Laborator 4 – Moștenirea

Tema 4.1

Analizați programul din fișierele `Ex4_VS.h`, `Ex4_VS.cpp`, `Cerc.h`, `Cerc.cpp`, `Grafica.h`, `Grafica.cpp` din anexa 4.

Tema 4.2

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

Considerații teoretice 4.2

Unul din avantajele programării orientate pe obiecte este faptul că se pot reutiliza și completa noi informații la codul deja existent. Această completare poate fi realizată chiar dacă nu se deține codul sursă al clasei de bază (de exemplu îmbunătățirea claselor din biblioteci). În contextul moștenirii clasa care se moștenește se numește clasă de bază iar clasa care moștenește se numește clasă derivată. Obiectul de bază (cel care se moștenește) este îmbunătățit prin derivare cu membrii noi păstrându-se membrii și 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 diferență că la `struct` membrii sunt implicit publici iar la `class` membrii sunt impliciți privați). `modificator_acces` reprezintă modul de moștenire al membrilor din clasa de bază și 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 moștenire (dacă lipsește clasa de bază).

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

Drepturile de acces ale membrilor din clasa derivată:

- În ceea ce privește membrii noi definiți aceștia vor avea drepturile de acces corespunzătoare modului de definire a lor.
- În ceea ce membrii moșteniți din clasa de bază acestora se vor stabili drepturile de acces în funcție de drepturile de acces avute în clasa de bază și în funcție de tipul moștenirii folosite astfel:

Drept acces avut în clasa bază	Tip moștenire	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 funcție de modificatorul de acces folosit în cazul moștenirii. La un membru definit `private` în clasa de bază nici o metodă nouă din clasa derivată nu va avea acces. Acel membru va exista în clasa derivată (clasa derivată va aloca spațiu 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 moștenite din clasa de bază. Prin moștenire nu se modifică tipul de acces la membrii pentru clasa de bază ci doar accesul la aceștia ca și membrii ai clasei derivate.

Deci din clasa de bază se vor moșteni toți membrii, cu (deocamdată) o excepție: constructorii și destructorii nu se moștenesc (ci se apelează).

În cazul în care în clasa derivată se definește un nou membru cu același nume și aceeași tip/prototip ca un membru deja existent în clasa de bază, acesta din urmă nu dispare și va putea fi accesat prin specificarea în față a clasei din care face parte folosind :: (operatorul de rezoluție).

Indicații 4.2

- ⇨ Se vor împărți membrii clasei `CERC` în membrii moșteniți din clasa de baza și în membrii adăugați de clasa derivată.
- ⇨ În clasa de bază membrii vor avea atributul public iar moștenirea va fi tot publică.

Tema 4.3

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

Considerații teoretice 4.3

În cazul obiectelor construite prin derivare (moștenire) constructorii și destructorii obiectelor de bază nu se moștenesc (adică nu sunt constructori ai clasei derivate) 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 și pe urmă cel al constructorului clasei derivate. Execuția 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 și se execută destructorul clasei derivate și pe urmă se apelează și se execută destructorul clasei de bază.

Indicații 4.3

- ⇨ Atenție la poziția cercurilor pe ecran. Explicați.

Tema 4.4

Să se apeleze explicit constructorii corespunzători din clasa de bază.

Considerații teoretice 4.4

In contextul moștenirii constructorii clasei de bază nu se moștenesc (ca și constructori ai clasei derivate) , aceștia urmând să se apeleze de fiecare dată de câte ori un obiect de tipul clasei derivate este instanțiat. Întotdeauna ordinea de execuție a constructorilor este dinspre clasa de bază spre

clasa derivată, prima dată executându-se constructorul clasei de bază și pe urmă cel al clasei derivate.

La rularea aplicației de la Tema 4.3 s-a obținut o imagine în care toate cercurile aveau același centru. Aceasta s-a datorat faptului că toți constructorii din clasa derivată (`CERC`) apelează implicit același constructor din clasa de bază (care stabilește poziția 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 dorește apelat). Specificarea explicită a constructorului din clasa de bază care se dorește 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.

Indicații 4.4

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

Tema 4.5

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

Considerații teoretice 4.5

Așa cum este descrisă moștenirea până în acest moment apare o dilemă. Dacă vrem să avem moștenire și acces la membrii moșteniți ar trebui ca în clasa de bază aceștia să fie definiți `public` pierzându-se astfel facilitățile oferite de încapsulare. Dacă vrem să păstrăm încapsularea în clasa de bază avem probleme la moștenire. Deci cu alte cuvinte membrii publici nu asigură încapsulare iar membrii privați nu asigură acces prin moștenire. 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 și de membrii claselor care moștenesc clasa respectivă. Membrii `protected` nu pot fi accesați din exteriorul clasei sau claselor care moștenesc clasa respectivă. Deci un membru definit `protected` este văzut ca un membru `public` în contextul moștenirii și ca un membru `private` pentru exteriorul claselor.

Tabelul cu modificatorii de acces în contextul moștenirii care include și tipul `protected` atât ca și tip de acces cât și ca tip de moștenire devine ca în tabelul următor.

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

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

Drept acces avut în clasa de bază	Tip moștenire	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];
```

Indicații 4.5

⇨ Atenție la atributul membrilor `r` și `c` din clasa `CERC` pentru a permite moștenirea ulterioara a acestei clase.

Tema 4.6

Să se modifice aplicația astfel încât la suprapunerea cercurilor afișarea acestora să rămână curată.

Considerații teoretice 4.6

Există două posibilități pentru a putea menține imaginea celorlalte cercuri intactă:

- prima posibilitate prevede salvarea parțială a ecranului în momentul în care se face desenarea iar în cazul mutării se restaurează partea salvată. Această posibilitate necesită multă memorie în funcție de rezoluție și adâncimea de culoare folosită.
- a doua posibilitate o reprezintă redesenarea completă a părții 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 soluție este folosită și de către interfața grafică din Windows care cere explicit aplicației să redeseneze partea afectată a interfeței.

Indicații 4.6

⇨ Pentru simplitate redesenați de câte ori e cazul toate cercurile.

Tema 4.7

Să se execute pas cu pas apelul constructorilor pentru a observa apelul și execuția acestora.

Tema 4.8

Să se introducă în aplicație facilitățile de inserare, respectiv ștergere, de noi cercuri la apăsarea

tastelor „F3″ și „F4”. Inserarea se va face întotdeauna în vector după ultimul cerc existent (fără a depăși o dimensiune maximă impusă, de exemplu 20) iar ștergerea va elimina ultimul cerc din vector (fără însă a șterge și cele 6 cercuri inițiale).

Indicaţii 4.8

- Redenumiți NR_CERCURI in MAX_CERCURI și introduceți o variabila `NrCercuri`.
- Atenție la codul aferent tastei `TAB`
- Nu uitați să ștergeți cercul de pe ecran (cu metoda `Sterge`) înainte de a îl distruge (cu `delete`).

Tema 4.9

Să se atașeze fiecărui cerc și un nume (șir de caractere). Acesta va fi dat ca parametru în constructor și va fi afișat în tot timpul funcționării în centrul cercului. Se va lua în considerare scenariul cel mai defavorabil în ceea ce privește durata de viață a șirului primit ca și parametru în constructor.

Considerații teoretice 4.9

Pentru afișarea unui text se va folosi funcția:

```
void writeText(char* text, int x, int y, int c)
```

din modulul Grafica.cpp/Grafica.h, care se va lua ca atare și apela.

Parametrii reprezintă:

`text`	șirul de text care va fi afișat
`x, y`	poziția pe ecran a textului
`c`	culoarea folosita la afișare

Indicații 4.9

- Atenție la alocarea memoriei pentru șirul de caractere în constructor și eliberarea memoriei în destructor.

Anexa 4

Ex4_VS.h

```
#include <windows.h>

// codurile pentru diferite taste
#define ESC   27
#define TAB    9
#define LEFT  75   // cu 0xE0 in fata
#define RIGHT 77   // cu 0xE0 in fata
#define UP    72   // cu 0xE0 in fata
#define DOWN  80   // cu 0xE0 in fata
#define F1    59   // cu 0 in fata
#define F2    60   // cu 0 in fata
#define F3    61   // cu 0 in fata
#define F4    62   // cu 0 in fata

#define NR_CERCURI 6
```

Ex4_VS.cpp

```
#include <conio.h>
#include "Ex4_VS.h"
#include "Grafica.h"
#include "Cerc.h"

CERC* pc[NR_CERCURI];
CERC cg1, cg2(100, 100, 25, BLUE);

int main()
{
int CercCurent = 0, gata = 0, k;

CERC cl1(500, 300, 75, RED), cl2(200, 100, 30);

    InitializeGraphicMode();

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

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

    while (!gata)
        switch (_getch())
        {
        case ESC:
            gata = 1;
            break;
        case TAB:
            CercCurent++;
            CercCurent %= NR_CERCURI;
            break;
        case 0xE0:            // pentru sageti se genereaza intai 0xE0
```

```
            switch (_getch()) //   apoi un cod specific
            {
            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;
            }
            break;
        case 0x00:            // pentru F1, F2 se genereaza intai 0x00
            switch (_getch()) //   apoi un cod specific
            {
            case F1: pc[CercCurent]->Creste(+10); break;
            case F2: pc[CercCurent]->Creste(-10); break;
            }
            break;
         }

    CloseGraphicMode();         //inchiderea modului grafic
    return 0;
}
```

Cerc.h

```
class CERC
{
      int x;
      int y;
      int r;
      int c;
      void Sterge();
public:
      CERC(int x0, int y0, int r0=20, int c0=WHITE);
      CERC();
      void Afiseaza();
      void Muta(int dx, int dr);
      void Creste(int dr);
};
```

Cerc.cpp

```
#include "Grafica.h"
#include "Cerc.h"

CERC::CERC()
//**********
// constructor implicit
{
    x = 320;
    y = 240;
    r = 50;
    c = WHITE;
}

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

```
    c = c0;
}

void CERC::Afiseaza()
//********************
// metoda de desenare a cercului
{
    circle(x, y, r, c);
}

void CERC::Sterge()
//******************
// metoda de stergere a cercului
{
    circle(x, y, r, BLACK);
}

void CERC::Muta(int dx, int dy)
//*****************************
// metoda care modifica pozitia unui cerc
{
    Sterge();   // stergere cerc

    x += dx;    // modificare pozitie x
    y += dy;    // modificare pozitie y

    Afiseaza(); // desenare in noua pozitie
}

void CERC::Creste(int dr)
//************************
// metoda care modifica raza unui cerc
{
    Sterge();   // stergere cerc

    r += dr;    // modificare raza

    Afiseaza(); // desenare in noua pozitie
}
```

Grafica.h

```
#include <windows.h>

// valorile pentru culori "clasice"
#define BLACK         (int)RGB(  0,  0,  0)
#define BLUE          (int)RGB(  0,  0,255)
#define GREEN         (int)RGB(  0,255,  0)
#define CYAN          (int)RGB(  0,255,255)
#define RED           (int)RGB(255,  0,  0)
#define MAGENTA       (int)RGB(255,  0,255)
#define BROWN         (int)RGB(128,  0,  0)
#define LIGHTGRAY     (int)RGB(255,255,204)
#define DARKGRAY      (int)RGB(  0,128,  0)
#define LIGHTBLUE     (int)RGB(  0,  0,128)
#define LIGHTGREEN    (int)RGB(153,204,  0)
#define LIGHTCYAN     (int)RGB(204,255,255)
#define LIGHTRED      (int)RGB(255,128,128)
#define LIGHTMAGENTA  (int)RGB(128,  0,128)
#define YELLOW        (int)RGB(255,255,  0)
#define WHITE         (int)RGB(255,255,255)
```

```
// prototipuri de functii considerate "de biblioteca"
void InitializeGraphicMode();
void CloseGraphicMode();
void circle(int x, int y, int r, int c);
void writeText(char* text, int x, int y, int c);
```

Grafica.cpp

```
#include <windows.h>
#include "Grafica.h"

//*********************************************************
// cod considerat "de biblioteca", luat ca atare
//*********************************************************

// pentru utilizare modul grafic
HWND console_handle;
HDC device_context;

void InitializeGraphicMode()
//**************************
// functie care face trecerea din modul text in modul grafic
{
    console_handle = GetConsoleWindow();
    device_context = GetDC(console_handle);
    Sleep(100);
}

void CloseGraphicMode()
//**********************
// functie care inchide modul grafic
{
    ReleaseDC(console_handle, device_context);
}

void circle(int x, int y, int r, int c)
//************************************
// functie care deseneaza un cerc (x,y,raza,culoare)
{
    HPEN pen = CreatePen(PS_SOLID, 1, (COLORREF)c);
    SelectObject(device_context, pen);
    SelectObject(device_context, GetStockObject(NULL_BRUSH));
    Ellipse(device_context, x - r, y - r, x + r, y + r);
    DeleteObject(pen);
}

void writeText(char* text, int x, int y, int c)
//*********************************************
// functie care afiseaza un text (text,x,y,culoare)
{
    SetBkColor(device_context, BLACK);
    SetTextColor(device_context, c);
    SetTextAlign(device_context, TA_CENTER);
    TextOut(device_context, x, y, text, strlen(text));
}
```