Laborator 1 - Structurarea datelor

Tema 1.1

Să se analizeze programul Ex1_VS.c din Anexa 1.

Considerații teoretice 1.1

Compilatoarele de C++ pot compila și surse C însă, când se dorește compilarea de surse în C, fișierele trebuie să aibă extensia .C (nu .CPP), deoarece limbajul se decide implicit după extensie.

În cazul fișierelor antet (cele cu extensia .H) definite de utilizator, pentru a putea fi căutate în directorul curent, includerea se va face în forma următoare:

```
#include "nume_fisier.h"
```

Modul grafic

Un program poate folosi monitorul în două moduri de lucru:

- *Modul de lucru text* – este modul în care ecranul este împărțit într-o matrice de "căsuțe" de dimensiune 80 x 25 (uzual). În fiecare căsuță se poate afișa un caracter.
- *Modul de lucru grafic* – este modul de lucru în care ecranul este împărțit într-o matrice de puncte (pixeli) de dimensiune 800x600, 1024x800, etc. În acest mod de lucru pe ecran pot fi afișate atât caractere (de această dată caracterele pot fi scrise de dimensiuni diverse și cu fonturi diferite), cât și diferite obiecte grafice (puncte, linii, cercuri, etc.).

În ambele cazuri ecranul are coordonata 0,0 în colțul din stânga sus.

Câteva funcții folosite pentru lucrul în modul grafic:

- `void InitializeGraphicMode(void)` - funcție care inițializează modul grafic.
- `void CloseGraphicMode()` - funcție care închide modul grafic.
- `void circle(int, int, int, int)` - funcție care desenează un cerc de centru `x` și `y`, de raza `r` și culoare `c`.

Deocamdată vom considera aceste funcții ca niște funcții de bibliotecă, le vom lua și folosi ca atare.

Variabile

Variabilele se declară, de obicei, în trei locuri: în interiorul funcțiilor (variabile locale), în cadrul definiției parametrilor funcției (parametri formali) și în afara oricărei funcții (variabile globale). Variabilele locale sunt declarate în interiorul unei funcții, ele fiind accesibile doar instrucțiunilor care se află în interiorul blocului în care sunt declarate variabilele. Parametri formali ai funcției se comportă ca orice altă variabilă locală din acea funcție. Variabilele globale sunt cunoscute în întreg programul și pot fi utilizate de către orice zonă a codului.

Funcții

Forma generală a definirii unei funcții:

```
specificator_de_tip nume_functie(lista_de_parametrii)
{
corpul_functiei
}
```

`specificatorul_de_tip` specifică tipul de date pe care îl returnează funcția. Dacă nu este specificat nici un tip, compilatorul presupune că funcția returnează un rezultat de tip `int`. `lista_de_parametrii` este o listă separată prin virgule de tipul şi numele parametrilor formali. Parametrii formali primesc valorile argumentelor atunci când este apelată funcția, compilatorul verificând în momentul apelului corespondența între parametrii actuali și cei formali.

În unele cazuri este preferat modelul care implică folosirea prototipului funcției (declararea ei), definirea funcției urmând a fi făcută ulterior. Forma generală a prototipului unei funcții:

```
specificator_de_tip nume_functie(lista_de_parametrii);
```

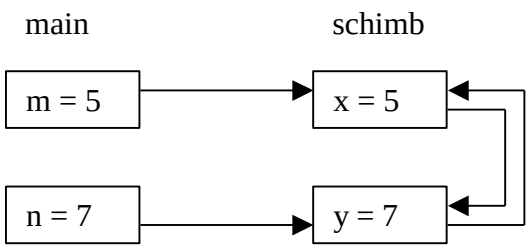
În general funcțiile pot primi argumente în două feluri: apel prin valoare şi apel prin pointer.

Să luăm următorul exemplu: dorim să facem o funcție care să interschimbe două valori. O primă variantă (scrisă rapid) ar arăta în felul următor:

```
#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;
}
```

Vom avea însă următoarea surpriză programul va afișa: 5 7 5 7. Ce s-a întâmplat de nu s-au schimbat totuși valorile între ele?

În funcția `main` am definit cele două variabile `m`, `n` pentru care se alocă memorie. În momentul în care apelăm funcția `schimb(m,n)`, această funcție folosește variabilele locale `x`, `y`, `r` pentru care se alocă memorie (pe stivă) atâta timp cât va rula această funcție. Variabila `x` va primi valoarea lui `m`, iar variabila `y` va primi valoarea lui `n`. Apoi se face schimbarea de valori între variabilele locale `x`, `y`, fără ca valorile din zona de memorie alocată pentru variabilele `m`, `n` să fie afectate. La revenirea din funcția `schimb` cele două variabile `m`, `n` vor avea tot aceleași valori ca mai înainte.



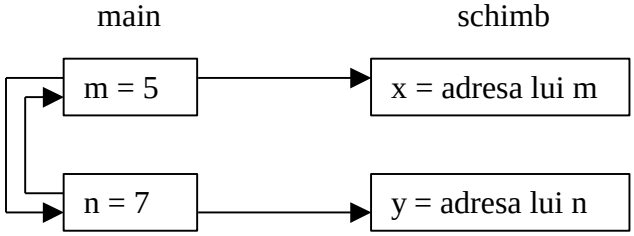
Schimbarea valorilor se face pentru variabilele locale x, y; variabilele m, n din main rămân neschimbate

Acesta a fost un exemplu de *apel prin valoare* (în urma căruia nu am reuşit să schimbăm valorile variabilelor folosite în apelarea funcției).

Pentru ca totuși să reușim ce ne-am propus vom folosi *apel prin pointer*. Exemplul nostru va arăta astfel:

```
#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;
}
```

Scris în acest fel programul va face ceea ce ne dorim. De fapt ce s-a modificat? Funcția `schimb` are acum ca și parametrii doi pointeri în care se vor memora adresele variabilelor `m`, `n`. De această dată în funcția `schimb` se vor schimba valorile de la adresele de memorie date de `x`, `y`, respectiv se vor schimba valorile din zona de memorie unde sunt memorate variabilele `m`, `n`.



Schimbarea valorilor se face pentru locaţiile de memorie a căror adrese sunt în x şi în y (respectiv locaţiile care conţin varibilele m şi n)

Lucrul cu tastatura

Pentru a afla codul generat la apăsarea unei taste se folosește funcția `getch()`. Aceasta așteaptă apăsarea unei taste și întoarce:

- pentru tastele normale codul ASCII al tastei apăsate.
- pentru tastele speciale valoarea 0 sau 0xE0 urmate, la o nouă apelare a funcției `getch()`, de un cod al tastei apăsate. Taste speciale sunt de exemplu: săgețile, F1 – F12, tastele INS, DEL, PAGE UP, PAGE DOWN, HOME și END.

Indicații 1.1

- ⇨ Atenție, nu uitați de fișierul Ex1_VS.h
- ⇨ Atenție la codul asociat TAB
- ⇨ Atenție la restricțiile compilatorului de C.

Tema 1.2

Să se modifice programul așa încât să gestioneze 4 (patru) cercuri.

Indicații 1.2

- Se vor declara variabilele x4, y4, r4, c4 și se va modifica corespunzător programul.
- Atenție la codul asociat TAB.

Tema 1.3

Să se modifice programul așa încât să gestioneze 10 (zece) cercuri.

Considerații teoretice 1.3

Vectori

Un vector (o matrice cu o singură dimensiune) este o colecție de variabile de același tip, apelate cu același nume. Accesul la un anumit element al vectorului se face cu ajutorul unui indice. Un vector ocupă locații de memorie contigue (consecutive). Forma generală pentru declararea unui vector este:

```
tip nume_variab[marime];
```

Vectorii trebuise declarați explicit astfel încât compilatorul să aloce spațiu în memorie pentru ei. `tip` declară tipul de bază al vectorului, care este tipul fiecărui element al său. `marime` indică numărul de elemente al vectorului. Accesarea unui element al vectorului se va face astfel:

```
nume_variab[indice]
```

unde `indice` va putea lua valori de la 0 la `marime-1`.

Culori

Culorile pot fi privite ca numere întregi (de 32 biți) care combină pe câte 8 biți valorile pentru R, G și B (fiecare pe 8 biți). În fișierul Ex1_VS.h se găsesc valorile pentru câteva culori uzuale:

```
BLACK           BLUE            GREEN           CYAN
RED             MAGENTA         BROWN           LIGHTGRAY
DARKGRAY        LIGHTBLUE       LIGHTGREEN      LIGHTCYAN
LIGHTRED        LIGHTMAGENTA    YELLOW          WHITE
```

Indicaţii 1.3

- Se vor folosi patru vectori, câte unul pentru fiecare variabilă `x`, `y`, `r` respectiv `c`.
- La inițializare se recomandă folosirea unei bucle for pentru a inițializa fiecare cerc, iar valorile atribuite să fie dependente de contorul buclei (de exemplu `x[k]=200+10*k`).
- Pentru culorile cercurilor se recomandă definirea unui vector global inițializat explicit cu culorile dorite, iar culoarea cercului `k` se va lua din acest vector prin indexare cu `k`.
- Datorită ordinii realizate în date se poate renunța la `switch(CercCurent)` și înlocui cu o singură linie cu apel de `Muta` și indexare (după variabila `CercCurent`) în tablourile corespunzătoare.
- Nu se va modifica funcția `Muta`.

Tema 1.4

Să se modifice programul astfel încât să folosească o matrice care să memoreze datele cercurilor.

Consideraţii teoretice 1.4

Matrici

Compilatorul de C admite declararea de matrici multidimensionale. Cea mai simplă formă de matrice multidimensională este cea bidimensională, care poate fi privită ca un vector de vectori. Definirea unei matrici se va face în felul următor:

```
tip nume_matrice[marime1][marime2];
```

`tip` reprezintă tipul de date pentru elementele matricii. Matricele bidimensionale sunt stocate în forma rând-coloană, unde primul indice (`marime1`) indică rândul iar al doilea (`marime2`) indică coloana. Accesul la un element al matricii se face sub forma:

```
nume_matrice[indice1][indice2];
```

unde `indice1` va lua valori între `0` și `marime1`, iar `indice2` va lua valori între `0` și `marime2`. În ceea ce privește stocarea efectivă în memorie a matricii, `indice2` se modifică mai repede decât `indice1` atunci când sunt parcurse succesiv elementele matricii.

Indicații 1.4

- Se va folosi o singură matrice, cu 4 coloane, corespunzătoare variabilelor `x`, `y`, `r` și `c`.
- Nu se va modifica funcția `Muta`.

Tema 1.5

Să se definească o structură CERC, aferentă unui cerc, și să se modifice corespunzător programul.

Considerații teoretice 1.5

Structuri

O structură este un grup de variabile unite sub același nume, ce pun la dispoziție un mod convenabil de păstrare a informațiilor legate între ele. O declarare de structură formează un șablon care poate fi folosit pentru a crea variabile de acel tip. Membrii (elementele sau câmpurile) unei structuri sunt variabilele care fac parte din structură. Forma generală a definirii unei structuri este:

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

unde `nume_structura` este numele structurii și `variabile_structura` este lista de variabile de tipul structurii. Cele două pot fi omise, dar nu ambele simultan. Atenție definiția se termină cu punct și virgulă, deoarece este o instrucțiune. Având definită o structură, definirea unei variabile de tipul structurii se va face astfel:

```
struct nume_structura nume_variabila;
```

Forma generală de acces la un membru al structurii este:

```
nume_variabila.nume_membru
```

În cazul unei variabile pointer la o structură accesul la un membru al structurii se va face astfel:

```
nume_pointer->nume_membru
```

Limbajul C permite definirea explicită a noi nume de tipuri de date prin utilizarea cuvântului cheie `typedef`. Forma generală de folosire `typedef` este:

```
typedef tip_vechi tip_nou;
```

Dacă `tip_vechi` este o structură, noul tip de date definit se poate folosi fără a fi nevoie să se specifice de fiecare dată cuvântul `struct` la definirea variabilelor.

Indicaţii 1.5

- ⇨ Se va defini un vector de elemente de tip `CERC`.
- ⇨ Se va modifica funcția `Muta` astfel încât să primească doar un parametru referitor la `CERC` și deplasările (în total 3 parametrii).
- ⇨ Atenție la declararea (prototipul), definirea și apelul corespunzător al funcției `Muta`.

Tema 1.6

Să se renunțe la variabilele `dx` și `dy` din `main` fără a se modifica însă funcția `Muta`.

Indicații 1.6

- ⇨ Se vor lua în considerare apeluri ale funcției `Muta` ori de câte ori este cazul.

Tema 1.7

Să se introducă două funcții `Sterge` și `Afiseaza` care să realizeze ștergerea și respectiv afișarea unui cerc primind ca și parametru cercul. Acestea vor fi singurele funcții care mai conțin funcția `circle`.

Tema 1.8

Să se adauge o facilitate suplimentară de modificat raza cercului curent: la apăsarea F1 raza crește cu 10 iar la apăsarea F2 raza scade cu 10.

Considerații teoretice 1.8

Tastele F1 și F2 sunt taste speciale care transmit câte 2 coduri:

- F1: 0 urmat de 59
- F2: 0 urmat de 60

Indicaţii 1.8

- ⇨ Implementarea va fi asemănătoare funcției `Muta`, iar numele funcției va fi `Creste`.

Anexa 1

Ex1_VS.c

```
#include <windows.h>
#include "Ex1_VS.h" // includerea fisierului antet propriu "Ex1_VS.h"
                    // care trebuie existe in directorul curent.

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

// prototip functie proprie
void Muta(int* x, int* y, int r, int c, int dx, int dy);

// variabile globale, pentru un cerc vom avea urmatoarele date:
// x, y = coordonatele centrului cercului
// r    = raza cercului
// c    = culoarea cercului
// se vor defini 3 cercuri

//variabilele pentru cele 3 cercuri
int x1,y1,r1,c1, x2,y2,r2,c2, x3,y3,r3,c3;

int main()
{
// variabile locale

int CercCurent = 0;
    // variabila care indica cercul curent, poate lua 3 valori:
    // 0 - pentru cercul x1, y1, r1, c1
    // 1 - pentru cercul x2, y2, r2, c2
    // 2 - pentru cercul x3, y3, r3, c3

int gata = 0;
int dx, dy;

    // cod
    InitializeGraphicMode();     // se face trecerea in modul grafic

    // initializarea variabilelor pt. cerc 1
    x1 = 100; y1 = 200; r1 = 25; c1 = YELLOW;
    circle(x1, y1, r1, c1);     // se deseneaza cercul 1

    // initializarea variabilelor pt. cerc 2
    x2 = 300; y2 = 200; r2 = 50; c2 = RED;
    circle(x2, y2, r2, c2);     // se deseneaza cercul 2

    // initializarea variabilelor pt. cerc 3
    x3 = 500;y3 = 200;r3 = 100; c3 = BLUE;
    circle(x3, y3, r3, c3);     // se deseneaza cercul 3

    // se intra intr-o bucla in care se ramane atata timp cat gata=0
    while (!gata)
        switch (_getch())     // se asteapta apasarea unei taste
        {
        case ESC:             // daca s-a apasat ESC se va iesi din bucla
            gata = 1;         // prin valoarea 1 pe care o primeste gata
            break;
        case TAB:             // daca s-a apasat TAB
```

```
            CercCurent++;     // trecem la cercul urmator
            CercCurent %= 3;  //   dupa cercul 2 urmeaza cercul 0
            break;
        case 0xE0:            // pentru sageti se genereaza intai 0xE0
            switch (_getch()) // apoi un cod specific
            {
            case LEFT:  dx = -10;dy =   0;break;    // 10 pixeli la stanga
            case RIGHT: dx =  10;dy =   0;break;    // 10 pixeli la dreapta
            case UP:    dx =   0;dy = -10;break;    // 10 pixeli in sus
            case DOWN:  dx =   0;dy =  10;break;    // 10 pixeli in jos
            default:    dx =   0;dy =   0;break;
            }
            // in functie de sageata apasata s-au dat valori pt. dx si dy

            // abia acum se muta cu dx si dy cercul curent
            switch (CercCurent)
            {
            case 0:                                 // daca CercCurent=0
                Muta(&x1, &y1, r1, c1, dx, dy); //  mutarea cercului 1
                break;
            case 1:                                 // daca CercCurent=1
                Muta(&x2, &y2, r2, c2, dx, dy); //  mutarea cercului 2
                break;
            case 2:                                 // daca CercCurent=2
                Muta(&x3, &y3, r3, c3, dx, dy); //  mutarea cercului 3
                break;
            }
            break;
         }

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

void Muta(int* x, int* y, int r, int c, int dx, int dy)
//*****************************************************
// functie care muta un cerc cu deplasamentele dx si dy
{
    // sterge cercul din pozitia veche, prin scrierea
    // unui cerc de culoarea fondului (negru) peste el
    circle(*x, *y, r, BLACK);

    // sunt modificate coordonatele centrului cercului
    *x += dx;           // mutare pe orizontala
    *y += dy;           // mutare pe verticala

    // desenare a cercului in noua pozitie
    circle(*x, *y, r, c);
}


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

Ex1_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)
```