Facultatea de Matematică şi Informatică
Universitatea din Bucureşti

Programare orientată pe obiecte

- suport de curs -

Andrei Păun
Anca Dobrovăț

An universitar 2021 – 2022
Semestrul II
Seriile 13, 14 si 15

Curs 1

Generalităţi 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

4. Prezenta la curs/seminar: nu e obligatorie!

Laborator – OBLIGATORIU

Să ne cunoaștem

Cine predă?

Curs: Anca Dobrovăț (13, 15) si Andrei Păun (14)

- anca.dobrovat@fmi.unibuc.ro
- apaun@fmi.unibuc.ro

Seminar: Daniela Cheptea (13), Florin Bilbie (14), TBA (15)

Laboratoare:

- Marius Micluta – Campeanu
- Eduard Stefan Deaconu
- Eduard Szmeteanca
- Octavian Comanescu
- Alexandra Murgoci

Agenda cursului

1. Regulamente UB si FMI

2. Utilitatea cursului de Programare Orientata pe Obiecte

3. Prezentarea disciplinei

4. Primul curs

Agenda cursului

1. Regulamente UB si FMI

2. Utilitatea cursului de Programare Orientata pe Obiecte

3. Prezentarea disciplinei

4. Primul curs

1. Regulamente UB si FMI

Lucruri bine de stiut de studenti:

- regulament privind activitatea studenților la UB: https://www.unibuc.ro/wp-content/uploads/sites/7/2018/07/Regulament-privind-activitatea-profesionala-a-studentilor-2018.pdf
- regulament de etică și profesionalism la FMI:

http://fmi.unibuc.ro/ro/pdf/2015/consiliu/Regulament_etica_FMI.pdf

Se consideră incident minor cazul în care un student/ o studentă:
a. preia codul sursă/ rezolvarea unei teme de la un coleg/ o colegă și pretinde că este rezultatul efortului propriu;

Se consideră incident major cazul în care un student/ o studentă:
a. copiază la examene de orice tip;

3 incidente minore = un incident major = exmatriculare

Agenda cursului

1. Regulamente UB si FMI

2. Utilitatea cursului de Programare Orientata pe Obiecte

3. Prezentarea disciplinei

4. Primul curs

2. Utilitatea cursului de Programare Orientata pe Obiecte

PYPL PopularitY of Programming Language

Captura din: http://pypl.github.io/PYPL

Worldwide, Feb 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.52 %	-1.7 %
2		Java	18.12 %	+1.2 %
3		JavaScript	8.9 %	+0.4 %
4	↑	C/C++	7.62 %	+1.1 %
5	↓	C#	7.39 %	+0.6 %
6		PHP	5.81 %	-0.3 %
7		R	4.04 %	+0.2 %
8		Objective-C	2.46 %	-1.1 %
9		Swift	2.03 %	+0.0 %

Majoritatea pot fi considerate limbaje OO.

Limbaje destul de cunoscute care nu sunt OO sunt Go, Julia și Rust

2. Utilitatea cursului de Programare Orientata pe Obiecte

Paradigme de programare → Stil fundamental de a programa

Dictează:

- **Cum se reprezintă datele problemei** (variabile, funcții, obiecte, fapte, constrângeri etc.)
- **Cum se prelucrează reprezentarea** (atribuiri, evaluări, fire de execuție, continuări, fluxuri etc.)
- **Favorizează un set de concepte si tehnici de programare**
- **Influențează felul în care sunt gândiți algoritmii de rezolvare a problemelor**
- **Limbaje – în general multiparadigmă (ex: Python – imperativ, funcțional, orientat pe obiecte)**

Agenda cursului

1. Regulamente UB si FMI
2. Utilitatea cursului de Programare Orientata pe Obiecte
3. Prezentarea disciplinei
    - 3.1 Obiectivele discipinei
    - 3.2 Programa cursului
    - 3.3 Bibliografie
    - 3.4 Regulament de notare si evaluare
4. Primul curs

3. Prezentarea disciplinei

3.1 Obiectivele disciplinei

- Curs de programare OO

- Oferă o **baza** de pornire pentru alte cursuri

- **Obiectivul general al disciplinei:**

Formarea unei imagini generale, preliminare, despre programarea orientată pe obiecte (POO).

- **Obiective specifice:**
  - 1. Înțelegerea fundamentelor paradigmei programarii orientate pe obiecte;
  - 2. Înțelegerea conceptelor de clasă, interfață, moștenire, polimorfism;
  - 3. Familiarizarea cu şabloanele de proiectare;
  - 4. Dezvoltarea de aplicații de complexitate medie respectând principiile de dezvoltare ale POO;
  - 5. Deprinderea cu noile facilităţi oferite de limbajul C++.

3. Prezentarea disciplinei

3.2 Programa cursului

1. Prezentarea disciplinei.
1.1 Principiile programării orientate pe obiecte.
1.2. Caracteristici.
1.3. Programa cursului, obiective, desfăşurare, examinare, bibliografie.

2. Recapitulare limbaj C (procedural) și introducerea în programarea orientată pe obiecte.
2.1 Funcții, transferul parametrilor, pointeri.
2.2 Deosebiri între C și C++.
2.3 Supradefinirea funcțiilor, Operații de intrare/ieșire, Tipul referință, Funcții în structuri.

3. Prezentarea disciplinei

3.2 Programa cursului

3. Proiectarea ascendenta a claselor. Incapsularea datelor in C++.
3.1 Conceptele de clasa și obiect. Structura unei clase.
3.2 Constructorii și destructorul unei clase.
3.3 Metode de acces la membrii unei clase, pointerul this. Modificatori de acces în C++.
3.4 Declararea și implementarea metodelor în clasă și în afara clasei.

4. Supraîncărcarea funcțiilor și operatorilor în C++.
4.1 Clase și funcții friend.
4.2 Supraîncărcarea funcțiilor.
4.3 Supraîncărcarea operatorilor cu funcții friend.
4.4 Supraîncărcarea operatorilor cu funcții membru.
4.5 Observații.

3. Prezentarea disciplinei

3.2 Programa cursului

5. Conversia datelor în C++.
5.1 Conversii între diferite tipuri de obiecte (operatorul cast, operatorul= și constructor de copiere).
5.2 Membrii constanți și statici ai unei clase in C++.
5.3 Modificatorul const, obiecte constante, pointeri constanți la obiecte și pointeri la obiecte constante.

6. Tratarea excepțiilor in C++.

7. Proiectarea descendenta a claselor. Mostenirea in C++.
7.1 Controlul accesului la clasa de bază.
7.2 Constructori, destructori şi moştenire.
7.3 Redefinirea membrilor unei clase de bază într-o clasa derivată.
7.4. Declaraţii de acces.

3. Prezentarea disciplinei

3.2 Programa cursului

8. Funcţii virtuale în C++.
8.1 Parametrizarea metodelor (polimorfism la executie).
8.2 Funcţii virtuale în C++. Clase abstracte.
8.3 Destructori virtuali.

9. Mostenirea multiplă şi virtuală în C++
9.1 Moştenirea din clase de bază multiple.
9.2 Exemple, observaţii.

10. Controlul tipului în timpul rulării programului în C++.
10.1 Mecanisme de tip RTTI (Run Time Type Identification).
10.2 Moştenire multiplă şi identificatori de tip (dynamic_cast, typeid).

3. Prezentarea disciplinei

3.2 Programa cursului

11. Parametrizarea datelor. Şabloane în C++. Clase generice
11.1 Funcții şi clase Template: Definiţii, Exemple, Implementare.
11.2 Clase Template derivate.
11.3 Specializare.

12. Biblioteca Standard Template Library - STL
12.1 Containere, iteratori şi algoritmi.
12.2 Clasele string, set, map / multimap, list, vector, etc.

3. Prezentarea disciplinei

3.2 Programa cursului

13. Şabloane de proiectare (Design Pattern)
13.1 Definiţie şi clasificare.
13.2 Exemple de şabloane de proiectare (Singleton, Abstract Object Factory).

14. Recapitulare, concluzii, tratarea subiectelor de examen.

3. Prezentarea disciplinei

3.3 Bibliografie

1. Bruce Eckel. Thinking in C++ (2nd edition). Volume 1: Introduction to Standard C++. Prentice Hall, 2000.

2. Bruce Eckel, Chuck Allison. Thinking in C++ (2nd edition). Volume 2: Practical Programming. Prentice Hall, 2003.

3. Bjarne Stroustrup: The C++ Programming Language, Adisson-Wesley, 3nd edition, 1997.

4. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

3. Prezentarea disciplinei

3.4 Regulament de notare şi evaluare

Curs si laborator: fiecare cu 2 ore pe săptămână.

Seminar: 1 ora pe săptămână.

Disciplina: semestrul II, durata de desfășurare de 14 săptămâni.

Materia este de nivel elementar mediu şi se bazează pe cunoștințele de C++ anterior dobândite.

Limbajul de programare folosit la curs şi la laborator este **C++**.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Programa disciplinei este împărțită în 14 cursuri.

Evaluarea studenților se face cumulativ prin:

- 3 lucrări practice (proiecte)
- Test practic
- Test scris

Toate cele 3 probe de evaluare sunt obligatorii.

Condiții de promovare - minim **nota 5 la fiecare** parte de evaluare enunțată - mai sus se păstrează la oricare din eventualele examene restante ulteriore aferente acestui curs.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Regulamentul de laborator este orientativ. Fiecare tutore de laborator are dreptul sa-l adapteze cerințelor grupelor sale!

Cele 3 lucrări practice se realizează si se notează in cadrul laboratorului, după următorul program:

Săptămâna 1: Test de evaluare a nivelului de intrare.
Săptămâna 2: Atribuirea temelor pentru LP1.
Săptămâna 3: Consultații pentru LP1.
Săptămâna 4: Predare LP1. **Termen predare LP1: TBA**.
Săptămâna 5: Evaluarea LP1.
Săptămâna 6: Atribuirea temelor pentru LP2.
Săptămâna 7: Consultații pentru LP2.
Săptămâna 8: Predarea LP2. **Termen predare LP2: TBA**.
Săptămâna 9: Evaluarea LP2.
Săptămâna 10: Atribuirea temelor pentru LP3.
Săptămâna 11: Consultații pentru LP3.
Săptămâna 12: Predarea LP3. **Termen predare LP3: TBA**.
Săptămâna 13: Evaluarea LP3.
Săptămâna 13/14: Test practic de laborator.

Prezenta la laborator in săptămânile 1, 2, 5, 6, 9, 10, 13, 14 pentru atribuirea si evaluarea lucrărilor practice si pentru susținerea testului practic este obligatorie.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Regulamentul de laborator este orientativ. Fiecare tutore de laborator are dreptul sa-l adapteze cerintelor grupelor sale!

Consultațiile de laborator se desfășoară pe baza întrebărilor studenților.

Prezenta la laborator in săptămânile 3, 4, 7, 8, 11, 12 pentru consultații este recomandată, dar facultativă.

Lucrările practice se realizează individual.

Notarea fiecărei lucrări practice se va face cu note de la 1 la 10.

Atribuirea temelor pentru lucrările practice se face prin prezentarea la laborator in săptămâna precizată mai sus sau in oricare din următoarele 2 săptămâni. **Indiferent de data la care un student se prezintă pentru a primi tema pentru una dintre lucrările practice, termenul de predare a acesteia rămâne cel precizat in regulament.** In consecință, tema pentru o lucrare practică nu mai poate fi preluată după expirarea termenului ei de predare.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Regulamentul de laborator este orientativ. Fiecare tutore de laborator are dreptul sa-l adapteze cerintelor grupelor sale!

Predarea lucrarilor practice se face la adresa indicata de tutorele de laborator, inainte de termenele limita de predare, indicate mai sus pentru fiecare LP.

Dupa expirarea termenelor respective, lucrarea practica se mai poate trimite prin email pentru o perioada de gratie de 2 zile (48 de ore).

Pentru fiecare zi partiala de intarziere se vor scadea 2 puncte din nota atribuita pe lucrare.

Dupa expirarea termenului de gratie, lucrarea nu va mai fi acceptata si va fi notata cu 1.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Nota laborator = medie aritmetica a celor 3 note obtinute pe proiecte.

Pentru evidentierea unor lucrari practice, tutorele de laborator poate acorda un bonus de **pana la 2 puncte** la nota pe proiecte astfel calculata.

Studentii care **nu obtin cel putin nota 5 pentru activitatea pe proiecte nu pot intra in examen** si vor trebui sa refaca aceasta activitate, inainte de prezentarea la restanta.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Testul practic (Colocviu) - in saptamana 14

- Consta dintr-un program care trebuie realizat individual intr-un timp limitat (90 de minute – in varianta fata in fata si 2h in varianta online) si va avea un nivel mediu.

- Notare: de la 1 la 10 (pot exista pana la 3 puncte bonus).

Testul practic este obligatoriu.

Studentii care **nu obtin cel putin nota 5 la testul practic de laborator nu pot intra in examen** si vor trebui sa il dea din nou, inainte de prezentarea la restanta.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Testul scris:

Consta dintr-un set de 18 intrebari

- 6 intrebari de teorie
- 12 intrebari practice.

Notarea testului scris se va face cu o nota de la 1 la 10 (1 punct din oficiu si cate 0,5 puncte pentru fiecare raspuns corect la cele 18 intrebari).

Studentii nu pot lua examenul decat daca obtin cel putin nota 5 la testul scris.

3. Prezentarea disciplinei

3.4 Regulament de notare si evaluare

Examenul se considera luat daca studentul respectiv a obtinut cel putin nota 5 la fiecare dintre cele 3 evaluari (activitatea practica din timpul semestrului, testul practic de laborator si testul scris).

In aceasta situatie, nota finala a fiecarui student se calculeaza ca medie ponderata intre notele obtinute la cele 3 evaluari, ponderile cu care cele 3 note intra in medie fiind:

- 25% - nota pe lucrarile practice (proiecte)
- 25% - nota la testul practic
- 50% - nota la testul scris

Seminar - maxim 0.5p care se adauga la nota de la testul scris, **daca si numai daca**, nota de la testul scris >=5.

3. Prezentarea disciplinei

Modificari

- Laborator: notare mai “clara”
- Seminar: 0.5 bonus la nota de la examenul scris pentru max. 20% din studenti
- Prezenta la curs: 0.5 bonus la nota de la examenul scris pentru primii 20% dintre studenti KAHOOT
- bonusuri dupa ce se promoveaza examenul scris

3. Prezentarea disciplinei

Kahoot

- Se va defini un nume unic de forma popescu131 (unde popescu este numele de familie si 131 este grupa)
- Daca sunt mai multi studenti cu acelasi nume in grupa respectiva (adaugati si initiala / initialele prenumelui)
- 131: popescup131 si popescupr131

Agenda cursului

1. Regulamente UB si FMI
2. Utilitatea cursului de Programare Orientata pe Obiecte
3. Prezentarea disciplinei
4. Primul curs
   - 4.1 Scurta recapitulare C++
   - 4.2 Principiile programarii orientate pe obiecte

4. Curs 1

4.1 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

4. Curs 1

4.1 Completări aduse de limbajul C++ faţă de limbajul 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.

4. Curs 1

4.1 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** ...)

4. Curs 1

4.1 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 şi false (1 si 0);
- C99 nu îl definește ca bool ci ca _Bool (fără true/false)
- <stdbool.h> pentru compatibilitate

4. Curs 1

4.1 Scurta recapitulare C++

Supraîncărcarea funcţiilor (un caz de *Polimorfism la compilare*)

- este folosirea aceluiasi nume pentru functii diferite
- functii diferite, dar cu inteles apropiat
- compilatorul foloseste numarul si tipul parametrilor pentru a diferentia apelurile

4. Curs 1

4.1 Scurta recapitulare C++

```
int abs(int i)
{
        cout << "Using integer abs()\n";
        return i<0 ? -i : i;
}

double abs(double d)
{
        cout << "Using double abs()\n";
        return d<0.0 ? -d : d;
}

long abs(long l)
{
        cout << "Using long abs()\n";
        return l<0 ? -l : l;
}

int main()
{
        cout << abs(-10) << "\n";
        cout << abs(-11.0) << "\n";
        cout << abs(-9L) << "\n";
        return 0;
}
```

```
Using integer abs()
10
Using double abs()
11
Using long abs()
9
```

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

int myfunc(int i); // these differ in types of parameters
double myfunc(double i);

int main() {
        cout << myfunc(10) << " "; // calls myfunc(int i)
        cout << myfunc(5.4); // calls myfunc(double i)
        return 0;
}

double myfunc(double i) { return i; }

int myfunc(int i) { return i; }
```

tipuri diferite pentru parametrul i

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

int myfunc(int i); // these differ in number of parameters
int myfunc(int i, int j);

int main()
{
        cout << myfunc(10) << " "; // calls myfunc(int i)
        cout << myfunc(4, 5); // calls myfunc(int i, int j)
        return 0;
}

int myfunc(int i) {return i;}

int myfunc(int i, int j) {return i*j;}
```

numar diferit de parametri

- daca diferenta este doar in tipul de date intors: eroare la compilare

```
int myfunc(int i); // Error: differing return types are
float myfunc(int i); // insufficient when overloading.
```

- sau tipuri care _par_ sa fie diferite

```
void f(int *p);
void f(int p[]); // error, *p is same as p[]

void f(int x);
void f(int& x);
```

Ambiguitati pentru polimorfism de functii

- erori la compilare
- majoritatea datorita conversiilor implicite

```
int myfunc(double d);
// ...
cout << myfunc('c'); // not an error, conversion applied
```

```
#include <iostream>
using namespace std;
float myfunc(float i);
double myfunc(double i);

int main(){
cout << myfunc(10.1) << " "; // unambiguous, calls myfunc(double)
cout << myfunc(10); // ambiguous
 return 0;
}

float myfunc(float i){   return i;}
double myfunc(double i){   return -i;}
```

- problema nu e de definire a functiilor myfunc,
- problema apare la apelul functiilor

```
#include <iostream>
using namespace std;
char myfunc(unsigned char ch);
char myfunc(char ch);

int main(){
cout << myfunc('c'); // this calls myfunc(char)
cout << myfunc(88) << " "; // ambiguous
return 0;
}
char myfunc(unsigned char ch)
{
    return ch-1;
}
char myfunc(char ch){return ch+1;}
```

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

int myfunc(int i);
int myfunc(int i, int j=1);

 int main()
{
 cout << myfunc(4, 5) << " "; // unambiguous
 cout << myfunc(10); // ambiguous
 return 0;
}

int myfunc(int i)
{
  return i;
}

int myfunc(int i, int j)
{
  return i*j;
}
```

- **ambiguitate intre char si unsigned char**
- **ambiguitate pentru functii cu param. impliciti**

```
// This program contains an error. #include
<iostream>
using namespace std;
void f(int x);
void f(int &x); // error

int main() {
int a=10;
f(a); // error, which f()?
return 0;
}

void f(int x) { cout << "In f(int)\n"; }
void f(int &x) { cout << "In f(int &)\n"; }
```

- doua tipuri de apel: prin valoare si prin referinta, ambiguitate!
- mereu eroare de ambiguitate

4. Curs 1

4.1 Scurta recapitulare C++

Funcţii cu valori implicite

Într-o funcţie se pot declara valori implicite pentru unul sau mai mulţi 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 sfârşitul listei.

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

4. Curs 1

4.1 Scurta recapitulare C++

Funcţii 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; }
```

4. Curs 1

4.1 Scurta recapitulare C++

Alocare dinamica

```
int *pi;
 pi=new int;

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

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

  pi=new int[2]; // aloca un vector de 2 elemente  de tip intreg
  delete [ ] pi; //elibereaza intreg vectorul

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

4. Curs 1

4.1 Scurta recapitulare C++

Tipul referinta

O referinţă 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 referinţă trebuie iniţializată in momentul declararii, devenind un alias (un alt nume) al obiectului cu care a fost iniţializată.

4. Curs 1

4.1 Scurta recapitulare C++

Tipul referinta

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

int main()
{
  int a = 20;
  int& ref = a;
  cout<<a<<" "<<ref<<endl; // 20 20
  ref++;
  cout<<a<<" "<<ref<<endl; // 21 21
  return 0;
}
```

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

4. Curs 1

4.1 Scurta recapitulare C++

Funcţii 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; }
```

4. Curs 1

4.1 Scurta recapitulare C++

Alocare dinamica

```
int *pi;
 pi=new int;

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

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

  pi=new int[2]; // aloca un vector de 2 elemente  de tip intreg
  delete [ ] pi; //elibereaza intreg vectorul

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

4. Curs 1

4.1 Scurta recapitulare C++

Tipul referinta

O referinţă 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 referinţă trebuie iniţializată in momentul declararii, devenind un alias (un alt nume) al obiectului cu care a fost iniţializată.

4. Curs 1

4.1 Scurta recapitulare C++

Tipul referinta

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

int main()
{
  int a = 20;
  int& ref = a;
  cout<<a<<" "<<ref<<endl; // 20 20
  ref++;
  cout<<a<<" "<<ref<<endl; // 21 21
  return 0;
}
```

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

4. Curs 1

4.1 Scurta recapitulare C++

Tipul referinta

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

int main()
{
  int a = 20;
  int& ref = a;
  cout<<a<<" "<<ref<<endl; // 20 20
  int b = 50;
  ref = b;
  ref--;
  cout<<a<<" "<<ref<<endl; // 49 49
  return 0;
}
```

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

4. Curs 1

4.1 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 poate face referinta catre un camp de biti.

4. Curs 1

4.1 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++

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

4. Curs 1

4.1 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 ⇒ modificarile parametrului nu au efect asupra argumentului;
- apel prin referinta - in parametru este copiata adresa unui argument ⇒ modificarile parametrului au efect asupra argumentului.
- functiile, cu exceptia celor de tip void, pot fi folosite ca operand in orice expresie valida.

4. Curs 1

4.1 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;
}
```

4. Curs 1

4.1 Scurta recapitulare C++

Functii in structuri

C

```
#include <stdio.h>
#include <stdlib.h>
struct test
{
  int x;
  void afis()
  {
    printf("x= %d",x);
  }
}A;

int main()
{
    scanf("%d",&A.x);
    A.afis(); /* error ‘struct test’ has no
member called afis() */
    return 0;
}
```

C++

```
#include <iostream>
using namespace std;
struct test
{
  int x;
  void afis()
  {
    cout<<"x= "<<x;
  }
}A;

int main()
{
    cin>>A.x;
    A.afis();
    return 0;
}
```

4. Curs 1

4.1 Scurta recapitulare C++

Functii in structuri

```
#include <stdio.h>
#include <stdlib.h>

struct test {
  int x;
  void (*afis)(struct test *this);
};

void afis_implicit(struct test *this) {
  printf("x= %d",this->x);
}

int main() {
  struct test A = {3, afis_implicit};
  A.afis(&A);
  return 0;
}
```

Q: Exista un mecanism prin care putem avea totusi functii in structuri in C?
A: Da, utilizand pointerii la functii

Q: Codul alaturat este valid si in C++?
A: Nu, pentru ca am folosit “this” ca identificator (mai tarziu despre “this”).

Q: Daca putem folosi, totusi, functii in structuri in C, de ce folosim clase?
A: Pentru ca e dificil de emulat ascunderea informatiei, principiu de baza in POO.

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

- ce este programarea
- definirea programatorului:
  - rezolva problema
- definirea informaticianului:
  - rezolva problema **bine**

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Rezolvarea “mai bine” a unei probleme

- “bine” depinde de caz
  - drivere: cat mai repede (asamblare)
  - jocuri de celulare: memorie mica
  - rachete, medicale: erori duc la pierderi de vieti
- programarea OO: cod mai corect
  - Microsoft: nu conteaza erorile minore, conteaza data lansarii

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Principalele concepte utilizate in POO sunt:

Obiecte

Clase

Mostenire

Ascunderea informatiei

Polimorfism

***Sabloane* – nu sunt utilizate strict POO (mai general, se refera la Programarea generica)**

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

O **clasa** defineste atribute si metode.

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

- mentioneaza proprietatile generale ale obiectelor din clasa respectiva
- clasele nu se pot “rula”
- folositoare la encapsulare (ascunderea informatiei)
- reutilizare de cod: mostenire

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Un **obiect** este o instanta a unei clase care are o anumita stare (reprezentata prin valoare) si are un comportament (reprezentat prin functii) la un anumit moment de timp.

- au stare si actiuni (metode/functii)
- au interfata (actiuni) si o parte ascunsa (starea)
- Sunt grupate in clase, obiecte cu aceleasi proprietati

Un **program orientat obiect** este o colectie de obiecte care interactioneaza unul cu celalalt prin mesaje (aplicand o metoda).

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Principalele concepte (caracteristici) ale POO sunt:

Incapsularea – contopirea datelor cu codul (metode de prelucrare si acces la date) în clase, ducând la o localizare mai bună a erorilor şi la modularizarea problemei de rezolvat;

Moştenirea - posibilitatea de a extinde o clasa prin adaugarea de noi functionalitati;

- multe obiecte au proprietati similare
- reutilizare de cod

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Principalele concepte (caracteristici) ale POO sunt:

Ascunderea informatiei

foarte importanta

public, protected, private

Avem acces?	**public**	**protected**	**private**
Aceeasi clasa	da	da	da
Clase derivate	da	da	nu
Alte clase	da	nu	nu

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Principalele concepte (caracteristici) ale POO sunt:

Polimorfismul (la executie – *discutii mai ample mai tarziu*) – într-o ierarhie de clase obtinuta prin mostenire, o metodă poate avea implementari diferite la nivele diferite in acea ierarhie;

4. Curs 1

4.2 Principiile programarii orientate pe obiecte

Alt concept important in POO:

Sabloane

- cod mai sigur/reutilizare de cod
- putem implementa lista inlantuita de
  - intregi
  - caractere
  - float
  - obiecte

Perspective

1. Se vor discuta directiile principale ale cursului, feedback-ul studentilor fiind hotarator in acest aspect

- intelegerea notiunilor

- intrebari si sugestii

2. Cursul 2:

- Introducere in OOP. Clase. Obiecte