PROGRAMAREA ORIENTATĂ PE OBIECTE C++

Conf.univ.dr. Ana Cristina DĂSCĂLESCU

Universitatea Titu Maiorescu

Conținut tematic

- Metode constructor
- Metoda destructor
- Alocarea dinamică a unui obiect
- Tablou de obiecte

Inițializarea obiectelor

➢ **Exemplu**

```
class Complex
{
    double re, im;
    public:
void init (double re, double im)
{
    this->re=re;
    this->im=im;
}
void afisare()
{
    cout<<re<<" "<<im<<endl;
}
```

- Inițializare

```
int main()
{
   Complex ob;

   ob.init(2.3,  4.5);
   ...
}
```

Valori reziduale

Inițializare prin valori concrete

Inițializarea obiectelor

➢Neajunsuri

- Nu există nici o constrângere din partea limbajului ca un obiect să fie inițializat.
- În practică, un obiect poate fi inițializat prin diferite moduri (nu se cunosc valori pentru toate datele membre)
- Declararea și inițializarea se realizează prin două operații.

➢Soluție

- Încapsularea în clasa a unor metode care au rolul de a crea și inițializa obiecte!!!

Metode de tip constructor

➢ **Constructorul** este o metodă membră clasei care se apelează în mod automat la crearea unui obiect.

➢ **Rolul constructorului**

- Alocarea spațiului de memorie necesar pentru a reține toți membri obiectului
- Inițializarea stării obiectului (inițializarea datelor membre)

➢ **Proprietăți**

- Are aceeași denumire cu cea a clasei
- Nu returnează nicio valoare
- Nu are tip returnat
- Se pot defini mai multi constructori, în funcție de tipul inițializării.
- Nu poate fi invocat ca o metodă membră uzuală

Metode de tip constructor

➢ **Sintaxa**

```
class IdClasa
{
      ...
      public:
      IdClasa(<lista parametri>);
}

IdClasa::IdClasa(<lista parametri>)
{
      ...
}
```

Tipuri de Constructori

➢Se consideră clasa:

```
class IdClasa  {…}
```

1) Constructor fară argumente

- Inițializază datele membre cu valori implicite
- Sintaxa: `IdClasa(){//initializare date membre}`

2) Constructor cu argumente

- Inițializează datele membre cu valorile parametrilor
- Sintaxa:

```
IdClasa(tip1 data1, tip2 data2,...,tipn datan)
{
//initializare date membre
}
```

Tipuri de Constructori

3) Constructor de copiere

- Inițializază datele membre ale unui obiect cu datele membere ale unui obiect creat anterior.
- Sintaxa:

```
IdClasa(IdClasa &ob_sursa)
{//initializare date membre}
```

4) Constructor de conversie

- Are rolul de a realiza conversia de la un tip de dată abstract la tipul clasei
- Sintaxa:

```
IdClasa(TDA x)
{
//initializare date membre
}
```

Tipuri de Constructori

```
class Produs{
    char nume[100];
    float pret;
    public:
```

```
Produs(){
        strcpy(this -> nume,"####");
        this -> pret = 0.0;
      }
```

```
Produs(char *nume, float pret){
   strcpy(this -> nume, nume);
   this -> pret = pret;}
```

```
};
int main()
{ Produs p1;
  Produs p2("Cola", 45.6);
}
```

Apel constructor fara argumente #### 0.0

Apel constructor cu argumente Cola 45.6

Metoda de tip constructor

➢Observații

- Dacă o clasă nu conține metode constructor, atunci compilatorul atașează unul implicit, care inițializează datele membre cu valori reziduale.
- Metoda constructor execută și operații de alocare dinamică atunci când datele membre sunt referite prin pointeri
- O metodă constructor nu pote 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*

Metoda Destructor

- **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
- **Proprietăți**
  - Are aceeași denumire cu cea a clasei, precedată de simbolul ~
  - Nu returnează nicio valoare
  - Nu are tip returnat
  - Nu are argumente

Metoda Destructor

➢ **Sintaxă**

```
class IdClasa {
. . .
~IdClasa ();
. . .
};
IdClasa::~IdClasa (){
//instructiuni
}
```

Metoda Destructor

➢ **Exemplu**

```
class Complex {
…
public:
~Complex(){ cout<<"Distrugere obiect:")};
void afisare();}
int main(){
 Complex z1(2,3);
 z1.afisare();
}
```

Output
2 3
Distrugere obiect

Metoda de tip destructor

➢Observații

- O clasă conține un singur destructor

- Metoda destructor se apelează implicit la închiderea blocului în care obiectul a fost construit

- Metoda destructor execută și operații de eliberare a zonei de memorie alocată pentru datele membre ale unui obiect

Tabloul de obiecte

- În limbajul C++, un **tablou de obiecte** se poate declara cu condiția respectării următoarei restricții:
  - Clasa trebuie sa conțină un constructor fără argumente!!!

- **Sintaxă**

```
IdClasa tab[dim];
```

- **Exemplu**

```
Complex tab[10];
```

Output
10 obiecte de tip Complex
cu datele membre 0 0

Alocare dinamică a obiectelor

- Alocare/eliberarea de memorie în limbajul C

```
void * malloc ( size_t size );
void free ( void * ptr );
```

- Dezavantaje:

- necesită conversia explicită a pointerului returnat și specificarea numărului de octeți

```
int *p=(int *) malloc(sizeof(int));
```

- nu permit inițializarea/eliberarea obiectelor

`malloc()` nu apelează constructorii

`free()` nu apelează destructorul

Alocare dinamică a obiectelor

➢Alocare spațiu pentru un singur obiect

```
IdClasa *ob = new IdClasa(lista_de_parametri);
```

- alocă un spațiu de memorie în **heap** pentru a reține un obiect de tipul `IdClasa` pentru inițializarea acestuia apelându-se constructorul clasei

- dacă lista de parametri e vidă atunci se apelează constructorul implicit

Alocare dinamică a obiectelor

➢Alocare spațiu pentru un tablou de obiecte

```
IdClasa = new IdClasa[dim_max];
```

- alocă un spațiu de memorie în **heap** pentru a reține un tablou de **dim_max** obiecte de tipul IdClasa, pentru inițializarea acestora apelându-se de dim_max ori **constructorul implict**.

- Dacă nu există um constructor implicit se generează eroare in momentul compilării!!!!