LIMBAJE DE PROGRAMARE

Lucrarea 1

UTILIZAREA MEDIULUI TURBO C

I. NOTIUNI TEORETICE

1. Introducere

In anul 1972 apare limbajul C, cu destinatie universala, avand ca autori pe Dennis M. Ritchie si Brian W. Kernigham de la Bell Laboratories. Limbajul C a fost proiectat in ideea de a asigura implementarea portabila a sistemului de operare UNIX. Un rezutat direct al acestui fapt este acela ca programele scrise in limbajul C au o portabilitate foarte buna.

In mod intuitiv spunem ca un program este portabil daca el poate fi transferat usor de la un tip de calculator la altul. La ora actuala se afirma ca programele scrise in C sunt cele mai portabile.

2. Meniul "C"

In randul de sus al ecranului avem un chenar cu meniul principal, iar in randul de jos linia de stare, unde sunt evidentiate tastele functionale, momentan active.

Meniul principal cuprinde urmatoarele puncte:

FILE	Administare fisiere
EDIT	Editor; combinatii Clipboard
SEARCH	Cautare si inlocuire parti text sursa
RUN	Start pentru programul actual in memoria de lucru
COMPILE	Compilare (traducere) text program
DEBUG	Tratare erori si depunere
OPTIONS	Optiuni
PROJECT	Concatenare programe
WINDOW	Functii de fereastra
HELP	Sistem de functii ajutatoare

Aceste puncte de meniu se pot selecta in trei moduri:
- Cu mouse-ul
- Prin combinatii de taste ALT+Litera de inceput
- F10 si deplasarea cursorului cu bara de selectie

Comutarile rapide cele mai importante sunt intotdeauna in linia de stare. In afara de asta le gasiti in fiecare submeniu. Unele dintre ele se vor tine minte, prin exercitiu, pe de rost:

ALT+X	Terminarea "C"-ului
ALT+F	Direct in meniul FILE
ALT+Nr.	Vizualizarea ferestrei Nr.
ALT+F3	Inchiderea ferestrei actuale
CTRL+F9	Compilarea si lansarea programului
F9	Numai compilare
F2	Salvarea textului sursa
F3	Apelarea textului sursa
CTRL+F5	Modificarea ferestrei

3. Comenzi de baza ale EDITORULUI

Editorul "C" este un program de prelucrare texte in format "nondocument" cu ajutorul caruia se scriu programele sursa, care urmeaza apoi a fi compilate.

3.1. Comenzi de baza:

Ctrl+N - insereaza un rand liber

Ctrl+T - sterge partea de cuvant din dreapta cursorului
Ctrl+QY - sterge randul din dreapta cursorului
Ctrl+Y - sterge randul in care se afla cursorul
Ctrl+QL - restaureaza o portiune de rand stearsa

3.2. Manipularea blocurilor:

Ctrl K+B - marcheaza inceputul blocului |
Ctrl K+K - marcheaza sfarsitul blocului | (Shift + sageti)
Ctrl K+C - copiaza blocul marcat la pozitia cursorului
Ctrl K+V - muta blocul la pozitia cursorului
Ctrl K+Y - sterge blocul marcat
Ctrl K+H - deselecteaza blocul marcat
Ctrl K+W - salveaza blocul marcat

3.3 Transmiterea unui bloc in clipboard

Pentru salvari intermediare sau pentru colaje din textul sursa programul pune la dispozitie un clipboard.

Modul de utilizare:
- se marcheaza un bloc in fereastra curenta;
- se selecteaza functia COPY din punctul EDIT al meniului principal;
- se selecteaza optiunea SHOW CLIPBOARD din meniul EDIT, se intra in mediul CLIPBOARD unde gasim marcat ultimul text salvat;
- selectati cu ALT+Nr. fereastra fisierul destinatie si asezati cursorul acolo unde doriti sa inceapa blocul ce urmeaza a fi inserat;
- dupa selectarea punctului PASTE din meniul EDIT, blocul va fi copiat din CLIPBOARD in noua sursa.

Optiunea rapida:
- se marcheaza un bloc in fereastra curenta;
- se tasteaza CTRL+INS;
- selectati cu ALT+Nr. fereastra fisierul destinatie si asezati cursorul acolo unde doriti sa inceapa blocul ce urmeaza a fi inserat;
- se tasteaza SHIFT+INS si blocul va fi copiat din CLIPBOARD in noua sursa.

Obs.: Pe masura avansarii, in cadrul viitoarelor laboratoare, veti descoperi noi comenzi ale editorului, care vor fi asimilate pe parcurs.

II. DESFASURAREA LUCRARII

1. Intrati in editorul mediului de programare Turbo C prin lansarea programului

tc.exe

Deschideti o fereastra (cu ajutorul lui NEW din meniul FILE) in care introduceti un text oarecare de la tastatura.
2. Salvati textul de mai sus intr-un fisier cu numele dat de dumneavoastra avand extensia .c .
3. In fisierul definit anterior marcati blocuri pe care sa le copiati la diferite pozitii, sa le mutati sau sa le stergeti. Observati diferenta dintre copiere si mutare. Deselectati apoi blocurile marcate.
4. Salvati un bloc marcat intr-un alt fisier cu nume nou.
5. Deschideti noi fisiere (implicit ferestre) intre care faceti transferuri de blocuri prin intermediul CLIPBOARD - ului.
6. Modificati dimensiunile ferestrelor (Ctrl + F5) cu ajutorul sagetilor si a combinatiilor Shift + sageata.
7. Inchideti toate ferestrele (Alt + F3) si apoi parasiti mediul Turbo C (Alt + X).

LIMBAJE DE PROGRAMARE

Lucrarea 2

NOTIUNI DE BAZA IN LIMBAJUL "C"

I. NOTIUNI TEORETICE

1. Scurt dictionar introductiv:

OBIECT = continutul unei zone de memorie
LVALOARE = expresie care se refera la un obiect
POINTER = o variabila care contine adresa altei variabile
EXPRESIE = combinatie de variabile si constante pentru a produce valori noi
( ) = lista de argumente a unei functii sau in expresii aritmetice
{ } = grupeaza instrumente compuse, functii
[ ] = incadreaza dimensiuni de masiv sau indicii elementelor de masiv
" " = incadreaza sir de caractere
' ' = incadreaza un caracter sau o secventa de evitare
; = termina o instructiune
/*...*/ = comentariu

2. Constante

2.1. Constante intregi ┐(generate pe 2 octeti)
Pot fi de urmatoarele tipuri:

- zecimala
- octala - este o constanta intreaga si incepe cu cifra 0

Ex.: 8 -> 010

- hexazecimala - este o constanta intreaga precedata

2.2. Constante lungi explicite (generate pe 4 octeti)
Este o constanta intreaga urmata de litera l sau L.

┐2.3. Costante flotante
Sunt alcatuite dintr-o parte intreaga, un punct zecimal, o parte fractionara, un "e" sau "E" si un exponent intreg cu semn.
Pot lipsi: partea intreaga, partea fractionara, punctul zecimal, litera "e" sau exponentul.

2.4. Constante caracter
Sunt reprezentate printr-un caracter scris intre apostrofuri.
Ex.: 'x'

2.5. Constante simbolice
Sunt identificatori cu valoare de constanta. Se introduc cu "#define".
Ex.: # define MAX 1000

3. Variabile

3.1. Clase de memorie┐
Din punctul de vedere al modului de lucru cu memoria avem urma toarea clasificare:

a) Variabile automatice
Se declara implicit in context sau cu identificatorul "auto".
Aceste variabile sunt locale fiecarui bloc si se distrug la parasirea blocului.

b) Variabile externe
Se declara cu "extern" sau implicit in context.
Valorile variabilelor externe se pastreaza de-a lungul intregului program.

c) Variabile statice
Se declara cu "static" si nu sunt recunoscute decat in cadrul fisierului in care au fost definite.
Sunt de doua tipuri:
- interne - in interiorul unei functii si nu se distrug la parasirea functiei;
- externe - in tot programul.

d) Variabile registru
Se declara cu "static". Sunt asemanatoare variabilelor auto, dar folosite mai des.

3.2. Tipuri de variabile

a) Variabile tip caracter -> se declara cu "char"
Sunt pe 1 octet (-128 - 127)

b) Variabile de tip intreg -> "int"
Pot fi: "short" (2 octeti)
"long" (4 octeti)
"unsigned" (fara semn)

```
Ex.:            short int x;
                long int y;
                unsigned int z;
```

c) Variabile de tip flotant:
"float" -> simpla precizie
"double"-> dubla precizie

```
     Tipuri aritmetice de variabile
     |                          |
   intregi                  flotante (reale)
   |     |                    |          |
char    int                  float      double
       (long, short, unsigned)
```

Obs.: intregii sunt intotdeauna cantitati cu semn.

4. Conversii

Conversiile de baza acceptate in programarea in limbajul "C" sunt:

```
char  ->   int     (Atentie! se poate produce un intreg cu semn)

int   ->   char    (se pierd bitii de ordin superior)

short ->   int

long  ->   int     (apare truncherea: surplusul de biti
      ->   short    de ordin superior se pierde)
      ->   char

float ->   double  (se adaoga zerouri pe partea fractionara)

int   ->   float
      ->   unsigned

char  ->   int

short ->   int
```

Obs.:
a) Daca un operand este "double" se converteste automat si cela

lalt operand rezultatul fiind de tip "double".
La fel pentru "long" si "unsigned".
b) Conversii prin asignare: valoarea membrului drept este conver
tita la valoarea membrului stang, care este si tipul rezultatu
lui.
c) Pot interveni si conversii logice.

5. Specificatorii de format:
%u   -> intreg fara semn
%d   -> intreg
%ld   -> intreg lung
%p   -> pointer
%f   -> valoare in virgula flotanta
%e   -> valoare in virgula flotanta si format exponential
%c   -> caracter
%s   -> sir
%x sau %X -> intreg in format hexazecimal

Obs.: Atunci cand in specificatorii de format apar si cifre.
De ex.: %5d   -> intreg pe 5 spatii cu aliniere de la dreapta
        %-5d   -> intreg pe 5 spatii cu aliniere de la stanga

6. Secvente de evitare:
\n   -> linie noua
\t   -> caracterul tab
\b   -> caracterul backspace
\\   -> caracterul backslash (\)
\/   -> caracterul slash (/)
\xhhh   -> caracterul reprezentat de codul ASCII hhh, unde hhh
reprezinta 1-3 cifre hexazecimale.

7. Fisierul prefix "stdio.h"┐
Acest fisier contine cateva rutine standard prefabricate, o serie de definitii
utile (si indispensabile) in lucrul cu fisierele, definitii de constante
simbolice, precum si redefinirea unor caractere speciale care nu exista pe toate
tipurile de tastaturi ale terminalelor.
Se recomanda includerea acestui fisier in toate programele scrise in limbajul
"C".

8. Executia unui program

Fiecare program in C are o functie primara "main". In mod normal, executia unui
program se sfarseste la sfarsitul functiei "main".

9. Rutine standard

9.1. Iesiri

a) Rutina "printf" -tipareste un mesaj pe ecran

Format: printf(<sir_format>, <obiect>, <obiect>, ...)

Obs.: <sir_format> trebuie sa fie incadrat intre ghilimele.

Ex.:   printf("rezultat %d \n", rezultat);
                           | |
       specificator de format   secventa de evitare

Program :

        /* HELLO.C -- Hello, world */

```
# include <stdio.h>

main ( )
{
        printf ("Hello,world \n");
}
```

Obs.:

```
Ctrl + F9       -> compilarea si executia programului
ALT + F5        -> vizualizarea ecranului
```

b) Rutina "puts"

Scrie pe ecran un sir urmat de caracterul linie noua (\n).

```
Astfel:     printf("Salut \n");     <=>     puts("Salut");
```

c) Rutina "putchar"

Scrie un caracter fara salt la linie noua.

```
Astfel:     printf("%c", 'x');   <=>  putchar('x');

            printf("%c \n", 'x'); <=>  putchar('x');
                                       putchar('\n');
```

Obs.: Codul rutinei "printf" este mare si de aceea pentru optimizare se prefera "puts" si "putchar".

9.2. Intrari

a) Rutina "scanf" - citeste un mesaj de la tastatura.

```
Format: scanf(<sir_format>, <adresa>, <adresa>, ...);

Ex.:   scanf("%d     %d", &x, &y);
                  |
                spatiul indica faptul ca intre cele doua valori
```

tastate pot exista oricate spatii goale (blanc, tab, linie noua).

```
Obs.:   scanf("%d , %d", &x, &y);
                  |
             virgula indica faptul ca cele doua valori vor fi
             despartite la citire prin virgula.
```

Program (transmiterea unei adrese):

```
        /*  HELLO.C  --  Hello, world */

# include <stdio.h>

main ( )
{
        char nume [25];
        printf("Nume:");
        scanf("%s", nume);
        printf ("Salut, %s \n", nume);
}
```

Obs.: Programul va afisa doar numele persoanei tastate, nu si prenumele, deoarece spatiul blanc indica terminarea caracterului de citit.

O solutie de remediere a acestui neajuns:

```
        /*  HELLO.C  --  Hello, world */

# include <stdio.h>
main ( )

{
        char nume [15], prenume [15];
        printf("Nume:");
        scanf("%s    %s", nume, prenume);
        printf ("Salut, %s %s \n", nume, prenume);
}
```

b. Rutina "gets"

Citeste intregul sir tastat pana cand se apasa Enter.

Format: gets(sir);

O alta solutie la problema de mai sus:

```
        /*  HELLO.C  --  Hello, world */

# include <stdio.h>
main ( )
{
        char nume [25];
        printf("Nume:");
        gets(nume);
        printf ("Salut, %s \n", nume);
}
```

c. Rutinele "getch" si "getche"
Aceste rutine citesc un caracter unic de la tastatura. Cele doua rutine nu au parametri ci returneaza o valoare de tip char.

Format: getch( ); lucreaza fara ecou
         getche( ); afiseaza mai intai caracterul tastat

Ex.:

```
#include <stdio.h>
main ( )
{
   char c;
   printf("Tastati un caracter");
   c = getch( );                    |
   putchar (c);                     |  <=>  putchar(getch( ));
}
```

Sau:

```
 #include <stdio.h>

  main ( )
  {
    printf("Tastati un caracter");
    getche( );
  }
```

II. DESFASURAREA LUCRARII

1. Scrieti si apoi rulati toate toate programele intalnite la paragrafele 9.1. si 9.2. Concluzionati.
2. Cum s-ar putea face rescrierea programelor anterioare cu vizualizarea ecranului fara folosirea Alt + F5.

LIMBAJE DE PROGRAMARE

Lucrarea 3

OPERATORI, EXPRESII, INSTRUCTIUNI

I. INTRODUCERE TEORETICA

1. Operatori si Expresii

O expresie combina variabile si constante pentru a produce valori noi.

1.1. Expresii primare:

identificator
constanta
sir
(expresie)
expresie primara[expresie]
expresie primara(lista de expresii)
lvaloare.identificator
expresie primara -> identificator

Obs.: O lista de expresii arata astfel:

expresie
lista de expresii, expresie

1.2. Operatori unari: *, &, -, !, ~, ++, --, (nume de tip), sizeof.

Expresii unare (se grupeaza de la stanga la dreapta):

* expresie
& lvaloare
- expresie
! expresie
~ expresie
++ lvaloare
-- lvaloare
lvaloare ++
lvaloare --
(nume de tip) expresie
sizeof expresie
sizeof (nume de tip)

Explicatii:

a) * -> operatorul de indirectare. Expresia care-l urmeaza este un pointer iar rezultatul este o lvaloare.

```
Ex.:  y  =  *px   /* y reprezinta continutul adresei  pe  care  o pointeaza px
*/
```

b) & -> operatorul de obtinere a unui pointer. Operandul este lvaloare iar rezultatul este pointer.

```
Ex.:    px = &x ;
        y = *px ;       <=>     y = x
```

Obs.: & se aplica numai la variabile.
Sunt gresite formatele de genul: &(x+1)
&3

c) - -> operatorul de negativare

d) ! -> operatorul de negare logica. Rezultatul este 0 sau 1.

e) ~ (tilda) -> operatorul de complementare la 1. Converteste bitii:
0 -> 1
1 -> 0

Operanzii sunt "short int" sau "long int".

f) ++ -> operatorul de incrementare a operandului cu 1.
Obs.: ++n -> incrementeaza pe n inaintea utilizarii valorii sale
n++ -> incrementeaza pe n dupa ce valoarea sa a fost utilizata

g) -- -> operatorul de decrementare. (La fel ca mai sus)

h) (nume de tip) -> operatorul de conversie de tip.
Produce conversia valorii expresiei la tipul denumit. Aceasta constructie se numeste "cast".

i) sizeof -> furnizeaza dimensiunea in octeti a operandului sau.
Obs.: sizeof(nume de tip) -> furnizeaza dimensiunea in octeti a unui obiect de tipul indicat

sizeof(tip) -> este luata ca o unitate

Ex.: sizeof(tip)-2 <=> (sizeof(tip))-2

1.3. Operatori multiplicativi: *, /, %
Se grupeaza de la stanga la dreapta.

* -> inmultire
/ -> impartire
% -> furnizeaza restul impartirii primei expresii la cea de-a doua.

Ex.: expresia (a/b)*b+a%b este egala cu a

1.4. Operatori aditivi: + (adunare), - (scadere)

1.5. Operatori de deplasare: <<, >>
Formatul expresiei de deplasare: expresie << expresie
expresie >> expresie

<< -> produce deplasarea la stanga a operandului din stanga cu un numar de pozitii binare dat de operandul din dreapta.
>> -> produce deplasarea la dreapta a operandului din stanga cu un numar de pozitii binare dat de operandul din dreapta.

Obs.: Operanzii din stanga trebuie sa fie de tip integral.
Operandul din dreapta este convertit la tipul "int". Bitii elibe
rati devin 0.

1.6. Operatori relationali: <, >, <=, >=

Produc valoarea 1 daca relatia specificata este adevarata
si 0 daca este falsa.

Obs.: i < x - 1 <=> i < (x - 1)

1.7. Operatori de egalitate: == (egalitate), != (diferit)
Obs.: Au precedenta mai mica decat operatorii relationali.
Ex.: a<b == c>d <=> 1 daca a<b si c>d
                     0 in rest

1.8. Operatorul pe biti SI: &

```
&  | 0  1
--------
0  | 0  0
   |
1  | 0  1
```

1.9. Operatorul pe biti SAU EXCLUSIV: ^

```
^  | 0  1
--------
0  | 0  1
   |
1  | 1  0
```

1.10. Operatorul pe biti SAU INCLUSIV: |

```
"|"| 0  1
--------
0  | 0  1
   |
1  | 1  1
```

Obs.: Poate fi folosit pentru a seta bitii.
Ex.: x = x | mask unde mask este o masca plasata pe bitii de interes.

1.11. Operatorii SI LOGIC (&&) si SAU LOGIC (||)
Se grupeaza de la stanga la dreapta si rezultatul este 0 sau 1.

1.12. Operatorul conditional: "? :"
Expresie conditionala: expresie1 ? expresie2 : expresie3

Daca expresie1 are valoarea de adevar:
1 - rezultatul este expresie2
0 - rezultatul este expresie3

Obs.: Atentie la corectitudinea scrierii.
lval = ET1 ? (ET2 ? e1 : e2) : e3 este corect scrisa
lval = ET1 ? e1 : ET2 ? e2 : e3 este incorect scrisa

1.13. Operatori de atribuire: =, +=, -=, *=, /=, %=, >>=, <<=,
&=, ^=, |=.

Se grupeaza de la dreapta la stanga.

Expresia de atribuire: lvaloare operator expresie

Daca operatorul este "=", valoarea expresiei inlocuieste pe cea a obiectului
referit de lvaloare.

Expresiile de forma:

E1 op= E2 sunt echivalente cu E1 = E1 op E2
unde "op" este unul dintre operatorii: +, -, *, /, %, >>, <<, &, ^ sau |

Ex.:
x *= y + 1   <=>   x = x * (y + 1)   si nu   x = x * y + 1

Obs.: Pentru += si -=, operatorul stang poate fi si un pointer.

1.14. Operatorul virgula: ,
Expresia virgula: expresia1,expresia2

Se grupeaza de la stanga la dreapta.

2. Instructiuni
2.1. Instructiunea conditionala (if..else)

```
        if(expresie)
                instructiunea1;
        else
                instructiunea2;
```

Obs.: "else" se leaga de ultimul "if" intalnit.

```
Ex.:    if (exp1) instr1;
        else    if (exp2) instr2;
                else    if (exp3) instr3;
                        else instr4;
```

2.2. Instructiunea "while"

```
        while (expresie) instructiune;
```

Instructiunea se executa repetat atat timp cat valoarea expresiei este diferita de 0.
Testul se desfasoara la inceput, asa incat daca de la prima evaluare expresia este 0, instructiunea din corpul while nu se desfasoare deloc.

2.3. Instructiunea "do...while"

```
        do instructiune while (expresie);
```

Instructiunea se executa repetat pana cand valoarea expresiei devine 0.
Testul are loc dupa fiecare executie a instructiunii, prin urmare cel putin o data instructiunea va fi parcursa.

2.4. Bucla "for":

```
        for (exp1; exp2; exp3)
                instructiune;
      unde:   exp1    -> initializarea buclei (optional)
              exp2    -> testul de sfarsit al buclei (optional)
              exp3    -> actualizarea variabilei index (optional)
```

```
Bucla for este echivalenta cu:  exp1;
                                while (exp2)
                                        { instructiune;
                                          exp3;
                                         }
```

2.5. Instructiunea "break":

```
        break;
```

Determina terminarea celei mai interioare instructiuni "while", "do...while", "for" sau "switch" care o contine.
Controlul trece la instructiunea care urmeaza dupa instructiunea astfel terminata.
Nu se foloseste in instructiuni "if..else" sau direct in corpul functiei.

2.6. Instructiunea "switch" (de comutare):

```
        switch (exp)
                { case exp1 : sir1
                        break;
                  case exp2 : sir2
                        break;
                  default : sir
                }
```

2.7. Instructiunea "continue":

Format: continue;
Determina trecerea controlului la sfarsitul ciclului "while", "do...while" sau "for" si reluarea urmatoarei iteratii a ciclu lui.

2.8. Instructiunea "return":

Format: return; sau return (expresie);

2.9. Instructiunea "goto":

```
        goto eticheta;
```

Obs.: eticheta incepe neaparat cu o litera.

2.10. Instructiunea eticheta

Orice instructiune poate fi precedata de o eticheta de forma:
identificator:

2.11. Instructiunea nula
Format: ;

```
Ex.:    for (nc=0; getchar()!=EOF; ++nc);
```

II. DESFASURAREA LUCRARII

Se va scrie un program in limbajul "C" care va prelua de la tastatura doi operanzi (doua numere) si un operator, va efectua calculele intre cei doi operanzi folosind operatorul si va afisa rezultatul.

Pentru inceput se poate alcatui programul in limbaj natural:

- se definesc doua variabile (operanzii) de tip "float" si una (operatorul) de tip "int";

- se preiau de la tastatura operanzii;

- se preia de la tastatura operatorul si daca el este:

```
        '+'     - se efectieaza adunarea;
        '-'     - se efectueaza scaderea;
        '*'     - se efectueaza inmultirea;
        '/'     - daca operandul doi este diferit de 0 se
                efectueaza impartirea, iar daca nu se afiseaza
                mesaj de eroare;
        orice altceva   - se afiseaza mesaj de eroare;
```

- se menine ecranul pentru vizualizare pana la apasarea unei noi taste.

In limbaj "C" programul devine:

```
# include<stdio.h>
 main()
  {
    int oper;
    float x,y;
    printf("\nOperand 1: ");
    scanf("%f",&x);
    printf("\nOperand 2: ");
    scanf("%f",&y);
    printf("\nOperator : ");
    if((oper=getche())=='+')
        printf("\nRezultat adunare: %f\n",x+y);
    else if(oper=='-')
        printf("\nRezultat scadere: %f\n",x-y);
    else if(oper=='*')
        printf("\nRezultat inmultire: %f\n",x*y);
    else if(oper=='/')
        if(y==0)
            printf("\nOperand 2 eronat pentru impartire\n");
        else
            printf("\nRezultat impartire: %f\n",x/y);
    else if(oper=='%'||oper=='&'||oper==':')
        printf("\nNu s-a efectuat operatia\n");
    else
        printf("\nEroare\n");
   getch();
 }
```

1. Scrieti programul de mai sus in editorul "C".

2. Modificati programul astfel incat sa se efectueze oricate cicluri de preluare a datelor si executare a operatiilor, iar parasirea programului sa se efectueze conditionata de o tasta.

3. Modificati programul astfel incat selectia sa se realizeze cu "switch".

III. RASPUNSURI

2. Programul modificat cu "while":

```
# include<stdio.h>

 main()
  {
   int sfarsit;
   sfarsit=1;
   while(sfarsit!='0')
   {
    int oper;
    float x,y;
    printf("\nOperand 1: ");
    scanf("%f",&x);
    printf("\nOperand 2: ");
    scanf("%f",&y);
    printf("\nOperator : ");
    if((oper=getche())=='+')
        printf("\nRezultat adunare: %f\n",x+y);
    else if(oper=='-')
        printf("\nRezultat scadere: %f\n",x-y);
    else if(oper=='*')
        printf("\nRezultat inmultire: %f\n",x*y);
    else if(oper=='/')
        if(y==0)
            printf("\nOperand 2 eronat pentru impartire\n");
        else
            printf("\nRezultat impartire: %f\n",x/y);
    else if(oper=='%'||oper=='&'||oper==':')
        printf("\nNu s-a efectuat operatia\n");
    else
        printf("\nEroare\n");
   printf("\nTastand 0 iesiti din program\n");
   sfarsit=getch();
   }
 }
```

3. Programul rescris cu optiunea "switch":

```
 # include<stdio.h>

 main()

{ int gata;
  gata=1;
   while(gata!='0')
   {
    int oper;
    float x,y;
    printf("Operand 1: ");
    scanf("%f",&x);
    printf("Operand 2: ");
    scanf("%f",&y);
    printf("Operator : ");
    oper=getche();
    switch(oper)
    {
      case'+':printf("\nRezultat adunare: %f\n",x+y);
      break;
      case'-':printf("\nRezultat scadere: %f\n",x-y);
      break;
```

```
      case'*':printf("\nRezultat inmultire: %f\n",x*y);
      break;
      case'/':if(y==0)
                printf("\nOperand 2 eronat pentru impartire\n");
              else
                printf("\nRezultat impartire: %f\n",x/y);
              break;
      case'%':
      case'&':
      case':':printf("\nNu s-a efectuat operatia\n");
              break;
      default: printf("\nEroare\n");
    }
   printf("\nTastand 0 iesiti din program\n");
   gata=getch();

   }
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 4

INSTRUCTIUNI (partea II)

I. NOTIUNI TEORETICE

1. Instructiuni (recapitulare)

1.1. Instructiunea conditionala (if..else)

```
if(expresie)
        instructiunea1;
else
        instructiunea2;
```

Obs.: "else" se leaga de ultimul "if" intalnit.

```
Ex.:    if (exp1) instr1;
        else    if (exp2) instr2;
                else    if (exp3) instr3;
                        else instr4;
```

1.2. Instructiunea "while"

```
while (expresie) instructiune;
```

Instructiunea se executa repetat atat timp cat valoarea expresiei este diferita de 0. Testul se desfasoara la inceput, asa incat daca de la prima evaluare expresia este 0, instructiunea din corpul while nu se desfasoare deloc.

1.3. Instructiunea "do...while"

```
do instructiune while (expresie);
```

Instructiunea se executa repetat pana cand valoarea expresiei devine 0. Testul are loc dupa fiecare executie a instructiunii, prin urmare cel putin o data instructiunea va fi parcursa.

1.4. Bucla "for":

```
for (exp1; exp2; exp3)
        instructiune;

unde:   exp1    -> initializarea buclei (optional)
        exp2    -> testul de sfarsit al buclei (optional)
        exp3    -> actualizarea variabilei index (optional)
```

Bucla for este echivalenta cu:

```
exp1;
while (exp2)
        { instructiune;
          exp3;
         }
```

1.5. Instructiunea "break":

```
break;
```

Determina terminarea celei mai interioare instructiuni "while", "do...while", "for" sau "switch" care o contine. Controlul trece la instructiunea care urmeaza dupa instructiunea astfel terminata. Nu se foloseste in instructiuni "if..else" sau direct in corpul functiei.

1.6. Instructiunea "switch" (de comutare):

```
        switch (exp)
                { case exp1 : sir1
                        break;
                  case exp2 : sir2
                        break;
                  default : sir
                }
```

2. Functia standard "exit"

Prototipul functiei: void exit(int cod);

Functia se afla in fisierele de tip header: stdlib.h si process.h
La apelul acestei functii au loc urmatoarele actiuni:

* se videaza zonele tampon (bufferele) ale fisierelor deschise in scriere;
* se inchid toate fisierele deschise;
* se intrerupe executia programului.

Parametrul acestei functii defineste starea programului la momentul apelului. Astfel, valoarea zero defineste o terminare normala a executiei programului, iar o valoare diferita de zero semnaleaza prezenta unei erori (terminarea anormala a executiei programului).

Deci, putem apela functia "exit" pentru a termina executia unui program, indiferent de faptul ca acesta se termina normal sau din cauza unei erori.

II. DESFASURAREA LUCRARII

1. Se da functia

```
          y = 3*x*x + 2*x - 10      daca x > 0
     si
          y = 5*x + 10              daca x <= 0
```

Sa se scrie un program care citeste valoarea lui "x", iar apoi calculeaza si afiseaza valoarea lui "y".

O posibila solutie:

```
#include <stdio.h>
main()  /* citeste pe x, calculeaza si afiseza valoarea
           lui y definita astfel:
                    y = 3*x*x + 2*x - 10      daca x > 0
                    y = 5*x + 10              daca x <= 0   */

  {
   float x,y;
   printf("\nIntroduceti pe x:");
   scanf("%f", &x);
   if(x>0)
      y = 3*x*x + 2*x - 10;
   else
      y = 5*x + 10;
   printf("x = %f\ty = %f\n",x,y);
   getch();
  }
```

2. Sa se modifice programul, de la punctul 1, astfel incat in structiunea "if...else" sa fie inlocuita prin operatorul conditional.

3. Sa se scrie un program care calculeaza si afiseaza valorile functiei:

f(x) = 5*x*x + 3*x - 7
pentru x=1, 2, ..., 10

O posibila solutie:

```
#include<stdio.h>

main()  /*afiseaza valorile functei f(x)=5*x*x + 3*x - 7
                pentru x=1, 2, 3,... , 10  */
  {
   int x;
   x = 1;
   while(x <= 10)
        {
         printf("\nx=%d\tf(x)=%d\n", x, 5*x*x + 3*x - 7);

         x=x+1;  /* mai bine x++,
                    din punct de vedere al compactizarii*/
        }  /*sfarsit while*/
   getch();
  }  /*sfarsit main*/
```

4. Compactati programul de mai sus astfel incat corpul instruc tiunii "while" sa contina o singura instructiune.

5. Sa se scrie un program care citeste un intreg n, apoi calcu leaza si afiseaza factorialul n!

Avem
n! = 1*2*...*(n-1)*n
Acest calcul implica un proces ciclic de forma:

```
 f=1 si i=1
 while(i<=n)
        {
         f=f*i;
         i=i++;
        }
```

In program trebuie inclusa si o secventa de evitare a valori lor nedorite care vor fi tratate cu functia "exit".

O posibila solutie:

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

main() /* citeste pe n din intervalul [1,170],
          calculeaza si afiseaza pe n! */

   {
     int n,i;
     double f;

     printf("\nValoarea lui n:");
```

```
    if(scanf("%d",&n) != 1)  /* valorile zecimale sunt truncheate
                               prin conversia de citire %d */
       {
        printf("\n Nu s-a tastat un numar\n");
        getch();
        exit(1);
       }
    if(n<0 || n>170)
       {
        printf("\n n nu apartine intervalului [0,170]\n");
        getch();
        exit(1);
       }
    f=1.0;
    i=2;
    while(i<=n)
        f=f*i++;
        printf("\n n=%d\t n!=%g\n", n, f);
    getch();
}
```

6. Modificati programul de la punctul 5. astfel incat sa putem calcula oricate valori factoriale fara a iesi din program.
Modificarea va fi facuta prin intermediul unui ciclu "while".

7. Modificati programul de la punctul 6. astfel incat calcularea factorialului sa se realizeze cu ajutorul instructiunii "for".

8. Modificati programul de la punctul 7. astfel incat sa putem calcula oricate valori factoriale fara a iesi din program, dar prin intermediul unui ciclu "do ... while".

9. Sa se scrie un program cu secvente de evitare asemanatoare celui de la punctul 5., care cu ajutorul instructiunii "for" sa calculeze a la puterea n.

10. Sa se scrie un program care citeste o cifra din intervalul [1,7] si afiseaza denumirea zilei din saptamana corespunzatoare cifrei respective (1-luni, 2-marti, etc.).

III. RASPUNSURI:

2. L4_2.C

```
#include <stdio.h>

main()  /* citeste pe x, calculeaza si afiseza valoarea
                lui y definita astfel:
                        y = 3*x*x + 2*x - 10      daca x > 0
                        y = 5*x + 10              daca x <= 0   */

  {
   float x,y;
   printf("\nIntroduceti pe x:");
   scanf("%f", &x);
   y = x>0 ? 3*x*x + 2*x - 10 : 5*x + 10;
   printf("x = %f\ty = %f\n",x,y);
   getch();
  }
```

4. L4_4.C

```
#include<stdio.h>

main()  /*afiseaza valorile functei f(x)=5*x*x +3*x  - 7
                pentru x=1, 2, 3,... , 10  */
  {
   int x;
   x = 0;
   while(++x <= 10)
        {
         printf("\nx=%d\tf(x)=%d\n", x, 5*x*x + 3*x - 7);
        } /* sfarsit while */
   getch();
  }  /*sfarsit main*/
```

6. L4_6.C

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

main() /* citeste pe n din intervalul [1,170],
          calculeaza si afiseaza pe n! */
   {
     int n,i;
     char stop;
     double f;
     stop=1;
     while(stop!='y')
     {
     printf("\nValoarea lui n:");
     if(scanf("%d",&n) != 1)    /* valorile zecimale sunt truncheate
                                   prin conversia de citire %d */
        {
         printf("\n Nu s-a tastat un numar\n");
         getch();
         exit(1);
        }
     if(n<0 || n>170)
        {
         printf("\n n nu apartine intervalului [0,170]\n");
         getch();
         exit(1);
        }
```

```
     f=1.0;
     i=2;
     while(i<=n)
         f=f*i++;
         printf("\n n=%d\t n!=%g\n", n, f);
     printf("\nDoriti sa parasiti programul? (y/n)\n");
     stop=getch();
    }
  }
```

7. L4_7.C

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

main() /* citeste pe n din intervalul [1,170],
          calculeaza si afiseaza pe n! */
   {
     int n,i;
     char stop;
     double f;
     stop=1;
     while(stop!='y')
     {
     printf("\nValoarea lui n:");
     if(scanf("%d",&n) != 1)    /* valorile zecimale sunt truncheate
                                   prin conversia de citire %d */
        {
         printf("\n Nu s-a tastat un numar\n");
         getch();
         exit(1);
        }
     if(n<0 || n>170)
        {
         printf("\n n nu apartine intervalului [0,170]\n");
         getch();
         exit(1);
        }
     for(f=1.0,i=2;i<=n;i++)
         f *= i;
         printf("\n n=%d\t n!=%g\n", n, f);
     printf("\nDoriti sa parasiti programul? (y/n)\n");
     stop=getch();
    }
  }
```

8. L4_8.C

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

main() /* citeste pe n din intervalul [1,170],
          calculeaza si afiseaza pe n! */
   {
     int n,i;
     char stop;
     double f;
     do
     {
     printf("\nValoarea lui n:");
     if(scanf("%d",&n) != 1)     /* valorile zecimale sunt truncheate
                                   prin conversia de citire %d */
        {
         printf("\n Nu s-a tastat un numar\n");
         getch();
         exit(1);
```

```
        }
     if(n<0 || n>170)
        {
         printf("\n n nu apartine intervalului [0,170]\n");
         getch();
         exit(1);
        }
     for(f=1.0,i=2;i<=n;i++)
         f *= i;
         printf("\n n=%d\t n!=%g\n", n, f);
     printf("\nDoriti sa parasiti programul? (y/n)\n");
     stop=getch();
    }while(stop!='y');
  }
```

9. L4_9.C

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

main()  /* citeste pe a si n, calculeaza si afiseaza pe a**n  */
{
 int i, n;
 long double a, p;  /* long double are specificatorul de format %Lf */

 printf("\nBaza a = ");
 if(scanf("%Lf",&a)!=1)
        {
         printf("\nNu s-a tastat un numar\n");
         getch();
         exit(1);
        }
 printf("Exponentul n = ");
 if(scanf("%d",&n) != 1 || n<0)  /* valorile zecimale sunt truncheate
                                 prin conversia de citire %d */
        {
         printf("Nu s-a tastat un numar pozitiv");
         getch();
         exit(1);
        }
 for(i=0, p=1.0; i<=n; i++)
        p *= a;
 printf("a=%Lf\tn=%d\ta**n=%Lf\n",a,n,p);
 getch();
}
```

10. l4_10.C

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

main()  /* citeste o cifra din intervalul [1, 7] si afiseaza
         denumirea zilei corespunzatoare cifrei respective */
{
 int i;
 puts("Tastati o cifra din intervalul [1,7].");
 scanf("%d",&i);
 switch(i)
 {
  case 1: puts("luni");
        break;
  case 2: puts("marti");
        break;
  case 3: puts("miercuri");
        break;
```

```
  case 4: puts("joi");
        break;
  case 5: puts("vineri");
        break;
  case 6: puts("sambata");
        break;
  case 7: puts("duminica");
        break;
  default: puts("Tasta gresita");
        break;
 }
 getch();
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 5

PROGRAMAREA PROCEDURALA. FUNCTII

I. NOTIUNI TEORETICE

1. Programarea procedurala, functii, apelul si revenirea din ele

Incepand cu primele limbaje de programare de nivel inalt s-a utilizat programarea procedurala. Daca se doreste executarea unui anumit set de instructiuni, cu date diferite sau in locuri diferite, acestea se grupeaza intr-o subrutina care va fi apela la, printr-un salt, ori de cate ori este nevoie. Acest salt este cu revenire la instructiunea urmatoare instructiunii care a facut saltul si de aceea el difera de salturile realizate prin instructiunea "goto". Secventa de instructiuni organizata in acest fel are diferite denumiri in diverse limbaje de programare:subprogram, subrutina, procedura, functie, etc.

In toate limbajele de programare se considera doua categorii de proceduri:
a) Proceduri care definesc o valoare de revenire.
b) Proceduri care nu definesc o valoare de revenire.

Procedurile din categoria a. de obicei se numesc functii. Valoare de revenire se mai numeste si valoare de intoarcere sau valoare returnata de functie.

In limbajul "C", toate subrutinele sunt denumite functii. Asa cum s-a mai spus, teoretic orice functie returneaza o valoare. In practica insa, valorile returnate sunt uneori ignorate,iar definirile recente, permit declararea functiilor de tip void, adica ele nu returneaza nici o valoare.

O functie poate fi declarata si definita. Prin declarare, o functie este facuta cunoscuta programului, astfel incat ea sa poata fi apelata de alte functii. La definire este specificat codul efectiv al functiei.

Limbajul "C" livreaza o serie de functii care au o utilizare frecventa in programare. Ele se pastreaza intr-un fisier special, in format OBJ, adica compilat si se adauga la faza de editare in program. Aceste functii sunt denumite functii standard de biblioteca. Ele au prototipurile in diferite fisiere de extensie h.

Exemple de astfel de functii sunt functiile cu ajutorul carora se realizeaza operatii de intrare/iesire (printf, scanf, etc.), rutine pentru calculul functiilor elementare (sqrt, sin, cos, atan, log, pow), etc.

Prototipurile functiilor standard se includ in program inaintea apelurilor lor prin directiva #include.

Exemple de fisiere cu prototipuri:
stdio.h (printf, scanf, gets, puts, etc.)
conio.h (putch, getch, etc.)
math.h (sqrt, sin, cos, etc.)

1.1. Declararea functiilor

In declararea functiilor se pot folosi doua stiluri:
a) Stilul clasic - specifica doar numele functiei si tipul valorii returnate:

```
tip nume_functie();
```

Nu exista informatii asupra parametrilor astfel incat nu se pot face verificari de eroare.

b) Stilul modern - se introduc si informatii despre parametrii functiei.
Constructia astfel formata se numeste prototip de functie:

```
                tip nume_functie(inf_p, inf_p, etc.);
        unde "inf_p" poate avea formele: "tip" sau "tip nume".
```

Obs.: Metoda b. este de preferat deoarece compilatorul poate face verificari privind numarul si tipul parametrilor la apelul functiilor.

1.2. Definirea functiilor

a) Stilul clasic defineste o functie in modul urmator:

```
      tip nume_functie(nume parametri)
        definitii_parametri
        {
        ...
        }
```

b) Stilul modern - definirile parametrilor apar in prima linie:

```
      tip nume_functie(inf_p, inf_p, etc.)
        {
        ...
        }
```

unde "inf_p" contine toate informatiile despre parametrii (tip si nume). Prima linie va fi deci identica cu prototipul functiei, cu o exceptie: la definire nu apare caracterul ";".
Obs. Cele doua stiluri prezentate mai sus definesc etape de dezvoltare ale limbajului "C".
In scrierea programelor recomandam folosirea cu precadere a stilului modern.

2. Specificatorii de format (recapitulare, completare)

Un specificator de format incepe cu un caracter procent (%). In continuarea sa, mai pot exista caracterele indicate mai jos:
caracterul minus "-" (optional) determina cadrarea stanga a datelor corespunzatoare. Daca lipseste, datele se cadreaza implicit in dreapta campului in care se scriu.

sir de cifre zecimale (optional) defineste dimensiunea minima a campului in care se afiseaza caracterele. Daca data este mai mica decat campul specificat, ea se va scrie in campul respectiv in dreapta sau in stanga functie de absenta sau prezenta lui minus.

un punct urmat de un numar (optional) defineste precizia.
Daca data este flotanta, precizia defineste numarul de zecimale. Daca data este un sir de caractere, precizia indica numarul de caractere care se scriu.
una sau doua litere definesc tipul de conversie aplicat.
Obs. Un specificator de format incepe cu caracterul % si se termina cu o litera.
Daca vrem sa afisam un caracter procent (%), tastam: %%
Primul caracter se ignora, iar cel de-al doilea se va afisa la terminal.

In continuare vom indica principalele conversii realizate prin specificatorii de format:

```
%c      -> Permite afisarea unui caracter.
%s      -> Permite afisarea unui sir de caractere.
```

Ex.:

Apelul:

```
      printf("*%10s*","abc");
    afiseaza:
      *       abc*

Apelul:
      printf("*%-10s*","abc");
    afiseaza:
      *abc       *

    Apelul:
      printf("*%10.5s*","Program");
    afiseaza:
      *     Progr*

%d      -> Afiseaza un intreg (int).┐

Ex.:
    Apelul:
      printf("*%10d*",123);
    afiseaza:
      *       123*

    Apelul:
      printf("*%-10d*",123);
    afiseaza:
      *123       *

    Apelul:
      printf("*%010d*",123);
    afiseaza:
      *0000000123*


%u      -> Afiseaza un intreg fara semn (unsigned).

%o      -> Datele de tip int si unsigned sunt convertite
                si afisate in octal┐
Ex.:
    Apelul:
      printf("*%10o*",123);
    afiseaza:
      *       173*

%x sau %X  -> Datele de tip int si unsigned sunt convertite si afisate in format
hexazecimal

Ex.:
    Apelul:
      printf("*%10x*",123);
    afiseaza:
      *        7b*

In cazul in care se foloseste litera mare X, cifrele hexa peste 9 se afiseaza cu
litere mari.

Apelul:
      printf("*%10x*",123);
    afiseaza:
      *        7B*

Daca litera "l" precede pe una din literele d, o, x, X sau u, se fac conversii
din tipul long sau long unsigned.

%f      -> Afiseaza o valoare float sau double in virgula flotanta.
```

%e sau %E -> Afiseaza o valoare float sau double in virgula flotanta si format exponential.

%g sau %G -> Alege intre %f si %e(sau E) astfel incat afisarea sa ocupe un numar minim de caractere.

Daca litera "L" precede una din literele f, e, E, g sau G, data care se afiseaza este de tip long double.

%p -> Afiseaza un pointer.

II. DESFASURAREA LUCRARII

1. Se va scrie un program alcatuit pe principiile programarii procedurale in care functia principala (main) va apela alte trei functii:
   - functia "preluare" care va citi de la tastatura doua variabile x si y si le va depune la adresele &x, &y.
   - functia "calcul" care va calcula rezultatul impartirii x/y in conditia y!=0.
   - functia "afisare" care va afisa rezultatul obtinut.

Programul va fi realizat in "stilul modern".

Amintim doi operatori unari necesari pentru functia "preluare":

a) * -> operatorul de indirectare. Expresia care-l urmeaza este un pointer (o variabila care contine adresa altei variabile) iar rezultatul este o lvaloare (o expresie care se refera la un obiect).
Ex.: y = *px /* y reprezinta continutul adresei pe care o pointeaza px */

b) & -> operatorul de obtinere a unui pointer. Operandul este lvaloare iar rezultatul este pointer.
Ex.: px = &x ;
y = *px ; <=> y = x
Obs.: & se aplica numai la variabile.
Sunt gresite formatele de genul: &(x+1)
&3

O posibila solutie:

```
#include<stdio.h>

 /* Declaratii de functii */

 void preluare(float*p1,float*p2);
 float calcul(float deimpartit,float impartitor);
 void afisare(float rezultat);

 const float INFINIT = 3.4e+38;

 /* Functia principala, punctul de pornire al programului*/
void  main()
 {
    float x,y,rez;
    preluare(&x,&y);
    rez=calcul(x,y);
    afisare(rez);
    getch();
  }   /*Sfarsit functie principala*/

  /*Definiri de functii*/

  void preluare(float*p1,float*p2)
```

```
  {
     printf(" \nIntroduceti primul numar:  ");
     scanf("%f",p1);
     printf("\nIntroduceti al doilea numar:  ");
     scanf("%f",p2);
   }

   float calcul(float deimpartit,float impartitor)
   {
      if(impartitor==0.0)
        return(INFINIT);
      else
        return(deimpartit/impartitor);
   }

   void afisare(float rez)
   {
      if(rez==INFINIT)
          printf("\nRezultat nedefinit\n");
      else
          printf("\nRezultat impartire =  %f\n",rez);
    }
```

2. Sa se modifice programul de mai sus astfel incat sa se elimine functia "preluare" dar, principial, comportarea sa ramana identica.

3. Sa se modifice programul de mai sus astfel incat sa se elimine functia "afisare" dar, principial, comportarea sa ramana identica.

4. Sa se modifice programul de mai sus astfel incat acesta sa contina doar functia principala (main) dar, principial, comportarea sa ramana identica.

5. Sa se scrie un program in care functia principala (main) apeleaza o alta functie numita factorial care calculeaza si returneaza pe n! pentru n in intervalul [1,170], altfel returneaza "-1".
Rezultatul va fi depus in programul principal in variabila "fact", iar rezultatul va fi afisat in urmatoarea forma:

```
    fact = factorial(x);
    printf("\nFactorialul format(e): %e\n    ",fact);
    printf("\nFactorialul format(f): %f\n    ",fact);
    printf("\nFactorialul format(.0f): %.0f\n",fact);
    printf("\nFactorialul format(d): %d\n    ",fact);
    printf("\nFactorialul format(ld): %ld\n  ",fact);
    printf("\nFactorialul format(g): %g\n    ",fact);
```

III. RASPUNSURI:

2. L5_2.C

```
#include<stdio.h>

 /* Declaratii de functii */

 float calcul(float deimpartit,float impartitor);
 void afisare(float rezultat);

 const float INFINIT = 3.4e+38;

 /* Functia principala, punctul de pornire al programului*/
void  main()
 {
    float x,y,rez;
    printf(" \nIntroduceti primul numar:  ");
    scanf("%f",&x);
    printf("\nIntroduceti al doilea numar:  ");
    scanf("%f",&y);
    rez=calcul(x,y);
    afisare(rez);
    getch();
  }   /*Sfarsit functie principala*/

  /*Definiri de functii*/

   float calcul(float deimpartit,float impartitor)
   {
      if(impartitor==0.0)
        return(INFINIT);
      else
        return(deimpartit/impartitor);
   }

   void afisare(float rez)
   {
      if(rez==INFINIT)
          printf("\nRezultat nedefinit\n");
      else
          printf("\nRezultat impartire =  %f\n",rez);
    }
```

3. L5_3.C

```
#include<stdio.h>

 /* Declaratii de functii */

 void afisare(float rezultat);

 const float INFINIT = 3.4e+38;
 /* Functia principala, punctul de pornire al programului*/
void  main()
 {
    float x,y,rez;
    printf(" \nIntroduceti primul numar:  ");
    scanf("%f",&x);
    printf("\nIntroduceti al doilea numar:  ");
    scanf("%f",&y);
    if(y==0.0)
        rez=INFINIT;
    else
        rez=x/y;
```

```
    afisare(rez);
    getch();
  }   /*Sfarsit functie principala*/

  /*Definiri de functii*/

   void afisare(float rez)
   {
      if(rez==INFINIT)
          printf("\nRezultat nedefinit\n");
      else
          printf("\nRezultat impartire =  %f\n",rez);
    }
```

4. L5_4.C

```
#include<stdio.h>
 const float INFINIT = 3.4e+38;
void  main()
 {
    float x,y,rez;
    printf(" \nIntroduceti primul numar:  ");
    scanf("%f",&x);
    printf("\nIntroduceti al doilea numar:  ");
    scanf("%f",&y);
    if(y==0.0)
          printf("\nRezultat infinit\n");
    else
      {  rez=x/y;
         if(rez==INFINIT)
             printf("\nRezultat nedefinit\n");
          else
             printf("\nRezultat impartire =  %f\n",x/y);
       }
     getch();
  }
```

5. L5_5.C

```
#include<stdio.h>

 /* Declaratii de functii */
double factorial(int n);

 /* Functia principala, punctul de pornire al programului*/
void  main()
 {
    int x;
    double fact;
    printf("\nIntroduceti un numar intreg : ");
    scanf("%d",&x);
    fact = factorial(x);
    printf("\nFactorialul format(e): %e\n    ",fact);
    printf("\nFactorialul format(f): %f\n    ",fact);
    printf("\nFactorialul format(.0f): %.0f\n",fact);
    printf("\nFactorialul format(d): %d\n    ",fact);
    printf("\nFactorialul format(ld): %ld\n  ",fact);
    printf("\nFactorialul format(g): %g\n    ",fact);
    getch();
  }

double factorial(int n)
/* calculeaza si returneaza pe n! pentru n in intervalul [0,170],
altfel returneaza -1 */
  {
```

```
 double f;
 int i;
 if ( n<0 || n>170)
    return(-1.0);
 for( i=2,f=1; i<=n; i++)
    f*=i;
 return(f);
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 6

ASPECTE ALE LIMBAJULUI C++

I. NOTIUNI TEORETICE

Observatii privind limbajul C++

Limbajul C++ este un superset al limbajului C avand o utilizare universala si care permite scrierea de programe orientate spre obiecte.

Faptul ca limbajul C++ este un superset al limbajului C inseamna ca orice program scris in limbajul C este in acelasi timp si un program scris in limbajul C++. Aceasta afirmatie este adevarata cu mici exceptii, de exemplu, compilatorul C++ face unele controale suplimentare fata de compilatorul C. Controalele in cauza se refera in primul rand la tipurile parametrilor func tiilor, precum si la tipurile valorilor returnate de ele.

Editarea functiilor

Modul de editare al functiilor apelate in cadrul unui program se poate face astfel:

* Functia principala "main" si functiile apelate se scriu in acelasi fisier.
* Functiile pot fi editate in fisiere separate.

Daca functiile sunt editate in fisiere separate avem doua posibilitati de compilare:

1. Folosind directiva #include in programul principal pentru a adauga fisierele de functii, care vor avea extensia .cpp (C++).

2. Utilizarea unui fisier de tip "Project" (cu extensia .prj). Un astfel de fisier, contine numele fiecarui fisier (impreuna cu extensia lui) care se compileaza si link-editeaza in vederea obtinerii fisierului executabil al programului.

In acest fisier pot fi indicate nu numai fisiere de tip sursa (cu extensia .cpp) ci si fisiere de tip obiect (cu extensia .obj) sau chiar fisiere cu biblioteci de functii, altele decat cele ale sistemului.

Obs.: Fisierele de tip "project" pentru limbajele Turbo C si C++ nu sunt compatibile. Ele se construiesc folosind meniul Project al celor doua sisteme integrate de dezvoltare.

Avantajele fisierelor de tip Project constau in aceea ca, la fiecare lansare se compileaza in mod automat numai sursele in care s-au facut modificari. De aceea, in cazul programelor sursa mari se recomanda impartirea lor in mai multe fisiere sursa si compilarea lor prin utilizarea fisierelor de tip Project.

In general, intr-un fisier sursa se grupeaza functii "inrudite", adica functii care prelucreaza in comun subseturi de date sau sunt logic legate intre ele.

NOTA: Deoarece programele de laborator nu sunt de dimensiuni mari, vom utiliza frecvent includerile de fisiere folosind constructia #include a preprocesorului.

II. DESFASURAREA LUCRARII

1. Se va scrie un program care calculeaza si listea pe m! pentru

m = 0, 1, 2, ..., 170.

Programul va apela functia factorial (conceputa in laboratorul precedent) pentru a-l calcula pe m! pentru o valoare data a lui m in intervalul (0, 170). Daca functia factorial de parametru n gaseste n<0 sau n>170 va returna valoarea -1.0 .
Atat functia principala "main" cat si functia factorial se vor gasi in acelasi fisier.

2. Sa se modifice programul de mai sus astfel incat desfasurarea sa se opreasca din 23 in 23 de afisari ale factorialului, iar continuarea derularii sa se faca prin apasarea unei taste.

3. Despartiti programul de la punctul precedent in doua fisiere cu extensia ".cpp" astfel incat unul dintre ele sa contina doar functia factorial iar celalalt programul principal.
Fisierul cu programul principal de pe care va fi lansata compilarea, va include pe langa "stdio.h" fisierul "conio.h" (in care se gaseste prototipul functiei "getch" pentru C++) si fisierul in care se afla functia "factorial". Asadar lista includerilor in fisierul programului principal va contine:

```
#include<stdio.h>
#include<conio.h>
#include "factorial.cpp"
```

Obs.: Toate fisierele ce vor fi alcatuite in continuare vor avea extensia .cpp .

4. Sa se scrie intr-un fisier o functie care are ca parametri doi intregi x si y, calculeaza si returneaza numarul aranjamentelor de x luate cate y.

La inceput functia testeaza daca x apartine intervalului [1,170], iar y intervalului [1, x]. In caz de eroare, functia returneaza valoarea -1.

Formula pentru calculul aranjamentelor este:

$$A(x,y) = x * (x-1) * (x-2) * ... * (x-y+1)$$

5. Sa se scrie un program care calculeaza si afiseaza numarul aranjamentelor de n obiecte luate cate k, pentru:

n = 1, 2, ...30 si k = 1, 2, ... n .

Se vor folosi doua cicluri de "for". Desfasurarea programului va fi stopata, din loc in loc, intr-un mod oarecum asemanator cu cel de la punctul 2., pentru vizualizarea ecranului.
Programul de fata apeleaza functia aranjamente definita in exercitiul precedent.

III. RASPUNSURI:

1. L6_1.C

```
#include<stdio.h>

double factorial(int n);   /* prototipul functiei factorial */

 /* Functia principala, punctul de pornire al programului*/
main()   /* afiseaza pe m! pentru m = 0, 1, 2, ..., 170 */
 {
    int m;

    for(m=0; m<171; m++)
        {
          printf("m=%d\tm!=%g\n",m,factorial(m));
         }
     getch();
  }

double factorial(int n)
/* calculeaza si returneaza pe n! pentru n in intervalul [0,170],
altfel returneaza -1 */
  {
    double f;
    int i;
    if ( n<0 || n>170)
       return(-1.0);
    for( i=2,f=1; i<=n; i++)
       f*=i;
    return(f);
   }
```

2. L6_2.C

```
#include<stdio.h>

double factorial(int n);   /* prototipul functiei factorial */

 /* Functia principala, punctul de pornire al programului*/
main()   /* afiseaza pe m! pentru m = 0, 1, 2, ..., 170 */
 {
    int m;

    for(m=0; m<171; m++)
        {
          printf("m=%d\tm!=%g\n",m,factorial(m));
          if((m+1)%23==0)
                {
                 printf("actionati o tasta pentru a continua\n");
                 getch();
                }
         }
     getch();
  }

double factorial(int n)
/* calculeaza si returneaza pe n! pentru n in intervalul [0,170],
altfel returneaza -1 */
  {
    double f;
    int i;
    if ( n<0 || n>170)
       return(-1.0);
    for( i=2,f=1; i<=n; i++)
```

```
       f*=i;
    return(f);
   }
```

3.1. L6_3_1.CPP

```
double factorial(int n)
/* calculeaza si returneaza pe n! pentru n in intervalul [0,170],
altfel returneaza -1 */
  {
    double f;
    int i;
    if ( n<0 || n>170)
       return(-1.0);
    for( i=2,f=1; i<=n; i++)
       f*=i;
    return(f);
   }
```

3.2. L6_3_2.CPP

```
#include<stdio.h>
#include<conio.h>
#include "l6_3_1.cpp"

main()   /* afiseaza pe m! pentru m = 0, 1, 2, ..., 170 */
 {
    int m;

    for(m=0; m<171; m++)
        {
          printf("m=%d\tm!=%g\n",m,factorial(m));
          if((m+1)%23==0)
                {
                 printf("actionati o tasta pentru a continua\n");
                 getch();
                }
         }
     getch();
  }
```

4. L6_4.CPP

```
double aranjamente(int x, int y)
/* calculeaza si returneaza numarul
aranjamentelor de x luate cate y */

{
 double a;
 int i;

 if(x<1 || x>170)
        return -1.0;
 if(y<1 || y>x)
        return -1.0;
 a=1.0;
 i=x-y+1;
 while(i<=x)
        a*=i++;
 return a;
}
```

5. L6_5.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l6_4.cpp" /* contine functia aranjamente */
```

```
main() /* calculeaza si afiseaza numarul aranjamentelor de
        n obiecte luate cate k, pentru n in intervalul [1,30]
        si k in intervalul [1,n] */

{
 int k, n;

 for(n=1; n<=30; n++)
       {
        printf("actionati o tasta pentru a continua\n");
        getch();
        printf("\nn= %d\n",n);
        for(k=1; k<=n; k++)
         {
          printf("\tk=%d\tA(n,k)=%g\n",k,
                        aranjamente(n,k));
          if(k%20==0)
           {
            printf("actionati o tasta pentru a continua\n");
            getch();
           } /* sfarsit if*/
         }  /* sfarsit for interior */
       }   /* sfarsit for exterior */
 printf("Sfarsitul programului\n");
 getch();
}      /* sfarsit main */
```

LIMBAJE DE PROGRAMARE

Lucrarea 7

CLASE DE MEMORIE. VARIABILE

I. NOTIUNI TEORETICE

1. Clase de memorie (recapitulare, completare)

La compilarea unui text sursa, compilatorul aloca memorie pentru variabilele programului. Se pot aloca variabile pe stiva, in registrii calculatorului, cat si in alte zone de memorie.

Programul defineste, cu ajutorul declaratiilor, modul de alocare al variabilelor. Se pot defini variabile utilizabile (vizibile) in tot programul, precum si variabile care au utilizari locale (cu vizibilitate restransa). Variabilele care pot fi utilizate in tot programul se numesc variabile globale, iar cele care pot fi utilizate numai intr-o anumita parte a programului se numesc variabile locale.

1.1. Variabile globale

Variabilele globale au o definitie si eventual una sau mai multe declaratii de variabila externa.
Definitia unei variabile globale coincide cu o declaratie obisnuita, care insa este scrisa in afara corpului oricarei functii a programului. O astfel de definitie, de obicei, se scrie la inceputul unui fisier sursa. Aceasta deoarece ea este valabila din locul in care este scrisa si pana la sfarsitul fisierului sursa respectiv.

In cazul in care programul se compune din mai multe fisiere sursa, o variabila globala poate fi utilizata intr-un fisier sursa in care nu este definita, daca este declarata ca variabila externa in acel fisier. O declaratie de variabila externa coincide cu o declaratie de variabila obisnuita precedata de cuvantul cheie extern.

Exemplu:
Un program se compune din doua fisiere sursa, f1.cpp si f2.cpp. In fisierul f1.cpp se definesc variabilele globale zi, luna, an de tip int. Aceste variabile se folosesc atat in f1.cpp, cat si in f2.cpp. Daca f1.cpp se include in fisierul f2.cpp, se declara variabilele respective ca externe in functiile din fisierul f2.cpp.

1.2. Variabile locale

Variabilele locale, spre deosebire de cele globale, nu sunt valabile in tot programul. Ele au o valabilitate locala, in unitatea in care au fost declarate.
Variabilele locale pot fi alocate pe stiva. In acest caz ele se numesc automatice. Acestea se declara in mod obisnuit in corpul unei functii sau la inceputul unei instructiuni compuse.

La apelul unei functii, variabilele automatice (declarate, de regula, inaintea primei instructiuni din corpul functiei respective) se aloca pe stiva. In momentul in care se revine din functie, variabilele automatice alocate la apel se elimina si stiva revine la starea de dinaintea apelului (curatirea stivei).
Rezulta ca variabilele automatice isi pierd existenta la revenirea din functia in care sunt declarate. Asadar, o variabila automatica este valabila numai in corpul functiei in care a fost declarata.

Observatie: In acelasi mod se comporta variabilele automatice declarate la inceputul unei instructiuni compuse. O astfel de variabila se aloca pe stiva in momentul in care controlul programului ajunge la instructiunea compusa

in care este declarata variabila respectiva si se elimina de pe stiva in momentul in care controlul programului trece la instructiunea urmatoare celei compuse.

Variabilele locale pot si sa nu fie alocate pe stiva. In acest scop ele se declara ca fiind statice. O declaratie de variabila statica este o declaratie obisnuita precedata de cuvantul cheie static.

Variabilele statice pot fi declarate atat in corpul unei functii cat si in afara corpului oricarei functii. Spre deosebire de variabilele automatice, o variabila statica nu se aloca pe stiva la executie, ci la compilare intr-o zona de memorie destinata acestora.
* O variabila statica declarata in corpul unei functii este definita numai in corpul functiei respective.
* O variabila statica declarata in afara corpurilor functiilor este locala fisierului sursa in care este definita.
Spre deosebire de variabilele globale, o astfel de variabila nu poate fi declarata ca externa. Deci ea nu poate fi utilizata in alte fisiere daca acestea se compileaza separat sau se includ inaintea declaratiei respective.

2. Initializarea
De multe ori se doreste ca unele variabile din program sa aiba valori initiale. Astfel, variabilelor globale li se pot da valori initiale la definirea lor, iar celelalte clase de variabile pot fi initializate la declararea lor.

2.1. Initializarea variabilelor simple
O variabila simpla poate fi initializata printr-o constructie de forma:

```
tip nume = expresie;
```

pentru variabile automatice sau globale, sau:

```
static tip nume = expresie;
```

daca variabila este statica.

Variabilele globale sau statice li se atribuie valori initiale la compilare, deci ele au valorile respective la lansarea programului. Faptul ca aceste variabile se initializeaza la compilare, cere ca expresiile utilizate pentru initializare sa fie constante.

* Variabilele globale si statice neinitializate au in mod implicit valoarea zero.
* O variabila automatica, neinitializata, are o valoare imprevizibila in momentul apelului functiei in al carui corp este declarata. In felul acesta, valoarea ei ramane nedefinita pana in momentul in care i se atribuie o valoare printr-o instructiune de atribuire.

In toate cazurile se fac conversii daca tipul expresiei de initializare nu coincide cu tipul variabilei pe care o initializeaza.

Exemple:
a.
```
int n=10;
```
/* definire de variabila globala. La lansarea programului n are valoarea 10. */

b.
```
#define MAX 100

int i=MAX*2;
```

/* i este variabila globala initializata cu valoarea 200 */
c.
```
int y;
```
/* y este variabila globala si are valoarea 0 la lansarea programului */

d. static int h;

```
/* h este variabila statica si are valoarea 0 la lansarea programului */
```

e.
```
tip functie(int m)
{
 int z=5;
 int a=z+m;
 ...
}
```

```
/*  z si a sunt variabile automatice initializate cu valorile 5,respectiv z+m */
```

II. DESFASURAREA LUCRARII

Obs.: Toate fisierele ce vor fi alcatuite in acest laborator vor avea extensia .cpp .

1. Sa se scrie intr-un fisier o functie

```
double factorial(int n)
```

care calculeaza si returneaza pe n! pentru n in intervalul [0,170]. Daca functia factorial de parametru n gaseste n<0 sau n>170 va returna valoarea -1.0.

Variabilele automatice vor fi initializate la declarare.

2. Sa se scrie intr-un fisier o functie care are ca parametri doi intregi x si y, calculeaza si returneaza numarul aranjamentelor de x luate cate y. La inceput functia testeaza daca x apartine intervalului [1, 170], iar y intervalului [1, x]. In caz de eroare, functia returneaza valoarea -1.

Formula pentru calculul aranjamentelor este:

$$A(x,y) = x * (x-1) * (x-2) * ... * (x-y+1)$$

Variabilele automatice vor fi initializate la declarare.

3. Sa se scrie intr-un fisier o functie care are ca parametri doi intregi x si y, calculeaza si returneaza numarul combinarilor de x luate cate y. La inceput functia testeaza daca x apartine intervalului [1, 170], iar y intervalului [0, x]. In caz de eroare, functia returneaza valoarea -1.
Formula pentru calculul combinarilor este:

$$C(x,y) = A(x,y) / y!$$

In acest scop, functia de fata apeleaza functiile aranjamente si factorial definite in exercitiile precedente. Variabilele automatice vor fi initializate la declarare.
Cum rezolvati cazurile y==0 si y==x ?

4. Sa se scrie un program care calculeaza si afiseaza numarul combinarilor de n obiecte luate cate k, pentru

$$n = 1, 2, ...30 \text{ si } k = 0, 2, ... n .$$

Se vor folosi doua cicluri de "for". Desfasurarea programului va fi stopata, din loc in loc, pentru vizualizarea ecranului. Continuarea derularii programului se va face prin apasarea unei taste. Programul de fata apeleaza functia definita in exercitiul precedent.

5. Un calcul mai rapid al functiei combinari se realizeaza daca se tine seama de relatia:

$$C(x,y) = C(x,x-y)$$

Aceasta relatie este utila sa se aplice pentru
$y > x/2$

Modificati fisierul de la punctul 3., respectand noua idee de calcul si apoi testati cu programul de la punctul 4. corectitudinea modificarilor.

6. O alta varianta pentru calculul numarului combinarilor este relatia:

$$C(x,y) = (x/1) * ((x-1)/2) * ((x-2)/3) * ... * ((x-y+1)/y)$$

In acest caz se evita calculul valorilor A(x,y) si y! care cresc rapid odata cu cresterea valorilor lui x si y.
Modificati fisierul de la punctul 3., respectand noua idee de calcul si apoi testati cu programul de la punctul 4. corectitudinea modificarilor.
Se observa ca pentru aceasta varianta nu mai este cazul sa se apeleze functiile aranjamente si factorial.

7. Conform exemplului din partea de ASPECTE TEORETICE (paragraful 1.1.) creati doua fisiere care contin variabilele comune zi, luna si an de tip int, insa intr-unul din ele fiind variabile globale, iar in celalalt declarate ca externe. Variabilele globale vor fi initializate la declarare.
Fisierul in care variabilele respective sunt declarate ca externe va contine si functia main care va afisa valorile acestor variabile.

III. RASPUNSURI:

1. L7_1.CPP

```
double factorial(int n)
/* calculeaza si returneaza pe n! pentru n in intervalul [0,170],
altfel returneaza -1 */
  {
    double f=1.0;
    int i=2;
    if ( n<0 || n>170)
       return(-1.0);
    while (i<=n)
       f*=i++;
    return(f);
   }
```

2. L7_2.CPP

```
double aranjamente(int x, int y)
/* calculeaza si returneaza numarul
aranjamentelor de x luate cate y */

{
 double a=1.0;
 int i=x-y+1;

 if(x<1 || x>170)
        return -1.0;
 if(y<1 || y>x)
        return -1.0;
 while(i<=x)
```

```
        a*=i++;
 return a;
}
```

3. L7_3.CPP

```
#include "l7_1.cpp"
#include "l7_2.cpp"

double combinari (int x, int y)
/* calculeaza si returneaza numarul combinarilor
de x luate cate y */

{
 if(x<1 || x>170)
    return -1.0;
 if(y<0 || y>x)
    return -1.0;
 if(y==0 || y==x)
    return 1.0;
 return aranjamente(x,y)/factorial(y);
}
```

4. L7_4.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_3.cpp" /* contine functia combinari */

main() /* calculeaza si afiseaza numarul combinarilor de
        n obiecte luate cate k, pentru n in intervalul [1,30]
        si k in intervalul [0,n] */

{
 int k, n;

 for(n=1; n<=30; n++)
       {
        printf("actionati o tasta pentru a continua\n");
        getch();
        printf("\nn= %d\n",n);
        for(k=0; k<=n; k++)
         {
          printf("\tk=%d\tC(n,k)=%g\n",k,
                          combinari(n,k));
          if((k+1)%20==0)
           {
            printf("actionati o tasta pentru a continua\n");
            getch();
           } /* sfarsit if*/
         }  /* sfarsit for interior */
       }   /* sfarsit for exterior */
 printf("Sfarsitul programului\n");
 getch();
}      /* sfarsit main */
```

5.1. L7_5.CPP

```
#include "l7_1.cpp"
#include "l7_2.cpp"

double combinari (int x, int y)
```

```
/* calculeaza si returneaza numarul combinarilor
de x luate cate y */

{
 if(x<1 || x>170)
    return -1.0;
 if(y<0 || y>x)
    return -1.0;
 if(y==0 || y==x)
    return 1.0;
 if(y>x/2)
    return aranjamente(x,x-y)/factorial(x-y);
 else
    return aranjamente(x,y)/factorial(y);
}
```

5.2. L7_5MAIN.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_5.cpp" /* contine functia combinari */

main() /* calculeaza si afiseaza numarul combinarilor de
        n obiecte luate cate k, pentru n in intervalul [1,30]
        si k in intervalul [0,n] */

{
 int k, n;

 for(n=1; n<=30; n++)
       {
        printf("actionati o tasta pentru a continua\n");
        getch();
        printf("\nn= %d\n",n);
        for(k=0; k<=n; k++)
         {
          printf("\tk=%d\tC(n,k)=%g\n",k,
                        combinari(n,k));
          if((k+1)%20==0)
           {
            printf("actionati o tasta pentru a continua\n");
            getch();
           } /* sfarsit if*/
         }  /* sfarsit for interior */
       }   /* sfarsit for exterior */
 printf("Sfarsitul programului\n");
 getch();
}      /* sfarsit main */
```

6.1. L7_6.CPP

```
double combinari (int x, int y)
/* calculeaza si returneaza numarul combinarilor
de x luate cate y */

{
 int i;
 double c=1.0;

 if(x<1 || x>170)
    return -1.0;
 if(y<0 || y>x)
    return -1.0;
 if(y==0 || y==x)
```

```
    return 1.0;
 for(i=1; i<=y; i++)
 c=(c*(x-i+1))/i;
 return c;
}
```

6.2. L7_6MAIN.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_6.cpp" /* contine functia combinari */

main() /* calculeaza si afiseaza numarul combinarilor de
        n obiecte luate cate k, pentru n in intervalul [1,30]
        si k in intervalul [0,n] */

{
 int k, n;

 for(n=1; n<=30; n++)
       {
        printf("actionati o tasta pentru a continua\n");
        getch();
        printf("\nn= %d\n",n);
        for(k=0; k<=n; k++)
         {
          printf("\tk=%d\tC(n,k)=%g\n",k,
                        combinari(n,k));
          if((k+1)%20==0)
           {
            printf("actionati o tasta pentru a continua\n");
            getch();
           } /* sfarsit if*/
         }  /* sfarsit for interior */
       }   /* sfarsit for exterior */
 printf("Sfarsitul programului\n");
 getch();
}      /* sfarsit main */
```

6.1. L7_7_1.CPP

```
int zi=7, luna=5, an=1999;
/* Definiri de variabile globale.
Aceste variabile pot fi folosite in toate
functiile care urmeaza din acest fisier */
```

6.2. L7_7_2.CPP

```
#include <stdio.h>
#include <conio.h>
#include "l7_7_1.cpp"
main()
{
 extern int zi, luna, an;
 printf("\nZiua=%d\tLuna=%d\tAnul=%d\n",zi,luna,an);
 getch();
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 8

TABLOURI

I. NOTIUNI TEORETICE

1. Declaratii de tablou

Un tablou, ca orice variabila simpla, trebuie declarat inainte de a fi utilizat. Declaratia de tablou, in forma cea mai simpla, contine tipul comun elementelor sale, numele tabloului si limitele superioare pentru fiecare indice, incluse intre parantezele patrate:

```
tip nume[lim1][lim2]...[limn];
```

unde:
tip - este un cuvant cheie pentru tipurile predefinite
limi - este limita superioara a indicelui al i-lea; aceasta inseamna ca indicele al i-lea poate avea valorile: 0,1,2,...,limi-1.

Limitele limi (i=1,2,...,n) sunt expresii constante. Prin expresie constanta intelegem o expresie care poate fi evaluata la compilare in momentul intalnirii ei de catre compilator.
La elementele unui tablou se poate face referire folosind variabile cu indici. In limbajul C, numele unui tablou este un simbol care are ca valoare adresa primului sau element.

La intalnirea unei declaratii de tablou, compilatorul aloca o zona de memorie necesara pentru a pastra valorile elementelor sale. Numele tabloului respectiv poate fi utilizat in diferite expresii si valoarea lui este chiar adresa de inceput a zonei de memorie care i-a fost alocata.

Exemple:

a. int vect[10];
Declaratia defineste tabloul vect de 10 elemente care are tipul int. Acestui tablou i se aloca 10*2 = 20 octeti.

vect este un simbol a carui valoare este adresa primului sau element, adica adresa lui vect[0]. Deci vect[0] are ca valoare valoarea primului element al tabloului, iar vect are ca valoare adresa acestui element.

b. char tab[100]

tab este un tablou unidimensional de tip char, care are 100 de elemente. I se aloca 100 de octeti si tab are ca valoare adresa elementului tab[0].

c. double mat[10][20]

mat este un tablou bidimensional de tip double. El reprezinta o matrice de 10 linii a 20 de coloane fiecare. Compilatorul rezerva pentru acest tablou 10*20*8 = 1600 octeti.

La elementele acestui tablou ne referim prin:

```
mat[0][0]  mat[0][1]  ...  mat[0][19]
mat[1][0]  mat[1][1]  ...  mat[1][19]
.....................................

mat[9][0]  mat[9][1]  ...  mat[9][19]
```

mat are ca valoare adresa elementului mat[0][0].

2. Functiile standard sscanf si sprintf

Biblioteca standard a limbajelor C si C++ contine functiile sscanf si sprintf care sunt analoage functiilor scanf si respectiv printf.
Functiile sscanf si sprintf au prototipurile in fisierul stdio.h.
Ele au un parametru in plus la apel si anume primul lor parametru este adresa unei zone de memorie in care se pot pastra caractere ale codului ASCII. Ceilalti parametrii sunt identici cu cei intalniti in corespondentele lor, scanf si respectiv printf.
Primul parametru al acestor functii poate fi numele unui tablou de tip char, deoarece un astfel de nume are ca valoare chiar adresa de inceput a zonei de memorie care ii este alocata.

Functia sprintf se foloseste, ca si functia printf, pentru a realiza conversii ale datelor de diferite tipuri din formatele lor interne, in formate externe reprezentate prin succesiuni de caractere. Diferenta consta in aceea ca, de data aceasta, caracterele respective nu se afiseaza pe terminalul standard, ci se pastreaza in zona de memorie definita de primul parametru al functiei sprintf. Ele se pastreaza sub forma unui sir de caractere si pot fi afisate ulterior din zona respectiva cu ajutorul functiei puts. De aceea, un apel al functiei printf poate fi intotdeauna inlocuit cu un apel al functiei sprintf, urmat de un apel al functiei puts. O astfel de inlocuire este utila cand dorim sa afisam de mai multe ori aceleasi date. In acest caz se apeleaza functia sprintf o singura data pentru a face conversiile necesare din format intern in format extern, rezultatele conversiilor pastrandu-se intr-un tablou de tip caracter. In continuare se pot afisa datele respective apeland functia puts ori de cate ori este necesara afisarea lor.

Functia sprintf, ca si functia printf returneaza numarul octetilor sirului de caractere rezultat in urma conversiilor efectuate.
Exemplu:

```
int zi,lina,an;
char data[20];
...
sprintf(data,"%02d/%02d/%d",zi,luna,an);
puts(data);
...
puts(data);
```

Functia sscanf realizeaza, ca si functia scanf, conversii din format extern in format intern. Deosebirea consta in faptul ca, de data aceasta, caracterele nu sunt citite din zona tampon corespunzatoare tastaturii, ci ele provin dintr-o zona de memorie a carei adresa este definita de primul parametru al functiei sscanf. Aceste caractere pot proveni in zona respectiva in urma apelului functiei gets. In felul acesta, apelul functiei scanf poate fi inlocuit prin apelul functiei gets urmat de apelul functiei sscanf. Astfel de inlocuiri sunt utile cand dorim sa eliminam eventualele erori aparute la tastarea datelor.
Functia sscanf, ca si functia scanf, returneaza numarul campurilor convertite corect conform specificatorilor de format prezenti in parametrul de control. La intalnirea unei erori, ambele functii isi intrerup executia si se revine din ele cu numarul campurilor tratate corect. Analizand valoarea returnata, se poate stabili daca au fost prelucrate corect toate campurile sau a survenit o eroare.

In caz de eroare se poate interveni pentru a introduce corect datele respective. In acest scop este necesar sa se elimine caracterele incepand cu cel din pozitia eronata.

Daca se utilizeaza secventa:

```
gets
sscanf
```

abandonarea caracterelor necorespunzatoare se face automat reapelandu-se functia gets. In cazul utilizarii functiei scanf este necesar sa se avanseze pana la caracterul newline aflat in zona tampon atasata tastaturii sau sa se videze zona respectiva prin functii speciale.

Exemplu:

```
        char tab[255];
        int zi,luna,an;
        ...
        gets(tab);
        sscanf(tab,"%d %d %d", &zi, &luna, &an);
```

Mentionam ca functia gets returneaza valoarea NULL la intalnirea sfarsitului de fisier.

II. DESFASURAREA LUCRARII

1. Se va scrie un program, utilizand rutina sscanf, care citeste un intreg pozitiv de tip long si apoi il afiseaza. In cazul tastarii unor caractere nenumerice se va afisa un mesaj de eroare si se va cere introducerea, din nou, a unui intreg, pana la o introducere corecta.

O posibila solutie:

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

void main()
{
  long n,i;
  int j;
  char tab[255];
  do
    {  printf("\nTastati un intreg pozitiv :");
       if(gets(tab)==NULL)
        { printf("s-a tastat EOF\n");
          exit(1);
        }
       if(sscanf(tab,"%ld",&n)!=1 || n<=0)
       /* valorile zecimale sau nenumerice sunt truncheate
       prin conversia de citire %ld */
        { printf("Nu s-a tastat un intreg pozitiv\n");
          j=1;
        }
       else
          j=0;
    }
  while(j);
  printf("%ld",n);
  getch();
}
```

Observatie: In cazul tastarii unor caractere nenumerice, daca acestea sunt precedate de caractere numerice, cele nenumerice vor fi ignorate.

2. Sa se modifice programul precedent astfel incat sa se inlocuiasca instructiunea "else" cu o instructiune "continue" plasata adecvat. De asemenea, ultimul printf sa fie substituit prin sprintf fara a afecta desfasurarea initiala a programului.

3. Pornind de la programul scris la punctul 2., completati-l astfel incat sa poata determina daca numarul introdus este prim si sa afiseze un mesaj corespunzator.

Indicatie: Un algoritm simplu de calcul ar putea fi impartirea succesiva a numarului n cu 2, 3, 4, ..., k unde k*k <= n.

Daca cel putin unul dintre aceste numere divide pe n, rezulta ca n nu este prim.

4. Alcatuiti o functie (intr-un fisier cu extensia .c) declarata:

```
void ordcresc(double tab[], int n)
```

care sorteaza elementele, in numar de n ale sirului tab, in ordine crescatoare.

Indicatie: Se parcurge sirul si daca

```
tab[i] > tab[i+1]
```

se inverseaza ordinea lor. Se reia parcurgerea de atatea ori pana cand nu se mai realizeaza nici o inversie.

5. Sa se completeze fisierul de mai sus cu un program care citeste un sir de numere si le afiseaza in ordine crescatoare.

6. Sa se modifice programul de mai sus astfel incat acesta sa ceara, la inceput, utilizatorului o parola de un caracter care, daca nu este tasta corect, sa forteze iesirea din program.

III. RASPUNSURI:

2. L8_2.C

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

void main()
{
  long n,i;
  int j;
  char tab[255];
  char data[20];
  do
    {  printf("\nTastati un intreg pozitiv :");
       if(gets(tab)==NULL)
        { printf("s-a tastat EOF\n");
          exit(1);
        }
       if(sscanf(tab,"%ld",&n)!=1 || n<=0)
        { printf("Nu s-a tastat un intreg pozitiv\n");
          j=1;
          continue;
        }
       j=0;
    }
  while(j);
  sprintf(data,"%ld",n);
  puts(data);
  getch();
}
```

3. L8_3.C

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

void main()
{
  long n,i;
  int j;
  char tab[255];

  do
    {  printf("\nTastati un intreg pozitiv :");
       if(gets(tab)==NULL)
        { printf("s-a tastat EOF\n");
          exit(1);
        }
       if(sscanf(tab,"%ld",&n)!=1 || n<=0)
        { printf("Nu s-a tastat un intreg pozitiv\n");
          j=1;
          continue;
        }
       j=0;
    }
  while(j);
  for(j=1,i=2;i*i<=n && j ;i++)
     if(n%i==0)     /*numarul nu este prim*/
       j=0;
  printf("Numarul : %ld",n);
  if(j==0)
    printf(" nu ");
  printf(" este prim\n");
  getch();
}
```

4. L8_4.C

```
 void ordcresc(double tab[],int n)
 /* ordoneaza elementele lui tab in ordine crescatoare*/
 {
   int i,ind;
   double t;
   ind=1;
   while(ind)
     {
       ind=0;
       for(i=0;i<n-1;i++)
        if(tab[i]>tab[i+1])
         { t=tab[i];
           tab[i]=tab[i+1];
           tab[i+1]=t;
           ind=1;
         }   /*sfarsit if*/
     }  /*sfarsit while*/
 }
```

5. L8_5.C

```
#include <stdio.h>
#define MAX 1000
```

```
void ordcresc(double tab[],int n);

void main()
{
  double v[MAX];
  int m,s,i;
  printf("\nIntroduceti nr. de elemente = ");
  scanf("%d",&m);
  printf("\nIntroduceti elementele sirului :  \n");
  for(s=0 ; s<m ; s++)
    scanf("%lf",&v[s]);
    ordcresc(v,m);
    for(i=0;i<m;i++)
    {  printf("v[%d]=%g\n",i,v[i]);
       if((i+1)%23==0)
         {  printf("Actionati o tasta pentru a continua\n");
                getch();
         }
    }
    getch();
}


 void ordcresc(double tab[],int n)
 /* ordoneaza elementele lui tab in ordine crescatoare*/
 {
   int i,ind;
   double t;
   ind=1;
   while(ind)
     {
       ind=0;
       for(i=0;i<n-1;i++)
        if(tab[i]>tab[i+1])
         { t=tab[i];
           tab[i]=tab[i+1];
           tab[i+1]=t;
           ind=1;
         }   /*sfarsit if*/
     }  /*sfarsit while*/

 }
```

6. L8_6.C

```
#include <stdio.h>
#define MAX 1000

void ordcresc(double tab[],int n);

void main()
{
  double v[MAX];
  int m,s,i;
  char par;

  printf("\nParola:");
  scanf("%c",&par);
  if(par != 's')
        exit(1);
  printf("\nIntroduceti nr. de elemente = ");
  scanf("%d",&m);
  printf("\nIntroduceti elementele sirului :  \n");
```

```
  for(s=0 ; s<m ; s++)
    scanf("%lf",&v[s]);
    ordcresc(v,m);
    for(i=0;i<m;i++)
    {  printf("v[%d]=%g\n",i,v[i]);
       if((i+1)%23==0)
         {  printf("Actionati o tasta pentru a continua\n");
                getch();
         }
    }
    getch();
}


 void ordcresc(double tab[],int n)
 /* ordoneaza elementele lui tab in ordine crescatoare*/
 {
   int i,ind;
   double t;
   ind=1;
   while(ind)
     {
       ind=0;
       for(i=0;i<n-1;i++)
        if(tab[i]>tab[i+1])
         { t=tab[i];
           tab[i]=tab[i+1];
           tab[i+1]=t;
           ind=1;
         }   /*sfarsit if*/
     }  /*sfarsit while*/
 }
```

LIMBAJE DE PROGRAMARE

Lucrarea 9

TABLOURI (partea II)

I. NOTIUNI TEORETICE

Initializarea tablourilor
Tablourile, ca si variabilele simple, pot fi initializate. Tablourile globale se initializeaza prin definitiile lor. Tablourile statice si automatice se initializeaza prin declaratiile lor.
Obs.: In limbajul "C", in toate cazurile, initializarile se fac prin expresii constante.
Tablourile globale si statice se initializeaza la compilare. De aceea, la lansarea programului, elementele lor au ca valori, valorile expresiilor cu care au fost initializate.

Tablourile automatice se initializeaza la executie, de fiecare data cand se apeleaza functia in care sunt declarate. Exista, insa, si unele versiuni ale limbajului "C" care nu pot initializa tablouri automatice.

Limbajul Turbo C permite initializarea tablourilor automatice.

* Un tablou unidimensional se initializeaza printr-o constructie de forma:

```
        tip nume[val] = { ec0, ec1, ec2, ..., eci };
```
sau:
```
        static tip nume[val] = { ec0, ec1, ec2, ..., eci };
```

daca tabloul este static.
Cu val, ec0, ec1, ..., eci am notat expresii constante.
Obs.:
a. Obligatoriu, i <= val - 1. Daca i >= val, constructia este eronata.

b. Daca i < val - 1, atunci elementele:

```
        nume[i+1], nume[1+2], ..., nume[val-1]
```

raman neinitializate.

Elementele neinitializate ale unui tablou global sau static au in mod implicit valoarea initiala egala cu zero.
Elementele neinitializate ale unui tablou automatic au valori initiale nedefinite.
In cazurile in care se initializeaza toate elementele unui tablou unidimensional, se poate omite expresia din parantezele patrate care defineste numarul elementelor tabloului.

Asadar constructiile:

```
        tip nume[] = { ec0, ec1, ec2, ..., ecn };
```
si:
```
        static tip nume[] = { ec0, ec1, ec2, ..., ecn };
```

sunt corecte si in ambele cazuri tablourile au n+1 elemente, iar elementul nume[i] este initializat cu valoarea expresiei eci.
Ca si in cazul variabilelor simple, daca expresia de initializare are un tip diferit de cel al tabloului, atunci valoarea ei se converteste spre tipul tabloului inainte de a fi atribuita elementului pe care-l initializeaza.

Precizare:
Intr-o declaratie sau definitie de tablou unidimensional se poate omite expresia care defineste numarul elementelor tabloului in urmatoarele cazuri:

1. Declaratia de tablou contine expresii constante pentru initializarea fiecarui element al tabloului.

2. Declaratia se refera la un tablou unidimensional care este parametru formal:

```
void functie(int n, double tab[])
```

3. Declaratie de tablou extern unidimensional:

```
extern int tab[];
```

Exemple:
1. `int sir[5] = { 1, 6, 3, 9, 7};`
2. `int sir[] = { 1, 6, 3, 9, 7};`
Aceasta definire este identica, ca urmari, cu cea precedenta.
3. `double tab[10] = { 9, 3, 4 };`
Primele 3 elemente ale tabloului tab sunt initializate:
`tab[0] = 9`
`tab[1] = 3`
`tab[2] = 4`

Restul de elemente ale tabloului vor avea valoarea initiala zero daca tab este un tablou global, sau o valoare imprevizibila daca tabloul este automatic.
Deoarece tablourile de tip char sunt utilizate frecvent, a fost introdusa o simplificare pentru initializarea tablourilor de acest tip.
Astfel, pentru un tablou de tip char se poate utiliza una din constructiile:

```
char nume[val] = sir;
```
sau:
```
static char nume[val] = sir;
```

unde sir reprezinta o succesiune de caractere delimitata prin ghilimele.

Astfel, elementele tabloului nume se initializeaza cu codurile ASCII ale caracterelor din compunerea sirului de caractere sir, iar dupa ultimul caracter se memoreaza caracterul NUL (\0).

* Tablourile multidimensionale pot fi initializate, si ele, prin constructii analoge cum ar fi:

```
tip nume[n][m] =
        {
          { ec11, ec12, ..., ec1m },
          { ec21, ec22, ..., ec2m },
          ....
          { ecn1, ecn2, ..., ecnm }
        };
```
unde n, m, ecij (i=1,2,...,n si j=1,2,...,m) - sunt expresii constante.

Numarul expresiilor constante poate fi mai mic decat m in oricare din acoladele corespunzatoare celor n linii ale tabloului bidimensional.
Pentru tablouri statice, declaratia este precedata de cuvantul cheie static.
Precizare:
Intr-o declaratie sau definitie de tablou multidimensional se poate omite numai expresia constanta din prima paranteza patrata. Ea poate fi omisa in una din urmatoarele cazuri:

1. La declaratiile sau definitiile care contin initializari;
2. La declaratiile de parametrii;

3. La declaratii de extern.

Exemple:
```
1. int mat[3][4] =
        {
          { 3, 4, 5, 2 },
          { 5, 8, 1, 3 },
          { 1, 9, 3, 2 }
        };
```

Tabloul mat se va initializa astfel:

```
mat[0][0]=3  mat[0][1]=4  mat[0][2]=5  mat[0][3]=2
mat[1][0]=5  mat[1][1]=8  mat[1][2]=1  mat[1][3]=3
mat[2][0]=1  mat[2][1]=9  mat[2][2]=3  mat[2][3]=2
```

```
2.  int mat[][4] =
        {
          { 3, 4, 5, 2 },
          { 5, 8, 1, 3 },
          { 1, 9, 3, 2 }
        };
```

Declaratie identica, ca efect, cu cea din exemplul precedent.

```
3. float t[][3] =
        {
          { 0, 1 },
          { -3 },
          { 1, 2, 4}
        };
```

II. DESFASURAREA LUCRARII

1. Realizati un program care sa efectueze inmultirea matricilor:

$$TAB = \begin{vmatrix} -1 & 0 & 1 & -1 \\ -1 & 0 & 1 & 2 \\ 10 & 20 & 40 & 30 \end{vmatrix} \quad si \quad MC = \begin{vmatrix} 1 \\ 2 \\ 3 \\ 4 \end{vmatrix}$$

O posibila solutie:

```
# include<stdio.h>

void main()

{
double tab[3][4]=
        { {-1, 0, 1, -1},
          {-1, 0, 1, 2},
          {10, 20, 40, 30}
        };
double mc[4]={1,2,3,4};
double rez[3];
int i, j;

for(i=0; i<3; i++)
        {
          rez[i]=0;
```

```
          for(j=0; j<4; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\n rez[%d]=%g", i, rez[i]);
         }
getch();
}
```

2. Modificati programul de mai sus astfel incat tablourile sa fie definite variabile globale. Stergeti expresiile care pot fi omise din definirea tablourilor.

3. Modificati programul de mai sus astfel incat sa se realizeze afisarea matricilor care se inmultesc, precum si a rezultatului inmultirii.

4. Realizati un program care efectueaza inmultirea unei matrici m x n cu o matrice coloana n x 1 si afiseaza rezultatul. Elemen tele matricilor (de tip double) se vor introduce de la tastatura.

5. In programul de mai sus extrageti functia de inmultire a matricilor din corpul functiei "main" (programare procedurala).

III. RASPUNSURI:

2. L9_2.C

```
# include<stdio.h>

double tab[][4]=
        { {-1, 0, 1, -1},
          {-1, 0, 1, 2},
          {10, 20, 40, 30}
        };
double mc[]={1,2,3,4};
double rez[3];

void main()

{

int i, j;

for(i=0; i<3; i++)
        {
          rez[i]=0;
          for(j=0; j<4; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\n rez[%d]=%g", i, rez[i]);
         }
getch();
}
```

3. L9_3.C

```
# include<stdio.h>

double tab[3][4]=
        { {-1, 0, 1, -1},
```

```
          {-1, 0, 1, 2},
          {10, 20, 40, 30}
        };
double mc[4]={1,2,3,4};
double rez[3];


void main()

{

int i, j;
printf("\n\nRezultatul inmultirii matricii\n\nTAB=");
for(i=0;i<3;i++)
      {
        for(j=0; j<4; j++)
                printf("\t%g", tab[i][j]);
        printf("\n");
      }
printf("\ncu matricea coloana \n\nMC=");
for(j=0; j<4; j++)
        printf("\t%g\n",mc[j]);
printf("\neste matricea linie\n\nREZ=");
for(i=0; i<3; i++)
        {
          rez[i]=0;
          for(j=0; j<4; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\t%g", rez[i]);
         }
printf("\n\n");
getch();
}
```

4. L9_4.C

```
# include<stdio.h>
double tab[20][20];
double mc[20];
double rez[20];


void main()

{
int i, j, m, n;

printf("\nIntroduceti numarul de linii ale primei matrici:");
scanf("%d",&m);
printf("\nIntroduceti numarul de coloane ale primei matrici:");
scanf("%d",&n);
printf("\nIntroduceti elementele primei matrici:\n");

for(i=0; i<m; i++)
        {
          for(j=0; j<n; j++)
             {
              printf("\ttab[%d][%d]=", i, j);
              scanf("%lf",&tab[i][j]);
             }
          printf("\n");
         }
```

```
printf("\nIntroduceti elementele matricii coloana:\n");

for(j=0; j<n; j++)
        {
          printf("\tmc[%d]=", j);
          scanf("%lf",&mc[j]);
        }

printf("\nRezultatul inmultirii celor doua matrici este:\n");

for(i=0; i<m; i++)
        {
          rez[i]=0;
          for(j=0; j<n; j++)
                rez[i] += tab[i][j]*mc[j];
          printf("\t rez[%d]=%g", i, rez[i]);
         }
getch();
}
```

5. L9_5.C

```
# include<stdio.h>

double tab[20][20];
double mc[20];
double rez[20];


void matrez(int m, int n, double m1[][],
                double m2[], double prod[]);


void main()

{
int i, j, m, n;

printf("\nIntroduceti numarul de linii ale primei matrici:");
scanf("%d",&m);
printf("\nIntroduceti numarul de coloane ale primei matrici:");
scanf("%d",&n);
printf("\nIntroduceti elementele primei matrici:\n");

for(i=0; i<m; i++)
        {
          for(j=0; j<n; j++)
             {
              printf("\ttab[%d][%d]=", i, j);
              scanf("%lf",&tab[i][j]);
             }
          printf("\n");
         }

printf("\nIntroduceti elementele matricii coloana:\n");

for(j=0; j<n; j++)
        {
          printf("\tmc[%d]=", j);
          scanf("%lf",&mc[j]);
        }

printf("\nRezultatul inmultirii celor doua matrici este:\n");
```

```
matrez(m, n, tab, mc, rez);

printf("\nREZ = [\t");
for(i=0; i<m; i++)
        printf("%g\t", rez[i]);
printf("]\n");
getch();
}

void matrez(int m, int n, double m1[20][20], double m2[20],
                        double prod[20])

{
 int i, j;
 for(i=0; i<m; i++)
        {
          prod[i]=0;
          for(j=0; j<n; j++)
                prod[i] += m1[i][j]*m2[j];
         }
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 10

POINTERI

I. NOTIUNI TEORETICE

Dupa cum se stie, un program impreuna cu datele sale este pastrat in memoria calculatorului. Memoria RAM (Random Access Memory) este o memorie cu acces aleator si care, la nivelul cel mai inferior, este alcatuita din biti ce memoreaza o valoare din doua, interpretate de obicei ca 0 si 1, atat timp cat calculatorul este in functiune.
Opt biti formeaza un octet, doi octeti alcatuiesc un cuvant, iar patru octeti un cuvant lung.
Un pointer este o variabila care contine adresa altei variabile, sau altfel zis reprezinta o variabila care pastreaza adresa unei date, in loc de a memora data insasi.
Pointerii se utilizeaza pentru a face referire la date cunoscute prin adresele lor.

1. Declaratia de pointer si tipul pointer

Un pointer se declara ca orice variabila, cu precizarea ca numele este precedat de caracterul *.
In general, un pointer se declara prin:

```
tip *nume_p;
```

ceea ce inseamna ca nume_p este un pointer care pointeaza spre o zona de memorie ce contine o data de tipul tip.

Comparand declaratia de mai sus cu cea uzuala:

```
tip nume;
```

putem considera ca tip* dintr-o declaratie de pointeri reprezinta tip dintr-o declaratie obisnuita.
Asadar, putem spune ca ┐tip*┐ reprezinta un tip nou, tipul pointer. Acest tip se spune ca este tipul pointer spre tip.

Precizare
Amintim faptul ca:
* -> operatorul de indirectare. Expresia care-l urmeaza este un pointer iar rezultatul este o lvaloare.
& -> operatorul de obtinere a unui pointer. Operandul este lvaloare iar rezultatul este pointer.

Daca avem declaratiile:

```
int x;
int *p;
float y;
```

atunci atribuirea:

```
p = &x;
```

este corecta, in timp ce:

```
p = &y;
```

nu este corecta, deoarece p poate contine numai adrese de zone de memorie in care se pastreaza date de tip int.

Daca exista declaratia:

```
float *q;
```

atunci se poate folosi atribuirea:

```
q = &y;
```

Exista cazuri in care se doreste ca un pointer sa fie utilizat cu mai multe tipuri de date. In acest scop, la declararea lui nu putem specifica un tip. Aceasta se realizeaza folosind cuvantul void:

```
void *nume_p;
```

Exemplu:

```
int x;
float y;
char c;
void *p;
...
p = &x;
...
p = &y;
...
p = &c;
...
```

Folosindu-ne de cuvantul cheie ┐void┐, lui p i s-au putut atribui adrese de zone de memorie care contin date de tipuri diferite.
Cand se folosesc pointeri de tip void, este necesar sa se faca conversii explicite prin expresii de tip ┐cast┐ (valoarea unui operand se converteste catre tipul tip folosind operatorul unar (tip)), pentru a preciza tipul datei spre care pointeaza un astfel de pointer:

(tip) operand

Astfel, daca se utilizeaza p declarat ca in exemplul de mai sus, atunci o atribuire de forma:

```
*p = 100;
```

nu este corecta, fiindca nu este definit tipul datei spre care pointeaza p. In cazul de fata, valoarea lui p trebuie convertita spre tipul int* folosind expresia cast:

```
(int*)p
```

In felul acesta atribuirea de mai sus devine:

```
*(int*)p = 100;
```

2. Functii necesare in programe ce opereaza cu pointeri

2.1. Functia "malloc"

Este declarata in fisierul header alloc.h .
Functia:

```
malloc(val)
```

"aduna" val octeti consecutivi din memoria disponibila, returnand adresa lor de inceput.

2.2. Expresia "sizeof"

```
sizeof(tip)
```

retuneaza numarul de octeti necesari unei variabile de tipul tip.
De exemplu sizeof(int) returneaza valoarea 2.

2.3. Functia "calloc"

Este declarata in fisierul header alloc.h .

Functia are doi parametri:

```
calloc(nr, dimens)
```

Primul parametru indica pentru cate obiecte se va aloca spatiu, iar al doilea dimensiunea fiecarui obiect in octeti.

Ex.:

```
int *sir;
...
sir = (int*)calloc(5, sizeof(int));
```

"sir" pointeaza la o zona de memorie suficienta pentru a contine 5 intregi (5 x 2 = 10 octeti).
Expresia (int*) indica faptul ca adresa va fi un pointer de tipul int, reprezentand o conversie de tip (cast). Expresia nu e necesara in Turbo C, ea fiind scrisa pentru portabilitatea programului.

II. DESFASURAREA LUCRARII

1. Scrieti un program care utilizeaza doua variabile intregi v si p, dintre care ultima este de tipul pointer. Lui v i se atribuie valoarea 200, iar lui p adresa variabilei v. Sa se afiseze valorile, locatia de memorie si valoarea pointata de cele doua variabile.

2. Sa se scrie un program care utilizeaza o singura variabila de tip pointer intreg p pentru care este alocata o zona de memorie cu functia malloc. Pentru o anumita marime aleasa afisati valoarea lui p si valoarea pointata de p.

3. Fie doua variabile intregi a si b initializate cu doua valori arbitrar alese. Prin intermediul altor variabile de tip pointer realizati un program care sa inverseze valorile celor doua variabile.

4. Conform principiilor programarii procedurale extrageti functia:

```
void invers(int *x, int *y)
```

din corpul functiei main definita la punctul anterior.

5. Realizati un program care cu ajutorul functiei calloc aloca spatiu pentru o variabila pointer de dimensiunea a 3 variabile de tip int. Initializati, apoi afisati atat adresele cat si continutul locatiilor de memorie.

6. Scrieti un program in care functia main apeleaza o functie de citire

```
void citire(float *p)
```

a unei valori reale de la tastatura, care apoi va fi si afisata.

III. RASPUNSURI

1. L10_1.C

```
# include <stdio.h>

void main()
{
  int v, *p;
  p = &v;
  v = 200;
  printf("\nLocatia de memorie a lui v:  %p\n", &v);
  printf("Valoarea lui v:                %d\n", v);
  printf("Valoarea lui p:                %p\n", p);
  printf("Valoarea pointata de p:        %d\n", *p);
  getch();
}
```

2. L10_2.C

```
# include <stdio.h>
# include <alloc.h>

void main()
{
  int *p;
  p = (int*)malloc(sizeof(int));
  *p = 200;
  printf("\nValoarea lui p:              %p", p);
  printf("\nValoarea pointata de p:       %d\n", *p);
  getch();
}
```

3. L10_3.C

```
# include <stdio.h>

void main()

{
   int a=100, b=123, s, *x, * y;
   printf("\nValorile de inceput pentru a si b
                sunt:\na=%d\nb=%d\n",a,b);
   x = &a;
   y = &b;
   s = *x;
   *x = *y;
   *y = s;
   printf("\nValorile dupa inversare pentru a si b
                sunt:\na=%d\nb=%d\n",a,b);
   getch();
}
```

4. L10_4.C

```
# include <stdio.h>

void invers(int *x, int *y);

void main()
{
   int a=100, b=123;
   printf("\nValorile de inceput pentru a si b
                sunt:\na=%d\nb=%d\n",a,b);
   invers(&a, &b);
   printf("\nValorile dupa inversare pentru a si b
                sunt:\na=%d\nb=%d\n",a,b);
   getch();
}

void invers(int *x, int *y)
{
   int s;
   s = *x;
   *x = *y;
   *y = s;
}
```

5. L10_5.C

```
# include <stdio.h>
# include <alloc.h>

void main()
{
  int i, *sir;
  sir = (int*)calloc(3,sizeof(int));
  *sir = 10;
  *(sir+1) = 20;
  *(sir+2) = 30;
  printf("\n\nLa adresele:");
  for(i=0; i<3; i++)
        printf("\n%p", (sir+i));
```

```
  printf("\nAvem valorile:");
  for(i=0; i<3; i++)
        printf("\n%d", *(sir+i));
  getch();
}
```

6. L10_6.C

```
# include <stdio.h>

void citire(float *p);

void main()
{
  float x;
  citire(&x);
  printf("\nValoarea atribuita lui x: %g",x);
  getch();
}

void citire(float *p)
{
   printf("\n\nIntroduceti o valoare reala: ");
   scanf("%f",p);
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 11

STRUCTURI DE DATE IN LIMBAJUL "C"

I. NOTIUNI TEORETICE

1. Introducere

Pe parcursul alcatuirii de programe apare de multe ori necesitatea prelucrarii grupate a mai multor date. Datele se grupeaza pentru a forma multimi de elemente care sa poata fi prelucrate atat element cu element cat si global. De regula, aceste grupe sunt multimi ordonate de date, adica datele unei astfel de grupe satisfac anumite relatii.
Cel mai simplu mod de grupare al datelor este tabloul.
Tabloul este o multime ordonata de date de un acelasi tip, relatia de ordine intre elementele sale fiind definita cu ajutorul indicilor (care determina si dimensiunea tabloului). Tipul comun elementelor tabloului este si tipul tabloului.

De multe ori este necesar a se grupa date, care nu sunt neaparat de acelasi tip, potrivit unei ierarhii. Datele grupate conform unei ierarhii se numesc structuri.
Un exemplu foarte simplu de structura este data calendaristica. Ea grupeaza trei date elementare: zi, luna si an. Componentele zi si an sunt date de tip intreg, iar componenta luna poate fi un sir de caractere.

2. Declaratia de structura

Formatul cel mai utilizat pentru a declara o structura este:

```
struct nume
        {
           lista_de_declaratii
        } nume1, nume2, ..., numen;
```

unde: nume, nume1, ..., numen sunt niste denumiri pentru identificare care pot lipsi, dar nu toate deodata.

Obs.:
* Daca nume este absent, atunci cel putin nume1 trebuie sa fie prezent.
* Daca lista nume1, nume2, ..., numen este absenta, atunci trebuie ca nume sa fie prezent.

Daca nume este prezent, atunci el va defini un tip nou, introdus prin declaratia de structura respectiva. Astfel nume1, nume2, ..., numen sunt structuri de tipul nume.

O structura de tip nume poate fi declarata si ulterior, utilizand formatul:

```
struct nume numele_noii_structuri;
```

Exemple:
a. Se va declara data_nasterii si data_angajarii ca structuri de tipul data_calendaristica (compusa din zi, luna, an):

```
struct data_calendaristica
        {
           int zi;
           char luna[10];
           int an;
        } data_nasterii, data_angajarii;
```

b. Se poate excepta introducerea data_calendaristica:

```
        struct
                {
                   int zi;
                   char luna[10];
                   int an;
                } data_nasterii, data_angajarii;
```

c. Putem defini tipul utilizator data_calendaristica si ulterior sa declaram data_nasterii si data_angajarii:

```
     struct data_calendaristica
                {
                   int zi;
                   char luna[10];
                   int an;
                } ;
        ...
     struct data_calendaristica  data_nasterii, data_angajarii;
```

Prima declaratie introduce tipul utilizator data_calendaristica, iar declaratia a doua defineste datele data_nasterii si data_angajarii ca fiind structuri de tipul data_calendaristica.

Extrapoland ideile de mai sus se poate usor defini o structura de date personale ale angajatilor unei institutii cuprinzad date ca: nume, prenume, adresa, locul nasterii, data nasterii, data angajarii, studii, sex, etc.

3. Accesul la componentele unei structuri
In general accesul la componentele unei structuri se realizeaza prin constructii de forma:

```
        nume.nume_data
```
sau:
```
        pointer -> nume_data
```

unde:

```
nume       - este numele structurii;
nume_data  - este numele componentei;
pointer    - este un pointer spre structura.
```

4. Declaratii de tip
In limbajul C se poate atribui un nume unui tip, indiferent daca el este un tip predefinit sau unul utilizator, utilizand o constructie de forma:

```
        typedef tip nume_tip;
unde:   tip       - este un tip predefinit sau un tip utilizator;
        nume_tip - numele care se atribuie tipului definit de tip;
```

Exemplu:
Prin declaratia

```
                typedef double REAL;
```
datele
```
                REAL x, y;
```
sunt de tip double.

II. DESFASURAREA LUCRARII

Nota: Primele doua puncte vor fi programe scrise in Turbo C si vor avea extensia .c, iar fisierele alcatuite de la punctul 3 pana la sfarsit, fiind in C++, vor avea extensia .cpp.

1. Scrieti un program care citeste numere complexe de la tastatura si afiseaza modulul lor. Se va face o definire globala typedef pentru numere complexe (introduse ca o structura).
Pentru un numar complex

$$z = x + i*y$$

modulul este radacina patrata din x*x + y*y. Functia de extragere a radacinii patrate este ┐sqrt┐ si este definita in fisierul header math.h. Modulul numarului complex va fi calculat prin intermediul unei functii.
Programul va citi si va returna module de numere complexe pana cand se va incerca introducerea unei valori nenumerice de la tastatura, moment in care executia programului va lua sfarsit.

O posibila solutie:

```
#include<stdio.h>
#include<math.h>

typedef struct {
                 double x;
                 double y;
                } COMPLEX;
double dmodul(COMPLEX *z);


void main()  /*citeste numere complexe si afiseaza modulul lor */
{
 COMPLEX complex;
 printf("\nIntroduceti partea reala si partea imaginara ");
 printf("\n ale numarului complex z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
  {
   printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
   printf("modul=%g \n",dmodul(&complex));
   printf("\n\nIntroduceti partea reala si partea imaginara");
   printf("\n ale numarului complex z = a + ib :");
   printf("\n(Valori nenumerice incheie executia programului)\n");
  }
}

double dmodul(COMPLEX *z)
/*calculeaza si returneaza modulul numarului complex z*/
{
  return sqrt(z->x * z->x + z->y * z->y);
}
```

2. Completati programul de mai sus cu o functie care sa calculeze si argumentul numarului complex.
Daca

$$z = x + i*y$$

atunci

$$\arg z$$

se calculeaza astfel:

a. Daca x = y = 0, atunci arg z = 0.
b. Daca y = 0 si x != 0

atunci: daca x > 0, arg z = 0;
        altfel arg z = pi = 3.1415926535.
c. Daca x = 0 si y != 0
atunci: daca y >0, arg z = pi/2;
        altfel arg z = 3*pi/2.
d. Daca x != 0 si y != 0 atunci
fie
        a = arctg(y/x)

Daca   x > 0 si y > 0, atunci arg z = a;
        x > 0 si y < 0, atunci arg z = 2*pi + a;
        x < 0, atunci arg z = pi + a.

Printr-un #define va fi introdusa initial si valoarea lui pi in program. Functia arctg se gaseste sub numele ┐atan┐ in fisierul math.h.

3. Desfaceti programul de mai sus in trei fisiere cu extensia .cpp urmand principiile cunoscute si apoi lansati compilarea in fisierul ce contine functia main.

4. Scrieti intr-un fisier o functie

        void sum_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)

care atribuie lui c suma numerelor complexe a si b.

5. Scrieti intr-un fisier o functie

        void scad_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)

care atribuie lui c diferenta numerelor complexe a si b.

6. Scrieti intr-un fisier o functie

        void mul_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)

care atribuie lui c produsul numerelor complexe a si b.

7. Scrieti intr-un fisier o functie

        void div_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)

care atribuie lui c rezultatul impartirii numerelor complexe a si b. Ce masuri trebuie sa luati?

8. Scrieti un program care cere introducerea a doua numere complexe si afiseaza rezultatele adunarii, diferentei, inmultirii si impartirii lor. Programul apeleaza fisierele scrise la punctele 4 - 7.

III. RASPUNSURI:

2. L11_2.C

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159265358979

typedef struct {
                 double x;
                 double y;
                } COMPLEX;
double dmodul(COMPLEX *z);
double darg(COMPLEX *z);


void main()
 /*citeste nr. complexe, afiseaza modulul si argumentul*/
{
 COMPLEX complex;
 printf("\nIntroduceti partea reala si partea imaginara ");
 printf("\n ale numarului complex z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
     {
       printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
       printf("modul=%g \t arg=%g \n",
      dmodul(&complex),darg(&complex));
       printf("\n\nIntroduceti partea reala si partea");
       printf("\nimaginara ale numarului complex z = a + ib :");
       printf("\n(Valori nenumerice incheie executia programului)\n");
     }
}


double dmodul(COMPLEX *z)
/*calculeaza si returneaza modulul numarului complex z*/
{
  return sqrt(z->x * z->x + z->y * z->y);
}

double darg(COMPLEX *z)
{
  double a;

  if(z->x==0 && z->y==0)
     return 0.0 ;
  if(z->y==0)
     if(z->x > 0)
        return 0.0;
 else /* y=0 si x<0 */
        return PI;
  if(z->x==0)
     if(z->y > 0)
        return PI/2;
     else
        return (3*PI)/2;

   /* x != 0 si y != 0 */
   a = atan(z->y/z->x);
   if(z->x < 0)
     return a+PI;
   else
     if(z->y < 0)      /* x>0 si y<0 */
       return 2*PI+a;
```

```
     else              /* x>0 si y>0 */
       return a;
}
```

3.1. L11_3_1.CPP

```
double dmodul(COMPLEX *z)
/*calculeaza si returneaza modulul numarului complex z*/
{
  return sqrt(z->x * z->x + z->y * z->y);
}
```

3.2. L11_3_2.CPP

```
double darg(COMPLEX *z)
{
  double a;

  if(z->x==0 && z->y==0)
     return 0.0 ;
  if(z->y==0)
     if(z->x > 0)
        return 0.0;
     else /* y=0 si x<0 */
        return PI;
  if(z->x==0)
     if(z->y > 0)
        return PI/2;
     else
        return (3*PI)/2;

   a = atan(z->y/z->x);
   if(z->x < 0)
     return a+PI;
      else
     if(z->y < 0)      /* x>0 si y<0 */
       return 2*PI+a;
     else              /* x>0 si y>0 */
       return a;
}
```

3.3. L11_3_3.CPP

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159265358979

typedef struct {
                 double x;
                 double y;
                } COMPLEX;

#include "l11_3_1.cpp"
#include "l11_3_2.cpp"

void main()
 /*citeste nr. complexe, afiseaza modulul si argumentul*/
{
 COMPLEX complex;
 printf("\nIntroduceti partea reala si partea imaginara ");
 printf("\n ale numarului complex z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
     {
```

```
       printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
       printf("modul=%g \t arg=%g \n",dmodul(&complex),darg(&complex));
       printf("\n\nIntroduceti partea reala si partea ");
       printf("\nimaginara ale numarului complex z = a + ib :");
       printf("\n(Valori nenumerice incheie executia programului)\n");
     }
}
```

4. L11_4.CPP

```
void sum_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  c->x = a->x + b->x;
  c->y = a->y + b->y;
}
```

5. L11_5.CPP

```
void scad_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  c->x = a->x - b->x;
  c->y = a->y - b->y;
}
```

6. L11_6.CPP

```
void mul_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  c->x = a->x * b->x - a->y * b->y;
  c->y = a->x * b->y + b->x * a->y;
}
```

7. L11_7.CPP

```
void div_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  double numitor;
  numitor = b->x * b->x + b->y * b->y;
  if(numitor==0)
        exit(1);
  c->x = (a->x * b->x + a->y * b->y) / numitor;
  c->y = (a->y * b->x - a->x * b->y) / numitor;
}
```

8. L11_8.CPP

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

typedef struct {
                 double x;
                 double y;
                } COMPLEX;

#include "l11_4.cpp"
#include "l11_5.cpp"
#include "l11_6.cpp"
#include "l11_7.cpp"

void main()
```

```
{
 COMPLEX a,b,c;
 printf("\n\nIntroduceti partea reala si partea imaginara ");
 printf("\n ale numarului complex a :\n");
 if(scanf("%lf %lf",&a.x,&a.y)!=2)
        {
          printf("\nEroare");
          exit(1);
         }
 printf("a = %g + i*(%g)\n",a.x,a.y);

 printf("\nIntroduceti partea reala si partea imaginara ");
 printf("\n ale numarului complex b :\n");
 if(scanf("%lf %lf",&b.x,&b.y)!=2)
        {
          printf("\nEroare");
          exit(1);
         }
 printf("b = %g + i*(%g)\n",b.x,b.y);

 sum_c(&a,&b,&c);
 printf("\na+b = %g + i*(%g)",c.x, c.y);
 scad_c(&a,&b,&c);
 printf("\na-b = %g + i*(%g)",c.x, c.y);
 mul_c(&a,&b,&c);
 printf("\na*b = %g + i*(%g)",c.x, c.y);
 div_c(&a,&b,&c);
 printf("\na/b = %g + i*(%g)\n",c.x, c.y);

 getch();
}
```

LIMBAJE DE PROGRAMARE

Lucrarea 12

MODUL GRAFIC

I. NOTIUNI TEORETICE

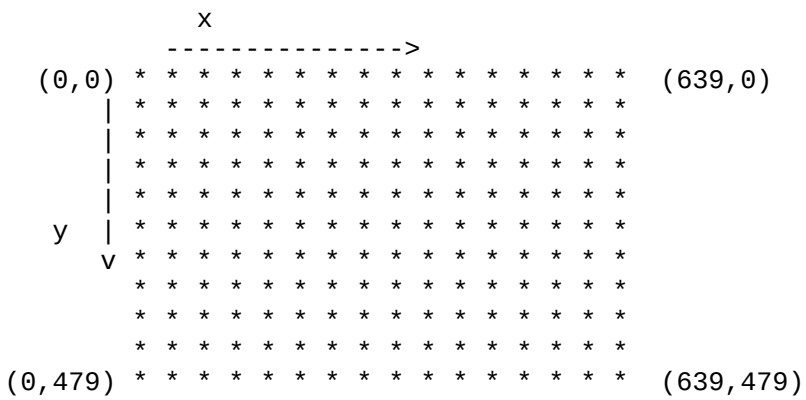
1. Introducere

Functionarea fizica a monitorului in modul grafic este rezolvata cu ajutorul adaptorului grafic (placa grafica) care asigura gestiunea memoriei ecran si controlul monitorului video. In plus, este nevoie de un driver grafic soft(in Turbo C acesta e un fisier de tip .BGI)care sa gestioneze logic(soft) modul grafic.
Un fisier cu extensia .BGI contine un driver soft care are mai multe moduri de lucru numite moduri grafice.
Orice program Turbo C care utilizeaza functii grafice va trebui sa inceapa cu includerea fisierului sursa graphics.h (#include<graphics.h>), fisier ce contine prototipurile acestor functii printre care se afla si functii ce permit selectarea modului grafic dorit.
Functiile grafice al caror prototip se afla in fisierul graphics.h sunt incluse ca rutine executabile intr-o biblioteca de functii C. Fisierul graphics.h si biblioteca de functii grafice graphics.lib constituie o interfata intre driverul grafic soft si utilizator.

Obs.: performantele programelor de grafica depind intr-o foarte mare masura de cele ale placii grafice din configuratia calculatorului IBM-PC cu care se lucreaza.

Functiile grafice C apelabile de catre utilizator sunt in numar de 78. Printre ele exista functii pentru desenare punct, linie, dreptunghi, poligon, paralelipiped dreptunghic, arc de cerc, arc de elipsa, cerc, elipsa, dar exista si functii cu ajutorul carora se pot colora formele sau se pot umple cu diferite culori si modele.

Pentru desenare este importanta cunoasterea modului de asezare pe ecran a sistemului de coordonate (x,y). In orice mod grafic, coltul stanga sus al ecranului reprezinta punctul de coordonate (0,0). Valoarea lui x, respectiv coloana, creste spre dreapta, iar valoarea lui y, respectiv linia, creste in jos (asa dupa cum se observa si in figura urmatoare).



2. Functii uzuale folosite in modul grafic Turbo C

2.1. Functia "initgraph"

Initializeaza sistemul grafic.

Sintaxa:

```
void initgraph(int *graphdriv,int *graphmode,char *pathdriv);
```

unde:

- graphdriv - este un pointer care precizeaza driverul;
- graphmode - este un pointer care precizeaza modul grafic;
- pathdriv - este un pointer la o expresie de tip caracter care precizeaza directorul ce contine fisierul .BGI curent.

In cazul in care graphdriv==DETECT sau 0 (functie de tipul placii grafice) se selecteaza automat modul grafic.

Rolul functiei initgraph consta in:

a) Selectarea modului grafic si incarcarea driver-ului grafic, specificati prin parametrii graphdriv si graphmode), cu cea mai mare rezolutie admisa, daca variabila graphdriv s-a initializat cu constanta DETECT;
b) Alocarea dinamica a unei zone de 4 Ko necesara executiei subprogramelor de hasurare a unor figuri;
c) Fixarea pozitiei punctului (pixelului) curent la (0,0);
d) Initializarea tuturor variabilelor sistemului grafic la valorile lor implicite.

2.2. Functia "closegraph"

Inchide modul grafic.

Sintaxa:

```
        void closegraph(void);
```

Functia face sa apara ecranul de dinaintea initializarii modului grafic si elibereaza zona de memorie ocupata pentru sistemul grafic.
Pentru inceput se recomanda folosirea urmatorului exemplu in scrierea programelor ce folosesc modul grafic:

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
/*alte declaratii*/
  ...........

/*initializare grafica */
initgraph(&gdriver,&gmode,"f:\\public\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }
  ..........
```

```
/*curatire completa*/
getch();
closegraph();
return 0;
}
```

2.3. Functiile "getmaxx" si "getmaxy"

Calculeaza numarul maxim de coloane, respectiv linii ce pot fi suportate de placa grafica.

Sintaxa:

```
int getmaxx(void);
```

respectiv:

```
int getmaxy(void);
```

2.4. Functia "moveto"

Muta pozitia cursorului grafic la punctul dat de coordonatele (x,y).

Sintaxa:

```
void moveto(int x, int y);
```

unde: x si y sunt expresii de tip intreg reprezentand coordonatele noii pozitii ale cursorului grafic.

2.5. Functia "outtextxy"

Afiseaza in modul grafic un text in locul specificat.

Sintaxa:

```
void outtextxy(int x, int y, char *text);
```

unde: x si y sunt coordonatele punctului de la care incepe afisarea textului.

Ex.: outtextxy(20,60,"Acesta este un text");

2.6. Functiile "getx" si "gety"

Intorc coordonata x, respectiv y a pozitiei curente a cursorului grafic.

Sintaxa:

```
int getx(void);
int gety(void);
```

Ex.: Afisarea coordonatelor unui punct oarecare:

```
char msg[100];  /* declaratia unui sir */
....
moveto(20,30); /* muta punctul curent la locatia (20,30) */
sprintf(msg,"(%d,%d)",getx(), gety()); /* creeaza mesajul */
outtextxy(20,30,msg); /* afiseaza mesajul la (20,30) */
```

2.7. Functia "setcolor"

Seteaza culoarea curenta pentru desenare.

Sintaxa:

```
void setcolor(int color);
```

unde: color este o expresie de tip intreg ale carei valori pot fi cuprinse intre 0 si 15 reprezentand toate culorile disponibile in graphics.h.

2.8. Functia "getmaxcolor"

Intoarce valoarea maxima ce poate fi folosita pentru culori.

Sintaxa:

```
int getmaxcolor(void);
```

Numarul intors de functie depinde de placa grafica cu care este dotat calculatorul.

2.9. Functia "setfillstyle"

Seteaza modelul de hasurare si culoarea.
Sintaxa:

```
void setfillstyle(int model, int color);
```

unde:

```
model - este o expresie intreaga (0 - 13) ce identifica
        un model predefinit;
color - este o expresie intreaga (0 - 15) a carei
        valoare reprezinta culoarea de hasurare
        (albul, WHITE, are indicativul 15 si este o
        culoare implicita);
```

Cu modelul si culoarea astfel selectate se pot hasura (prin apelul functiilor: bar, fillpoly, fillellipse, etc.) dreptunghiuri, poligoane, elipse, etc.

2.10. Functia "line"

Deseneaza un segment de dreapta intre doua puncte specificate.

Sintaxa:

```
void line(int x1, int y1, int x2, int y2);
```

unde: (x1,y1) si (x2,y2) sunt coordonatele punctelor intre care se va desena segmentul de dreapta.

2.11. Functia "lineto"

Deseneaza un segment de dreapta din pozitia curenta a cursorului grafic pana la punctul de coordonate (x,y).

Sintaxa:

```
void lineto(int x, int y);
```

unde: (x,y) sunt coordonatele punctului de sfarsit al segmentului de dreapta.

2.12. Functia "rectangle"

Deseneaza conturul unui dreptunghi.

Sintaxa:

```
void rectangle(int x1, int y1, int x2, int y2);
```

unde:
(x1,y1) - sunt coordonatele punctului ce reprezinta coltul stanga sus al dreptunghiului;
(x2,y2) - sunt coordonatele punctului ce reprezinta coltul dreap ta jos al dreptunghiului.

2.13. Functia "bar"

Deseneaza un dreptunghi fara contur hasurat cu modelul curent (dat de setfillstyle) si culoarea curenta.

Sintaxa:

```
void bar(int x1, int y1, int x2, int y2);
```

unde:
(x1,y1) - reprezinta coltul stanga sus al dreptunghiului;

(x2,y2) - reprezinta coltul dreapta jos al dreptunghiului.

2.14. Functia "drawpoly"

Deseneaza conturul unui poligon.
Sintaxa:
```
        void drawpoly(int nr_varfuri, int *pol);
```
unde:
nr_varfuri - reprezinta numarul de varfuri ale poligonului plus 1 (deoarece drawpoly nu inchide automat poligonul asa incat trebuie sa-l inchidem noi, ceea ce implica faptul ca ultima pereche de coordonate trebuie sa coincida cu prima);
*pol - este un pointer la un vector ce contine nr_varfuri perechi de valori (fiecare pereche de valori x si y reprezentand coordonatele unui varf al poligonului). Acesta poate fi si un tablou unidimensional cu numar par de elemente.

2.15. Functia "fillpoly"

Deseneaza si umple un poligon.

Sintaxa:
```
        void fillpoly(int nr_varfuri, int *pol);
```
unde:
nr_varfuri - reprezinta numarul de varfuri ale poligonului (fillpoly inchide automat poligonul);
*pol - este un pointer la un vector ce contine nr_varfuri perechi de valori (fiecare pereche de valori x si y reprezentand coordo natele unui varf al poligonului). Acesta poate fi si un tablou unidimensional cu numar par de elemente.

2.16. Functia "circle"

Deseneaza conturul unui cerc cu centrul si razele sale.

Sintaxa:
```
       void circle(int x, int y, int raza);
```
unde:
        (x,y) - sunt coordonatele centrului cercului;
        raza - este raza cercului.

2.17. Functia "ellipse"
Deseneaza conturul unui arc de elipsa.

Sintaxa:
```
void ellipse(int x, int y, int start_ungh, int stop_ungh,
               int xraza, int yraza);
```
unde:
(x,y) - sunt coordonatele centrului;
start_ungh si stop_ungh - sunt unghiurile de inceput si de sfarsit (in grade);
xraza si yraza - sunt razele (axele) orizontala si verticala.

Daca start_ungh=0 si stop_ungh=360, atunci se traseaza intreaga elipsa.
Unghiurile se dau in grade, in sens invers acelor de ceasornic (0 grade corespunde orei 3, iar 90 de grade orei 12).

2.18. Functia "fillellipse"

Deseneaza si umple (hasureaza) o elipsa.

Sintaxa:

```
void fillellipse(int x, int y, int xraza, int yraza);
```
unde:

(x,y) - sunt coordonatele centrului;
xraza si yraza - sunt razele (axele) orizontala si verticala.

Elipsa este umpluta cu modelul de hasurare curent si culoarea de hasurare curenta.

II. DESFASURAREA LUCRARII

1. Scrieti un program, utilizand modelul de la paragraful 2.2., care deseneaza o diagonala incepand din coltul stanga sus al ecranului si sfarsind in coltul din dreapta jos.

2. Scrieti un program care indica (prin coordonate) punctele (20,30) si (100,100) si traseaza o linie intre ele. A se vedea exemplul de la paragraful 2.6. Variati punctele.

3. Scrieti un program care deseneaza un patrat in centrul ecranului de dimensiune 100 pe 100. Variati pozitia si dimensiunile.

4. Scrieti un program care deseneaza un patrat, hasurat cu toate modelele (de culoarea alba), in centrul ecranului de dimensiune 100 pe 100.

5. Scrieti un program care deseneaza un poligon ce are varfurile 20,maxy/2), (maxx-20,20), (maxx-50,maxy-20), (maxx/2,maxy/2),
unde:

```
maxx = getmaxx();
maxy = getmaxy();
```

Mariti numarul de varfuri ale poligonului.

6. Scrieti un program care umple (hasureaza) cu toate modelele poligonul din programul de mai sus.

7. Scrieti un program care deseneaza un cerc in centrul ecranului de raza 100. Variati pozitia si raza cercului.

8. Scrieti un program care deseneaza o elipsa completa in centrul ecranului cu axa x (xraza) de 100 si axa y (yraza) de 50.

9. Scrieti un program care umple (hasureaza) cu toate modelele elipsa din programul de mai sus.

III. RASPUNSURI:

1. L12_1.C

```
#include<graphics.h>
#include<stdlib.h>

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


int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;

/*initializare grafica */
initgraph(&gdriver,&gmode, "c:\\tc\\bgi\\");
```

```
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }

/* Deseneaza o linie */


line(0,0,getmaxx(),getmaxy());

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

2. L12_2.C

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
/*alte declaratii*/
char msg[80];
/*initializare grafica */
initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }

/*muta punctul curent la locatia (20,30)*/
moveto(20,30);
/*creaza si afiseaza un mesaj la (20,30)*/
sprintf(msg,"(%d,%d)",getx(),gety());
outtextxy(20,30,msg);
/*deseneaza o linie la (100,100)*/
lineto(100,100);
/*creaza si afiseaza un mesaj la punctul curent*/
sprintf(msg,"(%d,%d)",getx(),gety());
outtext(msg);

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

3. L12_3.C

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
/*alte declaratii*/
int stanga,sus,dreapta,jos;

/*initializare grafica*/
initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }

stanga = getmaxx()/2 - 50;
sus = getmaxy()/2 - 50;
dreapta = getmaxx()/2 + 50;
jos = getmaxy()/2 + 50;

/* Patrat */
rectangle(stanga,sus,dreapta,jos);

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

4. L12_4.C

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
/*alte declaratii*/
int midx, midy, i;

/*initializare grafica */
initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }
```

```
midx = getmaxx()/2;
midy = getmaxy()/2;

for(i=0;i<13;i++)
  {
   setfillstyle(i,getmaxcolor());
/* desenare dreptunghi plin */
   bar(midx-50, midy-50, midx+50, midy+50);
   getch();
  }


/*curatire completa*/
getch();
closegraph();
return 0;
}
```

5. L12_5.C

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
/*alte declaratii*/
int maxx, maxy;
int pol[10];

/*initializare grafica */
initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }

maxx = getmaxx();
maxy = getmaxy();

pol[0] = 20;             /*primul varf*/
pol[1] = maxy/2;

pol[2] = maxx-20;       /*al doilea varf*/
pol[3] = 20;

pol[4] = maxx-50;       /*al treilea varf*/
pol[5] = maxy-20;

pol[6] = maxx/2;        /*al patrulea varf*/
pol[7] = maxy/2;

/* drawpoly nu inchide automat poligonul,
 asa ca il inchidem noi */
pol[8] = pol[0];
pol[9] = pol[1];
```

```
drawpoly(5,pol);

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

6. L12_6.C

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
/*alte declaratii*/
int maxx, maxy, i;

int pol[8];

/*initializare grafica */
initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }

maxx = getmaxx();
maxy = getmaxy();

pol[0] = 20;             /*primul varf*/
pol[1] = maxy/2;

pol[2] = maxx-20;       /*al doilea varf*/
pol[3] = 20;

pol[4] = maxx-50;       /*al treilea varf*/
pol[5] = maxy-20;

/* functia fillpoly inchide automat poligonul */
pol[6] = maxx/2;        /*al patrulea varf*/
pol[7] = maxy/2;

for(i=0;i<13;i++)
  {
   setfillstyle(i,WHITE);
   fillpoly(4,pol);
   getch();
  }

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

7. L12_7.C

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

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
int midx,midy;
int radius=100;

/*initializare grafica */

initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }
midx=getmaxx()/2;
midy=getmaxy()/2;
setcolor(getmaxcolor());
/*deseneaza un cerc*/
circle(midx,midy,radius);
/*curatire completa*/
getch();
closegraph();
return 0;
}
```

8. L12_8.C

```
#include<graphics.h>
#include<conio.h>

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
int xcenter, ycenter;
int start_ungh=0, stop_ungh=360;
int raza_x = 100, raza_y = 50;
/*initializare grafica */

initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }

xcenter = getmaxx()/2;
ycenter = getmaxy()/2;

/* deseneaza o elipsa */
```

```
ellipse(xcenter, ycenter, start_ungh, stop_ungh, raza_x, raza_y);

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

9. L12_9.C

```
#include<graphics.h>
#include<conio.h>

int main(void)
{
/*autodetectie cerere*/
int gdriver=DETECT,gmode,errorcode;
int xcenter,ycenter,i;
/*initializare grafica */
initgraph(&gdriver,&gmode,"c:\\tc\\bgi\\");
errorcode=graphresult();/*citirea rezultatului initializarii*/
if(errorcode!=grOk) /*exista o eroare*/
   {
   printf("\nEroare grafica:%s\n",grapherrormsg(errorcode));
   printf("Apasati orce tasta pentru oprire:");
   getch();
   exit(1);/*terminare cu un cod de eroare*/
   }
xcenter=getmaxx()/2;
ycenter=getmaxy()/2;
for(i=0;i<13;i++)
  {
   setfillstyle(i,WHITE);
   fillellipse(xcenter,ycenter,100,50);
   getch();
  }

/*curatire completa*/
getch();
closegraph();
return 0;
}
```

BIBLIOGRAFIE

[1] Aspru, O. "Grafica in Turbo C", Editura ADIAS, 1994;
[2] Bjarne, S. "The C++ Programming Language", Bell Telephone Laboratories, Incorporated, 1991;
[3] Negrescu, L. "Limbajul C", Editura Microinformatica, Cluj Napoca, 1994;
[4] Ritchie, D.; Kernigham, B. "The C Programming Language", Prentice-Hall, Inc., Englewood Clifs, New Jersey, 1978.