Laborator 3 – Constructori și destructori

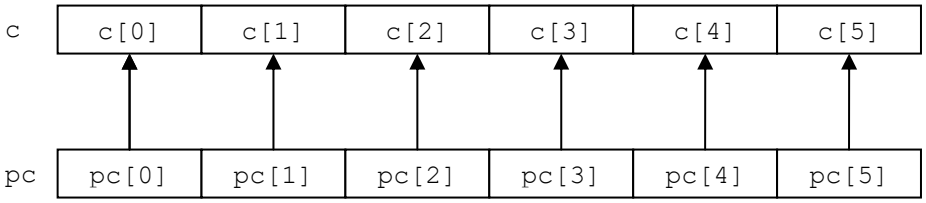
Tema 3.1

Analizați programul din fișierele `Ex3_VS.h`, `Ex3_VS.cpp`, `Cerc.h`, `Cerc.cpp`, `Grafica.h`, `Grafica.cpp` din anexa 3.

Tema 3.2

Să se adauge un vector de pointeri către clasa `CERC` de aceeași dimensiune cu vectorul existent. După inițializarea vectorului existent adresele cercurilor vor fi completate în vectorul de pointeri. În continuare programul va accesa structura de date numai prin intermediul vectorului de pointeri.

Considerații teoretice 3.2



Indicații 3.2

- ⇨ Inițializarea tabloului de pointeri se poate face de exemplu `pc[k]=&c[k]` într-o buclă

Tema 3.3

Să se modifice metoda Atribuie astfel încât ultimi 2 parametrii să aibă valorile implicite 100 respectiv "YELLOW". Se va apela apoi succesiv cu 4, 3 şi respectiv 2 parametrii şi se va observa de fiecare dată efectul la rulare.

Considerații teoretice 3.3

În C++ ne sunt permise funcții sau metode cu parametrii având valori implicite (valorile implicite pot fi doar constante). Parametrii impliciți se specifică la definirea (prototipul) funcției, iar dacă aceasta nu există atunci se specifică declararea (implementarea) funcției. Funcția (metoda) care are declarați parametrii impliciți poate fi apelată fie cu toți parametrii fie omițând parametrii (care vor lua valorile implicite). Declararea parametrilor impliciți se face de la dreapta spre stânga, în lista de parametrii, fără a omite vreunul.

```
#include <iostream.h>

long Calcul(int x, int y = 5, int z = 2);  prototipul functiei

void main()
{
 int m = 3, n = 7, o = 6;
 cout<<Calcul(m, n, o);           //va afisa 16 (3 + 7 + 6)
 cout<<Calcul(m, n);              //va afisa 12 (3 + 7 + 2)
 cout<<Calcul(m);                 //va afisa 10 (3 + 5 + 2)
}

long Calcul(int x, int y, int z)
```

```
{
  return x + y + z;
}
```

În exemplul de mai sus am definit o funcție cu trei parametrii dintre care doi (ultimii doi – vezi regula enunțată) au valori implicite. Funcția poate fi apelată în acest caz cu trei parametrii, cu doi parametrii (z ia valoarea implicită 2), cu un parametru (y ia valoarea implicită 5, iar z ia valoarea implicită 2).

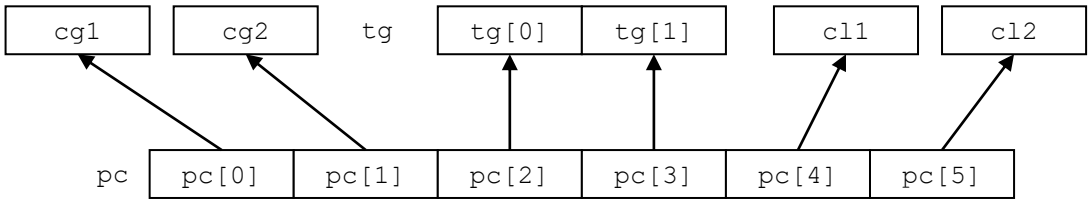
Indicații 3.3

- ⇨ Atenție, parametrii impliciți se specifică o singura dată, numai în interiorul clasei.

Tema 3.4

Să se înlocuiască cele 6 cercuri memorate sub forma vectorului "`c[6]`" cu 6 cercuri memorate în obiecte diferite astfel: 2 cercuri definite global (numite `cg1`, `cg2`), un tablou global de 2 cercuri (numit `tg[2]`) și 2 cercuri definite local (numite `cl1`, `cl2`).

Considerații teoretice 3.4



Indicații 3.5

- ⇨ În acest caz inițializarea tabloului de pointeri `pc[6]` trebuie făcută explicit element cu element.

Tema 3.5

Să se modifice metoda Atribuie astfel încât să devină constructor cu 4 parametrii dintre care ultimii 2 impliciți. Fiecare obiect individual definit va fi inițializat explicit în momentul definirii.

Considerații teoretice 3.5

Una dintre marile probleme ale programării este aceea a inițializării variabilelor (obiectelor). În cazul obiectelor dacă avem o metodă pentru inițializare, depinde de noi dacă o apelăm sau nu. Tocmai pentru a nu ne permite să uităm apelul unei metode de inițializare C++ introduce noțiunea de constructor.

Constructorul este o metodă a clasei care: are același nume cu al clasei căreia îi aparține, nu are tip returnat și la instanţierea unui obiect se apelează automat. Dacă programatorul nu declară explicit nici un constructor compilatorul va genera unul public, fără parametrii și care nu face nimic.

Pentru variabilele globale constructorul se apelează înainte de apelul lui `main()`, pentru variabilele locale se apelează la intrarea în funcția respectivă, iar în cazul variabilelor alocate dinamic se apelează doar in momentul alocării memoriei pentru obiect (new).

Rolul principal al constructorului este acela de a inițializa obiectul pentru care a fost apelat, aceasta însemnând inițializarea tuturor variabilelor membru cu valori determinate, aducând astfel obiectul într-o stare determinată.

Pentru apelarea constructorului cu parametrii la instanţierea obiectului vor trebui furnizați parametrii în următoarea formă:

```
nume_clasa obiect(lista_parametrii);
```

Indicații 3.5

- ⇨ În această fază programul nu va funcționa deoarece nu există constructor implicit care să poată fi apelat pentru elementele din tabloul `tg[2]`. Problema se va rezolva în tema următoare.

Tema 3.6

Să se adauge si un constructor fără parametrii care să inițializeze membrii obiectului cu valorile `320, 240, 50, WHITE`.

Considerații teoretice 3.6

Supraîncărcarea numelui funcțiilor reprezintă proprietatea compilatorului de a permite să existe mai multe funcții cu același nume dar diferind prin numărul sau prin tipul parametrilor. Deci pentru a face distincția între funcțiile cu același nume compilatorul se folosește doar de numărul și tipul parametrilor, nu și de tipul returnat de funcție.

În cazul în care avem definite mai multe funcții cu același nume, iar una sau unele dintre aceste funcții au și parametrii impliciți trebuie avut de grijă ca atunci când se apelează funcția respectivă compilatorul să poată decide pe care funcție s-o folosească.

Vom prezenta în continuare exemplul cu clasa `RATIONAL` prezentat în lucrarea anterioară.

```
#include <iostream.h>

class RATIONAL
 {
  private:
    int numarator;
    int numitor;
  public:
    RATIONAL(){numarator = 1; numitor = 1};
    RATIONAL(int x, int y = 1);
    void Adun_int(int y);
    void Initializez(int x, int y);
 };

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

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

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

void main()
{
 RATIONAL a, b(3), c(7,4); //instantierea obiectelor cu apelarea constructorilor
                           //sub diverse forme
 a.Adun_int(2);
}
```

Indicaţii 3.6

- Atenție la numărul de cercuri afișate.
- Dacă există constructor definit nu se mai generează automat un constructor implicit.

Tema 3.7

Să se includă în tabloul `pc[]` și două cercuri alocate dinamic inițializate fiecare în parte cu cate unul din cei doi constructori.

Considerații teoretice 3.7

În C alocarea dinamică a memoriei se realizează cu ajutorul funcției `malloc()`, iar dealocarea cu ajutorul funcției `free()`. În C++ există un sistem propriu alternativ bazat pe operatorii: `new` și `delete`.

Ca și `malloc()`, `new` alocă memorie din zona heap, dar spre deosebire de `malloc()`, el alocă automat memorie suficientă pentru a păstra obiectele de tipul specificat (pentru `malloc` este necesară specificarea mărimii memoriei ce urmează a fi alocate, cu ajutorul lui `sizeof`). În acelaşi timp `new` returnează automat un pointer de tipul specificat, pe când pentru `malloc` trebuia să folosim explicit un modelator de tip.

Operatorul `delete` trebuie folosit doar cu un pointer valid, alocat anterior prin utilizarea lui `new`. Formele generale pentru `new` şi `delete` sunt:

```
nume_clasa *variabila;       //declararea unui pointer de tip obiect
variabila = new nume_clasa;  //alocare de memorie cu new
delete variabila;            //eliberare de memorie cu delete
```

În momentul alocării memoriei, cu operatorul `new`, se apelează automat constructorul, în cazul de mai sus cel fără parametrii. Pentru apelarea constructorului cu parametrii operatorul new trebuie folosit astfel:

```
Variabila = new nume_clasa(lista_parametrii);
```

Destructorul este metoda opusă constructorului şi care se apelează la eliberarea memoriei prin operatorul `delete` sau la ieşirea din program. Destructorul are numele identic cu al clasei, dar precedat de semnul „~”, la fel ca si constructorul nu returnează nimic însă, spre deosebire de constructor, acesta nu are parametrii. Destructorul este folosit mai ales pentru a elibera memoria atunci când ea a fost alocată cu constructorul. Destructorul se apelează în ordine inversă constructorului, mai întâi pentru variabilele locale și apoi pentru cele globale, iar pentru variabilele alocate dinamic se apelează in momentul eliberării memoriei cu `delete`.

Indicaţii 3.7

- ⇨ Nu uitați eliberarea celor două obiecte alocate dinamic.

Tema 3.8

Să se execute programul pas cu pas pentru a urmări apelul celor 8 constructori.

Indicații 3.8

- ⇨ Constructorii obiectelor globale se apelează înainte de main iar, în lipsa break-point-urilor, execuția pas cu pas începe direct din main.

Anexa 3

Ex3_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
```

Ex3_VS.cpp

```
#include <conio.h>
#include "Ex3_VS.h"
#include "Grafica.h"
#include "Cerc.h"

int culori[6] = {RED, WHITE, BLUE, YELLOW, GREEN, BROWN};

CERC c[NR_CERCURI];

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

    InitializeGraphicMode();

    for (k = 0; k < 6; k++)    // initializari
    {
        c[k].Atribuie(50*k+100, 200, 10*k+25, culori[k]);
        c[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:  c[CercCurent].Muta(-10,  0); break;
            case RIGHT: c[CercCurent].Muta( 10,  0); break;
            case UP:    c[CercCurent].Muta(  0,-10); break;
```

```
            case DOWN:  c[CercCurent].Muta(  0, 10); break;
            }
            break;
        case 0x00:            // pentru F1, F2 se genereaza intai 0x00
            switch (_getch()) //   apoi un cod specific
            {
            case F1: c[CercCurent].Creste(+10); break;
            case F2: c[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:
      void Atribuie(int x0, int y0, int r0, int c0);
      void Afiseaza();
      void Muta(int dx, int dr);
      void Creste(int dr);
};
```

Cerc.cpp

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

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

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

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

    r += dr;    // modificare raza

    Afiseaza(); // desenare in noua pozitie
}

void CERC::Atribuie(int x0, int y0, int r0, int c0)
{
    x = x0;
    y = y0;
    r = r0;
    c = c0;
}
```

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, int, int, int);
```

Grafica.cpp

```
#include <windows.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);
}
```