

Laborator 2 - Încapsularea

Tema 2.1

Să se analizeze programul `Ex2_VS.c`

Indicații 2.1

⇒ A nu se uita de fișierul `Ex2_VS.h`

Tema 2.2

Să se modifice funcțiile referitoare la cerc astfel încât parametrul `CERC` să fie transmis prin referință.

Considerații teoretice 2.2

La considerații teoretice 1.1 am prezentat faptul că funcțiile pot primi argumente în două feluri: apel prin valoare și apel prin pointer. Acest lucru este valabil pentru compilatorul de C. Compilatorul C++ vine cu încă o metodă de transmitere a argumentelor: apelul prin referință.

Primele două moduri de transmitere a argumentelor le-am introdus prin intermediul unui exemplu, o funcție care interschimbă două valori. Vom exemplifica acest nou mod de transmitere a argumentelor prin intermediul aceluiași exemplu.

```
#include <iostream.h>

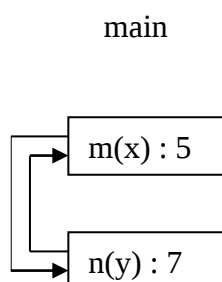
void schimb(int &x, int &y)
{
    int r;

    r = x;
    x = y;
    y = r;
}

void main()
{
    int m=5,n=7;

    cout<<m<<"  "<<n<<"  ";
    schimb(m,n);
    cout<<m<<"  "<<n;
}
```

În acest caz variabilele locale x , y vor referi variabilele m , n . De fapt variabila x va ocupa în memorie exact aceeași zonă cu variabila m , la fel se întâmplă cu variabilele y și n .



Variabilele x și y referind aceleași locații de memorie ca m și n , în momentul în care se face schimbarea de valori între x și y automat această schimbare se reflectă și pentru m și n

Indicații 2.2

⇒ Se va forța utilizarea compilatorului de C++ (prin modificarea extensiei fișierului sursa în *.CPP)

Tema 2.3

Să se definească tipul `CERC` în context C++ (fără operatorul `typedef`)

Considerații teoretice 2.3

La considerațiile teoretice 1.5 am prezentat modul de definire a unei structuri. Precizăm că acel mod de definire este specific compilatorului C (dar este totodată recunoscut și de compilatorul C++). În C++, o dată ce a fost declarată o structură, se pot declara variabile de acel tip folosind doar numele structurii, fără să mai trebuiască să fie precedată de cuvântul cheie *struct*. Motivul acestei diferențe între C și C++ este acela că în C un nume generic de structură nu este un nume de tip complet, în timp ce în C++ este.

În concluzie în C++ prin intermediul lui *struct* se vor defini noi tipuri de date. Forma generală va fi tot cea prezentată la punctul 1.5.

```
struct nume_structura
{
    tip1 nume_membru1;
    tip2 nume_membru2;
    ...
} variabile_structura;
```

Memoria ocupată de o structură este egală cu suma memoriei ocupate de fiecare membru al structurii:

$$\text{sizeof}(\text{struct}) = \sum(\text{sizeof}(\text{membrii}))$$

Ne vom folosi în continuare de următorul exemplu: o structură `RATIONAL`, pentru reprezentarea numerelor raționale (de forma m/n) și doua funcții: una de afișare și una care face adunarea unui număr rațional cu un întreg.

```
#include <iostream.h>

struct RATIONAL
{
    int numarator;
    int numitor;
};

void Afisez(RATIONAL x)
{
    cout<<x.numarator<<"/"<<x.numitor<<endl;
};

void Adun_int(RATIONAL &x, int y)
{
    x.numarator += y*x.numitor;
    Afisez(x);
}

void main()
{
    RATIONAL x;
```

```

x.numarator = 4;
x.numitor = 3;
Adun_int(x, 2);
}

```

Indicații 2.3

⇒ Atenție la sintaxa de definire (la numele tipului și la “;”).

Tema 2.4

Să se mute funcția `Muta` în interiorul structurii `CERC`.

Considerații teoretice 2.4

Într-o structură se pot defini nu numai câmpuri de date ci și funcții. Forma generală a definirii structurii va deveni:

```

struct nume_structura
{
    tip1 nume_membru1;
    tip2 nume_membru2;
    ...
    tip_returnat1 nume_functie1(lista_parametrii1);
    ...
};

```

Declararea funcțiilor definite într-o structură se modifică astfel:

```

tip_returnat1 nume_structura::nume_functie1(lista_parametrii)
{
    corp_functie;
};

```

Apelarea funcțiilor definite într-o structură se va face astfel:

```

nume_structura variabila;
variabila.nume_functie1(parametrii);

```

Funcțiile definite într-o structură mai poartă numele de metode.

Dimensiunea memoriei ocupată de structură nu se modifică datorită adăugării de metode în structură:

```
sizeof(struct) = Σ(sizeof(variabile_membru))
```

Revenim la exemplul prezentat la punctul anterior. Dorim să introducem în structura `RATIONAL` funcția `Adun_int`. Exemplul va deveni:

```

#include <iostream.h>

struct RATIONAL
{
    int numarator;
    int numitor;
    void Adun_int(RATIONAL &x, int y);
};

void Afisez(RATIONAL x)
{

```

```

    cout<<x.numarator<<"/"<<x.numitor<<endl;
};

void RATIONAL::Adun_int(RATIONAL &x, int y)
{
    x.numarator += y*x.numitor;
    Afisez(x);
}

void main()
{
    RATIONAL x;
    x.numarator = 4;
    x.numitor = 3;
    x.Adun_int(x,2);
}

```

Indicații 2.4

⇒ Apelul metodei Muta trebuie făcut folosind operatorul “.” legat de un cerc.

Tema 2.5

Să se renunțe la parametrul de tip CERC din cadrul metodei Muta.

Considerații teoretice 2.5

În exemplul cu clasa RATIONAL apelul funcției Adun_int era sub forma:

```
x.Adun_int(x,2);
```

Se observă că variabila `x` apare de două ori: la început fiind variabila care apelează funcția `Adun_int` și apoi ca parametru al funcției `Adun_int`. Apărând în două locuri ne punem întrbarea dacă nu este posibilă renunțarea la `x` ca și parametru? Pentru a putea face acest lucru vom fi nevoiți să ne folosim de pointerul `this`.

Atunci cand este apelată o funcție membru, i se pasează automat ca argument implicit pointerul `this`, care este un pointer către variabila care a generat apelarea (variabila care a invocat funcția).

Exemplul nostru va fi modificat astfel:

```

#include <iostream.h>

struct RATIONAL
{
    int numarator;
    int numitor;
    void Adun_int(int y);
};

void Afisez(RATIONAL x)
{
    cout<<x.numarator<<"/"<<x.numitor<<endl;
};

void RATIONAL::Adun_int(int y)
{
    this->numarator += y*this->numitor;
    Afisez(*this);
}

```

```
void main()
{
    RATIONAL x;
    x.numarator = 4;
    x.numitor = 3;
    x.Adun_int(2);
}
```

Indicații 2.5

- ⇒ Se va elimina parametrul de tip `CERC` și se va înlocui cu cercul indicat de pointerul `this`.
- ⇒ Atenție, `this` este pointer și uneori avem nevoie de cerc nu de adresa sa.

Tema 2.6

Să se introducă în structura `CERC` și celelalte funcții care țin de cerc în mod asemănător cu funcția `Muta`.

Indicații 2.6

- ⇒ Atenție la funcțiile `InitializeGraphicMode`, `CloseGraphicMode`, `circle`.
- ⇒ Se vor repeta temele 2.4 și 2.5 pentru fiecare din metodele respective.

Tema 2.7

Să se renunțe la scrierea lui `this` acolo unde este posibil.

Considerații teoretice 2.7

La membrii unei structuri se poate obține acces direct din cadrul unei funcții membru, fără nici un specificator de obiect sau structură.

`this->x` este echivalent cu `x`, unde `x` este un membru al structurii (atât câmp de date cât și metodă).

```
#include <iostream.h>

struct RATIONAL
{
    int numarator;
    int numitor;
    void Adun_int(int y);
};

void Afisez(RATIONAL x)
{
    cout<<x.numarator<<"/"<<x.numitor<<endl;
};

void RATIONAL::Adun_int(int y)
{
    numarator += y*numitor;
    Afisez(*this);
}

void main()
{
    RATIONAL x;
    x.numarator = 4;
    x.numitor = 3;
```

```
x.Adun_int(2);
}
```

Indicații 2.7

⇒ Deși se poate renunța la `this` nu uitați că de fapt **compilerul îl folosește**.

Tema 2.8

Să se găsească un membru (variabilă membru sau metodă) al structurii `CERC` care poate fi făcut privat și programul să funcționeze fără alte modificări.

Considerații teoretice 2.8

Într-o structură membrii pot fi de două tipuri: publici sau privați. Membrii publici sunt membrii care pot fi accesați atât de către metodele definite în structură cât și de funcțiile din afara structurii. Membrii privați sunt membrii care pot fi accesați doar de către metodele definite în structură. Forma generală pentru definirea structurii va deveni:

```
struct nume_structura
{
    public:
        lista_membrii_publici;
    private:
        lista_membrii_privati;
    public:
        lista_membrii_publici;
    ...
};
```

Implicit membrii unei structuri sunt publici.

Indicații 2.8

⇒ Membrul respectiv nu trebuie să fie folosit din exteriorul structurii.

Tema 2.9

Să se introducă o nouă metodă care să permită definirea tuturor datelor din structura `CERC` private.

Considerații teoretice 2.9

Pentru a declara variabilele din structură private este necesară o funcție publică care să facă inițializarea variabilelor respective. Devenind private variabilele nu vor mai putea fi modificate decât cu ajutorul metodelor definite în structură.

Exemplul cu structura `RATIONAL` ar putea fi rescris astfel:

```
#include <iostream.h>

struct RATIONAL
{
    private:
        int numator;
        int numitor;
    public:
        void Adun_int(int y);
        void Initializez(int x, int y);
};
```

```
void Afisez(RATIONAL x)
{
    cout<<x.numarator<<"/"<<x.numitor<<endl;
};

void RATIONAL::Initializez(int x, int y)
{
    numarator = x;
    numitor = y;
};

void RATIONAL::Adun_int(int y)
{
    numarator += y*numitor;
    Afisez(*this);
}

void main()
{
    RATIONAL x;
    x.Initializez(4,3);
    x.Adun_int(2);
}
```

Indicații 2.9

- ⇒ Se va folosi o metodă publică (cu patru parametri) care va inițializa toate variabilele respective.

Tema 2.10

Să se transforme structura CERC în clasa CERC.

Considerații teoretice 2.10

În C++ o structură este echivalentă cu o clasă, singura diferență fiind aceea că într-o structură membrii sunt implicit publici iar într-o clasă membrii sunt implicit privați. Forma generală a definirii unei clase este:

```
class nume_clasa
{
    public:
        lista_membrii_publici;
    private:
        lista_membrii_privati;
    public:
        lista_membrii_publici;
    ...
};
```

O variabilă definită de tipul unei clase este un obiect, iar în loc de definire vom spune că instanțiem un obiect de tipul clasei.

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

```
class RATIONAL
{
    private:
        int numarator;
        int numitor;
```

```
public:
    void Adun_int(int y);
    void Initializez(int x, int y);
};

void Afisez(RATIONAL x)
{
    cout<<x.numarator<<"/"<<x.numitor<<endl;
};

void RATIONAL::Initializez(int x, int y)
{
    numarator = x;
    numitor = y;
};

void RATIONAL::Adun_int(int y)
{
    numarator += y*numitor;
    Afisez(*this);
}

void main()
{
    RATIONAL x;           //instantierea obiectului x
    x.Initializez(4,3);
    x.Adun_int(2);
}
```

Indicații 2.10

⇒ Atenție la tipul implicit al membrilor.

Tema 2.11

Să se separe codul sursă aferent clasei `CERC` în fișiere separate (`cerc.cpp` și `cerc.h`) care să fie parte a proiectului.

Considerații teoretice 2.11

Prin directiva `#include` compilatorul știe că trebuie să citească nu numai fișierul sursă (care conține directiva) ci și fișierul care este specificat în directivă. Prin această directivă se introduc pentru citire și compilare fișiere antet pentru sistemul de fișiere al funcțiilor de bibliotecă.

Utilizatorii pot avea sursele scrise în mai multe fișiere, care vor trebui organizate în proiecte pentru ca compilatorul să știe de unde poate citi funcțiile. Proiectele sunt colecții de fișiere sursă (`.cpp`). Principala condiție într-un proiect este aceea că în fișierele sursă nu are voie să existe decât o singură funcție `main`.

Compilarea unui proiect se face prin compilarea fiecărui fișier component al proiectului.

Indicații 2.11

- ⇒ În proiect se includ doar fișierele `cpp`.
- ⇒ Fiecare fișier `cpp` este compilat separat, link-editarea este comună.

Anexa 2

Ex2_VS.c

```

#include <windows.h>
#include "Ex2_VS.h"

int culori[10] = { RED, WHITE, BLUE, YELLOW, GREEN,
                  BROWN, CYAN, MAGENTA, LIGHTRED, LIGHTGREEN };

CERC c[10]; // 10 cercuri

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

    InitializeGraphicMode();

    // initializarea cercurilor
    for (k = 0; k < 10; k++)
    {
        c[k].x = 50 * k + 100;
        c[k].y = 200;
        c[k].r = 10 * k + 25;
        c[k].c = culori[k];
        Afiseaza(&c[k]); // si afisarea lor
    }

    while (!gata)
    {
        switch (_getch())
        {
            case ESC:
                gata = 1;
                break;
            case TAB:
                CercCurent++;
                CercCurent %= 10;
                break;
            case 0xE0: // pentru sageti se genereaza intai 0xE0
                switch (_getch()) // apoi un cod specific
                {
                    case LEFT: Muta(&c[CercCurent], -10, 0); break;
                    case RIGHT: Muta(&c[CercCurent], 10, 0); break;
                    case UP: Muta(&c[CercCurent], 0, -10); break;
                    case DOWN: Muta(&c[CercCurent], 0, 10); break;
                }
                break;
            case 0x00: // pentru F1, F2 se genereaza intai 0x00
                switch (_getch()) // apoi un cod specific
                {
                    case F1: Creste(&c[CercCurent], +10); break;
                    case F2: Creste(&c[CercCurent], -10); break;
                }
                break;
        }
    }

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

void Afiseaza(CERC* ce)

```

```

//*****
// desenare a cercului
{
    circle(ce->x, ce->y, ce->r, ce->c);
}

void Sterge(CERC* ce)
//*****
// stergere a cercului
{
    circle(ce->x, ce->y, ce->r, BLACK);
}

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

    ce->x += dx;    // modificare pozitie x
    ce->y += dy;    // modificare pozitie y

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

void Creste(CERC* ce, int dr)
//*****
// functie care modifica raza unui cerc
{
    Sterge(ce);    // stergere cerc

    ce->r += dr;    // modificare raza

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

//*****
// 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);
}

```

Ex2_VS.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

// 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)

typedef struct
{
    int x;
    int y;
    int r;
    int c;
} CERC;

// prototip functii proprii
void Muta(CERC *ce, int dx, int dy);
void Creste(CERC* ce, int dr);
void Afiseaza(CERC* ce);
void Sterge(CERC* ce);

// prototipuri de functii considerate "de biblioteca"
void InitializeGraphicMode();
void CloseGraphicMode();
void circle(int, int, int, int);

```