



# **Programare orientată pe obiecte**

**- suport de curs -**

**Andrei Păun**  
**Anca Dobrovăț**

**An universitar 2021 – 2022**

**Semestrul II**

**Seriile 13, 14, 15**

**Curs 2**



## **Agenda cursului**

- Recapitularea discuțiilor din cursul anterior  
(Generalități despre curs, Reguli de comportament)
- Generalități despre OOP (Principiile programării orientate pe obiecte)
  - Clase, obiecte, modificatori de acces, funcții și clase prieten, constructori / destructor



## Generalități despre curs

1. Curs – seria 13: luni (10 -12), seria 14: marti (8 – 10), seria 15: vineri (12 - 14)
2. Laborator – pe semigrupe, in fiecare saptamana
3. Seminar - o data la 2 saptamani

**COLOCVIU: 23 mai 2022 (ora 8) – se sustine fizic in facultate**

**EXAMEN SCRIS: 14 iunie 2022 (ora 9) – se sustine fizic in facultate**

Daca cineva are o problema cu aceste date il/o rog sa ne anunte

In 2 saptamani datele acestea sunt fixate/finalizate



## 1. Scurta recapitulare C++

- Bjarne Stroustrup în 1979 la Bell Laboratories in Murray Hill, New Jersey
- 5 revizii: 1998 ANSI+ISO, 2003 (corrigendum), 2011 (C++11/0x), 2014, 2017 (C++ 17/1z)
- Următoarea plănuită în 2020 (C++2a)
- Versiunea 1998: Standard C++, C++98



## 1. Scurta recapitulare C++

- C++98: a definit standardul inițial, toate chestiunile de limbaj, STL
- C++03: bugfix o unică chestie nouă: value initialization
- C++11: initializer lists, rvalue references, moving constructors, lambda functions, final, constant null pointer, etc.
- C++14: generic lambdas, binary literals, auto, variable template, etc.



## 1. Scurta recapitulare C++

C++17:

- If constexpr()
- Inline variables
- Nested namespace definitions
- Class template argument deduction
- Hexadecimal literals
- etc

**typename** is permitted for template template parameter declarations (e.g.,

**template**<**template**<typename> **typename** X> **struct** ...)



## 1. Scurta recapitulare C++

- `<iostream>` (fără `.h`)
- `using namespace std;`
- `cout, cin` (fără `&`)
- `//` comentarii pe o linie
- declarare variabile
- Tipul de date `bool`
  - se definesc `true` și `false` (1 și 0);
  - C99 nu îl definește ca `bool` ci ca `_Bool` (fără `true/false`);
  - `<stdbool.h>` pentru compatibilitate.



## 1. Scurta recapitulare C++

### **Supraîncărcarea funcțiilor (un caz de *Polimorfism la compilare*)**

Utilizarea mai multor funcții care au același nume

Identificarea se face prin numărul de parametri și tipul lor.

**Tipul de întoarcere nu e suficient pentru a face diferența**

Simplicitate/corectitudine de cod





## 1. Scurta recapitulare C++

### Funcții cu valori implicite

Într-o funcție se pot declara valori implicite pentru unul sau mai mulți parametri.

La apel se poate omite specificarea valorii pentru acei parametri formali care au declarate valori implicite.

Argumentele cu valori implicite trebuie să fie amplasate la sfârșitul listei.

Valorile implicite se specifică o singură dată în definiție (de obicei în prototip).



## 1. Scurta recapitulare C++

### Funcții cu valori implicite

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

```
void f (int a, int b = 12); // prototip cu mentionarea valorii implicite pentru b
```

```
int main()
{
    f(1);
    f(1,20);
    return 0;
}
```

```
void f (int a, int b) { cout<<a<<" - "<<b<<endl; }
```



## 1. Scurta recapitulare C++

### Alocare dinamica

```
int *pi;
```

```
pi=new int;
```

```
delete pi; // elibereaza zona adresata de pi -o considera neocupata
```

```
pi=new int(2);// alocata zona si initializeaza zona cu valoarea 2
```

```
pi=new int[2]; // alocata un vector de 2 elemente de tip intreg
```

```
delete [ ] pi; //elibereaza intreg vectorul
```

```
//-pentru new se foloseste delete
```

```
//- pentru new [ ] se foloseste delete [ ]
```



## 1. Scurta recapitulare C++

### Tipul referinta

O referință este, in esenta, un pointer implicit, care actioneaza ca un alt nume al unui obiect (variabila).

```
int i;  
int *pi,j;
```

**int & ri=i; //ri este alt nume pentru variabila i**

```
pi=&i; // pi este adresa variabilei i  
*pi=3; //in zona adresata de pi se afla valoarea 3
```

**Pentru a putea fi folosită, o referință trebuie inițializată in momentul declararii, devenind un alias (un alt nume) al obiectului cu care a fost inițializată.**



## 1. Scurta recapitulare C++

### Tipul referinta

```
int a = 20;  
int& ref = a;  
cout<<a<<" "<<ref<<endl; // 20 20
```

**ref++;**

```
cout<<a<<" "<<ref<<endl; // 21 21
```

**Obs: spre deosebire de pointeri care la incrementare trec la un alt obiect de acelasi tip, incrementarea referintei implica, de fapt, incrementarea valorii referite.**



## 1. Scurta recapitulare C++

### Tipul referinta

```
int a = 20;  
int& ref = a;  
cout<<a<<" "<<ref<<endl; // 20 20
```

```
int b = 50;  
ref = b;
```

```
ref--;  
cout<<a<<" "<<ref<<endl; // 49 49
```

**Obs: in afara initializarii, nu puteti modifica obiectul spre care indica referinta.**



## 1. Scurta recapitulare C++

### Tipul referinta

- o referinta trebuie să fie initializata când este definita, dacă nu este membra a unei clase, un parametru de functie sau o valoare returnata;
- referintele nule sunt interzise intr-un program C++ valid.
- nu se poate obtine adresa unei referinte.
- nu se pot crea tablouri de referinte.
- nu se poate face referinta catre un camp de biti.



## 1. Scurta recapitulare C++

### Transmiterea parametrilor

C

```
void f(int x)
{   x = x *2;}
```

```
void g(int *x)
{   *x = *x + 30;}
```

```
int main() {
    int x = 10;
    f(x);
    printf("x = %d\n",x);
    g(&x);
    printf("x = %d\n",x);
    return 0;
}
```

C++

```
void f(int x){   x = x *2;} //prin valoare
void g(int *x){  *x = *x + 30;} // prin pointer
void h(int &x){ x = x + 50;} //prin referinta
```

```
int main() {
    int x = 10;
    f(x);
    cout<<"x = "<<x<<endl;
    g(&x);
    cout<<"x = "<<x<<endl;
    h(x);
    cout<<"x = "<<x<<endl;
    return 0;
}
```





## 1. Scurta recapitulare C++

### Transmiterea parametrilor

#### Observatii generale

- parametrii formali - sunt creati la intrarea intr-o functie si distrusi la retur;
- apel prin valoare - copiaza valoarea unui argument intr-un parametru formal  $\Rightarrow$  modificarile parametrului nu au efect asupra argumentului;
- apel prin referinta - in parametru este copiata adresa unui argument  $\Rightarrow$  modificarile parametrului au efect asupra argumentului.
- functiile, cu exceptia celor de tip void, pot fi folosite ca operand in orice expresie valida.



## 1. Scurta recapitulare C++

**Cand tipul returnat de o functie nu este declarat explicit, i se atribuie automat int.**

Tipul trebuie cunoscut inainte de apel.

```
f (double x) {  
    return x;  
}
```

**Prototipul unei functii: permite declararea in afara si a numarului de parametri / tipul lor:**

```
void f(int); // antet / prototip  
int main() { cout<< f(50); }  
void f( int x) { // corp functie; }
```



## **2. Principiile programarii orientate pe obiecte**

**Clasa**

**Obiect**

**Incapsularea**

**Modularitatea**

**Ierarhizarea.**

**Polimorfism la compilare si polimorfism la executie, etc.**



## 2. Principiile programarii orientate pe obiecte

### Obiecte

- au stare și acțiuni (metode/funcții)
- au interfață (acțiuni) și o parte ascunsă (starea)
- Sunt grupate în clase, obiecte cu aceleași proprietăți
- Un **program orientat obiect** este o colecție de obiecte care interactionează unul cu celălalt prin mesaje (aplicand o metodă).



## 2. Principiile programarii orientate pe obiecte

### Clase

O **clasa** definește atribute și metode.

```
class X{  
    //date membre  
    //metode (functii membre – functii cu argument  
implicit obiectul curent)  
};
```

- menționează proprietățile generale ale obiectelor din clasa respectivă
- folosite la încapsulare (ascunderea informației)
- reutilizare de cod: moștenire



## 2. Principiile programarii orientate pe obiecte

### Clase

- cu “class”
- obiectele instanțiază clase
- similare cu struct-uri și union-uri
- au funcții
- specificatorii de acces: public, private, protected
- default: private
- protected: pentru moștenire, vorbim mai târziu



## 2. Principiile programarii orientate pe obiecte

### Clase

```
class nume_clasă {  
    private variabile și funcții membru  
    specificator_de_acces:  
        variabile și funcții membru  
    specificator_de_acces:  
        variabile și funcții membru  
    // ...  
    specificator_de_acces:  
        variabile și funcții membru  
} listă_obiecte;
```

- putem trece de la public la private și iar la public, etc.



## 2. Principiile programarii orientate pe obiecte

### Clase

```
class employee {  
    // private din oficiu  
        char name[80];  
public:  
    // acestea sunt publice  
        void putname(char *n);  
        void getname(char *n);  
private:  
    // acum din nou private  
        double wage;  
public:  
    // înapoi la public  
        void putwage(double w);  
        double getwage();  
};
```

```
class employee {  
        char name[80];  
        double wage;  
public:  
        void putname(char *n);  
        void getname(char *n);  
        void putwage(double w);  
        double getwage();  
};
```





## 2. Principiile programarii orientate pe obiecte

### Clase

- se folosește mai mult a doua variantă
- un membru (ne-static) al clasei nu poate avea inițializare
- nu putem avea ca membri obiecte de tipul clasei (putem avea pointeri la tipul clasei)
- nu auto, extern, register



## 2. Principiile programarii orientate pe obiecte

### Clase

- variabilele de instanta (instance variables)
- membri de tip date ai clasei
  - in general private
  - pentru viteza se pot folosi “public” dar **NU LA ACEST CURS**



## 2. Principiile programarii orientate pe obiecte

### Clase

```
#define SIZE 100
// This creates the class stack.
class stack {
    int stck[SIZE];
    int tos;
public:
    void init();
    void push(int i);
    int pop();
};
```

- init(), push(), pop() sunt funcții membru
- stck, tos: variabile membru



## 2. Principiile programarii orientate pe obiecte

### Clase

- se creează un tip nou de date
- un obiect instanțiază clasa
- funcțiile membru sunt date prin semnătură
- pentru definirea fiecărei funcții se folosește ::

**stack** mystack;

```
void stack::push(int i) {  
    if(tos==SIZE) {  
        cout << "Stack is full.\n";  
        return; }  
    stck[tos] = i;  
    tos++;  
}
```



## 2. Principiile programarii orientate pe obiecte

### Clase

- :: scope resolution operator
- și alte clase pot folosi numele push() și pop()
- după instantiere, pentru apelul push()

**stack mystack;**

- mystack.push(5);
- programul complet în continuare



```
#include <iostream>
using namespace std;
#define SIZE 100
// This creates the class stack.
class stack {
    int stck[SIZE];
    int tos;

public:
    void init();
    void push(int i);
    int pop();
};

void stack::init()
{
    tos = 0;
}

void stack::push(int i) {
    if(tos==SIZE) {
        cout << "Stack is full.\n";
        return;
    }
    stck[tos] = i;
    tos++;
}
```

```
int stack::pop()
{
    if(tos==0) {
        cout << "Stack underflow.\n";
        return 0;
    }
    tos--;
    return stck[tos];
}

int main()
{
    stack stack1, stack2; // create two stack objects
    stack1.init();
    stack2.init();
    stack1.push(1);
    stack2.push(2);
    stack1.push(3);
    stack2.push(4);
    cout << stack1.pop() << " ";
    cout << stack1.pop() << " ";
    cout << stack2.pop() << " ";
    cout << stack2.pop() << "\n";
    return 0;
}
```



## 2. Principiile programarii orientate pe obiecte

### **Incapsularea**

- ascunderea de informații (data-hiding);
- separarea aspectelor externe ale unui obiect care sunt accesibile altor obiecte de aspectele interne ale obiectului care sunt ascunse celorlalte obiecte;
- definește apartenența unor proprietăți și metode față de un obiect;
- doar metodele proprii ale obiectului pot accesa starea acestuia.



## 2. Principiile programarii orientate pe obiecte

### Incapsularea (ascunderea informatiei)

foarte importanta

**public, protected, private**

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





## 2. Principiile programarii orientate pe obiecte

### Incapsularea (ascunderea informatiei)

```
class Test {  
    private:  
    int x;  
    public:  
    void set_x(int a) {x = a;}  
};
```

```
int main() {  
    Test t;  
    t.x = 34; // inaccesibil  
    t.set_x(34); // ok  
    return 0;  
}
```



## 2. Principiile programarii orientate pe obiecte

- **Agregarea** (ierarhia de obiecte) compunerea unui obiect din mai multe obiecte mai simple.  
(relație de tip “has a”)
- **Moștenirea** (ierarhia de clase) relație între clase în care o clasă mosteneste structura și comportarea definită în una sau mai multe clase  
(relație de tip “is a” sau “is like a”);



## 2. Principiile programarii orientate pe obiecte

### Agregarea

```
class Profesor
{
    string nume;
    int vechime;
};
```

```
class Curs
{
    string denumire;
    Profesor p;
};
```



## 2. Principiile programarii orientate pe obiecte

### Moștenire

- multe obiecte au proprietăți similare
- reutilizare de cod



## 2. Principiile programarii orientate pe obiecte

### Moștenire

- terminologie
  - clasă de bază, clasă derivată
  - superclasă subclasă
  - părinte, fiu
- mai târziu: funcții virtuale, identificare de tipuri în timpul rulării (RTTI)



## 2. Principiile programarii orientate pe obiecte

### Moștenire

- încorporarea componentelor unei clase în alta
- refolosire de cod
- detalii mai subtile pentru tipuri și subtipuri
- clasă de bază, clasă derivată
- clasa derivată conține toate elementele clasei de bază, mai adăugă noi elemente



## 2. Principiile programarii orientate pe obiecte

```
class building {  
    int rooms;  
    int floors;  
    int area;  
public:  
    void set_rooms(int num);  
    int get_rooms(); // ...  
};
```

### Moștenire

**tip acces: public, private, protected**

mai multe mai târziu

// house e derivată din building

```
class house : public building {  
    int bedrooms;  
    int baths;  
public:  
    void set_bedrooms(int num);  
    int get_bedrooms();};
```

public: membrii publici ai building  
devin publici pentru house



## 2. Principiile programarii orientate pe obiecte

### Moștenire

- house NU are acces la membrii privați ai lui building
- așa se realizează encapsularea
- clasa derivată are acces la membrii publici ai clasei de baza și la toți membrii săi (publici și privați)





## 2. Principiile programarii orientate pe obiecte

```
class building {  
    int rooms;  
    int floors;  
    int area;  
  
    public:  
        void set_rooms(int num);  
        int get_rooms(); //...};
```

```
class house : public building {  
    int bedrooms;  
    int baths;  
  
    public:  
        void set_bedrooms(int num);  
        int get_bedrooms();  
        void set_baths(int num);  
        int get_baths();  
  
};
```

### Moștenire

// school este de asemenea derivată  
din building

```
class school : public building {  
    int classrooms;  
    int offices;  
  
    public:  
        void set_classrooms(int num);  
        int get_classrooms();  
        void set_offices(int num);  
        int get_offices();  
};
```



## Moștenire

```
void building::set_rooms(int num)
{ rooms = num; }
void building::set_floors(int num)
{ floors = num; }
void building::set_area(int num)
{ area = num; }
int building::get_rooms()
{ return rooms; }
int building::get_floors()
{ return floors; }
int building::get_area()
{ return area; }
void house::set_bedrooms(int num)
{ bedrooms = num; }
void house::set_baths(int num)
{ baths = num; }
int house::get_bedrooms()
{ return bedrooms; }
int house::get_baths()
{ return baths; }
void school::set_classrooms(int num)
{ classrooms = num; }
void school::set_offices(int num)
{ offices = num; }
int school::get_classrooms()
{ return classrooms; }
int school::get_offices()
{ return offices; }
```

```
int main()
{
}
}
```

```
house h;
school s;
h.set_rooms(12);
h.set_floors(3);
h.set_area(4500);
h.set_bedrooms(5);
h.set_baths(3);
cout << "house has " << h.get_bedrooms();
cout << " bedrooms\n"; s.set_rooms(200);
s.set_classrooms(180);
s.set_offices(5);
s.set_area(25000);
cout << "school has " << s.get_classrooms();
cout << " classrooms\n";
cout << "Its area is " << s.get_area();
return 0;
```

house has 5 bedrooms  
school has 180 classrooms  
Its area is 25000



## 2. Principiile programarii orientate pe obiecte

### Polimorfism

- tot pentru claritate/ cod mai sigur
- Polimorfism la compilare: ex. `max(int)`, `max(float)`
- Polimorfism la execuție: RTTI



## 2. Principiile programarii orientate pe obiecte

### Șabloane

- din nou cod mai sigur/reutilizare de cod
- putem implementa listă înlănțuită de
  - întregi
  - caractere
  - float
  - obiecte



# Perspective

## Curs 3

- Class, struct, union
- Functii si clase prieten
- Functii inline
- Constructori / destructor