Laborator 4 – Moştenirea

Tema 4.1

Analizaţi programul din fişierele `Lab4.H,Lab4.CPP, CERC.H, CERC.CPP` din anexa 4.

Tema 4.2

Să se împartă clasa `CERC` în două clase astfel încât clasa `CERC` să devină o clasă care moşteneşte o altă clasă de bază. Clasa de bază trebuie făcută astfel încât să reprezinte un concept util.

Consideraţii teoretice 4.2

Unul din avantajele programării orientate pe obiecte este faptul ca se pot reutiliza şi completa noi informaţii la codul deja existent. Această completare poate fi realizată chiar dacă nu se deţine codul sursă al clasei de bază (de exemplu îmbunătăţirea claselor din biblioteci). În contextul moştenirii clasa care se moşteneşte se numeşte clasă de bază iar clasa care moşteneşte se numeşte clasă derivată. Obiectul de bază (cel care se moşteneşte) este îmbunătăţit prin derivare cu membrii noi păstrându-se membrii şi metodele deja avute.

Forma generală de definire a unei clase derivate este:

```
specific_clasa nume_clasa_derivata [:modificator_acces nume_clasa_baza]
      {
      [ [public:    ] lista_membrii_1 ]
      [ [private:   ] lista_membrii_2 ]
      }[lista_obiecte];
```

unde `specific_clasa` poate fi `struct` sau `class` (cu singura diferenţă că la `struct` membrii sunt implicit publici iar la `class` membrii sunt impliciţi privaţi). `modificator_acces` reprezintă modul de moştenire al membrilor din clasa de bază şi poate fi (deocamdată) unul din cuvintele cheie `public` sau `private`. Sintaxa prezentată este o generalizare a cazului de definire a unei clase fără a ne baza pe moştenire (dacă lipseşte clasa de bază).

Din punct de vedere al dimensiunii obiectului rezultat prin moştenire aceasta este egală cu dimensiunea obiectului moştenit la care se adaugă suma dimensiunii membrilor noi adăugaţi în clasa derivată.

Drepturile de acces ale membrilor din clasa derivată:

- În ceea ce priveşte membrii noi definiţi aceştia vor avea drepturile de acces corespunzătoare modului de definire a lor.
- În ceea ce membrii moşteniţii din clasa de bază acestora se vor stabili drepturile de acces în funcţie de drepturile de acces avute în clasa de bază şi în funcţie de tipul moştenirii folosite astfel:

Drept acces avut în clasa bază	Tip moştenire	Drept acces rezultat în clasa derivată
`public`	`public`	`public`
`private`		`inaccesibil !!!!`
`public`	`private`	`private`
`private`		`inaccesibil !!!!`

Un membru definit `public` în clasa de bază va primi dreptul de acces în clasa derivată în funcţie de modificatorul de acces folosit în cazul moştenirii. La un membru definit `private` în clasa de bază nici o metodă nouă din clasa derivată nu va avea acces la el. Acel membru va exista în clasa derivată (clasa derivată va aloca spaţiu pentru memorarea acelui membru) dar nici o metodă nou definită din clasa derivată nu va avea acces direct la acel membru decât prin intermediul metodelor moştenite din clasa de bază. Prin moştenire nu se modifică tipul de acces la membrii pentru clasa de bază ci doar accesul la aceştia ca şi membrii ai clasei derivate.

Deci din clasa de bază se vor moşteni toţi membrii, cu (deocamdată) o excepţie: constructorii şi destructorii nu se moştenesc (ci se apelează).

În cazul în care în clasa derivată se defineşte un nou membru cu acelaşi nume şi aceeaşi sintaxă ca un membru deja existent în clasa de bază, acesta din urmă nu dispare şi va putea fi accesat prin specificarea în fată exact a clasei din care face parte.

Indicaţii 4.2

- ⇨ Se vor împărţi membrii clasei `CERC` în membrii moşteniţi din clasa de baza şi în membrii adăugaţi de clasa derivata.
- ⇨ În clasa de bază membrii vor avea atributul public iar moştenirea va fi tot publică.

Tema 4.3

Să se adauge în clasa de bază (numită `POZITIE`) un constructor implicit şi un constructor cu doi parametrii. Aceştia vor iniţializa variabilele membru `x` şi `y` cu ceea ce era în constructorii clasei `CERC`. Constructorii clasei derivate nu vor mai iniţializa membrii moşteniţi `x` şi `y`.

Consideraţii teoretice 4.3

În cazul obiectelor construite prin derivare (moştenire) constructorii şi destructorii obiectelor de bază nu se moştenesc ci se apelează din clasa de bază.

La instanţierea unui obiect prima dată se apelează constructorul clasei derivate după care se apelează constructorul clasei de bază urmând ca să se execute prima dată codul constructorului clasei de bază şi pe urmă cel al constructorului clasei derivate. Execuţia constructorilor se face de la bază (interior, sâmbure) spre derivată (exterior, coajă) chiar dacă apelul constructorilor se face invers.

La distrugerea unui obiect prima dată se apelează şi se execută destructorul clasei derivate şi pe urmă se apelează şi se execută destructorul clasei de bază.

Indicaţii 4.3

- ⇨ Atenţie la poziţia cercurilor pe ecran. Explicaţi.

Tema 4.4

Să se apeleze explicit constructorii corespunzători din clasa de bază.

Consideraţii teoretice 4.4

In contextul moştenirii constructorii clasei de bază nu se moştenesc, aceştia urmând să se apeleze de fiecare data de câte ori un obiect de tipul clasei derivate este instanţiat. Întotdeauna ordinea de

execuţie a constructorilor este dinspre clasa de bază spre clasa derivata, prima dată executându-se constructorul clasei de bază şi pe urmă cel al clasei derivate.

La rularea aplicaţiei de la tema 4.3 s-a obţinut o imagine în care toate cercurile aveau acelaşi centru. Aceasta s-a datorat faptului că toţi constructorii din clasa derivată (`CERC`) apelează implicit acelaşi constructor din clasa de bază (care stabileşte poziţia cercurilor pe ecran). În cazul în care nu se specifică explicit care constructor din clasa de bază să se apeleze, compilatorul va apela implicit constructorul fără parametrii din clasa de bază (dacă acesta există) sau se va genera o eroare dacă acesta nu există. Pentru a apela explicit constructorul unei clase de bază se poate folosi următoarea sintaxă:

```
cls_deriv::cls_deriv([lista_param_formali]):cls_baza([lista_param_actuali])
  {
     // corp al constructorului clasei derivate
     ...
  }
```

unde `cls_deriv` reprezintă numele clasei de derivate (constructorul din clasa derivată) iar `cls_baza` reprezintă numele clasei de bază (constructorul din clasa de bază care se doreşte apelat). Specificarea explicită a constructorului din clasa de bază care se doreşte apelat se face doar la definirea (implementarea) constructorului respectiv din clasa derivată.

`lista_param_formali` reprezintă declararea de parametrii formali iar `lista_param_actuali` reprezintă expresiile de apel.

Indicaţii 4.4

- ⇨ Apelul explicit al constructorului implicit al clasei de baza poate lipsi fără a schimba funcţionarea programului. Verificaţi.

Tema 4.5

Să se modifice atributul membrilor din clasa de baza pentru a nu pierde încapsularea (din punct de vedere al ascunderii datelor).

Consideraţii teoretice 4.5

Aşa cum este descrisă moştenirea până în acest moment apare o dilemă. Dacă vrem să avem moştenire şi acces la membrii moşteniţi ar trebui ca în clasa de bază aceştia să fie definiţi `public` pierzându-se astfel facilităţile oferite de încapsulare. Dacă vrem să păstrăm încapsularea în clasa de bază avem probleme la moştenire. Deci cu alte cuvinte membrii publici nu asigură încapsulare iar membrii privaţi nu asigură moştenire. Pentru rezolvarea acestei dileme în limbaj a fost introdus un nou tip de acces numit `protected`. Un membru definit `protected` poate fi accesat atât de membrii clasei în care e definit cât şi de membrii claselor care moştenesc clasa respectivă. Membrii `protected` nu pot fi accesaţi din exteriorul clasei sau claselor care moştenesc clasa respectivă. Deci un membru definit `protected` este văzut ca un membru `public` în contextul moştenirii şi ca un membru `private` pentru exteriorul claselor.

Tabelul cu modificatorii de acces în contextul moştenirii care include şi tipul `protected` atât ca şi tip de acces cât şi ca tip de moştenire devine ca în tabelul următor.

Astfel un membru definit `protected` poate oferi încapsulare şi moştenire la oricât de multe nivele iar în momentul în care se doreşte „închiderea” unei clase (nu se mai permit dezvoltări ulterioare) se

poate folosi la moştenire modificatorul de acces `private` astfel încât ulterior membrii moşteniţi să devină inaccesibili.

Drept acces avut în clasa de bază	Tip moştenire	Drept acces rezultat în clasa derivată
`public`	`public`	`public`
`private`		`inaccesibil !!!!`
`protected`		**`protected`**
`public`	**`protected`**	**`protected`**
`private`		**`inaccesibil!!!`**
`protected`		**`protected`**
`public`	`private`	`private`
`private`		`inaccesibil !!!!`
`protected`		**`private`**

Sintaxa de declarare a unei clase derivate devine (cu protected):

```
specific_clasa nume_clasa_derivata [:modificator_acces nume_clasa_baza]
      {
      [ [public:    ] lista_membrii_1 ]
      [ [protected: ] lista_membrii_2 ]
      [ [private:   ] lista_membrii_3 ]
      }[lista_obiecte];
```

Indicaţii 4.5

⇨ Atenţie la atributul membrilor `r` şi `c` din clasa `CERC` pentru a permite moştenirea ulterioara a acestei clase.

Tema 4.6

Sa se modifice aplicaţia astfel încât la suprapunerea cercurilor afişarea acestora să rămână curata.

Consideraţii teoretice 4.6

Există două posibilităţi pentru a putea menţine imaginea celorlalte cercuri intacte:

- prima posibilitate prevede salvarea parţială a ecranului în momentul în care se face desenarea iar în cazul mutării se restaurează partea salvată. Această posibilitate necesită multă memorie în funcţie de rezoluţie şi adâncimea de culoare folosită.
- a doua posibilitate o reprezintă redesenarea completă a părţii afectate de mutare. Astfel, în momentul în care într-o zonă de ecran are loc o modificare a imaginii se va reface doar acea zonă de ecran, urmând ca restul imaginii să rămână nemodificată. Această soluţie este folosită şi de către interfaţa grafică din Windows care cere explicit aplicaţiei să redeseneze partea afectată a interfeţei.

Indicaţii 4.6

⇨ Pentru simplitate redesenaţi de cate ori e cazul toate cercurile.

Tema 4.7

Să se execute pas cu pas apelul constructorilor pentru a observa apelul şi execuţia acestora.

Consideraţii teoretice 4.7

A se revedea consideraţiile teoretice 3.8.

Tema 4.8

Să se introducă în aplicaţie facilităţile de inserare, respectiv ştergere, de noi cercuri la apăsarea tastelor „3" şi „4". Inserarea se va face întotdeauna în vector după ultimul cerc existent (fără a depăşi o dimensiune maximă impusă, de exemplu 20) iar ştergerea va elimina ultimul cerc din vector (fără însă a şterge şi cele 6 cercuri iniţiale).

Consideraţii teoretice 4.8

Tastele „3" şi „4" sunt taste normale pentru codurile sunt '3' respective '4'.

Indicaţii 4.8

- Introduceţi o variabila `NrCercuri`.
- Atenţie la codul aferent tastei `TAB`
- Nu uitaţi să ştergeţi cercul de pe ecran (cu metoda `Sterge`) înainte de a îl distruge (cu `delete`).

Tema 4.9

Să se ataşeze fiecărui cerc şi un nume (şir de caractere). Acesta va fi dat ca parametru în constructor şi va fi afişat în tot timpul funcţionării în centrul cercului. Se va lua în considerare scenariul cel mai defavorabil în ceea ce priveşte durata de viaţă a şirului primit ca şi parametru în constructor.

Consideraţii teoretice 4.9

În mediul de programare BORLAND C++ există în biblioteca grafică mai multe funcţii pentru afişarea pe ecran în mod grafic a unui anumit text de o anumită formă şi pe o anumită poziţie.

Funcţia:

```
void settextjustify(int horiz, int vert);
```

specifică modul de afişare pe ecran a textului în mod grafic. Textele afişate după apelul acestei metode vor fi aşezate orizontal şi vertical în jurul poziţiei curente, după cum este specificat. Există mai multe valori implicite definite în biblioteca `graphics.h` pentru parametrii `horiz` (`LEFT_TEXT, CENTER_TEXT, RIGHT_TEXT`) şi `vert (BOTTOM_TEXT, CENTER_TEXT, TOP_TEXT)`.

Această funcţie afectează textele scrise pe ecran cu ajutorul funcţiei `outtextxy` şi nu poate fi utilizată în modul text.

```
void fat outtextxy(int x, int y, char far *textstring);
```

Funcţia `outtextxy` afişează pe ecran un şir de caractere specificat prin parametrul `textstring` la poziţia specificată prin parametrii `x` şi `y`.

Indicaţii 4.9

- Atenţie la alocarea memoriei pentru şirul de caractere în constructor şi eliberarea memoriei în destructor.

Anexa 4

LAB4.H

```
#define TAB    9            //definire constante cu codurile ASCII ale tastelor
#define ESC   27

#define LEFT  75
#define RIGHT 77
#define UP    72
#define DOWN  80

#define UNU    ,1'
#define DOI    ,2'

// prototipuri de functii

void OurInitGraph(void);
```

LAB4.CPP

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

#include "lab4.h"
#include "cerc.h"

#define MAX_FIGURI 6                //Numarul maxim de cercuri care pot exista

CERC *pc[MAX_FIGURI];               //variabile Globale
CERC cg1,cg2(100,100,25,BLUE);      //două cercuri definite global

void main()
//*************************************************************************
{
int CercCurent=0,gata=0,k;                 //variabile locale
CERC cl1(500,300,75,RED),cl2(200,100,30); //doua cercuri definite local

//cod

  OurInitGraph();

  pc[0]=&cg1;
  pc[1]=&cg2;
  pc[2]=&cl1;
  pc[3]=&cl2;
  pc[4]=new CERC(400,200,100,YELLOW);   //cercuri definite dinamic
  pc[5]=new CERC;

  for(k=0;k<MAX_FIGURI;k++)         //afisare pe ecran a tuturor cercurilor
    pc[k]->Afiseaza();

  while(!gata)                      //bucla program principal
   switch(getch())                  //citire tasta curenta
   {
     case ESC:                      //terminare program
```

```
        gata=1;
        break;
      case TAB:                     //trece la urmatorul cerc
        CercCurent++;
        CercCurent%=MAX_FIGURI;
        break;
      case 0:                       //tasta speciala?
        switch(getch())
        {
          case LEFT:  pc[CercCurent]->Muta(   -10,  0 ); break;
          case RIGHT: pc[CercCurent]->Muta(    10,  0 ); break;
          case UP:    pc[CercCurent]->Muta(     0,-10 ); break;
          case DOWN:  pc[CercCurent]->Muta(     0, 10 ); break;
          default:    break;
        };break;
       case UNU:    pc[CercCurent]->Creste( +10     ); break;
       case DOI:    pc[CercCurent]->Creste( -10     ); break;

   }
  closegraph();
}

void OurInitGraph()
//******************
{
int gdriver = DETECT, gmode, errorcode;

   initgraph(&gdriver,&gmode,""); /* initialize graphics and local variables */
   errorcode = graphresult();     /* read result of initialization */
   if (errorcode != grOk)         /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);                  /* terminate with an error code */
   }
}
```

CERC.H

```
#include <graphics.h>

class CERC
{
   int  x;                // variabile membru implicit PRIVATE
   int  y;
   int  r;
   int  c;
 public:                  // metode PUBLICE
         CERC();          // constr. fara param
         CERC(int x0,int y0,int r0=20,int c0=WHITE);  // constr. cu param
    void Muta(int dx,int dy);
    void Afiseaza();
    void Creste(int dr);
 private:
    void Sterge();        // metoda PRIVATA
};
```

CERC.CPP

```
#include "cerc.h"
#include <graphics.h>

CERC::CERC()                          //constructor fara parametrii
//**********
{
    x =   320;
    y =   240;
    r =    50;
    c =  BLUE;
}

CERC::CERC(int x0,int y0,int r0,int c0) //constructor cu parametrii
//************************************
{
    x = x0;
    y = y0;
    r = r0;
    c = c0;
}

void CERC::Sterge()                   //metoda sterge cerc de pe ecran
//*****************
{
    setcolor(BLACK);
    circle(x,y,r);                    //stergere prin desenare cu culoarea BLACK
}

void CERC::Afiseaza()                 //metoda afisare cerc pe ecran
//*******************
{
    setcolor(c);
    circle(x,y,r);                    //desenare cerc pe ecran
}
void CERC::Muta(int dx,int dy)        //metoda mutare cerc
//****************************
{
    Sterge();                         //stergere de pe pozitia curenta
    x += dx;                          //mutare prin modificare pozitie
    y += dy;
    Afiseaza();                       //afisare in noua pozitie
}

void CERC::Creste(int dr)             //metoda modifica diametrul cercului
//***********************
{
    Sterge();                         //stergere pozitia curenta
    r += dr;                          //redimensionare raza
    Afiseaza();                       //afisare cu noua dimensiune
}
```