Fundamentele Programarii

Variabile si constante

Variabile si constante

- Variabilele sunt concepte fundamentale in limbajele de programare
- Cum se declara variabilele
- Tipuri de date fundamentale C++
  - Integer
  - Float / Double
  - Bool
  - Character

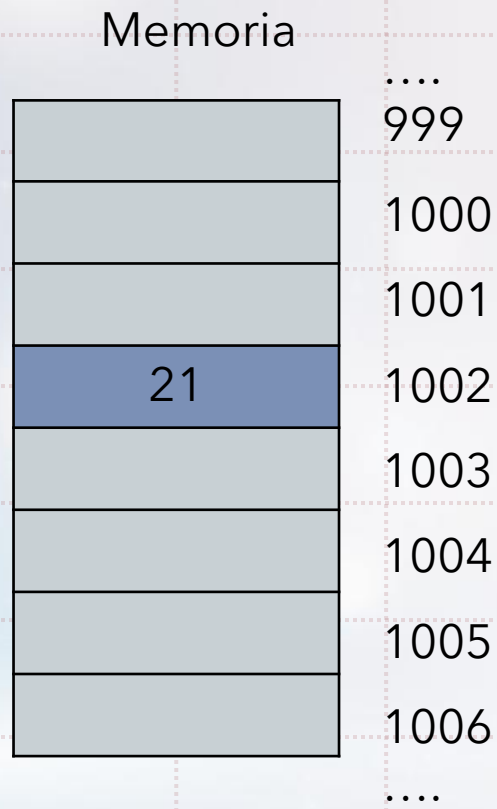
- Operatorul *sizeof*

- Ce este o constanta?
- Cum se declara constantele
- Constantele literale
- Expresii cu constante

Ce este o variabila?

- Conceptual, o variabilă este un **nume simbolic** folosit pentru o **locație de memorie**.
- Practic, **variabila are o adresă**, dar **nu este** adresa însăși.
- Ar fi mult mai dificil de programat daca ar trebui sa lucram doar cu adresele de memorie si am putea genera erori mai frecvent
- Majoritatea limbajelor de programare ne permit sa asociem un nume unei adrese de memorie.
- **Exemplu**: Dorim sa folosim numarul “21” in programul nostru.

---> “Pune valoarea “21” la locatia 1002”

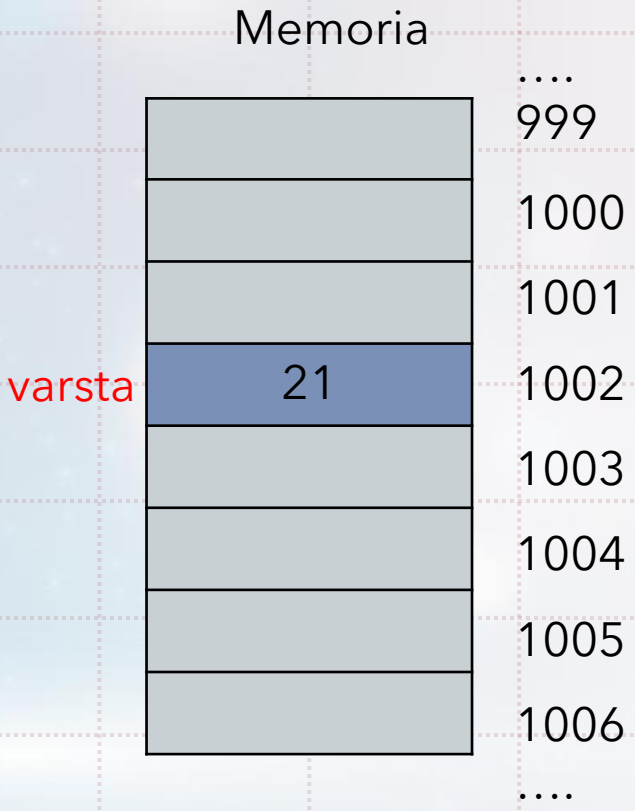


Ce este o variabila?

- Adresei 1002 ii putem asocia un nume varsta

--- > “Pune valoarea “21” in variabila varsta”
(nu ne intereseaza precis locatia)

- Daca compilam din nou, numelui varsta ii va corespunde o alta adresa (nu este nici o problema, codul va functiona in continuare).
- varsta este o variabila, deci continutul ei poate varia. Prin urmare, putem schimba valoarea ei cu “22”, de exemplu.



Ce este o variabila?

- O variabila este o **abstractizare a locatiei** din memoria calculatorului
- Variabilele permit programatorilor sa foloseasca nume relevante in locul adreselor din memorie
- Variabilele au doua proprietati importante
  - *Tipul* --- categoria lor (numar intreg, numar real, persoana,…)
  - *Valoarea* --- continutul lor (10; 3,14; “Simona”,…)

- Variabilele trebuie sa fie declarate inainte de a fi folosite
- Valoarea variabilelor poate fi schimbata

```
age = 21;   // eroare a compilatorului
```

```
int age;

age=21;
```

Declararea si initializarea unei variabile

- Declararea Variabilelor

TipulVariabilei **NumeleVariabilei**

```
int varsta;
double rata;
string nume;

Persona simona;
```

- **Numele variabilei este un identificator definit de utilizator!**

Declararea si initializarea unei variabile

Regulile pe care trebuie sa le respectam atunci cand dam nume variabilelor

- Pot contine litere, numere si bara jos (underscore) “_”;
- Trebuie sa inceapa cu o litera sau cu bara jos (nu pot incepe cu un numar);
- Nu putem folosi cuvintele cheie sau identificatorii predefiniti ale C++;
- Nu putem redeclara un nume in acelasi scop;
- Atentie, C++ face diferenta intre litere majuscule si minuscule (case sensitive)

Declararea si initializarea unei variabile

Nume de variabile

Corect	Incorect
Varsta	int
varsta	$varsta
_varsta	2021_varsta
Varsta_mea	Varsta mea
Varsta_ta_in_2021	Varsta+1
INT	cout
Int	return

Declararea si initializarea unei variabile

- Fii consistent cu conventiile pe care le stabilesti pentru numele variabilelor!

  Exemplu: numeleVariabilei sau numele_variabilei

- Evita sa incepi numele variabilelor cu bara jos!
- Foloseste nume cu inteles (nu prea lungi, nu prea scurte)!
- Incearca sa nu folosesti variabile fara sa le initializezi!
- Declara variabilele in apropierea locului unde le folosesti!

Declararea si initializarea unei variabile

Initializarea Variabilelor

```
int varsta; //variabila nu este initializata

int varsta = 21; //initializare in stilul C

int varsta(21); //constructor de initializare (POO)

int varsta{21}; //sintaxa de initializare C++11
```

Declararea si initializarea unei variabile

Exemplu

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

//Acest program calculeaza aria unui dreptunghi
int main() {
    cout << "Introduceti lungimea dreptunghiului ";
    int lungimeaDreptunghiului{ 0 };
    cin >> lungimeaDreptunghiului;



    cout << "Introduceti latimea dreptunghiului ";
    int latimeaDreptunghiului{ 0 };
    cin >> latimeaDreptunghiului;

    cout << "Aria dreptunghiului este " << lungimeaDreptunghiului * latimeaDreptunghiului << endl;

    return 0;
}
```

Declararea si initializarea unei variabile

Variabile globale si locale

- Variabilele folosite in exemplul anterior sunt **variabile locale**.
- **Variabilele locale** sunt cele declarate in interiorul unei functii; ele sunt sunt vizibile doar in cadrul acelei functii.
- **Variabilele globale** sunt cele definite in afara oricarei functii si pot fi accesate din orice parte a programului. Variabilele globale sunt automat initializate cu **zero**!

Declararea si initializarea unei variabile

Exemplu

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

int varsta = 21;                //variabila globala

int main() {
    int varsta = 20;                        //variabila locala
    cout << "Varsta este " << varsta << endl;
    return 0;
}
```

Ce va afisa programul de mai sus?

Varsta este 20

Tipuri de date fundamentale in C++

Sunt implementate direct de limbajul C++

- Caracter (`char`)
- Intreg (`int`)
  - ✓ Cu semn (`signed`)
  - ✓ Fara semn (`unsigned`)
- Real (`float, double`)
- Boolean (`bool`)

Marimea si precizia lor sunt dependente de platforma si de compilatorul folosite!

```
#include<climits>
```

Tipuri de date fundamentale in C++

Marimea tipurilor de date

- Exprimata in biti
- Cu cat mai multi biti sunt alocati unui anumit tip de date cu atat mai multe valori pot fi exprimate
- Cu cat mai multi biti sunt alocati unui anumit tip de date cu atat necesita o memorie mai mare

Tipuri de date fundamentale in C++

Marimea (in biti)	Valori reprezentabile	Formula de calcul
8	256	2^8
16	65 536	2^{16}
32	4 294 967 296	2^{32}
64	18 446 744 073 709 551 615	2^{64}

Tipul de date **caracter (char)**

- Folosit pentru a reprezenta caractere singulare ‘A’, ‘x’, ‘@’
- Marimea pe care o ocupa este **1 byte** (8 biti) →256 de valori

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

int main()
{
    char variabila;
    variabila = 'a';
    cout << "Caracterul declarat este: " << variabila;
    return 0;
}
```

Caracterul declarat este: a

Tipul de date **intreg** (**`int`**)

- Folosit pentru a reprezenta numere intregi
- Versiuni: cu semn (**`signed`**) si fara semn (**`unsigned`**)
- In mod implicit, tipul de date **`int`** este **`signed`** *(cu semn)!*

Tipul de date **intreg** (`int`)

- *Valorile marimilor variabilelor pot diferi de cele din tabelele urmatoare in functie de compilatorul si calculatorul folosit*

Numele tipului	Marimea standard	Interval de referinta
`signed short int`	16 biti	-2^15= -32 768, …… , 2^15-1= 32 767
`signed int`	32 biti	-2^31= -2 147 483 648 ,….., 2^31-1=2 147 483 647
`signed long int`	32 biti	-2^31= -2 147 483 648, ….., -2^31-1= 2 147 483 647
`signed long long int`	64 biti	-2^63, …… ,(2^63)-1

Numele tipului	Marimea standard	Interval de referinta
`unsigned short int`	16 biti	0 …., 2^16-1= 65 535
`unsigned int`	32 biti	0 …. , 2^32-1= 4 294 967 295
`unsigned long int`	32 biti	0, …., 2^32-1= 4 294 967 295
`unsigned long long int`	64 biti	0, …. , 2^64-1=18 446 744 073 709 551 614

Tipul de date **intreg** (`int`)

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

int main()
{
    int variabila_1;
    signed int variabila_2;
    unsigned int variabila_3;
    variabila_1 = 10;
    variabila_2 = -5;
    variabila_3 = 200;
    cout << "Variabila_1 = " << variabila_1 << "\n";
    cout << "Variabila_2 = " << variabila_2 << "\n";
    cout << "Variabila_3 = " << variabila_3 << "\n";
}
```

variabila_1 = 10
variabila_2 = -5
variabila_3 = 200

Tipul de date **intreg**

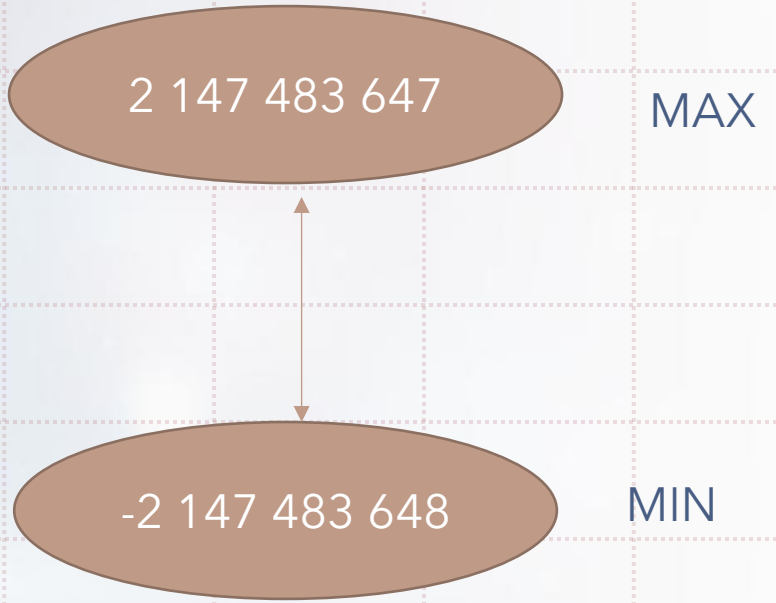
```cpp
#include<iostream>
using namespace std;

int main()
{
    int variabila_1;
    signed int variabila_2, variabila_3;
    variabila_1 = -2147483649;
    variabila_2 = 2147483648;
    variabila_3 = -2147483648;
    cout << "Variabila_1 = " << variabila_1 << "\n";
    cout << "Variabila_2 = " << variabila_2 << "\n";
    cout << "Variabila_3 = " << variabila_3 << "\n";

    return 0;
}
```

⋮

2 147 483 650 --- > - 2 147 483 646

2 147 483 649 --- > - 2 147 483 647

2 147 483 648 --- > - 2 147 483 648



variabila_1 = 2147483647
variabila_2 = -2147483648
variabila_3 = -2147483648

-2 147 483 649 --- > 2 147 483 647

-2 147 483 650 --- > 2 147 483 646

-2 147 483 651 --- > 2 147 483 647

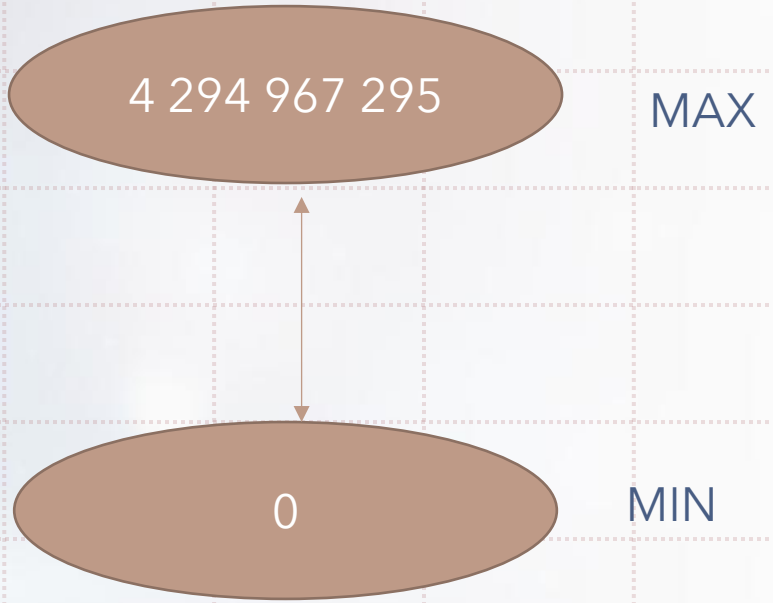
⋮

Tipul de date **intreg**

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

int main()
{
    unsigned int variabila_1, variabila_2, variabila_3,variabila_4;
    variabila_1 = -1;
    variabila_2 = 4294967295;
    variabila_3 = 0;
    variabila_4 = 4294967296;
    cout << "Variabila_1 = " << variabila_1 << "\n";
    cout << "Variabila_2 = " << variabila_2 << "\n\n";
    cout << "Variabila_3 = " << variabila_3 << "\n";
    cout << "Variabila_4 = " << variabila_4 << "\n";

    return 0;
}
```

```
Variabila_1 = 4294967295
Variabila_2 = 4294967295

Variabila_3 = 0
Variabila_4 = 0
```

⋮
4 294 967 298 --- > 2
4 294 967 297 --- > 1
4 294 967 296 --- > 0



-1 --- > 4 294 967 295
-2 --- > 4 294 967 294
-3 --- > 4 294 967 293
⋮

Tipul de date **real**

- Folosit pentru a reprezenta numere care nu sunt intregi

- Atunci cand numarul zecimalelor unui numar este infinit, computerul retine o aproximare a numarului (deoarece are o memorie finita).

- **IEEE 754** este **standardul internațional pentru reprezentarea și calculul numerelor reale (în virgulă mobilă) în calculatoare**.
- A fost publicat de **IEEE (Institute of Electrical and Electronics Engineers)** și este folosit de toate limbajele moderne: C, C++, Java, Python, etc.

- Practic, el definește **cum se stochează și cum se calculează numerele reale în memorie** — ca să existe rezultate **identice** indiferent de sistemul de operare, procesor sau compilator.
- Este standardul dominant; exista si alte reprezentari, dar nu la fel de mult folosite (reprezentarea fixa, reprezentarea zecimala, reprezentarea Binary-Coded Decimal, reprezentari logaritmice, rationale, reprezentatea Posit, s.am)

Tipul de date **real**

De ce a fost nevoie de acest standard?

Înainte de IEEE 754 (în anii ’70), fiecare calculator sau compilator avea **propriul mod** de a memora numere reale, ceea ce ducea la:

- erori diferite în calcule,
- rezultate incompatibile între platforme,
- dificultăți în schimbul de date binare.

IEEE 754 a uniformizat totul. De atunci, orice `double` sau `float` se comportă **exact la fel** pe orice sistem.

Structura unui număr IEEE 754

Forma matematica a unui numar real (care se poate normaliza) in sistem binar:

$$(-1)^{semn} \times (1 + mantisa\_stocata) \times 2^{exponent\_stocat - bias}$$

Această formulă e **baza întregului standard**.

Tipul de date **real**

Numele tipului	Marimea standard	Intervalul de referinta
`float`	32 biti	1.2×10^{-38} ––– 3.4×10^{38}
`double`	64 biti	2.2×10^{-308} ––– 1.8×10^{308}
`long double`	96 biti Pe Windows: 64 biti	3.3×10^{-4932} ––– 1.2×10^{4932} 2.2×10^{-308} ––– 1.8×10^{308}

❑ Pentru tipul **float**, cei 32 biti sunt impartiti astfel:
- Pentru semn --- 1 bit (0 = pozitiv, 1= negativ)
- Pentru exponent_stocat --- 8 biti
- Pentru mantisa_stocata (fractiune) --- 23 biti

❑ Pentru tipul **double**, cei 64 biti sunt impartiti astfel:
- Pentru semn --- 1 bit
- Pentru exponent_stocat --- 11 biti
- Pentru mantisa_stocata (fractiune) --- 52 biti

Cum sunt memorate numerele reale in calculator?

Exemple:

`1. double a= 7034.012` (in baza 10)

Pasul 1. Separam partea intreaga de partea fractionara: $7034.012 = 7034 + 0.012$

Pasul 2. Convertim partea intreaga in baza 2: 7034 (in baza 10) = 1101101111010 (in baza 2)

Pasul 3. Convertim partea fractionara in baza 2: 0.012 (in baza 10) ~ 0.0000001100….. (in baza 2)

Pasul 4. Combinam cele doua parti: 7034.012 (in baza 10) ~ 1101101111010. 0000001100…. (in baza 2)

Pasul 5. **Normalizam** (forma stiintifica binara): mutam virgula astfel incat sa avem un **singur "1"** inaintea ei

$1101101111010.\ 0000001100 = 1.10110111101000000001100\ldots \times 2^{12}$

Am mutat virgula cu 12 pozitii la stanga, deci **exponentul_real este 12** (daca ar fi trebuit sa mutam virgula spre dreapta cu 12 pozitii atunci exponentul_real ar fi fost -12)

Inainte de virgula, intotdeauna vom avea "1" si de aceea el nu se mai stocheaza in memorie si in formula matematica, cand reconsituim numarul real avem "1+mantisa" (vezi slide anterior)

Forma normalizata a numerelor reale memorate in C++

Pasul 6. Calculam campurile IEEE 754 (pentru `double`)

- **Semn:** numarul este pozitiv, deci semn = 0

- **Exponent:**

Exponent_real = 12. Standardul IEEE 754 pentru `double` foloseste un **bias (deplasare)** de 1023.
Exponent_ stocat = 12+1023=1035 (in baza 10)=10000001011 (in baza 2) (standardul aloca **11 biti;** daca mai trebuie adaugati biti atunci se mai adauga la stanga bitul 0, de cate ori este nevoie)

- **Mantisa_stocata:** luam doar partea de dupa virgula din forma normalizata,

Mantisa_stocata = 1011011110100000001100000010010011010110100011000100 (standardul retine primii **52 de biti** ai aceste secvente; daca sunt mai putini biti de 52, atunci se mai adauga la dreapta bitul 0, de cate ori este nevoie)

Pasul 7. Forma finala in memorie

```
[Semn][Exponent_stocat (11 biți)][Mantisa_stocata (52 biți)]
 0     10000001011    1011011110100000001100000010010011010110100011000100
```

Forma normalizata a numerelor reale memorate in C++

```
[Semn][Exponent_stocat (11 biți)][Mantisa_stocata (52 biți)]
 0     10000001011    1011011111010000000110000000100100110101101000110001 00
```

Cum se reconstruieste valoarea din memorie?

$$a = (-1)^{semn} \times (1 + mantisa\_stocata) \times 2^{exponent\_stocat - bias}$$

$= (-1)^0 \times (1 + 0.1011011111010000000110000000100100110101101000110001 00)$
$\times 2^{10000001011 - 1111111111}$

$= 1 \times 1.1011011111010000000110000000100100110101101000110001 00 \times 2^{12}$

$= 1101101111010.0000001100000001001001101011010001100010 0$ **(in baza 2)**

~ 7034. 0117538 (in baza 10)

Observatie! In calculator se pastreaza o aproximare a numarului 7034.012.

De ce apare acel **bias**?

Problema este că **exponentul poate fi negativ** (de exemplu, pentru numere mai mici decât 1).
Dar în memorie, **toți biții din câmpul exponent** sunt **biți fără semn (unsigned)**.
Adică nu avem un bit de semn pentru exponent.

Exemplu:
2^3 are exponent = +3
2^{-4} are exponent = −4

Calculatorul trebuie să poată reprezenta **ambele valori (pozitive și negative)** folosind doar **biți pozitivi**.

Soluția: folosim un *bias* (deplasare/translatie)
Ideea este simplă: se adaugă o constantă fixă (bias) la exponentul real, astfel încât toți exponenții să devină **numere pozitive**, deci ușor de stocat.

exponent_stocat = exponent_real + bias

In `float`: bias=127
In `double`: bias=1023

Gasiti reprezentarea lui “`float a=5.75;`” pe calculator!

Cum sunt memorate numerele reale in calculator?

Exemple: 2. `float` $a = 5.75$ (in baza 10)

Pasul 1. $5.75 = 5 + 0.75$

Pasul 2. 5 (in baza 10) = 101 (in baza 2)

Pasul 3. 0.75 (in baza 10) ~ 0.11 (in baza 2)

Pasul 4. 5.75 (in baza 10) ~ 101.11 (in baza 2)

Pasul 5. $101.11 = 1.0111 \times 2^2$

Pasul 6. **exponent_real**= 2; bias=127 (pentru float)

exponent_stocat = 2+127=129 (in baza 10)= 10000001 (in baza 2) (**8 biti**)

Semn= 0

Mantisa_stocata= 01110000000000000000000 (**23 biti** pentru float)

Pasul 7. Forma finala in memorie

```
[Semn][Exponent_stocat (8 biți)][Mantisa_stocata (23 biți)]
 0          10000001                    01110000000000000000000
```

Cum sunt memorate numerele reale in calculator?

```
[Semn][Exponent_stocat (8 biți)][Mantisa_stocata (23 biți)]
 0          10000001                    01110000000000000000000
```

$$a = (-1)^{semn} \times (1 + mantisa\_stocata) \times 2^{exponent\_stocat - bias}$$

a= 1 x 1.0111 x 2^2=101.11 (in baza 2) = 5.75 (in baza 10)

Exemple: 3. `float` $a = -5.75$ (in baza 10)

```
[Semn][Exponent_stocat (8 biți)][Mantisa_stocata (23 biți)]
 1          10000001                    01110000000000000000000
```

Cum sunt memorate numerele reale in calculator?

Exemple: 4. `float` $a = 0.125$ (in baza 10)

Pasul 1. $0.25 = 0 + 0.125$

Pasul 2. 0 (in baza 10) = 0 (in baza 2)

Pasul 3. 0.125 (in baza 10) ~ 0.001 (in baza 2)

Pasul 4. 0.125 (in baza 10) ~ 0.001 (in baza 2)

Pasul 5. $0.001 = 1.0 \times 2^{-3}$

Pasul 6. **exponent_real**= -3; bias=127 (pentru float)

exponent_stocat = -3+127=124 (in baza 10)= 1111100 (in baza 2) = 01111100 (8 biti)

Semn= 0

Mantisa_stocata= 00000000000000000000000 (23 biti pentru float)

Pasul 7. Forma finala in memorie

```
[Semn][Exponent_stocat (8 biți)][Mantisa_stocata (23 biți)]
 0          01111100                        00000000000000000000000
```

Cand nu se poate normaliza un numar?

Când numărul este **prea mic**, adică este mai aproape de zero decât **cea mai mică valoare normalizată** care poate fi reprezentată.

Pentru tipul float (32 biți): exponentul minim normalizat = 1 - bias = 1 - 127 = -126.
Asta înseamnă că:

$$x_{min,normalizat} = 1.0_2 \times 2^{-126} \approx 1.175 \times 10^{-38}$$

In acest caz, standardul IEEE 754, precizeaza ca un numar real, care nu se poate normaliza, se obtine prin

$$(-1)^{semn} \times (0 + mantisa\_stocata) \times 2^{1-bias}$$

Vezi programul “Vizualizare_biti”!

Tipul de date **real**

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

int main()
{
    float variabila_1;
    double variabila_2;
    variabila_1 = 3.1456898;
    variabila_2 = -4.5789065432;
    cout << "Variabila_1 = " << variabila_1 << "\n";
    cout << "Variabila_2 = " << variabila_2 << "\n";
    return 0;
}
```

Variabila_1 = 3.14569
Variabila_2 = -4.57891

Tipul de date **real**

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

int main()
{
    float variabila_1;
    double variabila_2;
    variabila_1 = 3.145689;
    variabila_2 = -4.5789065432;
    cout << "Variabila_1 = " << setprecision(9)<<variabila_1 << "\n";
    cout << "Variabila_2 = " << setprecision(17)<<variabila_2 << "\n";
    return 0;
}
```

Variabila_1 = 3.14568901 //se afiseaza cu 8 zecimale, dar doar primele **6 cifre sunt identice**

Variabila_2 = -4.5789065431999996 // se afiseaza cu 16 zecimale, dar ultima cifra a numarului declarat in program nu este afisata corect

Care este rezultatul acestui program?

```
#include<iostream>

using namespace std;

int main() {

    double a = 0.1, b = 0.2, c = 0.3;
    if (a + b == c)
        cout << "0.1+0.2 = 0.3" << endl;
    else
        cout<< "0.1+0.2 este diferit de 0.3!" << endl;

    return 0;
}
```

0.1 +0.2 este diferit de 0.3!

Tipul de date **boolean**

- Folosit pentru a reprezenta valorile **adevarat (true)** si **fals (false).**
- Valoarea **0** reprezinta **false;**
- Orice **valoare nenula** este **true**

Numele tipului	Marimea/precizia
bool	De obicei 8 biti **true** sau **false** (cuvinte cheie ale C++)

- **Vezi programul "TipuriDeDate"!**

Marimea unei variabile

Utilizarea operatorului **`sizeof`**

- Operatorul **`sizeof`** determina **marimea in bytes** a unui tip de date
- **Exemple:**

```
sizeof(int)
sizeof(double)

sizeof(numeVariabila)
sizeof numeVariabila
```

Utilizarea operatorului **`sizeof`**

- Operatorul **`sizeof`** primeste informatii de la doua fisiere C++ care sunt incluse in program

`<climits>` **`<cfloat>`**

- In versiunile mai vechi, aceste biblioteci trebuiau incluse explicit, acum nu mai este necesar.

- Aceste bibiloteci contin informatii despre marimea si precizia variabilelor

`INT_MAX` valoarea maxima care poate fi stocata intr-o variabila de tip int
`INT_MIN` valoarea minima care poate fi stocata intr-o variabila de tip int
`LONG_MIN` valoarea minima care poate fi stocata intr-o variabila de tip long int
`LONG_MAX` valoarea maxima care poate fi stocata intr-o variabila de tip long int
....

- **Vezi programul "OperatorulSizeOf"!**

Ce este o constanta?

- Constantele seamana foarte mult cu variabilele
- Au nume
- Ocupa spatiu
- Valorile lor sunt date de programator

Valoarea lor nu se mai poate modifica odata ce au fost declarate!

Exemplu: numarul lunilor intr-un an este o constanta

Tipuri de constante in C++

- Constante literale
- Constante declarate (folosind cuvantul cheie **`const`**)
- Expresii constante (folosind cuvantul cheie **`constexpr`**)
- Constante definite (**`#define`**)
- Enumerari (folosind cuvantul cheie **`enum`**)

Tipuri de constante in C++

Constante literale

- Constante literale intregi

`12` - un intreg
`12U` –un intreg fara semn
`12L` – un intreg long
`12LL` – un intreg long long

- Constante literale reale

`12.1` - un double
`12.1F` – un float
`12.1L` – un long double

- Constante literale caracter sunt cele scrise intre ghilimele simple ' ' si caracterele speciale

`\n` - linie noua
`\r` - return
`\t` - tab
`\b` - backspace

Exemplu:

```
cout << "Hello\tthere\nmy friend\n";
```

Hello there
my friend

Tipuri de constante in C++

Constante declarate

- Sunt constantele declarate folosind cuvantul cheie **`const`**
- Constantele trebuie initializate atunci cand sunt declarate sau in momentul executiei pentru ca altfel compilatorul va genera o eroare
- Daca incercam sa modificam valoarea unei constante atunci compilatorul va gerenera o eroare.
- **`const double pi {3.1415926};`**
- **`const int nrLuniAn {12};`**

Tipuri de constante in C++

Expresii constante

- Au fost introduse in C++11 si imbunatatite in C++14.
- Se introduc cu ajutorul cuvantului cheie **`constexpr`**
- Trebuie initializate atunci cand sunt declarate pentru ca altfel compilatorul va genera o eroare
- Spre deosebire de variabilele **`const`,** variabilelele **`constexpr`** trebuie sa fie initializate in momentul compilarii; Variabilele **`const`** pot fi initializate in momentul compilarii sau in timpul expecutiei programului.
- Tipul de date in expresiile constante trebuie sa fie **constante literale**
- Toate tipurile **`constexpr`** sunt **`const`**

```
constexpr float x = 43.0;
constexpr int y{ 108 };

constexpr int z; //Eroare
```

Tipuri de constante in C++

Expresii constante

- Exemplul 1

```
int m = 0;
const int n = m + 1;
```

n=1

- Exemplul 2

```
int m = 0;
constexpr int w = m+1;
```

Eroare

- Exemplul 3

```
constexpr int w = 0+1;
```

w=1

Tipuri de constante in C++

Constante definite

- Sunt constantele date cu ajutorul unei directive de precompilare

```
#define  pi  3.1415926;
```

- Prepocesorul inlocuieste pi cu valoarea respectiva oriunde o intalneste in cod (inlocuire “oarba”), dar cum preprocesorul nu intelege codul C++, este posibil ca uneori sa se genereze erori
- **Recomandare: Nu folosi prea des constante definite in C++!**

Declararea si utilizarea constantelor in C++

Problema:

Avem o firma care ofera servicii de menaj, cu urmatorul tarif:

Pret: 30 lei pe camera
Alte taxe (transport): 6%
Estimarea este valabila 30 zile.

In consola dorim ca posibilul client sa abia posibilitatea sa introduca numarul de camere in care doreste sa se faca menaj, dupa care sa se afiseze o aproximare a pretului acestor servicii:

Estimare:

Numar de camere: 3
Pret pe camera: 30 lei
Cost: 90 lei
Alte taxe:5.4 lei
==========
Cost total estimat: 95.4 lei
Estimarea este valabila 30 zile.

Vezi programul “ServiciiMenaj”!

Microsoft Visual Studio Debug Console

```
In cate camere mari doriti sa facem curatenie? 2
In cate camere mici doriti sa facem curatenie? 3

Estimarea pretului serviciilor noastre este urmatoarea:
Numarul de camere mari: 2
Numarul de camere mici: 3
Pretul pentru o camera mare: 35 lei
Pretul pentru o camera mica: 25 lei
Costul estimat pentru camerele mari: 70 lei
Costul estimat pentru camerele mici: 75 lei
Alte taxe: 8.7 lei
=======================================
Estimarea totala: 153.7 lei
Estimarea aceasta este valabila pentru 30 zile

C:\Users\nisto\OneDrive\Desktop\Curs C++\Curs4\Programe\Project3\Debug\Project3.exe (process 19252) exited with code 0.
Press any key to close this window . . .
```

Enumerari

- O **enumerare** este un tip de dată definit de programator prin care un şir de constante întregi, bine precizate, capată un tip comun, introdus prin cuvântul cheie **enum**.
- De exemplu, declaraţia

```
enum Logic {indecis, fals, adev};
```

defineşte un nou tip de dată, tipul `Logic`, fiecare variabilă de acest tip având numai una dintre cele trei valori: `indecis, fals` sau `adev`. O variabilă de tip `Logic` se declară, la fel ca orice alt tip de variabilă:

```
Logic omega;
```

şi poate fi initializată astfel:

```
omega=fals;
```

Enumerari

- Avem şi urmatoarea variantă de declaraţie:

```
enum Logic {indecis, fals, adev} omega;
```

şi chiar şi varianta cu iniţializare:

```
enum Logic {indecis, fals, adev} omega=fals;
```

- Enumerările sunt utilizate pentru a da mai multă claritate programului. Ele sunt de fapt nişte codificari numerice. In exemplul precedent, cele trei valori, indecis, fals şi adev, sunt tratate de compilator drept identificatori de constante întregi cu trei valori implicite în ordine:

```
indecis=0, fals=1 şi adev=2.
```

Enumerari

- Programatorul poate impune alte valori, dar numai în momentul declaraţiei, în modul urmator:

```
enum Fuzzy{imposibil,neverosimil,incert=5,plauzibil,sigur=10};
```

Acum avem

```
imposibil=0, neverosimil=1, incert=5, plauzibil=6, sigur=10.
```

Valorile care nu sunt impuse de programator se obţin prin incrementarea celor precedente cu câte o unitate. Prima valoare este în mod implicit zero.

- Programatorul trebuie să se asigure că valorile obţinute sunt distincte, altfel utilitatea codificării este îndoielnică.

- In expresii aritmetice orice dată de tip enumerare este tratată ca un număr întreg, conversia către int fiind **implicită**, dar conversia unui întreg către un tip enumerat trebuie cerută **explicit**.

Vezi programul Enumerari-Exemplul1!

https://test-master.space