Clase I

Curs 2

Curs anterior

- Modelarea problemelor
- Tipuri de date
- Îmbunătăţiri aduse de C++

Cuprins

- Clase
  - Specificatori de acces
  - Contructorii
  - Destructorii
- Separarea codului

Program

- Programare structurată
  - Structuri de date + Algoritmi = Program
- Programare orientată obiect
  - Date + Metode = Obiect

Clase

- Definiţie
  - O clasa este o implementare a unui tip de date (concret, abstract sau generic). Defineşte atribute şi funcţii care descriu structura de date şi operaţiile care se pot efectua cu acest tip de date

- Exemple
  - Universitate
  - Student
  - Profesor
  - Amfiteatru
  - Sală
  - etc

Clase

- Clasa este un grup de obiecte care au caracteristici comune
- Clasele în C++ pot fi definite folosind cuvintele cheie struct sau class

Sintaxă

```
class X {
    // variabile membru
    // functii membru
};

struct X {
    // variabile membru
    // functii membru
};
```

Exemplu

```
struct Universitate{
    char *nume, *adresa;
    void init (char *nume, char* adresa);
};
```

Accesul la date

- Structuri – public
- Clase – privat

Obiecte

- Este un element finit şi particular al unui model
- Obiectele sunt create prin declararea de variabile

Sintaxă

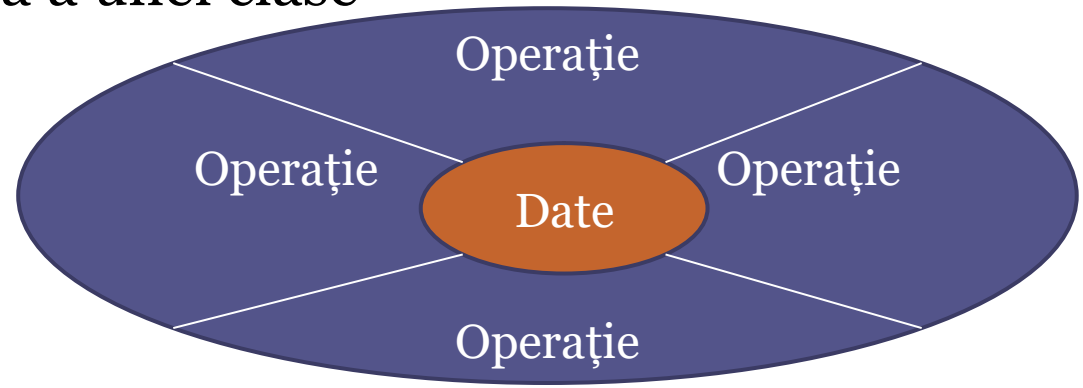
TDA numeVariabila;

Exemplu

Universitate ubb;
Universitate *uvt = new Universitate;

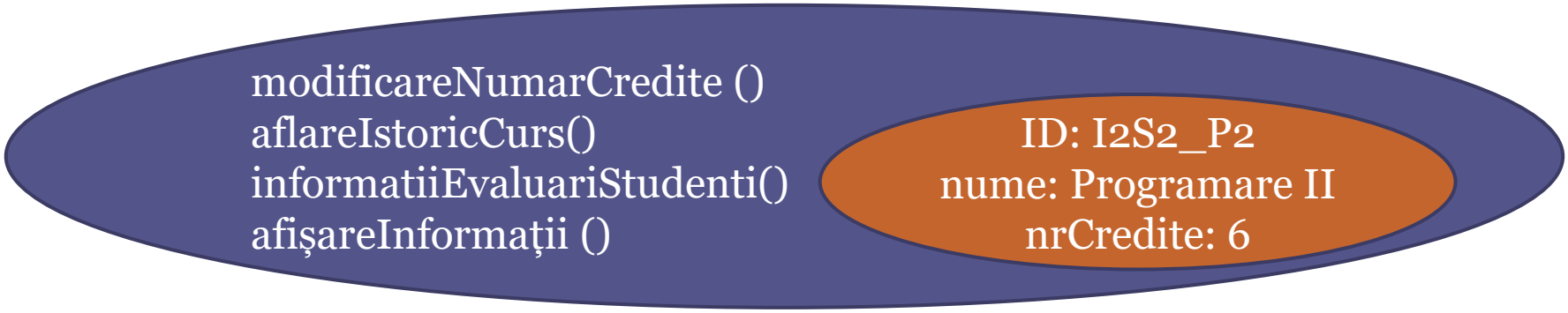
Obiecte

- Obiectele sunt entităţi create la execuţie (run-time), de bază ale unui sistem orientat obiect
- Fiecare obiect are asociate date şi funcţii care definesc operaţiile care au sens asupra obiectului
- Este o entitate care există în lumea reală
- Un obiect este o instanţă a unei clase

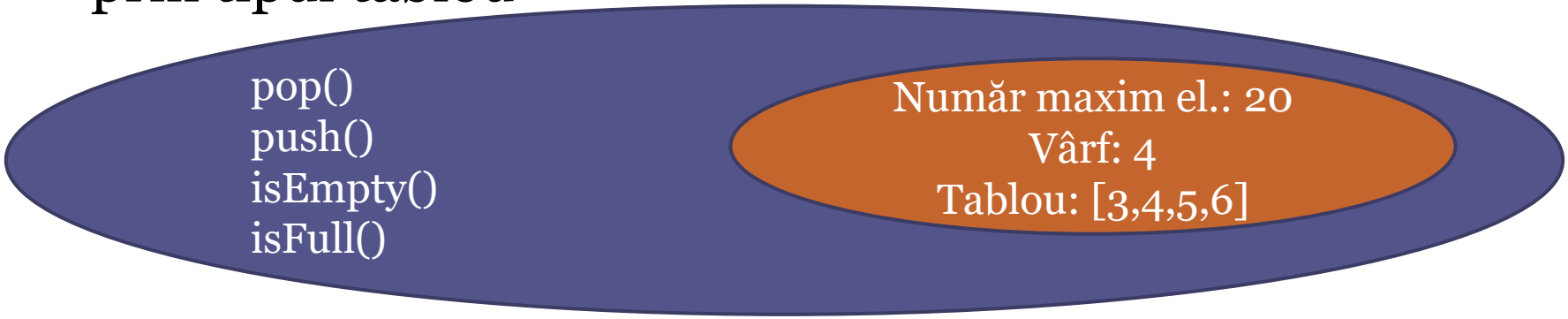


Obiecte

- Exemplu un obiect care descrie cursul Programare II



- Exemplu un obiect care descrie o stivă implementată prin tipul tablou



Exercițiu

- Florărie
  - Identificare obiectelor dintr-o florărie
  - Interacţiunile dintre ele
  - Proprietăţile comune
  - Ce clase se pot defini?

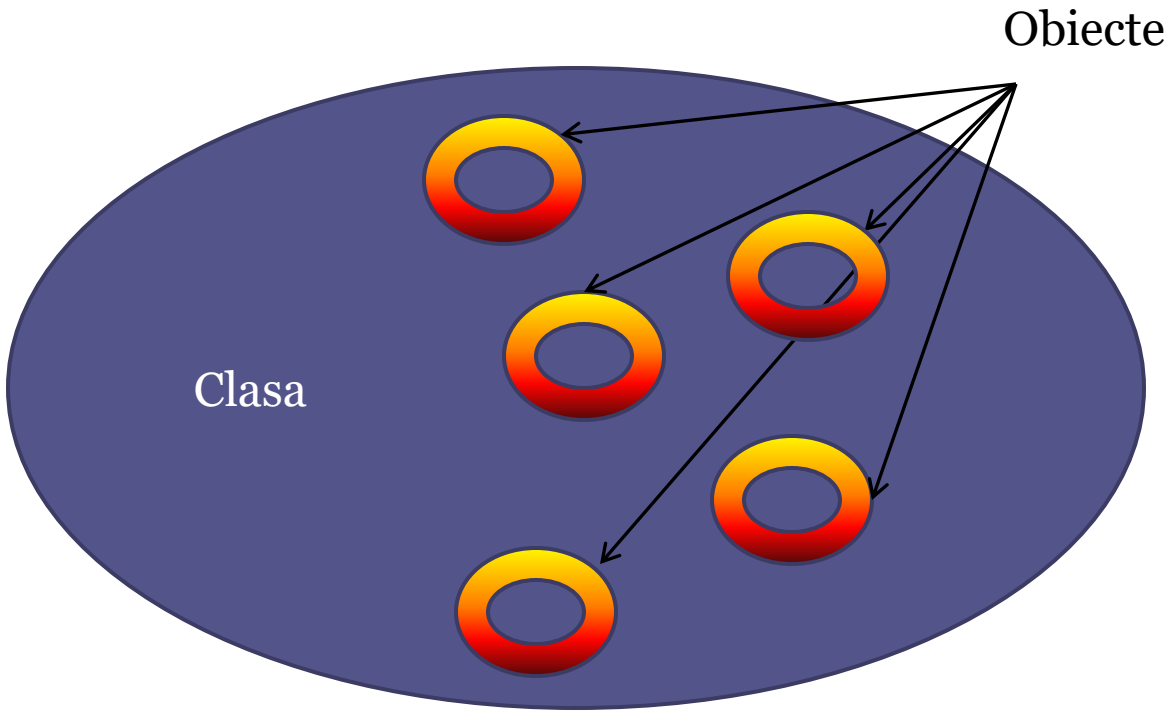
Exercițiu

FLORĂRIE

- Identificare obiectelor dintr-o bibliotecă
  - Produse, accesorii, plante ghiveci, plante aranjamente, client, angajat,
- Interacţiunile dintre ele
  - Un client comandă un aranjament
  - Un angajat comandă produse lipsă
  - Un angajat folosește anumite accesorii care trebuie extrase din gestiune
- Proprietăţile comune
  - Produsele care poate fi ghiveci, plante aranjamente
- Ce clase se pot defini?
  - Florărie, Client, Angajat, Produs, Accesoriu, Floare

Clase

- Caracterizează o colecție de obiecte similare



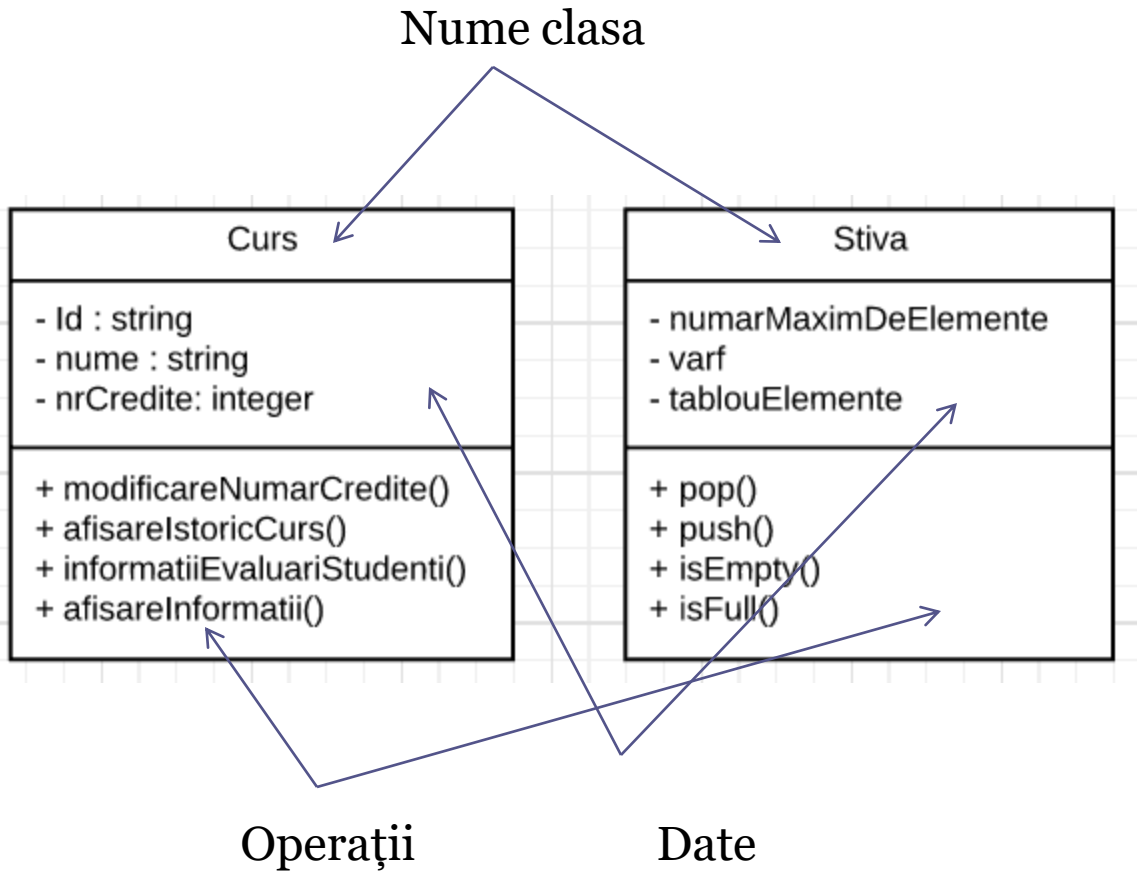
Clase. Definire

- Sintaxă

```
class numeClasă [: Listă clase de bază]{
    Date/câmpuri/variabile membre; //informaţii
    Funcţii/metode membre; //comportament
} [listăDeVariabile];
```

! Nu uitaţi să puneţi caracterul “;” la sfârşitul declaraţiei unei clase.

Clase. Reprezentare grafică



Clase. Exemplu

class Curs {

```
char * cursId;
char *nume;
int nrCredite;
```

Date membre
Câmpuri
Proprietăți

```
void modificareNumarCredite (int);
void afisareIstoricCurs();
void informatiiEvaluariStudenti()
void afisareInformatii();
```

Metode/Funcţii membre

};

class Stiva {

```
int numarMaximDeElemente;
int varf;
double * tablouElemente;
```

```
double pop();
void push(int);
bool isEmpty();
bool isFull();
```

};

Specificatori de acces

- Vizibilitatea datelor şi funcţiilor membre
  - public
    - Accesate de funcţiile membre ale clasei şi toate funcţiile nemembre ale programului
  - private
    - Accesate doar de funcţiile membre sau prietene clasei
  - protected
    - Asemănător cu privat, dar permite accesul claselor derivate la datele şi funcţiile membre
  - default
    - Toate datele definite într-o clasă sunt private dacă nu este specificat un specificator de acces

Specificatori de acces

```
class Curs {

 char * cursId;
 char *nume;
 int nrCredite;

 void modificareNumarCredite (int);
 void afisareIstoricCurs();
 void informatiiEvaluariStudenti()
 void afisareInformatii();

};
```

Vizibilitate privată/ implicită

Specificatori de acces

```
class Curs {
  char * cursId;
public:
  char *nume;
protected:
  int nrCredite;
public:
  void modificareNumarCredite (int);
  void afisareIstoricCurs();
  void informatiiEvaluariStudenti()
private:
  void afisareInformatii();
};
```

Vizibilitate - Default(privată)

Vizibilitate - publică

Vizibilitate - protejată

Vizibilitate - publică

Vizibilitate - privată

Specificatori de acces

- Observaţii
  - Domeniul de utilizare al unui specificator de acces ţine până la întâlnirea altui specificator de acces
  - Specificatorul de acces implicit este private
  - Funcţiile membre clasei ar trebui să fie declarate publice
    - Cu excepţia funcţiilor membre clasei care sunt accesate doar de alte funcţii membre ale aceleaşi clase

Specificatori de acces

- Observaţii
  - Datele membru ale unei clase ar trebuie să fie declarate ca fiind private (private) pentru a respecta principiul încapsulări
    - Este util să existe funcţii **set** şi **get** pentru a accesa datele membru într-un mod controlat
      - Exemplu funcţie **set** pentru variabila *numărCredite* a clasei *Curs*

```
void setNumarCredite(int nrCredite){
        numarCredite = nrCredite;
}
```

      - Exemplu funcţie **get** pentru variabila *numărCredite* a clasei *Curs*

```
int getNumarCredite() const {
        return numarCredite;
}
```

Specificatori de acces

- Observaţii
  - Datele membru ale unei clase ar trebuie să fie declarate ca fiind private (private) pentru a respecta principiul încapsulări
    - Este util să existe funcţii set şi get pentru a accesa datele membru într-un mod controlat
      - Exemplu funcţie **set** pentru variabila *numărCredite* a clasei *Curs*

```
void setNumarCredite(int nrCredite){
        numarCredite = nrCredite;
}
```

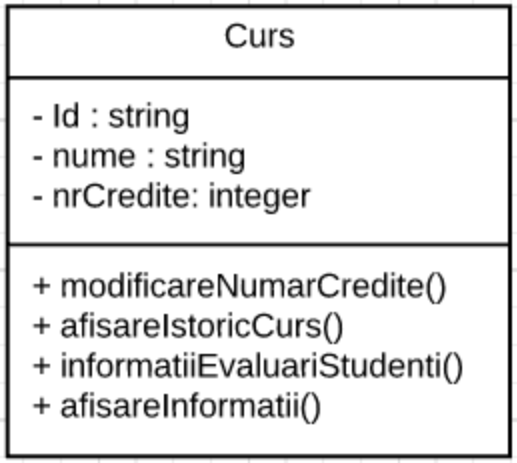
      - Exemplu funcţie **get** pentru variabila *numărCredite* a clasei *Curs*

```
int getNumarCredite() const {
        return numarCredite;
}
```

Care este rolul cuvântului cheie const după prototipul funcţiei?

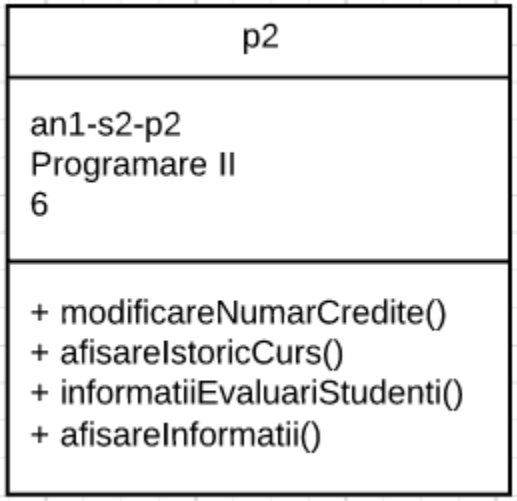
Obiecte

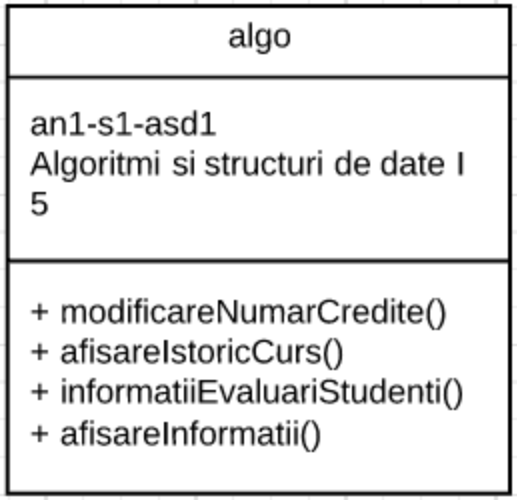
- Instanţierea obiectelor
  - Crearea unui obiect de un anumit tip
- Sintaxă
  - numeClasa numeObiect;
- Exemplu
  - Curs c;

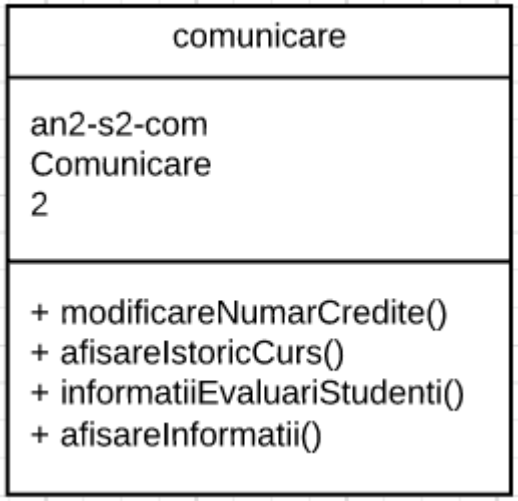


Obiecte

- Exemplu
  - Curs p2;
  - Curs algo;
  - Curs comunicare;







Pointeri la obiecte

- Declararea unui pointer de tip Curs
  - Curs *engleza;
- Declararea şi iniţializarea unui pointer de tip Curs folosind constructorul implicit
  - Varianta 1
    - Curs *engleza = new Curs()
  - Varianta 2
    - Curs *engleza;
    - engleza = new Curs();

Tablouri de obiecte

- Declararea si inițializarea unui tablou de obiecte
  - Declararea
    - Curs *sir;
  - Inițializarea
    - sir = new Curs [10];
- Declarea unui tablou de pointeri la obiecte
  - Declararea
    - Curs **sir;
  - Inițializarea
    - sir = new Curs* [10];
  - Inițializarea obiectelor
    - sir[1] = new Curs(); sir[2] = new Curs(); ....

Tablouri de obiecte

- Declararea si inițializarea unui tablou de obiecte
  - Declararea
    - Curs *sir;
  - Inițializarea
    - sir = new Curs [10];

- Declarea unui tablou de pointeri la obiecte
  - Declararea
    - Curs **sir;
  - Inițializarea
    - sir = new Curs* [10];
  - Initializarea obiectelor
    - sir[1] = new Curs(); sir[2] = new Curs(); ....

1. Care este diferența dintre cele două declarații de tablouri?

2. Care metodă este mai eficientă?

Acessarea membrilor în ineriorul clasei

```
class Curs {
  char * cursId;
public:
  char *nume;
protected:
  int nrCredite;
public:
  void modificareNumarCredite (int a){
        nrCredite = a;
  }

  void afisareIstoricCurs(){
     cout << “Curs: “ << nume << endl;
     informatiiEvaluariStudenti();
  }

  void informatiiEvaluariStudenti()
private:
  void afisareInformatii();
};
```

- Variabile membre
  - pot fi accesate de toate metodele clasei indiferent de modificatori de acces
- Metodele unei classe
  - pot apela metode definite în clasă indiferent de modificatorii de acess
  - au acces la variabile membre clasei
  - pot apela metode externe clasei
  - au acces la variabile globale

Acessarea membrilor în exteriorul clasei

```
class Curs {
  char * cursId;
public:
  char *nume;
protected:
  int nrCredite;

public:
  void modificareNumarCredite (int );
  void afisareIstoricCurs();
  void informatiiEvaluariStudenti();

private:
  void afisareInformatii();
};
```

- Variabile membre
  - pot fi accesate doar dacă sunt declarete ca fiind publice
- Metodele unei classe
  - Pot fi apelate în exterirul clasei doar dacă sunt declarate publice
- Pot fi accesate cu ajutorul operatorilor
  - .
  - ->

  prin intermediul obiectelor clasei

Acessarea membrilor în exteriorul clasei

```
class Curs {
  char * cursId;
public:
  char *nume;
protected:
  int nrCredite;

public:
  void modificareNumarCredite (int );
  void afisareIstoricCurs();
  void informatiiEvaluariStudenti();

private:
  void afisareInformatii();
};
```

- Membri au vizibilitate publică
- Pot fi accesati cu ajutorul operatorilor
  - .
  - ->

  prin intermediul obiectelor clasei

- Exemplu

```
int main() {
  Curs alg, *p2;
  Curs tab[10];

  alg.nrCredite = 8;
  p2->nrCredite = 8;
  tab[1].nrCredite = 8;

   alg. modificareNumarCredite (4);
   p2->alg. modificareNumarCredite (4);
   tab[1]. modificareNumarCredite (4);
 }
```

Clase

- Metode speciale
  - Constructori
    - Crearea obiectelor
  - Destructori
    - Distrugera obiectelor

Constructori

- Constructor
  - Funcţie utilizată pentru a iniţializa instanţele (obiectele) unei clase
- Caracteristici
  - Funcţia are acelaşi nume cu clasa
  - Nu are tip de return (nici măcar void)
  - Nu poate fi funcţie virtuală

Constructori

- O clasă poate avea unul sau mai mulţi constructori
  - Cu număr diferit de parametrii
  - Constructorul implicit (default) nu are parametri

- Procesul de creare al unui obiect
  - Alocarea memoriei
  - Găsirea constructorului corespunzător
  - Apelarea constructorului pentru a iniţializa starea obiectului, membri clasei au fost anterior construiţi/iniţializaţi

Constructori. Exemplu

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume);
  Curs(char *nume, char *ID_curs);
  Curs(char *nume, char *ID_curs, int numarCredite);
};

void foo(){
  Curs c1 = Curs("algoritmica"); //Corect
  Curs c2("Programare III", "I2S1_P3"); //Corect
  Curs c3("Logic", "E3S1_LOG", 10); //Corect
  Curs c4; //EROARE
  Curs *c5 = new Curs("Structuri de date", "I2S2_SD"); //Corect
}
```

Care ar fi o soluție?

Constructori. Exemplu

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume);
  Curs(char *nume, char *ID_curs);
  Curs(char *nume, char *ID_curs, int numarCredite);
};

void foo(){
  Curs c1 = Curs("algoritmica"); //Corect
  Curs c2("Programare III", "I2S1_P3"); //Corect
  Curs c3("Logic", "E3S1_LOG", 10); //Corect
  Curs c4; //EROARE
  Curs *c5 = new Curs("Structuri de date", "I2S2_SD"); //Corect
}
```

Adaugarea unui constructor fară parametrii. Curs(){}

Cum am putea simplifica codul astfel încât sa nu trebuiască sa scriem 4 constructori?

Constructori. Exemplu

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume);
  Curs(char *nume, char *ID_curs);
  Curs(char *nume, char *ID_curs, int numarCredite);
};

void foo(){
  Curs c1 = Curs("algoritmica"); //Corect
  Curs c2("Programare III", "I2S1_P3"); //Corect
  Curs c3("Logic", "E3S1_LOG", 10); //Corect
  Curs c4; //EROARE
  Curs *c5 = new Curs("Structuri de date", "I2S2_SD"); //Corect
}
```

Curs(char *nume = NULL, char *ID_curs = NULL, int numarCredite = 1);

Constructori. Exemplu

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume = NULL, char
    *ID_curs = NULL, int
    numarCredite = 1);
};
```

```
Curs::Curs(char *_nume, char
*idCurs, int
credite) {

  if (_nume != NULL) {
     nume = new char [ strlen(_nume)
+ 1];
    strcpy (nume, _nume);
  }

  if (idCurs != NULL) {
    cursId= new char [ strlen(idCurs) +
1];
    strcpy( cursId, idCurs);
  }

  nrCredite = credite;
}
```

Constructori. Exemplu

Folosind iniţializarea membrilor

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume = NULL, char
    *ID_curs = NULL, int
    numarCredite = 1);
};
```

```
Curs::Curs(char *_nume, char
*idCurs, int
credite): nrCredite(credite) {

  if (_nume != NULL) {
     nume = new char [ strlen(_nume)
+ 1];
    strcpy (nume, _nume);
  }

  if (idCurs != NULL) {
    cursId= new char [ strlen(idCurs) +
1];
    strcpy( cursId, idCurs);
  }


}
```

Constructori. Exemplu

- Folosind iniţializarea membrilor cum am putea scrie definiţia următorilor constructori?

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume )
  Curs(char *nume, char * idCurs);
};
```

Constructori. Exemplu

- Folosind iniţializarea membrilor cum am putea scrie definiţia următorilor constructori?

```
class Curs {
  char * cursId;
  char *nume;
  int nrCredite;

public:
  Curs(char *nume )
  Curs(char *nume, char * idCurs);
};
```

```
Curs::Curs(char * _nume) : cursId(NULL),
nrCredite(1) {
   if (_nume != NULL) {
      nume = new char[strlen(_nume)+1];
      strcpy(nume, _nume);
   }
}

Curs::Curs(char * _nume, char* idCurs) :
nrCredite(2) {
   if (_nume != NULL) {
      nume = new char[strlen(_nume)+1];
      strcpy(nume, _nume);
    }
    if (idCurs != NULL) {
       cursId  = new char[strlen(idCurs)+1];
      strcpy(cursId , idCurs);
    }
}
```

Constructori. Constructor implicit

- Constructorul implicit (default)
- Prototip: X()
- Dacă nu există nici un constructor definit într-o clasă, compilatorul generează un constructor implicit, care realizează inițializări implicite pentru datele membru ale clasei
- Dacă o clasă are membri const sau referinţă atunci constructorul implicit nu mai este generat automat, deoarece membri const şi referinţă trebuie iniţializaţi
- Dacă o clasă are definit un constructor atunci constructorul implicit nu se mai generează automat

Constructori. Constructor implicit

```
class Data {
public:
  // constructor implicit
  Data(int zi=0, int luna=0, int an=0);
};

class String {
 public:
  String(); // constructor implicit
};

class Student {
  Data ziNastere;
  String nume;
  // constructor implicit generat
automat care apelează constructorii
claselor Data şi String
};
```

```
class Data {

public:
  // NU se generează constructor
implicit
  Data(int day);

};


class Test {
  const int a;
  int& r;

  // NU se generează constructor
implicit

};
```

Constructori. Constructor de copiere

- Constructor cu un argument care este o referinţă la clasa proprie
- Sintaxă
  - X (const X&obj);
- Este apelat
  - La declararea obiectelor precum X obj = obiect_sursă
  - Când se transmit argumente funcţiilor foo(X)
  - Când se creează un obiect temporar în timpul evaluări unei expresii
- Dacă nu este definit pentru o clasă, compilatorul generează automat unul care copiază bit-cu-bit conţinutul obiectului sursă în obiectul destinaţie
- Pentru a evita copierea unui obiect, constructorul de copiere poate fi declarat privat (nu este nevoie de implementare)
- Pentru o copie „în adâncime” a unui obiect complex, constructorul de copiere este obligatoriu de implementat

Constructori. Constructor de copiere

```
class Curs {
 public:
   //constructor
   Curs(char *nume = NULL, char *cursId= NULL, int numarCredite = 0);
   //constructor de copiere
   Curs(const Curs &ref);
 };

void g(Curs d) { }

void foo() {
 Curs c("Programare II", "I1S2_P2", 10); // apel constructor definit de utilizator
 Curs c1; // apel constructor implicit
 Curs c2 = c; // apel constructor de copiere
 Curs c3(c); // apel explicit constructor de copiere
 c2 = c; // NU se apeleaza constructorul de copiere
 g(c); // constructorul de copiere este apelat
}
```

Constructori. Constructor de copiere

```
class Curs {
 public:
   Curs(char *nume = NULL, char *
cursId = NULL, int numarCredite = 2);

    Curs(const Curs &ref);

};
```

```
Curs::Curs (const Curs & ref) :
numarCredite(ref.numarCredite) {

  if (ref.nume != NULL) {
    nume = new    char[strlen(ref.nume) + 1];
    strcpy(nume, ref.nume);
  }

  if (ref.cursId != NULL) {
     cursId = new   char[strlen(ref.cursId )+1];
    strcpy(cursId, ref. cursId );
  }

}
```

Costructori. Contructor de mutare

```
vector<string> updateNames(vector<string> ans)
{
 while (hasNewName()) {
  ans.push_back( getNewName() );
 }
 return ans;
}
// ...
auto allNames = updateNames( getSavedNames() );
```

- Moving constructor
  - introdus în stanarul de C++11
  - Cauză
    - de multe ori se efectuau copii inutile a obiectelor
  - Scop
    - de a impiedeca cat se poate de mult astfel de copii
- Sintaxă
  - X(const X &&);
- Apel
  - implicit de catre compilator atunci când identifică o situație când o copie a unui obiect nu este necesară
  - explict prin funcția move() din STL

Destructor

- Definiţie
  - O funcţie membră a unei clase a cărui rol este de a dealoca instanţele din memorie
- Sintaxă
  - ~X()
- Caracteristici
  - Funcţia nu are tip de return
  - Funcţia nu are parametrii
  - Funcţia este prefixată cu ~
- Procesul de distrugere a unui obiect
  - Apelul funcţiei destructor
  - Apelarea destructorilor datelor membru
  - Eliberarea memoriei

Destructor

- O clasă poate avea doar un destructor
- Compilatorul generează un destructor implicit dacă nu este unul definit în clasă

```
class Curs {
  char *idcurs;   char *nume;   int numarCredite;
 public:
   Curs(char *nume=NULL, char *ID_curs=NULL,int
numarCredite=1);
   ~Curs();
};

Curs:: Curs(char *n, char *id, int nr){
   ....
  cout << ”S-a creat cursul ” << nume << endl;
}
Curs:: ~Curs() {
   cout << ”S-a sters cursul ” << nume << endl;
    if (cursId != NULL) delete [] cursId;
    if (nume != NULL) delete [] nume;
}
```

```
void foo() {
 Curs c(“Prog. II”, “I1S2_P2”, 10);

 Curs c1(”Algoritmica”);
 Curs *cc = new Curs(“Logica”);
 delete cc;
}
```

Care este rezultatul apelului funcției foo()?

Destructor. Observații

- Destructori se apelează în ordinea inversă a creări obiectelor

- Dacă o clasă conţine membri pointeri trebuie implementate:
  - Constructor
  - Constructor de copiere
  - Destructor
  - Supraîncărcarea operatorului =

Separarea codului

- Fişiere sursă care conţin implementarea codului sursă
  - extensie .cpp
  - Implementarea clasei
  - Programul principal
  - Programare de test
  - ...
- Fişierele header
  - Fişiere separate care conţin definiţia clasei
  - Permit compilatorului să recunoască clasele când sunt folosite în alt loc
  - Au extensia .h

Interfață vs. Implementare

- Interfețe
  - Descriu serviciile pe care clienţi unei clase le pot folosi şi modul în care se obţin aceste servicii
    - Definiţia clasei conţine o listă a prototipurile funcţiilor publice
- Implementarea funcţiilor membre
  - Într-un fişier sursă separat de definiţia clasei
    - Folosirea operatorului de definire a scopului :: pentru a lega definiţia de declaraţia funcţiei
  - Detaliile de implementare sunt ascunse
    - Codul clientului nu trebuie să ştie detaliile de implementare

Exemplu

- Definirea unei clase Color care are ca atribute cele 3 culori fundamentale red, geen, blue.
- Declararea clasei in fișier .h

```
#ifndef COLOR_H_INCLUDED
#define COLOR_H_INCLUDED
class Color{
 int red, green, blue;
public:
 Color(int=0, int=0, int=0);
 void display();
private:
 double validateColor(double);
};
#endif // COLOR_H_INCLUDED
```

Exemplu

- Definirea unei clase Color care are ca atribute cele 3 culori fundamentale red, geen, blue.
- Declararea clasei in fișier .h

```
#ifndef COLOR_H_INCLUDED
#define COLOR_H_INCLUDED
class Color{
 int red, green, blue;
public:
 Color(int=0, int=0, int=0);
 void display();
private:
 double validateColor(double);
};
#endif // COLOR_H_INCLUDED
```

Ce sunt acestea?
Care este rolul lor?

Exemplu

- Definirea unei clase Color care are ca atribute cele 3 culori fundamentale red, geen, blue.
- Declararea clasei in fișier .h

```
#ifndef COLOR_H_INCLUDED
#define COLOR_H_INCLUDED
class Color{
 int red, green, blue;
public:
 Color(int=0, int=0, int=0);
 void display();
private:
 double validateColor(double);
};
#endif // COLOR_H_INCLUDED
```

Directive de precompilare care asigură unicitatea declarării clasei Color într-un proiect

Exemplu

- Definirea unei clase Color care are ca atribute cele 3 culori fundamentale red, geen, blue.
- Implementarea/definirea metodelor clasei în fişiere cu extensia .cpp

```
#include <Color.h>

Color::Color(int r, int g, int b){
 red = validateColor(r);
 blue = validateColor(b);
 green = validateColor(g);
 cout << "Constructor Color "; display(); cout << endl;
}
void Color::display(){
 cout << "[" << red << "," << green << "," << blue << "]";
}
double Color::validateColor(double c){
 return (0<=c && c<256)? c : 0;
}
```

Exemplu

- Definirea unei clase Color care are ca atribute cele 3 culori fundamentale red, geen, blue.
- Crearea de unui obiect de tip Color și afisarea de informații despre obiect

```
// main.cpp

#include<Color.h>
int main()
{
   Color c(12,45,100);
   c.display()
}
```

Exemplu

- Definirea unei clase Color care are ca atribute cele 3 culori fundamentale red, geen, blue.

- Exercițiu
  - Creați un tablou de culori. Afișați culorile stocate în tablou.

  - Afișați culorile care au cea mai mare concentrație de roșu.

Test curs

- Propuneți o abstractizare (clasă) pentru noțiunea de plantă
  - Adaugați prototipuri în clasă pentru operațiile pe care le considerați necesare clasei

Curs următor

- Continuare Clase
  - Funcţii şi membri statici
  - Funcţii friend
  - Autoreferinţe
  - Relaţii dintre clase