

PROGRAMOWANIE

1

ostatnia aktualizacja 05.04.2024

Marek Brancewicz

Wydział Fizyki

Uniwersytet w Białymstoku

m.brancewicz@uwb.edu.pl



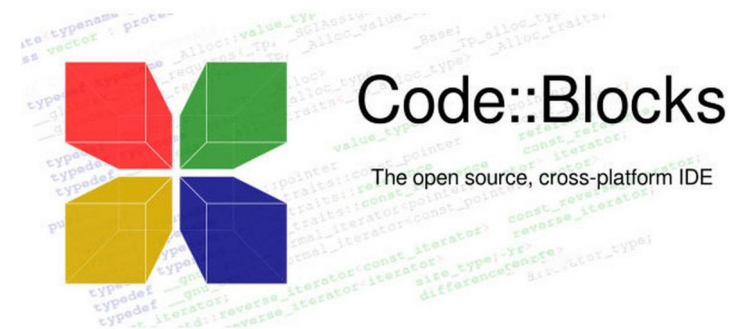
WYDZIAŁ FIZYKI
UNIwersytet w BIAŁYMSTOKU



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.

```
FF 30 20 B4 FC 90 F7 60 B1 3C
#TDEDL
TDEDL          6 7 00      JMP      (#0036)
TDEDL          6 7 00      CMP      ###00
TDEDL          6 7 00      BCC      #TDF6
TDEDL          6 7 00      STY      0
TDEDL          6 7 00      PHA
TDEDL          6 7 FB      LSR      #FB78
TDEDL          6 7 35      ROL      #35
TDEDL          6 7 4 4      DEFS     #4
TDEDL          6 7 4 4      DEFS     #TDA3
TDEDL          6 7 10      DEFS     #FE10
TDEDL          6 7 10      DEFS     ###00
TDEDL          6 7 10      DEFS     #TDC6
TDEDL          6 7 10      STAB
TDEDL          6 7 10      STAB
TDEDL          6 7 40      STA     (#40),Y
TDEDL          6 7 40      INC     #40
```

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 **środków 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.



```
Hello, World_
```

```
#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 i stałych (np. `int x,n; const float pi=3.1415926;`)
- funkcja główna (sterująca) – `int main() {}`
- `return 0;` (zwraca kody błędów)
- białe znaki (spacje, enter) są ignorowane
- każda linia powinna się kończyć średnikiem ";" (są wyjątki)
- komentarze "//"

Struktra projektu

- `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 strukturze (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

```
#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:

$!((x \leq 5) || (x > 12) || (a \neq 7) \&\& (a > 15))$

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: $ax^2+bx+c=0$, 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();` *ignorowanie zawartości strumienia; jeden wyraz lub 1000 znaków do końca linii*

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

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)

*Iterator
(musi być integer)*

*wartość początkowa
iteratora*

*wyrażenie logiczne
(pętla się wykona jeżeli prawda)*

zmiana iteratora w każdym kroku

```
for (int i=1; i<=10; i++)  
{  
    cout<<i<<endl;  
}
```

zestaw instrukcji do wykonania

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) // dzwiek (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);
```

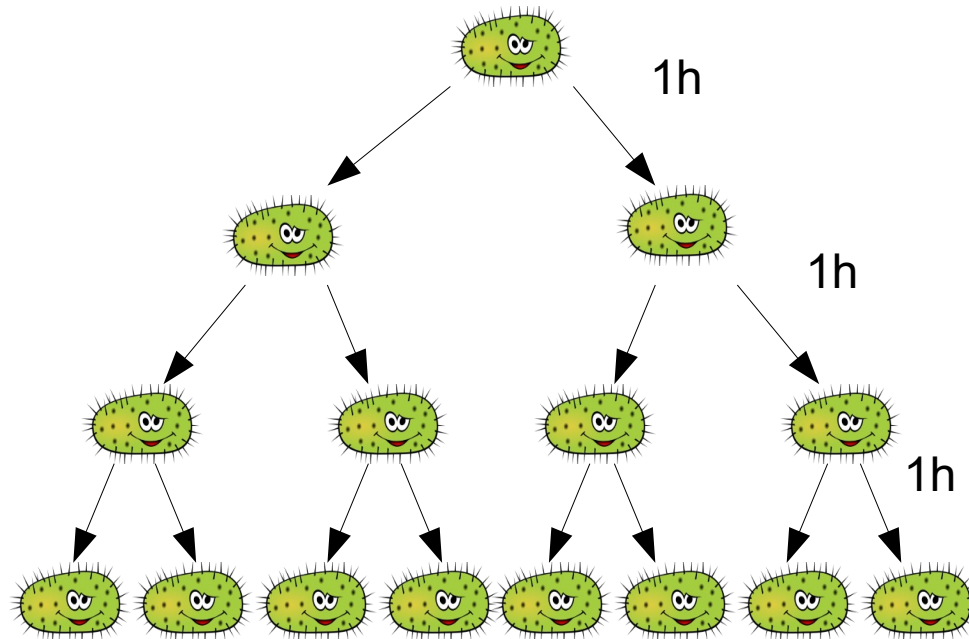
*Zestaw instrukcji wykona
się przynajmniej raz.*

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).

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
```

```
// przykładowy 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

```
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
    }
}
```

Składnia „switch case”

zmienna kontrolna:

int

char

~~**float**~~

~~**string**~~

Użyteczne

<code>break;</code>	<i>zatrzymuje pętlę i przechodzi dalej</i>	<i>kończy program i zwraca 0</i>
<code>exit(0);</code>	<code>(#include <cstdlib>)</code>	
<code>system("cls");</code>	<code>(#include <cstdlib>)</code>	<i>czyści ekran</i>
<code>getch();</code>	<code>(#include <conio.h>)</code>	

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 π metodą Monte Carlo. Algorytm: 1) Wylosuj n współrzędnych punktów (par liczb) wewnątrz kwadratu o boku $2r$ (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 π . Spraw, aby program działał w pętli licząc π dla liczb $n=10, 100, 1000 \dots$, 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 π , względną różnicę między rzeczywistą liczbą π 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];`

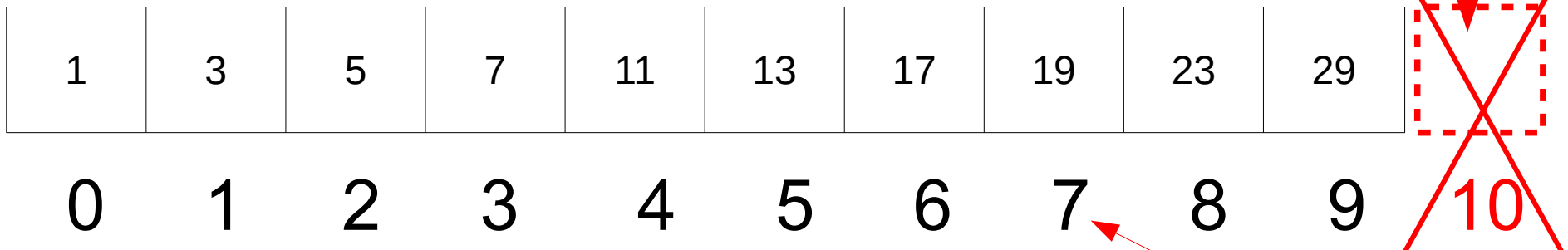
typ danych

nazwa tablicy

rozmiar tablicy

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";
```

c	o	m	p	u	t	e	r	\0
0	1	2	3	4	5	6	7	8

NULL sign
(koniec
łańcucha)



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

cout.width(10);                 // szerokość 10 znaków
cout<<setw(10)<<12.3;

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

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

8. Zapisywanie i czytanie z pliku tekstowego

12.04.2024

Zapisywanie danych do pliku tekstowego

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

(podobnie jak deklaracja zmiennej, tworzenie obiektu *plik1* klasy *fstream*)

`fstream plik1;`

(tryb zapisu)

(tryb
dodawania
zawartości,
append)

(otwiera plik w aktualnej
lokalizacji, jeżeli nie
istnieje – będzie
utworzony)

`plik1.open("filename.txt", ios::out | ios::app);`

`plik1 << x << " " << y << endl;` (zapisywanie 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

(tryb odczytu)

`plik1.open("filename.txt", ios::in);` (pamiętaj, żeby zamknąć plik po odczycie)

`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 wejściowego 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)

`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 integer lub float)

`atof(linia.c_str());` **`#include <cstdlib>`**

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 nazwę pliku: ";  
cin >> nazwapliku;  
plik1.open(nazwapliku.c_str(),ios::in);  
// tutaj operacje na zawartości pliku  
plik1.close();
```

Zadanie 14

Napisz program testowy, który zapisuje i odczytuje różne typy danych do/z pliku (teksty, liczby, np. imię, numer telefonu). Program powinien zapytać użytkownika, czy chce utworzyć nowy plik z danymi, odczytać dane z istniejącego pliku lub zakończyć program.

Wprowadź modyfikację do programu, która pozwoli na dopisywanie nowych danych do istniejącego pliku, będzie również czytać je wszystkie i wypisywać na ekranie.

Zadanie 15 (wysłać)

W wielu przypadkach naukowe urządzenia pomiarowe tworzą tekstowe pliki wyjściowe zawierające kolumny danych liczbowych. Utwórz taki plik, który będzie zawierał 2 kolumny danych liczbowych najlepiej za pomocą arkusza kalkulacyjnego.

Napisz program, który odczyta te dane z pliku, przechowa w tablicy (tablicach), wypisze liczbę danych (par danych) na ekranie. Następnie, niech program wykona na tych kolumnach dowolną operację matematyczną (np. dodawanie) i zapisze wczytane 2 kolumny danych wraz z 3 będącą wynikiem działania do pliku tekstowego. Zadbaj o odpowiednie wyrównanie kolumn.

Na koniec program zapyta użytkownika czy wypisać wszystkie dane (3 kolumny) na ekranie. W zależności od odpowiedzi użytkownika zrobi to lub nie i zakończy działanie po naciśnięciu klawisza.

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

19.04.2024

`string napis;`

`napis="Ala ma kota";` string jest wygodniejsze niż używanie `char[XX]`

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 typu string (integer)
lub numer ostatniego znaku (null sign)

ASCII table

American Standard Code for Information Interchange

0		24	↑	48	0	72	H	96	`	120	x	144	É	168	ƒ	192	ˆ	216	ě	240	-
1	☺	25	↓	49	1	73	I	97	a	121	y	145	Ê	169	ç	193	˜	217	ĵ	241	~
2	☼	26	→	50	2	74	J	98	b	122	z	146	Ë	170	¸	194	¸	218	ı	242	˘
3	♥	27	←	51	3	75	K	99	c	123	<	147	Ě	171	ˆ	195	ˆ	219	■	243	˙
4	♦	28	⌞	52	4	76	L	100	d	124	!	148	Š	172	Č	196	ˆ	220	■	244	˚
5	♠	29	↔	53	5	77	M	101	e	125	>	149	Ẹ	173	§	197	ˆ	221	ı	245	§
6	♣	30	▲	54	6	78	N	102	f	126	~	150	Ɔ	174	«	198	ˆ	222	ü	246	÷
7		31	▼	55	7	79	O	103	g	127	Δ	151	Š	175	»	199	ˆ	223	■	247	˚
8		32		56	8	80	P	104	h	128	Ç	152	š	176	ˆ	200	ˆ	224	ó	248	˚
9	o	33	!	57	9	81	Q	105	i	129	ü	153	ö	177	ˆ	201	ˆ	225	β	249	˚
10		34	"	58	:	82	R	106	j	130	é	154	ü	178	ˆ	202	ˆ	226	ô	250	˚
11	♂	35	#	59	;	83	S	107	k	131	â	155	ı	179	ˆ	203	ˆ	227	Ń	251	ı
12	♀	36	\$	60	<	84	T	108	l	132	ä	156	ı	180	ˆ	204	ˆ	228	ń	252	ı
13		37	%	61	=	85	U	109	m	133	å	157	Ł	181	ˆ	205	ˆ	229	ň	253	ı
14	☾	38	&	62	>	86	V	110	n	134	ć	158	×	182	ˆ	206	ˆ	230	š	254	■
15	*	39	'	63	?	87	W	111	o	135	ç	159	č	183	ˆ	207	ˆ	231	š	255	
16	▶	40	<	64	@	88	X	112	p	136	ı	160	á	184	ˆ	208	ˆ	232	ř		
17	◀	41	>	65	A	89	Y	113	q	137	ë	161	í	185	ˆ	209	ˆ	233	ú		
18	↑	42	*	66	B	90	Z	114	r	138	š	162	ó	186	ˆ	210	ˆ	234	ř		
19	∴	43	+	67	C	91	[115	s	139	š	163	ú	187	ˆ	211	ˆ	235	ú		
20	☹	44	,	68	D	92	\	116	t	140	ı	164	ř	188	ˆ	212	ˆ	236	ý		
21	§	45	-	69	E	93]	117	u	141	ž	165	á	189	ˆ	213	ˆ	237	ý		
22	■	46	.	70	F	94	^	118	v	142	ň	166	ž	190	ˆ	214	ˆ	238	ı		
23	‡	47	/	71	G	95	_	119	w	143	č	167	ž	191	ˆ	215	ˆ	239	ı		

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
---	----

'a'=

a

cudzysłów gdy łańcuch znaków !

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 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;
```

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 małe litery.

Zadanie 17

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*.

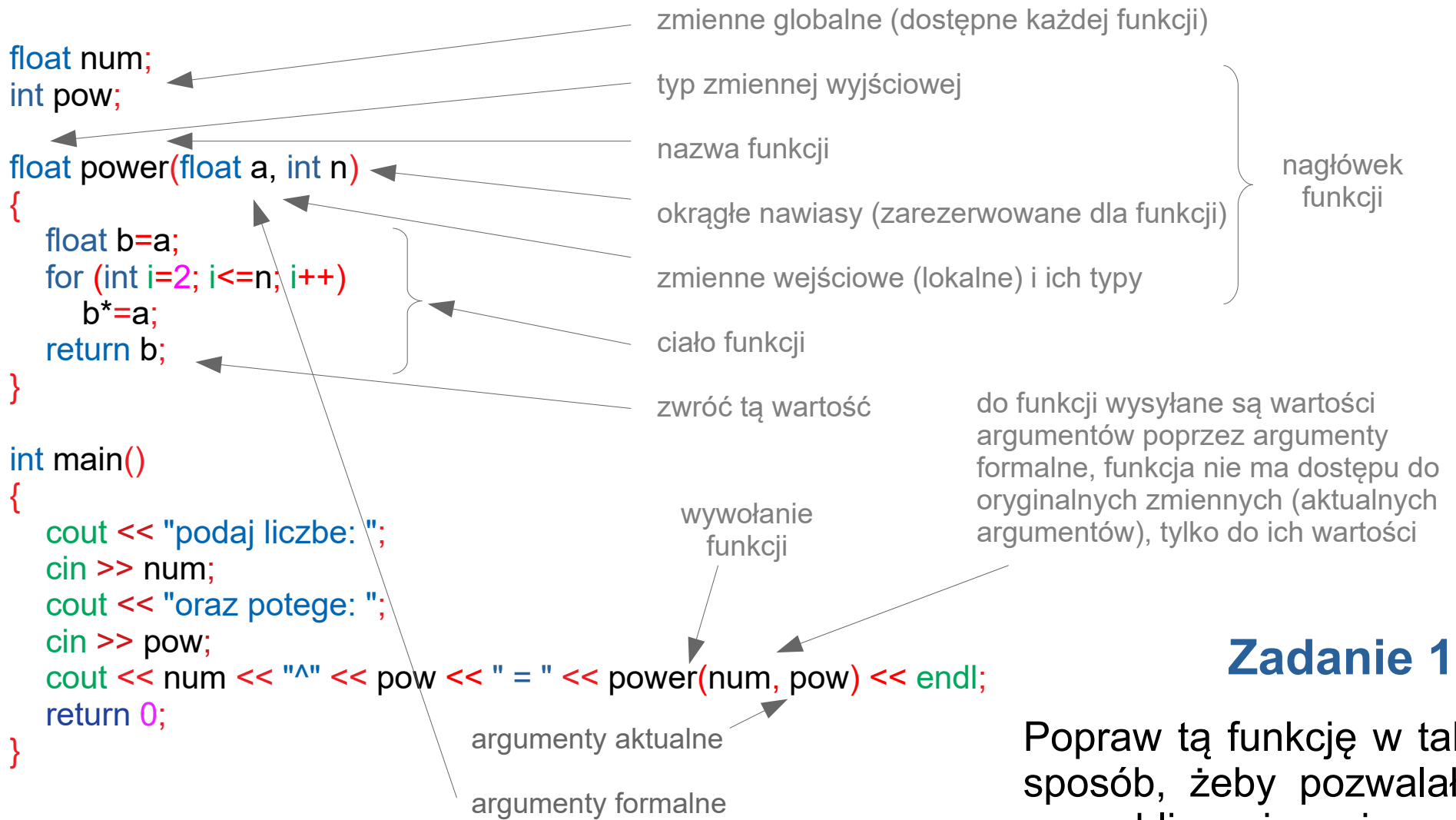
10. Funkcje. Podejście proceduralne.

26.04.2024

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

Przykład 1

Funkcja, która oblicza i zwraca żądaną potęgę podanej liczby



Zadanie 16

Popraw tą funkcję w taki sposób, żeby pozwalała na obliczenie ujemnej potęgi liczby.

Można napisać funkcję poniżej main(), ale nagłówek funkcji musi być powtórzony powyżej main().

Przykład 2

Funkcja, która coś robi ale niczego nie zwraca – procedura

```
void main_menu() ← może również mieć jakieś argumenty
{
  cout << "..." << endl;
  cout << "..." << endl;
  cout << "..." << endl;
} ← w przypadku procedury

int main() ← brak return
{
  main_menu();
  return 0;
} ← to jest is również funkcja
```

Zadanie 18^(wysłać)

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

Zadanie 19

Przyjrzyj się bliżej programom, które napisałeś(aś) na tych zajęciach. Zastanów się, czy warto by było zastosować w nich funkcje.

11. Wskaźniki. Dynamiczna alokacja pamięci.

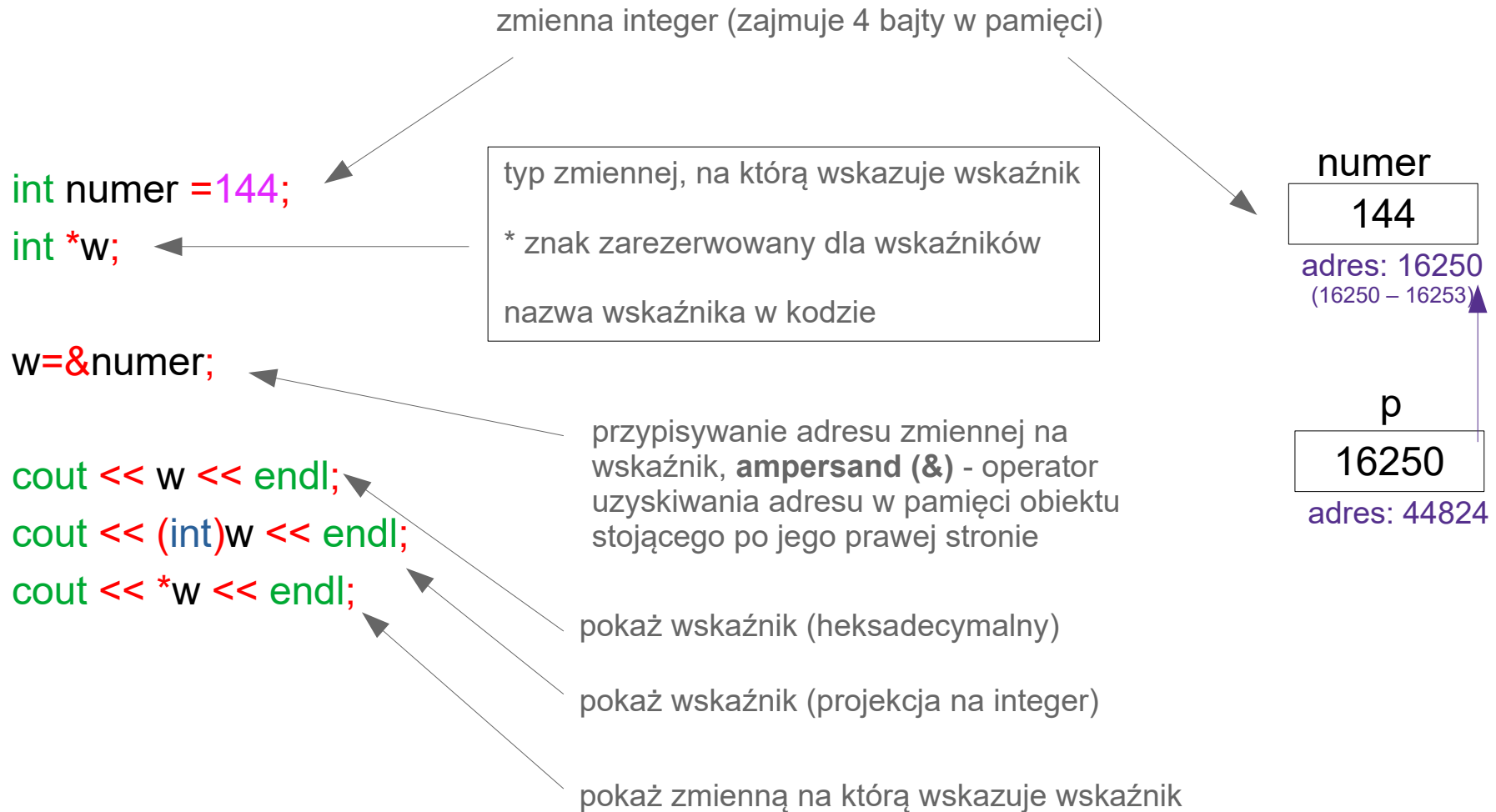
10.05.2024

Każda komórka (1 Bajt) w pamięci RAM posiada swój własny numer (adres) zachowany w formacie heksadecymalnym (krótszy zapis niż dziesiętkowy).

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

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

Użycie wskaźnika pojedynczej zmiennej



Zalety stosowania wskaźników:

- 1) Dynamiczna rezerwacja 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 tab

0	1	2	3	4	5
144	245	12	36	45	111
16250	16254	16258	16262	16266	16270

```
cin >> size;
```

```
int *tab;
```

```
tab=new int[size];
```

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

```
{
```

```
    *tab=i;
```

```
    cout << (int)tab << " " << *tab << endl;
```

```
    tab++;
```

```
}
```

```
delete [] tab;
```

deklaracja wskaźnika o nazwie tab

dynamiczna alokacja tablicy

inkrementacja wskaźnika (adres +4 w przypadku integera, +8 jeśli double)

czyszczenie pamięci (zawsze, jeżeli użyto operatora new)

$p = \&tab[0] \leftrightarrow p = tab$

Nazwa tablicy jest adresem jej pierwszego elementu!

Pomiar czasu pracy programu

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

```
#include <cstdlib>
```

```
clock_t start, stop;
```

```
float time;
```

```
start=clock();
```

```
// operacje do pomiaru czasu
```

```
stop=clock();
```

```
time=(float)(stop-start)/CLOCKS_PER_SEC;
```

```
cout << "czas operacji [s]: " << time << endl;
```

dodatkowe biblioteki

dwie zmienne typu clock_t

zmienna do przechowywania czasu
w sekundach

projekcja na typ float

liczba cykli procesora

zegar procesora (taktowanie)

Przekazywanie funkcjom oryginalnych zmiennych z głównego programu do pracy z nimi

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

```
int main()
{
    float a=1.2, b=2.3, c=3.1;
    cout << avg(a,b,c) << endl;
    cout << a << "; " << b << "; " << c << endl;
    return 0;
}
```

Czy w przypadku tablic funkcje zawsze pracują na oryginalnych danych ???

```
float avg1(float tab[100], int n)
```

```
float avg2(float tab[], int n)
```

```
float avg3(float *tab, int n)
```

Zwróć uwagę na różnice gdy używamy wskaźników lub nie.

Zadanie 23

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

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.

12. Ćwiczenia programistyczne

17.05.2024

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

24.05.2024

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

07.06.2024

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

14.06.2024

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

