§3. Constructori şi destructori.

La crearea oricărui obiect va fi apelată obligatoriu o funcție membră specială – **constructorul** clasei. Dacă acest lucru nu va fi făcut direct de către programator, el va fi făcut implicit de către compilator. La distrugerea oricărui obiect va fi apelată obligatoriu o altă funcție membră specială – **destructorul** clasei.

Definiţie 1. Constructorul este funcția membră specială a clasei, destinată pentru inițializarea fiecărui obiect al acesteia. Inițializarea se face imediat după crearea obiectului (imediat după declararea variabilei de tip clasă), adică după ce el a fost declarat şi lui i s-a alocat memorie.

Definiţie 2. Destructorul este funcția membră specială a clasei, destinată pentru efectuarea unor operații adăugătoare la distrugerea fiecărui obiect al clasei, când expiră termenul lui de „viață”. (De exemplu, eliberarea memoriei alocate obiectului, închiderea sau ştergerea unor fişiere etc. Deci, destructorul are funcția de a elimina toate „urmele” obiectului la distrugerea lui)

În procesul lucrului cu obiectele create ale unor clase, pot apărea situații când obiectele nu sunt pregătite pentru operațiile efectuate. Exemple care ar descrie astfel de situații ar putea fi: încercarea de a afişa informația despre proprietățile unui obiect, el ne-fiind inițializat; operații cu membrii ne-inițializați ai unui obiect. Dacă uneori astfel de operații generează erori nesemnificative, în unele cazuri urmările por fi mai grave.

Pentru a evita, astfel de situații, în cadrul claselor sunt utilizate nişte funcții membre cu destinație specială – constructorii. Constructorul este o funcție membră care are menirea de a pregăti obiectele create pentru operațiile ulterioare. De aceea la procesul de creare a oricărui obiect participă şi constructorul clasei.

Exemple de acțiuni puse în responsabilitatea constructorilor pot fi următoarele:

- inițializarea câmpurilor;
- alocarea de memorie şi inițializarea câmpurilor de tip pointer;
- deschiderea unor fişiere;
- etc.

Constructorul unei clase are următoarele particularităţi:

• numele constructorului trebuie să fie identic cu numele clasei (obligatoriu);

• pentru o clasă se pot defini mai mulți constructori, fiecare inițializând obiectele în moduri diferite. Constructorii trebuie să se deosebească între ei prin numărul de argumente sau tipul argumentelor;

• constructorul nu returnează nici o valoare şi deci, spre deosebire de celelalte funcții, îi lipseşte din antet sau prototip descrierea tipului valorii returnate;

• constructorul poate fi apelat explicit ca o funcție membră obişnuită. Dacă nu se apelează nici un constructor la crearea unui obiect, sistemul singur va apela unul din ei, şi anume – constructorul implicit.

Este prezentată următoarea schemă sintactică, care descrie utilizarea constructorilor:

```
class nume_clasa
{
...
nume_clasa([tip1 param1,...,tipN paramN]);
...
};
//-----------------------
nume_clasa :: nume_clasa([tip1 param1,...,
                          tipN paramN])
{
 //instructiuni
}
```

Fiind o funcție membră, constructorul poate fi implementat atât în interiorul clasei, cât şi în exteriorul ei.

Există următoarele tipuri de constructori:

1) constructorul implicit;
2) constructori cu parametri;
3) constructorul de copiere;

O clasă poate să nu aibă nici un constructor, poate avea unul sau mai mulți constructori. Chiar dacă programatorul nu defineşte în interiorul clasei nici un constructor, sistemul va crea automat un constructor – constructorul fără parametri, care se numeşte constructor implicit. Totodată, constructorul implicit poate fi definit şi de către programator.

Tradițional, constructorul cu parametri atribuie valorile argumentelor sale câmpurilor obiectului care se creează cu acest constructor.

Constructori cu parametri pot fi definiți mai mulți şi ei, având toți același nume, trebuie neapărat să se deosebească între ei prin numărul de argumente sau prin tipul argumentelor.

Constructorul de copiere permite construirea unui obiect nou în baza unui obiect existent.

Dacă programatorul nu defineşte un constructor de copiere, sistemul va genera un constructor de copiere automat. În urma inițializării cu ajutorul constructorului de copiere, se obțin două obiecte identice.

O altă categorie de erori posibile în procesul de prelucrare a obiectelor ar putea fi ne-eliberarea de memorie, legată de careva obiect, atunci când obiectul îşi termină existența sa, ne-eliberarea altor resurse, ce țin de obiect. În asemenea situații are loc o risipă de resurse, deoarece este pierdut complet controlul asupra lor, odată cu lichidarea obiectului. Astfel de situații pot fi ocolite, utilizând nişte funcții membre cu destinație specială, numite destructori.

Destructorul este o funcție membru care are menirea de a efectua o serie de operații în contextul distrugerii obiectului. De aceea la procesul de distrugere a oricărui obiect participă şi destructorul clasei.

Exemple de acțiuni puse în responsabilitatea destructorilor pot fi următoarele:

- eliberarea memoriei asociate cu obiectul;
- închiderea unor fişiere etc.

Destructorul unei clasei are următoarele particularități:

- numele destructorului este identic cu numele clasei, fiind precedat de simbolul „~”;
- similar cu constructorul, destructorul nu returnează nici o valoare;
- destructorul nu are parametri;
- destructorul nu poate fi apelat explicit după modelul unei funcții membre obişnuite. Apelarea lui are loc automat atunci, când obiectul ajunge la sfârşitul domeniului său de scop.

Este prezentată următoarea schemă sintactică, care descrie utilizarea destructorilor:

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

Clasa poate şi să nu aibă destructor. În acest caz, este creat un destructor implicit, care de fapt nu efectuează anumite operații în procesul de distrugere a obiectului. Clasa poate avea cel mult un destructor definit de programator.

Destructorul se execută în ordine inversă executării constructorului. Deci la sfârşitul lucrului unei funcții, obiectele create în ea se distrug în ordinea inversă decât cea în care ele au fost create.

Exemplu. Să se definească clasa „Angajat”, obiectele căreia ar conține informația despre angajații unei întreprinderi (de exemplu, numele, anul naşterii, funcția, salariu). Să se definească pentru această clasă trei tipuri de constructori şi destructorul. Să se definească funcțiile membre pentru a) Setarea numelui angajatului, b) Accesarea numelui angajatului, c) Modificarea salariului angajatului, d) Setarea salariului unui angajat ca media aritmetică dintre salariile altor doi angajați, e) Afişarea informației despre angajat la ecran.

Programul:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
class Angajat
{
 // membrii privati
 private:
  char* Nume;
  char* Functia;
  int An;
  double Salariu;
```

```
// membrii publici
 public:
  Angajat(); // constructor implicit
  Angajat(char* N,char* F,int A,double S);
  // constructor cu parametri
  Angajat(char* N,int A); // constructor cu
parametri
  Angajat(double S); // constructor cu
parametri
  Angajat(Angajat& X); // constructor de
copiere
  ~Angajat(); // destructorul
  void SetNume(char* N);
  char* GetNume();
  int ModificaSalariu(double S);
  void SalariuMediu(Angajat& X,Angajat& Y);
  void Afisare();
};
```

```
Angajat::Angajat()
{

Nume=(char*)malloc(sizeof("Necunoscut")+
1);
 Functia=(char*)malloc(sizeof("-")+1);
 strcpy(Nume,"Necunoscut");
 strcpy(Functia,"-");
 An=0;
 Salariu=0;
}
```

```
Angajat::Angajat(char* N,char* F,int
A,double S)
{
 Nume=(char*)malloc(sizeof(N)+1);
 Functia=(char*)malloc(sizeof(F)+1);
 strcpy(Nume,N);
 strcpy(Functia,F);
 An=A;
 Salariu=S;
}
```

```
Angajat::Angajat(char* N,int A)
{
Nume=(char*)malloc(sizeof(N)+1);
Functia=(char*)malloc(sizeof("-") +1);
 strcpy(Nume,N);
 strcpy(Functia,"-");
 An=A;
 Salariu=0;
}
```

```
Angajat::Angajat(double S)
{
Nume=(char*)malloc(sizeof("Necunoscut")+
1);
 Functia=(char*)malloc(sizeof("-")+1);
 strcpy(Nume,"Necunoscut");
 strcpy(Functia,"-");
 An=0;
 Salariu=S;
}
```

```
Angajat::Angajat(Angajat& X)
{
 Nume=(char*)malloc(sizeof(X.Nume)+1);

Functia=(char*)malloc(sizeof(X.Functia)
+1);
 strcpy(Nume,X.Nume);
 strcpy(Functia,X.Functia);
 An=X.An;
 Salariu=X.Salariu;
}
```

```
Angajat::~Angajat()
{
 printf("Obiectul (%s) a fost
distrus.\n",Nume);
 free(Nume);
 free(Functia);
}
```

```
void Angajat::SetNume(char* N)
{
 free(Nume);
 Nume=(char*)malloc(sizeof(N)+1);
 strcpy(Nume,N);
}

char* Angajat::GetNume()
{
 return Nume;
}
```

```
// Daca se poate modifica, metoda
// modifica salariul
// si returneaza valoarea 1(Adevar)
// Daca nu se poate - nu modifica
// si returneaza 0(Fals)

int Angajat::ModificaSalariu(double S)
{
 if(Salariu+S < 0) return 0;
 Salariu=Salariu+S;
 return 1;
}
```

```
void Angajat::SalariuMediu(Angajat& X,
Angajat& Y)
{
 Salariu=(X.Salariu+Y.Salariu)/2;
}

void Angajat::Afisare()
{
 printf("Nume: %s | Functie: %s |
  Anul nasterii: %4i | Salariu: %g\n",
  Nume,Functia,An,Salariu);
}
```

```
void main()
{
 clrscr();
 // obiect creat cu
 // constructorul implicit
 Angajat a;
 a.Afisare();
 a.SetNume("Petru Moraru");
 a.Afisare();
 printf("Numele angajatului (a):
          %s\n",a.GetNume());
```

```
// Urmatoarele doua instructini
// comentate sunt
// interzise, deoarece datele
// membre sunt private:

// printf("%s",a.Nume);
// a.Nume is not accesible

// a.An=1993;
// a.An is not accesible
```

```
 // obiect creat cu 1-ul constructor cu parametri
 Angajat b("Ion Padure","Manager",1992,1125.75);
 b.Afisare();

 // obiect creat cu al 2-lea constructor cu parametri
 Angajat c("Dan Balan",1984);
 c.Afisare();

 // obiect creat cu al 3-lea constructor cu parametri
 Angajat d(756.45);
 d.Afisare();

 // obiect creat cu constructorul de copiere
 Angajat e(b);
 e.Afisare();

 c.SalariuMediu(b,d); c.Afisare();
 getch();
} // sfarsitul functiei main
```

Rezultat:

```
Nume: Necunoscut | Functie: - | Anul nasterii:    0 | Salariu: 0
Nume: Petru Moraru | Functie: - | Anul nasterii:    0 | Salariu: 0
Numele angajatului (a): Petru Moraru

Nume: Ion Padure | Functie: Manager | Anul nasterii: 1992 | Salariu:
1125.75

Nume: Dan Balan | Functie: - | Anul nasterii: 1984 | Salariu: 0

Nume: Necunoscut | Functie: - | Anul nasterii:    0 | Salariu: 756.45

Nume: Ion Padure | Functie: Manager | Anul nasterii: 1992 | Salariu:
1125.75

Nume: Dan Balan | Functie: - | Anul nasterii: 1984 | Salariu: 941.1

Obiectul (Ion Padure) a fost distrus.
Obiectul (Necunoscut) a fost distrus.
Obiectul (Dan Balan) a fost distrus.
Obiectul (Ion Padure) a fost distrus.
Obiectul (Petru Moraru) a fost distrus.
```

Analiza programului principal **main** pe blocuri:

1)

```
// obiect creat cu constructorul implicit
Angajat a;
a.Afisare();
a.SetNume("Petru Moraru");
a.Afisare();
printf("Numele angajatului (a): %s\n",a.GetNume());
```

Se afişează:

```
Nume: Necunoscut | Functie: - | Anul nasterii:    0 | Salariu: 0
Nume: Petru Moraru | Functie: - | Anul nasterii:    0 | Salariu: 0
Numele angajatului (a): Petru Moraru
```

2)

```
// Urmatoarele doua instructini comentate sunt
// interzise, deoarece datele membre sunt private:
// printf("%s",a.Nume);  // a.Nume is not accesible
// a.An=1993;            // a.An is not accesible
```

3)

```
// obiect creat cu 1-ul constructor cu parametri
Angajat b("Ion Padure","Manager",1992,1125.75);
b.Afisare();
```

Se afişează:

Nume: Ion Padure | Functie: Manager | Anul nasterii: 1992 | Salariu: 1125.75

4)

```
// obiect creat cu al 2-lea constructor cu parametri
Angajat c("Dan Balan",1984);
c.Afisare();
```

Se afişează:

```
Nume: Dan Balan | Functie: - | Anul nasterii: 1984 |
Salariu: 0
```

5)

```
// obiect creat cu al 3-lea constructor cu parametri
Angajat d(756.45);
d.Afisare();
```

Se afişează:

```
Nume: Necunoscut | Functie: - | Anul nasterii:    0 |
Salariu: 756.45
```

6)

```
// obiect creat cu constructorul de copiere
Angajat e(b);
e.Afisare();
```

Se afişează:

```
Nume: Ion Padure | Functie: Manager | Anul nasterii: 1992 | Salariu: 1125.75
```

7)

```
c.SalariuMediu(b,d);
c.Afisare();
```

Se afişează:

```
Nume: Dan Balan | Functie: - | Anul nasterii: 1984 | Salariu: 941.1
```

8) Aceste mesaje apar după ce se termină lucrul programului.

Destructorii se apelează automat de către sistem pentru fiecare obiect creat din program, în ordine inversă:

```
Obiectul (Ion Padure) a fost distrus.
Obiectul (Necunoscut) a fost distrus.
Obiectul (Dan Balan) a fost distrus.
Obiectul (Ion Padure) a fost distrus.
Obiectul (Petru Moraru) a fost distrus.
```

(~§3)