Clase și obiecte. Inițializarea și distrugerea obiectelor

Mihai Gabroveanu

Sintaxa definirii unei clase

```
class IdNumeClasa {
            [<IdProtecție>:] <ListaMembri>
            [<IdProtecție>:] <ListaMembri>
            …
            [<IdProtecție>:] <ListaMembri>
   };
```

unde:

<*IdProtectie*>:=private | protected | public

<*ListaMembri*>:=<ListaDateMembre><ListaFunctiiMembre>

Protecția datelor și funcțiilor membre

- ***private*** – datele şi funcţiile aflate sub influenţa acestui modificator NU pot fi accesate din afara clasei. *În mod implicit datele și funcțiile unei clase sunt private*.
- ***protected*** - datele şi funcţiile aflate sub influenţa acestui modificator nu pot fi accesate din afara clasei, cu excepția claselor *derivate*
- ***public*** – datele și funcțiile aflate sub influența modificatorului public, pot fi accesate în afara clasei

Obiecte

- Obiectul reprezintă ***o instanță a unei clase***.
- Sintaxa declarării unui obiect

  IdNumeClasă idOb1, …, idObN;
- Accesarea datelor și funcțiilor membre

  idOb.idDataMembru

  idPointerOb ->idDataMembru

  idOb.idMetodaMembru(lista de parametri);

  idPointerOb ->idMetodaMembru(lista de parametri);

Exemplu:Clasa Complex

```
class Complex {
      private:
          float re;
          float im;
      public:
          void citire();
          void afisare();
          float modul();
};
```


Exemplu:Clasa Complex

```
void Complex::citire(){
     printf("Dati partea reala:");scanf("%f",&re);
     printf("Dati partea imaginara:");scanf("%f",&im);
}
void  Complex::afisare(){
      printf("%g+%g*i", re, im);
}
float Complex::modul(){
      return sqrt(re*re + im*im);
}
```


Exemplu:Clasa Complex

```
int main(){
    Complex z;
    z.re = 2.5;  // Incorect deoarece re este private
    z.citire();
    Complex *p=&z;
    p -> afisare();
}
```


Inițializarea Obiectelor

- În cazul datelor de tipuri predefinite, este posibilă inițializarea acestora la momentul declarării.

  Exemplu:

  int x=10;

  float a[2]={3,8};

- În cazul obiectelor, inițializarea la declarare se face prin intermediul unor funcții speciale numite **constructori.**

Constructor – Definiție

- **Constructorul** este o funcție membră specială a unei clase ce se apelează în mod automat la crearea unui obiect
- Rol:
  - alocare
  - inițializare

Constructor – Sintaxă (I)

Declararea constructorului

```
class IdNumeClasa {
   ...
   IdNumeClasa (<listaParametri>);
   ...
};
IdNumeClasa::IdNumeClasa (<listaParametri>){
   //instructiuni
}
```

Definirea constructorului

Constructor – Sintaxă (II)

- Inițializarea obiectelor

IdNumeClasa idObiect(<listaParametri>);

sau

IdNumeClasa idObiect = valParam;

în cazul în care lista de parametri e formată dintr-un singur parametru

Exemplu:Clasa Complex

Constructor (declarare)

```
class Complex {
      private:
          float re;
          float im;
      public:
          Complex(float r, float i);
          void citire();
          void afisare();
          float modul();
};
```

Exemplu:Clasa Complex

```
Complex::Complex(float r, float i){
      re = r;
      im = i;
}

int main(){
      Complex z(7,3);
      z.afisare();
}
```

Constructor (definire)

Constructori - Caracteristici

- au același nume cu cel al clasei din care fac parte
- nu returnează nimic (nici măcar tipul void)
- o clasă poate avea mai mulți constructori
- nu pot primi ca parametri instanțe ale clasei ce se definește, ci doar pointeri sau referințe la instanțele clasei respective
- constructorii nu sunt apelaţi explicit (în general)
- constructorii nu se *moștenesc*
- constructori nu pot fi funcții virtuale

Tipuri de constructori (I)

- **Constructori impliciți**
  - *definit de utilizator* – constructor ce nu are niciun parametru
  - *generat de compilator* – dacă o clasă nu are niciun constructor definit atunci compilatorul generează unul automat, fără parametri al cărui corp nu conține nicio instrucțiune
  - constructor cu toți parametri impliciți

Exemplu:Constructor Implicit

```
class Complex {
      …
      public:
          Complex(){
            re = 0;
            im = 0;
            printf ("Apel constructor\n");
          }
      …
};
…
Complex z;
z.afisare();
…
```

Output

```
Apel constructor
0+0*i
```

Tipuri de constructori (II)

- **Constructori cu parametri**
  - *cu parametri ce nu iau valori implicite*
  - *cu parametri ce iau valori implicite*
- Funcții cu parametri impliciți

tip_r nume_funcție(tip$_1$ p$_1$, ..., tip$_n$ p$_n$, tip$_{n+1}$ p$_{n+1}$=vi$_{n+1}$,..., tip$_m$ p$_m$=vi$_m$);

p$_{n+1}$, …, p$_m$ = **parametri impliciți.**

La apelul funcției aceștia pot să lipsească, caz în care ei au valorile implicite specificate de declarare.

Exemplu:Constructor cu parametri impliciți

```
class Complex {
      …
      public:
          Complex(float r = 0, float i = 0){
            re = r;
            im = i;
          }
      …
};
…
Complex z1(2,3), z2(4), z3=5, z4;
z1.afisare();z2.afisare();
z3.afisare();z4.afisare();
…
```

Output

```
2+3*i
4+0*i
5+0*i
0+0*i
```


Exemplu: Constructor cu parametri

```
class Persoana {
  private:
    char *nume;
    int varsta;
  public:
    Persoana(char *n, int v){
      nume = new char[strlen(n)+1];
      strcpy(nume, n);
      varsta = v;
    }
    void afisare(){
      cout<<"Nume:"<<nume<<endl;
      cout<<"Varsta:"<<varsta<<endl;
    }
    void setNume(char *n){
      if(strlen(nume)<strlen(n)){
        delete nume;
        nume = new char[strlen(n)+1];
      }
      strcpy(nume, n);
    }
    void setVarsta(int v){
      varsta = v;
    }
};
```


Exemplu: Constructor cu parametri (cont)

```
int main(){
   Persoana p1("Mihai",21);
   Persoana p2=p1;
   p1.afisare();
   p2.afisare();
   p2.setNume("Misu");
   p2.setVarsta(24);
   p1.afisare();
   p2.afisare();
   getch();
}
```

Output

```
Nume:Mihai
Varsta:21
Nume:Mihai
Varsta:21
Nume:Misu
Varsta:21
Nume:Misu
Varsta:24
```

Corect ar fi Mihai

Tipuri de constructori (III)

- **Constructori de copiere -** iniţializarea obiectelor la *declarare* cu alte obiecte deja create
  - *definiți de utilizator*
  - *generați de compilator*

Constructori de copiere – Sintaxa declarării

```
class IdNumeClasa {
   . . .
    IdNumeClasa (IdNumeClasa &ob);
        sau
    IdNumeClasa (const IdNumeClasa &ob);
   . . .
};
```

Recomandat

Constructori de copiere - Utilizare

Constructorii de copiere se apelează în următoarele cazuri:

- Crearea de obiecte cu iniţializare, pornind de la un obiect care există (IdClasa ob2=ob1 sau IdClasa ob2(ob1)).
- Apelul unei funcţii care lucrează cu obiecte transferate prin valoare, când este nevoie de crearea unei copii a obiectului pe stivă (cazul f(ob);).
- Returnarea dintr-o funcţie a unui obiect prin valoare (return ob;)

Exemplu:Constructor de copiere (I)

```
class Complex {
      …
      public:
          Complex(const Complex &z){
            re = z.re;
            im = z.im;
            printf("Apel constructor de copiere\n") ;
          }
      …
};
void f(Complex z){
      ...
}
…
Complex z1(2,3);
z1.afisare();
Complex z2 = z1;
z2.afisare();
f(z1);
```

Output

```
2+3*i
Apel constructor de copiere
2+3*i
Apel constructor de copiere
```

Exemplu:Constructor de copiere (II)

```
class Persoana {
  . . .
 Persoana(const Persoana &p){
 nume =
  new char[strlen(p.nume)+1];
  strcpy(nume, p.nume);
  varsta = p.varsta;
  cout<<"Apel constructor de
copiere\n";
    }

};
```

```
int main(){
 Persoana p1("Mihai",21);
 Persoana p2=p1;
 p1.afisare();
 p2.afisare();
 p2.setNume("Misu");
 p2.setVarsta(24);
 p1.afisare();
 p2.afisare();
}
```

```
Apel constructor cu
parametri
Apel constructor de
copiere
Nume:Mihai
Varsta:21
Nume:Mihai
Varsta:21
Nume:Mihai
Varsta:21
Nume:Misu
Varsta:24
```


Destructori

- **Destructorul** este o funcție membră specială a unei clase ce apelează în mod automat distrugerea unui obiect
- **Rol:** eliberarea zonelor alocate dinamic, resurselor, etc.
- Tipuri
  - Definit de utilizator
  - Generat de compilator

Destructor – Sintaxă

Declararea destructorului

```
class IdNumeClasa {
   ...
    ~IdNumeClasa ();
   ...
};
```

Definirea destructorului

```
IdNumeClasa::~IdNumeClasa (){
   //instructiuni
}
```


Exemplu:Destructor (I)

```
class Complex {
      …
      public:
          ~Complex();
      …
};
Complex::~Complex(){
      printf("Distrugere obiect:");
      afisare();
}
void main(){
      Complex z1(2,3), z2(4,7);
      z1.afisare();
      z2.afisare();
}
```

Output

```
2+3*i
4+7*i
Distrugere obiect: 4+7*i
Distrugere obiect: 2+3*i
```

Exemplu:Destructor (II)

```
class Persoana {
  ...
 ~Persoana(){
   cout<<"Distrug
obiectul:"<<nume<<endl;
   delete nume;
  }

};
```

```
int main(){
 Persoana p1("Mihai",21);
 Persoana p2=p1;
 p1.afisare();
 p2.afisare();
 p2.setNume("Misu");
 p2.setVarsta(24);
 p1.afisare();
 p2.afisare();
}
```

```
Apel constructor cu parametri
Apel constructor de copiere
Nume:Mihai
Varsta:21
Nume:Mihai
Varsta:21
Nume:Mihai
Varsta:21
Nume:Misu
Varsta:24
Distrug obiectul:Misu
Distrug obiectul:Mihai
```


Destructori - Caracteristici

- Are același nume cu numele clasei şi este precedat de ~
- Nu are parametri
- Nu returneaza nimic (nici macar void)
- O clasa poate avea un singur destructor
- Pot fi funcţii virtuale

Temă

- Implementați clasa NumarRațional.
- Implementați clasa Carte.
- Implementați clasa Student în care numele se va aloca dinamic.