stronie.

```
*****
*
*
*
*
*
```

**Zadanie5 (1,5 punkt)**

Napisać program piszący 20 razy nazwisko ucznia. Każde w nowej linii, i przesunięte dwie kolumnę w lewo wykorzystaj instrukcję FOR. Początek pisania taki aby 20 napisów zmieściło się na ekranie.  
np.

Kowalski  
Kowalski  
Kowalski

**Zadanie6 (1,5 punkt)**

Napisać program drukujący liczby, ich kwadraty i sześciany od numeru w dzienniku do numeru w dzienniku+15

Przykład wydruku zadania (wygląd musi być identyczny)

I=4	I*I=16	I*I*I=64
I=5	I*I=25	I*I*I=125
.....		
.....		

**ZADANIE 7 (1,5 punkt)**

Liczby pierwsze w postaci  $p$ ,  $p+2$ ,  $p+6$ ,  $p+8$  nazywa się czworaczkami (np.: 101,103,107,109...). Nie wiemy, czy jest ich skończenie, czy nie skończenie wiele. Napisz program znajdujący czworaczki.

---

# Część 6 tablice.

## TABLICE w C++.

**Tablica** jest to struktura złożona z elementów tego samego typu.

Elementy tablicy są skazywane przez **indeks** lub **zespół indeksów**. Określają one miejsce gdzie dany element znajduje się w tablicy.

---

tablica jednowymiarowa			
1	3	4	10
A[0]	A[1]	A[2]	A[3]

w komórce jest umieszczona A[0]=1 A[1]=3 A[2]=4 A[3]=10

tablica dwuwymiarowa					
A[0][0]	A[0][1]	A[0][2]	A[0][3]	A[0][4]	A[0][5]
3	1	2	10	6	5
1	3	4	20	7	9
A[1][0]	A[1][1]	A[1][2]	A[1][3]	A[1][4]	A[1][5]
Pierwsza kolumna					szósta kolumna

czyli w komórce jest np.

A[0][0]=3 A[1][4]=7 A[0][2]=2 A[1][5]=9

miejsce w tablicy określone jest przez podanie nazwy tablicy i w nawiasach kwadratowych podajemy wiersz potem po przecinku kolumnę.

### Deklaracja tablic:

Tablice wymagają przed użycie deklaracji komputer musi wiedzieć ile miejsca zarezerwować w pamięci i w jaki sposób rozmieścić kolejne elementy tablicy.

#### Przykład

```
int TAB[15];
```

*wytlumaczenie:*

należy zarezerwować 15 kolejnych komórek pamięci dla 15 liczb całkowitych typu int. Jednowymiarowa tablica (wektor) będzie się nazywać "TAB", a jej kolejne elementy zostaną ponumerowane przy pomocy indeksu.

Należy zwrócić uwagę, że w C++ zaczynamy liczyć numery komórek od zera a nie od jedynki.

Tablicz TAB wygląda następująco

TAB[0], TAB[1], TAB[2], TAB[3], .... TAB[14].

#### Przykład

```
float TAB_DWU[10][5];
```

Tablica TAB\_DWU jest tablicą dwuwymiarową i składa się z 50 elementów typu rzeczywistego (10 wierszy i 5 kolumn)

#### Przykład

Nadawanie wartości początkowych elementom tablicy. Wartości takie należy podawać w nawiasach klamrowych.

```
const int b[4]={ 1,2,33,444};
```



### *Wytłumaczenie:*

Elementom jednowymiarowej tablicy (wektora) b przypisano wartości: b[0]=1; b[1]=2; b[2]=33; b[3]=444;

### Przykład

Nadawanie wartości początkowych elementom tablicy dwuwymiarowej.

```
int TAB[2][3]={ { 1, 2, 3 }, {2, 4, 6 } };
```

### Przykład

Nadawanie wartości początkowych elementom tablicy znakowej.

```
char hej[5]="Ahoj";  
char hej[5]={ 'A', 'h', 'o', 'j' };
```

### Przykład

Tablica uzupełniona zerami przez domniemanie.

```
float T[2][3]={ { 1, 2.22 }, { .5 } };
```

kompilator uzupełni zerami wartości pozostałych komórek.

### Przykład

```
char D[ ]="Jakis napis"  
int A[ ][2]={ { 1,2 }, { 3,4 }, { 5,6 } }
```

Jeśli nawias kwadratowy zawierający wymiar pozostawimy pusty, to kompilator obliczy jego domniamaną zawartość w oparciu o podaną zawartość tablicy.

### **Przykład 24(23)**

#### Wykonaj:

- 1)Zapisz w zeszycie temat zadnia.
- 2)uruchom przykład
- 3)Zapisz w zeszycie siedem linijek wypisywania tablicy po jej wczytywaniu, czyli działanie drugiej pętli.

#### Temat :

Przykład demonstruje wczytywanie do tablicy siedmioelementowej jednowymiarowej oraz wypisanie wczytanych wartości.

```

#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    int a[7];
    for(int i=0;i<=6;i++)
    {
        cout<<"podaj A["<<i<<"]=";
        cin>>a[i];
    }
    cout<<"w tablicy A jest"<<"\n";
    for(int i=0;i<=6;i++)
    {
        cout<<"w komorce A["<<i<<"]="<<a[i]<<"\n";
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

### **ZADANIE 32g**

Napisać program rezerwujący tablicę jednowymiarową pięć elementów o nazwie t\_trzy\_litery\_nazwiska\_ucznia np. t\_kow. Do tablicy wpisz najmniejsze liczby pierwsze od wartości najmniejszej do największej. Użyj następującego przypisania wartości początkowych int ale[4]={4,2,0,-2};

### **ZADANIE 33**

Napisać program obliczający sumę oraz jego średnia arytmetyczną wczytanego ciągu do tablicy siedmioelementowej. Wykonaj schemat blokowy.

Wskazówka: Wykorzystaj treść przykładu poprzedniego oraz zadanie poprzednie.

### **ZADANIE 34**

Napisać program obliczający iloczyn wczytanego ciągu do tablicy pięcioelementowej.

Wykonaj schemat blokowy.

Wskazówka: Zastanów się, jaką wartość będzie miała zmienna ILOCZYN przed przystąpieniem do obliczania iloczynu.

### **ZADANIE 35**

Napisać program zliczający ilość wyrazów ciągu, które są większe od zera, równe zero, mniejsze od zera dla ciągu wczytanego do tablicy ośmioelementowej. Wykonaj schemat blokowy.

Wskazówka: Utwórz trzy liczniki L\_mniejsze, L\_wieksze, L\_rowne. Nadaj im wartości zero przed procesem zliczania. Powiększaj wartość odpowiedniego licznika o jeden po sprawdzeniu odpowiednich warunków, jaki spełnia każdy element wczytanego ciągu.

Przykładowy wygląd ekranu:

Liczb większych od zera jest=2

Liczb mniejszych od zera jest=4

Liczb większych od zera jest=2

### **Przykład 25**

Wykonaj:

- 1) Wpisz w zeszycie temat programu.
- 2) Narysuj w zeszycie macierz (tablicę) jednostkową o wymiarze [(liczba\_liter\_nazwiska) % 2] + 4.
- 3) Po uruchomieniu zapisz listing.

Temat: Program generujący macierz jednostkową 7\*7 ( siedem wierszy i siedem kolumn). Jako przykład użycia pęli podwójnej oraz tablicy dwuwymiarowej.

**Macierz jednostkowa** to tablica liczb, która po przekątnej ma jedynki a pozostałe liczby to zera.

gdy  $i = j$  to  $a[i][j] = 1$   
     $i <> j$  to  $a[i][j] = 0$

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

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    int a[7][7];
    for(int i=0; i<7; i++)
    {
        for(int j=0; j<7; j++)
        {
            if(i==j) {a[i][j]=1;}
            else
                a[i][j]=0;
        }
    }
    for(int i1=0; i1<7; i1++)
    {
        for(int j1=0; j1<7; j1++)
        {
            cout<<a[i1][j1]<<" ";
        }
        cout<<"\n";
    }

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

**ZADANIE 36**

Temat: Napisać program generowania następującej tablicy 8\*8. Do generowania tablicy użyj tablic i pętli.

```
9 1 1 1 1 .....
1 9 1 1 1 .....
1 1 9 1 1 .....
1 1 1 9 1 .....
1 1 1 1 9 .....
.....
```

Wydrukuj tablicę wierszami . Zapisz wzór do generowania.

### ZADANIE 37

Napisać program generowania tablicy 6\*6 która będzie wyglądać następująco :

```
1 0 0 0 0 0
1 1 0 0 0 0
1 1 1 0 0 0
1 1 1 1 0 0
1 1 1 1 1 0
1 1 1 1 1 0
1 1 1 1 1 1
```

Wydrukuj tablicę. Napisz wzór do generowania.

Wykonaj schemat blokowy.

### Przykład 26

Wykonaj:

- 1)Wpisz w zeszycie temat programu.
- 2)Po rozwiązaniu zapisz listing.
- 3)Narysuj schemat blokowy.

Temat: Program wczytujący macierz 3 na 3. Macierz wypisywana jest wierszami. Znajdowany jest element **maksymalny i minimalny**. Wypisywany jest element minimalny oraz maksymalny wraz z miejscem w tablicy.

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

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    int macierz[3][3]; // deklaracja tablicy
    // wczytywanie macierzy
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            cout << "podaj a" << "[" << i << "][" << j << "]=";
            cin >> macierz[i][j];
        }
    }
    cout << "\n"; // nowy wiersz
    for(int ii=0;ii<3;ii++)
    {
        for(int ji=0;ji<3;ji++)
        {
            cout << macierz[ii][ji] << "\t";
        }
        cout << "\n";
    }

    int max=macierz[0][0]; // nadanie wartości początkowej elementowi maksymalnemu
    int min=macierz[0][0]; // nadanie wartości początkowej elementowi minimalnemu
    int minx=0; // // nadanie wartości początkowej dla określenia miejsca w tabeli dla max i min
    int miny=0;
    int maxx=0;
    int maxy=0;
```

```
    for(int z=0;z<3;z++)
```

```

{
for(int x=0;x<3;x++)
{
if (macierz[z][x]<min)
{
min=macierz[z][x];
minx=z;
miny=x;
}
if (macierz[z][x]>max)
{
max=macierz[z][x];
maxx=z;
maxy=x;
}
}
}
cout << "max: a[" << maxx << "][" << maxy << "] = " << max
<< "\nmin: a[" << minx << "][" << miny << "] = " << min;
cout<<"\n";
system("PAUSE");
return EXIT_SUCCESS;
}

```

### **ZADANIE 38**

Napisz program wczytujący macierz 5 na 2. Podczas wczytywania wyświetlany jest indeks wczytywanego elementu. Macierz wypisywana jest wierszami. Do macierzy jednowymiarowej A o pięciu elementach zapisz średnie arytmetyczne wierszy. Znajdź element maksymalny i minimalny macierzy (tablicy) A. Wypisz element minimalny oraz maksymalny wraz z miejscem w tablicy. Przy rozwiązywaniu całego zadania używaj pętli.

### **ZADANIE 39**

Napisz program wczytujący macierz 3 na 4. Podczas wczytywania zapisany jest indeks wczytywanego elementu. Macierz po wczytaniu wypisywana jest wierszami. Znajdowana jest suma wszystkich elementów. Komputer pyta się o liczbę i następuje mnożenie macierzy przez liczbę. Po pomnożeniu wyświetlana jest tablica wynikowa. Przy rozwiązywaniu całego zadania używaj pętli.

### **ZADANIE 39a**

Napisz program obliczający wyznacznik 3x3 metodą Sarussa. Wczytaj macierz oraz wydrukuj ją po wczytaniu wierszami oraz wartość wyznacznika.

### **ZADANIE 40**

#### **Część teoretyczna zadania**

#### **Wykonaj:**

- a) zapisz w zeszycie, co to jest algorytm, wymień sposoby przedstawiania algorytmów, cechy charakterystyczne poprawnego algorytmu.
- b) zapisz w zeszycie, co to sortowanie, wymień metody sortowania oraz na czym polega metoda sortowania bąbelkowego.
- c) przeczytaj ze zrozumieniem metodę sortowania bąbelkowego  
( dla ciągu 1 2 9 7 10 8 4 przebiegi i krok → patrz poniżej)

#### **Teoria w celu rozwiązania zadania.**

**Algorytm** → jest to pewien ciąg czynności, który prowadzi do rozwiązania danego problemu.

**Algorytmy można przedstawiać m.in. następującymi sposobami:**

- słowny opis
- schemat blokowy
- lista kroków
- drzewo algorytmu

- drzewo wyrażeń
- w pseudojęzyk
- w język programowania.

### Cechy charakterystyczne poprawnego algorytmu:

1. **Poprawność** - dla każdego przypisanego zestawu danych, po wykonaniu skończonej liczby czynności, algorytm prowadzi do poprawnych wyników.
2. **Jednoznaczność** - w każdym przypadku zastosowania algorytmu dla tych samych danych otrzymamy ten sam wynik.
3. **Szczegółowość** - wykonawca algorytmu musi rozumieć opisane czynności i potrafić je wykonywać.
4. **Uniwersalność** - algorytm ma służyć rozwiązywaniu pewnej grupy zadań, a nie tylko jednego zadania. Przykładowo algorytm na rozwiązywanie równań w postaci  $ax + b = 0$  ma je rozwiązać dla dowolnych współczynników  $a$  i  $b$ , a nie tylko dla jednego konkretnego zadania, np.  $2x + 6 = 0$

**Sortowanie**—oznacza proces porządkowania według pewnego klucza np.: od największego do najmniejszego, najmniejszego do największego, alfabetycznie itp. Sortowanie może odbywać się według wielu różnych algorytmów, różniących się ilością wykonywanych operacji oraz szybkością działania.

### Metody sortowania.

#### Sortowanie bąbelkowe (angielskie bubble sort).

Jest to sortowanie polegające na przeglądaniu po kolei elementów porządkowanego ciągu i zamienianiu miejscami sąsiadujących elementów tak, aby spełniały relację porządkującą; w ten sposób elementy mniejsze ("lżejsze") przesuwają się na początek ciągu. Dla  $n$  elementów ciągu złożoność sortowania bąbelkowego wynosi  $O(n^2)$ . Sortowanie bąbelkowe jest sortowaniem stabilnym.

#### Sortowanie bąbelkowe z kresem górnym

Może się zdarzyć sytuacja, gdy tablica będzie już uporządkowana, zanim zrobimy  $n-1$  przebiegów lub po  $k$  przebiegach więcej niż  $k$  liczb od końca będzie na swoim miejscu. Ten problem także da się ominąć. Wystarczy do naszego programu wprowadzić zmienną, która będzie reprezentować w której pozycji nastąpiła ostatnia zamiana elementów tablicy. Ta zmienna to jakby kres następnego przebiegu. Ten wariant sortowania nazywa się sortowaniem bąbelkowym z kresem górnym.

#### Sortowanie przez wybór (angielskie selectsort)

Jest to sortowanie, w którym w każdym kroku algorytmu znajduje się najmniejszy element w sortowanym ciągu, po czym przenosi się ten element na kolejną pozycję do ciągu wynikowego (przez zamianę elementów miejscami).

#### Sortowanie przez wstawianie (insertion sort)

Sortowanie przez wstawienie to algorytm, którego czas działania wynosi  $O(n^2)$ . Jest on skuteczny dla małej ilości danych. Jest to jeden z prostszych i jeden z bardziej znanych algorytmów sortowania. Jest on stabilny i nie wymaga dodatkowej pamięci (działa w miejscu). Często stosujemy go podczas gry w karty, biorąc je ze stołu. Biorąc po jednej karcie ze stołu wstawiamy ją w odpowiednie miejsce do kart, które mamy w ręce.

#### Sortowanie przez zliczanie (counting sort)

Sortowanie przez zliczanie jest jednym z najszybszych algorytmów sortowania danych, a przy tym bardzo prostym do wytłumaczenia. Algorytm ten działa w czasie  $O(n)$ , tak więc jest to sortowanie w czasie liniowym. Mimo swoich zalet, sortowanie przez zliczanie ma swoje dwie poważne wady. Po pierwsze - do tego sortowania potrzebna jest dodatkowa pamięć (czyli nie jest to sortowanie w miejscu), a po drugie - tym sposobem można sortować tylko liczby całkowite z określonego przedziału. Niewiedomo dlaczego, ale algorytm ten mimo swojej prostoty nie jest powszechnie znany, a tym bardziej używany.

#### Sortowanie pozycyjne (radix sort)

Stosowane jest do sortowania elementów, które składają się z szeregu pozycji (mogą to być liczby, gdzie pozycjami są poszczególne cyfry; wyrazy - w tym przypadku są to poszczególne litery; mogą to także być inne dane, np. daty). Algorytm ten wymaga użycia innego algorytmu sortowania podczas swego działania, co ważne sortowanie to musi być stabilne. Gdy jako dodatkowego algorytmu sortowania użyjemy sortowania przez zliczanie to algorytm sortowania pozycyjnego działa w czasie  $O(n)$ . Sortowanie pozycyjne stosuje się nie tylko

do sortowania liczb, czy wyrazów. W ten sposób możemy także sortować np. daty. Musimy jednak pamiętać, aby sortować od pozycji najmniej znaczących. W przypadku dat - najpierw sortujemy je według dni, potem według miesięcy, a na końcu według lat. Sortowanie pozycyjne możemy także zastosować do sortowania rekordów baz danych. Na przykład chcemy posortować książkę telefoniczną według nazwisk, a w razie gdyby się one powtarzały to według imion, a w przypadku identyczności imion i nazwisk według numeru telefonu. Aby otrzymać taki wynik powinniśmy tę książkę telefoniczną posortować najpierw według numeru telefonu, potem według imion, a na końcu według nazwisk. Złożoność obliczeniowa takiego sortowania pozycyjnego na pewno nie będzie  $O(n)$ . Wynika to z tego, że do posortowania np. nazwisk trudno jest użyć sortowania przez zliczanie.

### Sortowanie wyrazów

Z sortowaniem wyrazów jest podobnie jak z sortowaniem liczb. W tym przypadku pozycjami są poszczególne litery danego wyrazu. Jednakże jest pewna różnica jeśli chodzi o sortowanie wyrazów różnej długości. W tym przypadku odpowiednią ilość znaków dopisujemy za wyrazem, a nie jak to było w przypadku liczb - przed. Znak ten musi być traktowany jako wyżej w alfabecie, niż wszystkie inne - może to być na przykład spacja.

### Sortowanie QuickSort

Jest to jeden z najpopularniejszych algorytmów sortowania. Wpłynęły na to dwie rzeczy. Po pierwsze jest ono bardzo szybkie (jak sama nazwa wskazuje), a po drugie - proste do wytłumaczenia i implementacji.

Pesymistyczny czas jego działania wynosi  $O(n^2)$ , a średni  $O(n \cdot \lg(n))$ . Mimo tego w praktyce jest to najszybszy algorytm sortowania dużych tablic danych.

Sortowanie szybkie opiera się na technice "dziel i zwyciężaj". Wejściowa tablica jest dzielona (po przestawieniu niektórych z jej elementów) na dwie mniejsze. Każdy element pierwszej tablicy nie jest większy niż każdy element drugiej tablicy. Liczbę, według której wykonuje się podziału to najczęściej pierwszy element tablicy. Następnie dla tych dwóch podtablic wywołany jest rekurencyjnie ten sam algorytm. Wywołania rekurencyjne kończą się aż któraś z kolejnych podtablic będzie zawierała tylko jeden element. QuickSort działa w miejscu.

Inna odmiana QuickSort

Wyżej wspomniane jest, że algorytm szybkiego sortowania ma pesymistyczny czas działania  $O(n^2)$ . Występuje to w przypadku, gdy tablica wejściowa jest posortowana odwrotnie, tzn. jej wyrazy stanowią ciąg nierosnący.

## Opis do algorytmu przedstawionego poniżej.

Metoda **bąbelkowa** polega: (porządkowanie od najmniejszego do największego—lub odwrotnie), ustawiamy się na pierwszym wyrazie ciągu i porównujemy go z drugim jeśli pierwszy jest większy od drugiego to zamieniamy je miejscami, następnie umieszczamy się na drugim i porównujemy z trzecim dokonując przestawienia lub nie. Porównując parami dochodzimy do końca ciągu (ustawiamy się na przedostatnim wyrazie). Jeśli podczas przechodzenia nastąpiła zmiana to musimy to zapamiętać jeśli nie to znaczy, że ciąg jest ustawiony prawidłowo. Po przejściu całego ciągu ustawiamy się ponownie na początek i sprawdzamy czy w poprzednim przechodzeniu było przestawienie jeśli nie było zamian to kończymy program jeśli było przestawienie to proces powtarzamy tyle razy aż nie będzie przestawienia.

Tablice uporządkować w porządku rosnącym.

9	2	7	10	8	4
---	---	---	----	---	---

przebieg porządkowania będzie następujący

#### pierwszy przebieg

zamienione elementy

krok 1 2 9 7 10 8 4

9 2

krok 2 2 7 9 10 8 4

9 7

krok 3 2 7 9 8 10 4

10 8

krok 4 2 7 9 8 4 10

10 4

#### drugi przebieg

zamienione elementy

krok 1 2 7 8 9 4 10

9 8

krok 2 2 7 8 4 9 10

9 4

#### trzeci przebieg

zamienione elementy

krok 1 2 7 4 8 9 10

8 4

#### czwarty przebieg

zamienione elementy

krok 1 2 4 7 8 9 10

7 4

w piątym przebiegu nie nastąpiła zmiana tzn., że można skończyć sortowanie.

### **Treść praktyczna zadania**

Temat: Dokonaj sortowania tablicy jednowymiarowej metodą bąbelkową z użyciem listy kroków, która jest zapisana poniżej.

#### Rozwiązanie:

- 1) nazwa klasy nazwisko\_ucznia\_z15a np. Kowalski\_z15a,
- 2) Przepisz do zeszytu listę kroków dla sortowania bąbelkowego.
- 3) Przepisz wzór do przestawiania elementów w ciągu (ten ze skrytką).
- 4) Utwórz tablicę d\_trzy\_pierwsze\_litery\_nazwiska np. d\_kow o siedmiu elementach. Nadaj elementom tablicy następujące wartości:  
d\_kow[0]= numer\_z\_dziennika;  
d\_kow[1]= liczba\_liter\_imienia;  
d\_kow[2]= liczba\_liter\_nazwiska;  
d\_kow[3]= -miesiąc\_urodzenia;  
d\_kow[4]= dzień\_urodzenia;  
d\_kow[5]= numer\_but  
d\_kow[6]= wag\_w\_kg
- 5) Wyświetl tablicę d\_kow
- 6) Dokonaj sortowania wg listy kroków  
Użyj zmiennej SKRYTKA\_trzy\_pierwsze\_litery. Użyj dwóch pętli i\_nazwisko\_ucznia, j\_nazwisko\_ucznia.
- 7) Wyświetl tablicę po sortowaniu.

Algorytmy mogą być przedstawiane z użyciem listy kroków. Poniższa lista kroków przedstawia sortowanie metodą bąbelkową.

- KROK1: Dla  $j = 0, 1, \dots, n-1$ : wykonaj KROK2  
KROK2: Dla  $i = 0, 1, \dots, n-1$ : jeśli  $d[i] > d[i+1]$ , to  $d[i] \leftarrow d[i+1]$   
KROK3: Zakończ algorytm.

#### Opis:

Zapis oznacza  $d[i] \leftarrow d[i+1]$ , że należy zamienić miejscami elementy  $d[i]$  i  $d[i+1]$ . Zamianę można uzyskać poprzez:

SKRYTKA =  $d[i]$ ;

$d[i] = d[i+1]$ ;

$d[i+1] = \text{SKRYTKA}$ ;

gdzie SKRYTKA jest zmienną pomocniczą a  $d[i]$  jest aktualnym elementem.

Pamiętaj, że pierwszy element tablicy to zero np.  $d[0]$ . Musisz uwzględnić ten fakt w konstrukcji pętli tak, aby zacząć od elementu zero czyli wartość początkowa zmiennej sterującej pętlą ma wartość zero. Musisz pomyśleć również jaka będzie ostatnia wartość pętli sterującej (w liście kroków jest to  $n-1$ ).

*Koniec zadania 40*



## **PRZYKŁADOWE ZADANIA NA SPRAWDZIAN**

### **ZADANIE 1**

Napisz program, który zarezerwuje tablicę 2\*4. Do tablicy wczytaj z klawiatury kolejne liczby naturalne wierszami. Wypisz zawartość tablicy po wczytaniu danych wierszami. Nazwa tablicy to Twoje inicjały (bez polskich liter).

### **ZADANIE 2**

Zarezerwuj tablicę:

- jednowymiarową,
- nazwie tab\_dwie\_pierwsze litery nazwiska ucznia,
- wymiar tablicy 8+miesiąc urodzenia,
- zawartości tablicy to pierwiastki z kolejnych liczb naturalnych

tab[0]=0

tab[1]=1

tab[2]=1.41

.....

czyli tab[i]=pierwastek(i)

### **ZADANIE 3**

Wygenerować następującą tablicę używając pętli podwójnej.

1 0 0 0 0

0 4 0 0 0

0 0 9 0 0

0 0 0 16 0

0 0 0 0 25

oraz wydrukować na ekranie wierszami.

### **ZADANIE 4**

Wczytać dwie macierze (tablice) A i B o wymiarze 2\*3. Wypisz wierszami macierze A B. Oblicz macierz C=2A-3B. Wypisz wierszami macierze C. Następnie program powinien zliczać ilość wyrazów macierzy (tablicy) C większych od zera, równych zero oraz mniejszych od zera.

### **ZADANIE 5**

Wczytaj dowolne dane do tablicy jednowymiarowej a <sub>i</sub> siedmio elementowej bez użycia pętli.		
Wypisz wczytaną tablicę w następujący sposób:	a[0]=2 a[1]=1 ..... a[6]=5	Oblicz następującą sumę $S = \sum_{i=3}^6 (2 * a_i - 2)$

### **ZADANIE 6**

Oblicz rozpiętość ciągu ośmio elemnetowego wczytanego z klawiatury do tablicy jednowymiarowej .Wypisz ten ciąg wyraz po wyrazie. Rozpiętość ciągu to różnica między elementem maksymalnym a minimalnym.



## Część 6

### **Funkcje, Struktury, Pliki, Operacje na tekstach, Wskaźniki, Generator losowy, Grafika.**

#### **ZADANIE →praca domowa**

Przepisz pytania wraz numerami pytań. Pod treścią pytań udziel odpowiedzi.

##### Pytanie1

Opisz funkcję poprzez uwzględnienie:

- cel stosowania
- skład funkcji
- zestaw parametrów
- treść funkcji
- instrukcja return →z przykładem funkcji o dwóch return
- wywołanie funkcji
- Zanotuj, na czym polega przekazywanie przez wartości oraz referencję.
- zmienne lokalne i globalne.

##### Pytanie2

Co to jest rekurencja?

Podaj definicję silni.

Podaj rekurencyjną definicję silni.

Podaj rekurencyjną definicję ciągu Fibonaciego. Jakie zastosowania może mieć ciąg Fibonaciego.

##### Pytanie3

Przepisz tabelę z poleceniami graficznymi. (patrz parę stron poniżej)

##### Pytanie4

- Podaj definicję wskaźnika.
- Jaka jest różnica pomiędzy odwoływaniem się do zmiennej poprzez podanie jej nazwy a poprzez wskaźnik?
- Zapisz deklarację wskaźnika, co oznacza gwiazdka przed nazwą zmiennej.

##### Pytanie5

Opisz:

- Inicjacja funkcji losowej.
- Losowanie liczby.

##### Pytanie6

Co to są struktury w CPP, jak są nazywane.

Co to jest rekord, jak jest zbudowany wykonaj przykładowy rysunek.

Dokonaj opisu teoretycznego tworzenia struktury.

##### Pytanie7

Odpowiedzi udziel na podstawie przykładów umieszczonych w instrukcji.

Jaki plik nagłówkowy jest konieczny abyś mógł korzystać z plików.

Podaj przykład tworzenia obiektu do zapisu pliku na dysku.

Podaj przykład tworzenia obiektu do odczytu pliku na dysku.

Podaj przykład sprawdzania czy plik został otwarty prawidłowo.(while.....)

Podaj przykład sprawdzania czy podczas odczytu z pliku nie ma błędów.

Podaj przykład wywołania metody zamykania pliku.

Podaj przykład sprawdzania czy jest koniec pliku. (while.....)

## Funkcje

### Definiowanie funkcji

#### Cel stosowania funkcji.

- Części programu zdefiniowane, jako funkcja może być wykorzystywana w wielu miejscach programu, co zmniejsza wielkość kodu całego programu.
- Program napisany z użyciem funkcji jest czytelniejszy łatwiejszy do analizowania.

#### Składnia funkcji.

```
typ_wyniku nazwa_funkcji(zestaw parametrów)
{
    treść funkcji;
}
```

*zestaw parametrów* → są nazwy obiektów przekazywanych do wnętrza funkcji wraz z ich typem, oddzielone przecinkami, parametry są w nawisach okrągłych, gdy funkcja nie ma parametrów to są puste nawiasy.

*treść funkcji* → to instrukcje i operacje wykonywane przez funkcję.

#### instrukcja powrotu return.

Służy do zakończenia wykonania zawierającej ją funkcji i może mieć postać:

```
return;
return stała;
return Wyrażenie;
return (wyrażenie);
```

Przykład:

Definiujemy funkcję dodaj() zwracającą, poprzez instrukcję return wartość przekazanego jej w momencie wywołania argumentu powiększoną o 5.

```
float dodaj(float x)
{
    x+=5;
    return x;
}
```

Funkcja dodaj() zwraca wartość i nadaje tę wartość zmiennej wynik zadeklarowanej na zewnątrz funkcji i znanej w programie głównym. A oto program w całości.

wewnątrz funkcji mogą być dwa polecenia return

np.

```
float wartosc_bezwzględna(float x)
{
    if (x>=0)
        return x;
    if (x<0)
        return -x;
}
```

### wywołanie funkcji:

wywołanie funkcji to napisanie jej nazwy łącznie z nawiasami (w których znajdują się parametry lub wartości, które mają przyjąć parametry).

uwagi:

- nie wolno definiować jednej funkcji wewnątrz innej funkcji
- nie możesz używać funkcji przed jej zdefiniowaniem
- funkcje powinny być (ze względu na styl programowania) definiowane poniżej funkcji głównej, gdy chcesz użyć funkcję przed jej zdefiniowaniem musisz przed funkcją główną napisać instrukcję nagłówka funkcji, która zdefiniujesz pod funkcją główną np.

int dodawanie(int a, int b);

- funkcja która nie przypisuje żadnej wartości jest typu pustego czyli **void** i nie musimy mieć również return

np.

```
void gwiazdki (void)
{
    cout<<"*****";
}
```

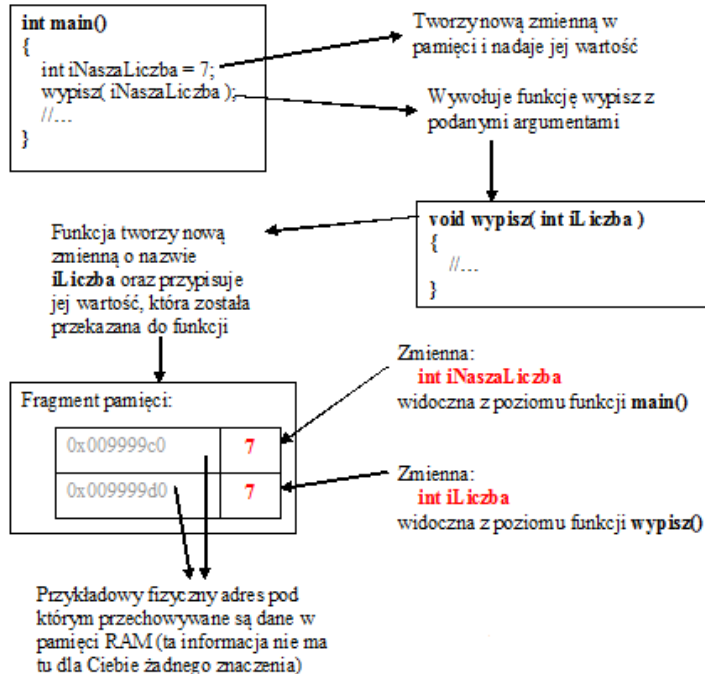
*Zmienne lokalne* → zmienne, które są definiowane wewnątrz funkcji, zmienne nie są widoczne poza funkcją.

*Zmienne globalne* → zmienne zdefiniowane poza funkcją main oraz poza innymi funkcjami są widoczne we wszystkich funkcjach programu.

## Przekazywanie argumentów funkcji przez *wartości* oraz przez *referencję*.

**Przekazywanie przez wartości** → W przypadku wywołania funkcji z przekazywaniem argumentu przez wartość, wykonywana jest kopia lokalna tego argumentu. Wywołana funkcja wykonuje swoje operacje na utworzonej kopii. Po zakończeniu działania funkcji, kopia lokalna jest niszczone.

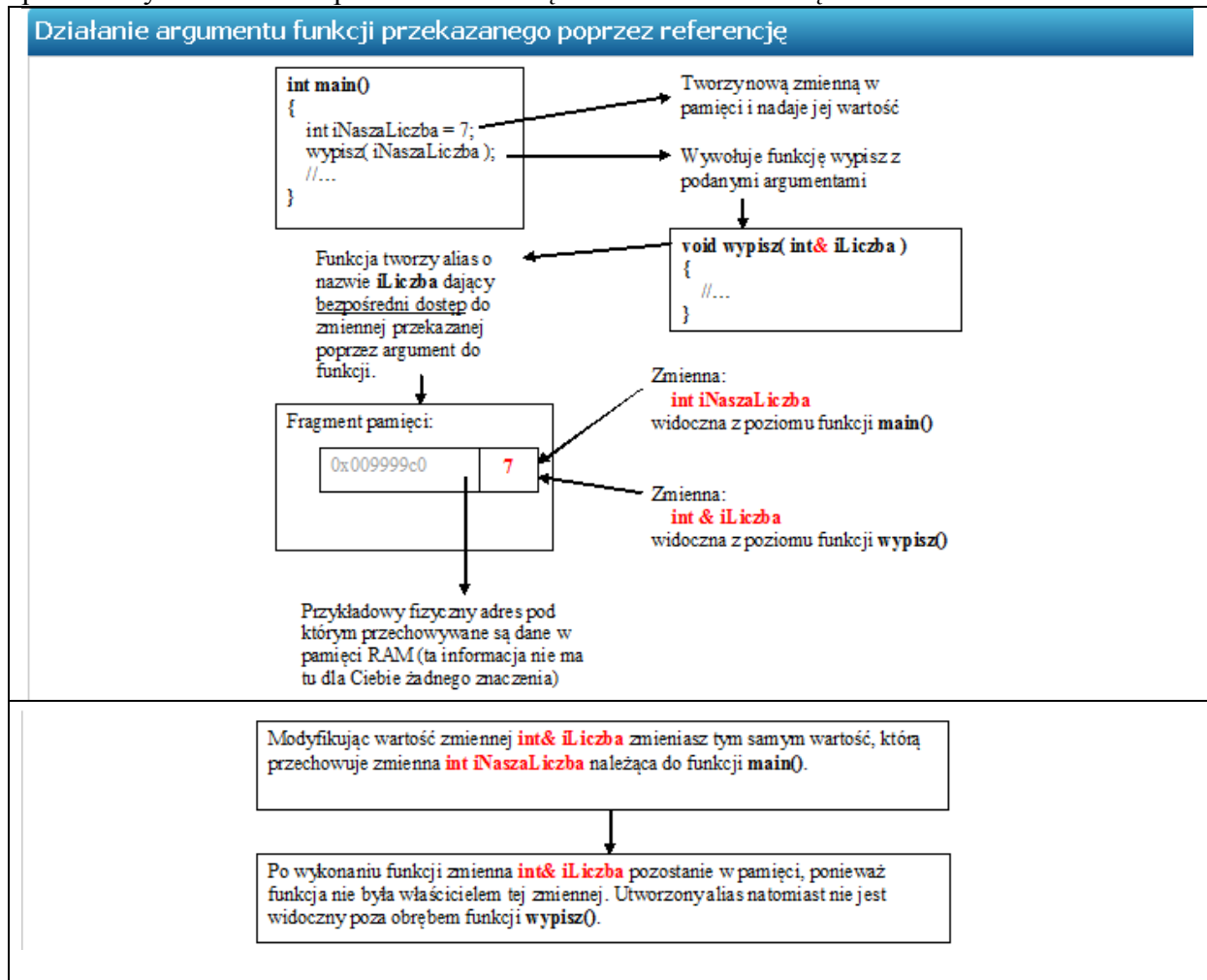
### Opis działania przekazywania wartości poprzez argument do funkcji



Zmienną **int iLiczba** można swobodnie modyfikować – zmiana jej wartości nie wpłynie na wartość, którą przechowuje zmienna **int iNaszaLiczba** należąca do funkcji **main()**.

Po wykonaniu funkcji zmienna **int iLiczba** zostanie usunięta z pamięci.

**Przekazywanie przez referencję** → za pomocą znaku & umieszczonym przed nazwą zmiennej odwołujemy się bezpośrednio do adresu w pamięci, pod którym zmienna jest przechowywana. W ten sposób zmienna będzie miała zmienioną wartość.



**Do czego jest potrzebne przekazywanie argumentów funkcji przez wartości oraz przez referencję?**

- 1) Przekazywanie parametrów przez wartość stosuje się wtedy, gdy funkcja ma na ich podstawie jedynie coś wyliczyć.
- 2) Przekazywanie parametrów przez referencję, czyli poprzez adres zmiennej stosujemy wtedy, gdy funkcja powinna coś w tej zmiennej umieścić, np. wynik obliczeń. W ten sposób pojedyncza funkcja może zwrócić wiele różnych wyników w kilku zmiennych, których adresy otrzymała na liście parametrów.
- 3) Referencja jest wygodna także wtedy, gdy przekazywany obiekt (zmienna) ma duży rozmiar.
- 4) Przekazanie przez wartość zawsze wymaga tworzenia kopii obiektu (zmiennej), którą otrzymuje funkcja do swojej wyłącznej dyspozycji. Przy dużych obiektach (zmiennych) prowadzi to do intensywnych operacji pamięciowych, spowalniając działanie programu.
- 5) Referencja natomiast wymaga jedynie przesłania adresu obiektu - zwykle 4 bajty. Jest zatem bardzo szybka. Jednakże z uwagi na bezpośredni dostęp do obiektu, programista musi być bardzo ostrożny, aby przypadkowo nie zmienić danych, których funkcja zmieniać nie powinna.



### **Przykład 27**

Wykonaj:

- Przepisz temat zadania.
- Przepisz uwagę.
- Wpisz i uruchom przykład.

Temat:

Przykład wypisuje gwiazdki z zużyciem zdefiniowanej funkcji. Funkcja nie ma, return (czyli nie zwraca swojej wartości) oraz nie ma parametrów.

Uwaga:

Gdy definiujemy funkcję pod funkcją główną (main) to musimy uprzedzić kompilator o tym, że funkcja będzie zdefiniowana później w tym przypadku poprzez:

void gwiazdki(void);

zapis ten musi być nad słowem kluczowym main.

### **Treść przykładu**

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
void gwiazdki(void); // uprzedzam kompilator ->funkcja gwiazdki będzie później
```

```
int main(int argc, char *argv[])
```

```
{
    gwiazdki();
    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
void gwiazdki (void)
```

```
{
    cout<<"*****";
}
```

### **ZADANIE 41**

Wykonaj:

- Przepisz temat zadania.
- Rozwiąż zadanie.
- Wpisz listing do zeszytu.

Temat: Napisać program z użyciem procedur wyświetlający na monitorze trzy prostokąty oraz trzy trójkąty konstruowane z znaków "\*".

Zdefiniuj procedury:

PROSTOKAT\_nazwisko\_ucznia

TROJKAT\_nazwisko\_ucznia

Następnie wywołaj trzy razy te procedury. Drukuj trójkąty i prostokąty naprzemian.

## **ZADANIE 42**

Wpisz dwa przykłady (Przykład pierwszy, Przykład drugi) jako osobne programy, które są umieszczone poniżej.

Zanotuj w zeszycie wyniki działania obu przykładów poprzez przepisanie wyniku działania (wygląd ekranu) podpisz, który wygląd, czego dotyczy.

### **Przykład pierwszy**

```
//przekazywanie przez wartości
#include <cstdlib>
#include <iostream>

using namespace std;
void przekaz(int x);
int main(int argc, char *argv[])
{
    int x=10;
    przekaz(x);
    cout<<"\n";
    cout<<"na zewnątrz funkcji x="<<x<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
void przekaz (int x)
{
    x = 2*x;
    cout<<"wewnątrz funkcji x="<<x<<endl;
}
```

### **Przykład drugi**

```
//przekazywanie przez referencję
#include <cstdlib>
#include <iostream>

using namespace std;
void przekaz(int &x);
int main(int argc, char *argv[])
{
    int x=10;
    przekaz(x);
    cout<<"\n";
    cout<<"na zewnątrz funkcji x="<<x<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
void przekaz (int &x)
{
    x = 2*x;
    cout<<"wewnątrz funkcji x="<<x<<endl;
}
```

### Przykład 28

Wykonaj:

- Przepisz temat zadania.
- Przepisz do zeszytu, co to jest tablicowanie funkcji oraz uzupełnij napisy „uzupełnij sam”
- Wpisz i uruchom przykład.

Temat: Tablicowanie funkcji

Tablicowanie funkcji polega na wypisaniu wartości funkcji dla określonego przedziału argumentów funkcji z zadany krok.

Np. dla  $y=x*x*x-2$  z krokiem 1 z przedziału  $\langle 2,10 \rangle$

$f(2)=6$

$f(3)=25$

$f(4)=62$

$f(\text{uzupełnij sam})=\text{uzupełni sam}$

.....

Treść przykładu.

Program tablicujący funkcję  $y=x*x$  od 5 do 15 z krokiem 0.1. Z wykorzystaniem definiowania funkcji.

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

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

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

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

```
//definiowanie funkcji
```

```
float f(float x)
```

```
{
```

```
    return x*x;
```

```
}
```

```
//rozpoczecie programu
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    float x=5;
```

```
    int z=0;
```

```
    for(;x<=15;x=x+0.1)
```

```
    {
```

```
        z++;
```

```
        cout.precision(3); //określenie miejsca po przecinku --> jedno
```

```
        cout<<"f("<<x<<"=";
```

```
        cout.precision(4);
```

```
        cout<<f(x)<<"\n";
```

```
        if (z%22==0) //wydruk 22 wiersze
```

```
        //poprzez sprawdzenie podzielności licznika wierszy --> z przez 22
```

```
        {
```

```

        system("PAUSE");
    }
}
system("PAUSE");
return EXIT_SUCCESS;
}

```

### **ZADANIE 43**

Ztablicować funkcję z krokiem 0.1 przy wykorzystaniu definiowania funkcji z zatrzymywaniem ekranu po 15 wyświetleniach. Nazwa funkcji f\_nazwisko\_ucznia, nazwa argumentu x\_nazwisko\_ucznia.

$$f(x) = \begin{cases} x^2 - 12x + 36 & \text{dla } < 5, 6 > \\ 1 & \text{dla } < \frac{\pi}{4}, 5 > \\ \operatorname{tg}(x) & \text{dla } < 0, \frac{\pi}{4} > \end{cases}$$

### **Przykład 29**

Napisać program obliczający pole powierzchni całkowitej oraz objętość walca. W tym celu zdefiniuj funkcje Pc oraz V ze zmiennymi H i R. Oblicz pole oraz objętość dla

- R=5 oraz H=10 (ściśle określone)
- dla dowolnych wczytanych R i H wczytanych z klawiatury

```

#include <cstdlib>
#include <iostream>
#include <math.h>
#define H 10.0
#define R 5.0

```

```

float v(float h, float r)
{
    return M_PI*r*r*h;
}
float Pc(float h, float r)
{
    return (2*M_PI*r*r+2*M_PI*r*h);
}

```

```
using namespace std;
```

```

int main(int argc, char *argv[])
{
    cout<<"H="<<H<<"cm\tr="<<R<<"cm\n";
    cout<<"V("<<H<<","<<R<<")=\t"<<v(H,R)<<"cm^3\n";
    cout<<"Pc("<<H<<","<<R<<")=\t"<<Pc(H,R)<<"cm^2\n";
    float h,r;
}

```

```

cout<<"Podaj r=";
cin>>r;
cout<<"Podaj H=";
cin>>h;
cout<<"V("<<h<<","<<r<<")=\t"<<v(h,r)<<"cm^3\n";
cout<<"Pc("<<h<<","<<r<<")=\t"<<Pc(h,r)<<"cm^2\n";
system("PAUSE");
return EXIT_SUCCESS;
}

```

#### **ZADANIE 44**

Napisać program obliczający pole powierzchni oraz obwód prostokąta.

Zdefiniuj funkcje:

POW\_nazwisko\_ucznia

OBW\_nazwisko\_ucznia.

Wywołaj funkcje POW oraz OBW z A=4 i B=5. Wypisz na ekranie wyniki w odpowiednim formacie oraz z jednostkami. Napisz tak program ,aby komputer pytał się o A i B następnie liczył POW i OBW dla różnych A i B. Wypisz na ekranie wyniki w odpowiednim formacie oraz z jednostkami.

### **Przykład 29a**

Temat: Funkcja typu bool określająca czy wczytana liczba jest pierwsza.

#### Algorytm:

Określamy ilość podzielników wczytanej liczby. Jeśli ilość podzielników jest 2 to liczba jest pierwsza wtedy funkcja przyjmuje wartość true w przeciwny przypadku to false.

```
#include <cstdlib>
#include <iostream>

using namespace std;
bool pierwsza(int liczba)
{
    int ilosc=0;
    for (int i=1;i<=liczba;i++)
    {
        if(liczba%i==0)
        {ilosc++;}
    }
    if (ilosc==2)
    {return true;}
    else
    {return false;}
}
int main(int argc, char *argv[])
{
    int liczba;
    cout<<"podaj liczbe=";
    cin>>liczba;
    if (pierwsza(liczba)==true)
    {cout<<"liczba jest pierwsza";}
    else
    {cout<<"liczba nie jest pierwsza";}
    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### **ZADANIE 45a**

Temat: Program wykorzystujący funkcję z poprzedniego zadania do znajdowania bliźniaczych liczb pierwszych w przedziale od <1,max> gdzie max jest liczba wczytną do program.

Bliźniacze liczby pierwsze to liczby pierwsze różniące się o dwa np. 11 13.

## Rekurencja

### REKURENCJA

Rekurencja zachodzi wtedy, gdy jakaś funkcja wywołuje samą siebie.

### Sinia definicja

W matematyce definiowana jest pojęcie silni np.  $n!$

Def:

$0! = 1$

$1! = 1$

.....

$5! = 1 * 2 * 3 * 4 * 5 = 120$

### PRZYKŁAD 30

Obliczyć wartość silni stosując rekurencję. Silnia w sposób rekurencyjny jest definiowana:

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

Wykonaj:

a) zapisz definicję rekurencyjną silni

b) sprawdź i zapisz w zeszycie do jakiej wielkości można obliczać silnię z użyciem tego przykładu.

c) wpisz przykład do zeszytu.

```
#include <cstdlib>
#include <iostream>

double silnia(int n);

using namespace std;

int main(int argc, char *argv[])
{
    int n;
    double wynik;
    cout<<"Obliczam silnie z liczby naturalnej"<<endl;
    cout<<"Podaj liczbe n=";
    cin>>n;
    wynik=silnia(n);
    cout<<"silnia("<<n<<"")="<<wynik;
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

double silnia(int n)
{
    if (n==0)
    {
        return 1;
    }
    else
    {
        return n*silnia(n-1);
    }
}
```

### Zadanie 46

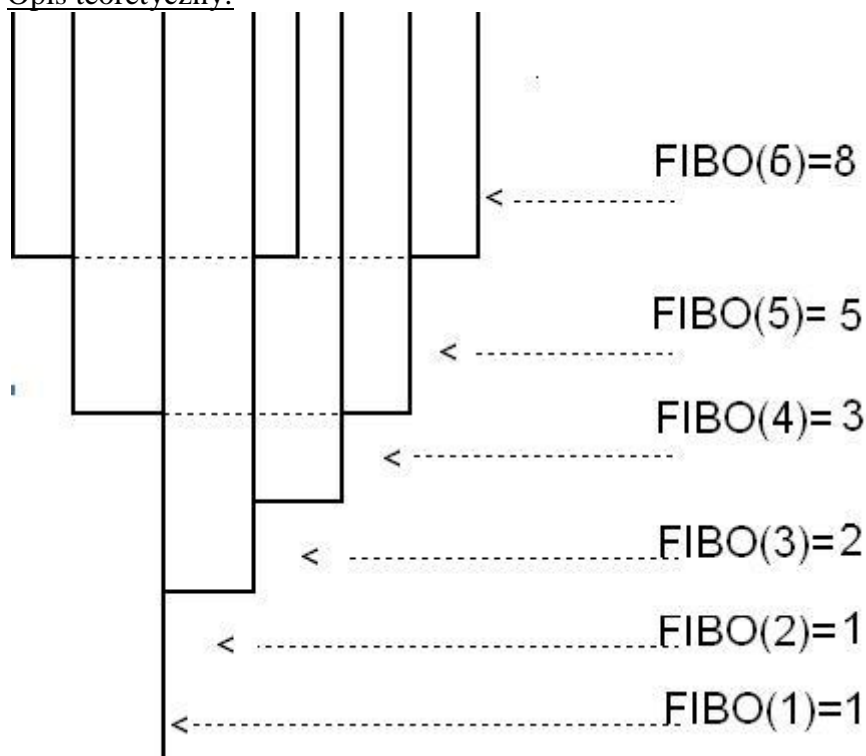
Napisz program rekurencyjnego znajdowania wartości ciągu Fibonaciego. Który element mamy obliczyć wczytujemy z klawiatury.

Przykładowy wygląd ekranu:

Podaj, który element ciągu Fibonaciego, który należy obliczyć=6

FIBO(6)=8

Opis teoretyczny:



Liczby Fibonacciego FIBO(N) pojawiają się przy opisie ilościowym niektórych zjawisk przyrodniczych. Jeśli rozrastanie roślin odbywa się zgodnie z zasadą : każdy pęd wypuściwszy pęd boczny , przez rok odpoczywa i dopiero w następnym roku puszcza nowy pęd, to liczba pędów w N-tym roku może być obliczona zgodnie z wcześniej podanym ciągiem rekurencyjnym FIBO.

$$F_n = \begin{cases} 0 & \text{dla } n = 1 ; \\ 1 & \text{dla } n = 2 ; \\ F_{n-1} + F_{n-2} & \text{dla } n > 2 . \end{cases}$$

### ZADANIE 45aa

W podpunktach a) b) c) nazwa funkcji to CIAG\_nazwisko\_ucznia np. CIAG\_kowalski  
a)

Dany jest ciąg liczbowy  $a_1=3$   $a_2=7$   $a_3=16$   $a_4=32$   $a_5=57$   $a_6=93$ ..

Napisać w zeszycie wzór rekurencyjny obliczający n-ty wyraz ciągu. Napisz na podstawie wzoru rekurencyjnego program obliczający n-ty wyraz ciągu. Oblicz 17-ty wyraz tego ciągu. Zapisz program w zeszycie.

### ZADANIE 45ab

b)

Dany jest ciąg liczbowy  $a_1=2$   $a_2=7$   $a_3=12$   $a_4=17$   $a_5=22$   $a_6=27$ ..



Napisać w zeszycie wzór rekurencyjny obliczający  $n$ -ty wyraz ciągu. Napisz na podstawie wzoru rekurencyjnego program obliczający  $n$ -ty wyraz ciągu. Oblicz 20-ty wyraz tego ciągu. Zapisz funkcję rekurencyjną w zeszycie.

#### **ZADANIE 45ac**

c)

Dany jest ciąg liczbowy  $a_1=1$   $a_2=9$   $a_3=36$   $a_4=100$   $a_5=225$  ..

Napisać w zeszycie wzór rekurencyjny obliczający  $n$ -ty wyraz ciągu. Napisz na podstawie wzoru rekurencyjnego program obliczający  $n$ -ty wyraz ciągu. Oblicz 7-ty wyraz tego ciągu. Zapisz funkcję rekurencyjną w zeszycie.

#### **Zadanie 45b**

Temat: Iteracja a rekurencja dwa rozwiązania tego samego problemu.

#### **Specyfikacja problemu**

Dane wejściowe: Dowolna liczba naturalna  $n$ ,  $n \geq 1$ .

Dane wyjściowe: wartość  $n$ -tego wyrazu ciągu  $a_n$ .

Dany jest algorytm w postaci **listy kroków** dotyczący ciągów.

**Krok 1:** Ustaw wartości zmiennych  $roznica=1$  i  $pocz=1$ .

**Krok 2:** Dla  $i=1, 2, \dots, n-2, n-1$  wykonaj krok 3.

**Krok 3:** Zmienną  $roznica$  zwiększ dwukrotnie.

**Krok 4:** Zwróć  $a_n=pocz+roznica$ .

Wykonaj:

1) tabelę symulacji listy kroków.

n	pocz	Roznica	i	$a_n$
6	1	1	1	-
...	...	...	...	...

2) Wykonaj schemat blokowy

3) Napisz program z użyciem pętli `do...while`.

4) Napisz program z użyciem pętli `for....`

5) zapisz wartości  $a_1, a_2, a_3, a_4, a_5, a_6$ .

6) Podaj wzór na  $a_n=f(n)$ .

7) Napisz program obliczający wyraz  $a_n$  korzystający we wzoru z punktu 5

8) Zaproponuj wzór rekurencyjny dla rozwiązania problemu

9) Napisz rekurencyjne rozwiązanie problemu

## Struktury

**Struktury** są to typy definiowane przez użytkownika. Możemy zgrupować informacje różnych typów w jednej zmiennej, czyli zbierają one w całość informacje różnych typów. Tak struktura nazywana jest rekordami.

**REKORD** – jest strukturą składającą się ze stałej liczby składników, nazywanych **polami**. Pola – mogą być różnych typów, każde z nich ma **nazwę-identyfikator**, który używany jest do jego wybierania.

NAZWISKO	IMIE	WZROST	WAGA
----------	------	--------	------

}  
R  
E  
K  
O  
R  
D

NAZWISKO IMIE WZROST WAGA → to są pola o takich nazwach (identyfikatorach)

### Opis teoretyczne tworzenia struktury.

struct nazwa typu

```
{  
    typ_pola_1 nazwa_pola_1;  
    typ_pola_1 nazwa_pola_2;  
    .....  
    typ_pola_1 nazwa_pola_n;  
}; // musi być średnik
```

### Przykład 31

Definiuje rekord o polach nazwisko, imię, wiek, waga. Wczytuje się dane do dwóch rekordów oraz wypisuje rekordy w postaci tabelarycznej.

- Wpisz i uruchom przykład.
- Wpisz listing do zeszytu.
- Wpisz do zeszytu definicję struktury.
- Wpisz do zeszytu definicję rekordu wraz z rysunkiem
- Wpisz do zeszytu Opis teoretyczne tworzenia struktury.

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

```
using namespace std;
```

```
struct daneosobowe //początek definiowania rekordu  
{  
    char    nazwisko[20];  
    char    imie[15];
```

```

        int        wiek;
        float      waga;
                // nazwa pola   typ pola
    };           //koniec definiowania recordu
int main(int argc, char *argv[])
{
    daneosobowe osoba1,osoba2;
        //zadeklarowanie danych o dwóch osobach jako recordy
    cout<<"podaj dane o pierwszej osobie"<<endl;
    cin>>osoba1.nazwisko;
    cin>>osoba1.imie;
    cin>>osoba1.wiek;
    cin>>osoba1.waga;
    cout<<"podaj dane o drugiej osobie"<<endl;
    cin>>osoba2.nazwisko;
    cin>>osoba2.imie;
    cin>>osoba2.wiek;
    cin>>osoba2.waga;
    cout<<"posiadamy informacje o następujących osobach"<<endl;
    cout<<"                osoba pierwsza"<<endl;
    cout.width(15);cout<<"nazwisko";
    cout.width(15);cout<<"imie";
    cout.width(15);cout<<"wiek";
    cout.width(15);cout<<"waga"<<endl;
    cout.width(15);cout<<osoba1.nazwisko;
    cout.width(15);cout<<osoba1.imie;
    cout.width(15);cout<<osoba1.wiek;
    cout.precision(6);cout.width(15);cout<<osoba1.waga<<endl;
    cout<<"                osoba druga"<<endl;
    cout.width(15);cout<<osoba2.nazwisko;
    cout.width(15);cout<<osoba2.imie;
    cout.width(15);cout<<osoba2.wiek;
    cout.precision(6);cout.width(15);cout<<osoba2.waga<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

#### **ZADANIE 45c**

Napisz program do tworzenia bazy danych. Baza ta będzie przechowywać nazwę produktu oraz jego cenę. Program wczytuje cztery rekordy oraz wyświetla ich zawartość w postaci tabeli (bez ramek → czyli równe kolumny, niezależnie od wielkości/długości danych). Nazwy pól rekordów, z trzema literami nazwiska ucznia. Np. waga\_kow

#### **ZADANIE 47**

Napisać (program obliczający długość wektora w przestrzeni. Zadeklarować (record o nazwie PUNKT, który będzie miał trzy pola X – pierwsza współrzędna punktu Y,Z –, następne współrzędne punktu. Następnie zadeklarować dwa punkty w przestrzeni o nazwach W1 W2. Następnie obliczyć długość wektora operując na polach recordów.

### ZADANIE 46b

Napisać program na dodawanie, odejmowanie, mnożenie dzielenie liczb zespolonych. Działania te powinny być procedurami. Procedurą powinno być również wczytywanie i wyprowadzanie danych. Procedura wyprowadzania danych powinna być tak wykonana, że są zapisane liczby w postaci  $ZW=a+j*b$ . Program powinien prezentować możliwe działania do wykonania. Wybór działania zadanego przez użytkownika powinien odbywać się za pomocą instrukcji CASE. Zapisz dane liczby zespolone jako Z1 i Z2 ,wynik jako ZW. Z1,Z2,ZW zdefiniowane jako record o polach R,U. R– część rzeczywista U– część urojona.

#### Podstawy teoretyczne.

Liczba zespolona może być przedstawiona w postaciach:

- algebraicznej
- trygonometrycznej
- wykładniczej

W zadaniu korzystać będziemy z postaci algebraicznej

Gdy mamy dwie liczby zespolone

$Z1=a1+j*b1$  np.  $Z1=2+3*j$

oraz

$Z2=a2+j*b2$  np.  $Z2=4+j*5$  to

$a1=2$   $b1=3$   $a2=4$   $b2=5$ .

gdzie:

$j = \sqrt{-1}$ czyli $j*j = -1$
----------------------------------

*dodawanie:*

$Z1+Z2=(a1+j*b1) + (a2+j*b2)=a1+a2 +j*(b1+b2) = aw + j*bw$

np.  $Z1+Z2=(2+3*j)+(4+5*j)=2+4+(3+5)*j=6+j*8$

*odejmowanie:*

$Z1-Z2=(a1+j*b1) - (a2+j*b2)=a1-a2 +j*(b1-b2) = aw + j*bw$

np.  $Z1-Z2=(2+3*j)-(4+5*j)=2-4+(3-5)*j=-2-j*2$

*mnożenie*

$Z1*Z2=(a1+j*b1) * (a2+j*b2)=a1*a2-b1*b2 + j(b2*a1+b1*a2)$  np.

$Z1*Z2=(2+3*j)*(4+5*j)=2*4-3*5 +j(3*4+2*5)=-7+j*22$

*dzielenie*

$Z1/Z2=(a1+j*b1)/(a2+j*b2)=(a1*a2+b1*b2)/(a2*a2+b2*b2)+$

$+j*(b1*a2-b2*a1)/(a2*a2+b2*b2)$

Gdy  $Z2=0$  czyli  $a2=0$  oraz  $b2=0$  to działanie  $Z1/Z2$  jest niewykonalne. Program powinien uwzględnić ten przypadek.

## PRZYKŁADOWE ZADANIA NA SPRAWDZIAN

### ZADANIE1

Narysować w trybie tekstowym postać człowieka przy wykorzystaniu procedur:

–GŁOWA

–RECE

–TULW

–NOGI

wywołać procedury tak aby powstała postać człowieka

### ZADANIE2

Ztablicować funkcję z krokiem 0.3 przy wykorzystaniu definiowania funkcji.

$$f(x) = \begin{cases} x^2 - 4 & \text{dla } < 5, 3\pi > \\ 1 - x^3 & \text{dla } (\pi, 5) \\ \operatorname{tg}(x) + \operatorname{ctg}(x) & \text{dla } < 0, \pi > \end{cases}$$

### ZADANIE 3

Za pomocą RECORDÓW zapisać dane o czterech uczniach. Dane obejmują nazwisko imię oraz oceny z czterech przedmiotów. Wydrukować dane w postaci zestawienia (bez ramki).

	PRZEDMIOT1	PRZEDMIOT2	PRZEDMIOT3	PRZEDMIOT4
UCZEN1	?	?	?	?
UCZEN2	?	?	?	?
UCZEN3	?	?	?	?
UCZEN4	?	?	?	?

Wyrównywanie danych w tabelach wykonaj bez użycia spacji.

### ZADANIE 4

Napisać program podnoszenia liczby zespolonej Z1 do kwadratu. Do definiowania liczb zespolonych użyj struktur. Działanie to powinny być funkcją. Funkcją powinno być również wczytywanie i wyprowadzanie danych.

### ZADANIE 5

Napisz program wykorzystujący funkcję o nazwie pierwsza\_nazwisko, gdzie argumentem jest sprawdzana liczba a wynikiem wartość prawda lub fałsz, w zależności czy liczba jest pierwsza. Program działa w przedziale <1,max> gdzie max jest liczbą wczytaną do program. Program wypisuje liczby pierwsze z przedziału <1,max>, które przy dzieleniu przez 3 dają resztę 1.

### ZADANIE 6

a) Nazwa funkcji rekurencyjnej to CIAG\_nazwisko\_ucznia np. CIAG\_kowalski Dany jest ciąg liczbowy a1=3 a2=7 a3=11 a4=15 a5=19 a6=23.. Napisz program stosujący wzór rekurencyjny (definicja funkcji rekurencyjnej) i obliczający n–ty wyraz ciągu. Oblicz 17–ty wyraz tego ciągu. b) Dany jest ciąg  $a_n = (n+1)^2$ . Metodą różnicową w zeszycie ustal wzór rekurencyjny dla tego ciągu. Nazwa funkcji rekurencyjnej to REK\_nazwisko\_ucznia np. REK\_kowalski. Napisz program stosujący wzór rekurencyjny (definicja funkcji rekurencyjnej) i obliczający n–ty wyraz ciągu. Oblicz 5–ty wyraz tego ciągu.

## Operacje na plikach

### Korzystanie z plików.

#### 1) Plik nagłówkowy

```
#include <fstream>
```

#### 2) Uchwyt do obsługi pliku

Utworzenie obiektu, który umożliwi nam komunikację ze wskazanym przez nas plikiem. Obiekt tworzy podobnie jak zmienne. Typem danych będzie **ifstream**, natomiast nazwa zmiennej może być dowolna ( w ramach dozwolonych nazw) np.

```
ifstream plik_wejscowy;
```

Przez utworzenie uchwytu będziemy rozumieli utworzenie możliwości komunikacji z plikiem i tak teraz za pomocą zmiennej plik\_wejscowy możemy komunikować się z dowolnym plikiem znajdującym się na dysku.

#### Uwaga:

Klasa ifstream posiada tylko i wyłącznie metody umożliwiające odczytywanie zawartości pliku ( nic nie zapiszesz).

#### Uwaga:

Zmienna za pomocą której można dostać się do danych pliku zazwyczaj nazywana jest **uchwytem do pliku**.

#### 3) Otwieranie wybranego pliku

Po utworzony **uchwyt do pliku**, należy wskazać jaki plik na dysku chcielibyśmy otworzyć do odczytu. Czynność tą wykonuje się przy pomocy metody:

```
nazwa_pliku_uchwytu. open(ścieżka do pliku wraz z nazwą pliku)
```

```
np.   zapis_na_dysku.open("plik_out.txt");           // otwarcie pliku tekstowego
```

#### Uwaga:

Jeżeli plik będzie istniał na dysku oraz nie będzie on zablokowany do odczytu przez inną aplikację, to wówczas otwarcie pliku zakończy się powodzeniem.

**Ścieżka względna** to taka, która nie zawiera pełnej ścieżki do pliku, tj. nie rozpoczyna się ona od litery (czy też nazwy) dysku.

Ścieżką względną będzie:

- podanie samej nazwy pliku (wraz z jego rozszerzeniem),
- podanie ścieżki względem katalogu roboczego aplikacji.

**Ścieżka bezwzględna** określa natomiast pełną ścieżkę do pliku i zaczyna się od litery (nazwy) dysku, a kończy się na pełnej nazwie pliku.

#### Uwaga

#### Ślashe i backslashe w ścieżce do pliku

Lepszym rozwiązaniem jest podawanie w ścieżkach **slashes (znak /)** zamiast **backslashes (znak \)**. Ślashe jak również backslashe są traktowane jako znaki równoważne w ścieżkach do plików.

#### 4) Poprawność otwarcia pliku

Otworzenie pliku może zakończyć się powodzeniem jak i niepowodzeniem.  
Do stwierdzenia poprawności otwarcia pliku używa się metodę **good**, należącej do klasy `ifstream`.

np.

```
if( plik_do_otwarcia.good() )
{
    cout<<"plik udało się otworzyć";
}
else
{
    cout<<"otwarcie pliku się nie powiodło";
}
```

#### 5)Odczytywanie tekstu z pliku z użyciem **getline**.

**getline**→ wczytuje jeden wiersz z pliku.

np.

```
string odczytana_linia;
getline( plik, odczytana_linia );
```

wczytana linia zostanie zapisana do zmiennej **odczytana\_linia**.

Uwaga;

Niepowodzenia przy wczytywaniu z pliku

- Wczytanie kolejnego wiersza tekstu zakończy się niepowodzeniem, gdy w pliku nie będzie więcej tekstu do odczytania.
- Odczyt może zakończyć się również niepowodzeniem w wyniku innych czynników, takich jak np. awaria urządzenia (np. wysunięto płytę CD, z której odczytywaliśmy zawartość pliku).

Uwaga

Funkcja `getline` zwraca wartość logiczną **true** w przypadku sukcesu, natomiast **false** w przypadku niepowodzenia.

np.

```
string wiersz;
for( bool bWczytano = getline( plik, wiersz ); bWczytano; bWczytano = getline( plik, wiersz ) )
{
    cout << wiersz << endl;
}
```

#### 6)Odczytywanie tekstu z pliku z użyciem **strumienia >>**.

Dane w pliku mogą być zapisane w postaci liczb, liczby te powinny być do zmiennych liczbowych.

a)otwarcie pliku.

```
np.  
ifstream plik_dane;  
plik.open( "plik_dane.txt" );
```

b) odczytanie danych.

**Operator >>** umożliwia wczytywanie danych do zmiennych w taki sam sposób, w jaki wczytywaliśmy dane wprowadzane przy pomocy klawiatury.

```
np.  
int zmienna;  
float zmienna1;  
string zmienna2;  
char zmienna3;  
plik_dane3 >> zmienna;  
plik_dane >> zmienna;  
plik_dane1 >> zmienna1;  
plik_dane2 >> zmienna2;
```

Uwaga

Jeżeli zmienna będzie typu char to wówczas odczytany zostanie jeden znak, który nie jest białym znakiem.

Uwaga

Pamiętaj, że operator >> zawsze pomija wszelkie napotkane białe znaki.

Białymi znakami są nazywane te symbole, które są używane w tekście i nie posiadają swojej reprezentacji graficznej (nie są wyświetlane). Przykładem takiego znaku jest spacja, tabulacja czy też znak przejścia do nowej linii (enter).

### 7) Zamykanie otwartego pliku

Każdy otwarty plik należy zamykać zaraz po zakończeniu z nim pracy.

Do tego celu służy **metoda close**, której wywołanie wyglądać może następująco:

```
plik_otwarty.close();
```

### 8) Metoda ifstream::eof

Metoda eof zwraca prawdę, jeżeli ostatnio wykonana operacja odczytu danych została zakończona z powodu osiągnięcia końca pliku. Informację tą bardzo często wykorzystuje się wtedy, gdy chcemy odczytać zawartość całego pliku nie wiedząc ile danych się w nim znajduje.

```
while( !plik.eof() )  
{  
    string s_Wiersz;  
    getline( plik, s_Wiersz ); // odczytujemy wiersz z pliku  
}
```

### 9) Metoda ifstream::bad

Zwraca ona prawdę, jeżeli ostatnio wykonana operacja odczytu danych zakończy się niepowodzeniem z powodu wystąpienia błędu sprzętowego. Przez błąd sprzętowy należy rozumieć pojawienie się badsectorów na dysku, niedostępność urządzenia na którym



znajdował się otwarty plik (np. pendrive lub dysk sieciowy) lub zabranie dostępu do pliku przez inny proces.

np.

```
bool odczytajPlik(string sNazwaPliku )
{
    ifstream plik;
    plik.open( sNazwaPliku.c_str() );
    if( !plik.good() )
        return false; //Nie udało się otworzyć pliku

    while( !plik.eof() )
    {
        string sWiersz;
        getline( plik, sWiersz );
        if( plik.bad() ) //podczas próby odczytania danych wystąpił błąd sprzętowy
        {
            plik.close();
            return false; //wychodzimy z funkcji i informujemy, że odczytanie pliku zakończyło
            się niepowodzeniem
        }
        cout << sWiersz << endl;
    }
    plik.close();
    return true;
}
```

#### 10)Metoda ifstream::fail

Metoda fail zwróci prawdę, gdy odczytanie danych zakończy się niepowodzeniem z powodu wystąpienia błędu sprzętowego lub z powodu błędu logicznego jaki miał miejsce podczas odczytu danych. Przez błąd logiczny odczytu danych należy rozumieć sytuację, w której aplikacja została zaprogramowana tak, aby odczytywała liczby, natomiast w pliku znalazły się również inne znaki np. nieoczekiwane litery alfabetu bądź znaki specjalne.

np.

```
int iLiczba;
plik >> iLiczba;
if( plik.fail() )
    cout << "Nie udało sie wczytac liczby!" << endl;
```

#### 11)Metoda ifstream::good

Metoda good zwraca prawdę wtedy, gdy strumień danych jest w prawidłowym stanie tj. żaden błąd nie wystąpił podczas pracy z plikiem. Wspomnianą metodę warto wykorzystywać przede wszystkim do sprawdzania, czy otwarcie pliku zakończyło się powodzeniem.

np.

```
ifstream plik;
plik.open( "plik.txt" );
```

```

if( plik.good() )
    cout << "Plik zostal otwarty" << endl;
else
    cout << "Nie udalo sie otworzyc pliku" << endl;

```

## 12)Metoda ifstream::clear

Funkcje jak również metody do odczytu danych z pliku działają tylko wtedy, gdy strumień danych jest w poprawnym stanie. Jeżeli w trakcie pracy z plikiem wystąpił jakikolwiek błąd (tj. napotkano koniec pliku, wystąpił błąd logiczny lub sprzętowy, albo nie udało się otworzyć pliku do odczytu), to wówczas kontynuowanie pracy z obiektem std::ifstream nie będzie możliwe. Aby móc wznowić pracę ze strumieniem, należy skorzystać z metody clear, która czyści tzw. flagi błędów.

np.

```

while( !plik.eof() )
{
    int iLiczba;
    plik >> iLiczba; //Wczytujemy liczbę
    if( plik.fail() )
    {
        cout << "Nie udalo sie wczytac liczby!" << endl;
        plik.clear(); //Czyścimy flagi błędów
        char cZnak;
        plik >> cZnak; //Wczytujemy znak
        if( plik.fail() )
            break; //Nie udało się wczytać znaku - wychodzimy z pętli (jeden znak zawsze
            powinno dać się odczytać jeżeli plik działa prawidłowo i nie napotkaliśmy końca pliku)
        else
            cout << "Napotkano znak '" << cZnak << "' << endl;

    } else
        cout << "Liczba = " << iLiczba << endl;
}

```

## 13)Flagi w programowaniu

**Flagi** są to zmienne, które przechowują informacje o stanie obiektu. Jedna flaga opisuje jeden stan i może przyjmować tylko dwie wartości tj. zero lub jeden. Wartość 'jeden' można utożsamiać z wartością true, choć znacznie częściej mówi się, że flaga jest ustawiona. Analogicznie jest z wartością 'zero', którą utożsamia się z wartością false i wówczas mówi się, że dana flaga nie jest ustawiona. Flagi błędów można więc wyobrazić sobie jako zmienne, które przechowują informacje o błędach jakie wystąpiły podczas używania pliku.

```

bool bFlagaFail = false;
bool bFlagaBad = false;
bool bFlagaEOF = false;
bool bFlagaGood = true;

```

Tym samym, w przypadku napotkania końca pliku podczas odczytywania danych, ustawiana jest flaga `bFlagaEOF = true`, dzięki czemu możemy później poznać przyczynę zakończenia odczytywania danych z pliku.

np.

```
#include <string>
#include <fstream>
#include <iostream>
```

```
bool wczytajLiczbe( ifstream & plik, int & iLiczba )
{
    plik.clear(); //Wyczyszczenie ewentualnych flag błędów
    plik >> iLiczba;
    if( plik.bad() )
    {
        cout << "Wystapil blad sprzetowy!" << endl;
        plik.close();
        return false;
    } else
    if( plik.fail() )
    {
        cout << "Nie udalo sie wczytac liczby!" << endl;
        return false;
    } else
        cout << "Liczba = " << iLiczba << endl;

    return true;
}
```

```
bool wczytajZnak( ifstream & plik, char & cZnak )
{
    plik.clear(); //Wyczyszczenie ewentualnych flag błędów
    plik >> cZnak;
    if( plik.bad() )
    {
        cout << "Wystapil blad sprzetowy!" << endl;
        plik.close();
        return false;
    } else
    if( plik.fail() )
    {
        cout << "Nie udalo sie wczytac znaku!" << endl;
        return false;
    } //if
    return true;
}
```

```
bool odczytajPlik( string sNazwaPliku )
{
```

```

    ifstream plik;
    plik.open( sNazwaPliku.c_str() );
    if( !plik.good() )
    {
        cout << "Nie udało sie otworzyc pliku." << endl;
        return false;
    } //if
    while( !plik.eof() )
    {
        int iLiczba;
        char cZnak;

        if( !wczytajLiczbe( plik, iLiczba ) && plik.bad() )
            return false; //wczytanie liczby nie powiodło się z powodu błędu sprzętowego
        else
            if( !wczytajZnak( plik, cZnak ) )
            {
                if( plik.bad() )
                    return false; //wczytanie liczby nie powiodło się z powodu błędu sprzętowego
                else
                    break; //nie ma więcej danych w strumieniu (bo jeden znak zawsze powinno się dać
                    odczytać)

            } //if
        cout << "Napotkany znak = " << cZnak << " " << endl;
    } //while
    plik.close();
    return true;
}

int main()
{
    if( odczytajPlik( "cpp0x.txt" ) )
        cout << "Plik zostal wczytany!" << endl;

    return 0;
}

```

### **Przykład 32**

#### **Temat:**

Zapisanie do pliku tekstowego o nazwie **plik\_out.txt** tekstu *tutaj podaj Twoje nazwisko*.

```
#include <cstdlib>
#include <iostream>
#include <fstream> // konieczny dla plikow

using namespace std;

int main(int argc, char *argv[])
{
    //=====
    //zapisywanie pliku tekstowego na dysk
    ofstream zapis_na_dysku; // utworzenie obiektu do zapisu na dysku
    cout<<"umiem zapisywac do pliku dyskowego"<<endl;
    zapis_na_dysku.open("plik_out.txt"); // otwarcie pliku tekstowego
    zapis_na_dysku<<"tutaj podaj Twoje nazwisko"; //zapisanie do pliku
    zapis_na_dysku.close(); //zamkniecie pliku
    cout<<"zamknalem plik"<<endl;

    //=====
    //odczytywanie pliku z dysku
    ifstream odczyt_z_dysku; // utworzenie obiektu do odczytu z dysku
    odczyt_z_dysku.open("plik_out.txt"); // otwarcie pliku tekstowego
    char pobierz;
    if (!odczyt_z_dysku.is_open()) //sprawdzenie czy plik otwarty
    { cout<<"nie otwarto pliku"<<endl; system("PAUSE");return 1;}
    {
        while(odczyt_z_dysku.good())
        { // sprawdzanie czy podczas odczytu nie ma bledu
            odczyt_z_dysku>>pobierz;
            cout<<pobierz;
        }
    }
    cout<<endl;
    odczyt_z_dysku.close();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### **Przykład 33**

#### **Temat:**

Zapisanie do pliku tekstowego o nazwie **wynik\_mediana.txt** obliczonej medianie ciągu liczbowego.

#### **Definicja mediany**

Jest to wartość tego elementu ciągu, dla którego ilość wyrazów mniejszych i większych od jego wartości jest taka sama.

#### **Algorytm obliczania mediany.**

Aby obliczyć medianę ze zbioru  $n$  obserwacji, sortujemy je w kolejności od najmniejszej do największej (lub odwrotnie) i numerujemy od 1 do  $n$ .

Następnie, jeśli  $n$  jest nieparzyste, medianą jest wartość ciągu w środku (czyli element ciągu o numerze  $(n+1)/2$ .)

Jeśli natomiast  $n$  jest parzyste, wynikiem jest średnia arytmetyczna między dwiema wartościami ciągu, czyli wartość ciągu o numerze  $n/2$  i obserwacją numer  $n/2+1$ .

Wykonaj:

- Utwórz folder na dysku C: o nazwie cztery pierwsze litery (bez polskich liter). W pliku tekstowym, który zapiszesz we wcześniej założonym folderze, zapisz numer\_w\_dzienniku+20 liczb naturalnych większych lub równych 1 i mniejszych lub równych 5. Nazwa pliku tekstowego dane\_cztery\_pierwsze\_litery\_nazwiska.txt.
- Zapisz temat
- Zapisz w zeszycie co jest mediana.
- Uruchom przykład
- Zapisz przykład w zeszycie

```
#include <cstdlib>
```

```
#include <iostream>
```

```
#include <fstream> // konieczny dla plikow
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    int tablica[50];
```

```
    int i=0;
```

```
    int temp;
```

```
    system("cls");
```

```
    //=====
```

```
    //odczytywanie pliku z dysku
```

```
    ifstream odczyt_z_dysku;          // utworzenie obiektu do odczytu z dysku
```

```
    odczyt_z_dysku.open("c:/Dev-Cpp/dane.txt");// otwarcie pliku tekstowego
```

```
    if (!odczyt_z_dysku.is_open())    //sprawdzenie czy plik otwarty
```

```
    { cout<<"nie otwarto pliku"<<endl; system("PAUSE");return 1;}
```

```
    {
```

```
        while(!odczyt_z_dysku.eof())
```

```
        { // sprawdzanie czy podczas odczytu nie ma konca i pliku
```

```
            odczyt_z_dysku>>tablica[i];
```

```
            i++;
```

```
        }
```

```
        odczyt_z_dysku.close();
```

```
    }
```

```
    cout<<"ilosc wyrazow ciagu";
```

```
    cout<<"i="<<i<<endl;
```

```
    ofstream zapis_na_dysku;          // utworzenie obiektu do zapisu na dysku
```

```
    zapis_na_dysku.open("c:/Dev-Cpp/wynik_mediana.txt"); // otwarcie pliku tekstowego
```

```
    // sortowanie babelkowe ciagu
```

```
    for(int m=0;m<=(i-2);m++)
```

```
    {
```

```
        for(int n=0;n<=(i-2);n++)
```

```

    {
        if (tablica[n]>tablica[n+1])
        {
            temp=tablica[n];
            tablica[n]=tablica[n+1];
            tablica[n+1]=temp;
        }
    }
}
//wypisanie ciagu
for(int m=0;m<=(i-1);m++)
{
    cout<<"a["<<m<<"]="<<tablica[m]<<endl;
}
//sprawdzenie czy jest parzysta/nieparzysta ilosc wyrazow ciagu
if ( i%2==0)//parzyste
{
    cout<<"parzysta ilosc wyrazow tablicy"<<endl;
    cout<<"mediana="<<(tablica[(i-1)/2]+tablica[(i-1)/2+1])/2<<endl;
    zapis_na_dysku<<"parzysta ilosc wyrazow tablicy"<<endl;
    zapis_na_dysku<<"mediana="<<(tablica[(i-1)/2]+tablica[(i-1)/2+1])/2<<endl;}
else // nieparzyste
{
    cout<<"nieparzysta ilosc wyrazow tablicy"<<endl;
    cout<<"mediana="<<tablica[(i-1)/2]<<endl;
    zapis_na_dysku<<"nieparzysta ilosc wyrazow tablicy"<<endl;
    zapis_na_dysku<<"mediana="<<tablica[(i-1)/2]<<endl;
}
zapis_na_dysku.close();           //zamkniecie pliku
system("PAUSE");
return EXIT_SUCCESS;
}

```

### Zadanie 48a

W kolejnych wierszach pliku o nazwie „liczby-1.txt” znajdują się liczby zapisane w systemie dziesiętnym. Napisz programy, które dadzą odpowiedzi do poniższych podpunktów:

- Ile jest wszystkich liczb?
- Ile jest liczb podzielnych przez „5” ?
- Jaka liczba jest największa a jaka najmniejsza. Wypisz te liczby?
- Ile jest liczb, których liczba cyfr jest liczbą nieparzystą?
- Ile jest liczb, których liczba cyfr jest równa „3” a ile liczb, których liczba cyfr jest równa „4”?
- Ile jest liczb, których pierwsza cyfra jest równa ostatniej?
- Ile jest liczb pierwszych?

Dane wyjściowe: (wygląd pliku wyjściowego, dane przykładowe)

- 230
- 123

.....  
nazwa pliku wyjściowego „odpowiedz-kowalski\_47a.txt”

Rozwiązanie:

1)skorzystaj z kodu programu do rozwiązania podpunktu a)

2)rozwiązanie do punktu b)

-utwórz nowy licznik o nazwie licznik\_kow\_b i wyzeruj go

-w petli pobierającej z pliku tekstowego liczbę instrukcją if sprawdź podzielność, użyj reszty z dzielenia, zwiększ licznik gdy liczba jest podzielna przez 5.

3)rozwiązanie do punktu c)

-utwórz dwie zmienne max i min nadaj im wartości dla max=-1 dla min=999999

-w petli pobierającej z pliku tekstowego liczbę znajdź max i min według listy kroków:

KROK1: Jeśli liczba>max to max=liczba

KROK2: Jeśli liczba<min to min=liczba

-w petli pobierającej z pliku tekstowego liczbę instrukcją if sprawdź podzielność, użyj reszty z dzielenia, zwiększ licznik gdy liczba jest podzielna przez 5.

4)rozwiązanie do punktu d)

-dodaj obsługę obiektów string poprzez:

#include<string>

-utwórz obiekt(zmienną) o nazwie tekst\_kowalski

-utwórz licznik licznik\_kow\_d i wyzeruj go

-wewnątrz petli pobierz do zmiennej string tekst\_kowalski liczbę z pliku tekstowego liczbę

-sprawdź jaka jest długość(mierzona w ilości znaków) długość liczby zapisanej w zmiennej tekst\_kowalski do określenia długości zastosuj metodę **.size()** np. **tekst\_kowalski.size()**

sprawdzenie długości i sprawdzenie czy długość jest liczbą nieparzystą wykonaj wewnątrz instrukcji if, jeśli długość jest nieparzystą to zwiększ licznik\_kow\_d.

5)rozwiązanie do punktu e)

-utwórz licznik licznik\_kow\_e\_3 i wyzeruj go

-utwórz licznik licznik\_kow\_e\_5 i wyzeruj go

-sprawdź jaka jest długość(mierzona w ilości znaków) długość liczby zapisanej w zmiennej tekst\_kowalski do określenia długości zastosuj metodę **.size()** np. **tekst\_kowalski.size()**

sprawdzenie długości i sprawdzenie czy długość jest 3 wykonaj wewnątrz instrukcji if, jeśli długość jest 3 to zwiększ licznik\_kow\_e\_3.

-sprawdź jaka jest długość(mierzona w ilości znaków) długość liczby zapisanej w zmiennej tekst\_kowalski do określenia długości zastosuj metodę **.size()** np. **tekst\_kowalski.size()**

sprawdzenie długości i sprawdzenie czy długość jest 5 wykonaj wewnątrz instrukcji if, jeśli długość jest 3 to zwiększ licznik\_kow\_e\_5.

6)rozwiązanie do punktu f)

-utwórz licznik licznik\_kow\_f i wyzeruj go

-należy sprawdzić instrukcją if czy pierwszy znak liczby zapisanej w zmiennej tekst\_kowalski, (pierwszy znak to tekst\_kowalski [0]) jest równy ostatniemu ostatniemu (ostatni znak to tekst\_kowalski [tekst\_kowalski-1]) jeśli tak to zwiększ licznik\_kow\_e.

7)rozwiązanie do punktu g)

-utwórz licznik licznik\_kow\_g i wyzeruj go

-należy wykorzystać funkcję z przykładu 29a(dołącz-kopiowanie funkcję przed funkcję main)

-utwórz zmienną czy\_pierwsza\_kowalski typu bool. Przypisz do tej zmiennej wynik funkcji sprawdzającej czy liczba jest pierwsza, gdy wartość zmiennej czy\_pierwsza\_kowalski jest true to zwiększ licznik licznik\_kow\_g.



```
#include<iostream>
#include<fstream>

using namespace std;

int main()
{
    ifstream inFile;
    ofstream outFile;
    int liczba, licznik = 0;
    inFile.open("liczby-1.txt");
    outFile.open("odpowiedz_kowalski.txt");
    while(inFile >> liczba)
    {
        licznik++;
    }
    outFile << "a\n";
    outFile << licznik;
    inFile.close();
    outFile.close();
}
```

**Zadanie 49**

Napisz program, który odczyta zawartość pliku, a następnie wypisze na ekranie tylko te wiersze, w których znajduje się wyraz wprowadzony przez użytkownika.

**Zadanie 50a**

W pliku tekstowym (in\_cztery\_pierwsze\_litery\_nazwiska.dat) zapisz dwie liczby. Pierwsza liczba to przeciwprostokątna trojkątna, druga liczba to przyprostokątna. W pliku tekstowym (out\_cztery\_pierwsze\_litery\_nazwiska.dat) zapisz Twoje nazwisko, pole oraz obwód tego prostokąta.

Program będzie sprawdzał dane i wydawał komunikat np.

Boki muszą być większe od zera

Przeciwprostokątna musi być większa od przyprostokątnej

**Zadanie 51**

W pliku tekstowym (in\_cztery\_pierwsze\_litery\_nazwiska.in) zapisz 20+numer\_z\_dziennika liczb naturalnym większych lub równych -10 i mniejszych lub równych 12.. W pliku tekstowym (out\_cztery\_pierwsze\_litery\_nazwiska.out) zapisz Twoje nazwisko, średnią tego ciągu, wartość maksymalną i minimalną.

## **Generator losowy.**

### Inicjacja funkcji losowej.

W celu zainicjowania generatora wpisz instrukcję → `srand(time(0));`

`srand(time(0));` uruchamia generator liczb pseudolosowych, czyli jak gdyby "włącza bęben maszyny losującej".

### Losowanie liczby.

`liczba = rand()%zakres;`

do zmiennej `liczba` zostanie zapisana liczba naturalna z przedziału `<0; zakres)`.

### **Przykład 34**

Losowanie liczby z zakresu `<0,100)` aż do wciśnięcia dowolnego klawisza.

- Wpisz i uruchom przykład.
- Wpisz listing do zeszytu.
- Wpisz do zeszytu jak inicjujemy generator losowy, do czego jest on porównywany.
- Jak losujemy liczbę z zakresu?

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

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    int x;
    srand(time(0));
    do
    {
        x = rand()%100;
        cout<<x<<endl;
    }
    while(!kbhit());
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### **ZADANIE 49**

Używając instrukcji `for` napisz program losujący 20 liczby z zakresu `<-4;8>`.

### **Zadanie 50**

#### Uwaga:

Użyj nazw zmiennych:

`a_trzy_litery_nazwiska`.

Gra "większe-mniejsze". Zasady gry:

Celem użytkownika jest odgadnięcie liczby, wylosowanej przez komputer. Liczba jest zapamiętana i ukryta przed użytkownikiem. Użytkownik podaje kolejne liczby, natomiast

komputer w zależności od tego czy podana liczba jest mniejsza czy większa od wylosowanej zwraca komunikat "Za dużo" lub "Za mało". Gra kończy się, gdy użytkownik prawidłowo odgadł wartość liczby. Prowadzony jest licznik, który pokazuje ile razy użytkownik zgadywał. Liczba losowana jest z przedziału  $\langle \text{numer\_w\_dzienniku}, \text{numer\_w\_dzienniku} + 50 \rangle$ . Program wyświetli na początku działania informację, z jakiego przedziału wylosował liczbę.

Przykładowy wygląd ekranu:

Losuję liczbę z przedziału  $\langle 5, 55 \rangle$

Zgadujesz 1 raz

Podaj liczbę= 6

Za mało

Zgadujesz 2 raz

Podaj liczbę= 46

Za dużo

.....

Uwaga: do powtarzania użyj pętla do...while lub while warunek zakończenia to, że liczba losowana jest równa liczbie zgadywanej, do zliczania utwórz zmienną licznik\_nazwisko i dokonuj inkrementacji tej zmiennej po przejściu pętli

**ZADANIE 50a**

Napisz program losujący n różnych liczb ze zbioru  $\{1 \dots m\}$  możliwych  $n \leq m$ , m,n wczytywane z klawiatury.

## Napisy w CPP

### 1)Wprowadzenie

W języku C++ zdefiniowana jest **klasa string**, dostępna po dołączeniu pliku **nagłówkowego string**.

```
#include <string>
```

Pozwala ona na tworzenie napisów i manipulowanie.

### 2)Konstruktory

Obiekt klasy string można utworzyć na kilka sposobów:

```
string( )  
string(const string& wzor, size_type start = 0, size_type ile = npos)  
string(const char* wzor)  
string(const char* wzor, size_type ile)  
string(size_type ile, char c)  
string(const char* start, const char* kon)
```

są przeciążonymi konstruktorami klasy string. W ostatnim z nich typem argumentów może być dowolny iterator wskazujący na znaki: w najprostszym przypadku są to po prostu wskaźniki do znaków C-napisu.

### Przykłady

string s;	tworzy napis pusty;
string s1 = "Acapulco"; string s2("Acapulco");	tworzy napis zainicjowany kopią C-napisu;
string s("Acapulco",n);	tworzy napis zainicjowany kopią pierwszych n znaków C-napisu podanego jako pierwszy argument. Argument n jest typu size_type;
string s(n,'x');	tworzy napis zainicjowany n powtórzeniami znaku (w tym przypadku znaku 'x'). Argument n jest typu size_type. Dla n= npos zgłasza wyjątek length_error. Zauważmy, że nie ma konstruktora pobierającego pojedynczy znak jako jedyny argument. Nie jest to wielka strata, gdyż zawsze możemy użyć powyższego konstruktora w formie s(1,'x').
Jeśli s jest obiektem klasy string, to	
string s1(s);	tworzy napis zainicjowany kopią napisu s (jest to wywołanie konstruktora kopiującego);
string s2(s,n);	tworzy napis zainicjowany kopią napisu s poczynając od znaku na pozycji n, licząc pozycje, jak zwykle, od zera. Argument n jest typu size_type. Jeśli ma wartość większą lub równą długości napisu s, to zgłaszany jest wyjątek out_of_range. Zauważmy różnicę:

	jeśli s byłoby C-napisem, to n miałyby interpretację liczby znaków licząc od początku!
string s2(s,n,k);	tworzy napis zainicjowany kopią napisu s poczynając od pozycji n i uwzględniającej co najwyżej k znaków. Jeśli wartość n+k jest większa od długości napisu s to błędu nie ma: interpretowane to jest jako wszystkie znaki od pozycji n. W szczególności wartością k może być npos. Wszystkie trzy ostatnie przypadki są implementowane w postaci jednego konstruktora o dwóch parametrach domyślnych:
<pre>string(const string&amp; s, size_type start = 0,         size_type ile = npos);</pre> <p>Jest też konstruktor wykorzystujący jawnie fakt, że obiekt klasy string jest kolekcją znaków. Jego szczególnym przypadkiem jest konstruktor, którego dwoma argumentami są dwa wskaźniki do znaków w zwykłym C-napisie: nowo utworzony napis C++ zainicjowany zostanie ciągiem znaków od tego wskazywanego przez pierwszy argument (włącznie) do znaku wskazywanego przez drugi argument, ale bez niego (czyli wyłącznie). Na przykład</p> <pre>const char* cnapis = "0123456789"; string s(cnapis+1,cnapis+7); cout &lt;&lt; s &lt;&lt; endl;</pre> <p>wydrukuje '123456'. Jest to przykład bardziej ogólnego mechanizmu iteratorów, o których powiemy więcej w rozdziale o iteratorach . Na razie możemy iteratory traktować jako pewne uogólnienie wskaźników. Na przykład</p> <pre>string s("Warszawa"); string s1(s.begin()+3, s.end()-2); cout &lt;&lt; s1 &lt;&lt; endl;</pre> <p>wydrukuje 'sza'. Metody begin i end zwracają iteratory wskazujące na pierwszy oraz na pierwszy za ostatnim znak napisu. Dodawanie do i odejmowanie od iteratorów liczb całkowitych działa podobnie jak dla wskaźników. Jak zwykle, pierwszy iterator wskazuje na pierwszy znak który ma być uwzględniony, podczas gdy drugi na pierwszy znak który ma już być opuszczony.</p>	

### 3)Metody i operatory

#### a)operatora przypisania.

```
const char* cstr = "strin";
string s1, s2, s3, s(" C++");
s1 = cstr;
s2 = 'g';
s3 = s;
cout << s1 << s2 << s3 << endl;
```

wydrukuje ' string C++'. Przypisanie jest głębokie, co znaczy, że na przykład po przypisaniu s1=s2 obiekt s1 jest całkowicie niezależny od obiektu s2: późniejsze zmiany s2 nie wpływają na s1 i vice versa.

b)konkatenowanie - operatora dodawania.

```
string s1 = "C";
const char* cn = "string";

string s = s1 + '-' + cn;
cout << s << endl;
```

utworzy i wypisze ' C-string', bo wynikiem pierwszego złożenia będzie napis C++ zawierający ' C-', który następnie zostanie złożony z C-napisem ' string'. Natomiast konstrukcja

```
const char* cn = "C";
string s1 = "string";

string s = cn + '-' + s1;
cout << s << endl;
```

byłaby błędna, gdyż pierwsze „dodawanie” dotyczyłoby C-napisu i znaku, a nie napisu C++.

Operator '+' dodaje prawy argument, który może być napisem C++, C-napisem lub pojedynczym znakiem, do napisu C++ będącego lewym argumentem.

c)Napis może też być traktowany jako tablica znaków, pierwszy znak ma indeks [0]

```
string s("Basia");
s[0] = 'K';
for (int i = 0; i < 5; i++) cout << s[i];
```

wyświetli napis ' Kasia'.

d)operatory porówniania '==', '!=', '>', '>=', '<', '<='.

Przykład: Sortowanie napisów

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

void insertionSort(string[],int);

int main(void) {
    int i;
    string krolowie[] = {
        string("Zygmunt"), string("Michal"),
        string("Wladyslaw"), string("Anna"),
```



```

string("Jan"),    string("Boleslaw")
                };

const int ile = sizeof(krolowie)/sizeof(string);

insertionSort(krolowie, ile);

    for ( i = 0; i < ile; i++ )
        cout << setw(10) << krolowie[i] << endl;
    }

void insertionSort(string a[], int wymiar) {
    if ( wymiar <= 1 ) return;

    for ( int i = 1 ; i < wymiar ; ++i ) {
        int j = i;
        string v = a[i];
        while ( j >= 1 && v < a[j-1] ) {
            a[j] = a[j-1];
            j--;
        }
        a[j] = v;
    }
}

```

e)operatory wstawiania i wyjmowania ze strumienia, '<<' i '>>'.

operator '>>' działa tak, że pomijane są wiodące białe znaki, a wczytywanie kończy się po napotkaniu pierwszego białego znaku za napisem - nie da się więc wczytać w ten sposób napisu złożonego z wielu słów.

f)określanie długości napisu

**size( )**  
**length( )**

Na przykład:

jeśli napis="ZSE", to napis.size() zwróci 3.

g)sprawdzenie czy napis jest pusty

**bool empty( )** - zwraca, w postaci wartości logicznej, odpowiedź na pytanie czy napis jest pusty?

h)wyciągnięcie określonego znaku z łańcucha znaków

**char& at(size\_type n)** - zwraca odniesienie do n-tego znaku (licząc od zera) z napisu, na rzecz którego została wywołana. Zakres jest sprawdzany: jeśli n jest większe od lub równe długości napisu, wysyłany jest wyjątek `out_of_range`. Metoda ta zatem ma działanie podobne

do operatora indeksowania, ale, ze względu na sprawdzanie zakresu, jest mniej efektywna, choć bardziej bezpieczna.

Na przykład `string("Ula").at(2)` zwraca literę 'a'.

#### i)zmian długości napisu

`void resize(size_type n, char c = '\0')` - zmienia rozmiar napisu na n. Jeśli n jest mniejsze od aktualnej długości napisu, pozostałe znaki są usuwane. Jeśli n jest większe od długości napisu, napis jest uzupełniany do długości n znakami c - domyślnie znakami ' \0'.

#### j)kasowanie znaków z napisu

`void clear( )` - usuwa wszystkie znaki z napisu, pozostawiając go pustym. Równoważna wywołaniu metody `resize(0)`.

#### k)wycinanie ciągu znaków z napisu

`string substr(size_type start = 0, size_type ile = npos)` - zwraca napis będący podciągiem napisu na rzecz którego metoda została wywołana. Podciąg składa się ze znaków od pozycji start i liczy ile znaków. Jeśli start+ile jest większe niż długość napisu, to błędu nie ma; do podciągu brane są wszystkie znaki napisu od tego na pozycji start.

Na przykład

```
string s("Pernambuco");  
cout << s.substr(5,3) << endl;
```

wypisze 'mbu'.

#### l)kopiowanie ciągu znaków

`size_type copy(char cn[], size_type ile, size_type start = 0)` - kopiuje do C-napisu cn podciąg złożony z ile znaków, poczynając od tego na pozycji start. Zwraca liczbę przekopiowanych znaków. Znak ' \0' nie jest dostawiany. Zwróćmy uwagę na kolejność argumentów start i ile - odwrotną niż w metodzie substr.

Na przykład

```
char nap[] = "xxxxxx";  
string s("Barbara");  
string::size_type siz = s.copy(nap,3,2);  
cout << "Skopiowano " << siz << " znaki: \n"  
      << nap << endl;
```

wypisze ' Skopiowano 3 znaki: rbaxxx'.

#### m)zmiana ciągu znaków

void swap(string s1) - zamienia napis s1 z tym, na rzecz którego metodę wywołano. Na przykład

```
string s("Arles"), s1("Berlin");
s.swap(s1);
cout << s << " " << s1 << endl;
```

wypisze ' Berlin Arles'.

n)napis według wzorca

```
string& assign(const string& wzor)
string& assign(const char* wzor)
string& assign(string wzor, size_type start, size_type ile)
string& assign(const char* wzor, size_type ile)
string& assign(size_type ile, char c)
```

string& **assign(const char\* start, const char\* kon)** - ustala zawartość napisu i zwraca odniesienie do niego (zastępuje operator przypisania). Argumenty mają podobną postać jak dla konstruktorów. W ostatniej metodzie typem argumentów może być iterator wskazujący na znaki: na przykład są to po prostu wskaźniki do znaków C-napisu.

W najprostszym przypadku argumentem jest inny napis w postaci napisu C++ lub C-napisu; tak więc po

```
string s1("xxx"), s2("Zuzia");
s1.assign(s2);
s2.assign("Kasia");
```

wartością s1 będzie ' Zuzia' a zmiennej s2 ' Kasia'.

Tak jak dla konstruktorów, jako argument można podać podnapis napisu C++ o podanej pozycji początku i długości - za duża długość znaczy aż do końca. Można też podać podnapis C-napisu złożony z podanej liczby znaków licząc od początku:

```
string s1("0123456789"), s2;
const char* p = "0123456789";
s1.assign(s1,2,5);
s2.assign(p, 5);
cout << s1 << " " << s2 << endl;
```

wypisze ' 23456 01234'.

Za pomocą metody assign można utworzyć napis złożony z ile powtórzeń znaku (piąta forma z wymienionych powyżej). Można też użyć wskaźników do znaków w C-napisie jako iteratorów wyznaczających podciąg; trzeba tylko pamiętać, że wskazywany podnapis nie zawiera wtedy znaku wskazywanego jako górne ograniczenie podciągu:

```
string s1("0123456789"), s2;
const char* p = "0123456789";
s1.assign(5,'x');
s2.assign(p+3,p+5);
cout << s1 << " " << s2 << endl;
```

wypisze 'xxxxx 34'.

#### o)wstawianie tekstu

**string& insert(size\_type gdzie, const string\* wzor)**  
**string& insert(size\_type gdzie, const char\* wzor)**  
**string& insert(size\_type gdzie, string wzor, size\_type start, size\_type ile)**  
**string& insert(size\_type gdzie, const char\* wzor, size\_type ile)**  
**string& insert(size\_type gdzie, size\_type ile, char c)** - modyfikuje napis i zwraca odniesienie do niego, wstawiając na pozycji o indeksie gdzie znaki z innego napisu, opisywanego przez pozostałe argumenty. Znaki od pozycji gdzie są „przesuwane” w prawo za fragment wstawiony. Znaczenie argumentów określających ciąg znaków do wstawienia jest takie samo jak dla metody assign. Na przykład

```
string s1("mama"), s2("plastyka");
const char* p = "temat";
s1.insert(2,p,2).insert(6,s2,4,4);
cout << s1 << endl;
```

wypisze 'matematyka'.

Prócz tych form istnieją jeszcze trzy inne formy tej metody:

**iterator insert(iterator gdzie, char c)**  
**void insert(iterator gdzie, size\_type ile, char c)**  
**void insert(iterator gdzie, const char\* start, const char\* kon)** - gdzie pierwszym argumentem jest iterator (uogólniony wskaźnik) do znaku w napisie, przed który wstawiany jest ciąg określany przez pozostałe argumenty. W trzeciej z tych form rolę dwóch ostatnich argumentów mogą pełnić nie tylko wskaźniki do znaków, ale ogólnie iteratory wskazujące na znaki (na przykład iteratory typu `string::iterator`). Na przykład

```
string s1("abbccdd");
s1.insert(s1.insert(s1.begin()+5,'c')+1,2,'d');
cout << s1 << endl;
```

wypisze ' abbccddddd'.

#### p)modyfikacja napisu

**string& append(const string& wzor)**  
**string& append(const string& wzor, size\_type start, size\_type ile)**  
**string& append(const char\* wzor)**  
**string& append(const char\* wzor, size\_type ile)**  
**string& append(size\_type ile, char c)**  
**string& append(const char\* start, const char\* kon)** - modyfikuje napis i zwraca odniesienie do niego, dodając na końcu tego napisu napis określany przez argumenty. Znaczenie argumentów określających ciąg znaków do wstawienia jest takie samo jak dla metod insert. W ostatniej metodzie typem argumentów może być dowolny iterator wskazujący na znaki: tu są to po prostu wskaźniki do znaków C-napisu.

### r)usuwanie fragmentu tekstu

string& erase(size\_type start = 0, size\_type ile = npos)

iterator erase(iterator start)

iterator erase(iterator start, iterator kon) - usuwa fragment napisu, zwracając odniesienie do zmodyfikowanego napisu lub iterator odnoszący się do zmodyfikowanego napisu i wskazujący na pierwszy znak za fragmentem usuniętym. Pierwsza forma usuwa ile znaków (domyślnie npos, czyli wszystkie) od pozycji start (domyślnie od pozycji zerowej). Druga forma usuwa wszystkie znaki od pozycji wskazywanej przez iterator start, a trzecia od znaku wskazywanego przez iterator start do znaku poprzedzającego znak wskazywany przez iterator kon. Na przykład

```
string s("0123456789");
string::iterator it = s.erase(s.begin()+3,s.end()-3);
cout << s << " " << *it << endl;
```

wypisze '012789 7'.

### s)zamiana tekstu

string& replace(size\_type start, size\_type ile, const string& wzor)

string& replace(size\_type start, size\_type ile, const string& wzor, size\_type s, size\_type i)

string& replace(size\_type start, size\_type ile, const char\* wzor, size\_type i)

string& replace(size\_type start, size\_type ile, const char\* wzor)

string& replace(size\_type start, size\_type ile, size\_type i, char c)

string& replace(iterator start, iterator kon, const string& wzor)

string& replace(iterator start, iterator kon, const char\* wzor)

string& replace(iterator start, iterator kon, const char\* wzor, size\_type i)

string& replace(iterator start, iterator kon, size\_type i, char c)

string& replace(iterator start, iterator kon, const char\* st1, const char\* kn1) - usuwa fragment napisu określony pierwszymi dwoma argumentami i wstawia na to miejsce napis określony pozostałymi argumentami. W ostatniej z tych metod typem dwóch ostatnich argumentów może być dowolny iterator wskazujący na znaki: w najprostszym przypadku są to po prostu wskaźniki do znaków C-napisu. Zasady określania napisów lub podnapisów są te same co dla metod insert. Metody replace zwracają odniesienie do zmodyfikowanego napisu. Na przykład

```
string s("0123456789");
const char* p("abcdef");
s.replace(0,2,p,2).replace(s.end()-2,s.end(),p+4,p+6);
cout << s << endl;
```

wypisze 'ab234567ef'.

### r)znajdowanie tekstu

size\_type find(const string\* s, size\_type start = 0)

size\_type find(const char\* p, size\_type start = 0)

size\_type find(const char\* p, size\_type start, size\_type ile)

size\_type find(char c, size\_type start = 0)

size\_type rfind(const string\* s, size\_type start = npos)

size\_type rfind(const char\* p, size\_type start = npos)  
size\_type rfind(const char\* p, size\_type start, size\_type ile)  
size\_type rfind(char c, size\_type start = npos) - szukają, poczynając od pozycji start, podnapisu określonego przez pozostałe argumenty. Rodzina metod rfind działa analogicznie, ale przeglądanie odbywa się od pozycji startowej w kierunku początku napisu. Wszystkie metody zwracają pozycję (indeks) pierwszego znaku poszukiwanego podnapisu w napisie przeszukiwanym. Jeśli przeszukiwanie zakończyło się porażką, zwracane jest npos. W przykładzie poniżej find szuka w napisie 'abc345abcAB' poczynając od pozycji 3 (czyli od cyfry '3') napisu złożonego z dwóch pierwszych znaków C-napisu p (czyli napisu 'ab'):

```
string s("abc345abcAB");  
const char* p("abcdef");  
string::size_type i = s.find(p,3,2);  
cout << s.substr(i-1,5) << endl;
```

Fragment wypisuje '5abcA'.

#### t)znajdowanie tekstu od pozycji

size\_type find\_first\_of( /\* args \*/ )  
size\_type find\_last\_of( /\* args \*/ )  
size\_type find\_first\_not\_of( /\* args \*/ )  
size\_type find\_last\_not\_of( /\* args \*/ ) - mają typ wartości i argumentów takie same jak odpowiednie metody find i rfind. Pierwsze dwie metody szukają, poczynając od pozycji start, pierwszego wystąpienia jakiegokolwiek znaku należącego do napisu określonego przez pozostałe argumenty. Kierunek przeszukiwania dla metod z \_last\_ w nazwie jest od pozycji start wstecz, a dla metod z \_first\_ w nazwie - do przodu. Metody z drugiej pary, te z \_not\_ w nazwie, działają podobnie, ale szukają wystąpienia znaku nie należącego do napisu określonego przez pozostałe argumenty. Jeśli odpowiedni znak został znaleziony, zwracana jest jego pozycja; jeśli nie, zwracane jest npos. Na przykład

1. string s("abc123.,!");
2. const char\* p = "!.?:1234";
3. string::size\_type i = s.find\_first\_of(p);
4. string::size\_type k = s.find\_last\_not\_of(p,s.size()-1,5);
5. string s1(s.begin()+i,s.begin()+k+1);
6. cout << i << " " << k << " " << s1 << endl;

wypisze '3 5 123'. W linii czwartej jako drugi argument podaliśmy s.size()-1, bo przeszukiwanie odbywa się do tyłu, a więc zacząć trzeba od znaku ostatniego, a nie od pierwszego. Uwzględniamy przy tym tylko 5 pierwszych znaków ze wzorca p, a więc znaki '!.?:'. W linii piątej tworzymy nowy obiekt klasy string i inicjujemy go wycinkiem napisu s. W drugim argumentcie dodajemy do s.begin()+k jedynekę, gdyż wycinek zawiera znaki tylko do poprzedzającego ten wskazywany przez drugi iterator.

#### u)porównywanie tekstów

```
int compare(const string& wzor)  
int compare(size_type start, size_type ile, const string& wzor)  
int compare(size_type start, size_type ile, const string& wzor,
```

```
size_type s, size_type ile1)
int compare(const char* p)
int compare(size_type start, size_type ile, const char* p, size_type i = npos) - porównują napis
lub jego podciąg określony przez start i ile z napisem wyznaczonym przez pozostałe
argumenty. Wynikiem jest -1, jeśli napis porównywany jest leksykograficznie wcześniejszy
od napisu podanego jako argument, zero, jeśli są identyczne, a +1, jeśli jest leksykograficznie
późniejszy.
```

w)wstawianie na koniec tekstu

void **push\_back(char c)** - wstawia na koniec napisu znak c - s.push\_back(c) działa jak wywołanie s.insert(s.end(),c), tyle że jest bezrezultatowe (metoda insert zwraca przy takim wywołaniu iterator). Ten sam efekt można uzyskać za pomocą metody append lub operatora '+='.

#### z)inne metody

const char\* **c\_str( )** - zwraca wskaźnik do stałego C-napisu złożonego ze znaków napisu C++, na rzecz którego była wywołana. C-napis kończy się znakiem '\0', a zatem może być argumentem funkcji operujących na zwykłych C-napisach. Pamiętać tylko należy, że zwracany wskaźnik jest typu const, a więc w razie konieczności modyfikacji uzyskany C-napis trzeba przekopiować do zwykłej, modyfikowalnej tablicy znaków.

**iterator begin( )** - zwraca iterator (uogólniony wskaźnik, patrz rozdział o iteratorach) wskazujący na pierwszy znak napisu.

**iterator end( )** - zwraca iterator wskazujący na znak pierwszy za ostatnim napisu.

#### **reverse\_iterator rbegin( )**

reverse\_iterator rend( ) - zwraca iterator „odwrotny” wskazujący na początek i koniec napisu w odwrotnym porządku. Na przykład po

```
string s1("korab");
string s2(s1.rbegin(),s1.rend());
```

s2 będzie zawierać napis ' barok'.

Prócz metod klasy string biblioteka włączana za pomocą pliku nagłówkowego string dostarcza bardzo przydatną funkcję (a więc nie metodę klasy):

#### **istream& getline(istream& str, string& s)**

istream& getline(istream& str, string& s, char eol) - wczytuje ze strumienia str jedną linię tekstu i wstawia ją do napisu s. Linia może zawierać białe znaki (prócz znaku końca linii). Znak końca linii jest wyjmowany ze strumienia, ale nie jest włączany do wynikowego napisu. Znak, który ma pełnić rolę znaku końca linii, można podać jako trzeci argument funkcji - domyślnie jest nim '\n'. Funkcja zwraca odniesienie do strumienia str, tak więc nieco dziwna konstrukcja

```
string s1,s2;
getline(cin,s1) >> s2;
```

### **ZADANIE 50a1**

Temat: Program, który po wczytaniu wyrazu do zmiennej tekstowej, wydrukuje go w pionie.

Wykonaj:

1)Uruchom przykład.

2)następnie wypisz wczytany tekst w kolumnie miesiąc\_urodzenia+30.

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

using namespace std;

void gotoxy(int x, int y)
{
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main(int argc, char *argv[])
{
    string tekst;
    cout<<"Podaj tekst";
    cin>>tekst;
    system("cls");
    for (int i=0;i<tekst.size();i++)
    {
        gotoxy(0,i);
        cout<<tekst[i];
    }
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

### **ZADANIE 50a2**

Temat: Program, który po wczytaniu wyrazu do zmiennej tekstowej oraz pozycji górnego lewego rogu kwadratu x\_poz i y\_poz, wydrukuje go w postaci kwadratu.

```
(x_poz,y_poz)kwadrat
      w      a
      a      r
      d      d
      r      a
      a      w
      tardaw
```

### **ZADANIE 50a3**

Temat: Program, który po wczytaniu wyrazu do zmiennej tekstowej, wydrukuje go w dwóch liniach:

```
k a r t
w d a
```



### **ZADANIE 50b**

Napisz program, który po wczytaniu liczby obliczy sumę jej cyfr. Np. 3468 suma=21

#### Algorytm:

Wczytaj liczbę do zmiennej int. Zamień liczbę na zmienną string. Odczytaj pojedyncze znaki ze zmiennej string. Zmieniaj pojedynczy znak na zmienną typu int i dodawaj do zmiennej suma. Rzutowanie zmiennej na *cyfra=(int)tekst* powoduje przypisanie do zmiennej cyfra numeru kodu ASCII, aby uzyskać prawidłowe wartości cyfr należy odjąć 48.

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <cstdio>

using namespace std;

int main(int argc, char *argv[])
{
    int liczba;
    string tekst;
    string tmp;
    cout<<"podaj liczbę=";
    cin>>liczba;
    sprintf((char*)tmp.c_str(), "%d", liczba);
    tekst = tmp.c_str();
    cout<<"wczytana liczba="<<tekst<<"\n";
    int cyfra,suma=0;
    for (int i=0;i<=tekst.size()-1;i++)
    {
        cyfra = (int)tekst[i]-48;
        suma+=cyfra;
    }
    cout<<"suma="<<suma<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

## **ZADANIE 50b1**

Temat: Zamiana liczby dwójkowej na dziesiętną

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int bin_to_dec(string bin)
{
    int i;
    int dec = 0;
    for(i = 0; i < bin.size(); i++)
    {
        dec = 2 * dec + (bin[i]-48);
    }
    return dec;
}

int main(int argc, char *argv[])
{
    string liczba_bin;
    int liczba_dec;
    cout<<"podaj liczbe binarna=";
    cin>>liczba_bin;
    liczba_dec=bin_to_dec(liczba_bin);
    cout<<"liczba po zamianie dec=";
    cout<<liczba_dec<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

## **ZADANIE 50b2**

Temat: Zamiana dziesiętnej na liczbę dwójkową.

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

string dec_to_bin (int dec)
{
    string bin = "";
    while (dec >= 1)
    {
        if (dec % 2 == 0)
            bin = '0' + bin;
        if (dec % 2 == 1)
            bin = '1' + bin;
        dec = dec/2;
    }
    return bin;
}

int main(int argc, char *argv[])
{
    string liczba_bin;
    int liczba_dec;
    cout<<"podaj liczbe dziesiętna=";
    cin>>liczba_dec;
    liczba_bin=dec_to_bin(liczba_dec);
    cout<<"liczba po zamianie bin=";
    cout<<liczba_bin<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

## **ZADANIE 50b3**

Temat: Działania na liczby dwójkowych zapisanych w pliku tekstowym

W kolejnych wierszach pliku o nazwie „liczby-2.txt” znajdują się liczby zapisane w systemie binarnym. Napisz programy, które dadzą odpowiedzi do poniższych podpunktów.

- Ile jest liczb parzystych?
- Wypisz liczby wartości liczb binarnych postaci dziesiętnej.
- Która liczba jest największa a która najmniejsza. Wypisz te liczby w postaci binarnej?
- Wypisz liczby, które są liczbami pierwszymi. Zapisz je w postaci (binarna : dziesiętna)

Dane wyjściowe: (wygląd pliku wyjściowego, dane przykładowe)

- 29
- 101
- 23
- 145

.....

nazwa pliku wyjściowego „odpowiedz-kowalski\_50b3.txt”

Rozwiązanie:

1)skorzystaj z kodu programu ze zdefiniowanymi funkcjami dec\_to\_bin oraz bin\_to\_dec (przykłady poprzednie)

2)rozwiązanie do punktu a)

-utwórz nowy licznik o nazwie licznik\_kow\_a i wyzeruj go

-w pętli pobierającej z pliku tekstowego liczbę, zamień liczbę z bin na dec i instrukcją if sprawdź podzielność, użyj reszty z dzielenia, zwiększ licznik gdy liczba nie jest podzielna przez 2.

3)rozwiązanie do punktu b)

zamień liczbę z bin na dec i umieść w nowym wierszu liku

4)rozwiązanie do punktu c)

zamień liczbę z bin na dec i umieść w nowym wierszu liku

-utwórz dwie zmienne max i min nadaj im wartości dla max=-1 dla min=999999

-w petli pobierającej z pliku tekstowego liczbę znajdź max i min według listy kroków:

KROK1: Jeśli liczba>max to max=liczba

KROK2: Jeśli liczba<min to min=liczba

zapisz max i min w pliku w postaciach bin

5)wykorzystaj funkcję określającą czy liczba jest pierwsza (przykład 29a), zapisz liczbę pierwsza w pliku w dwóch postaciach bin : dec.

### **ZADANIE 50c**

Palindromem nazywamy słowo, które czytane od lewej i od prawej strony jest takie samo. Wyraz „kajak” jest Palindromem. Wyraz „owca” nie jest Palindromem

W pliku palindromy.txt umieszczono w kolejnych wierszach 1000 słów o długościach od 1 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 palindromy.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. Plik wyjściowy palindromy\_kowalski.txt.

### **ZADANIE 50d**

Napisz program, który po wczytaniu tekstu (WYRAZU→każda litera inna, wielkość liter nie ma znaczenia) i dwóch liter odpowie ile liter znajduje się pomiędzy tymi literami np.

**Lato** i litery „L” „o” wynik programu→2

**krowa** i litery „k” „r” wynik programu→0

### **ZADANIE 50e**

Program, który będzie wycinał tekst z pomiędzy wczytanych liter

### **ZADANIE 50f**

Napisz program, który po wczytaniu tekstu (WYRAZ) i dwóch liter, pierwsza litera→ jaką literę zastępujemy

druga litera → na jaką literę zamienić

np. wyraz→mama

pierwsza litera „m”

druga litera „t”

wynik programu→tata

### **ZADANIE 50g**

Napisz program, który będzie zliczał ilość znaków w tekście. Wygląd działania programu dla tekstu matematyka:

a - 3 razy

m - 2 razy

t - 2 razy  
y - 1 raz  
e - 1 raz  
k - 1 raz

zadziała i wpisze pierwszy wczytany wiersz do napisu s1, a pierwsze słowo następnego wiersza do napisu s2.

## **Deklaracja wskaźnika**

### Definicja

**Wskaźnik na zmienną** jest to zmienna, która przechowuje w pamięci RAM adres innej zmiennej.

### Wy tłumaczenie

Wszystkie zmienne, jakie deklarujemy są niczym innym jak tylko komórkami pamięci operacyjnej RAM. Zatem odwołując się do zmiennej poprzez jej nazwę odwołujemy się do przydzielonej jej (zmiennej) pamięci. Wartością wskaźnika jest natomiast adres pamięci RAM, gdzie znajduje się taka zmienna.

### Deklaracja wskaźnika

typ \*nazwa;

Gwiazdka przed nazwą zmiennej informuje o tym, że jest on wskaźnikiem.

### Wyświetlanie początku adresu pamięci RAM gdzie znajduje się zmienna

Aby pobrać adres dowolnej zmiennej wystarczy napisać: **&nazwa\_zmiennej**. Otrzymamy adres pierwszego bajta danych wybranej zmiennej. Adresy zmiennych są wyświetlane w postaci szesnastkowej.

### Wielkość wskaźnika.

Wskaźnik wskazuje na dane, a sam najczęściej zajmuje **4 bajty** bez względu na to, na jakie dane wskazuje.

## **Przykład 34a**

Temat: Wyświetlanie adresu pierwszego bajtu pamięci dla zmiennej.

Wykonuj:

Uruchom przykład.

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    int zmienna_calkowita;
```

```
    cout<<"pierwszy bajt pamięci RAM zmiennej całkowitej=";
```

```
    cout<<&zmienna_calkowita<<endl;
```

```
    system("PAUSE");
```

```
    return EXIT_SUCCESS;
```

```
}
```

## **ZADANIE 51**

Temat: Wyświetlanie adresu pierwszego bajtu pamięci dla zmiennej typu tablicowego, float oraz rekordowego.

Wykonaj:

1)Utwórz zmienną typu float o nazwie `zmienna_kow` i wyświetl adres pamięci dla tej zmiennej,

2)Utwórz zmienną typu tablicowego o trzech elementach o nazwie `tab_kow` wpisz dane do tablicy {liczba\_liter\_nazwiska, liczba\_liter\_imienia,waga} i wyświetl adres pamięci dla tej tablicowej oraz w nowych wierszach adresy dla wszystkich elementów tablicy,

3)Utwórz zmienną typu rekordowego o nazwie rec\_kow o polach Id\_kow nazwisko\_kow i wyświetl adres pamięci dla tej zmiennej oraz w nowych wierszach adresy pól,

### **Przykład 35**

- Wpisz i uruchom przykład.
- Wpisz listing do zeszytu.
- Wpisz uwagi zawarte pod przykładem.

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

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    int i=20;
    int *wskaznik;
    wskaznik=&i;
    cout<<"wskaznik jest typu integer"<<"\n";
    cout<<"wskaznik wskazuje na zmienna i ktora ma wartosc "<<i<<"\n";
    cout<<"wartosc zmiennej i poprzez wskaznik "<<*wskaznik<<endl;
    cout<<"wartosc wskaznika to adres gdzie przechowywana jest wartosc zmiennej i:
"<<wskaznik;
    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Uwagi:

- \*wskaznik → oznacza wartość, która znajduje się pod adresem będącym wartością wskaźnika wskaznik;
- wskaznik → oznacza adres, czyli wartość wskaźnika o nazwie wskaznik;
- &i → oznacza adres komórki, w której znajduje się wartość zmiennej i,
- adresy podawane są w postaci szesnastkowej,
- wskaźnik musi być związany ze zmienną;
- związanie wskaźnika ze zmienną może być użyte dla tych samych typów,
- można zmienić przypisanie wskaźnika do zmiennej.

### **ZADANIE 52**

Napisz program, który zainicjuje zmienne typu integer i\_nazwsko\_ucznia, j\_nazwisko\_ucznia i nada im wartości początkowe odpowiednio 100+numer\_z\_dziennika oraz 200+numer\_z\_dziennika. Zadeklaruj wskaźnik o nazwie wsk\_nazwisko\_ucznia. Powiąż zmienną i\_nazwsko\_ucznia ze wskaźnikiem, wypisz adres przechowywania zmiennej i\_nazwsko\_ucznia, następnie powiąż zmienną j\_nazwisko\_ucznia ze wskaźnikiem, wypisz adres przechowywania zmiennej j\_nazwisko\_ucznia.

---

## **Dynamiczne zarządzanie pamięcią new i delete**

1)Dlaczego stosujemy dynamiczne zarządzanie pamięciom.

Zarezerwowane zmienne w sposób statyczny zajmują pamięć, aż do końca działania program. Zmienne rezerwowane dynamicznie mogą zwalniać pamięć poprzez ich skasowanie z pamięci.

#### 2) Instrukcje stosowane w dynamicznym zarządzaniu pamięcią.

Dynamiczne zarządzanie pamięcią za pomocą operatorów **new** i **delete**.

#### 3) Dynamiczne przydzielenie pamięci instrukcja new

```
wskaznik1 = new typ_zmiennej;
```

```
wskaznik2 = new typ_zmiennej[ ilosc_elementow_danego_typu ];
```

#### 4) Okreslanie wielkości pamięci dla zmiennej.

a) Typ zmiennej informuje operator new, o rozmiarze pamięci jaka ma zostać przydzielona.

b) Jeśli chcemy aby nowo przydzielony blok był tablicą to aktualną składnię uzupełniamy o dodatkowy parametr, w którym określamy ilość elementów tak samo jak robiliśmy to w przypadku tablic.

c) Jeśli przydział pamięci powiódł się, to wartość zmiennej wskaźnik będzie różna od zera. Jeśli wartość wskaźnika będzie równa 0, to pamięć nie została przydzielona. Wartość 0 bardzo często jest zastępowana stałą **NULL**. Zalecane jest jednocześnie korzystanie ze stałej **NULL**, ponieważ standardy związane z wartością 0 mogą się kiedyś zmienić, a w związku z tym Twoje programy przestałyby działać.

#### 5) Powody, dla których pamięć nie mogła zostać przydzielona to:

a) Rozmiar bloku pamięci, który chcesz zarezerwować jest zbyt duży;

b) System nie posiada więcej zasobów pamięci i w związku z tym nie może Ci jej przydzielić.

#### 6) Zwalnianie pamięci przydzielonej dynamicznie instrukcja delete

a) Jeśli pamięć dla danych, na które wskazuje zmienna wskaźnik została przydzielona bez parametru określającego ilość elementów w tablicy, to usuwana jest następującą składnią:

```
delete wskaznik;
```

b) Jeśli natomiast przydzieliliśmy pamięć z użyciem parametru określającego ilość elementów tablicy to musimy poinformować operator delete o tym, że wskaźnik wskazywał na tablicę rekordów. Aby to zrobić dopisujemy zaraz za operatorem nawiasy kwadratowe []. Nie podajemy jednak w nich rozmiaru tablicy, ponieważ operator ten sam ustala rozmiar bloku jaki został przydzielony, a następnie go usuwa z pamięci. Składnia tej operacji wygląda następująco:

```
delete[] wskaznik_do_tablicy;
```

#### 7) Kopiowanie bloków pamięci

Jeśli będziemy chcieli przekopiować zawartość pamięci z jednego miejsca do drugiego możemy zrobić to conajmniej na dwa sposoby.

**Sposób pierwszy** to wykorzystanie jakiegokolwiek pętli i kopiowanie danych bajt po bajcie. Przykład:

```
for( int i = 0; i < ilosc; i++ ) nowybufor[ i ] = starybufor[ i ];
```

Rozwiązanie to nie należy on do najwydajniejszych.

---



**Sposób drugi** to wykorzystanie funkcji, która służy do kopiowania bloków pamięci.

```
void * memcpy( void * adres_docelowy, const void * adres_zrodlowy, size_t ilosc );
```

-Jako pierwszy parametr (adres\_docelowy) podajemy adres do pamięci pod którym mają się znaleźć nowe dane.

-Drugi parametr (adres\_zrodlowy) to miejsce z którego dane mają zostać pobrane i również określamy je za pomocą adresu.

-Trzecim, a zarazem ostatnim parametrem (ilosc) jest ilość bajtów, jaka ma zostać przekopiowana ze źródła do celu.

---

### **Przykład 34a**

```
#include<iostream>
#include<string>
#include<cstring>
#include<conio.h>
using namespace std;
int main()
{
    int rozmiar = 0;
    int dlugosc = 0;
    char * tablica = NULL;
    cout << "Pusty wiersz konczy dzialanie programu." << endl;
    for( int i = 0; i < 40; i++ ) cout << "-";

    cout << endl;
    string tWiersz;
    do
    {
        getline( cin, tWiersz );
        if( tWiersz.length() > 0 )
        {
            tWiersz += "\r\n"; //dopisanie nowego wiersza
            if( dlugosc + tWiersz.length() + 1 > rozmiar ) //potrzeba więcej pamięci niż jest
dostępne
            {
                cout << "Tworzy nowy blok pamieci!" << endl;
                int tNarzutDanych = 20; //jeśli ustawisz 0 to rezerwacja będzie się odbywała za
każdym razem
                rozmiar = tWiersz.length() + dlugosc + 1 + tNarzutDanych; //nowy rozmiar bloku
                char * tNoweDane = new char[ rozmiar ]; //rezerwacja nowego bloku pamięci,
który pomieści stare i nowe dane
                if( tablica != NULL ) memcpy( tNoweDane, tablica, dlugosc ); //jeśli stara tablica
istnieje to skopiuj dane do nowej tablicy

                memcpy( & tNoweDane[ dlugosc ], & tWiersz[ 0 ], tWiersz.length() ); //skopiuj
dane do nowej tablicy w wyznaczone miejsce
```

---

```
    if( tablica != NULL ) delete[] tablica; //zwolnij pamięć zajmowaną przez stare dane

    tablica = tNoweDane; //nadaj nowy wskaźnik zmiennej tablica
} else
{ //jest wystarczająca ilość pamięci nie wymagana rezerwacja
    cout << "Jest wystarczająca ilość miejsc!" << endl;
}
memcpy( & tablica[ dlugosc ], & tWiersz[ 0 ], tWiersz.length() ); //skopiuj dane do
tablicy w wyznaczone miejsce
    dlugosc = tWiersz.length() + dlugosc; //zapisz długość tekstu
    tablica[ dlugosc ] = 0; //oznacz miejsce końca tekstu w tablicy
}
} while( tWiersz.length() != 0 );

if( tablica != NULL )
{
    cout << "Dane jakie wypisales to: " << endl;
    cout << tablica << endl;
    delete[] tablica;
} else cout << "Nie wpisales niczego!";

getch();
return( 0 );
}
```

---

## Grafika w CPP

### Instalacja grafiki oraz używanie grafiki

- Pobierz od nauczyciela program instalacyjny bibliotek graficznych i zainstaluj te pliki na dysku twardym. Pojawią się trzy pliki winbgim.h, winbgim.cpp, libbgi.a.
- Pliki winbgim.h, winbgim.cpp kopiujemy do katalogu c:\dev-cpp\include
- Plik libbgi.a kopiujemy do katalogu c:\dev-cpp\lib
- Tworzymy nowy projekt Nowy→Projekt→Console Application→nazwa projektu→OK.
- Z menu Projekt wybieramy Opcje projektu→zakładka Parametry→w oknie Konsolidator wpisujemy -lbgi -lgdi32 (minus tekst lbgi spacja minus tekst lgdi32)→OK.
- Z menu Projekt→Dodaj do projektu→przechodzimy do folderu i dwukrotnie klikamy na winbgim (niebieska ikona)
- do treści programu dołączamy #include <winbgim.h>
- [http://math.uni.lodz.pl/~polrola/strony/06071-podyplom/o\\_grafice.html](http://math.uni.lodz.pl/~polrola/strony/06071-podyplom/o_grafice.html) możesz skorzystać z tej strony

<i>Polecenie</i>	<i>Opis</i>
Delay (x)	Opóźnienie wykonywania programu o x milisekund.
Initwindow (x, y)	Uruchamia okno o rozmiarze „x” na „y” pikseli
Setfillstyle (styl, kolor)	Ustawia styl i kolor wypełniania
Bar (x1, y1, x2, y2)	Rysuje prostokąt od punkt x1 y1 do punkt x2 y2
Putpixel (x, y, kolor)	Rysuje punkt w punktu „x,y” o danym kolorze
Getmaxx()	Pobiera rozmiar okna w poziomie
Getmaxy()	Pobiera rozmiar okna w pionie
Circle(x, y, r)	Rysuje okrąg o promieniu r

### Przykład 36

- Zainstaluj grafikę.
- Przepisz tabelę z poleceniami graficznymi.
- Wykonaj program poniżej.

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

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    initwindow(400,300);
    line(20,20,100,100);
    while(!kbhit());
    closegraph();
    return EXIT_SUCCESS;
```

}

### **ZADANIE 53**

Narysować biały kwadrat składający się z czterech linii w trybie graficznym o boku  $100+2*(\text{numer\_z\_dziennika})$  oraz podpisać, jaka to figura. (czcionką wielkości około 2 centymetry, kolor czerwony). Wykorzystaj opis funkcji graficznych w pliku opis\_funkcje\_graficzne\_cpp.pdf.

### **ZADANIE 52a**

Narysować w trybie graficznym trójkąt równoboczny o boku  $200+5*(\text{numer z dziennika})$  pikseli oraz oznaczyć jego wierzchołki literami A B C wraz z współrzędnymi punktu np. A(230,67) .

W zeszycie narysuj układ współrzędnych (tylko jedna ćwiartka). Pamiętaj, że punkt (0,0) jest w górnym lewym rogu układu(ekranu) i strzałki osi będą skierowane na dół. Narysuj trójkąt. Obok wierzchołków zapisz ich współrzędne. Do określenia współrzędnych wierzchołków musisz posłużyć się wzorem na wysokość w trójkącie równobocznym. Wzór ten zawiera pierwiastek dlatego pamiętaj o podłączeniu funkcji matematycznych do programu poprzez `#include <math.h>`

### **ZADANIE 53**

Narysować kółka olimpijskie w trybie graficznym. Sprawdź ułożenie tych kółek oraz kolory. Wykorzystaj opis funkcji graficznych w pliku opis\_funkcje\_graficzne\_cpp.pdf.

### **PRZYKŁAD 37**

Wykorzystując instrukcję graficzną LINE(X1,Y1,X2,Y2) z czterema parametrami X1,Y1 – współrzędne początku odcinka oraz X2,Y2 współrzędne końca odcinka zdefiniować procedurę KWADRAT(X,Y,D) X,Y – współrzędne lewego górnego rogu kwadratu D–długość boku kwadratu. Treść tego przykładu wykorzystaj w następnym zadaniu. Po wpisaniu programu zmodyfikuj jego treść tak, aby otrzymać trzy kwadraty w różnych miejscach o różnych wielkościach.

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

```
using namespace std;
void kwadrat(int x, int y, int d)
{
    line(x,y,x+d,y);
    line(x,y,x,y+d);
    line(x+d,y,x+d,y+d);
    line(x,y+d,x+d,y+d);
}

int main(int argc, char *argv[])
{
    initwindow(500,500);
    kwadrat(100,100,200);
    while(!kbhit());
    closegraph();
    return EXIT_SUCCESS;
}
```

### **ZADANIE 54**

Wykorzystując instrukcję graficzną LINE(X1,Y1,X2,Y2) z czterema parametrami X1,Y1 – współrzędne początku odcinka oraz X2,Y2 współrzędne końca odcinka zdefiniować procedurę TROJKAT(X,Y,D) X,Y – współrzędne górnego wierzchołka trójkąta D–długość boku trójkąta. Wykorzystaj treść przykładu poprzedniego w tym zadaniu. Wywołaj trzy razy procedurę TROJKAT tak, aby otrzymać trzy różne Trójkąty w różnych miejscach ekranu.

### **ZADANIE 55**

Zdefiniować procedurę ROMB(X,Y,D,ALFA)

X,Y – współrzędne dowolnego wierzchołka rombu D– długość boku rombu. Wywołaj trzy razy procedurę ROMB tak aby otrzymać trzy różne romby w różnych miejscach ekranu.

## **Wysyłanie danych na drukarkę**

Funkcja fprintf()

Pozwala wyprowadzać dane nie tylko na monitor, ale także na drukarkę.

Program przesyła tekst na drukarkę.

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

```
int main(void)
{
    clrscr();
    fprintf(stdout, "Drukuje...\n");
    fprintf(stdprn, "Pierwsza proba drukowania\n");
    fprintf(stdprn, "Autor: .....");
    fprintf(stdout, "Koniec drukowania.");
    fprintf(stdout, "Skonczylem, naciśnij cosik...");
    getch();
    return 0;
}
```

## **ZADANIE 58**

```
int x=1; //deklaracja zmiennej int w tradycyjny sposób
int *wskaznik; //deklaracja wskaźnika na typ
int wskaznik = &x; //przypisanie adresu zmiennej wskaźnikowi
*wskaznik = 99; //zapis równoważny z "x=99;"
```

W drugiej linii zadeklarowaliśmy wskaźnik na typ int. Zatem jak można się domyślać deklaracja wskaźnika wygląda następująco:

```
int *pA;
```

Aby wskaźnik wskazywał na początek tablicy A[12], musimy go jeszcze zainicjować:  
pA = &A[0];

Zapisy:

```
*pA[0]      *pA;      A[0]
*(pA[0]+1)  *(pA+1)   A[1]
*(pA[0]+2)  *(pA+2)   A[2]   itd.
```

są równoważne i oznaczają kolejne wyrazy tablicy A[ ].

Jeśli A jest nazwą tablicy, to zapis:

\*A

oznacza wskazanie do początku tablicy A, a zapisy:

```
*(A+1)      *(pA+1)   A[1]
*(A+8)      *(pA+8)   A[8] itd.
```

### Uwagi:

#### Uwaga 1

Pomiędzy wskaźnikami a tablicami istnieje bardzo ścisły związek. Do ponumerowania (określenia) elementów w tablicy służą tzw. indeksy.

#### Uwaga2

Każda operacja korzystająca z indeksów może zostać wykonana przy pomocy wskaźników; posługiwanie się wskaźnikiem zamiast indeksu na ogół przyspiesza operację.

### Przykład

Program, który nada wartości tablicy jednowymiarowej o wymiarze 10. Wpisz cztery liczby dodatnie i cztery ujemne. Kompilator domyślnie zapisze dwa zera. Program wypisze wartości tablicy na ekran oraz policzy ilość wyrazów większych mniejszy i równych zero.

```
# include "stdio.h"
# include <conio.h>
int a[ ][2]={ {1,2},{3,4},{5,6},{7,8},{9,10},{11,12} };
char b[ ]={ "Poniedzialek" };
int i;
int *pa;
char *pb;

void main()
{
    pa = &a[0][0];
    pb = b; // lub pb = b[0];
    clrscr();
    for (i=0; i<12; i++)
        printf("%d\t%c\n", *(pa+i), *(pb+i));
    getch();
}
```

```
}
```

## Rozwiązania

### Zadanie 9

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{ int x;
  scanf("%d",&x);
  printf("wczytana liczba x=%d",x);
  printf("\nkwadrat liczby x=%d",x);
  printf("    %d",x);
  printf("^2=%d",x*x);
  printf("\nszescian liczby x=%d",x);
  printf("    %d",x);
  printf("^3=%d",x*x*x);
  printf("\n");
  printf("    %x",x);
  printf("\n");
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

To samo zadanie co wyżej czyli Zadanie 9

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

using namespace std;

int main(int argc, char *argv[])
{
  int x_kle = 0;

  cout << "Podaj x:";
  scanf("%d", &x_kle);
  cout << "\n";

  printf("wczytana liczba x=%d\n", x_kle);
  printf("kwadrat liczby x=%d  %d^2=%d\n", x_kle,x_kle,x_kle*x_kle);
  printf("szescian liczby x=%d  %d^3=%d\n", x_kle,x_kle,x_kle*x_kle*x_kle);

  system("PAUSE");
  return EXIT_SUCCESS;
}
#include <cstdlib>
#include <iostream>
using
namespace std;
int main()
{
```



```

int a,b,wylosowana;
cout
<<"Program losuje liczbe z podanego przedzialu" <<endl;
cout
<<"Podaj poczatek przedzialu\n";
cin >>a;
cout <<"Podaj
koniec przedzialu\n";
cin >>b;
srand(time(NULL));
wylosowana=rand()%(b-a+1)+a;
cout
<<"Komputer wylosowal liczbe z przedzialu <" <<a <<";"
<<b <<">" <<endl;
cout <<"Wylosowana liczba to "
<<wylosowana <<endl;
cout
<<endl;
system("PAUSE");
return EXIT_SUCCESS;
}

```

żeby przywrócić poprzedni stan czcionki po prostu wybierasz zwykły kolor

```

HANDLE hOut;
hOut = GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(hOut,BACKGROUND_WHITE)

```

W linijce

```

SetConsoleTextAttribute(hOut,BACKGROUND_WHITE);

```

ma być

```

SetConsoleTextAttribute(hOut,FOREGROUND_WHITE);

```

### 3. Jak korzystać z nowej biblioteki

Wykorzystywanie bibliotek jest już prostą sprawą. W tym celu wystarczy wpisać na początku pliku:

```

#include "console.ddt"
using namespace ddt::console;

```

Jeśli plik nie znajduje się w tym samym katalogu co kod Twojego programu, kompilator pokaże błąd kompilacji i poinformuje Cię, że nie mógł znaleźć określonego pliku.

#### 9.4. Przestrzeń nazw

Wszystkie funkcje, jakie zostały umieszczone w tej bibliotece znajdują się w przestrzeni nazw **ddt::console**. Oznacza to, że jeśli chcesz skorzystać jakiegokolwiek funkcję z tego programu musisz ją poprzedzić zapisem **ddt::console**, czyli **ddt::console::nazwa\_funkcji()**. Jeśli nie chcesz pisać co chwilę tak długiej nazwy, wystarczy że dodasz na początku programu zapis **using namespace ddt::console;**, przez co Twoje wywołanie funkcji skróci się do zapisu **nazwa\_funkcji()**.

#### 9.5. Poznajemy polecenia konsoli

Ponieważ obsługa konsoli pod Windowsem nie należy do najłatwiejszych postanowiłem, że utworzę do Twojego użytku bibliotekę, która będzie dawała Ci możliwość korzystania z funkcji w tak samo prosty sposób w Dev-C++ jak programiści używający platformy środowiska programowania.

##### 9.5.1. void clrscr(void);

Funkcja **clrscr** służy do czyszczenia całego ekranu konsoli. Ekran zostanie wyczyszczony na aktualnie ustawiony kolor tła. Domyślnym kolorem tła zazwyczaj jest kolor czarny, więc jeśli chcesz uzyskać inny kolor, będziesz musiał najpierw wywołać funkcję odpowiedzialną za zmianę koloru tła. Zapis **void clrscr(void)**; oznacza tyle, że funkcja o nazwie **clrscr** nie zwraca żadnej wartości i nie przyjmuje żadnych parametrów wejściowych.

##### 9.5.2. void gotoxy(int x, int y);

Funkcja **gotoxy** służy do ustawiania migającego kursora w wybranym miejscu na ekranie. Lewy górny narożnik ma współrzędne (1,1), natomiast prawy dolny ma współrzędne (80,25). Słowo **void** oznacza, że funkcja nie zwraca żadnej wartości. Parametrami nazywamy zmienne umieszczone między nawiasami. W tym przypadku są to dwie zmienne typu int. W pierwszym polu podajemy współrzędną X, a w drugim współrzędną Y. Wywołanie takiej funkcji może wyglądać na przykład tak:

```

ddt::console::gotoxy(70,11); //kolumna=70; wiersz=11;

```

##### 9.5.3. int wherex(void);

Jeśli chcemy pobrać aktualną kolumnę w której znajduje się kursor w konsoli, wykorzystujemy do tego funkcję **wherex**. Funkcja zwraca liczbę, która określa pozycję kursora w osi X. Parametr **void** oznacza, że funkcja nie przyjmuje żadnych parametrów wejściowych.

#### 9.5.4. int wherey(void);

Jeśli chcemy pobrać aktualny wiersz w której znajduje się kursor w konsoli, wykorzystujemy do tego funkcję **wherey**. Funkcja zwraca liczbę, która określa pozycję kursora w osi Y. Parametr **void** oznacza, że funkcja nie przyjmuje żadnych parametrów wejściowych.

#### 9.5.5. void textattr(int kolor);

Kolejną funkcją, którą zamieściłem w tej bibliotece to **textattr**. Służy ona do ustawiania koloru tła i czcionki, który będzie pisany. Funkcja nie zwraca żadnej wartości. Jedynym parametrem jaki funkcja przyjmuje jest to liczba typu **int** reprezentująca kolor tła i tekstu. Maksymalna ilość kolorów dla tekstu i dla tła wynosi 16. Pierwszy kolor to 0 ostatni kolor to 15 zarówno dla tła jak i dla czcionki. Aby ustawić odpowiednie kolory tła i tekstu posługujemy się wzorem:

```
int kolorTla=1; //min wartosc=0; max wartosc=15;
int kolorTekstu=14; //min wartosc=0; max wartosc=15;
dtdt::console::textattr(kolorTla*16+kolorTekstu);
```

#### 9.5.6. void textcolor(int kolorTekstu);

Jeśli mamy potrzebę zmienić tylko kolor tekstu, możemy do tego celu wykorzystać funkcję **textcolor**. Funkcja nie zwraca żadnej wartości. Jedynym parametrem, jaki przyjmuje to liczba typu **int** reprezentująca nowy kolor czcionki.

#### 9.5.7. void textbackground(int kolorTla);

Jeśli mamy potrzebę zmienić tylko kolor tła, możemy do tego celu wykorzystać funkcję **textbackground**. Funkcja nie zwraca żadnej wartości. Jedynym parametrem, jaki przyjmuje to liczba typu **int** reprezentująca nowy kolor tła.

### Zadania 16

1. Napisać i uruchomić program wykonujący konwersję liczb ósemkowych na dziesiętne i odwrotnie.
2. Przy pomocy pojedynczego wywołania funkcji printf() wydrukować kilka złożonych napisów typu:  
\* suma 2+4 to 6  
\* działanie 5\*7\*27+6-873 daje wynik.....

### Rozwiązanie zadania 26

2. #include <cstdlib>

```
#include <iostream>
```

```
using namespace std;
```

```
int main(int argc, char *argv[])
{
    float a, sigma=0;
    for (;;) // nieskończona pętla
    {
        printf("\n Podaj liczbę do sumowania\n");
        scanf("%f", &a);
        if (a<0) continue;
        if (a==0) break;
        sigma+=a; // inaczej sigma = sigma + a;
        printf("\n SUMA CZESCIOWA: %f",sigma);
    }
    printf("Nastapil BREAK \n");
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

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

```
#include <math.h>

using namespace std;

int main(int argc, char *argv[])
{
    initwindow(400,300);
    line(100,100,200,100);
    line(150,100-50*sqrt(3),200,100);
    line(150,100-50*sqrt(3),100,100);
    while(!kbhit());
    closegraph();

    return EXIT_SUCCESS;
}
```

### Przykład:

Program poniżej generuje dźwięki i "odlicza".

[P036.CPP]

```
#include <dos.h>
#include <stdio.h>

void main()
{
    int czestotliwosc=5000, n=10, milisekundy=990;
    printf("\n");
    start:
    {
        sound(czestotliwosc);
        delay(milisekundy);
        nosound();
        czestotliwosc/=1.2;
        printf("%d\b", --n);
        if (n) goto start; //petle strukturalne zrob sam(a)
    }

    koniec ;
} // Tu jest instrukcja pusta.
```

Odp:

Przykład 13

```
// honda.kola=5; // teraz honda bedzie miala 5 kol
// gdy pole kola jest public
cout<<"podaj ile kol ma motor honda=";
// cin>>honda.kola; //mozesz wczytac ilosc kol do pola kola
// gdy pole kola jest public
```

Zad 14

```
#include <cstdlib>
```

```

#include <iostream>

using namespace std;

class Tsamochod_nie
{
private:
    string kolor_ni, zmianowy_kolor_ni;
    int rok_ni, wiek_ni;
    float predkosc_ni, ilosc_paliwa_ni, zmianowa_predkosc_ni, pelny_bak_ni, spalanie_ni;
public:
    Tsamochod_nie()
    {
        rok_ni = 2001;
        kolor_ni = "czerwony";
        predkosc_ni = 61;
        ilosc_paliwa_ni = 21;
        pelny_bak_ni = ilosc_paliwa_ni;
        wiek_ni = 11;
    }
    Tsamochod_nie(int rokk_ni, string kolorr_ni, float predkoscc_ni, float ilosc_paliwaa_ni);
    ~Tsamochod_nie()
    {
        cout << "Zezłomowałeś go!" << endl;
    }
    void pokaz_dane()
    {
        cout << "Rok produkcji: " << rok_ni << endl;
        cout << "Wiek samochodu: " << wiek_ni << endl;
        cout << "Kolor samochodu: " << kolor_ni << endl;
        cout << "Prędkość samochodu: " << predkosc_ni << endl;
        cout << "Ilość paliwa: " << ilosc_paliwa_ni << endl;
    }
    void starzenie_sie()
    {
        wiek_ni++;
        cout << "Samochód się postarzał o rok!" << endl;
    }
    void zmiana_koloru()
    {
        cout << "Podaj kolor na jaki chcesz przemalować auto: ";
        cin >> zmianowy_kolor_ni;
        kolor_ni = zmianowy_kolor_ni;
        cout << endl << "Przemalowałeś auto! Teraz jest koloru: " << kolor_ni << endl;
        zmianowy_kolor_ni = "";
    }
    void zmiana_predkosci()
    {
        cout << "Podaj jaką prędkość ma osiągać teraz Twoje auto: ";
        cin >> zmianowa_predkosc_ni;
        predkosc_ni = zmianowa_predkosc_ni;
        cout << endl << "Zmienileś prędkość auta! Teraz osiąga ona prędkość " <<
predkosc_ni << " km/h." << endl;
    }
    void palenie_paliwa()
    {
        spalanie_ni = ilosc_paliwa_ni/2;
        ilosc_paliwa_ni = ilosc_paliwa_ni - spalanie_ni;
    }

```

```

        cout << "Jadąc spaliłeś " << spalanie_ni << " litrów obecnej ilości benzyny w baku.
Teraz w baku masz " << ilosc_paliwa_ni << " litrów." << endl;
    }
    void tankowanie_ni()
    {
        ilosc_paliwa_ni = pelny_bak_ni;
        cout << "Zatankowałeś! Teraz w Twoim baku jest " << ilosc_paliwa_ni << " litrów
benzyny." << endl;
    }
};

Tsamochod_nie::Tsamochod_nie(int rokk_ni, string kolorr_ni, float predkoscc_ni, float ilosc_paliwaa_ni)
{
    rok_ni = rokk_ni;
    kolor_ni = kolorr_ni;
    predkosc_ni = predkoscc_ni;
    ilosc_paliwa_ni = ilosc_paliwaa_ni;
    pelny_bak_ni = ilosc_paliwa_ni;
    wiek_ni = 2012-rokk_ni;
}

int main(int argc, char *argv[])
{
    system("chcp 1250");
    cout << endl;
    Tsamochod_nie samochod_niec;
    Tsamochod_nie samochod_nie(1900,"biały",100,32);
    Tsamochod_nie samochod_ni(1999,"czarny",120,40);

    cout << "Utworzyłeś 1 auto!" << endl << endl << endl;

    samochod_niec.pokaz_dane();
    samochod_niec.starzenie_sie();
    samochod_niec.zmiana_koloru();
    samochod_niec.zmiana_predkosci();
    samochod_niec.palenie_paliwa();
    samochod_niec.tankowanie_ni();
    samochod_niec.pokaz_dane();
    samochod_niec.starzenie_sie();
    samochod_niec.pokaz_dane();
    samochod_niec.~Tsamochod_nie();

    cout << "Utworzyłeś 2 auto!" << endl << endl << endl;

    samochod_nie.pokaz_dane();
    samochod_nie.starzenie_sie();
    samochod_nie.zmiana_koloru();
    samochod_nie.zmiana_predkosci();
    samochod_nie.palenie_paliwa();
    samochod_nie.tankowanie_ni();
    samochod_nie.pokaz_dane();
    samochod_nie.starzenie_sie();
    samochod_nie.pokaz_dane();
    samochod_nie.~Tsamochod_nie();

    cout << "Utworzyłeś 3 auto!" << endl << endl << endl;

    samochod_ni.pokaz_dane();
    samochod_ni.starzenie_sie();
    samochod_ni.zmiana_koloru();
    samochod_ni.zmiana_predkosci();
    samochod_ni.palenie_paliwa();

```

```

samochod_ni.tankowanie_ni();
samochod_ni.pokaz_dane();
samochod_ni.starzenie_sie();
samochod_ni.pokaz_dane();
samochod_ni.~Tsamochod_nie();

system("PAUSE");
return EXIT_SUCCESS;
}

```

### Zadanie

Uzupełnij tabelkę

Zapis tradycyjny	Zapis skrócony pierwszy	Zapis skrócony drugi
zmienna = zmienna + zmienna1	zmienna+=zmienna1	
p=p+1	p+=1	p++
zmie=zmie+5	zmie+= 5	
?	ujka - = 1	?
jj = jj % 2	?	
?	u*=10	
hitrus= hitrus/4	?	

### ZADANIE 56

- Zmienne z trzema literami nazwiska.

Pewien przedsiębiorca pożyczył od banku dużą kwotę pieniędzy. Bank określił warunki spłaty pojedynczej raty na minimum 1 tys złotych miesięcznie. Przedsiębiorca chce szybciej spłacić kredyt, dlatego postanowił, że każda spłata raty miesięcznej będzie o pół tysiąca złotych wyższa od poprzedniej. Napisz program z użyciem pętli, który obliczy ile będzie wynosiła rata po okresie półtorarocznym spłat.

Przykładowy wydruk (działania programu, tylko jeden napis dla całego programu): Wartość ostatniej raty to: np. 2.5 tys zł.

*Wskazówka:* zmienna sterująca pętlą nie musi być typu int, może być liczbą rzeczywistą

### Zadanie → zadanie domowe

Zapisz w zeszycie odpowiedzi na następujące pytania. Przepisz najpierw pytanie następnie udziel odpowiedzi.

- dla jednej instrukcji wykonywanej przez pętlę,
- dla wielu instrukcji wykonywanej przez pętlę,
- przebieg wykonania pętli for,
- zapisz przykłady instrukcji for  
-pełna instrukcja for

- z wyrażeniem inicjującym obliczonym poza pętlą
- z obliczanym warunkiem przzerwania pętlę
- e) kiedy pętla nie będzie wykonana
- f) napisz na podstawie przykładu 14 instrukcję, która wydaje dźwięk
- g) napisz definicję tablicy, indeks lub zespół indeksów.
- h) narysuj tablicę jedno wymiarową oraz dwuwymiarową z oznaczeniami adresu wszystkich elementów.  
Jak zaczynamy liczyć numery komórek.

### Przykład

Założmy, że mamy plik, w którym dane mamy zapisane następująco:

10 dodac 5

3 odjac 2

77 zonk 3

99 dodac 1

Jak nietrudno zauważyć, w każdym wierszu podanego pliku znajduje się liczba, później wyraz i ponownie liczba. Taki plik możemy sobie w łatwy sposób wczytać przy pomocy operatora >>, a następnie wykonać na nim jakieś operacje, np. operacje matematyczne. Poniższy kod prezentuje w jaki sposób można wczytać powyższy plik oraz jak wykonywać na nim operacje. Kod źródłowy:

C/C++

```
#include <fstream>
```

```
#include <iostream>
```

```
#include <string>
```

```
void wykonajOperacje( int liczba1, string napis, int liczba2 )
```

```
{
    if( napis == "dodac" )
    {
        cout << liczba1 << " + " << liczba2 << " = " << liczba1 + liczba2 << endl;
    } else if( napis == "odjac" )
    {
        cout << liczba1 << " - " << liczba2 << " = " << liczba1 - liczba2 << endl;
    } else
    {
        cout << "Nieznana operacja \"" << napis << "\" - nie mozna wykonac obliczen." <<
endl;
    }
}
```

```
bool wczytajPlik( string nazwaPliku )
```

```
{
    ifstream plik;
    plik.open( nazwaPliku.c_str() );
    if( !plik.good() )
        return false;

    while( true ) //pętla nieskończona
    {
        int a;
        string b;
        int c;
```

```

    plik >> a >> b >> c;
    if( plik.good() )
        wykonajOperacje( a, b, c );
    else
        break; //zakończ wczytywanie danych - wystąpił jakiś błąd (np. nie ma więcej danych
w pliku)

    } //while
    return true;
}

int main()
{
    if( !wczytajPlik( "dane.txt" ) )
        cout << "Nie udało się otworzyć pliku!" << endl;

    return 0;
}

```

W wyniku działania powyższego programu, na standardowym wyjściu pojawią się następujące komunikaty:

10 + 5 = 15

3 - 2 = 1

Nieznana operacja "zonk" - nie można wykonać obliczeń.

99 + 1 = 100

Podsumowanie

W niniejszym rozdziale dowiedziałeś się, w jaki sposób można wczytywać dane z pliku przy pomocy operatora >>. W kolejnym rozdziale zajmiemy się obsługą błędów, jakie mogą wystąpić podczas wczytywania danych.

Zadanie domowe

Otwórz plik tekstowy, w którym będą znajdowały się tylko liczby całkowite, oddzielone od siebie spacjami. Następnie napisz program, który wczyta wszystkie liczby z pliku i wypisze sumę wszystkich liczb na standardowym wyjściu.

```

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    int zmienna1 = 213;
    int tablica[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    struct
    {
        int liczba;
        long long duzaLiczba;
    } struktura;
    cout << "Adres zmienna1=" <<&zmienna1 << endl << endl;
    cout << "Adres tablica=" <<&tablica << endl;
    cout << "Adres tablica[0]=" <<&tablica[ 0 ] << endl;
    cout << "Adres tablica[1]=" <<&tablica[ 1 ] << endl << endl;
    cout << "Adres struktura=" <<&struktura << endl;
}

```



```
cout << "Adres struktura.liczba=" <<&( struktura.liczba ) << endl;  
cout << "Adres struktura.duzaLiczba=" <<&( struktura.duzaLiczba ) << endl;  
getch();  
return( 0 );  
}
```