PROGRAMOWANIE 1

Marek Brancewicz

Wydział Fizyki
Uniwersytet w Białymstoku

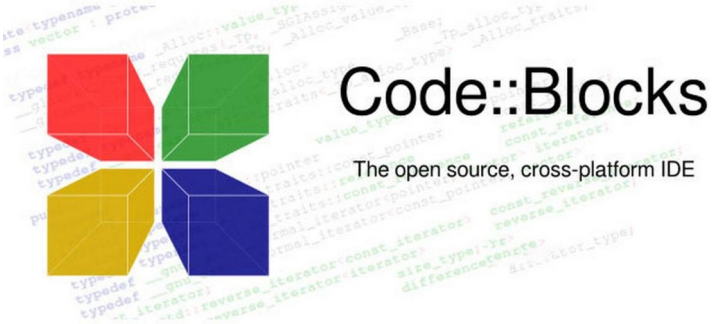
m.brancewicz@uwb.edu.pl

WYDZIAŁ FIZYKI
UNIWERSYTET W BIAŁYMSTOKU

1. Wprowadzenie

23.02.2024

1. Organizacja zajęć
2. Literatura i kursy on-line
3. Podstawowe pojęcia
4. CodeBlocks, pobieranie, instalacja



Literatura (PL & ENG)

[1] **mnóstwo kursów on-line !!!**

[2] W. Porębski, *Język C++ : wprowadzenie do programowania*, wyd. 2, Komputerowa Oficyna Wydawnicza "Help", Warszawa 1999

[3] J. Grębosz, *Symfonia C ++ standard : programowanie w języku C++ orientowane obiektowo*, Wydawnictwo "Edition 2000" : Oficyna Kallimach, Kraków 2005

[4] S. Prata, *Język C++*, wyd. 5, Wydawnictwo Helion, Gliwice 2006

[5] A. Koenig, *Accelerated C++ : practical programming by example*, 22nd printing, Addison-Wesley, Boston 2013

Język programowania

Zbiór zasad określających, kiedy ciąg symboli tworzy **program komputerowy** oraz jakie obliczenia opisuje.

Program komputerowy

Sekwencja symboli opisująca realizowanie obliczeń zgodnie z pewnymi regułami zwanymi **językiem programowania**. Program jest zazwyczaj wykonywany przez komputer (np. wyświetlenie strony internetowej), zwykle bezpośrednio, jeśli wyrażony jest w języku zrozumiałym dla danej maszyny lub pośrednio – gdy jest interpretowany przez inny program (**interpreter**). Program może być ciągiem instrukcji opisujących modyfikacje stanu maszyny, ale może również opisywać obliczenia w inny sposób (np. rachunek lambda)

Kod źródłowy

Zapis programu komputerowego przy pomocy określonego **języka programowania**, opisujący operacje, jakie powinien wykonać komputer na zgromadzonych lub otrzymanych danych. Kod źródłowy jest wynikiem pracy programisty i pozwala wyrazić w czytelnej dla człowieka formie strukturę oraz działanie **programu komputerowego**. Jest on zwykle zapisywany w pliku tekstowym.

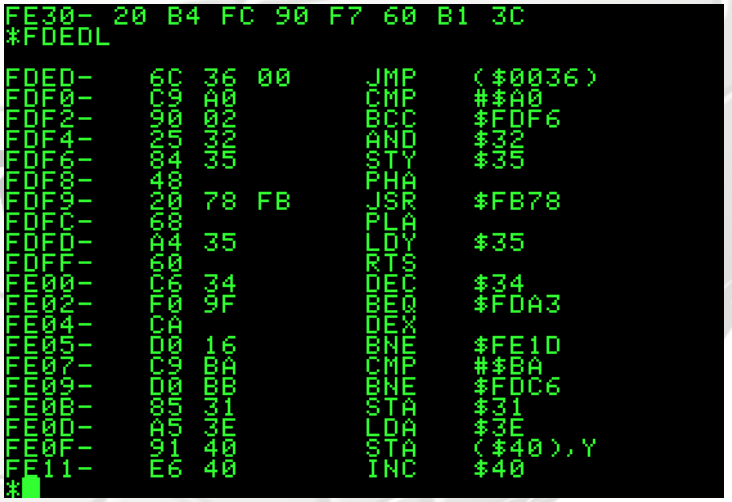
Kompilator

Program służący do automatycznego tłumaczenia kodu napisanego w jednym języku (**języku źródłowym**) na równoważny kod w innym języku (**języku wynikowym**). Proces ten nazywany jest kompilacją. W informatyce kompilatorem nazywa się najczęściej program do tłumaczenia **kodu źródłowego** w **języku programowania** na **język maszynowy**. Niektóre z nich tłumaczą najpierw do **języka asemblera**, a ten na język maszynowy jest tłumaczony przez asembler.

Język maszynowy

(kod maszynowy)

Zestaw rozkazów procesora, w którym zapis programu wyrażony jest w postaci liczb binarnych stanowiących rozkazy oraz ich argumenty.



Zintegrowane środowisko programistyczne

(**IDE**, od ang. integrated development environment) – program lub zespół programów (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.

Interpreter

Analizuje kod źródłowy programu, a przeanalizowane fragmenty wykonuje. Realizowane jest to w inny sposób niż w procesie **kompilacji**, podczas którego nie wykonuje się wejściowego programu (**kodu źródłowego**), lecz tłumaczy go do wykonywalnego **kodu maszynowego** lub kodu pośredniego, który jest następnie zapisywany do pliku w celu późniejszego wykonania.

Wykonanie programu za pomocą interpretera jest wolniejsze, a do tego zajmuje więcej zasobów systemowych niż wykonanie kodu skompilowanego, lecz może zająć relatywnie mniej czasu niż **kompilacja** i uruchomienie. Jest to zwłaszcza ważne przy tworzeniu i testowaniu kodu, kiedy cykl edycja-interpretacja-**debugowanie** może często być znacznie krótszy niż cykl edycja-kompilacja-uruchomienie-**debugowanie**.

Debugger

Program komputerowy służący do dynamicznej analizy innych programów, w celu odnalezienia i identyfikacji zawartych w nich błędów, zwanych z angielskiego bugami (robakami). Proces nadzorowania wykonania programu za pomocą debuggera określa się mianem **debugowania**. Debugger jest standardowym wyposażeniem większości współczesnych **środowisk programistycznych**.

2. Pierwszy program w C++.

23.02.2024

1. Pierwszy program w C++ (`"Hello world"`)
2. Struktura programu
3. Struktura projektu
4. Typy zmiennych, zmienne globalne i lokalne
5. Wczytywanie danych z klawiatury i wypisywanie ich na ekranie.

```cpp
#include <iostream>
using namespace std;
// tutaj zazwyczaj definicje zmiennych
int main()
{
	cout << "HELLO WORLD!" << endl;
	return 0;	// tutaj kod błędu
}
```

Struktura programu

- dyrektywy preprocesora (#), np: dołączanie bibliotek do programu (`#include <library>`), definicje stałych (`#define grav 9.81`)
- `using namespace std;` (przestrzeń nazw)
- definicje zmiennych (globalnych) i stałych (np. `int x,n;` `const float pi=3.1415926;`)
- funkcja główna (sterująca) – `int main() {}`
- `return 0;` (zwraca kody błędów)
- białe znaki (spacje, entery) są ignorowane
- każda linia powinna się kończyć średnikiem (`;`) (są wyjątki)
- komentarze: `//` (jedna linia) lub `/*` (początek) i `*/` (koniec)

Struktura projektu (CodeBlocks)

- main.cpp – plik tekstowy zawierający kod źródłowy
- plik *.cbp – plik projektu (CodeBlocks project) tekstowy w formacie xml zawierający wszystkie informacje o projekcie, np. kompilator, nazwa projektu, pliki wchodzące w skład projektu
- folder obj – zawiera plik pośredni (przejściowy) potrzebny do kompilacji (plik *.o)
- folder bin – zawiera plik wykonywalny aplikacji (*.exe)
- plik *.depend – plik zależności, tekstowy, zawierający informacje o bibliotekach
- plik *.layout – plik xml, przechowuje informacje o srukturze (układzie) projektu

Typy zmiennych

Nazwa	Rozmiar [B] (32 bit)	Rozmiar [B] (64 bit)	Zakres
bool	1	1	**false** or **true**
char	1	1	from -128 to 127
unsigned char	1	1	from 0 to 255
short	2	2	from -32 768 to 32 767
unsigned short	2	2	from 0 to 65 535
int	4	4	from -2 147 483 648 to 2 147 483 647
unsigned int	4	4	from 0 to 4 294 967 295
long	4	4	from -2 147 483 648 to 2 147 483 647
unsigned long	4	4	from 0 to 4 294 967 295
long long	8	8	from -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807
unsigned long long	8	8	from 0 do 18 446 744 073 709 551 615
float	4	4	3.4 E +/-38 (7 numbers)
double	8	8	1.7 E +/-308 (15 numbers)
long double	8	12	1.7 E +/-308 (15/20 numbers)
string	4	4	table (chain) of characters

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout<<"size of bool = "<<sizeof(bool)<<endl;
    cout<<"size of char = "<<sizeof(char)<<endl;
    cout<<"size of unsigned char = "<<sizeof(unsigned char)<<endl;
    cout<<"size of short = "<<sizeof(short)<<endl;
    cout<<"size of unsigned short = "<<sizeof(unsigned short)<<endl;
    cout<<"size of int = "<<sizeof(int)<<endl;
    cout<<"size of unsigned int = "<<sizeof(unsigned int)<<endl;
    cout<<"size of long = "<<sizeof(long)<<endl;
    cout<<"size of unsigned long = "<<sizeof(unsigned long)<<endl;
    cout<<"size of long long = "<<sizeof(long long)<<endl;
    cout<<"size of float = "<<sizeof(float)<<endl;
    cout<<"size of double = "<<sizeof(double)<<endl;
    cout<<"size of long double = "<<sizeof(long double)<<endl;
    cout<<"size of string = "<<sizeof(string)<<endl;
    return 0;
}
```

Zmienna globalna

Zmienna istniejąca przez cały czas życia programu i widziana z wielu miejsc w programie. W C++ każda zmienna globalna jest **zmienną statyczną**.

Zmienna lokalna

Zmienna zdefiniowana i dostępna wyłącznie w określonym bloku programu, tworzona w momencie wejścia do tego bloku oraz usuwana z pamięci w momencie wyjścia z danego bloku. Tym samym zasięg zmiennej lokalnej oraz czas jej życia pokrywają się i obejmują blok, w którym zmienna lokalna jest zdefiniowana. Zmienna lokalna ma więc określony, ograniczony zakres istnienia i dostępności.

To w jakich blokach programowych można tworzyć zmienne lokalne definiuje składnia konkretnego języka programowania. Typowymi blokami, w których można w różnych językach programowania tworzyć zmienne lokalne, są moduły, podprogramy oraz w pewnych językach programowania także instrukcje blokowe (lub inne instrukcje strukturalne, np. pętla for w języku C.

Zmienna lokalna w danym bloku przesłania zdefiniowaną zmienną globalną lub zmienną lokalną z bloku nadrzędnego o tym samym identyfikatorze. Tym samym programista nie może wprost, za pomocą danego identyfikatora, w bloku o zdefiniowanej zmiennej lokalnej, odwołać się do zmiennej zewnętrznej o tym samym identyfikatorze co zdefiniowana zmienna lokalna, choć może to zrobić za pomocą innych konstrukcji, jeżeli są dostępne w danym języku programowania, np. selekcja, wskaźnik, przemianowanie, nakładanie zmiennych lub inne.

Zadanie 1

(Wczytywanie i wyświetlanie liczb i tekstów)

Napisz prosty program wczytujący liczby (całkowite lub rzeczywiste) z klawiatury, wykonaj proste obliczenia na wczytanych danych i wypisz wynik działania na ekranie z odpowiednim opisem (komentarzem). W programie wykorzystaj deklarację zmiennych liczbowych i tekstowych.

3. Instrukcja warunkowa "if"

01.03.2024

Czyli jak nauczyć komputer podejmowania decyzji?

Przykłady:

1. Weryfikacja kodu dostępu (PIN)
2. Logowanie z nazwą użytkownika i hasłem

Typy zmiennych do wykorzystania:

`int` – liczba całkowita (np. `int PIN, pin, Pin;`)

`string` – tekstowa (np. `string password="a12b34c";`)

`bool` – logiczna, prawda (1) lub fałsz (0) (np. `bool c;`)

Operatory porównania (wyrażenia boolowskie):

`==`	logiczne porównanie	`(a==b)`
`<`	mniejsze niż	`(a<b)`
`>`	większe niż	`(a>b)`
`<=`	mniejsze lub równe	`(a<=b)`
`>=`	większe lub równe	`(a>=b)`
`!=`	nie równe	`(a!=b)`

Operatory logiczne:

&& - iloczyn logiczny "i"

|| - suma logiczna "lub"

! - zaprzeczenie logiczne (negacja) "nie"

Pytanie:

Jaka zmienna będzie odpowiednia do przechowywania PIN?

Ćwiczenie:

x=3, a=7

Określ wartość wyrażenia logicznego:

```
!((x<=5)||(x>12)||(a!=7)&&(a>15))
```

Składnia „if”

```
if (a>2)  // warunek logiczny 1
{
   // instrukcje 1, np.:
   cout << "a is greater than 2" << endl;
}
else if (a<2)   // warunek logiczny 2
{
   // instrukcje 2, np.:
   cout << "a is smaller than 2" << endl;
}
else   // pozostałe przypadki
{
   // instrukcje w pozostałych przypadkach. np.:
   cout << "a is equal 2" << endl;
}
```

Zadanie 2

(Sprawdzanie wieku kandydata na prezydenta)

Napisz program, który zapyta użytkownika o rok urodzenia i na tej podstawie określi jego wiek. W zależności od wieku kandydata program ma sprawdzić czy jest on osoba pełnoletnią (18+) i czy może kandydować na stanowisko prezydenta (35+). W zależności od wieku kandydata program ma wyświetlić odpowiedni komentarz. Można rozbudować program o inne opcje związane z wiekiem.

Zadanie 3 (wysłać)

(Rozwiązywanie równania kwadratowego)

Napisz program do rozwiązywania równania kwadratowego o postaci: $\mathbf{ax^2+bx+c=0}$, na podstawie wczytywanych parametrów **a**, **b** i **c**. Zastosuj instrukcję warunkową „if” w celu wykonania odpowiednich obliczeń i komentarzy w zależności od wartości parametru *delta*. W razie potrzeby wykorzystaj bibliotekę `<cmath>`.

Sprawdzanie poprawności danych strumienia wejściowego

```
cin.fail();
```

sprawdza poprawność strumienia wejściowego, zwraca PRAWDA, jeżeli znaku wczytany ze strumienia jest nieprawidłowy

```
cin.clear();
```

czyszczenie flag błędu

```
cin.ignore();
```

```
cin.ignore(1000,'\n');
```

ignorowanie zawartości strumienia; jeden wyraz lub 1000 znaków do końca linii

Zadanie*

(*pomysł do zrealizowania dopiero po wprowadzeniu funkcji)

Napisać funkcję sprawdzającą czy wprowadzona dana w zależności od opcji jest: tekstem, liczbą całkowitą, liczbą zmiennoprzecinkową. Wynikiem powinna być prawda lub fałsz.

4. Pętle

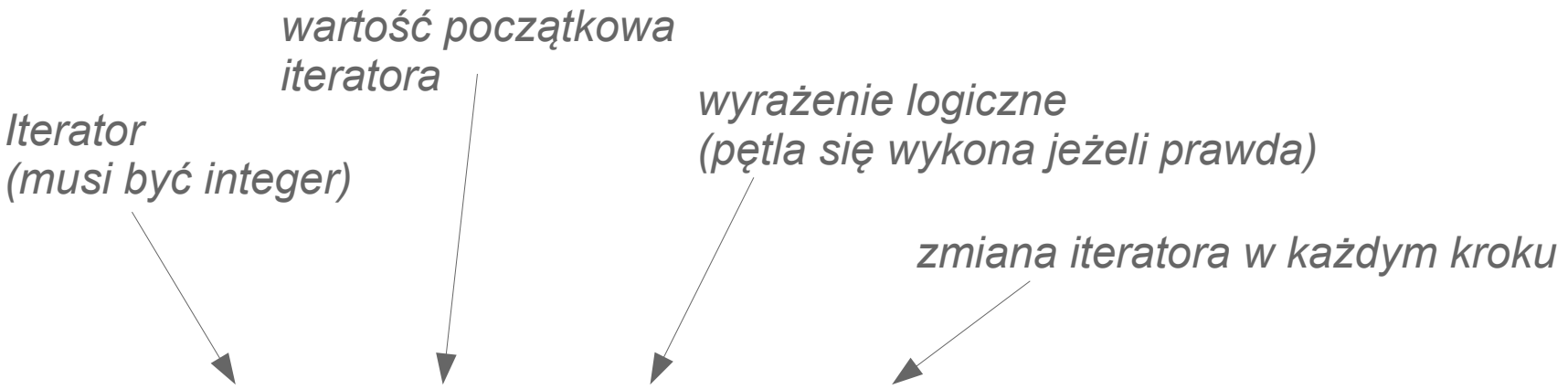
08.03.2024

- **pętla** – instrukcja powtarzalna (iteracyjna)
- **iteracja** – pojedyncze wykonanie pętli
- **iterator** – licznik (przechowuje numer iteracji)

Pętle w C++:

- `for`
- `while`
- `do ... while`

Składnia pętli ”for” (kontrolowana iteratorem)



Inkrementacja: `i++` to samo co `i=i+1` oraz `i+=1`

Dekrementacja: `i--` to samo co `i=i-1` oraz `i-=1`

Zadanie 4

Stosując podane poniżej rozwiązania, napisz program, który będzie udawał odliczanie do startu rakiety (z dźwiękiem). Program powinien odliczać od 10 do 1 co 1 sekundę i napisać "START", gdy licznik dojdzie do 0.

```
#include <windows.h>     // windows
#include <unistd.h>      // linux
Sleep(1000)              // czekaj 1000 ms
Beep(2000,500)           // dźwięk (f[Hz], t[ms])
system("cls")            // windows
system("clear")          // linux
```

Składnia pętli ”while”, ”do … while”

(kontrolowane warunkiem logicznym, nie posiadają iteratora)

```
while (log. condition)

{

    instructions;

}
```

Zestaw instrukcji może się nie wykonać ani razu.

```
do

{

    instructions;

} while (log. condition);
```

Zestaw instrukcji wykona się przynajmniej raz.

W obu przypadkach zestaw wykonywanych instrukcji powinien wpływać na zmianę warunku logicznego, który zakończy wykonywanie pętli. W innym przypadku pętle nie skończą się, co może być pożądanym efektem (wyjątkowo).

Zagnieżdżanie pętli (petla w pętli)

Można zagnieżdżać różne rodzaje pętli wewnątrz innych pętli.

`break;` //przerwanie wykonywania pętli, jeśli wystąpi w pętli wewnętrznej to tylko ta zostanie przerwana

```
j=1;
while (j<5)
{
    i=1;
    while (i<10)
    {
        cout<<"*";
        if (i==5)
            break;
        i++;
    }
    cout<<endl;
    j++;
}
```

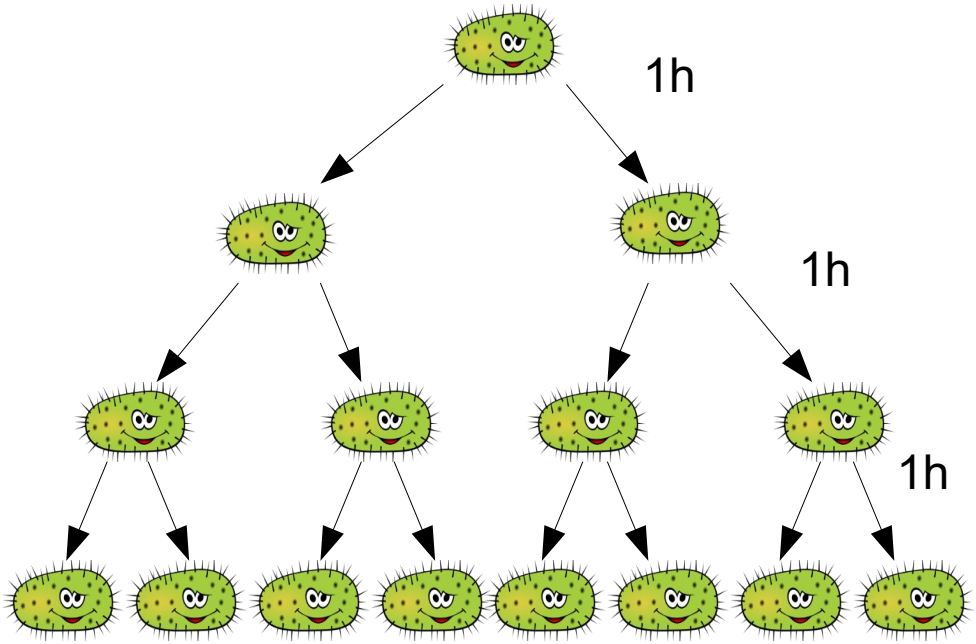
`continue;` //przerywa wykonywanie danej iteracji pętli, program przechodzi do kolejnej iteracji (tej samej pętli)

```
i=0;
while (i<5)
{
    i++;
    if (i==3)
        continue;
    cout<<i;
}
```

Zadanie 5

Rysunek przedstawia przyrost populacji bakterii w czasie. Napisz program, który policzy ile godzin potrzeba, aby populacja bakterii powiększyła się od jednego organizmu do 1 000 000 000. Zastosuj pętlę "while".

```
0 h – population: 1
1 h – population: 2
2 h – population: 4
3 h – population: 8
...
```



Zadanie 6

a) Napisz program, który zapyta użytkownika o liczbę dodatnią. Jeżeli użytkownik poda liczbę ujemną lub zero, program powinien poprosić ponownie o podanie dodatniej liczby. Program powinien powtarzać próbę wczytania dodatniej liczby do momentu, gdy użytkownik taką liczbę poda. Zastosuj pętlę „`do...while`”.

b) Napisz program, który zapyta użytkownika o liczbę rzeczywistą. Jeżeli użytkownik wprowadzi błędną zmienną do strumienia wejściowego to program ma zapytać ponownie o podanie liczby rzeczywistej. Program ma działać w pętli aż do momentu, gdy użytkownik wprowadzi prawidłową liczbę rzeczywistą.

5. Liczby (pseudo)losowe

15.03.2024

- Kompilator GCC posiada wbudowane funkcje generujące liczby losowe (np. rand), ale jak one działają?

- Czas w komputerze (*Unix time*, *POSIX time*) - liczba sekund od 01.01.1970 i cały czas się zmienia

- Jak zamienić tą liczbę (np. 1 234 567 890) na „losową” liczbę, powiedzmy od 1 do 100?

- Zastosować operację reszty z jej dzielenia przez 100; wynik będzie od 0 do 99 (np.:1229 % 100 = 29), potem można dodać 1.

Obliczanie kolejnych liczb losowych

$x_1=(s+b)\%n$ – zaczynamy od liczby zależnej od czasu

$x_2=(s+x_1+b)\%n$

$x_3=(s+x_2+b)\%n$ ... i tak dalej ...

x_i – kolejna liczba (pseudo)losowa

s – Unix time

n – liczba wylosowanych liczb lub zakres (0 ... n-1)

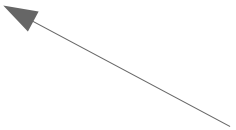
b – jakaś stała (np. 1)

Funkcje

```
#include<cstdlib>

rand()%n;              // liczba losowa od 0 do n-1
rand()%n+1;            // liczba losowa od 1 do n

// przykladowy zakres od 51 do 75 (25 liczb):
rand()%25;             // liczba losowa od 0 do 24
rand()%25+51;          // liczba losowa od 51 do 75
```



Ile jest liczb całkowitych od 51 do 75

Randomizacja

```
#include<time.h>   // (or <ctime>)


srand(time(NULL)); // rozpoczyna randomizacje z
wykorzystaniem czasu komputera, stosujemy raz w
programie, przed pierwszym uzyciem funkcji rand()


RAND_MAX           // maksymalna liczba losowa
```

Zadanie 7

Napisz program, który podobnie jak w losowaniu LOTTO wybierze losowo 6 liczb spośród 49 i wypisze je na ekranie. Zastanów się czy program faktycznie działa jak losowanie LOTTO i dlaczego? Popraw program tak aby działał jak losowanie LOTTO

Zadanie 8

Napisz program (grę), który wylosuje liczbę od 1 do 100 i poprosi użytkownika o jej zgadnięcie. Po próbie odgadnięcia program powinien wyświetlić odpowiedni komentarz, np.: „Zgadłeś”, „Za duża” lub „Za mała”. Program powinien pamiętać numer próby i po odgadnięciu wypisać go na ekranie wraz ze słowem „Zgadłeś”. Zastosuj pętlę „while” („do … while”) oraz instrukcję warunkową „if”.

Zadanie 9 (wysłać)

Napisać program do generowania liczb rzeczywistych z zakresu, który użytkownik poda z klawiatury. Wykorzystaj metodę generowania liczb całkowitych poznaną na zajęciach w połączeniu z odpowiednimi operacjami matematycznymi.

6. Instrukcja wielokrotnego wyboru „switch case”

22.03.2024

Składnia „switch case”

```
switch (choice)
{
    case 1:
    {
        // instrukcje gdy choice==1
    }
    break;  // koniecznie!
    case 2:
    {
        // instrukcje gdy choice==2
    }
    break;  // koniecznie!
    case 3:
    {
        // instrukcje gdy choice==3
    }
    break;  // koniecznie!
    default:
    {
        // instrukcje gdy choice jest inne niż wyżej
    }
}
```

zmienna kontrolna:

int
char
~~**float**~~
~~**string**~~

Użyteczne

```
break;
```
zatrzymuje pętlę i przechodzi dalej

```
exit(0);
```
(#include <cstdlib>)
kończy program i zwraca 0

```
system("cls");
```
(#include <cstdlib>)
czyści ekran

```
getch();
```
(#include <conio.h>)

Czeka na wciśnięcie klawisza. Podobne do getchar() z <stdio.h> ale nie czeka na enter.

Nieskończone pętle

```
while(true);
for(;;);
```

Pomiar czasu

```
#include<ctime>

float sum=0, add=1, elapsed;
clock_t start, stop;                        // zmienne clock_t
int iterations=1000*1000*1000;

int main ()
{
    start=clock();                          // start pomiaru czasu
    for (int i=0;i<iterations;i++)
    {
        sum+=add;
        add/=2.0;
    }
    stop=clock();                           // stop pomiaru czasu
    elapsed=float(stop-start)/CLOCKS_PER_SEC;
    cout<<"Time measured: "<<elapsed<<endl;
    return 0;
}
```

Zadanie 10

Napisz program – kalkulator, z wykorzystaniem instrukcji wielokrotnego wyboru „switch case”. Program będzie wyświetlał menu główne z dostępnymi operacjami matematycznymi. Spraw, żeby program działał w pętli, dopóki użytkownik nie wybierze opcji WYJŚCIE z menu.

Zadanie 11 (wysłać)

Napisz program do obliczania przybliżonej wartości liczby PI metodą Monte Carlo. Algorytm: 1) Wylosuj *n* współrzędnych punktów (par liczb) wewnątrz kwadratu o boku 2*r* (*r*-dowolne) i środku w początku układu współrzędnych. 2) Sprawdź ile z nich (*m*) mieści się wewnątrz okręgu o promieniu *r* i środku w początku układu współrzędnych. 3) Zauważ, że stosunek pól tych figur powinien być taki jak stosunek odpowiednich ilości punktów wewnątrz kwadratu i okręgu i można z niego łatwo policzyć wartość PI. Spraw, aby program działał w pętli licząc PI dla liczb *n*=10, 100, 1000 ..., czyli dla kolejnych potęg 10. Ustaw liczbę tych pętli (potęg) tak, aby czas działania programu nie przekraczał minuty (około). Na ekranie wypisuj: nr iteracji, liczbę wylosowanych punktów *n,* obliczone PI, względną różnicę między rzeczywistą liczbą PI a obliczoną, czas trwania iteracji.

7. Tablice (macierze)

05.04.2024

Tablica – zmienna złożona reprezentująca uporządkowany (ponumerowany) zestaw zmiennych jednego typu

deklaracja: `int dane[10];`

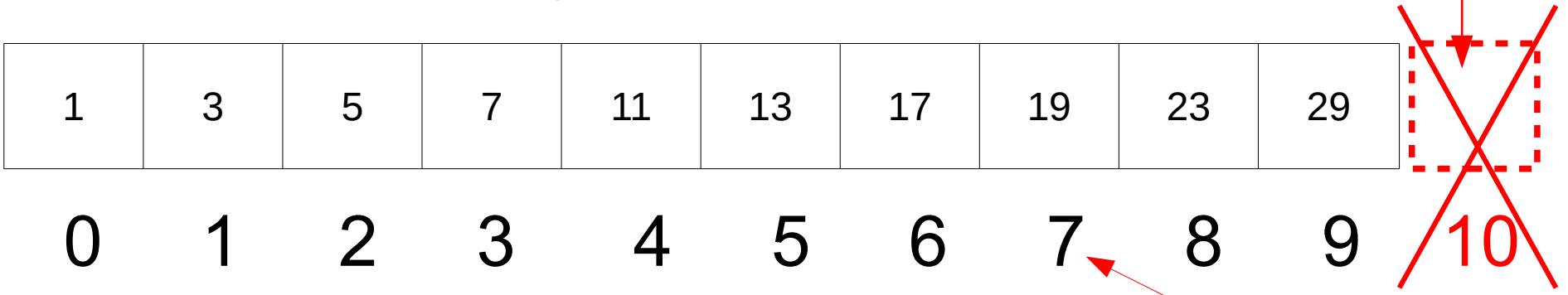
typ danych

nazwa tablicy

rozmiar tablicy,w przypadku tablicy statycznej jej rozmiar musi być wartością stałą, znaną na etapie kompilacji

nawiasy kwadratowe są zarezerwowane w C++ wyłącznie dla tablic

ostrożnie z wywoływaniem tej zmiennej !!!



numeracja: od **0** do **rozmiar-1**

index

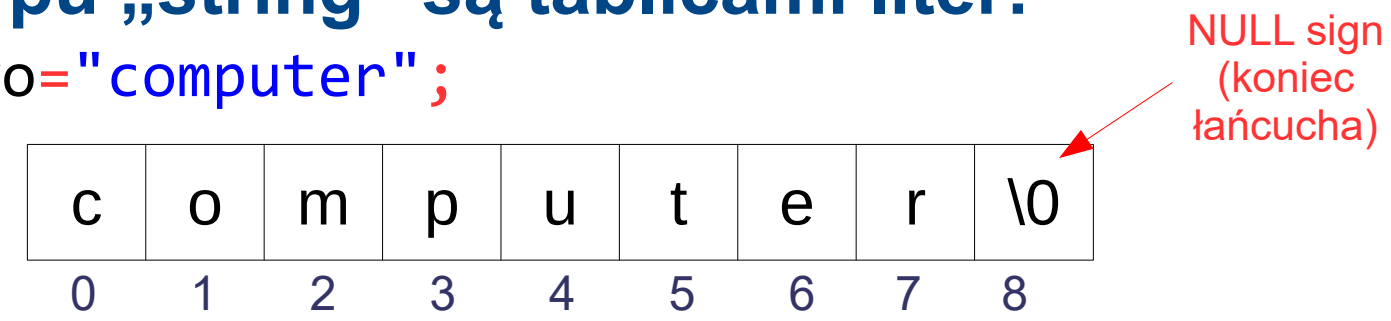
index – unikalny numer miejsca w tablicy (pozycja)

Przykład użycia (odwołania do elementu):

```
cout<<dane[3];   //wypisz element o indeksie 3
cin>>dane[i];    //zapisz do elementu o indeksie i
```

Zmienne typu „string” są tablicami liter:

```
string slowo="computer";
```




```
cout<<slowo[3];        //wypisze litere "p" na ekranie
```

Wielowymiarowe tablice:

```
float macierz[5][4];                          //macierz 2D
double dane[10][10][10][10][10];              //macierz 5D
```

Zadanie 12

Napisz program obliczający średnią liczb wczytywanych z klawiatury. Wykorzystaj zmienną tablicową do przechowania danych wejściowych. Obliczenia wykonaj na tej właśnie tablicy danych

Zadanie 13

Napisz program, który obliczy a następnie wypisze na ekranie kolejne liczby ciągu Fibonacciego. Komputer zapyta użytkownika ile liczb wypisać. Obliczone liczby przechowaj w tablicy a dopiero potem wypisz je na ekranie.

*Niezależnie od powyższego programu sprawdź, który i jaki jest maksymalny element ciągu możliwy do policzenia przez komputer.

Manipulacja strumieniem wyjściowym

```
#include<iomanip>

cout<<setprecision(6);                    // 6 liczb znaczacych
cout<<fixed<<setprecision(6);             // 6 liczb po przecinku

cout.width(10);                           //szerokosc 10 znakow
cout<<setw(10)<<12.3;

cout<<left<<12.3<<endl;                   //wyrownanie do lewej
cout<<right<<12.3<<endl;                  //wyrownanie do prawej

cout.unsetf(std::ios_base::floatfield);   //wylaczenie fixed
```

8. Zapisywanie i czytanie z pliku tekstowego

12.04.2024

Zapisywanie danych do pliku tekstowego

```
#include<fstream>          //biblioteka odpowiedzialna za pracę z plikami

fstream plik1;             //tworzenie obiektu plik1 klasy fstream

                           //tryb zapisu | dodawanie zawartości
plik1.open("filename.txt",ios::out|ios::app);
        //otwiera plik w aktualnej lokalizacji, jeżeli nie istnieje – będzie utworzony

plik1<<x<<"   "<<y<<endl;
                    //zapisanie zmiennych x i y rozdzielonych 3 spacjami do pliku
                            (kierowanie strumienia wyjściowego do obiektu plik1)
plik1.close();
                //zamykanie pliku, ważne, zawsze zamknij plik jeżeli został otwarty !
```

OBIEKTOWOŚĆ !!!

open, ***close***, ***good***, ***eof*** są metodami pracującymi na obiekcie ***plik1*** klasy ***fstream***

Czytanie danych z pliku tekstowego

```
plik1.open("filename.txt",ios::in);          //tryb odczytu

plik1.good();        //(prawda, jeżeli otwieranie pliku powiodło się, fałsz, jeśli nie lub plik nie istnieje)

plik1.eof();     //prawda, jeżeli podczas czytania osiągnięto koniec pliku

plik1>>a; //czytanie jednej zmiennej z pliku i zapisywanie jej na zmienną a, (kierowanie strumienia wyjściowego z obiektu plik1 na zmienną a)

plik1>>a>>b>>c;      //czytanie kolejno 3 zmiennych z pliku, zapisywanie ich na zmienne a, b, c i przejście do następnej linii

      //Powyższą metodę najczęściej stosujemy gdy wczytujemy dane uporządkowane o znanym typie.
```

Pamiętaj, żeby zamknąć plik po odczycie !

```
#include<cstdlib>

getline(plik1,linia); //czyta całą linię z plik1 (bez względu na to co się w niej znajduje), zapisuje na zmienną linia typu string i zwraca prawdę

atoi(linia.c_str());          //przekształca string na int lub float

atof(linia.c_str());
```

Metoda c_str() konwertuje ciąg znaków zapisany w zmiennej typu string na ciąg który może być zapisany w tablicy znaków.

Otwieranie pliku o nazwie podanej przez użytkownika

```
string nazwapliku;
fstream plik1;
cout<<"PODAJ NAZWE PLIKU: ";
cin>>nazwapliku;
plik1.open(nazwapliku.c_str(),ios::in);
    // tutaj operacje na zawartości pliku
plik1.close();
```

Zadanie 14

Napisz 2 programy które:

1: Zapyta użytkownika o nazwę pliku do zapisu danych, wyświetli menu (wykorzystaj `switch`), w którym będzie opcja dodania wpisu (rekordu) do pliku lub zakończenia programu. W przypadku wyboru opcji dodania rekordu zapyta użytkownika o numer (`int`), imię osoby (`string`) i jakąś liczbę rzeczywistą (`float`), którą jej przypiszemy (np. średnia ocen). Będzie zapisywał (lub dopisywał) te dane do pliku tekstowego o podanej na początku nazwie (jeden wiersz – jedna osoba np.: `3 Marek 3.45`). Za pomocą tego programu stwórz plik zawierający przynajmniej kilkanaście rekordów (1linia = jeden rekord = 3 dane).

2: Zapyta użytkownika o nazwę pliku z którego ma odczytać dane (wskazać plik stworzony w pierwszym programie) otworzy taki plik i odczyta dane zapisując je w tablicach o odpowiednim rozmiarze i typie. Program powinien policzyć wcześniej ile ma rekordów, zakładając, że tej liczby nie zna. Po odczytaniu danych i zapisaniu ich do tablic zamykamy plik. Następnie korzystając już z zapełnionych tablic program wypisze ich zawartość na ekranie w uporządkowany i czytelny sposób.

Zadanie 15

Struktura pliku tekstowego z danymi wygląda następująco: 3 linie komentarza, które mogą zawierać dowolne informacje tekstowe o danych, następnie nieznana liczba linii danych liczbowych (rzeczywistych) w 4 kolumnach: w pierwszej z nich znajduje się jakaś zmierzona wielkość fizyczna (powiedzmy X), w drugiej niepewności (dX) pomiaru odpowiednich wielkości X, trzecia kolumna zawiera wyniki pomiarów drugiej wielkości fizycznej (powiedzmy Y), a 4 to oczywiście niepewności (dY) pomiarów odpowiednich wielkości Y. Stwórz taki plik korzystając z notatnika i arkusza kalkulacyjnego. Nie zapomnij o tym, żeby do zapisu liczb zmiennoprzecinkowych używać kropki (notacja angielska).

Napisz program, który przeczyta zawartość takiego pliku (dane liczbowe koniecznie do jednaj tablicy) i wypisze na ekranie 3 linie komentarza, liczbę wierszy z danymi i zapyta czy wypisać wszystkie dane na ekranie. Jeżeli użytkownik potwierdzi taką chęć to program wypisze dane, jeżeli nie – zakończy program.

9. Zmienne tekstowe (string) i operacje na nich.

19.04.2024

```
string napis;      // wygodniejsze niż char[xx]
napis="Ala ma kota";
```

Przechowywanie w RAM – tablica znaków (char):

A	l	a		m	a		k	o	t	a	\0
0	1	2	3	4	5	6	7	8	9	10	11

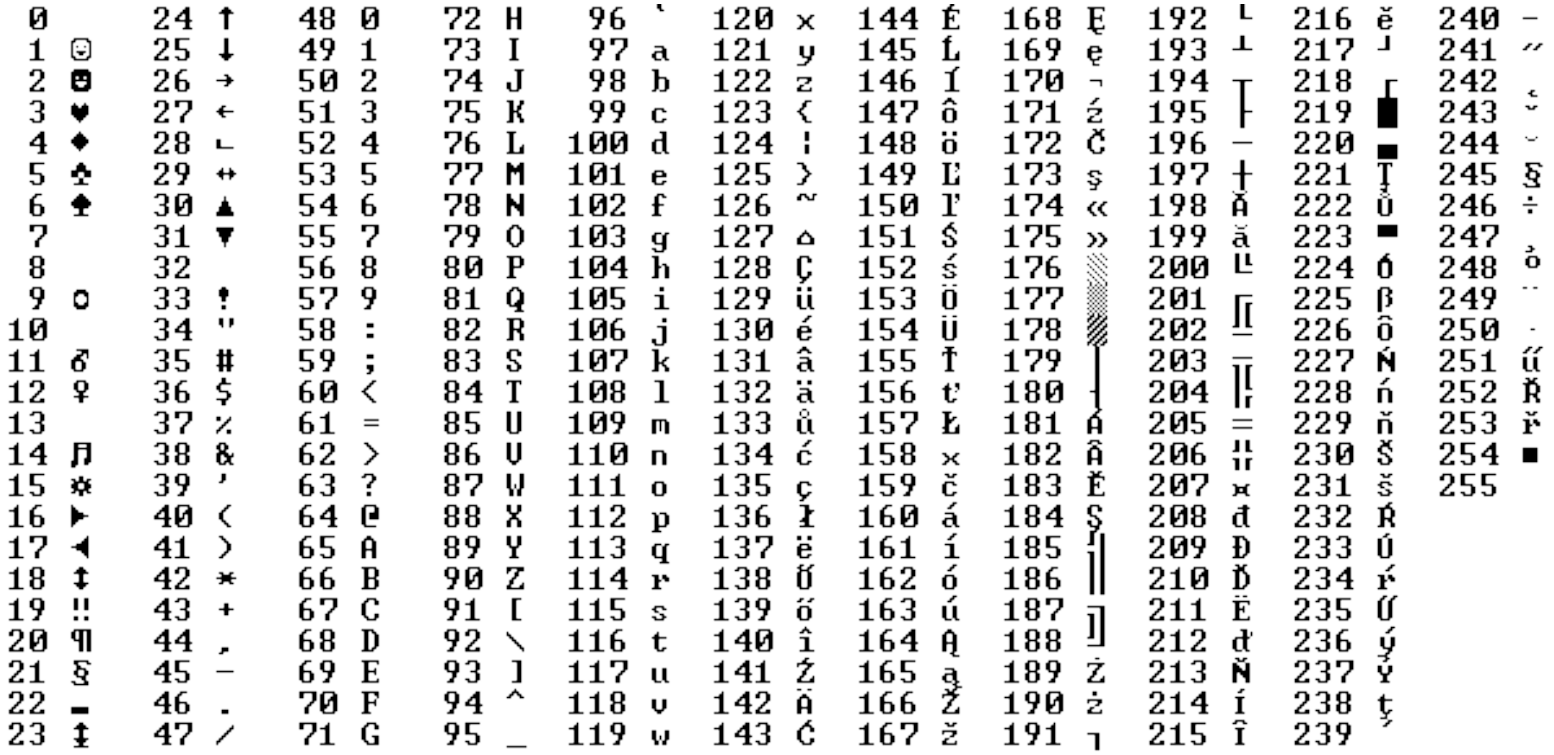
Pierwszy znak ma zawsze index=0

NULL sign kończy łańcuch

```
cout<<napis[4];                    //znak nr 4
cout<<napis.length(); //długość zmiennej lub
                      numer ostatniego znaku (null sign)
```

ASCII table

American Standard Code for Information Interchange



A	l	a		m	a		k	o	t	a	\0
65	108	97	32	109	97	32	107	111	116	97	0

duże litery (65-90) + 32 = małe litery (97-122)

Ważne

"a"= | a | \0 | //cudzysłów gdy łańcuch znaków

'a'= | a | //apostrof gdy pojedyncza litera

W strumieniu wejściowym spacja jest traktowana jako separator !

```
cin>>napis;
getline(cin,napis);
```

//użycie getline zamiast cin pozwala czytać z klawiatury napisy zawierające spacje aż do znaku następnej linii (enter, ¶)

```
#include<string>
            //biblioteka do operacji na string-ach)
```

Konkatenacja – łączenie

```
string jeden="Ala ma ";
string dwa="kota";
string trzy=jeden+dwa;
```

Znajdowanie pozycji frazy

```
string napis="Ala ma kota";
string szuk="ma";
int position=napis.find(szuk);
                        //jeżeli brak to position=-1
```

```
napis.erase(3,5);
                          //usuwa 5 znaków zaczynając od 3
napis.insert(11,"dostaw");
                    //wstawia string "dostaw" od pozycji 11
napis.replace(4,2,"zastepstwo");
    //zastępuje 2 znaki tekstem "zastepstwo" od pozycji 4
string nowynapis=napis.substr(4,7);
     //wycina nowy tekst z 7 kolejnych znaków, ze starego
                                        zaczynając od pozycji 4
```

Zamiana znaków małe ↔ DUŻE

```
#include<algorithm>
transform(napis.begin(),napis.end(),napis.begin(),::toupper);
transform(napis.begin(),napis.end(),napis.begin(),::tolower);
```

Procedura zmiany zmiennej typu string na tablicę znaków typu char

```
#include<cstring>

string napis="Ala ma kota";
cout<<napis<<endl<<endl;
int n=napis.length();
char tab_char[n];
strcpy(tab_char,napis.c_str());
for (int i=0;i<n;i++)
  cout<<tab_char[i]<<"  "<<int(tab_char[i])<<endl;

 //metoda c_str() konwertuje ciąg znaków zapisany w zmiennej
typu string na ciąg który może być zapisany w tablicy znaków
```

Zadanie 16

Napisz program, który będzie sprawdzał ile określonych w zapytaniu liter (np. 'a') jest w podanym z klawiatury tekście (zdaniu). Wielkość liter nie ma znaczenia, zliczamy zarówno duże jak i male litery.

Zadanie 17 (wysłać)

Opracuj prosty schemat szyfrowania oparty na numerach znaków z tablicy ASCII, np. przesunięcie numeru znaku o jeden lub więcej ("szyfr Cezara"). Osoba szyfrująca tworzy plik *sentence.txt* zawierający zdanie do zaszyfrowania oraz pisze program, który odczyta to zdanie, zaszyfruje a następnie zapisze zaszyfrowane zdanie do pliku *encrypted.txt*. Osoba odszyfrowująca znając szyfr pisze program, który odczyta zaszyfrowane zdanie, odszyfruje a odszyfrowane zdanie zapisze do pliku *decrypted.txt*. Sukcesem jest identyczna zawartość (co do treści) plików *sentence.txt* i *decrypted.txt*.

Ćwiczenia z programowania

26.04.2024

Zadanie

Napisz program do sortowania liczb metodą bąbelkową (rosnąco). Program zapyta użytkownika o rozmiar tablicy liczb, którą ma wylosować, posortuje tablicę rosnąco, wypisze obie tablice (wylosowaną i posortowaną) obok siebie na ekranie w postaci dwóch kolumn. Dodatkowo program zmierzy czas trwania operacji sortowania i również wypisze go na ekranie.

10. Funkcje. Podejście proceduralne.

10.05.2024

- Funkcja – podprogram
- Funkcja główna (`int main()`) – zarządza pozostałymi funkcjami (jeżeli istnieją)
- **Programowanie proceduralne** (paradygmat programowania polegający na podziale kodu na podprogramy tzn. fragmenty (bloki) wykonujące specyficzne operacje)

Przykład 1. Funkcja do obliczania potęgi.

```
#include<iostream>

using namespace std;

float num=2.3;          //zmienne globalne (każda funkcja ma dostęp)
int pow=4;

float power(float,int)      //deklaracja funkcji; typ wyniku, nazwa,
                          typy parametrów wejściowych, nawiasy okrągłe
int main()                                  //funkcja sterująca (główna)
{           //argumenty aktualne
    cout<<power(num,pow)<<endl;                       //wywołanie funkcji
    return 0;
}
           //argumenty formalne (może być bez)
float power(float a,int n)              //definicja funkcji (nagłówek)
{
    float b=a;              //ciało funkcji; zmienne lokalne, operacje
    for(int i=2;i<=n;i++)
        b*=a;
    return b;                                       //zwracana wartość
}
```

Przykład 2. Funkcja do wyświetlania menu.

```
    //umieszczenie definicji funkcji i jej ciała przed main() - nie
  trzeba wcześniej deklarować, ale jak jest więcej takich funkcji to
                        robi się niepraktyczne (kwestia subiektywna)

void main_menu()          //void – nie zwraca nic (kiedyś procedura)
{
    cout<<"   TYTUL PROGRAMU   "<<endl;
    cout<<"====================="<<endl;
    cout<<" Wybierz opcje:"<<endl;
    cout<<"  1. Zrob to"<<endl;
    cout<<"  2. Zrob tamto"<<endl;
    cout<<"  3. Wyjscie"<<endl;
               //nie ma return, może być, ale bez wartości zwracanej
}

int main()
{
    main_menu();                                    //wywołanie funkcji
    //reszta programu
    return 0;
{

     // Można się zastanowić, czy warto, żeby funkcja menu() zwracała
                        coś, np. liczbę odpowiadającą wybranej opcji.
```

Przykładowe deklaracje

```
double fun1(double);
          //zwraca liczbę double, przyjmuje liczbę double

float fun2(float x,float y);
       //zwraca liczbę float, przyjmuje dwie liczby float

void fun3(float x,int y);
    //niczego nie zwraca, przyjmuje liczbę float oraz int

void fun4();
             //niczego nie zwraca,  niczego nie przyjmuje
```

Przekazywanie tablic do funkcji

Tablice są przekazywane do funkcji inaczej niż standardowe zmienne. W przypadku tablic do funkcji zawsze trafia oryginalna zawartość tablicy, co oznacza, że jeśli funkcja zmieni coś w tablicy, to po zakończeniu jej wykonywania ta zmiana będzie zmianą trwałą.

Szczegółowo mechanizm przekazywania tablic do funkcji zostanie przedstawiony podczas omawiania wskaźników. Teraz musi wystarczyć nam wiedza o tym jak stworzyć funkcję przyjmującą jako parametr tablicę i jak taką funkcję wywołać.

Przekazując tablicę do funkcji nie podajemy jej rozmiaru, ale możemy (np: `void WypiszTab(int tab[5],int rozmiar)`). Kompilator jednak nie bierze pod uwagę liczby która pojawia się w nawiasach kwadratowych przy nazwie tablicy.

Przeanalizuj pzykład poniżej:

```
void WypiszTab(int tab[],int rozmiar)  //bez rozmiaru tablicy
{
    for(int i=0;i<rozmiar;i++)
        cout<<tab[i]<<" ";
    cout<<endl;
}

void PowiekszTab(int tab[],int rozmiar)
{
    for(int i=0;i<rozmiar;i++)
        tab[i]++;
}

int main()
{
    const int r=7;
    int tab[r]={1,2,3,4};
                    //reszta tablicy zostanie wypełniona zerami
    WypiszTab(tab,r);
    PowiekszTab(tab,r);
    WypiszTab(tab,r);
    return 0;
}
```

Wartości domyślne parametrów funkcji

Parametrom funkcji można przypisać wartości domyślne. Jeśli parametr ma przypisaną wartość domyślną, to podczas wywoływania tej funkcji nie musimy podawać jego wartości, w takiej sytuacji będzie on miał przypisaną wartość domyślną. Jeśli jednak podamy wartość dla danego parametru, to przyjmie on taką wartość jak została podana podczas wywołania funkcji.

```
void NapiszTekst(string napis,int ile=5)
{
    for(int i=0;i<ile;i++)
        cout<<napis<<endl;
}

int main()
{
    NapiszTekst("Witaj",2);
    NapiszTekst("Hello");
    NapiszTekst("Czesc",1);
    return 0;
}
```

Przeciążanie funkcji

W języku C++ możemy tworzyć funkcje o takich samych nazwach, ale muszą się one w takim przypadku różnić typem parametrów lub ich liczbą. Dopuszczalna jest sytuacja, w której funkcje mają taki sam typ parametrów ale w imnnej kolejności. Przykłady:

```
float dodaj()
float dodaj(int i)
float dodaj(float a, double b)
float dodaj(double a, float b)
```

Podczas wywołania funkcji kompilator wybierze jedną z nich na podstawie typu argumentu(ów).

W przypadku przeciążania nazw funkcji kompilator nie bierze pod uwagę zwracanego typu. Poniższe dwie funkcje są nieprawidłowe, ponieważ mają taki sam typ parametr:

```
int dodaj(float i)
float dodaj(float i)
```

Jeśli chcemy poinformować kompilator, że dana liczba jest typu float to możemy dodać po liczbie literę f: np.: `dodaj(2.4,3f)`.

- Deklaracja funkcji może, ale nie musi zawierać nazw argumentów formalnych, może być po prosu kopią definicji (nagłówka).
- Do funkcji wysyłane są kopie wartości argumentów poprzez argumenty formalne, funkcja nie ma dostępu do oryginalnych zmiennych (aktualnych argumentów), tylko do ich wartości. Wszelkie operacje są wykonywane na kopii i nie są widoczne poza blokiem funkcji.
- Instrukcja return oznacza powrót z funkcji do miejsca jej wywołania. Wykonanie instrukcji `return` powoduje zakończenie wykonywania funkcji (również `main()`). Jeśli funkcja powinna coś zwrócić to zwracana rzecz (wartość lub zmienna) pojawia się po słowie `return`. W obrębie jednej funkcji może pojawić się więcej niż jedna taka instrukcja. Jeżeli funkcja nic nie zwraca to po `return;` jest średnik lub pomijamy `return`.

Przypomnienie: zmienne lokalne i globalne

W funkcjach możemy zadeklarować nowe zmienne, będą one **zmiennymi lokalnymi**. Oznacza to, że będą dostępne tylko w tej funkcji, w której zostały zadeklarowane. Inne funkcje nie mają do nich dostępu. W różnych funkcjach możemy deklarować zmienne o takich samych nazwach. Zmienna lokalna nie ma przypisanej żadnej konkretnej wartości początkowej. Zmienne tworzone w obrębie danego bloku (np. funkcji) są przechowywane w pamięci tylko w momencie wykonywania tego bloku. Po jego zakończeniu, wszystkie zmienne w nim utworzone zostają z pamięci usunięte.

Zmienne globalne to zmienne zadeklarowane poza jakąkolwiek funkcją, również poza funkcją `main()`. Są dostępne dla wszystkich funkcji. Jeśli jakaś funkcja zmieni wartość zmiennej globalnej, to w innej funkcji taka zmiana jest widoczna. Wynika to z tego, że zmienne globalne są przechowywane w pamięci już po uruchomieniu programu i cały czas znajdują się w tym samym miejscu w pamięci (są przechowywane pod tym samym adresem). Wartość domyślna zmiennej globalnej to zero.

Przesłanianie zmiennych

Zmienne globalne mogą być **przesłonięte**, jeśli wewnątrz funkcji (lub bloku) zadeklarujemy inną zmienną o tej samej nazwie (choć niekoniecznie tym samym typie). Wówczas nazwa tej zmiennej w ciele funkcji odnosi się do zmiennej lokalnej. Zmienna globalna istnieje, ale jest w zakresie funkcji (bloku) niewidoczna.

Przykład 3. Rekurencja (rekursja).

Rekurencja to mechanizm w którym funkcja wywołuje sama siebie. Należy zachować szczególną ostrożność przy wykorzystaniu rekurencji; wywoływanie nie może powtarzać się bez końca lub zbyt dużo razy. Wpłynie to negatywnie na prędkość działania programu i może doprowadzić do wyczerpania pamięci.

Przykładem algorytmu, w którym możemy zastosować rekurencję jest obliczanie silni (przykład obok). Po podaniu przez użytkownika liczby całkowitej wywoływana jest funkcja obliczająca silnię i wypisywany jest wynik, który ta funkcja zwróciła.

```
int silnia(int n)
{
  if(n<=1)
    return 1;
  return n*silnia(n-1);
}

int main()
{
  int a;
  cout<<"Podaj liczbę:"<<endl;
  cin>>a;
  cout<<a<<"!="<<silnia(a)<<endl;
  return 0;
}
```

Zadanie 18

Popraw (rozbuduj) funkcję przedstawioną jako przykład w taki sposób, żeby pozwalała na obliczenie ujemnej potęgi danej liczby. Zadbaj o to, żeby program liczył tylko całkowite potęgi, nawet przy przypadkowym podaniu potęgi jako liczby zmiennoprzecinkowej. Niech program napisze jakie działanie wykonał i jaki jest jego wynik, np.: `(-2.1)^3=-9.261`

Zadanie 19 (wysłać)

Przekształć program KALKULATOR (Zadanie 10) tak, aby używał funkcji (działania matematyczne, menu główne). Dodaj działanie potęgowania i obliczania silni. Program powinien działać w pętli aż do wyboru opcji wyjścia.

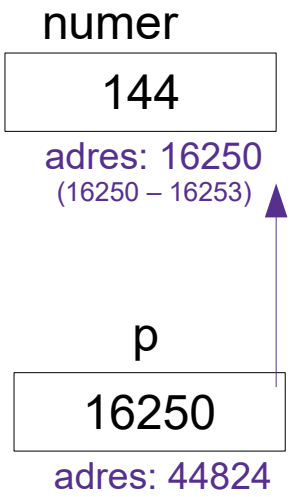
11. Wskaźniki. Dynamiczna alokacja pamięci.

23.05.2024

Każda komórka (1 Bajt) w pamięci RAM posiada swój własny numer (adres) zachowany w formacie heksadecymalnym (HEX) Taki format zapisu jest krótszy niż dziesiątkowy.

Wskaźnik (pointer) – zmienna, która przechowuje adres w pamięci RAM innej zmiennej (jej pierwszego Bajta).

Dynamiczna alokacja pamięci – w każdym momencie, na żądanie, nie tylko przy uruchomieniu i zakończeniu programu (statyczna alokacja pamięci).



Użycie wskaźnika pojedynczej zmiennej

```
int main()
{
  int numer=144;                    //zmienna int (4 bajty w RAM)
  int *w;              //typ zmiennej, *, nazwa wskaźnika w kodzie
  w=&numer;     //adres zmiennej numer przypisujemy na wskaźnik
  cout<<w<<endl;                          //wypisz wskaźnik (hex)
  cout<<*w<<endl;//wypisz zmienną, na którą wskazuje wskaźnik
  return 0;
}
```

& (ampersand) - operator uzyskiwania adresu w pamięci obiektu stojącego po jego prawej stronie

Zalety stosowania wskaźników:

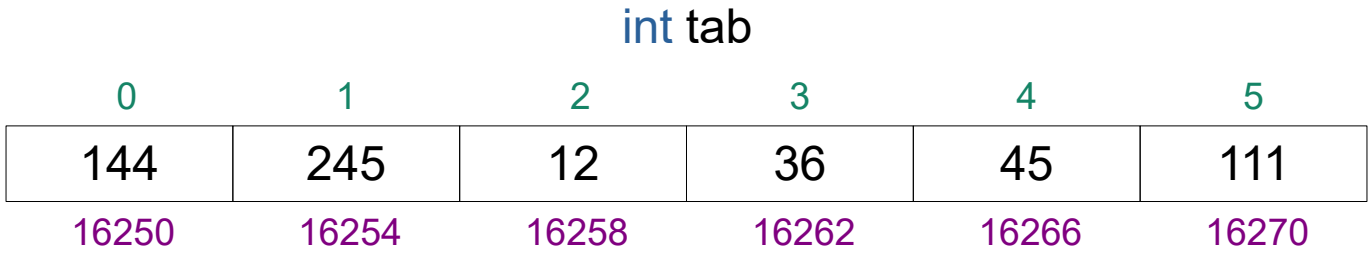
1) Dynamiczne rezerwowanie i zwalnianie zasobów pamięci RAM.

2) Zwiększenie szybkości zapisu/odczytu komórek w tablicy.

3) Przekazywanie funkcjom oryginalnych zmiennych z głównego programu do pracy z nimi.

4) Możliwość współpracy z urządzeniami zewnętrznymi.

Wskaźnik tablicy (dynamiczna alokacja pamięci)




```
int rozm=10;                                        //rozmiar tablicy
int *tab;                                  //deklaracja wskaźnika o nazwie tab
tab=new int[rozm];                                  //dynamiczna alokacja tablicy
for(int i=0; i<size; i++)
{
  cout<<tab<<"   "<<*tab<<endl;      //adres komórki i jej zawartość
  tab++;          //inkrementacja wskaźnika (4 dla int, 8 dla double)
}
delete [] tab; //usuń tablicę o nazwie tab (zawsze jeżeli było new)
```

```
p=&tab[0] ⇔ p=tab
//nazwa tablicy jest adresem jej zerowego elementu
```

Pomiar czasu pracy programu (powtórzenie)

```
#include <time.h>
#include <cstdlib>

clock_t start, stop;                    //2 zmienne typu clock_t
float tsec;                  //zmienna do przechowywania czasu w sek.
start=clock();                          //numer cyklu procesora

    // operacje do pomiaru czasu

stop=clock();                           //numer cyklu procesora
tsec=(float)(stop-start)/CLOCKS_PER_SEC;
       //liczba cykli rzutowana na float / liczba cykli na sekundę
cout<<"czas operacji [s]: "<<tsec<<endl;
```

Szybkość zapisu / odczytu

Zadanie 20

Napisz program który: **a)** utworzy 2 lub więcej dużych tablic z danymi liczbowymi, **b)** wykona na tablicach szereg operacji odczytu i zapisu (np. działania matematyczne) – raz bez użycia wskaźników a następnie z użyciem wskaźników, **c)** zmierzy czas wykonywania operacji w obu przypadkach i wypisze go na ekranie. Operacje na tablicach powinny być oczywiście takie same w obu przypadkach. Jeżeli zmierzony czas będzie za mały (współczesne komputery są szybkie), spróbuj zwiększyć rozmiar tablic lub wykonać te same operacje kilka razy w pętli.

Przekazywanie funkcjom oryginalnych zmiennych z głównego programu

```
float srednia(float &x, float &y, float &z)
//float avg(float x, float y, float z)
{
    float s=(x+y+z)/3;
    x+=1000; y+=1000; z+=1000;
    return s;
}

int main()
{
    float a=1.2, b=2.3, c=3.1;
    cout<<”przed funkcja: ”<<a<<”; ”<<b<<”; ”<<c<<endl;
    cout<<”funkcja: ”<<srednia(a,b,c)<<endl;
    cout<<”po funkcji: ”<<a<<”; ”<<b<<”; ”<<c<<endl;
    return 0;
}
```

Czy w przypadku tablic funkcje zawsze pracują na oryginalnych danych?

Dlaczego?

```
float srednia(float tab[], int n)
float srednia(float tab, int n)
float srednia(float *tab, int n)
```

Zwróć uwagę na różnice gdy używamy wskaźników lub nie.

Zadanie 21

Napisz prosty program sprawdzający, czy funkcje pracują na oryginalnych tablicach czy nie.

12. Ćwiczenia programistyczne

xx.xx.2024

Zadanie 21 (analiza tekstu)

Napisz program, który wczyta cały tekst z pliku tekstowego, a następnie wykona analizę ilości znaków. Program wypisze ilość wszystkich znaków w tekście a następnie ilość konkretnych znaków, ale tylko i wyłącznie tych, które w tekście wystąpiły przynajmniej raz.

13. Ćwiczenia programistyczne

xx.xx.2024

Zadanie 22 (prosta baza danych)

Napisz program do utworzenia i zarządzania prostą bazą danych tekstowych i numerycznych (np. książki, przedmioty kolekcjonerskie). Program powinien posiadać opcje wypisywania rekordów, dodawania rekordów do bazy, usuwania niechcianych rekordów oraz modyfikowania istniejących rekordów.

14. Ćwiczenia programistyczne

xx.xx.2024

Zadanie 23 (problem komiwojażera)

Napisz program, który rozwiąże problem komiwojażera. Zdefiniuj w pliku tekstowym (lub wylosuj) współrzędne miejsc dostarczenia przesyłek w wybranym kartezjańskim układzie współrzędnych. Program powinien znaleźć najkrótszą drogę pomiędzy punktami zaczynając od pierwszego z nich.

15. Ćwiczenia programistyczne

xx.xx.2024

Zadanie 24 (sortowanie bąbelkowe)

Napisz program do sortowania liczb metodą bąbelkową. Program zapyta użytkownika o rozmiar tablicy liczb, którą ma wylosować, następnie posortuje tablicę rosnąco lub malejąco (napisać funkcję do sortowania), a następnie wypisze obie tablice obok siebie na ekranie w postaci dwóch kolumn. Dodatkowo program zmierzy czas trwania operacji sortowania i również wypisze go na ekranie.

16. Test zaliczeniowy

14.06.2024