LUCRAREA DE LABORATOR NR. 3

CLASE

Domeniul programării orientate spre obiect are la bază unificarea a două concepte şi anume *datele aplicaţiei* şi *codul necesar prelucrării lor*. Pentru a ajunge la acest scop se oferă utilizatorului câteva facilităţi care permit definirea unor tipuri de date proprii şi a unor operatori care să utilizeze aceste tipuri de date. Astfel, se vor obţine tipuri de date care se comportă la fel ca şi tipurile de date şi operatorii standard (tipurile de date: întregi, reale, caracter şi respectiv, operatorii: adunare, scădere, înmulţire, împărţire, s.a.m.d.). De exemplu, un programator poate creea tipurile de date matrice şi complex, pentru a putea scrie programe în care să poată prelucra numere complexe şi să poata efectua operaţii cu matrici. Astfel, pot apărea următoarele declarări şi, respectiv, operaţii:

```
complex a, b, c;
matrice p, q, r;
. . .
c = a + b;
p = q * r;
```

Tipurile de date "complex" şi "matrice" se numesc **CLASE**; variabilele a, b, c, p, q, r se numesc **OBIECTE** (sau *instanţieri ale claselor* "complex" şi "matrice"), iar operatorii "+" şi "*" se numesc **METODE** ale aceloraşi clase. Prin unificarea datelor înţelegem că o **CLASĂ** conţine atât structurile de date necesare descrierii unui **OBIECT**, cât şi **METODELE (funcţiile)** cu ajutorul cărora obiectele în cauză vor fi manipulate. În acest mod creşte gradul de abstractizare dar programele devin mai uşor de înteles, de depanat şi de întreţinut.

Clase

Clasele sunt implementarea unor tipuri abstracte de date (***Abstract Data Type - ADT***). O caracteristică importantă a unui limbaj de programare este aceea de a oferi programatorului posibilitatea de a-şi construi tipuri de date ca cele standard, acestea numindu-se tipuri definite de utilizator (*user defined*). În limbajul C se poate folosi cuvântul cheie ***typedef*** prin intermediul căruia se poate defini orice tip de date utilizator. Se mai poate folosi împreună cu tipul structură (***struct***) obţinându-se astfel noi tipuri de date. Cu toate acestea, operaţiile care se pot utiliza asupra acestor noi tipuri de date rămân tot cele standard, acestea acţionând numai asupra tipurilor standard.

O clasă reprezintă o metodă logică de organizare a datelor şi funcţiilor unei aceleaşi familii. O clasă este declarată folosind cuvântul cheie **class**, care are o funcţionalitate similară cu cuvântului cheie **struct** din C. Diferenţa fundamentală este aceea că o clasă poate include între membrii săi şi funcţii, nu numai date ca în structura **struct** din C. Forma generală a unei clase este:

```
class nume_clasă
{
 etichetă_permisiune_1:
                              membrul 1;
etichetă_permisiune_2:
                              membrul 2;  ...
} nume_obiect;
```

unde **nume_clasă** reprezintă numele clasei (tip definit de utilizator), iar câmpul opțional **nuime_obiect** reprezintă unul sau mai mulți identificatori valizi de obiecte. Corpul declarației poate conține **membrii**, care pot fi atât declarații de date, cât și declarații de funcții și, opțional, **etichete de permisiune** (**specificatori de acces**), care pot fi una din următoarele cuvinte cheie: **private:**, **public:** sau **protected:**. Acestea fac referință la permisiunea pe care membrii o dobândesc:

- **private** - membrii clasei sunt accesibili numai din cadrul altor membri ai aceleiași clase sau din clase prietene (***friend***) ;
- **protected** - membrii clasei sunt accesibili din cadrul membrilor aceloraşi clase sau clase prietene (***friend***) și, totodată, din membrii claselor derivate (***derived***);
- **public** - membrii sunt accesibili de oriunde clasa este vizibilă.

Efectul unui specificator de acces durează până la următorul specificator de acces.
O clasă reunește datele și operațiile în cadrul aceluiași tip de dată. În plus, există posibilitatea protejarii elementelor componente, atât date cât și funcții membre, protejarea nu împotriva necunoașterii lor de către utilizator ci mai ales împotriva distrugerii lor accidentale. Acestă proprietate se numește **încapsulare**.
Deci, o

clasa = date + operatii.

Membrii unei clase au implicit atributul de acces **private**. Metodele asociate datelor trebuie să fie însă accesibile utilizatorului clasei, astfel încât trebuie să fie declarate **public**. Se admite ca în definiția clasei să apară doar prototipurile funcțiilor membre. Definițiile efective ale funcțiilor se pot face ulterior, în orice alt loc în cadrul programului.
Dacă se declară membrii unei clase înainte de a include o etichetă de permisiune, aceștia vor fi considerați (implicit) **private**.
O clasă are un domeniu de definiţie (este cunoscută în acest domeniu) care începe de la prima poziţie după încheierea corpului clasei şi se întinde până la sfârşitul fişierului în care este introdusă definiţia ei şi al fişierelor care îl includ pe acesta. Datele şi funcţiile membre ale clasei care nu sunt declarate public au în mod implicit, ca domeniu de definiţie, domeniul clasei respective, adică sunt cunoscute şi pot fi folosite numai din funcţiile membre ale clasei. Datele şi funcţiile membre publice ale clasei au ca domeniu de definiţie întreg domeniul de definiţie al clasei, deci pot fi folosite în acest domeniu. Pentru definirea unei funcţii în afara clasei (dar, bineînţeles în domeniul ei de definiţie) numele funcţiei trebuie sa fie însoţit de numele clasei respective prin intermediul operatorului de rezoluţie (::). Sintaxa de definire a unei funcţii în afara clasei este următoarea:

tip_returnat nume_clasă::nume_funcţie(listă_argumente)
{
corpul funcţiei
}

Exemplul 1: Declararea unei clase și a unui obiect aparținând acelei clase.

```
class CDreptunghi
{
          int x, y;
  public:
           void set_valori (int,int);
           int aria (void);
  } drept;
```

S-a declarat clasa **CDreptunghi** și un obiect **drept** al acestei clase. Această clasă conține 4 membri: două variabile de tipul **int** (**x** și **y**) în secțiunea **private** (fiind secțiunea cu permisiune implicită) și două funcții în secțiunea **public**: **set_valori()** și **aria()**, în cazul acestora declarându-se numai prototipul lor.
Trebuie înțeleasă diferența dintre numele clasei și numele obiectului. În exemplul anterior **CDreptunghi** reprezintă numele clasei (adică un tip utilizator), pe când **drept** este un obiect de tip **CDreptunghi**. Este vorba de exact aceeași diferență ca cea dintre int și a din declarația următoare:
int a;
unde **int** este tipul (*numele clasei*), iar **a** este variabila (*numele obiectului*).

Exemplul 2: Program în care se declară o clasă CDreptunghi, un obiect drept, se setează dimensiunile dreptunghiului (lungime și lățime) la valorile 5, respectiv 4 și se afișează aria dreptunghiului.

```
#include <iostream.h>

class CDreptunghi
 {
         int x, y;
  public:
        void set_valori (int,int);
        int aria (void) {return (x*y);}
};
```

```
void CDreptunghi::set_valori (int a, int b)
{
  x = a;
  y = b;
}

int main ()
{
CDreptunghi drept;
drept.set_valori (5,4);
cout << "aria: " << drept.aria();
return 0;
}
```

În exemplul 2 funcțiile **set_valori** și **aria** aparțin, ambele, clasei **CDreptunghi**. Diferența apare la nivelul definirii lor. În timp ce funcția **aria** este definită în interiorul clasei, funcția **set_valori** este declarată în interiorul clasei și definită în afara clasei.
Ca regulă, se preferă definirea funcțiilor care au un corp mic în interiorul claselor, iar în cazul funcțiilor cu corp mare se preferă declararea în interiorul clasei și definirea în afară.

Făcând analogia clasă-tip de date, se subînțelege că se pot declara mai multe obiecte aparținând aceleași clase.

Exemplul 3:

```
#include <iostream.h>

class CDreptunghi
{
    int x, y;
  public:
    void set_valori (int,int);
    int aria (void) {return (x*y);}
};

void CDreptunghi::set_valori (int a, int b)
{
  x = a;
  y = b;
}
```

```
int main ()
{
  CDreptunghi drept1, drept2;
  drept1.set_valori (5,4);
  drept2.set_valori (6,5);
  cout << " aria drept1: " << drept1.aria() << endl;
  cout << " aria drept2: " << drept2.aria() << endl;
  return 0;
}
```

Pointeri la clase

Este posibilă crearea de pointeri care să "arate" (indice) către clase. Pentru acest lucru se va considera că odată declarată clasa, aceasta devine un tip valid, deci se va putea folosi numele clasei ca tip al acelui pointer. De exemplu:

CDreptunghi * drept;

este un pointer la un obiect al clasei CDreptunghi.

La fel ca în cazul structurilor, pentru a referi (accesa) direct un membru al obiectului de tip pointer la tipul clasă, se va folosi operatorul ->. Iată un exemplu de combinații posibile:

Exemplul 4:

```
#include <iostream.h>

class CDreptunghi
{
    int lungime, latime;
  public:
    void set_valori (int, int);
    int aria (void) {return (lungime * latime);}
};

void CDreptunghi::set_valori (int a, int b)
{
  lungime = a;
  latime = b;
}

int main ()
{
  CDreptunghi a, *b, *c;
  CDreptunghi * d = new CDreptunghi[2];
  b= new CDreptunghi;
```

```
  c= &a;
  a.set_valori (1,2);
  b->set_valori (3,4);
  d->set_valori (5,6);
  d[1].set_valori (7,8);
  cout << "aria lui a: " << a.aria() << endl;
  cout << "aria dreptunghiului indicat de *b: " << b->aria() << endl;
  cout << " aria dreptunghiului indicat de *c: " << c->aria() << endl;
  cout << "aria primului dreptunghi d - d[0]: " << d[0].aria() << endl;
  cout << "aria celui de al doilea dreptunghi d - d[1]: " << d[1].aria() << endl;
  return 0;
}
```

Efectul execuției programului este:

aria lui a: **2**
aria dreptunghiului indicat de *b: **12**
aria dreptunghiului indicat de *c: **2**
aria primului dreptunghi d - d[0]: **30**
aria celui de al doilea dreptunghi d - d[1]: **56**

În următorul tabel sunt recapitulate metodele pentru a interpreta pointeri și operatorii (*****, **&**, **.**, **->**, **[ ]**) care apar în exemplul 4.

***x**	se interpretează	indicat de către pointerul **x**
&x	se interpretează	adresa lui **x**
x.y	se interpretează	membrul **y** al obiectului **x**
(*x).y	se interpretează	membrul **y** al obiectului indicat de către pointerul **x**
x->y	se interpretează	membrul **y** al obiectului indicat de către pointerul **x** (echivalent cu expresia precedentă)
x[0]	se interpretează	primul obiect indicat de către **x** (vector, tablou)
x[1]	se interpretează	al doilea obiect indicat de către **x** (vector, tablou)
x[n]	se interpretează	al (n+1)-lea obiect indicat de către **x** (vector, tablou)

Desfășurarea lucrării

1. Pentru programul din Exemplul 2 să se realizeze următoarele modificări:
- În interiorul clasei să se definescă funcţia *set_valori ()* şi să se declare funcţia *aria()* ;
- În exteriorul clasei să se definească funcţia aria();
- Să se introducă de la tastatură valorile pentru dimensiunile dreptunghiului.

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

class CDreptunghi
{
        int x, y;
public:
        void set_valori (int a, int b)
        {
                x = a;
                y = b;
        }

        int aria (void);
};

int CDreptunghi::aria (void)
        {return (x*y);}

int main ()
{
        int a,b;
        CDreptunghi drept;
        cout << "introduceti lungimea dreptunghiului: \n a=";
        cin >> a;
        cout << "introduceti latimea dreptunghiului: \n b=";
        cin >> b;
        drept.set_valori (a,b);
        cout << "Aria dreptunghiului este: " << drept.aria();
        getch();
        return 1;

}
```

2. Să se definească o clasă Cerc având:
- Date private – *raza*;
- Funcţii publice *setarea_razei()*, *lungimea()* cercului şi *aria()* cercului.

Se declară obiectul cerculeţ de tip Cerc. Se introduce de la tastatură raza cerculeţului şi se afişează lungimea şi aria cerculeţului.

```
#include <iostream.h>
#include <stdio.h>
#include <math.h>

class Cerc
{
	int raza;
public:
	void set_raza(int a){raza=a;};
	float lungime (void) {return (2*M_PI*raza);};
	float arie (void) {return (M_PI*raza*raza);};

};

int main()
{
	Cerc cerculet;
	int r;
	cout << "introduceti raza cercului \n raza=";
	cin  >> r;
	cerculet.set_raza(r);
	cout << "Lungimea cercului este: " << cerculet.lungime();
	cout << "\nAria cercului este: " << cerculet.arie();
	return 1;
}
```

3. Să se rescrie programul creat la Exercițiul 1 astfel încât să poată fi introduse de la tastatură un număr de "n" dreptunghiuri pentru care să se precizeze dimensiunile și să se afișeze ariile; toate instanțele clasei **CDreptunghi** vor fi de tip pointer (**CDreptunghi *instanță** sau **CDreptunghi *instanță[n]**). La terminarea programului se va apela operatorul de ștergere (eliberare a memoriei) **delete** pentru toate instanțele create!

```
// exemplu de pointeri la clase
#include <iostream.h>

class CDreptunghi
{
	int x, y;
public:
	void set_valori (int, int);
	int aria (void) {return (x * y);}
};

void CDreptunghi::set_valori (int a, int b)
```

```
{
	x = a;
	y = b;
}

int main ()
{
	int n,a,b,i;
	cout << "introduceti numarul de instante: ";
	cin >> n;
	CDreptunghi * instanta = new CDreptunghi[n];
	for (i=0; i<n ; i++)
	{
		cout << "(" <<i<<") introduceti lungimea, latimea: ";
		cin >> a >> b;
		instanta[i].set_valori (a,b);
		cout << "   aria instantei[" << i+1 << "] : " << instanta[i].aria() << endl;
	}

	return 0;
}
```

4. Să se descrie o clasă CComplex ce permite memorarea părţii reale şi a celei imaginare într-un număr complex şi calculează modulul acestuia. Datele nu vor dispune de nicio protectie în interiorul clasei astfel încât se poate avea acces direct la acestea din afara clasei. Să se realizeze un program care foloseşte această clasă şi citeşte date într-un număr complex afişându-l împreună cu modulul sau.

```
#include <iostream>
#include <fstream>
#include <math.h>
using namespace std;

class CComplex
{
public:
	float a,b;
	float modul()
	{
		return sqrt(pow(a,2)+pow(b,2));
	}
};

void main()
{
```

```
	complex z;
	cout<<"Introduceti nr complex: \n";
	cout<<"\tPartea reala: ";
	cin>>z.a;
	cout<<"\tPartea imaginara: ";
	cin>>z.b;

	cout<<"\nNumarul complex este: ("<<z.a<<","<<z.b<<")";
	cout<<"\nModulul este: "<<z.modul()<<endl;
	getch();
}
```

5. Să se descrie o clasă CComplex ce permite memorarea părții reale și a celei imaginare într-un număr complex și calculează modulul acestuia. Datele nu vor dispune de nicio protectie în interiorul clasei astfel încât se poate avea acces direct la acestea din afara clasei. Să se realizeze un program care citește un șir de numere complexe și afișează acele numere care au maximul si minimul dintre module împreună cu modulele lor.

```
#include <iostream>
#include <fstream>
#include <math.h>
using namespace std;
class CComplex
{
public:
	float a,b;
	float modul()
	{
		return sqrt(pow(a,2)+pow(b,2));
	}
};
void main()
{
	CComplex v[20];			//un şir de maxim 20 elemente reprezentând
					//numere complexe
	float max, min;			//modulele minim şi maxim
	int n;				 //dimensiunea reala a şirului
	int m, M;			//indicele elementului minim, respectiv, maxim

	cout<<"Introduceti n: ";
	cin>>n;
	for(int i=0;i<n;i++)
	{
		cout<<"Numarul complex "<<i+1<<" este: "<<endl;
		cout<<"\tpartea reala: ";
		cin>>v[i].a;
		cout<<"\tpartea imaginara: ";
```

```
            cin>>v[i].b;
        }

        max=v[0].modul();
        min=v[0].modul();
        m=0;
        M=0;

        for(int i=1;i<n;i++)
            if (v[i].modul()<min)
            {
                min=v[i].modul();
                m=i;
            }
            else if (v[i].modul()>max)
            {
                max=v[i].modul();
                M=i;
            }

cout<<"\nNr complex cu modul maxim: ("<<v[M].a<<","<<v[M].b<<") - modul: "<<max;
cout<<"\nNr complex cu modul minim: ("<<v[m].a<<","<<v[m].b<<") - modul: "<<min;
 getch();
}
```

6. Să se realizeze o aplicație cu un tip abstract de date (o clasă) ce permite memorarea datelor într-un număr complex, caz în care partea reală şi cea imaginară se consideră a fi protejate spre vizibilitate în afara clasei. Clasa va conține elementele de bază, adică, memorarea datelor într-un număr complex şi afişarea acestuia.

```
#include <iostream>
#include <fstream>
#include <math.h>
using namespace std;
class CComplex
{
private:                                    //era implicit
        float a,b;                          //a - partea reala, b - partea imaginara
public:

 /*funcție ptr. introducerea datelor într-un număr complex
acesta funcție va fi definită în interiorul clasei, catalogata fiind ca şi funcție in-line*/



        void citire(char* s)
        {
```

```
            cout<<"Introduceti datele nr. complex "<<s<<endl;
            cout<<"\tpartea reala: ";
            cin>>a;
            cout<<"\tpartea imaginara: ";
            cin>>b;
        }
```

*/*funcţie pentru afişarea datelor într-un număr complex declarată în interiorul clasei şi definită în exteriorul acesteia, catalogată find ca funcție off-line*/*

```
        void afisare(char*);
};
```

// definirea funcției afișare în exteriorul clasei

```
void complex::afisare(char* s)
{
        cout<<"Nr. complex "<<s<<" este: "<<"Re:" <<a<<"Im"<<b<<endl;
}

void main()
{
        CComplex z1;            //declarăm un obiect (static)la clasa anterior definită
        z1.citire("z1");        // apelăm funcțiile de citire şi afişare pentru obiectul z1
        z1.afisare("z1");
        cout<<endl;             // introducem un rând liber
        CComplex  * z2;         //declarăm un obiect (dinamic)la clasa anterior definită
        z2=new CComplex;
        z2->citire("z2");
        z2->afisare("z2");

        getch();
}
```

4) Să se realizeze o aplicație cu un tip abstract de date (o clasă) ce permite memorarea datelor într-un număr complex, caz în care partea reală şi cea imaginară se consideră a fi protejate spre vizibilitate în afara clasei. Clasa va conține elementele de bază, adică, memorarea datelor într-un număr complex şi afişarea acestuia. Se vor realiza şi operațiile de sumă, respectiv, produs între două numere complexe.

```
#include <iostream>
#include <fstream>
#include <math.h>
using namespace std;
class CComplex
{
```

```
private:                        //era implicit
        float a,b;              //a - partea reală, b - partea imaginară
public:
        void citire(char* s)
        {
                cout<<"Introduceti datele nr. complex "<<s<<endl;
                cout<<"\tpartea reala: ";
                cin>>a;
                cout<<"\tpartea imaginara: ";
                cin>>b;
        }

        void afisare(char*)
        {
        cout<<"Nr. complex "<<s<<" este: "<<"Re:" <<a<<"Im"<<b<<endl;
        }


        / / TREI  VARIANTE DE CALCUL A SUMEI A DOUĂ NUMERE COMPLEXE

        CComplex CComplex::suma1(CComplex z)            //returnează valoarea
        void CComplex::suma2(CComplex z, CComplex& r)
                                                // utilizează parametru referință
        void CComplex::suma3(CComplex z, CComplex* r)   //utilizează parametru pointer


        //PRODUSUL A DOUA NUMERE COMPLEXE
        CComplex produs(CComplex);
};


CComplex CComplex::suma1(CComplex z)
{
        CComplex r;
        r.a=a+z.a;
        r.b=b+z.b;
        return r;
}

void CComplex::suma2(CComplex z, CComplex& r)
{
        r.a=a+z.a;
        r.b=b+z.b;
}

void CComplex::suma3(CComplex z, CComplex* r)
{
```

```
        r->a=a+z.a;
        r->b=b+z.b;
}

void main()
{
        CComplex z1;
        z1.citire("z1");
        z1.afisare("z1");
        cout<<endl;
        CComplex* z2;
        z2=new CComplex;
        z2->citire("z2");
        z2->afisare("z2");
        cout<<endl
        CComplex z3;
        z3=z1.suma1(*z2);
 /*s-a calculat suma dintre nr. complexe z1 și z2, sumă ce se va înscrie în z3*/
        z3.afisare("z1+z2");
        CComplex z4;
        z1.suma2(*z2,z4);           //z4=z1+z2
        z4.afisare("z1+z2");
        CComplex z5;
        z1.suma3(*z2,&z5);          //z5=z1+z2
        z5.afisare("z1+z2");
        getch()
}
```