UNIVERSITATEA TEHNICĂ A MOLDOVEI

Programarea Orientată pe Obiecte (POO)

Prelegere

Particularități: clase și obiecte. Funcții și clase friend

UNIVERSITATEA TEHNICĂ A MOLDOVEI

SUMAR

1. Cuvântul cheie „**this**”
2. Date statice
3. Tablouri de obiecte
4. Funcții **friend**
5. Clase **friend**

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Cuvântului cheie “**this**”

§ Semnificația cuvântului cheie “**this**” este de pointer către **obiectul curent**.

§ Utilizarea lui se justifică în două situaţii:

1. De multe ori, constructorii au unul sau mai mulți parametri de același tip și cu același nume ca datele membru. În acest caz, pentru a face diferența între parametru și data membru, se utilizează cuvântul cheie “**this**”.

De exemplu, dacă data membru este x, și parametrul constructorului este tot x, atunci data membru se adresează prin **this->**x.

2. Uneori se dorește ca anumite metode să întoarcă:
   - obiectul curent
   - un pointer către acesta
   - o referință către obiectul curent

§ În toate aceste cazuri, valoarea returnată se obține cu ajutorul operatorului “**this**”.

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Exemplu de utilizare

```
class num{
 int x;
 public:
 num (int x);
 num& incr();
 void print();
};
num::num(int x){ this->x = x; }
num& num::incr() {x++; return *this; }
void num::print(){ cout<<x<<endl; }
int main(){
 num a(3);          a.print();     //afis 3
 a.incr();  a.print();              //afis 4
 a.incr().incr();
 a.print();                          //afis 6
}
```

- O prima utilizare a lui "**this**" se observă în alcătuirea constructorului. Parametrul este tot x, așa că, pentru a accesa data membru, utilizăm expresia **this->**x.
- O a doua utilizare a lui "**this**" este în cadrul metodei membru **incr**() care incrementează valoarea reținută de data membru x și returnează o referință către obiectul curent, cu ajutorul căreia se poate din nou, incrementa x.
- Astfel, dacă obiectul a are inițial valoarea 3, x va reține 3, a.**incr**() are ca efect faptul că x va reține 4, dar în același timp și referința către x, căreia i se poate aplica din nou **incr**().

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Cuvântului cheie “**this**”

- Accesul membrilor clasei prin pointerul this se realizează
  - operatorul săgeată -> pentru pointerul this la obiect: **this-**>x
  - operatorul punct . pentru pointerul this dereferenţiat: (***this**).x
- Folosirea parantezelor care încadrează pointerul dereferenţiat ***this** este obligatorie pentru că operatorul . are precedenţă mai mare decât *
- Pointerul **this** este folosit implicit pentru a referi datele membre şi funcţiile membre ale unui obiect. El poate fi folosit şi explicit
- Tipul pointerului **this** depinde de tipul obiectului şi de caracterul const sau non-const al funcţiei membre apelate de obiect
- Pointerul **this** al unui obiect nu este parte a obiectului, dar el este introdus de compilator ca prim argument în fiecare apel al funcţiilor nestatice realizat de obiect
- Fiecare obiect are acces la propria adresă de memorie

prin intermediul pointerului numit **this**

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 1

Să se creeze o clasă care să simuleze funcționalitatea funcției putere (pow).

```
class putere{
 double b;  int e;  double valoare;
 public:
 putere(double baza, int exponent);
 double calculeaza() {return this->valoare ;}
};
putere::putere(double baza, int exponent){
 this->b=baza; this->e=exponent; this->valoare=1;
 if(exponent==0) return;
 for(;exponent>0;exponent--)
 this->valoare=this->valoare * this->b;
}
```

```
int main(){
 putere x(4,3); putere y(2.5,1); putere z(5.7,0);
 cout<<x.calculeaza()<<endl;      // afiseaza 64
 cout<<y.calculeaza()<<endl;      // afiseaza 2.5
 cout<<z.calculeaza()<<endl;      // afiseaza 1
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Date membru de tip **static**

- O dată membră a unei clase poate fi declarată static în declaraţia clasei. În acest caz, va exista o singură copie a unei date de tip static, care nu aparţine nici unuia dintre obiectele (instanţele) clasei, dar este partajată de toate acestea.
- O variabilă membră de tip static a unei clase există înainte de a fi creat un obiect din clasa respectivă şi, dacă nu este iniţializată explicit, este iniţializată implicit cu 0.
- Declararea unei date statice se face în interiorul unei clase, iar definiția trebuie să se facă în afara clasei și este permisă doar o singură definiție. Sintaxa declarări unei date statice:

```
class A {
  static int i;
  public:  // ....
 };
```

  iar ulterior, definiția: **int** A::i=1;
- Este asemănătoare cu declararea unei variabile globale: **int** i=1;

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 2

Se consideră următorul Program

```
class S{
 int v;
 static int s;   // declaratia var. statice s
 public:
 S() { v = 0;}
 int gets() {return s;}   int getv() {return v;}
 void incs() {s++;}       void incv() {v++;}
};
int S::s;    // definitia var. statice s a clasei S
int main (){
 S x, y;
 cout << "Inainte de incrementare\n";
 cout <<"x.s: "<<x.gets()<<"y.s: "<<y.gets()<< endl;
 cout <<"x.v: "<<x.getv()<<"y.v: "<<y.getv()<< endl;
 x.incs();  x.incv();    cout << "Dupa incrementare\n";
 cout <<"x.s: "<<x.gets()<<"y.s: "<<y.gets()<< endl;
 cout <<"x.v: "<<x.getv()<<"y.v: "<<y.getv()<< endl;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Date membru de tip **static**

- La execuţia programului se afişează conţinutul variabilelor **s** şi **v** pentru obiectele **x** şi **y**.
- Diferenţa între comportarea unei date membre de tip **static** şi a unei date normale este: după incrementarea variabilelor **s** şi **v** pentru obiectul **x**, obiectele **x** şi **y** văd aceeaşi valoare a variabilei statice **s** şi valori diferite ale variabilei normale **v**.
- Astfel, rezultatul afișat este după cum urmează:

```
Inainte de incrementare
x.s: 0y.s: 0
x.v: 0y.v: 0
Dupa incrementare
x.s: 1y.s: 1
x.v: 1y.v: 0
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 3

De la tastatură se citesc coordonatele a n puncte. Să se elaboreze un program, cu utilizarea obiectelor de tip punct, prin intermediul căruia se va determina cele mai îndepărtate două puncte și se va afișa distanța dintre ele.

```
class Point{
 int x, y;
 public:
 Point (int x, int y){this->x=x;this->y=y;}
 void print();
 int X()const {return x;}
 int Y()const {return y;}
 static int n;
};
int Point::n=4;
void Point::print(){cout<<"x="<<x<<" "<<"y="<<y<<endl;
}
```

dr. Silviu GÎNCU

```
float distance(Point a, Point b){
 return pow(pow((b.X()-a.X()),2)+pow((b.Y()-a.Y()),2),0.5);
}
int main(){
 Point *t[Point::n];
 t[0]=new Point(2,4);   t[1]=new Point(1,6);  t[2]=new Point(5,3);   t[3]=new Point(0,0);
 cout<<"Coordonatele punctelor"<<endl;
 for(int i=0;i<(Point::n);i++) t[i]->print();
 cout<<"Punctele cele mai indepartate"<<endl;
 float m=distance(t[0],t[1]);   int p1=0,p2=1;
 for(int i=0;i<(Point::n)-1;i++)
 for(int j=i+1;j<(Point::n);j++)
 if(m<distance(t[i],t[j])){ m=distance(t[i],t[j]);  p1=i; p2=j;}
 cout<<"Distanta maxima "<<setprecision(2)<<m<<endl;
 cout<<"Coordonatele punctelor sunt"<<endl;
 t[p1]->print();  t[p2]->print();
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 4

De la tastatură se citesc datele despre n (n<100) sportivi, care au participat la o competiţie sportivă. Despre fiecare sportiv se citeşte numele, prenumele, data naşterii (an/luna/zi), şi timpul (ore:min:sec) obţinut la concurs. Să se afişeze la ecran datele despre fiecare sportiv, sportivul cu timpul minim şi sportivul de vârstă maximă. Se consideră că fiecare lună are exact 30 de zile.

Obiectul data	
Atribute	zi, luna an
Metode	Constructor, inițializare, transformare_in_zile, citire, afisare

Obiectul timp	
Atribute	ora, minute, sec
Metode	Constructor, inițializare, transformare_in_secunde, citire, afisare

Obiectul sportiv	
Atribute	Nume, prenume, data_nașterii, timpul_probei
Metode	Constructor, destructor, citire, afisare

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Clasa Data

```
class data{
 int zi, luna, an;
 public:
 data(int=0, int=0, int=0);
 void set_data(int, int, int);
 int in_zile(){return an*360+luna*30+zi;}
 void citire();
 void afisare();
};
void data::set_data(int zi, int luna, int an){
 this->zi=zi; this->luna=luna; this->an=an;
 while(this->zi>30){this->zi=this->zi-30;this->luna++;}
 while(this->luna>12){this->luna=this->luna-12;this->an++;}
}
data::data(int zi, int luna, int an){set_data(zi,luna,an);}
void data::citire(){ int zi, luna, an;
 cout<<"Scrieti data (zi/luna/an): ";
 cin>>zi>>luna>>an; set_data(zi,luna,an);
}
void data::afisare(){cout<<zi<<"/"<<luna<<"/"<<an<<" ";}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Clasa Timp

```
class timp{
 int ora, minute, sec;
 public:
 timp(int=0, int=0, int=0);
 void set_timp(int, int, int);
 int in_sec(){return ora*360+minute*30+sec;}
 void citire();
 void afisare();
};
void timp::set_timp(int ora, int minute, int sec){
 this->ora=ora; this->minute=minute; this->sec=sec;
 while(this->sec>59){this->sec=this->sec-60;this->minute++;}
 while(this->minute>59){
  this->minute=this->minute-60;this->ora++;}}
 timp::timp(int ora, int minute, int sec){set_timp(ora,minute,sec);}
void timp::citire(){int ora,  minute, sec;
 cout<<"Scrieti timpul (ora:minute:sec): ";
 cin>>ora>>minute>>sec; set_timp(ora,minute,sec);}
void timp::afisare(){cout<<ora<<":"<<minute<<":"<<sec<<" ";}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Clasa Sportiv

```
class sportiv{
 char *nume, *prenume;
 public:
 sportiv();
 ~sportiv();
 void citire();
 void afisare();
 data d;   timp t;
};
sportiv::sportiv(){nume=new char [15]; prenume=new char [15];}
sportiv::~sportiv(){delete [] nume;  delete [] prenume;}
void sportiv::citire(){
 cout<<"Dati datele despre sportiv"<<endl;
 cout<<"Nume=";cin>>nume;
 cout<<"Prenume=";cin>>prenume;  d.citire(); t.citire();
}
void sportiv::afisare(){
 cout<<nume<<" "<<prenume<<" ";
 d.afisare(); t.afisare(); cout<<endl;}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Funcția principală

```
int main(){
 sportiv s[10];
 timp tmin;
 data dmax;
 int p=0,i,n,q=0;
 cout<<"Dati nr. de sportivi "; cin>>n;
 for(i=0;i<n;i++) s[i].citire();
 dmax=s[0].d;   tmin=s[0].t;
 cout<<"Datele despre sprotivi"<<endl;
 for(i=0;i<n;i++){
  s[i].afisare();
  if(dmax.in_zile()<s[i].d.in_zile()){dmax=s[i].d;p=i;}
  if(tmin.in_sec()>s[i].t.in_sec()){tmin=s[i].t;q=i;}
 }
 cout<<"Sportivul cu timpul minim: "<<endl;
 s[q].afisare();
 cout<<"Sportivul cu virsta maxima:"<<endl;
 s[p].afisare();
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Funcții **friend**

- Funcțiile prietene (**friend**) sunt funcții ne-membre ale unei clase, care au acces la datele membre private ale unei clase. Ele se declară în interiorul clasei.
- Prototipurile unor astfel de funcții sunt precedate de cuvântul cheie **friend**.
- Spre deosebire de funcțiile membre, funcțiile prietene ale unei clase nu posedă pointerul implicit **this**.
- Accesul la datele clasei se realizează prin intermediul parametrului de tip clasă (nu prin numele obiectului clasei ca în cazul metodelor clasei).
- Declararea unei funcţii prietene f() clasei X se realizează astfel:

```
class X{
        friend tip_returnat  f(parametru_de_tip_clasă,lista_de_parametri);
};
 tip_returnat f(parametru_de_tip_clasă, lista_de_parametri){
   // corpul functiei
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 5

Se consideră clasa Array, în care toate metodele, cu excepția celor speciale sunt declarate la nivel privat. Să se creeze o funcție de tip friend prin intermediul căreia se va realiza accesul la metodele protejate din cadrul clasei.

```
class Array{
  public:
    Array(){c=0;t=NULL;}
    Array(int const);
    Array(Array const &);
    ~Array();
 private:
    void print();
    void put_n(int);
    friend void prieten(Array &c);
    int *t,c;
};
```

```
void prieten(Array &c){
 c.put_n(50); c.print();
}
int main(){
 Array a;
 prieten(a);
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 6

Să se elaboreze un program prin intermediul căruia se va determine suma elementelor a unui obiect de tip vector cu un obiect de tip matrice.

```
#define dim 8
class Matrix;
class Array{
  public:
    Array();
    ~Array();
    void print();
friend Matrix suma(Matrix &,Array &);
  private:
int *t,c;
};
Array::Array(){c=dim; t=new int[c];
 for(int i=0;i<c;i++) t[i]=rand()%99;
}
```

```
class Matrix{
  public:
     Matrix();
    ~Matrix();
     void print();
friend Matrix suma(Matrix &,Array &);
  private:  int **t,c;
};
Matrix::Matrix(){c=dim;t=new int*[c];
 for(int i=0;i<c;i++)t[i]=new int[c];
 for(int i=0;i<c;i++)
 for(int j=0;j<c;j++)
 t[i][j]=rand()%99;
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

```
Array::~Array(){delete t;}
void Array::print(){
 cout<<"Elementele vectorului: ";
 for(int i=0;i<c;i++)
 cout<<setw(5)<<t[i]; cout<<endl;
}
void Matrix::print(){
 cout<<"Elementele matric: ";
 for(int i=0;i<c;i++){
 for(int j=0;j<c;j++)
  cout<<setw(5)<<t[i][j];
 }
 cout<<endl;
}
Matrix::~Matrix(){
 for(int i=0;i<c;i++) [] delete t[i];
 delete t;
}
```

```
Matrix suma(Matrix &m, Array &a){
 Matrix k;
 for(int i=0;i<dim; i++)
  for(int j=0;j<dim; j++)
   k.t[i][j]=m.t[i][j]+a.t[i];
 return k;
}
int main(){
 Array ob_1;  ob_1.print();
 Matrix ob_2; ob_2.print();
 cout<<"suma elementelor"<<endl;
 Matrix ob_3=suma(ob_2,ob_1);
 ob_3.print();
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Clase **friend**

- În cazul în care se doreşte ca toate funcţiile membre ale unei clase să aibă acces la membrii privaţi ai altei clase (să fie funcţii prietene), prima clasă poate fi declarată clasa prietenă pentru cea de-a doua clasă conform următorului model:

```
class cls1;
class cls2{
    friend cls1;
};
```

- Relaţia de clasa prietenă nu este tranzitivă. Astfel, dacă clasa cls1 este clasa prietenă a clasei cls2, iar clasa cls2 este clasă prietenă a clasei cls3, aceasta nu implică faptul că cls1 este clasă prietenă pentru cls3.

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 7

În cadrul acestui program se prezintă modalitatea de crearea a claselor prietene

```
class Patrat;
class Dreptunghi {
 int lungime, latime;
 public:
 int aria(){return (lungime*latime); }
 void conv (Patrat a);
};
class Patrat {
 int latura;
 public:
 void seteaza_latura(int a){latura=a;}
 friend class Dreptunghi;
};
void Dreptunghi::conv(Patrat a){
 lungime=a.latura; latime=a.latura;
}
```

```
int main() {
 int lp;  Patrat ptrt;
 Dreptunghi drpt;
 cout<<"Dati lat. patratului:";
 cin>>lp;
 ptrt.seteaza_latura(lp);
 drpt.converteste(ptrt);
 cout<<"Aria dreptunghiului:";
 cout<<drpt.aria();
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Problema 8

Se consideră clasa Punct, definită prin coordonatele x și y. Să se creeze clasa segment, care va avea în calitate de date, două puncte (definite prin intermediul clasei Punct), să se implementeze o metodă de determinare a lungimii segmentului și de afișare a coordonatelor acestuia.

```
class Segment;
class Punct{ double x, y;
 void afisare();
 public:
 Punct(double x=0, double y=0);
 friend class Segment;
};
class Segment{ Punct o,v;
 public:
 Segment (Punct o, Punct v);
 double lungime();
 void afisare();
};
Punct::Punct(double x, double y){
 this->x=x; this->y=y;}
void Punct::afisare(){
 cout<<"("<<x<<","<<y<<")";}
```

```
Segment::Segment (Punct o, Punct v){
this->o=o; this->v=v;}
double Segment::lungime() {
 return sqrt((o.x-v.x)*(o.x-v.x)+(o.y-
v.y)*(o.y-v.y));}
void Segment::afisare(){
cout<<"["; o.afisare(); cout<<",";
 v.afisare(); cout<<"]";}
int main(){ Punct o(1,0), v(4,4);
 Segment s(o,v);
 s.afisare();
 cout<<"\nLungime =";
 cout << s.lungime();
}
```

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Teme pentru acasă

- A învăța și repeta conceptele prezentate în cadrul lecției
- A depana programele prezentate în cadrul lecției

dr. Silviu GÎNCU

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Prelegerea următoare

1. Supraîncărcarea operatorilor. Privire de ansamblu
2. Reguli privind supraîncărcarea operatorilor
3. Supraîncărcarea operatorilor binari
4. Supraîncărcarea operatorilor unari
5. Conversii

dr. Silviu GÎNCU