Alocarea dinamică de memorie. Date și funcții statice.

Mihai Gabroveanu

Alocare dinamică de memorie

- 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 destructorii
- În programele C++ nu vom utiliza malloc() și free() pentru a aloca/elibera obiecte dinamic

Operatorul **new** pentru alocare dinamică (I)

- Operatorul **new** pentru alocarea dinamică
  ```
  idPointer = new tip;
  ```
  alocă un spațiu de memorie în **heap** pentru a reține o dată de tipul tip
- Exemplu
  ```
  float *p;
  p= new float;
  *p=2.67;
  ```


Operatorul **new** pentru alocare dinamică (II)

- operatorul new pentru alocarea dinamică și inițializare
  ```
  idPointer = new tip(expresie);
  ```
  alocă un spațiu de memorie în **heap** pentru a reține o dată de tipul tip și inițializează data din zona alocată cu expresie
- exemplu
  ```
  float *p;
  p= new float(2.67);
  ```

Operatorul **new** pentru alocare dinamică (III)

- operatorul new pentru alocarea dinamică de tablouri

  idPointer = new tip[dim_max];

  alocă un spațiu de memorie în **heap** pentru a reține un tablou de dim_max elemente de tipul tip
- exemplu

```
float *t;
t= new float[10];
t[0]=2.3;
t[1]=7.2;
```


Operatorul **delete**

- operatorul delete eliberează zonele alocate dinamic

  delete idPointer;
  delete []idPointerTablou;
- exemplu

```
int *p = new int;
float *t = new float[10];
. . .
delete p;
delete []t;
```


Alocare dinamică de obiecte (I)

- Alocare spațiu pentru un singur obiect

  idPointerObiect = 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 cu parametri lista_de_parametri

  Dacă lista de parametri e vidă atunci se apelează constructorul implicit

Alocare dinamică de obiecte (II)

- Alocare spațiu pentru un tablou de obiecte

  idPointerObiect = 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 implicit**.
- Dacă nu există un constructor implicit se generează eroare în momentul compilării

Eliberarea obiectelor alocate dinamic (I)

- Eliberare obiect alocat dinamic

  delete idPointerObiect;

  se eliberează spațiul de memorie indicat de idPointerObiect. Înainte de eliberare se apelează automat destructorul obiectului referit.

Eliberarea obiectelor alocate dinamic (II)

- Eliberare tablou de obiecte:

  delete []idPointerTablouDeObiecte;

  se eliberează spațiul de memorie indicat de idPointerTablouDeObiecte. Înainte de eliberare se apelează automat destructorul fiecărui obiect din tablou.

Exemplu

```
class Punct{
      private:
          float x;
          float y;
      public:
          Punct(){
            x=0;
            y=0;
            printf("Apel constructor
                    implicit Punct(0,0)\n");
          }
          Punct(float x, float y){
             this -> x = x;
             this -> y = y;
             printf("Apel constructor Punct(%g,%g)\n", x,y);
          }
          ~Punct(){
             printf("Apel destructor (%g, %g)\n", x,y);
           }
};

int main(){
    Punct *p1 = new Punct;
    Punct *p2 = new Punct(3,4);
    Punct *t = new Punct[3];
    delete p1;
    delete p2;
    printf("Eliberez tabloul:\n");
    delete []t;
    getch();
}
```

Output

```
Apel constructor implicit Punct(0,0)
Apel constructor Punct(3,4)
Apel constructor implicit Punct(0,0)
Apel constructor implicit Punct(0,0)
Apel constructor implicit Punct(0,0)
Apel destructor pentru obiectul (0, 0)
Apel destructor pentru obiectul (3, 4)
Eliberez tabloul:
Apel destructor pentru obiectul (0, 0)
Apel destructor pentru obiectul (0, 0)
Apel destructor pentru obiectul (0, 0)
```


Pointerul this

- Este un parametru introdus de compilator în mod implicit la apelului unei funcţii membru nestatice dintr-o clasă
- El indică către obiectul curent pentru care a fost apelată funcția membru
- Exemplu:

  idOb.numeFunctie(lista de parametri);

în momentul apelului, pe langă parametri transmişi explicit, se mai transmite un parametru numit this ce indică către idObj

Date membre statice (I)

- Datele membre statice sunt date membre ale unei clase ce sunt **partajate** de toate obiectele
- Spre deosebire de datele membre obişnuite care există în câte un exemplar pentru fiecare obiect, datele membre statice nu aparţin fiecărui obiect al clasei, ci există într-un singur exemplar comun tuturor obiectelor clasei respective.
- Se iniţializează automat cu 0 (daca sunt de tipuri predefinite)

Date membre statice (II)

- Se declară în interiorul clasei cu ajutorul cuvântului cheie static

```
class IdClasa {
    ...
    static tip idDataMembStatica;
    ...
};
```

- Se definesc în afara clasei

```
tip IdClasa::idDataMembStatica = val_initiala;
```

- Referirea datelor membre statice:

```
IdClasa:: idDataMembStatica          // Recomandat
idObiect.idDataMembStatica
idPointerObiect -> idDataMembStatica
```


Date membre statice. Exemplu

```
class Produs{
      private:
          char nume[100];
          float pret; //fara TVA
          static float PROCENT_TVA;
      public:
          Produs(char *nume, float pret){
             strcpy(this -> nume, nume);
             this -> pret = pret;
          }
          void afisare(){
               printf("Nume: %s\n", nume);
               printf("Pret fara TVA: %g\n", pret);
               printf("Pret cu TVA: %g\n", getPretCuTVA());
          }
          float getPretCuTVA(){
              return pret*(1+PROCENT_TVA);
          }
};
float Produs::PROCENT_TVA = 0.19;
```

Declarare

Definire

```
int main(){
    Produs p1("CPU", 100);
    Produs p2("HDD", 200);
    p1.afisare();
    p2.afisare();
}
```

OUTPUT

```
Nume: CPU
Pret fara TVA: 100
Pret cu TVA: 119
Nume: HDD
Pret fara TVA: 200
Pret cu TVA: 238
```


Funcții membre statice (I)

- **Funcţiile membre statice** se utilizează cu scopul de a sugera faptul că acţionează asupra instanţelor clasei văzute ca un întreg şi nu asupra instanţelor clasei în mod individual.
- Pot acționa doar asupra atributelor *statice* (ale *claselor*),
- Nu pot acţiona asupra atributelor obiectului curent (deoarece nu li se transmite pointerul *this*)
- Nu pot apela decât funcţii membre statice

Funcții membre statice (II)

- Sintaxa declarării/definiri

```
class IdCasa{
    static tip idMetodaStatica(lista_de_parametri);
};
tip IdCasa::idMetodaStatica(lista_de_parametri){
...
}
```

- Utilizare

```
IdClasa:: idMetodaStatica(lista_de_parametri); // Recomandat
idObiect.idMetodaStatica(lista_de_parametri);
idPointerObiect -> idMetodaStatica(lista_de_parametri);
```


Funcții membre statice. Exemplu

```
class Produs{
      . . .
          static float PROCENT_TVA;
      public:
          static float getTVA();
          static void setTVA(float TVA);
};
float Produs::PROCENT_TVA = 0.19;
float Produs::getTVA(){
    return PROCENT_TVA;
}
void Produs::setTVA(float TVA){
    PROCENT_TVA = TVA;
}
int main(){
    Produs p1("CPU", 100);
    Produs p2("HDD", 200);
    p1.afisare();
    p2.afisare();
    Produs:: PROCENT_TVA = 0.25; Inaccesibil!
    Produs::setTVA(0.25);
    p1.afisare();
    p2.afisare();
}
```

```
OUTPUT
Nume: CPU
Pret fara TVA: 100
Pret cu TVA: 119
Nume: HDD
Pret fara TVA: 200
Pret cu TVA: 238
Nume: CPU
Pret fara TVA: 100
Pret cu TVA: 125
Nume: HDD
Pret fara TVA: 200
Pret cu TVA: 250
```


Funcții membre constante (I)

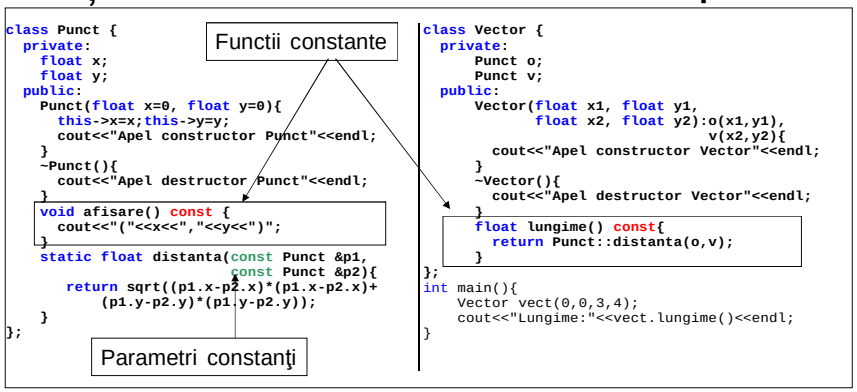
- **Funcţiile membre constante** sunt funcţii ce au acces la datele membre dintr-o clasă fără însă a putea modifica valorile acestora
- Pot apela numai funcţii membre constante

Funcții membre constante (II)

- Sintaxa declarării/definiri

```
class IdCasa{
   tip idMetodaConstanta(lista_de_parametri) const;
};
tip IdCasa::idMetodaConstanta (lista_de_parametri) const{
...
}
```

Funcții membre constante.Exemplu




```
class Punct {
  private:
    float x;
    float y;
  public:
    Punct(float x=0, float y=0){
      this->x=x;this->y=y;
      cout<<"Apel constructor Punct"<<endl;
    }
    ~Punct(){
      cout<<"Apel destructor Punct"<<endl;
    }
    void afisare() const {
      cout<<"("<<x<<","<<y<<")";
    }
    static float distanta(const Punct &p1,
                          const Punct &p2){
       return sqrt((p1.x-p2.x)*(p1.x-p2.x)+
            (p1.y-p2.y)*(p1.y-p2.y));
    }
};
```

```
class Vector {
  private:
      Punct o;
      Punct v;
  public:
      Vector(float x1, float y1,
             float x2, float y2):o(x1,y1),
                                 v(x2,y2){
        cout<<"Apel constructor Vector"<<endl;
      }
      ~Vector(){
        cout<<"Apel destructor Vector"<<endl;
      }
      float lungime() const{
        return Punct::distanta(o,v);
      }
};
int main(){
    Vector vect(0,0,3,4);
    cout<<"Lungime:"<<vect.lungime()<<endl;
}
```

Output:

```
Apel constructor Punct
Apel constructor Punct
Apel constructor Vector
Lungime:5
Apel destructor Vector
Apel destructor Punct
Apel destructor Punct
```


Temă

- Sa se implementeze clasa Matrice în care elementele sunt reprezentate sub forma unui tabel unidimensional alocat dinamic.
- Să se implementeze clasa DataCalendaristica care să permită afişarea datei sub mai multe formate (25.07.2019, 25 iulie 2019). Să se implementeze o metodă care calculează diferența dintre două date calendaristice