



Tutoriat 1

Ana-Maria Rusu & Ionuț-Daniel Nedelcu

Grupa 242

Facultatea de Matematică și Informatică a Universității din București

Mențiune:

**toate programele sunt rulate în Visual Studio Code
compilator g++, comanda terminal: g++ sursa.cpp -o sursa && ./sursa**

Ce vom face azi?

Referințe

Pointeri

Alocarea
dinamică

Clase &&
obiecte

Exerciții



01

REFERINȚE

Referințele

→ **Definiție:** referințele sunt variabile ce conțin **adresa unei zone de memorie**. Altfel spus, ele reprezintă aliasuri (adică alte denumiri) ale unor variabile deja existente.

→ **Sintaxa:** `tip_de_date &nume_referinta = valoare;`

→ **Utilizare:** folosim o referință mai ales pentru a evita construirea unui nou obiect (de exemplu, uneori într-o funcție dorim să transmitem obiectul original, ci nu o copie a sa. Vom vedea această utilitate pe parcursul materiei).

Proprietăți

1) o referință trebuie sa fie inițializată în momentul declarării

```
#include<iostream>
using namespace std;
int main()
{
    int numar = 5;
    int& referinta_numar; //referinta_numar' declared as reference but not initialized
    referinta_numar=numar;
    //corect este : int& referinta_numar=numar

    return 0;
}
```

Proprietăți

2)referințele nu pot fi reasignate

```
int n=5, m=10;  
int &p=n;  
int &p=m; //eroare : redeclaration of 'int& p'
```

3)referinta nu poate sa fie nula

Exemplu

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

int main()
{
    int numar = 5;
    int& referinta_numar = numar;
    cout<<"valoarea lui numar: "<<numar<<endl;
    cout<<"adresa lui numar: "<<&numar<<endl;
    cout<<"valoarea lui referinta_numar: "<<referinta_numar<<endl;
    cout<<"adresa lui referinta_numar: "<<&referinta_numar<<endl;
    /*
    In memorie avem:
    variabila    numar        referinta_numar
    valoare      5             5
    adresa       0x5ff394      0x5ff394

    numar si referinta_numar se afla la aceeasi adresa de memorie
    */
```


Exemplu

```
numar=7;
cout<<"valoarea lui numar: "<<numar<<endl; //7
cout<<"adresa lui numar: "<<&numar<<endl; //0x5ffe94
cout<<"valoarea lui referinta_numar: "<<referinta_numar<<endl; //7
cout<<"adresa lui referinta_numar: "<<&referinta_numar<<endl; //0x5ff394

referinta_numar=9;
cout<<"valoarea lui numar: "<<numar<<endl; //9
cout<<"adresa lui numar: "<<&numar<<endl; //0x5ffe94
cout<<"valoarea lui referinta_numar: "<<referinta_numar<<endl; //9
cout<<"adresa lui referinta_numar: "<<&referinta_numar; //0x5ff394
return 0;
}
```



02

POINTERI

Pointeri

→ **Definiție:** Pointerii sunt variabile care **conțin adresa unei zone de memorie**.

→ **Sintaxa:** `tip_de_date* nume_pointer = valoare;`

→ **În lucrul cu pointeri se folosesc doi operatori unari:**

& -> extragerea adresei unei variabile

***** -> referirea conținutului zonei de memorie indicate de pointer

Proprietăți

- 1) un pointer poate fi declarat NULL (ex: `int* p=nullptr;` sau `int* p=NULL;`)
- 2) un pointer poate primi o valoare mai târziu în program, nu neapărat la declarare (adică putem modifica valoarea pointerilor).
- 3) tipul pointerului trebuie să fie același cu tipul datei către care pointează

Exemplu

```
#include<iostream>
using namespace std;
int main()
{
    int numar=5;
    int *p=&numar;
    cout<<"numar: "<<numar<<endl;//5
    cout<<"p: "<<p<<endl;//0x5ffe9c
    cout<<"adresa numar: "<<&numar<<endl;//0x5ffe9c
    cout<<"adresa p: "<<&p<<endl;//0x5fff90
    cout<<"*p: "<<*p<<endl;//5
```

Exemplu

```
/*  
In memorie avem:  
variabila      numar      p  
valoare        5          5  
adresa         0x5ffe9c    0x5fff90  
  
numar si p se afla la adrese diferite!  
*/  
    *p=10;  
    cout<<"numar: "<<numar<<endl;//10  
    cout<<"p: "<<p<<endl;//0x5ffe9c  
    cout<<"adresa numar: "<<&numar<<endl;//0x5ffe9c  
    cout<<"adresa p: "<<&p<<endl;//0x5ffe9c  
    cout<<"*p: "<<*p<<endl;//10  
  
//pentru a intelege, putem sa spunem ca p STOCHEAZA adresa de memorie, iar *p este valoarea DE  
//LA adresa de memorie indicata de p.  
    return 0;  
}
```



03

Alocarea Dinamică



Din seria amintiri din
Colocviu P00 iunie
2024 color... vor urma
și alte meme-uri



De ce alocare dinamică?

→ Utilizarea variabilelor dinamice reprezintă o tehnică numită **alocare dinamică**.

→ Ea se implementează prin intermediul **pointerilor** și are avantajul că folosește **doar atâta memorie cât este necesară**.

Sintaxa

→ Alocarea dinamică a memoriei în C++ se face utilizând operatorii (cuvintele cheie) **new** și **delete**.

→ Sintaxa:

- **Alocarea a unui element** : `tip_date pointer = new tip_date(initializare);`
- **Dezalocarea a unui element** : `delete pointer;`
- **Alocarea unui array** : `tip_date pointer = new tip[nr_elemente];`
- **Dezaloarea unui tablou array** : `delete[] pointer;`

Exemplu

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

int main()
{

    int* n = new int (3);
    cout<<"n: "<<n<<endl; //adresa lui n
    cout<<"*n: "<<*n<<endl; //valoarea lui n, 3

    delete n;
    cout<<"n: "<<n<<endl; //adresa lui n
    cout<<"*n: "<<*n<<endl; //o valoare random din memorie ( de ce? pentru ca inca nu i-am
    atribuit o valoare, doar am alocat o zona de memorie)
```

Exemplu

```
int* array= new int[4];
cout<<"array: "<<array<<endl;// adresa la care incepe array-ul
cout<<"*array: "<<*array<<endl; // alta valoare random din memorie

for(int i=0; i<4;i++)
    array[i]=i;

for(int i=0;i<4;i++)
{
    cout<<"*array["<<i<<"]: "<<*(array+i)<<endl;
    // sau cout<<"array["<<i<<"]: "<<array[i]<<endl;
    cout<<"array["<<i<<"]: "<< array+i<<endl;
}
```

Exemplu

```
/*
```

in memorie:

variabila	array[0]	array[1]	array[2]	array[3]
valoare	0	1	2	3
adresa	0x5ff370	0x5ff374	0x5ff378	0x5ff37c

zona de memorie este continua. astfel elementele din array vor fi stocate la diferenta de 4 octeti unele de altele (vezi diferenta dintre oricare 2 adrese consecutive de mai sus)

```
*/
```

```
delete[] array;
```

```
return 0;
```

```
}
```

04

Clase și Obiecte

Programare Orientată pe Obiecte



Moștenire

Abstractizare



Încapsulare

Polimorfism



Obiecte

→ **Definiție: obiectul** este un element al unei clase, care are stare (date membre - variabile) și acțiuni (metode membre - funcții/metode). Acțiunile sunt considerate interfața, iar starea e considerată partea “ascunsă” de utilizator.

→ **Definiție: clasa** definește atributele și metodele obiectului. Practic, aceasta menționează proprietățile obiectelor din ea.

→ Clasele sunt asemănătoare cu “struct”, diferența fiind accesul la date.

Clase

Modificatori de acces

- In C++ avem 3 modificatori de acces: private, protected, public
- Ei exprimă modul în care putem accesa datele/metodele unui obiect
- De exemplu, class are acces default private, iar struct public

<u>Avem acces?</u>	public	protected	private
Aceeasi clasa	da	da	da
Clase derivate	da	da	nu
Alte clase	da	nu	nu

private

Acces: aceeași clasă

protected

Acces: aceeași clasă &
Clase derivate

public

Acces: aceeași clasă &
Clase derivate &
Alte clase

Exemplu

```
#include<iostream>
using namespace std;
class A
{
    private:
        int x=3;
    public:
        int y;
        int valX() { return x;}
    private:
        int valY() { return y;}
};
```

```
int main()
{
    A a;
    a.y=5;
    cout<<a.valX()<<endl;
    //cout<<a.valY(); vom obtine o eroare
    //pentru ca metoda valY este private
}
```

→ variabilele x și y ale obiectului a (datele membre) din clasa A, se accesează scriind a.x, respectiv a.y
→ nu putem accesa variabila x în mod direct, deoarece aceasta este într-o secțiune privată a clasei. În schimb, cu ajutorul funcției valX(), care este publică, putem avea acces la ea (getter - noțiune viitoare)
→ pentru variabila y lucrurile stau invers: variabila y e publică, iar metoda valY() e privată, deci nu o putem accesa. (OBS: nu e o practică bună. Variabilele nu trebuie să fie publice)

Încapsularea datelor

→ Mecanismul prin care datele și funcțiile sunt plasate în aceeași structură (clasă) și stabilirea nivelului de acces la conținutul acesteia.

→ **Încapsularea datelor** (adică protejarea/ascunderea lor) e foarte importantă. Nu am vrea ca un utilizator să aibă acces la informații confidențiale.

→ Variabilele declarate în clase sunt private (by default) și e important să rămână așa. Pentru a le accesa/modifica, vom folosi funcții publice numite **getteri/setteri**. (OBS: cu getteri vom vedea valorile variabilelor, iar cu setteri modificăm valorile acestora)

Exemplu

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

class Carte
{
private:
    string titlu = "Exemplu";
    string autor = "Eu";
    int an;
public:
    int setAn(int val) {an = val;}
    string getTitlu() {return titlu;}
    string getAutor() {return autor;}
    int getAn() {return an;}
};
```

```
int main()
{
    Carte a, b;
    a.setAn(2000);
    b.setAn(2005);
    cout << a.getTitlu() << " ";
    cout << b.getAutor() << " ";
    cout << a.getAn() << " " << b.getAn();
    return 0;
}
```

Compunerea/Agregarea

→ Se referă la definirea unui obiect ce îl include pe altul deja existent, din altă clasă

→ Indică existența unui obiect dependentă de alt obiect

→ **Avantaje:**

- **Reutilizarea codului:** Obiectele mai mici pot fi reutilizate în mai multe clase.
- **Modularitate:** Proiectarea claselor devine mai modulară și mai flexibilă, prin combinarea componentelor mici și independente.
- **Încapsularea:** Obiectele componente pot fi ascunse de utilizatori, furnizând doar interfețele necesare pentru a interacționa cu ele.

Exemplu

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

class Motor
{
public:
    void pornesteMotor(){cout << "Motorul a pornit." << endl;}
};

class Masina
{
private:
    Motor motor;
public:
    void pornesteMasina()
    {
        motor.pornesteMotor();
        cout << "Masina a pornit." << endl;
    }
};
```

Exemplu

```
int main()
{
    Masina masina;
    masina.pornesteMasina();
    return 0;
}
```

```
class A
{
    A a; // aici avem eroare // corect este A *a;
}
// exemplu separat de Masina si Motor
```

- Clasa Masina conține în definiția sa un obiect de tip Motor
- Spunem că un obiect de tip Masina este creat prin compunere, deoarece este creat pe baza unui tip de obiect deja existent
- **Important:** Nu putem să avem în definiția clasei un obiect de tipul clasei, doar un pointer catre acesta.



05

Exerciții