Laborator 3 – Constructori şi destructori

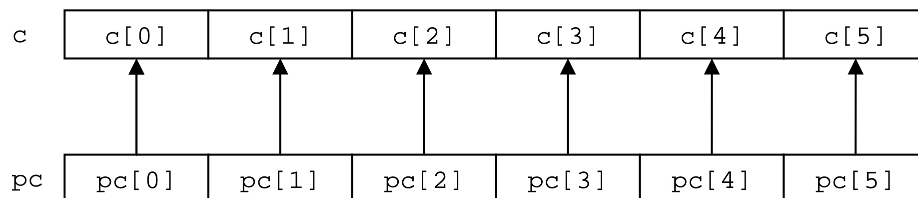
Tema 3.1

Analizaţi programul din fişierele `EX3.H`, `EX3.CPP`, `CERC.H`, `CERC.CPP` din anexa 3.

Tema 3.2

Să se adauge un vector de pointeri către clasa `CERC` de aceeaşi dimensiune cu vectorul existent. După iniţializarea 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.

Consideraţii teoretice 3.2



Indicaţii 3.2

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

Tema 3.3

Sa 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.

Consideraţii teoretice 3.3

În C++ ne sunt permise funcţii sau metode cu parametrii având valori implicite (valorile implicite pot fi doar constante). Parametrii impliciţi se specifică la definirea (prototipul) funcţiei, iar dacă aceasta nu există atunci se specifică declararea (implementarea) funcţiei. Funcţia (metoda) care are declaraţi parametrii impliciţi poate fi apelată fie cu toţi parametrii fie omiţând parametrii (care vor lua valorile implicite). Declararea parametrilor impliciţi 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 funcţie cu trei parametrii dintre care doi (ultimii doi – vezi regula enunţată) au valori implicite. Funcţia 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).

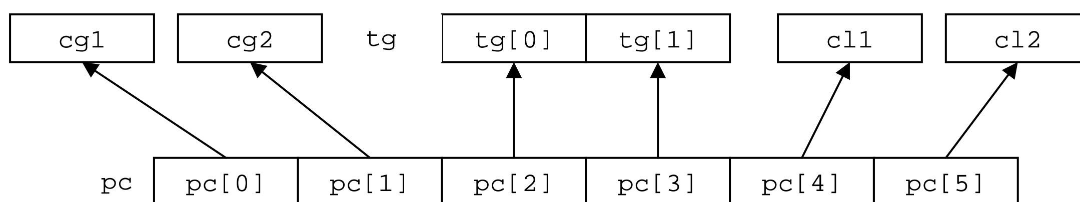
Indicaţii 3.3

- Atenţie, parametrii impliciţi se specifica o singura data, 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]`) şi 2 cercuri definite local (numite `cl1`, `cl2`).

Consideraţii teoretice 3.4



Indicaţii 3.5

- În acest caz iniţializarea 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 impliciţi. Fiecare obiect individual definit va fi iniţializat explicit în momentul definirii.

Consideraţii teoretice 3.5

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

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

Pentru variabilele globale constructorul se apelează înainte de apelul lui `main()`, pentru variabilele locale se apelează la intrarea în funcţia 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 iniţializa obiectul pentru care a fost apelat, aceasta însemnând iniţializarea tuturor variabilelor membru cu valori determinate, aducând astfel obiectul într-o stare determinată.

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

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

Indicaţii 3.5

- În această fază programul nu va funcţiona 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

Sa se adauge si un constructor fără parametrii care sa iniţializeze membrii obiectului cu valorile `320, 240, 50, WHITE`.

Consideraţii teoretice 3.6

Supraîncărcarea numelui funcţiilor reprezintă proprietatea compilatorului de a permite să existe mai multe funcţii cu acelaşi nume dar diferind prin numărul sau prin tipul parametrilor. Deci pentru a face distincţia între funcţiile cu acelaşi nume compilatorul se foloseşte doar de numărul şi tipul parametrilor, nu şi de tipul returnat de funcţie.

În cazul în care avem definite mai multe funcţii cu acelaşi nume, iar una sau unele dintre aceste funcţii au şi parametrii impliciţi trebuie avut de grijă ca atunci când se apelează funcţia respectivă compilatorul să poată decide care funcţie 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

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

Tema 3.7

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

Consideraţii teoretice 3.7

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

Ca şi `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 şi apoi pentru cele globale, iar pentru variabilele alocate dinamic se apelează in momentul eliberării memoriei cu `delete`.

Indicaţii 3.7

- Nu uitaţi eliberarea celor două obiecte alocate dinamic.

Tema 3.8

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

Consideraţii teoretice 3.8

Pentru depanarea aplicaţiilor mediile de programare C şi C++ pun la dispoziţie rularea pas cu pas şi posibilitatea de afişare a valorilor diferitelor variabile, la un moment dat, oriunde în program. Toate aceste funcții sunt disponibile în meniul *Debug*.

Pentru rularea pas cu pas avem la dispoziţie opţiunea *Debug->Step into* (sau mai rapid cu *F11*), care face executarea programului instrucţiune cu instrucţiune, în momentul întâlnirii unei funcţii va intra în funcţie şi se va putea executa instrucţiune cu instrucţiune. O altă opţiune este aceea de executare a funcţiei fără însă a mai intra în ea, opţiune care se realizează prin *Debug->Step over* (sau cu tasta *F10*). În rest această opţiune se comportă la fel ca şi cea realizată cu *F11*. Pentru a rula programul până la un anumit punct, şi doar de acolo să pornească rularea pas cu pas, se foloseşte opţiunea *Debug->Run to cursor* (sau *Ctrl+F10*).

O altă opţiune a mediului este definirea de puncte de întrerupere (`breakpoint`), programul rulând până întâlneşte o instrucţiune pe care s-a definit un punct de întrerupere. Definirea punctului de întrerupere se face prin opţiunea *Debug->Toggle breakpoint* (sau cu combinaţia de taste *F9*). Pentru a rula aplicația până la primul punct de întrerupere se folosește opțiunea *Debug->Go* sau mai rapid cu *F5*.

Pentru vizualizarea valorilor diverselor variabile, în cursul rulării pas cu pas, se foloseşte opţiunea *Debug->QuickWatch..* (sau combinaţia *Shift+F9*).

Indicaţii 3.8

- Constructorii obiectelor globale se apelează înainte de main iar, în lipsa break-pointurilor execuţia pas cu pas începe direct din main.

Anexa 3

Ex3.h

```
#define TAB         9
#define ESC        27

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

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

#define NR_CERCURI 6

// prototipuri de functii

void OurInitGraph(void);
```

Ex3.cpp

```
#include <graphics.h>

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

#include "ex3.h"
#include "cerc.h"

// variabile Globale

CERC c[NR_CERCURI];

void main()
//*********
{
// variabile locale

int CercCurent=0,gata=0,k;

// cod

   OurInitGraph();

   for(k=0;k<NR_CERCURI;k++)                    //initializari
      {
      c[k].Atribuie(30*k+100,200,5*k+25,k+1);
      c[k].Afiseaza();
      }

   while(!gata)
     switch(getch())
       {
       case ESC:
```

```
          gata=1;
          break;
        case TAB:
          CercCurent++;
          CercCurent%=NR_CERCURI;
          break;
        case UNU:    c[CercCurent].Creste(  +10   ); break;
        case DOI:    c[CercCurent].Creste(  -10   ); break;
        case 0:
          switch(getch())
            {
              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;
              default:    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

```
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 dy);
    void Creste(int dr);
 };
```

Cerc.cpp

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

void CERC::Sterge()
```

```
//*****************
{
  setcolor(BLACK);
  circle(x,y,r);
}

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

void CERC::Muta(int dx,int dy)
//*****************************
{
  Sterge();              //stergere

  x += dx;               //mutare
  y += dy;

  Afiseaza();            //afisare in noua pozitie
}

void CERC::Creste(int dr)
//**********************
{
  Sterge();              //stergere

  r += dr;               //redimensionare

  Afiseaza();            //afisare cu noua dimensiune
}

void CERC::Atribuie(int x0,int y0,int r0,int c0)
//*********************************************
{
  x = x0;               // atribuiri spre variabilele private
  y = y0;
  r = r0;
  c = c0;
}
```