Cap.1 Elemente de bază ale limbajului C 1
1.1 Structura unui program C 1
1.2 Vocabularul limbajului 1
1.3 Tipuri de date 1
1.4 Directive preprocesor 2
1.5 Exerciţii şi teste grilă 2
Cap.2 Tipuri fundamentale de date 4
2.1 Constante 4
2.2 Variabile 5
2.3 Exerciţii şi teste grilă 5
Cap.3 Funcţii de intrare/ieşire standard 7
3.1 Clasificarea funcţiilor de intrare/ieşire 7
3.2 Funcţiile getch şi getche 7
3.3 Funcţia putch 7
3.4 Macrourile getchar şi putchar 8
3.5 Funcţiile gets şi puts 9
3.6 Funcţia printf 9
3.7 Funcţia scanf 11
3.8 Exerciţii şi teste grilă 13
Cap.4 Operatorii limbajului C 18
4.1 Precedenţa operatorilor 18
4.2 Operatorul de atribuire simplă 19
4.3 Operatori aritmetici 19
4.4 Operatorii relaţionali 20
4.5 Operatori logici 20
4.6 Operatorii la nivel de bit 21
4.7 Operatori compuşi de atribuire 22
4.8 Operatorul de conversie explicită (cast) 23
4.9 Operatorul sizeof 23
4.10 Operatorii de adresare 23
4.11 Operatorul condiţional 23
4.12 Operatorul virgulă 24
4.13 Exerciţii şi teste grilă 24
Cap.5 Instrucţiunile limbajului C 29
5.1 Instrucţiunea vidă 29
5.2 Instrucţiunea expresie 29

5.3 Instrucţiunea compusă 30
5.4 Instrucţiunea if 30
5.5 Funcţia standard exit 32
5.6 Instrucţiunea while 33
5.7 Instrucţiunea for 34
5.8 Instrucţiunea do-while 36
5.9 Instrucţiunea continue 38
5.10 Instrucţiunea break 39
5.11 Instrucţiunea switch 40
5.12 Instrucţiunea goto 42
5.13 Funcţiile standard sscanf şi sprintf 43
5.14 Header-ul ctype.h 46
5.15 Funcţii matematice uzuale 47
5.16 Exerciţii şi teste grilă 48
Cap.6 Tablouri *55*
6.1 Declararea tablourilor 55
6.2 Iniţializarea tablourilor 55
6.3 Prelucrări elementare ale vectorilor 56
6.3.1 Citirea elementelor unui vector 56
6.3.2 Determinarea elementului minim/maxim 56
6.3.3 Determinarea primului element cu o anumită proprietate 56
6.3.4 Determinarea ultimului element cu o anumită proprietate 57
6.3.5 Eliminarea tuturor elementelor cu o anumită proprietate 57
6.3.6 Eliminarea elementului din poziţia k dată (1<=k<=n) 57
6.3.7 Inserarea unui element y în poziţia k dată (1<=k<=n) 57
6.3.8 Permutarea circulară cu o poziţie spre stânga 58
6.3.9 Permutarea circulară cu o poziţie spre dreapta 58
6.3.11 Algoritmul de căutare binară 59
6.3.12 Interclasarea vectorilor 59
6.4 Prelucrări elementare ale matricilor 60
6.4.1 Citirea elementelor unei matrici 61
6.4.2 Tipărirea elementelor unei matrici 61
6.4.3 Determinarea elementului maxim/minim 61
6.4.4 Identificarea elementelor specifice unei matrici pătratice 61
6.5 Exerciţii şi teste grilă 62
Cap.7 Pointeri *68*
7.1 Variabile pointer 68
7.2 Aritmetica pointerilor 69
7.3 Legătura pointer – tablou 70

7.4 Exerciţii şi teste grilă 74
Cap.8 Şiruri de caractere 79
8.1 Folosirea şirurilor 79
8.2 Tablouri de şiruri 79
8.3 Funcţii standard pentru prelucrarea şirurilor de caractere 80
8.3.1 Lungimea unui şir de caractere 80
8.3.2 Copierea unui şir de caractere 80
8.3.3 Concatenarea şirurilor de caractere 81
8.3.4 Compararea şirurilor de caractere 82
8.3.5 Căutarea în şiruri de caractere 82
8.4 Exemple de utilizare a funcţiilor standard 83
8.5 Funcţii pentru conversii de date 85
8.6 Exerciţii şi teste grilă 86
Cap.9 Structuri 91
9.1 Definirea tipurilor structură 91
9.2 Iniţializarea structurilor 92
9.3 Operaţii permise asupra structurilor 93
9.4 Exerciţii şi teste grilă 94
Cap.10 Exploatarea fişierelor 96
10.1 Noţiunea de fişier 96
10.2 Deschiderea unui fişier 96
10.3 Închiderea unui fişier 97
10.4 Funcţia de verificare a sfârşitului unui fişier 97
10.5 Funcţii de citire/scriere caractere 97
10.6 Funcţii de citire/scriere pe şiruri de caractere 98
10.8 Funcţii de citire/scriere a fişierelor pe blocuri de octeţi 100
10.10 Exerciţii şi texte grilă 100
Răspunsuri la testele grilă 104
Bibliografie 106

Cap.1 Elemente de bază ale limbajului C

1.1 Structura unui program C

În C, elementul de bază al unui program este **funcţia**. O funcţie este o secţiune de program construită conform anumitor regului pentru declaraţii şi instrucţiuni de prelucrare a datelor problemelor. Nu este permisă definirea unei funcţii în interiorul altei funcţii. Structura cea mai generală a unui program C este următoarea :

```
directive preprocesare
declaraţii globale
funcţie1
funcţie2
……..
main
```

Orice program conţine funcţia **main** care este funcţia principală a unui program. Execuţia programului începe cu execuţia acestei funcţii.

Pentru specificarea explicaţiilor necesare unei mai bune înţelegeri şi utilizări a programelor se foloseşte **comentariul**, care are sintaxa : **/*……text comentariu…….*/**
Textul din comentariu poate avea mai multe linii . Se poate folosi şi forma: **// .....text comentariu** caz în care comentariul se referă la textul scris până la sfârşitul liniei respective.
Exemplu : Programul următor va realiza doar afişarea unui mesaj cu ajutorul funcţiei **printf**.

```
#include<stdio.h>
/* includerea bibliotecii standard  pentru citirea şi scrierea datelor */
void main()       /* funcţia principală */
{
      printf("Test C primul program !"); /* afişare mesaj */
}
```

1.2 Vocabularul limbajului

Elementele de bază ale limbajului, numite şi entităţi sintactice sau atomi lexicali, fac parte din următoarele categorii :

- **cuvinte rezervate** : sunt nume rezervate instrucţiunilor, tipurilor fundamentale şi sintaxei de definire a funcţiilor şi a tipurilor de date
- **identificatori** : sunt nume de date, constante sau variabile. Sunt formate dintr-un şir de caractere care începe cu o literă sau cu '_' , următoarele caractere putând fi litere, cifre sau '_'
- **constante** : sunt valori fixe reprezentând caractere, şiruri de caractere, numere întregi sau raţionale
- **delimitatori** : reprezintă simboluri care separă entităţile (spaţiu, tab etc) .

Observaţie : limbajul C face distincţie între literele mici şi mari.

1.3 Tipuri de date

Prin **tip de dată** înţelegem necesitatea definirii următoarelor aspecte :

- dimensiunea zonei de memorie asociate
- mulţimea valorilor corespunzătoare tipului
- timpul de viaţă asociat datei
- mulţimea operaţiilor prin care valorile tipului pot fi prelucrate (modificate sau testate) şi semnificaţia acestor operaţii
- operatorii utilizaţi şi restricţii în folosirea acestora

În C se lucrează cu valori ce pot fi stocate în **variabile** sau **constante**. Valorile **constante** nu se modifică pe parcursul rulării programului. Dacă au asociat un nume, atunci se numesc **constante simbolice** şi se declară printr-o directivă de preprocesare numită **macrodefiniţie** având sintaxa :

#define nume valoare

Exemplu : **`#define Ok 1`**

Dacă nu au nume, constantele se autoreprezintă prin însăşi maniera lor de scriere.

Variabilele sunt datele care îşi pot modifica valoarea pe parcursul execuţiei programului. Orice variabilă are asociat un nume şi o zonă de memorie care va fi prelucrată binar conform unor reguli specifice de interpretare a tipurilor de date.

Observaţie : orice variabilă trebuie declarată înainte de utilizarea sa.

Tipurile de date pot fi **predefinite** (puse la dispoziţie de limbaj) sau **derivate** (definite de utilizator).

O altă clasificare posibilă este următoarea :

- **simple** (scalare), care conţin o singură valoare de un anumit tip
- **compuse**, care conţin mai multe valori de acelaşi tip sau de tipuri diferite
- **pointeri**, care conţin adrese de memorie ale unor entităţi

1.4 Directive preprocesor

Reprezintă operaţii care vor fi efectuate înaintea compilării şi anume :

- includerea altor fişiere
- verificarea anumitor condiţii, a parametrilor de mediu sau a definiţiilor
- realizarea macrodefiniţiilor

Directivele de preprocesare încep cu caracterul # .

Exemplul 1 :

```
#include<stdio.h>
/*este inclus header-ul standard pentru intrări/ieşiri */
```

Exemplul 2 :

```
#include "file1.h"   // sunt incluse fişierele utilizatorului cu numele
#include "file1.c"   // specificat
```

Limbajul C conţine un mare număr de funcţii pentru prelucrarea datelor. Ele sunt organizate, în funcţie de scopul urmărit, în biblioteci numite fişiere **header** având extensia **h**. Exemple de biblioteci uzuale :

- **stdio.h , io.h** – pentru operaţii de citire/scriere de la dispozitivele standard
- **stdlib.h , math.h** – pentru prelucrări numerice
- **ctype.h** – pentru prelucrarea sau verificarea caracterelor
- **mem.h , string.h** – pentru prelucrarea zonelor de memorie şi a şirurilor de caractere
- **alloc.h , malloc.h , stdlib.h** - pentru alocarea memoriei
- **conio.h** - pentru interfaţa cu consola
- **graphics.h** - pentru interfaţa grafică
- **dos.h** - pentru interfaţa cu sistemul de operare

1.5 Exerciţii şi teste grilă

1. La compilare se sesizează :
a) erorile de sintaxă şi semantice
b) erorile de calcul
c) nerespectarea ordinii operaţiilor din modelul matematic
d) furnizarea unor date eronate la operaţia de citire

2. Delimitarea unui text între **/*** ***/** are rol de :
a) separare a subprogramelor în interiorul unui program
b) a delimita instrucţiunile care se execută cu prioritate
c) a permite utilizatorului să introducă mesaje explicative
d) nu au nici o semnificaţie

3. Care din următoarele cuvinte nu reprezintă un nume ?
a) a_X **b) a1b2c3**
c) 1abc **d) _ABC**

4. Care din următoarele instrucţiuni defineşte o constantă **MAXSIZE** cu valoarea 80 ?

```
a) constant MAXSIZE=80;
b) #define MAXSIZE 80
c) #define MAXSIZE=80
d) constant MAXSIZE=80
```

5. Definirea corectă a unei constante simbolice numită **TRUE** , care are valoarea 1 este :

```
a) int TRUE=1;
b) #define TRUE=1;
c) #define TRUE 1;
d) #define TRUE 1
```

6. Definirea corect a unei constante numită **GST** cu valoarea .125 este :

```
a) #define GST 0.125
b) GST .125;
c) float GST=0.125;
d) #define GST .125
```

7. Care din numele de variabile de mai jos nu este valid ?

```
a) go4it          b) go_cart
c) 4season        d) _what
```

8. Definiţi o constantă simbolică PI cu valoarea 3.14:

```
a) #define 3.14 PI;
b) #define float PI 3.14;
c) #define float PI=3.14;
d) #define PI 3.14
e) #define PI=3.14
```

Cap.2 Tipuri fundamentale de date

Limbajul C lucrează cu cinci tipuri de bază : **int**, **char**, **float**, **double** şi **void**. Tipul **void** are semnificaţia de "nimic" sau "orice tip" în funcţie de context. O prezentare a acestor tipuri apare în tabelul următor :

Tip	Număr de biţi	Domeniu de valori
char	**8**	**-128….127**
unsigned char	**8**	**0….255**
signed char	**8**	**-128…127**
int	**16**	**-2^{15}…….$2^{15}-1$**
unsigned int	**16**	**0…..$2^{16}-1$**
short int	**16**	**-2^{15}…….$2^{15}-1$**
long int	**32**	**-2^{31}….$2^{31}-1$**
unsigned long int	**32**	**0….$2^{32}-1$**
float	**32**	**valoarea absolută$\in\{3.4*10^{-38}$….$3.4*10^{38}\}$**
double	**64**	**valoarea absolută$\in\{1.7*10^{-308}$….$1.7*10^{308}\}$**
long double	**80**	**valoarea absolută$\in\{3.4*10^{-4932}$…$1.1*10^{4932}\}$**

Tipul **char** este folosit de obicei la prelucrarea caracterelor, dar poate fi folosit şi ca întreg de format scurt. Modificatorii de tip **signed** şi **unsigned** sunt folosiţi pentru datele de tip întreg pentru a specifica utilizarea, respectiv neutilizarea bitului de semn. Tipul logic nu este predefinit în C. Pentru el, convenţia de utilizare este : fals se consideră valoarea 0, true se consideră orice valoare nenulă.

2.1 Constante

a) **Constante întregi** : pot fi exprimate în bazele 8, 10 sau 16 . Constantele în baza 8 au întotdeauna prima cifră 0, iar cele în baza 16 au prefixul "**0x**" sau "**0X**".

Exemple :

```
0172  /* are 0 în faţă, este considerată în baza 8 */
120   /* este considerată implicit în baza 10 */
0x78  /* are 0x, este considerată în baza 16 */
```

Constantele de tip **long** au adăugată la sfârşit litera **l** sau **L** .

Exemplu : **1L 1000000L 581l**

Pentru constantele **unsigned** se adaugă la sfârşit **u** sau **U** .

Exemplu : **0u 12000u 20000lu**

b) **Constante caracter** : sunt reprezentate de unul sau mai multe caractere încadrate între apostrofuri .

Exemple : **'X' '1' '\n' '\t' '%'**

Pentru a putea utiliza anumite caractere speciale se folosesc secvenţele de evitare prezentate în tabelul de mai jos :

Secvenţa	Valoare hexazecimală	Caracter ASCII	Semnificaţia
\0	**0**	**NULL**	**terminator de şir**
\a	**0x07**	**BELL**	**generator de sunet**
\b	**0x08**	**BS**	**back space**
\f	**0x0C**	**FF**	**sfârşit de linie**
\n	**0x0A**	**LF**	**linie nouă**
\r	**0x0D**	**CR**	**salt la începutul rândului**
\t	**0x09**	**HT**	**tab orizontal**
\v	**0x0B**	**VT**	**tab vertical**
****	**0x5C**	****	**back slash**
\'	**0x27**	**'**	**apostrof**
\"	**0x22**	**"**	**ghilimele**

b) **Constante reale** : sunt în virgulă mobilă şi au în reprezentarea lor următoarele componente :
- partea întreagă
- punctul zecimal
- partea fracţionară
- e sau E şi un exponent

Exemple : **123.4 12e6 -111.2**

c) **Şiruri de caractere** : se scriu între ghilimele, iar la sfârşitul şirului compilatorul adaugă automat terminatorul de şir **'\0'**.

Exemplu : **"Testare siruri"**

2.2 Variabile

Declaraţia unei variabile are sintaxa :

tip_bază listă_variabile_declarate;

Lista poate fi formată din una sau mai multe variabile separate între ele prin virgulă. O variabilă poate să includă în declaraţie şi iniţializarea sa.

Exemple :

```
int n, k=0;
float media;
char c=65;
unsigned long int f;
double salar;
```

2.3 Exerciţii şi teste grilă

1. Care din următoarele nu este un tip de date în C ?

a) int b) numeric
c) float d) double

2. Tipul de date **INT** în C este reprezentat pe :

a) 2 octeţi b) 8 octeţi
c) 16 octeţi d) 32 octeţi

3. Tipul de date **DOUBLE** este reprezentat pe :

a) 8 biţi b) 16 biţi
c) 32 biţi d) 64 biţi

4. Tipul de date **CHAR** este reprezentat pe :

a) 4 biţi b) 8 biţi
c) 16 biţi d) 32 biţi

5. Care este valoarea maximă a unui tip de date cu semn exprimat pe 8 biţi ?

a) (2 la puterea 8) minus 1
b) (2 la puterea 7) minus 1
c) 2 la puterea 16
d) (2 la puterea 16) minus 1

6. Ce tip de constantă este **27U** ?

a) constantă integer universală
b) constantă short int
c) constantă unsigned integer
d) constantă caracter

7. Pentru fiecare dintre constantele aflate în coloana A) alegeţi din coloana B) tipul său:

Coloana A)	**Coloana B)**
A1) 5.0	B1) constantă întreagă
A2) 5	B2) constantă reală
A3) '5'	B3) const. hexazecimală
A4) 05	B4) constantă octală
A5) "5"	B5) constantă caracter
A6) 0x5	B6) constantă şir de caractere

a) A1→B2, A2→B1, A3→B5, A4→B1, A5→B6, A6→B3
b) A1→B2, A2→B1, A3→B5, A4→B4, A5→B5, A6→B3
c) A1→B2, A2→B1, A3→B5, A4→B4, A5→B6, A6→B3
d) A1→B2, A2→B1, A3→B5, A4→B4, A5→B6, A6→eronată
e) A1→B2, A2→B1, A3→B5, A4→B1, A5→B6, A6→eronată

8. Care dintre următoarele valori sunt constante flotante scrise corect?

1) 2307.98 2) +54.3 3) –20.07
4) –198. 5) .13 6) 1.9 E4

7) +2.7E+3 8) 2.e+4
a) 1), 2), 3), 6), 7)
b) toate mai puţin 5)
c) toate
d) toate mai puţin b)
e) primele cinci

9. Care dintre valorile de mai jos sunt constante întregi scrise corect?
a) 123 b) −17 c) +843
d) 0154 e) --67

10. Care dintre construcţiile de mai jos reprezintă constante tip caracter?
1) ″ ″ 2) ’\’ 3) ’a’
4) ’ ″ ’ 5) ’\\’ 6) ’\13’
7) ″a″ 8) ’ ’
a) 2), 3) ,8) b) toate
c) toate mai puţin 5) si 6)
d) 2), 3), 4), 8)
e) 3), 4), 5). 6), 8)

11. Care dintre următoarele declaraţii de variabile declară corect o variabilă **x** ce poate memora valori reale?
a) float x; b) double x;
c) unsigned float x;
d) x:float; e) x:double;

12. Care dintre liniile de program de mai jos realizează iniţializarea corectă a variabilei **x** la declararea sa?
a) int x==2; b) x:int=2;
c) int x=2; d) int x 2;
e) x=2:int;

13. Care dintre variabile vor avea valori întregi după execuţia secvenţei de program următoare?

```
int a=3, b, c;
float x=-11.23;
char d;
b=x; d='A'; c='M'-'N';
```

a) variabila **x** b) variabila **c**
c) variabila **d** d) variabila **a**
e) variabila **b**

14. Considerăm variabilele **a**, **b**, **c**, **d** şi **e**. Alegeţi varianta corectă a declaraţiilor, astfel încât atribuirile următoare să nu fie însoţite de conversii care să modifice valorile atribuite.

```
a=3; b=8; c=2.1; d=-3.5; e='B';
```

a) float a,b,c.d; char e;
b) int a,b,c,d; char e;
c) int a,b,e; float c,d;
d) int a,b; float c,d; char e;
e) int c,d; float a,b; char e;

15. O declaratie de genul :

```
int i=7.3;
```

va avea urmatorul efect :
a) semnalarea unei erori din partea compilatorului
b) va atribui lui **i** valoarea **7.3** şi va semnala un avertisment din partea compilatorului
c) va modifica tipul variabilei **i**
d) va atribui lui **i** valoarea **7**

16. Declaraţia corectă pentru definirea unui întreg numit **suma** este :
a) suma:integer; b) integer suma;
c) int suma; d) suma int;

17. Declaraţia corectă pentru definirea unei variabile caracter numită **litera** este :
a) litera:=char; b) char litera;
c) litera: char;
d) character litera;

18. Definirea corectă a unei variabile numită **bani** care poate fi utilizată pentru a memora valori reale simplă precizie este :
a) bani: real; b) real bani;
c) float bani; d) bani float;

19. Definirea corectă a unei variabile întregi numită **total** iniţializată cu zero este :
a) total: integer=0;
b) total=0, int;
c) int total=0;
d) int=0 , total;

20. Ce număr este echivalent cu **–4e3** ?
a) −4000 b) −400
c) .004 d) .0004

Cap.3 Funcţii de intrare/ieşire standard

3.1 Clasificarea funcţiilor de intrare/ieşire

Prin **intrări/ieşiri** înţelegem un set de operaţii care permit schimbul de date între un program şi un periferic. În general, operaţia de introducere a datelor de la un periferic se numeşte operaţie de **citire**, iar cea de ieşire pe un periferic **scriere**. Numim **terminal standard** terminalul de la care s-a lansat programul.

Funcţiile de citire/scriere se pot clasifica, după tipul datelor manevrate, în următoarele categorii:

- pentru caractere
- pentru şiruri de caractere
- cu format

În funcţie de locul de efectuare a operaţiilor de citire/scriere, funcţiile se împart în :

- funcţii de citire/scriere la consolă
- funcţii de citire/scriere într-o zonă de memorie
- funcţii de citire/scriere într-un fişier oarecare

Funcţiile utilizate mai frecvent pentru realizarea operaţiilor de intrare/ieşire folosind terminalul standard sunt :

- pentru intrări : **getch**, **getche**, **gets** şi **scanf**
- pentru ieşiri : **putch**, **puts** şi **printf**

La acestea se adaugă macrourile **getchar** pentru intrări şi **putchar** pentru ieşiri. Aceste macrouri sunt definite în header-ul **stdio.h** şi folosirea lor implică includerea acestui fişier.

3.2 Funcţiile getch şi getche

Funcţiile **getch** şi **getche** sunt independente de implementare. Ambele permit citirea direct de la tastatură a unui caracter. Funcţia **getch()** citeşte de la tastatură *fără ecou*, deci caracterul tastat nu se afişează pe ecanul monitorului. Ea permite citirea de la tastatură atât a caracterelor corespunzătoare codului ASCII, cât şi a celor corespunzătoare unor funcţii speciale cum ar fi tastele F1, F2 etc. sau combinaţii de taste speciale. La citirea unui caracter al codului ASCII, funcţia returnează codul ASCII al caracterului respectiv.

În cazul în care se acţionează o tastă care nu corespunde unui caracter ASCII, funcţia **getch()** se **apelează de două ori** : la primul apel funcţia returnează valoarea *zero*, iar la cel de-al doilea apel se returnează o valoare specifică tastei acţionate.

Funcţia **getche()** este analogă cu funcţia **getch**, cu singura diferenţă că ea realizează *citirea cu ecou* a caracterului tastat. Aceasta înseamnă că se afişează automat pe ecran caracterul tastat. Ambele funcţii nu au parametri şi se pot apela ca operanzi în expresii. La apelarea lor se vizualizează *fereastra utilizator* şi se aşteaptă tastarea unui caracter. Programul continuă după tastarea caracterului.

Funcţiile **getch** şi **getche** au prototipurile în fişierul **conio.h**, deci utilizarea lor implică includerea acestui fişier.

3.3 Funcţia putch

Funcţia **putch** afişează un caracter pe ecranul monitorului. Ea are un parametru care determină imaginea afişată la terminal. Funcţia poate fi apelată astfel : **putch(expresie);**

Prin acest apel se fişează imaginea definită de valoarea parametrului expresie. Valoarea parametrului se interpretează ca fiind codul ASCII al caracterului care se afişează. Dacă valoarea se află în intervalul [32,126], atunci se afişează un caracter imprimabil al codului ASCII. Dacă valoarea respectivă este în afara acestui interval, atunci se afişează diferite imagini care pot fi folosite în diverse scopuri, cum ar fi de exemplu trasarea de chenare.

Funcţia **putch** afişează caractere colorate în conformitate cu culoarea curentă setată în modul *text* de funcţionare al ecranului. La revenirea din funcţia **putch** se returnează valoarea parametrului de apel, adică codul imaginii afişate. Prototipul funcţiei se află în fişierul **conio.h**.

Exemplul 1: Să se scrie un program care citeşte un caracter imprimabil şi-l afişează apoi pe ecran.

```
#include<conio.h>
void main()
{
        putch(getch());
}
```

Exemplu 2 : Se citeşte de la tastatură un caracter fără ecou, se afişează caracterul, apoi se trece cursorul pe linia următoare.

```
#include<conio.h>
void main()
{
      clrscr();
      putch(getch());
      putch('\n');
      getch();
}
```

3.4 Macrourile getchar şi putchar

Aceste macrouri sunt definite în fişierul **stdio.h**. Ele se apelează la fel ca funcţiile. Macroul **getchar** permite citirea cu ecou a caracterelor de la terminalul standard. Se pot citi numai caractere ale codului ASCII, nu şi caractere corespunzătoare tastelor speciale. Prin intermediul macroului **getchar** caracterele nu se citesc direct de la tastatură. Caracterele tastate la terminal se introduc într-o zonă tampon până la acţionarea tastei **Enter**. În acest moment, în zona tampon, se introduce caracterul de rând nou (*newline*) şi se continuă execuţia lui **getchar**. Se revine din funcţie returnându-se codul ASCII al caracterului curent din zona tampon. La un nou apel al lui **getchar** se revine cu codul ASCII al caracterului următor din zona tampon. La epuizarea tuturor caracterelor din zona tampon, apelul lui **getchar** implică tastarea la terminal a unui nou set de caractere care se reîncarcă în zona tampon.

Un astfel de mod de desfăşurare a operaţiei de citire implică o anumită organizare a memoriei şi accesului la caractere, organizare care conduce la noţiunea de **fişier**.

În general, prin **fişier** se înţelege o mulţime ordonată de elemente păstrate pe suporturi de memorie externă. Elementele unui fişier se numesc **înregistrări**. Cu toate că fişierele sunt în general păstrate pe discuri, este util să se considere organizate în fişiere chiar şi datele care se tastează sau se afişează la terminal. În acest caz **înregistrarea** este un rând afişat la terminal sau succesiunea de caractere tastată la terminal şi terminată la apăsarea tastei **Enter**.

Fişierele conţin o înregistrare specială care marchează **sfârşitul de fişier**. Această înregistrare se realizează la tastatură prin secvenţe speciale, spre exemplu tastând **<Ctrl>+Z** al cărui ecou este **^Z.**

Macroul **getchar** returnează valoarea constantei simbolice **EOF** (End of File) la întâlnirea sfârşitului de fişier. Această constantă este definită în fişierul **stdio.h** şi în general are valoarea **–1**. Macroul **getchar** se apelează fără parametri şi de obicei este un operand al unei expresii : **getchar();**.

Macroul **puchar** afişează un caracter al codului ASCII. El returnează codul caracterului afişat sau –1 la eroare. Se poate apela cu : **putchar(expresie);** Valoarea expresiei reprezintă codul ASCII al caracterului care se afişează.

Exemplu : Să se scrie un program care citeşte un caracter folosind macroul **getchar**, îl afişează folosind macroul **putchar** şi trece cursorul pe linia următoare.

```
#include<stdio.h>
#include<conio.h>
void main()
{   clrscr();
    putchar(getchar());
    putchar('\n');
    getch();
}
```

3.5 Funcţiile gets şi puts

Funcţia **gets** poate fi folosită pentru a introduce de la terminalul standard o succesiune de caractere terminată prin acţionarea tastei **Enter**. Citirea se face cu ecou şi se pot citi numai caracterele codului ASCII. Funcţia are ca parametru adresa de început a zonei de memorie în care se păstrează caracterele citite. De obcei, acestă zonă de memorie este alocată unui tablou unidimensional de tip **char**. Deoarece numele unui tablou are ca valoare adresa de început a zonei de memorie alocată, rezultă că numele unui tablou poate fi utilizat ca parametru al funcţiei **gets**. În felul acesta, caracterele citite se vor păstra în tabloul respectiv.

Funcţia **gets** returnează adresa de început a zonei de memorie în care s-au păstrat caracterele. La întâlnirea sfârşitului de fişier (<Ctrl>+Z) se returnează valoarea zero. Zero nu reprezintă o valoare posibilă pentru **gets** şi de aceea, ea poate fi folosită pentru a semnala sfârşitul de fişier. De obicei, valoarea returnată de **gets** nu se testează faţă de zero, ci faţă de constanta simbolică **NULL** definită în fişierul **stdio.h**.

Rezultă că dacă **tab** este declarat prin **: char tab[255];** atunci apelul : **gets(tab);** păstrează în **tab** succesiunea de caractere tastată de la terminal în linia curentă. Totodată, caracterul **newline** se înlocuieşte cu **NUL** (‘\0’).

Funcţia **puts** afişează la terminalul standard un şir de caractere ale codului ASCII. După afişarea şirului respectiv, cursorul trece automat pe o linie nouă (deci caracterul **NUL** se înlocuieşte cu **newline**). Funcţia are ca parametru adresa de început a zonei de memorie care conţine caracterele de afişat. În cazul în care şirul de caractere care se afişează se păstrează într-un tablou unidimensional de tip **char**, drept parametru se poate folosi numele acestui tablou.

Funcţia **puts** returnează codul ultimului caracter al şirului de caractere afişat (caracterul care precede **NUL**) sau –1 la eroare. Dacă **tab** are declaraţia de mai sus şi el păstrează un şir de caractere, atunci apelul : **puts(tab);** afişează la terminalul standard şirul respectiv de caractere şi apoi trece cursorul pe linia următoare. Funcţiile **gets** şi **puts** au prototipurile în fişierul **stdio.h**.

Exemplul : Să se scrie un program care citeşte de la intrarea standard numele şi prenumele unei persoane, afişează iniţialele personei respective pe câte un rând, fiecare iniţială fiind urmată de un punct.

```
#include<stdio.h>
#include<conio.h>
void main()
{   char nume[30];
    char prenume[30];
    clrscr();
    gets(nume); gets(prenume);
    putchar(nume[0]);
    putchar('.');
    putchar(prenume[0]);
    putchar('.');
    puts("\nPentru a termina programul actionati o
                tasta ");
    getch();
}
```

3.6 Funcţia printf

Pentru scrierea cu format a datelor se foloseşte funcţia **printf** care face scrierea datelor în fişierul standard de ieşire (**stdout**). Sintaxa de utilizare este :

int printf(“mesaje si lista de formate”, expr_1, expr_2, ….,expr_n);

Funcţia **printf** realizează următoarele :

- acceptă o serie de argumente de tip expresie pe care, după ce le evaluează, le transformă în şiruri de caractere conform formatului specificat
- scrie şirurile în fişierul standard de ieşire (sunt acceptate secvenţele de evitare)

Dacă numărul de argumente specificate în format nu corespunde cu numărul argumentelor din lista de expresii, atunci apar rezultate neaşteptate care pot avea efecte dăunătoare. Rezultatul întors de funcţie, în caz de succes, este numărul de octeţi scrişi, iar în caz de eroare, valoarea întoarsă este EOF. **Specificatorii de format** folosiţi pentru **printf** sunt prezentaţi în tabelul următor :

Specificator	Semnificaţie
%e , %E	**Număr real de forma *iiii.zzzzzz* , unde nr.zecimale *z* este dat de precizie (6 implicit)**
%f	**Număr real de forma *i.zzzzzz* , unde nr. zecimale este dat de precizie (6 implicit) şi pentru partea întreagă este folosită doar o cifră**
%g , %G	**Număr real care suprimă caracterele terminale care nu influenţează valoarea , adică cifrele 0 de la sfârşit şi punctul zecimal , dacă are partea fracţionară 0**
%i	**Număr întreg în baza 8, 10, sau 16 în funcţie de primul sau primele două caractere**
%d	**Număr întreg în baza 10**
%o	**Număr întreg în baza 8 ; nu este necesară scrierea cifrei 0 la începutul numărului**
%x	**Număr întreg în baza 16 ; nu este necesară scrierea secvenţei 0x la începutul numărului**
%u	**Număr întreg fără semn**
%s	**Şir de caractere**
%c	**Un singur caracter**

Expresiile afişate se pot alinia la stânga sau la dreapta şi se poate forţa afişarea semnului astfel :

- semnul plus afişează explicit semnul expresiei
- semnul minus aliniază expresia afişată la stânga
- absenţa oricărui semn semnifică alinierea expresiei afişate la dreapta

Pentru numerele întregi şi pentru şirurile de caractere se poate specifica un număr care înseamnă spaţiul folosit pentru afişare. Dacă spaţiul necesar este mai mic sau egal cu numărul specificat, atunci se vor afişa suplimentar spaţii (sau zerouri, dacă numărul este precedat de cifra 0) până la completarea spaţiului de afişare.

Pentru numerele reale se pot specifica, opţional, semnul pentru aliniere şi două numere separate prin punct. Primul precizează dimensiunea totală de afişare, iar al doilea precizia, adică numărul de zecimale afişate.

În cazul şirurilor de caractere, specificarea a două numere separate prin punct indică faptul că primul număr reprezintă numărul de caractere din şir care se vor afişa, iar al doilea reprezintă limita superioară de tipărire, completarea făcându-se cu spaţii la dreapta sau stânga, în funcţie de modul de aliniere. Poate apare fenomenul de trunchiere a şirului afişat în cazul în care dimensiunea acestuia depăşeşte limita inferioară.

În cazul unui număr întreg, al doilea număr indică o completare la stânga cu zerouri până se ajunge la dimensiunea de afişare specificată.

Exemple:

Valoarea datei	Specificator	Afişare
3.14159265	%5f	3.141593
123.672	%7f	123.672000
3.14159265	%7.2f	3.14
123.672	%10.1f	123.7
-123.672	%10.1f	-123.7
3.14159265	%10.0f	3
123.672	%10.0f	124

Numărul zecimalelor se defineşte prin precizia indicată în specificatorul de format. Dacă ea este absentă atunci se afişează 6 zecimale. Ultima cifră afişată este rotunjită prin lipsă sau prin adaos.

Exemple:

Valoarea datei	Specificator	Afişare
3.14159265	%e	3.141593e+00
123.672	%e	1.236720e+02
123.672	%.1E	1.2E+02
0.673	%E	6.73000E-01
123.672	%.0E	1E+02

Numărul zecimalelor se defineşte prin precizia indicată în specificatorul de format. Dacă ea este absentă atunci se afişează 6 zecimale. Ultima cifră afişată este rotunjită prin lipsă sau prin adaos. Exponentul începe cu litera **e** dacă specificatorul de format se termină cu **e** şi cu **E** dacă el se termină cu **E**. Urmează un semn plus sau minus dacă exponentul este negativ. După semn se află un întreg zecimal de cel puţin două cifre.

Exemplul 1: folosirea afişării cu format pentru numere întregi

```
#include<stdio.h>
void main()
{     int nr=4321;
      printf("\n nr=%d",nr);         //  nr=4321
      printf("\n nr=%-d*",nr);       //  nr=4321*
      printf("\n nr=%6d",nr);        //  nr=   4321
      printf("\n nr=%-6d*",nr);      //  nr=4321   *
      printf("\n nr=%6.8d",nr);      //  nr=00004321
      printf("\n nr=%-6.8d*",nr);    //  nr=00004321*
}
```

Exemplul 2 : folosirea afişării cu format pentru numere reale

```
#include<stdio.h>
#include<conio.h>
void main()
{    double x=123.01234567;
     clrscr();
     printf("\n x=%f",x);        //  x=123.012346
     printf("\n x=%-f*",x);      //  x=123.012346*
     printf("\n x=%16f",x);      //  x=          123.012346
     printf("\n x=%-16f*",x);    //  x=123.012346          *
     printf("\n x=%.10f",x);     //  x=123.0123456700
     printf("\n x=%-.10f*",x);   // x=123.0123456700*
     printf("\n x=%12.4f",x);    //  x=       123.0123
     printf("\n x=%-12.4f*",x); //  x=123.0123       *
     getch();
}
```

Exemplul 3 : folosirea afişării cu format pentru şirurile de caractere

```
#include<stdio.h>
#include<conio.h>
void main()
{   char s[44]="Testare comportament printf pentru siruri !";
    clrscr();
    printf("\nsirul=%s",s);
    printf("\nsirul=%-s*",s);
    printf("\nsirul=%50s",s);
    printf("\nsirul=%-50s*",s);
    printf("\nsirul=%50.60s",s);
    printf("\nsirul=%-50.60s*",s);
    printf("\nsirul=%20.30s",s);
    printf("\nsirul=%-20.30s*",s);
    getch();
}
```

3.7 Funcţia scanf

Funcţia de citire cu format **scanf** are sintaxa :

scanf("lista de formate" , adresa_var1 , adresa_var2,…..);

Această funcţie realizează următoarele operaţii :

- citeşte din fişierul standard de intrare **stdio** o secvenţă de câmpuri de intrare, caracter cu caracter, până la terminarea introducerii câmpurilor şi apăsarea tastei <**Enter**> ;
- formatează fiecare câmp conform formatului specificat în lista de formate. Din caracterele citite se calculează valori numerice sau literale, conform tipului fiecărei variabile, dimensiunilor de format specificate şi a separatorilor de câmpuri predefiniţi (spaţiu, tab şi enter) sau impuşi explicit ;

- valorile astfel construite sunt stocate la adresele variabilelor specificate ca argumente ;

Ordinea formatelor variabilelor trebuie să coincidă cu ordinea listei adreselor variabilelor în care se face citirea. Fiecare variabilă care se doreşte a fi citită trebuie corelată cu un format specific. **Observaţie** :Indiferent de formatul folosit, la întâlnirea unui spaţiu în introducerea datelor, este terminată citirea variabilei.

Pentru funcţia de citire **scanf** trebuie folosit operatorul adresă **"&"**. Pentru variabilele citite cu această funcţie trebuie precizate adresele la care se stochează în memoria calculatorului valorile variabilelor. Funcţia va introduce valorile citite direct la acele adrese. Singurul caz în care nu este obligatorie folosirea operatorul adresă pentru citirea valorii unei variabile cu funcţia **scanf** este citirea unui şir de caractere .

Observaţie : citirea cu ajutorul funcţiei **scanf** a şirurilor de caractere care conţin spaţii este imposibilă.

În cazul în care formatul specificat este necorespunzător, rezultatul obţinut poate fi neprevăzut. Valoarea întoarsă de **scanf** în caz de succes, este numărul de variabile care au fost citite corect. Dacă nu a fost citită nici o variabilă (de exemplu s-a introdus un şir în loc de un număr) funcţia întoarce valoarea 0. Dacă apare o eroare înaintea oricărei citiri şi asignări, funcţia returnează **EOF** (constantă de sistem având valoarea întreagă –1).

Specificatorii de format ai funcţiei **scanf()** sunt prezentaţi în tabelul următor:

Cod	Semnificaţie
%c	Citeşte un caracter
%d	Citeşte un întreg zecimal
%i	Citeşte un întreg zecimal
%e	Citeşte un număr float
%f	Citeşte un număr float
%g	Citeşte un număr float
%o	Citeşte un număr octal fără semn
%s	Citeşte un şir de caractere
%x	Citeşte un număr hexazecimal fără semn
%p	Citeşte un pointer
%n	Argumentul asociat primeşte o valoare întregă egală cu numărul de caractere deja citite
%u	Citeşte un număr întreg fără semn
%[ ]	Scanare pentru un set de caractere

O caracteristică foarte interesantă a funcţiei **scanf()** este numită *scanset*. Un specificator *scanset* se poate crea prin includerea unei liste de caractere în interiorul unor paranteze drepte. Spre exemplu, iată un specificator *scanset* conţinând literele 'ABC' : **%[ABC]**. Când **scanf()** întâlneşte un specificator *scanset*, se începe citirea caracterelor şi depozitarea lor într-un tablou punctat de argumentul corespunzător. Citirea va continua cât timp caracterul citit face parte din *scanset*. În momentul în care caracterul citit nu face parte din *scanset*, funcţia **scanf()** opreşte citirea pentru acest specificator şi avansează la următorul specificator din şirul de control. Folosind semnul **–** în *scanset* se specifică un domeniu. De exemplu, următorul specificator se referă la literele de la 'A' la 'Z' : **%[A-Z]**.
Uneori când *scansetul* este mare, este mai uşor să specificăm ceea ce nu face parte din *scanset*. Pentru a realiza acest lucru, setul trebuie precedat de semnul **^.** De exemplu, **[^0123456789]**. Când **scanf()** întâlneşte acest *scanset*, va citi orice caracter exceptând cifrele de la 0 la 9. Se poate suprima asignarea unui câmp punând un asterisc imediat după semnul **%**. Această proprietate este foarte utilă când introducem informaţii care conţin şi caractere de care nu avem nevoie. De exemplu, dându-se :

```
int j,k;
scanf("%d%*c%d",&j,&k);
```

şi datele de intrare sub forma : **555-2345**, **scanf()** va asigna valoarea 555 variabilei **j**, va înlătura semnul **–** şi va asigna valoarea 2345 variabilei **k**.

Iată un exemplu de *scanset* care acceptă caractere litere mici şi litere mari. Încercaţi să introduceţi câteva litere, apoi orice alt caracter şi apoi din nou litere. După ce apăsaţi tasta **Enter** numai literele introduse înaintea caracterelor care nu au fost litere vor fi conţinute în **str**.

```
#include<stdio.h>
void main()
{
     char str[80];
     printf("Introduceti litere si apoi orice altceva\n");
```

```
        scanf("%[a-zA-Z]",str);
        printf("%s",str);
    }
```

Dacă doriţi să citiţi un şir conţinând spaţii folosind funcţia **scanf(),** va trebui să utilizaţi *scansetul* următor:

```
#include<stdio.h>
void main()
{
    char str[80];
    printf("Introduceti litere si spatii\n");
    scanf("%[a-zA-Z ]",str);
    printf("%s",str);
}
```

Se pot specifica de asemenea semne de punctuaţie, simboluri şi cifre, astfel că, virtual, se poate citi orice tip de şir.

Programul următor ilustrează efectul pe care îl are prezenţa unor caractere non-spaţiu în şirul de control. El ne permite să introducem o valoare zecimală, cifrele din stânga punctului zecimal sunt asignate unei variabile întregi, iar cele din dreapta punctului zecimal sunt asignate unei alte variabile întregi.

```
#include<stdio.h>
void main()
{
    int j,k;
    printf("Introduceti un numar zecimal : ");
    scanf("%d.%d",&j,&k);
    printf("stanga:%d\t dreapta:%d",j,k);
}
```

Observaţie : Dacă este posibilă apariţia erorilor la introducerea datelor, este necesar ca imediat după apariţia unei erori să folosim una din funcţiile :

- **fflush(stdin);** - pentru golirea buffer-ului fişierului standard de intrare
- **fflushall();** - pentru golirea tuturor buffer-elor fişierelor

Exemple :

```
char a[20];
int n;
scanf("%[A-Z]s",a);
/*citeşte un şir format numai din litere mari */
scanf("%[a-zA-Z]s",a);
/* citeşte un şir format din litere mari si mici */
scanf("%3d",&n);
/* citeşte un număr întreg de cel mult trei cifre */
```

3.8 Exerciţii şi teste grilă

1. Să se determine ce tipăreşte următoarea instrucţiune :

```
unsigned n=100;
printf("%04x",n);
```

a) 0100 b) 100
c) 0064 c) A10

2. Ce face secvenţa ?

```
float n=16;
printf("%x",n+1);
```

a) afişează numărul în baza 16
b) dă eroare de compilare deoarece nu am folosit o variabilă la scriere
c) chiar dacă nu este semnalată nici o eroare la compilare, nu se afişează valoarea dorită, deoarece nu am folosit un specificator de format adecvat

3. Fie declaraţiile :

```
int n;long double x;char s[100];
```

Care din secvenţele următoare sunt corecte pentru citirea variabilelor ?

```
a)  scanf("%i %s %lg",&n,&s,&x);
b)  scanf("%d %s %Lg",&n,&s,&x);
c)  scanf("%d %s %Lg",&n,s,&x);
d)  scanf("%d %c %lf",&n,&s,&x);
```

e) scanf("%d %",&n,&s,&x);

4. Fie declaraţia : **char s[20];**
Care din secvenţele de mai jos citesc corect un şir de caractere **s** ?
a) scanf("%c",s);
b) scanf("%c",&s);
c) scanf("%s",s);
d) scanf("%s",&s);
e) scanf("%",s);
f) scanf("",s);

5. Fie declaraţia : **char str[80];**
Care din următoarele secvenţe vor face citirea unui şir de maxim 80 caractere care să nu conţină caracterul "." ?
a) scanf("%[.]s",str);
b) scanf("%[^.]s",str);
c) scanf("%80[^.]s",str);
d) scanf("%80[^.]c",str);
e) scanf("%80[.^]s",str);

6. Fie declaraţia : **char s[100];**
Care din următoarele secvenţe va face citirea unui şir de caractere care să conţină numai cifre?
a) scanf("%[0123456789]",s);
b) scanf("%s",s);
c) scanf("%[^0-9]s",s);
d) scanf("%[09]s",s);
e) scanf("%[0-9]s",s);
f) scanf("%['0'-'9']",s);
g) scanf("%['0'...'9']",s);

7. Cum putem introduce un număr **x** întreg de maxim 4 cifre care să nu conţină cifra 0?
a) scanf("%4d",&x);
b) scanf("%4[^0]s",x);
c) scanf("%04d",&x);
d) scanf("%d0",&x);

8. Fie declaraţiile : **int a, b, c;**
şi apelul:
scanf("%2d%3d%4d",&a,&b,&c);
Care va fi valoarea variabilelor după introducere, dacă la intrare se tastează **123456**?
a) a=123 , b=345 , c=56
b) a=12 , b=345 , c=6
c) a=123456 , b=0 , c=0

9. Ce se întâmplă dacă se foloseşte secvenţa următoare ?
```
int m, n;
scanf("%d,%d",&m,&n);
```
a) obţinem eroare la compilare pentru că în interiorul formatului s-a pus virgulă
b) nu apar erori la compilare; deoarece în interiorul formatului s-a pus virgulă, nu se vor citi corect numerele
c) nu apar erori la compilare; deoarece s-a pus virgulă, numerele introduse trebuie separate prin virgulă
d) nu apar erori la compilare; pentru numerele introduse poate fi folosită orice secvenţă delimitatoare

10. Fie secvenţa următoare :
```
int a; char str[20];
scanf("%i", &a);
fflush(stdin);
gets(str);
```
Datele de intrare se introduc astfel :
```
100
abcd
```
Ce se întamplă dacă scoatem funcţia **fflush** din secvenţa anterioară ?
a) este semnalată eroare la compilare
b) nu se mai citeşte de pe linia a doua şirul **"abcd"**, acesta luând valoarea **""**(şir vid)
c) ambele date sunt citite corect, numărul luând valoarea **100** iar şirul valoarea **"abcd"**

11. Fie următoarele declaraţii de variabile:
int a; float x; char m;
Care dintre instrucţiunile de mai jos realizează citirea corectă a variabilelor **a**,**x** şi **m**?
a) scanf("%d %f %c",&a,&x,&m);
b) scanf("%d,%f,%c",a,x,m);
c) scanf("%f.%d.%c",&a,&x,&m);
d) scanf("a=%d,x=%f,c=%c",a,x,m);
e) scanf("a=%d\nx=%f\nc=%c\n",
&a,&x,&m);

12. Fie declaraţiile:
int a=34; float x=6.25;
Precizaţi care dintre instrucţiunile de afişare următoare trebuie executată astfel încât să se afişeze pe ecran rândul de mai jos:
34##:#6.250
unde prin "#" am simbolizat caracterul spaţiu.
a) printf("\n%4d:%-10f",a,x);
b) printf("\n%-4d:%6.3f",a,x);
c) printf("\n%6d:%10f",a,x);
d) printf("\n%-d:%-.3f",a,x);
e) printf("\n%d:%f",a,x);

13. Dacă de la tastatură se introduce caracterul "a", iar codurile literelor mici sunt succesive, începând cu 97, ce afişează programul următor?

```
#include<stdio.h>
#include<conio.h>
void main()
{
      char c, p;
      p=getchar():
      int n=p+259;
      c=n;
      putchar(c);
}
```

a) 356 b) 'a' c) 'd'
b) 100 e) programul este greşit

14. Care dintre secvenţele de mai jos nu conţin erori şi afişează cuvintele „Program" şi „simplu" unul sub altul (fiecare pe câte un rând) ?

a)
```
{
      printf("Program");
      printf("\nsimplu");
}
```
b)
```
{
      printf("Program\n");
      printf("simplu");
}
```
c)
```
{
      printf("Program\nsimplu");
      printf("\n");
}
```
d)
```
{
      printf("Program");
      printf("simplu\n");
}
```
e) nici unul dintre programele anterioare

15. Funcţiile **getchar()**, **getch()** şi **getche()** citesc de la tastatură un caracter. Ce deosebiri există între cele trei funcţii ?

a) funcţiile **getchar** şi **getche** realizează citire cu ecou, iar **getch** citeşte caracterul fără ecou
b) funcţia **getchar** citeşte caracterul cu ecou, iar funcţiile **getche** şi **getch** realizează citirea fără ecou
c) funcţiile **getchar** şi **getch** preiau caracterul numai după apăsarea tastei ENTER
d) funcţiile **getchar** şi **getche** preiau caracterul de îndată ce a fost tastat, fără să mai aştepte „confirmarea" cu ENTER
e) toate cele trei funcţii au prototipul în header-ul conio.h

16.
```
char x='A';
putchar(x);
putchar(x+1);
```
Referindu-ne la codul de mai sus şi presupunând că funcţia **putchar** ia ca argument un întreg, ce vom avea la ieşire după execuţie ?

a) BA b) A66 c) AB
d) Se va genera o avertizare la compilare şi execuţia nu este cea aşteptată

17. Unde scrie funcţia **printf()** ?

a) stdout b) stdio
c) stdin d) stderr

18.
```
C's a
      "fun" language!
```
Selectaţi codul care va produce ieşirea de mai sus:

a)
```
printf("C's a\"fun\"
                language!\n");
```
b)
```
printf("C's a
      \"fun\"language!");
```
c)
```
printf("C's a\n  "fun"
                language!\n");
```
d)
```
printf("C's a\n  \"fun\"
                language!\n");
```

19. `short int x;` /*presupunem că x este pe 16 biţi*/
Care este numărul maxim care poate fi tipărit folosind **printf("%d\n",x)**, presupunând că **x** este declarat aşa cum am arătat mai sus ?

a) 127 b) 255
c) 32767 d) 65536

20. Fie secvenţa :
```
printf("\n%10.0f",x);
```
Pentru **x=3.14159265** se va afişa :

a) 3 b) 3.14
c) 3.1415 d) 3.141593

21. Fie secvenţa :
```
printf("\n.1E",x);
```
Pentru **x=123.672** se va afişa :

a) 1.2E+02 b) 1.236E+02
c) 123E+02 d) 1E+02

22. Fie secvenţa :
```
printf("\n*%15.10S*",x);
```
Pentru **x="program c++"**, se va afişa :

a) * program c+* (5 spaţii în faţă)
b) *program c++*
c) *program c*
d) * program c++* (4 spaţii în faţă)

23. Fie secvenţa :
```
printf("\n*%07d*",x);
```
Pentru **x=123**, se va afişa :
a) *0000123* b) *123*
c) * 123* d) *123   *

24. Fie secvenţa :
```
printf("\n%.4e",x);
```
Pentru **x=-123.5e20**, se va afişa :
a) -1.2350e+22 b) -1.235e+22
c) -12.3500e+21 d) -1.2350e+20

25. Care este efectul apelului funcţiei **printf("%x",a),** unde **a** este o variabilă de tip întreg de valoare **0xAF5** :
a) 0xAF5 b) 5365 c) AF5
d) valoarea binară a variabilei **a**

26. Care este efectul următorului program :
```
#include <stdio.h>
#include <conio.h>
void main(void)
{
     putchar(getche()-'A'+'a');
     printf("\n");
}
```
a) citeşte şi afişează un caracter
b) citeşte o literă mare şi afişează litera mică corespunzătoare
c) citeşte o literă mică şi afişează litera mare corespunzătoare
d) citeşte un caracter

27. Se defineşte constanta simbolică **SIR** astfel:
```
#define SIR "EXAMEN ADMITERE"
```
Efectul apelului lui **printf("%-10.6s",SIR)** este următorul :
a) EXAMEN ADMITERE
b) EXAMEN (4 spaţii in faţă)
c) EXAMEN (4 spaţii după)
d) EXAMEN ADM

28. Care este efectul apelului funcţiei **printf("%o",a)** unde **a** este o variabilă de tip întreg de valoare **0xAF5** :
a) 05365 b) 0xAF5
c) AF5 d) 5365

29. Se dă următorul program :
```
#include <stdio.h>
void main()
{
        double x;
        scanf("%lf",&x);
        printf("%e",d);
}
```
Efectul acestui program este :
a) citeşte şi afişează un număr real
b) citeşte un număr cu exponent şi îl afişează
c) citeşte un număr real şi afişează valoarea sa cu exponenţială
d) afişează un număr real

30. Care este efectul instrucţiunii **printf("%u",x)** unde **x** este o variabilă de tip întreg de valoare **0xAF25** :
a) -20669 b) AF25
c) 44837 d) 0xAF25

31. Care este efectul apelului funcţiei **printf("%x",b)** unde **b** este o variabilă de tip întreg de valoare **0x12ABC** :
a) 12AB b) 12ABC
c) 0x12ABC d) 2ABC

32. Care este efectul apelului functiei **printf("%lx",b)** unde **b** este o variabilă de tip întreg în dublă precizie de valoarea **0x12ABC** :
a) 0x12ABC b) 12ABC
c) ABC d) 012ABC

33. Fie următorul program :
```
#include <stdio.h>
main()
{
     int k;
     scanf("%d",&k);
     printf("%x",k);
}
```
În urma lansării în execuţie a programului, valoarea lui **k** introdusă va fi **65**. Care va fi rezultatul afişat ?
a) 65 b) 41 c) 32 d) 60

34. Precizaţi care sunt valorile afişate, dacă se vor citi în ordinea indicată valorile numerice **5 2 -3**:
```
{  int a, b;
    scanf("%d%d%d",&a,&b,&a);
    printf("%d",a);
    printf("%d\n",b);
    printf("%d",a+b);
}
```
a) 5 2
   7
b) -3 2
   -1
c) 5 2 7
d) există erori de sintaxă

35. Care sunt valorile tipărite de programul următor :

```
void main(void)
{  int a,b;
   a=5; b=13;
   printf("%d+%2d=%2d",a, b, a+b);
}
a) a+b=a+b     b) 5+13=18
c) 5+13:2=18:2 d) a+b:2=a+b:2
```

36. Scrieţi o instrucţiune care afişează valoarea variabilei întregi **total** :

```
a) printf("%s\n",'total');
b) printf("%d\n",total);
c) printf('%s\n',"total");
d) printf total;
```

37. Scrieţi o instrucţiune care citeşte un caracter şi-l memorează în variabila **litera** :

```
a) scanf('litera');
b) scanf("litera");
c) scanf litera;
d) scanf("%c",&litera);
```

38. Scrieţi o instrucţiune care afişează valoarea unei variabile de tip real **small_value** cu trei zecimale exacte :

```
a) printf('small_value:3');
b) printf("%.3f\n",small_value);
c) printf("%s\n","small_value:3");
d) printf "%.3f",small_value;
```

Cap.4 Operatorii limbajului C

Limbajul C dispune de o gamă extinsă de operatori. Pe lângă setul de operatori uzuali, limbajul are definiţi operatori care oferă facilităţi asemănătoare limbajelor de asamblare. Există astfel operatori aritmetici, operatori de atribuire simplă sau compusă, operatori logici, operatori de prelucrare pe biţi etc.

4.1 Precedenţa operatorilor

Precedenţa operatorilor determină ordinea de evaluare a operaţiilor dintr-o expresie. În funcţie de precedenţă, operatorii C sunt împărţiţi în 15 categorii prezentate în tabelul următor :

Categorie	Operatori	Semnificaţie
1. Prioritate maximă	**( )** **[ ]** **→ .**	Apel de funcţie Expresie cu indici Selectori de membru la structuri
2. Operatori unari	**!** **~** **+ -** **++ --** **& *** **sizeof** **(tip)**	Negare logică Negare bit cu bit (complementare cu 1) Plus şi minus unari Incrementare/decrementare (pre şi post) Obţinerea adresei/indirectare Dimensiune operand (în octeţi) Conversie explicită de tip - cast
3. Operatori de multiplicare	*** /** **%**	Înmulţire/împărţire Restul împărţirii întregi
4. Adunare , scădere	**+ -**	Plus şi minus binari
5. Deplasări	**<< >>**	Deplasare stânga/dreapta pe biţi
6. Relaţionali	**< <=** **> >=**	Mai mic/mai mic sau egal Mai mare/mai mare sau egal
7. Egalitate	**==** **!=**	Egal Diferit
8.	**&**	SI logic bit cu bit
9.	**^**	SAU EXCLUSIV bit cu bit
10.	**|**	SAU logic bit cu bit
11.	**&&**	SI logic
12.	**||**	SAU logic
13. Op. condiţional	**?:**	Operatorul condiţional (ternar)
14. Operatori de atribuire	**=** ***= /=** **%=** **+= - =** **&= ^=** **!=** **<<=** **>>=**	Atribuire simplă Atribuire produs , cât , rest Atribuire sumă , diferenţă Atribuire SI , SAU EXCLUSIV , SAU (bit) Atribuire cu deplasare stânga/dreapta
15. Virgula	**,**	Evaluare expresie1 , expresie2 . Valoarea rezultatului este expresie2.

Cei din prima categorie au prioritatea maximă. Precedenţa descreşte cu cât creşte numărul categoriei din care face parte operatorul. Operatorii din aceeaşi categorie au acelaşi grad de precedenţă. Ordinea de evaluare a operaţiilor este de la stânga la dreapta, cu excepţia operatorilor unari (categoria 2), a operatorului condiţional (categoria 13) şi a operatorilor de atribuire (categoria 14) care au ordinea de evaluare de la dreapta la stânga.

Totuşi ordinea de efectuare a operaţiilor nu este întotdeauna perfect determinată. Se poate face o reorganizare a expresiilor pentru a obţine un cod mai eficient, dar ordinea de efectuare a

operaţiilor nu este strict definită. În funcţie de context, acelaşi operator poate avea semnificaţii diferite. Spre exemplu operatorul **&** (ampersand) poate fi considerat ca :

- operatorul binar SI pe biţi (**a & b**)
- operatorul unar, adresa unui operand (**&a**)

Semnificaţia depinde de numărul de argumente folosite, unul sau două .

4.2 Operatorul de atribuire simplă

Acest operator (**=**) realizează memorarea valorii unei expresii într-o variabilă. Are sintaxa :

variabila=expresie;

Efectul este stocarea valorii expresiei din membrul drept la adresa variabilei scrise în membrul stâng.

Exemplul 1:

```
char c;
int i,k;
float x;
c='a';
i=3;
k='d';
/* k=100 ; 'd' de tip caracter este convertit la un tip întreg   */
x=c+i;
/* x=100 ; se face conversie la un tip întreg : 97+3=100 */
```

În plus, atribuirea însăşi are o valoare, şi anume valoarea variabilei din stânga după memorarea conţinutului valorii expresiei. Datorită acestui efect, rezultă în final o valoare a expresiei de atribuire care poate fi folosită direct într-o altă atribuire. De aceea este permisă **atribuirea multiplă**.

Exemplul 2 :

```
int i,j,k ;
i=j=k=1;
```

care este echivalentă cu secvenţa :

```
k=1; j=k; i=j;
```

Se observă o mai bună compactare a codului sursă în primul caz, de folosire a atribuirii multiple. Toate cele trei variabile au după atribuire valoarea 1.

Exemplul 3:

```
int i,j,k;
j+1=i;
i=1+j=k;
```

Ambele instrucţiuni de atribuire sunt incorecte, pentru că **j+1** nu este o variabilă.

4.3 Operatori aritmetici

Operatorii aritmetici din C pot folosi, mai puţin operatorul modulo care poate lucra numai cu numere întregi, atât numere întregi cât şi numere reale.

Exemplul 1 : Folosirea operatorilor aritmetici .

```
int i,j,n;
float x;
n=10*4-7;   /* n=33 */
i=9/2;      /* i=4  ca rezultatul împărţirii a două numere întregi */
j=n%i;      /* j=1 ca restul împărţirii a două numere întregi */
x=n;        /* x=33.00  - ca număr real*/
x=x%i;
/* se obţine eroare operatorul % fiind definit numai pentru numere întregi*/
```

Exemplul 2 : La acelaşi rezultat contează tipul variabilei din stânga .

```
int i;
float x;
i=7./2;
/* i=3 pentru că 3.5 real este convertit la tipul întreg al lui  i */
X=7./2;    /* x=3.5 pentru că x este de tip real */
```

În concluzie, **rezultatul este convertit la tipul variabilei din membrul stâng al atribuirii**.

Exemplul 3: În operaţiile cu constante contează dacă ele sunt de tip întreg sau real.

```
int i,j;
i=5/2+7/2;
/* rezultatele împărţirii dintre două numere întregi sunt convertite tot la
 numere  întregi şi  i=2+3=5  */
j=5./2+7/2.;
/* rezultatele împărţirii dintre un număr real şi un număr întreg sunt
 numere reale, 2.5+3.5=6.0 , deci j=6 */
```

Spre deosebire de ceilalţi operatori aritmetici, operatorii de incrementare şi decrementare sunt specifici limbajelor de asamblare. Ei sunt mult mai rapizi, efectul lor constând în mărirea/micşorarea variabilei cu 1 în modul următor :

- **++var** sau **var++** : variabila *var* este mărită cu o unitate, în primul caz, înainte de utilizarea ei (**preincrementare**), iar în al doilea caz, după utilizarea ei (**postincrementare**)
- **--var** sau **var--** : variabila *var* este micşorată cu o unitate, în primul caz, înainte de utilizarea ei (**predecrementare**), iar în al doilea caz, după utilizarea ei (**postdecrementare**)

Are importanţă dacă operatorii de incrementare şi decrementare se folosesc la stânga sau la dreapta variabilei.

Exemplul 4:

```
int i,j=7;
i=j++;     /*  i=7 , j=8 */
```

sau

```
i=++j;     /*  i=8 , j=8 */
```

Exemplul 5:

```
int a=2,b=3,c,d;
c=d=(a++ +1)-b--;       /* a=3 , b=2 , d=0 , c=0 */
```

4.4 Operatorii relaţionali

Operatorii relaţionali (categoriile 6 şi 7) pot fi folosiţi pentru date de tip aritmetic şi pointeri. Rezultatul este 0 dacă relaţia nu este îndeplinită şi 1 dacă este îndeplinită.

Exemplu :

```
int egal,x,y;
x=17;
y=2*x-1;
egal=x==y;   /*  egal=0 */
```

4.5 Operatori logici

Cum în limbajul C nu există tipul boolean, operatorii logici admit operanzi de orice tip scalar (simplu) pe care îi interpretează conform comparaţiei cu 0. Dacă valoarea este 0, expresia este considerată falsă, iar dacă valoarea este nenulă, expresia se consideră adevărată. Rezultatul unei expresii logice este de tip întreg şi se bazează pe convenţiile : adevărat=1 , fals=0.

Tabela de adevăr a operatorilor logici				
x	**y**	**x && y**	**x || y**	**!x**
0	**0**	**0**	**0**	**1**
0	**!=0**	**0**	**1**	**1**
!=0	**0**	**0**	**1**	**0**
!=0	**!=0**	**1**	**1**	**0**

Exemplul 1: Două forme echivalente pentru verificarea egalităţii lui x cu 0.

```
x==0
```

sau

```
!x
```

Exemplul 2 : Mai multe variante de verificare a condiţiei ca x şi y să fie ambii 0.

```
a) x==0 && y==0
b) !x && !y
c) !(x!=0||y!=0)
```

```
d) !(x||y)
```

4.6 Operatorii la nivel de bit

Posibilitatea utilizării operatorilor pe biţi oferă limbajului C facilităţi asemănătoare cu cele ale limbajelor de asamblare. **Operanzii pot fi numai tipuri întregi** .
Fie biţii b1 şi b2. Rezultatul aplicării operatorilor pe biţi este :

Operatori pe biţi					
b1	**b2**	**b1 & b2**	**b1 ^ b2**	**b1 | b2**	**~ b1**
0	**0**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**0**
1	**1**	**1**	**0**	**1**	**0**

Operatorii **<<**, respectiv **>>**, sunt numiţi operatori de deplasare pe biţi la stânga, respectiv la dreapta. Ei au sintaxa :

```
a<<n    /* echivalent cu a*2^n */
a>>n    /* echivalent cu a/2^n */
```

Exemplul 1: Fie declaraţia `int j;`
atunci expresia `j&1` are valoarea 0 dacă **j** este par şi valoarea 1, dacă **j** este impar.
Exemplul 2:

```
int t=1,j=6,k,n;
t=1<<j;          /* t=1*2^6 */
k=t>>2;          /* k=2^6/2^2=2^4 */
n=t&k;
/* n=0 pentru că t şi k nu au nici un bit de pe aceeaşi poziţie egal cu 1 */
```

Exemplul 3: Fie constanta **0u**, o constantă întreagă de tip unsigned care se reprezintă pe 16 biţi, deci este practic reprezentată ca 16 de 0. Operaţia **~(0u)** neagă cei 16 biţi de 0 care vor lua valoarea 1, deci se va obţine valoarea 2^{16}-1=65535.
Exemplul 4: Tipărirea rezultatului operaţiilor pe biţi a două numere.

```
#include<stdio.h>
void main()
{     int a,b;
      printf("introduceti cele doua numere : ");
      scanf("%d %d",&a,&b);
      printf("a|b=%d\n",a|b);
      printf("a&b=%d\n",a&b);
      printf("a^b=%d\n",a^b);
}
```

Exemplul 5: Operatorii **|** şi **&** pot fi folosiţi pentru a seta sau şterge anumite câmpuri de biţi.

```
(k&1<<j)>>j ;  /* determină valoarea bitului de pe poziţia j */
k=k&~(1<<j);   /* setează bitul de pe poziţia j la 0 */
k=k|1<<j;      /* setează bitul de pe poziţia j la 1 */
k=k^1<<j;      /* complementează bitul de pe poziţia j */
```

Exemplul 6: Programul determină numărul format prin extragerea a **n** biţi consecutivi dintr-un număr dat **x** începând cu poziţia **p**.

```
#include<stdio.h>
void main()
{     int n,p,x;
      printf("numarul : "); scanf("%d",&x);
      printf("numarul de biti : "); scanf("%d",&n);
      printf("pozitia : "); scanf("%d",&p);
      printf("numarul este : %d", (x>>p)&~(~0<<n));
}
```

În partea stângă a operatorului **&** se elimină primii **p** biţi prin deplasarea la dreapta. Pentru a extrage cei **n** biţi, se realizează în partea dreaptă un filtru, setând primii **n** biţi pe 1 şi ştergându-i pe ceilalţi.

Exemplul 7: Un caz concret de utilizare a operaţiilor la nivel de bit este cel al reprezentării mulţimilor de valori. Astfel, să presupunem că dorim să ţinem evidenţa literelor majuscule dintr-un şir de caractere introduse de la consolă. În acest scop se poate folosi câte un singur bit pentru a codifica absenţa (0) sau prezenţa (1) fiecărei litere. Deoarece alfabetul latin este format din 26 litere, pentru a păstra evidenţa tuturor majusculelor trebuie să folosim o variabilă (**m**) de tip long (32 de biţi). Dintre cei 32 de biţi din reprezentarea variabilei vom utiliza numai 26, informaţia referitoare la o literă oarecare **x** fiind păstrată în bitul din poziţia **'Z'-x**. În absenţa majusculelor, toţi biţii fiind 0, condiţia de mulţime vidă se exprimă sub forma **!m**. Includerea în mulţime a unei majuscule x se realizează prin intermediul atribuirii:

```
m=m|(1L<<'Z'-x)
```

Se remarcă utilizarea constantei **1L**, de tip **long**. Dacă în locul ei s-ar fi utilizat constanta 1 (implicit de tipul **int**), atunci informaţiile referitoare la primele 10 litere din alfabet nu ar fi fost actualizate, deoarece rezultatul deplasării la stânga, cu mai mult de 15 biţi, a unei valori de tip **int** este nul. Verificarea prezenţei în mulţime a caracterului **x** se poate realiza în următoarele două variante :

```
m&(1L<<'Z'-x)   sau   (m>>'Z'-x)&1
```

4.7 Operatori compuşi de atribuire

Operatorul de atribuire poate fi combinat cu o serie de operatori aritmetici şi operatori la nivel de bit rezultând enunţuri prescurtate şi uneori optimizări ale timpului de execuţie.

Expresia **var= var operator expresie;** mai poate fi scrisă şi **var operator = expresie;**

Există 10 combinaţii posibile permise operatorilor de atribuire compusă şi anume :

- 5 combinaţii cu operatori aritmetici : **+= , - = , *= , /= , %=**
- 5 combinaţii cu operatori pe biţi : **|= , &= , ^= , <<= , >>=**

Executarea unei instrucţiuni care foloseşte operatori compuşi de atribuire este mult mai rapidă deoarece compilatorul, cunoscând că primul operand şi rezultatul au aceeaşi locaţie de memorie, foloseşte un număr mai mic de operaţii, având nevoie de un timp de execuţie mai mic decât atribuirea în cazul general.

Exemplul 1: Folosirea operatorului de aflare a restului .

```
int k=6,j=5;
k%=j;     /*  k=6%5=1 */
```

Exemplul 2: Folosirea operatorului de deplasare pe biţi .

```
int k=7,j=3;
k<<=j;        /* k=k*2^3=7*8=56 */
```

Exemplul 3 : Folosirea operatorului de deplasare la dreapta pe biţi şi apoi a operatorului aritmetic de adunare.

```
int a=3,b=5,c;
a+=b>>=c=2;          /* c=2 , b=5/2^2=1 , a=3+1=4 */
```

Exemplul 4 : Folosirea operatorului SAU EXCLUSIV pentru interschimbarea valorilor a două variabile.

```
unsigned int k=6,j=9;
k^=j^=k^=j;
```

Exemplul 5 : Folosirea operatorului ŞI pe biţi .

```
long k;
k&=~k;      /* k=0 , indiferent de valoarea sa iniţială */
```

Exemplul 6 :

```
unsigned short k;
short j;
k|=~k;     /*  k=2^16-1=65535 , indiferent de valoarea sa anterioară */
j|=~j;     /*  dacă  j  este cu semn, atunci valoarea sa devine –1 */
```

Exemplul 7:

```
int  k;
k^=k;       /* k=0 , indiferent de valoarea sa anterioară */
```

Exemplul 8:

```
int k=-25;
k<<=2;       /*  k= -25*2^2= -100 */
```

4.8 Operatorul de conversie explicită (cast)

Pentru a modifica ordinea de evaluare a operaţiilor în mod explicit, se foloseşte perechea de paranteze rotunde. Dar **perechea de paranteze rotunde poate fi folosită şi ca operator**. Acesta se numeşte *operator de conversie explicită*, în limba engleză "**cast**". El este utilizat pentru ca operatorul să poată alege explicit tipul dorit, precizând între paranteze conversia cerută. Sintaxa de utilizare este:

(tip_conversie) expresie;

Este necesară utilizarea cast-ului când se doreşte folosirea unui alt tip decât cel implicit.

Exemplul 1:

```
float a,n;
a=(int)sqrt(n);  /* se determină partea întreagă a lui radical din n */
```

Exemplul 2:

```
int a, b;
c=(double) a/b;      /* rezultatul real al împărţirii a două numere întregi */
```

4.9 Operatorul sizeof

Efectul aplicării operatorului **sizeof** este aflarea dimensiunii în octeţi a unui tip de date sau a rezultatului unei expresii. Sintaxa de utilizare este **sizeof(expresie);**

Perechea de paranteze rotunde este obligatorie. Deoarece tipul operanzilor este determinat încă de la compilare, evaluarea se poate face şi în etapa de preprocesare.

Exemplu :

```
sizeof(double);  /* are valoarea 8 */
long a[100];     /* tablou cu 100 elemente de tip long */
sizeof(a);       /* are valoarea 100*4=400 */
```

4.10 Operatorii de adresare

Se utilizează pentru obţinerea adresei unui element sau pentru obţinerea conţinutului de la o anumită adresă. Cazurile tipice în care sunt folosiţi sunt :

- accesul la câmpurile unei structuri sau ale unei uniuni
- memorarea adresei unui element
- accesul indirect la un element
- accesul la un element al unui tablou

Operatorii de adresare sunt :

[ ] indexare
. selecţie directă
→ selecţie indirectă
& determinare adresă
***** adresare indirectă

4.11 Operatorul condiţional

Este un operator cu trei operanzi. Se foloseşte pentru situaţiile în care există două variante de obţinere a rezultatului, în funcţie de îndeplinirea unei condiţii. Are sintaxa :

expr_cond ? rezultat_1 : rezultat_2;

Semnificaţia este :

daca expr_cond atunci rezultat=rezultat_1
altfel rezultat=rezultat_2

Exemplul 1:

```
double max,a,b;
max=(a>b)?a:b;  /* se determină maximul a două numere */
```

Exemplul 2:

```
unsigned k;
printf("%s",k?"diferit de 0 ":"egal cu 0");
```

/* se afişează dacă numărul k este sau nu diferit de 0 */

Exemplul 3 : Determinarea maximului a trei numere.

```
max=(a>b)?(a>c?a:c):(b>c?b:c);
```

Exemplul 4 : Tipăreşte relaţia dintre două valori întregi.

```
printf("%d %c %d\n",a,(a>b)?'>':(a<b)?'<':'=',b);
```

4.12 Operatorul virgulă

De obicei, virgula este folosită ca delimitator pentru declaratori, variabile sau parametri ai unei funcţii. O altă semnificaţie este aceea că reprezintă o expresie care evaluează componentele sale în ordine de la stânga la dreapta.

Exemplul 1:

```
int i=7,a,j=3;
a=i,i+=j;       /* a=7 , i=10 */
```

sau

```
i+=j,a=i;       /* i=10 , a=10 */
```

Exemplul 2:

```
double x, y, temp;
(x>y)?(temp=x, x=y, y=temp):y ;
/* are ca efect ordonarea crescătoare a variabilelor x şi y ; ca rezultat se
  obţine cea mai mare dintre valorile x şi y */
```

4.13 Exerciţii şi teste grilă

1. Fie expresia **a<b&&a<c**. Să se verifice afirmaţiile următoare :

a) expresia este incorectă sintactic
b) este corectă şi este echivalentă cu : **(a<b)&&(a<c)**
c) este corectă şi este echivalentă cu expresia : **a<(b&&a)<c**

2. Să se verifice utilizarea căror operaţii este permisă :

a) a-=b; **b) -a=b;**
c) -a=-b; **d) a<<=2;**

3. Fie o expresie care conţine operatori aritmetici, relaţionali şi de atribuire şi nu conţine paranteze. Să se precizeze, care va fi ordinea de evaluare a operatorilor ?

a) de atribuire, relaţionali, aritmetici
b) aritmetici,relaţionali, de atribuire
c) de atribuire, aritmetici, relaţionali

4. Care din următoarele afirmaţii sunt adevărate ?

a) operatorii pe biţi au ordinea de evaluare înaintea operatorilor logici
b) operatorii de atribuire compusă au o precedenţa mai mică decât operatorul de atribuire simplă
c) operatorii relaţionali au o precedenţă mai mare decât operatorii logici
d) toţi operatorii relaţionali au acelaşi nivel de prioritate

5. Fie declaraţia : **int i,x,y;** Să se verifice care expresii sunt corecte :

a) (i+1)++ **b) i+++ 1** **c) --x+y**
d) ++i++ **e) &x+1** **f) &(x+1)**
g) -x&1

6. Precizaţi efectul operaţiilor :

```
int x,y,z;
scanf("%d" , &x); y-=(z=x,x<0);
```

Pentru obţinerea rezultatului erau necesare parantezele rotunde ?

7. Fie declaraţia : **unsigned n;**
Care din expresiile următoare determină octetul inferior al variabilei **n** ?

a) n%256 **b) n>>4** **c) n & 0xFF**
d) (n<<4)>>4 **e) n>>8**
f) (n<<8)>>8

8. Fie definiţiile :

```
unsigned m=0xF0F,n=0xF0F0;
char format[]="\n n=%4x";
```

Să se găsească care vor fi valorile afişate după următoarele operaţii :

a) printf(format,m);
b) printf(format,n);
c) printf(format,~m);

```
d) printf(format,~n);
e) printf(format m|n);
f) printf(format,m&n);
g) printf(format,m^n);
```

9. Evaluaţi rezultatul şi corectitudinea următoarelor secvenţe :

```
      int i=1,j=2,k=-7;
      double x=0.0,y=2.5;
a) -i-5*j>=k+1      b) 3<j<5
c) i+j+k==-2*j      d) x/!!y
e) x&&i||j-3
```

10. Ce valoare vor avea variabilele **x** si **y** după secvenţa de instrucţiuni ?

```
            int x=7,y;
            y=x<<=2;
a) 32       b) 28       c) 64
```

11. Să se determine care din următoarele expresii sunt corecte :

```
a) z=x++==4||y--<5;
b) a=x==y==z;
```

12. Realizaţi cu ajutorul operatorului condiţional **"?"** urmatoarele operaţii :

a) determinarea valorii absolute a unui număr **x**
b) determinarea minimului în valoare absolută a două numere
c) determinarea maximului a trei numere

13. Ce face următorul program ?

```
#include<stdio.h>
main()
{  int i,j;
   scanf("%d% d" ,&i ,&j);
   printf("\n i=%d , j=%d", i, j);
   i=i-j , j=j+i , i=j-i;
   printf("\n i=%d , j=%d", i, j);
}
```

a) realizează un calcul algebric oarecare
b) adună valorile absolute ale lui **i** şi **j**
c) interschimbă valorile variabilelor **i** şi **j**

14. Care din cele trei expresii sunt echivalente între ele ?

```
a) 1<<n+1;          b) 1<<(n+1)
c) (1<<n)+1
```

15. Fie expresia : **(int) (sqrt(n))**
Are importanţă folosirea perechii de paranteze rotunde pentru tipul **int** ? Dar pentru **sqrt(n)** ?

16. Să se transcrie sub formă de expresii de atribuire expresiile de mai jos :

```
a) y=2x^3-8x^2+7x-1
b) y=(ax+b)/(cx+d)
c) a=max(x+y,x-y)
d) m=max(3x^2+1,6x-20)
e) x=1, 5a^3-2, 8a^2+3, 7a-1
f) x=a*b*c-(a-b)(a-c)+(b-c)
g) dacă (x+y)!=0 atunci z=x/(x+y)
altfel z=0
```

17. Să se transcrie sub formă de expresie de atribuire :

a) y=|2x+10|
b) a=max(x,y,z)
c) dacă x>y si x>z atunci u=v, altfel u=v/2
d) dacă x>y sau x>z,atunci p=a+b,altfel p=a-b
e) dacă nici una din condiţiile i>j, i>k, i>n nu are loc, atunci x=a, altfel x=b
f) dacă c are ca valoare codul ASCII al unui caracter alb, atunci i se măreşte cu 1, altfel se micşorează cu 1
g) dacă a>b>1 şi a<10, atunci x=a/b, altfel x=100
h) dacă c are ca valoare un cod diferit de codul caracterelor albe, atunci i=x+y, altfel i=x-y
i) dacă valorile variabilelor a,b si c pot reprezenta lungimile laturilor unui triunghi, atunci x are ca valoare perimetrul acestui triunghi, altfel are valoarea 0.

18. Fie declaraţia : **int x,i,j;** Să se scrie o expresie care are ca valoare întregul format cu **j** biţi din **x** , începând cu al **i**-lea bit.

19.
```
double x=1/2.0+1/2;
printf("x=%.2f\n",x);
```
Ce se va tipări după execuţia codului de mai sus ?

a) x=0.00   b) x=0.25
c) x=0.50   d) x=0.75

20. Care va fi valoarea lui **i** după execuţia următoarelor instrucţiuni :

```
      i=4;  i=i?18:2;
```

a) 4   b) 18
c) 2   d) 9

21. Se declară variabila

```
unsigned int a=0x3FF
```

Care este valoarea expresiei **a>>3** ?

a) 3FF   b) 0xBF
c) 0x7F   d) 0x3FC

22. Se dau **int a=0,b=1,c=2.** Valoarea variabilei **cp** din expresia:

cp=rez=a+b+c>c&&b*c<a+b*c

este :

a) 1 b) 0
c) 2 d) 1.5

23. Se dă variabila **b=0xFF10** de tip **unsigned int** şi expresia **b>>3**. Efectul acestei expresii este :

a) 1FF2
b) valoarea binară a variabilei b
c) 0xFF13 d) 0x1FE2

24. Se dau variabilele de tip întreg **a=0x125,b=0455,c=293,d=0xAF,x,y** şi expresia **x=y=(a==b&a==c)^d**. Variabilele **x** si **y** au valoarea :

a) AF b) 0257
c) 10101111 d) 0xAF

25. Fie expresia de mai jos :

(x>-1) && !(x>1)||(x>=5)

Intervalul selectat este :

a) x∈ (-1,1] ∪[5, ∞)
b) x ∈(-∞ ,-1)∪ [1,∞)
c) x ∈(- ∞,-1)∪ (1,5)
d) x ∈[-1,1] ∪[5,∞)

26. Care este intervalul descris de expresia logică de mai jos ?

(x<=-2)||(x>-1)&&!(x>=1)||(x>5)

a) x∈ (-∞ , -2] ∪ (-1,1] ∪ [5, ∞)
b) x ∈(-∞ ,-2] ∪ [-1,1] ∪ (5, ∞)
c) x∈(-∞ ,-2] ∪ (-1,1] ∪ [5, ∞)
d) x ∈(-∞ ,-2] ∪ (-1,1) ∪ (5, ∞)

27. Care din următorii operatori relaţionali este invalid ?

a) == b) <> c) < d) >

28. Care din următorii operatori are prioritatea cea mai mică ?

a) == b) + c) - d) !

29. După execuţia secvenţei de cod, ce valoare va avea variabila **z** ?

```
int z;
int x=5;
int y=-10;
z:=x--y;
```

a) -10 b) -5 c) 5 d) 15

30. Ce rezultat va produce operaţia :

0xB & 0x5

a) 0xE b) 0xF
c) 0x1 d) 0x8

31. Fie relaţia **4/5|3+4%5&3**. Care este rezultatul evaluării ei ?

a) 4 b) 3 c) 1 d) 1

32. Fie secvenţa de cod :

```
int x=1*2+4/4+3%5;
```

Valoarea lui **x** este :

a) se va genera eroare b) 3
c) 6 d) 5

33. Presupunând următoarea secvenţă de cod :

```
int a=5,b=2;
float c=a/b;
```

Care va fi valoarea lui **c** ?

a) 2.5 b) 2 c) 3 d) 1

34. Avem următorul program :

```
int main()
{  float x, y, z;
    x=1.3; y=1.2; z=x%y;
    return 0;
}
```

La sfârşitul programului variabila **z** va avea valoarea :

a) 0.1 b) 0 c) 1
d) programul generează eroare la compilare

35. Fie secvenţa :

```
double x,y;
…….
x=9; y=3(x-10);
……..
```

Care afirmaţie este adevarată :

a) y=-3 b) y=3
c) eroare la compilare deoarece lipseşte operator
d) eroare la compilare deoarece nu se poate scădea un double

36. Urmăriţi secvenţa de mai jos şi precizaţi valoarea variabilei **y**:

```
int a, b=3;
int x=2;
int z=2*b+x;
```

a) 2 b) 3 c) 4 d) 5
e) secvenţa este eronată

37. Ştiind că în conformitate cu standardul ASCII literele mari au codurile succesive începând cu 65, precizaţi care dintre variabilele **x**, **y**, **z** şi **u** vor avea valoarea –2 la finele execuţiei programului de mai jos.

```
void main()
{
    int x=-3, y=1, z=2, u;
    x++;
    y+=2;
    z-=1+sqrt(y+2);    // radical de ordin doi
```

```
      u='A'-'C';
}
```
a) x b) y c) z d) u e) nici una

38. Se consideră variabilele întregi **x**,**y** şi **z**, fiind cunoscute valorile **x=4** şi **y=2**. Care dintre expresiile de mai jos are valoarea 0 ?

a) x+y>x%y+1 b) z=(x-y!=0)
c) x-2*y=0 d) !x
e) x && y

39. Ştiind că **a**, **b**, **c**, **d**, **x** sunt variabile reale cu **a<=b** şi **c<=d**, scrieţi sub formă de expresie : **x∈[a, b]** sau **x∈[c, d]**.

a) (x>=a||x<=b)&&(x>=c||x<=d)
b)
((x>=a)&&(x<=b))||((x>=c)&&(x<=d))
c)
(!(x<a)&&!(x>b))||(!(x<c)&&!(x>d))
d)
((x>=a)||(x<=b))&&((x>=c)||(x<=d))
e) (x>=a && x<=b) || (x>=c && x<=d)

40. Fie declaraţiile de variabile:

```
      int x=4, z=13;
      float z;
```

Care dintre instrucţiunile de mai jos nu atribuie corect valoarea 8.5 variabilei **z** ?

a) z=(x+y)/2.;
b) z=((float)x+y)/2:
c) z=(x+y.)/2;
d) z=(x+y)/(float)2;
e) z=(float)(x+y)/2;

41. Ce afişează programul următor, dacă valoarea citită de la tastatură este 2 ?

```
#include<stdio.h>
void main()
{
      int x,y,z;
      scanf("%d", &x);
      y=--x;
      y+=3;
      z=x-2*y++;
      printf("%d", z++);
}
```
a) −9 b) −8 c) −7
d) −6 e) −5

42. Fie trei variabile întregi **a**, **b**, **x**. Precizaţi care dintre expresiile condiţionale de mai jos nu este o transcriere corectă a enunţului: „*dacă numărul y este pozitiv, atunci x ia valoarea lui a, în caz contrar x ia valoarea lui b*”.

a) x=(y>0) ? a : b;
b) x=y>0 ? b: a;
c) x=!y>0 ? b : a;
d) y>0 ? x=a :x=b;
e) !(y>0) ? x=b : x=a;

43. Ce valoare afişează programul următor ?

```
#include<stdio.h>
void main()
{
     int x=5, y;
     y=(sizeof(x-1)==sizeof(int)) ?
         sizeof('x') : sizeof(3);
     printf("%d", y);
}
```
a) 3 b) 1 c) 2 d) 4
e) programul este eronat

44. Ce valori va afişa programul următor ?

```
#include<stdio.h>
void main()
{
  int a=10, b=6, c=4, d;
  d=(c=a-6, a=b%c, b+=a, a/2);
  printf("\n%d %d %d %d",a,b,c,d);
}
```
a) 0 16 −6 5 b) 2 8 4 1
c) 4 2 8 1 d) −6 0 16 5
e) alte valori

45. Ce valoare afişează programul de mai jos ?

```
#include<stdio.h>
void main()
{
      int a=3, b=2, n=4, x;
      x=(a<<n) + (a&b) + (n|b);
      printf("%d", x);
}
```
a) 2 b) 8 c) 51
d) 56 e) programul este greşit

46. Fie variabilele întregi **a=1,b=2,c=3,d=4**. Care dintre construcţiile de mai jos sunt expresii scrise corect, cu valoarea 0 ?

a) !d b) a+b<d c) a*b+c
d) a=b<c e) (a<b)!=(b<c)

47. Pentru care dintre seturile de valori ale variabilelor **x, y, z** de mai jos expresia **(x<y)<((z!=x)<((z-y)<x)** are valoarea 1?

a) x=3; y=5; z=4 b) x=4; y=3; z=4
c) x=3; y=4; z=3 d) x=5; y=4; z=3
e) x=5; y=5; z=5

48. Care dintre următoarele expresii au valoarea 1 dacă şi numai dacă valorile variabilelor întregi **x** şi **y** sunt numere pare ?

a) x-y==2 b) x*y%4==0
c) (x+y)%2==0 d) y%x==2
e) (x%2==0)&&(y%2==0)

49. Care dintre următoarele expresii sunt adevărate dacă şi numai dacă valorile

variabilelor **x** şi **y** sunt numere naturale consecutive ?

```
a) x-y==1        b) (x==1)&&(y==2)
b) (x-y==1)&&(y-x==1)
c) y==x+1
e) (x-y==1)||(y-x==1)
```

50. Se consideră următoarele declaraţii de variabile : `int a,b,e; float c,d;`. Care dintre instrucţiunile de mai jos sunt incorecte ?

```
a) a=a*b;          b) e=a<c;
c) –b=a+a/b;       d) d=(a+b)/2;
e) c*d=a-b;
```

51. Fie variabilele **x,y,z** de tipul **int**, fiind cunoscute valorile iniţiale **x=3, y=5**. Care dintre instrucţiunile de mai jos trebuie executată astfel încât, după execuţie, valoarea variabilei **z** să fie 21 ?

```
a) z=2*x+3*y--;    b) z=2*x+3*--y;
c) z=2*x--+3*y;    d) z=2*--x+3*y;
e) z=2*x+3*y;
```

52. Fie trei variabile întregi **a,b,x**. Scrieţi cu ajutorul unei expresii condiţionale enunţul „*dacă x nu aparţine [a,b] , atunci x ia valoarea lui a, în caz contrar x ia valoarea lui b*”.

```
a) x=((x<a)||(x>b))? a : b;
b) x=(x<a||x>b)? a : b;
c) x=((x<a)&&(x>b))? a : b;
d) x=(x<a)||(x>b)? b : a;
e) (x<a)||(x>b))? (x=a) : (x=b);
```

53. Ştiind că în standardele ASCII caracterele literă mare au codurile succesive începând cu 65, deduceţi ce valoare va afişa programul următor.

```
#include<stdio.h>
void main()
{
      int x,y,z,p;  char m,n;
      m='C'; n='A';
      x=m; y=2*m-n; z=3;
      p=x<y?(y<z ? z:y):(z<x?x:z);
      printf("\n%d", p);
}
```

a) 1 b) 3 c) 69 d) 67 e) 0

54. Ştiind că valorile de tipul **int** se memorează pe 2 octeţi, iar cele de tipul **float** pe 4 octeţi, de câte ori va afişa programul următor valoarea 2?

```
#include<stdio.h>
void main()
{
      int x;  char c;
      x='A';
      printf("%d", sizeof(x));
      c='A';
      printf("%d",sizeof(c));
      printf("%d",sizeof(float)-2);
      x=sizeof(int); x=++x/2;
      printf("%d", x==2);
}
```

a) nici o data b) o data
c) de 2 ori d) de 3 ori
e) de 4 ori

55. Ce valori afişează programul următor ?

```
#include<stdio.h>
void main()
{
      int x=10, y=6, m, n, p;
      n=(m=x++, y++, p=x+y);
      printf("\n%d %d %d",m n,p);
}
```

a) 10 18 16 b) 11 18 18
c) 10 18 18 d) 11 18 17
e) 10 18 17

56. Ce valoare putem introduce la citirea variabilei **y**, astfel încât programul de mai jos să tipărească 1 ?

```
#include<stdio.h>
void main()
{
   int x=2, y, z;
   scanf("%d", &y);
   z=y+3*x++;
 printf("\n%d",(z%2==0&&x>=1)?1:0);
}
```

a) 2 b) 3 c) 4
d) orice valoare pară
e) orice valoare impară

57. Fie variabilele **a,b,c** de tipul **int**, cu valorile **a=11, b=5, c=7**. Care dintre expresiile de mai jos are valoarea 1?

```
a) (a|~b)&1        b) (~a&b)|1
c) (a&~b)|1        d) (a&b)|~1
e) (~a|b)&1
```

58. Precizaţi valoarea pe care o va avea variabila **c** în urma execuţiei programului de mai jos:

```
#include<stdio.h>
void main()
{
      char c='d';
      int n=99;
      c=n+1=c-1;
}
```

a) 'd' b)'c' c) 'b'
d) NULL e) atribuirea este greşită

Cap.5 Instrucţiunile limbajului C

Limbajul C a fost prevăzut cu instrucţiuni menite să permită realizarea simplă a structurilor proprii programării structurate. Structura secvenţială se realizează cu ajutorul instrucţiunii **compuse**, structura alternativă cu ajutorul instrucţiunii **if**, structura repetitivă condiţionată anterior, prin intermediul instrucţiunilor **while** şi **for**, structura selectivă cu ajutorul instrucţiunii **switch**, iar structura repetitivă condiţionată posterior cu ajutorul instrucţiunii **do-while**.

Limbajul C are şi alte instrucţiuni care reprezintă elemente de bază în construirea structurilor amintite mai sus. Astfel de instrucţiuni sunt : instrucţiunea **expresie** şi instrucţiunea **vidă**. Alte instrucţiuni prezente în limbaj asigură o flexibilitate mai mare în programare. Acestea sunt instrucţiunile: **return**, **break**, **continue** şi **goto** .

5.1 Instrucţiunea vidă

Se reduce la punct şi virgulă (**;**). Ea nu are nici un efect. Instrucţiunea vidă se utilizează în construcţii care cer prezenţa unei instrucţiuni, dar nu trebuie să se execute nimic în punctul respectiv. Astfel de situaţii apar frecvent în cadrul structurii alternative şi repetitive, aşa cum se va vedea în continuare.

5.2 Instrucţiunea expresie

Se obţine scriind punct şi virgulă după o expresie. Deci, instrucţiunea expresie are sintaxa :

expresie ;

În cazul în care expresia din compunerea unei instrucţiuni expresie este o expresie de atribuire, se spune că instrucţiunea respectivă este o instrucţiune de **atribuire**. Un alt caz frecvent utilizat este acela când expresia este un operand ce reprezintă apelul unei funcţii. În acest caz, instrucţiunea expresie este o instrucţiune de **apel** a funcţiei respective.

Exemplul 1:

```
int x;.....   x=10;
```

Este o instrucţiune de atribuire, variabilei **x** i se atribuie valoarea 10.

Exemplul 2:

```
double y;..... y=y+4;
```
sau
```
y+=4;
```

Este o instrucţiune de atribuire, care măreşte valoarea lui **y** cu 4.

Exemplul 3:

```
putch(c-'a'+'A');
```

Este o instrucţiune de apel a funcţiei **putch**.

Exemplul 4:

```
double a;   .....   a++;
```

Este o instrucţiune expresie, care măreşte valoarea lui **a** cu unu.

Exemplul 5:

```
double a; ..... ++a;
```

Are acelaşi efect ca şi instrucţiunea expresie din exemplul precedent.

Observaţie : Nu orice expresie urmată de punct şi virgulă formează o instrucţiune expresie efectivă. De exemplu construcţia **a;** deşi este o instrucţiune expresie, ea nu are nici un efect.

Exemplul 6 : Să se scrie un program care citeşte valorile variabilelor **a, b, c, d, x** de tip **double** şi afişează valoarea expresiei **$(a*x^2+b*x+c)/(a*x^2+b*x+d)$** dacă numitorul este diferit de 0 şi 0 în caz contrar.

```
#include<stdio.h>
#include<conio.h>
void main()
{  double a,b,c,d,x;
   doube y,z,u;
   clrscr();
   printf("a="); scanf("%lf",&a);
   printf("b="); scanf("%lf",&b);
```

```
    printf("c="); scanf("%lf",&c);
    printf("d="); scanf("%lf",&d);
    printf("x="); scanf("%lf",&x);
    y=a*x*x;
    z=b*x;
    u=y*x+z+d;
    printf("exp=%g\n",u ?(y+z+c)/u:u);
    getch();
}
```

5.3 Instrucţiunea compusă

Instrucţiunea compusă este o succesiune de instrucţiuni incluse între acolade, succesiune care poate fi precedată şi de declaraţii :

```
{
    declaraţii
    instrucţiuni
}
```

Dacă declaraţiile sunt prezente, atunci ele definesc variabile care sunt definite atât timp cât controlul programului se află la o instrucţiune din compunerea instrucţiunii compuse.

Exemplu : Presupunem că într-un anumit punct al programului este necesar să se permute valorile variabilelor întregi **a** şi **b**. Aceasta se poate realiza astfel :

```
{   int t;
    t=a;a=b;b=t;
}
```

Variabila **t** este definită din momentul în care controlul programului ajunge la prima instrucţiune din instrucţiunea compusă (**t=a;**). După execuţia ultimei instrucţiuni a instrucţiunii compuse, variabila **t** nu mai este definită (nu mai ocupă memorie).

Instrucţiunea compusă se utilizează unde este necesară prezenţa unei instrucţiuni, dar procesul de calcul din punctul respectiv este mai complex şi se exprimă prin mai multe instrucţiuni. În acest caz instrucţiunile respective se includ între acolade pentru a forma o instrucţiune compusă. Acest procedeu de a forma o instrucţiune compusă din mai multe instrucţiuni se utilizează frecvent în construirea structurilor alternative şi ciclice.

5.4 Instrucţiunea if

Are următoarele formate :

format1 :
```
if(expresie)
        instrucţiune;
```

format2:
```
if(expresie)
    instrucţiune1;
else
    instrucţiune2;
```

La întâlnirea instrucţiunii **if** întâi se evaluează expresia din paranteze. Apoi, în cazul formatului 1, dacă expresia are valoarea diferită de zero (adică **true**) se execută **instrucţiune**, altfel se trece în secvenţă la instrucţiunea următoare instrucţiunii **if**. În cazul formatului 2, dacă expresia are o valoare diferită de zero, atunci se execută **instrucţiune1** şi apoi se trece în secvenţă la instrucţiunea aflată după **instrucţiune2**, altfel (condiţia este zero, adică **false**) se execută **instrucţiune2**.

În mod normal, în ambele formate, după execuţia instrucţiunii **if** se ajunge la instrucţiunea următoare ei. Cu toate acestea, este posibilă şi o altă situaţie când instrucţiunile din compunerea lui **if** definesc ele însele un alt mod de continuare a execuţiei programului.

Deoarece o instrucţiune compusă este considerată ca fiind un caz particular de instrucţiune, rezultă că instrucţiunile din compunerea lui **if** pot fi instrucţiuni compuse. De asemenea, instrucţiunile respective pot fi chiar instrucţiuni **if**. În acest caz se spune că instrucţiunile **if** sunt **imbricate**.

Exemplul 1: Se dă funcţia : $\mathbf{y=3x^2+2x-10}$, pentru $\mathbf{x>0}$ şi $\mathbf{y=5x+10}$, pentru $\mathbf{x<=0}$. Să se scrie un program care citeşte valoarea lui **x** şi afişează valoarea lui **y**.

```
#include<stdio.h>
#include<conio.h>
void main()
{  float x,y;
   clrscr();
   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();
}
```

Exemplul 2: Să se scrie un program care citeşte valoarea lui **x**, calculează şi afişează valoarea lui **y** definită ca mai jos :

$$y=\begin{cases} 4x^3+5x^2-2x+1, & \text{pentru } x<0 \\ 100, & \text{pentru } x=0 \\ 2x^2+8x-1, & \text{pentru } x>0 \end{cases}$$

```
#include<stdio.h>
#include<conio.h>
void main()
{  float x,y,a;
   clrscr();
   scanf("%f",&x);
   a=x*x;
   if(x<0)
     y=4*x*a+5*a-2*x+1;
   else
     if(!x)
       y=100;
     else
       y=2*a+8*x-1;
   printf("x=%f\ty=%f\n",x,y);
   getch();
}
```

Exemplul 3: Să se scrie un program care citeşte valorile variabilelor neîntregi **a** şi **b**, calculează rădăcina ecuaţiei : **ax+b=0** şi afişează rezultatul.

```
#include<stdio.h>
#include<conio.h>
void main()
{  double a,b;
   clrscr();
   if(scanf("%lf %lf",&a,&b)!=2)
   /* validează numărul coeficienţilor reali citiţi */
      printf("coeficienti eronati\n");
   else
      if(a)
           printf("a=%g\tb=%g\tx=%g\n",a,b,-b/a);
      else
           if(!b)
                printf("ecuatie nedeterminata\n");
           else
                printf("ecuatia nu are solutie\n");
   getch();
}
```

Programul de mai sus conţine o instrucţiune **if** imbricată. Pentru a mări claritatea programelor se obişnuieşte să se decaleze spre dreapta (cu un tabulator) instrucţiunile din compunerea instrucţiunii **if**. În acest program s-a realizat test relativ la valorile tastate pentru **a** şi **b**. Dacă funcţia **scanf()** nu

returnează valoarea 2, înseamnă că nu s-au tastat două numere de la terminal. De aceea, în astfel de situaţii se afişează mesajul "*coeficienţi eronaţi*".

Exemplul 4: Să se scrie un program care citeşte coeficienţii **a,b,c,d,e,f** ai unui sistem de două ecuaţii lineare cu două necunoscute, determină şi afişează soluţia acestuia atunci când are o soluţie unică.

```
#include<stdio.h>
#include<conio.h>
void main()
{  double a,b,c,d,e,f,x,y,det1,det2,det;
   clrscr();
   if(scanf("%lf%lf%lf%lf%lf%lf",&a,&b,&c,&d,&e,&f)!=6)
      printf("coeficienti eronati\n");
   else
      if(!(det=a*e-b*d))
           printf("sistemul nu este determinat\n");
      else
      {   det1=c*e-b*f;
          det2=a*f-c*d;
          x=det1/det;
          y=det2/det;
          printf("x=%g\ty=%g\n",x,y);
      }
   getch();
}
```

5.5 Funcţia standard exit

Funcţia **exit** are prototipul : **void exit(int *cod*);**

şi el se află în fişierele header **stdlib.h** şi **process.h**. La apelul acestei funcţii au loc următoarele acţiuni:

- se videază zonele tampon (bufferele) ale fişierele deschise în scriere
- se închid toate fişierele deschise
- se întrerupe execuţia programului

Parametrul acestei funcţii defineşte starea programului la momentul apelului. Valoarea 0 defineşte o terminare normală a execuţiei programului, iar o valoare diferită de 0 semnalează prezenţa unei erori (terminarea anormală a execuţiei programului).

În concluzie, putem apela funcţia **exit** pentru a termina execuţia unui program, indiferent de faptul că acesta se termină normal sau din cauza unei erori.

Exemplu : Să se scrie un program care citeşte valorile variabilelor **a,b,c**, calculează şi afişează rădăcinile ecuaţiei de gradul 2 : $\mathbf{ax^2+bx+c=0}$.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
void main()
{  double a,b,c,d,delta;
   clrscr();
   printf("coeficientul lui x patrat : ");
   if(scanf("%lf",&a)!=1)
   {  printf("coeficientul lui x patrat eronat\n");
      exit(1); /* terminare la eroare*/
   }
   printf("coeficientul lui x : ");
   if(scanf("%lf",&b)!=1)
   {  printf("coeficientul lui x eronat\n");
      exit(1);
   }
   printf("termenul liber : ");
   if(scanf("%lf",&c)!=1)
   {  printf("termenul liber eronat\n");
```

```
      exit(1);
   }
 /* afişează coeficienţii citiţi */
   printf("a=%g\tb=%g\tc=%g\n",a,b,c);
   if(!a && !b && !c)
   {  printf("ecuatie nedeterminata\n");
      exit(0); /* terminare fără erori*/
   }
   if(!a&&!b)
   {  printf("ecuatia nu are solutie\n");
      exit(0);
   }
   if(!a)
   {  printf("ecuatie de gradul 1\n");
      printf("x=%g\n",-c/b);
      exit(0);
   }
   delta=b*b-4*a*c;
   d=2*a;
   if(delta>0)
   {  printf("ecuatia are doua radacini reale si
            distincte\n");
      delta=sqrt(delta);
      printf("x1=%g\tx2=%g\n",(-b+delta)/d,
                                (-b-delta)/d);
      exit(0);
   }
   if(!delta)
   {  printf("ecuatia are radacina dubla\n");
      printf("x=%g\n",-b/d);
      exit(0);
   }
   printf("radacini complexe\n");
   delta=sqrt(-delta)/d;
   d=-b/d;
   printf("x1=%g+i(%g)\n",d,delta);
   printf("x2=%g-i(%g)\n",d,delta);
   getch();
}
```

5.6 Instrucţiunea while

Are sintaxa : **while(expresie) instrucţiune;**

Prima parte din acest format constituie antetul instrucţiunii **while**, iar **instrucţiune** este corpul ei. La întâlnirea acestei instrucţiuni întâi se evaluează expresia din paranteze. Dacă ea are valoarea *true* (este diferită de 0), atunci se execută **instrucţiune**. Apoi se revine la punctul în care se evaluează din nou valoarea expresiei din paranteze. În felul acesta, corpul ciclului se execută atât timp cât expresia din antetul ei este diferită de 0. În momentul în care **expresie** are valoarea 0, se trece la instrucţiunea următoare instrucţiunii **while**.

Corpul instrucţiunii **while** poate să nu se execute niciodată. Într-adevăr dacă **expresie** are valoarea 0 de la început, atunci se trece la instrucţiunea următoare instrucţiunii **while** fără a executa niciodată corpul instrucţiunii respective.

Corpul instrucţiunii **while** este o singură instrucţiune care poate fi compusă. În felul acesta avem posibilitatea să executăm repetat mai multe instrucţiuni grupate într-o instrucţiune compusă. Corpul instrucţiunii **while** poate fi o altă instrucţiune **while** sau să fie o instrucţiune compusă care să conţină instrucţiunea **while**. În acest caz se spune că instrucţiunile **while** respective sunt imbricate.

Instrucţiunile din corpul unei instrucţiuni **while** pot să definească un alt mod de execuţie a instrucţiunii **while** decât cel indicat mai sus. Astfel, se poate realiza terminarea execuţiei instrucţiunii **while** fără a se mai ajunge la evaluarea expresiei din antetul ei. De exemplu, dacă în corpul unei

instrucţiuni **while** se apelează funcţia **exit**, atunci se va termina execuţia ciclului **while**, deoarece se întrerupe chiar execuţia programului. Despre instrucţiunea **while** se spune că este o instrucţiune *ciclică condiţionată anterior*.

Exemplul 1: Să se scrie un program care calculează şi afişează valoarea polinomului $\mathbf{p(x)=3x^2-7x-10}$ pentru **x=1,2,….,10**.

```
#include<stdio.h>
#include<conio.h>
void main()
{  int x;
   clrscr();
   x=1;
   while(x<=10)
   {  printf("x=%d\tp(x)=%d\n",x,3*x*x-7*x-10);
      x++;
   }
   getch();
}
```

Exemplul 2: Să se scrie un program care citeşte un şir de întregi separaţi prin caractere albe şi afişează suma lor. După ultimul număr se va tasta un caracter alb urmat de un caracter nenumeric (de exemplu, caracterul sfârşit de fişier **<Ctrl>+Z**, o literă, etc.), iar după aceea se va acţiona tasta **Enter**.

```
#include<stdio.h>
#include<conio.h>
void main()
{  int i,s=0;
   clrscr();
   while(scanf("%d",&i)==1)
     s+=i;
   printf("suma=%d\n",s);
   getch(); }
```

Exemplul 3: Să se scrie un program care citeşte un întreg $\mathbf{n\in[0,170]}$, calculează şi afişează pe **n!**. Avem: **n!=1*2*3*…..*n** , pentru **n >0** şi **0!=1** , prin definiţie.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{  int n,i;
   double f;
   clrscr();
   printf("n=");
   if(scanf("%d",&n)!=1)
   {  printf("nu s-a tastat un intreg\n");
      exit(1);
   }
   if(n<0 || n>170)
   {  printf("n nu apartine intervalului [0,170]\n");
      exit(1);
   }
   f=1.0;
   i=2;
   while(i<=n)
     f*=i++;
   printf("n=%d\tn!=%g\n",n,f);
   getch();
}
```

5.7 Instrucţiunea for

Instrucţiunea **for**, ca şi instrucţiunea **while**, se utilizează pentru a realiza o structură *repetitivă condiţionată anterior*. Are sintaxa :

for(exp1 ; exp2 ; exp3) /* antet */
instrucţiune; /* corpul ciclului*/

unde **exp1**, **exp2** şi **exp3** sunt expresii.

Expresia **exp1** se numeşte partea de **iniţializare** a ciclului **for**, iar **exp3** este partea de **reiniţializare** a lui. Expresia **exp2** este condiţia de **terminare** a ciclului **for** şi ea joacă acelaşi rol cu expresia din ciclul **while**. Instrucţiunea **for** se execută astfel :

- Se execută secvenţa de iniţializare definită de **exp1**
- Se evaluează **exp2**. Dacă are o valoare diferită de 0 (este *true*), atunci se execută instrucţiunea care formează corpul ciclului. Altfel, (expresia are valoarea 0 adică *false*) se termină execuţia instrucţiunii **for** şi se trece la instrucţiunea următoare.
- După executarea corpului ciclului se execută secvenţa de reiniţializare definită de **exp3**. Apoi se reia execuţia de la pasul 2 .

Ca şi în cazul instrucţiunii **while**, instrucţiunea din corpul ciclului **for** nu se execută niciodată dacă **exp2** are valoarea 0 chiar de la început. Expresiile din antetul lui **for** pot fi şi vide. Caracterele punct şi virgulă vor fi întotdeauna prezente. În general, instrucţiunea **for** poate fi scrisă cu ajutorul unei secvenţe în care se utilizează instrucţiunea **while** astfel :

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

Această echivalare nu are loc într-un singur caz şi anume atunci când, în corpul instrucţiunii se utilizează instrucţiunea **continue**. Reciproc, orice instrucţiune **while** poate fi scrisă cu ajutorul unei instrucţiuni **for** în care **exp1** şi **exp3** sunt vide. Astfel, instrucţiunea **while(exp) instrucţiune;** este echivalentă cu instrucţiunea **for(; exp ;) instrucţiune;** .

O instrucţiune **for** de forma **for(; ;) instrucţiune;** este validă şi este echivalentă cu instrucţiunea : **while(1) instrucţiune;**. Un astfel de ciclu se poate termina prin alte mijloace decât cel obişnuit, cum ar fi instrucţiunea de revenire dintr-o funcţie, un salt la o etichetă etc. Din cele de mai sus rezultă echivalenţa celor două cicluri **while** şi **for**. Se recomandă folosirea instrucţiunii **for** în ciclurile în care sunt prezente părţile de iniţializare şi reiniţializare, aşa numitele *cicluri cu pas*.

Exemplul 1: Să se scrie un program care citeşte întregul **n** din intervalul **[0,170]**, calculează şi afişează pe **n!** .

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{  int n,i;
   double f;
   clrscr();
   printf("n=");
   if(scanf("%d",&n)!=1)
   {  printf("nu s-a tastat un intreg\n");
      exit(1);
   }
   if(n<0 || n>170)
   {  printf("n nu apartine intervalului [0,170]\n");
      exit(1);
   }
   for(f=1.0,i=2;i<=n;i++) f*=i;
   printf("n=%d\tn!=%g\n",n,f);
   getch();
}
```

Exemplul 2: Următorul program continuă să cicleze până când este tastată litera **q**. În loc să verifice variabila de control a ciclului, instrucţiunea **for** verifică dacă de la tastatură a fost introdus caracterul **q**.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int i;
```

```
        char ch;
        clrscr();
        ch='a';
        for(i=0;ch!='q';i++)
        {    printf("pas: %d\n",i);
             ch=getche();
        }
        getch();
}
```

În acest caz, testul de condiţie care controlează ciclul nu are nimic în comun cu variabila de control a ciclului. Variabila **ch** a primit o valoare iniţială pentru a ne asigura că ea nu conţine accidental chiar litera **q** în momentul în care programul începe execuţia.

Exemplul 3: O altă variantă a lui **for** este aceea că scopul său poate fi gol ca în programul următor:

```
#include<stdio.h>
#include<conio.h>
void main()
{
     char ch;
     clrscr();
     for(ch=getche();ch!='q';ch=getche());
     printf("\n am gasit pe q");
     getch();
}
```

Instrucţiunea care asignează lui **ch** o valoare a fost mutată în interiorul ciclului. Aceasta înseamnă că atunci când ciclul porneşte, este apelată funcţia **getche().** Apoi valoarea lui **ch** este testată împreună cu **q**, se execută inexistentul corp al lui **for** şi apoi are loc o nouă apelare a funcţiei **getche()** în secţiunea de incrementare a ciclului. Acest proces se repetă până când utilizatorul introduce litera **q**. Scopul lui **for** poate fi gol deoarece limbajul C admite instrucţiunea vidă.

5.8 Instrucţiunea do-while

Realizează structura ciclică condiţionată posterior. Această instrucţiune poate fi realizată cu ajutorul celorlalte instrucţiuni definite până în prezent. Cu toate acestea, prezenţa ei în limbaj măreşte flexibilitatea în programare. Sintaxa ei este :

```
do
   instrucţiune      /* corpul ciclului */
while(expresie);
```

Instrucţiunea se execută în felul următor : se execută **instrucţiune**, se evaluează **expresie**; dacă aceasta are o valoare diferită de 0 (*true*) atunci se revine la execuţia instrucţiunii, altfel (expresia are valoarea 0) se trece în secvenţă, la instrucţiunea următoare instrucţiunii **do-while**.

Se observă că în cazul acestei instrucţiuni întâi se execută **instrucţiune** şi apoi se testează condiţia de repetare a execuţiei ei. Instrucţiunea **do-while** este echivalentă cu secvenţa :

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

În cazul instrucţiunii **do-while** corpul ciclului se execută cel puţin o dată, spre deosebire de cazul instrucţiunii **while** şi **for**, când este posibil să nu execute niciodată.

Exemplul 1: Să se scrie un program care afişează factorii primi ai unui număr întreg nenegativ introdus de la tastatură.

```
#include<stdio.h>
#include<conio.h>
void main()
{  unsigned n,d=2,e;
   clrscr();
   printf("\nIntroduceti numarul : ");
   scanf("%u",&n);
   printf("     Desc. in factori primi este :\n");
   do
   {  e=0;
      while(!(n%d))
```

```
        {  n=n/d;
           e++;
        }
        if(e) printf("%d ^ %d\n",d,e);
        d=d+((d==2) ?1:2);
     }while(n>1);
     getch();
}
```

Exemplul 2: Faptul că ciclul **do** execută întotdeauna corpul ciclului cel puţin o dată face ca el să fie perfect pentru a verifica intrarea într-un meniu. De exemplu, următorul program va repeta ciclul de citire a opţiunii până când utilizatorul va introduce un răspuns valid:

```
#include<stdio.h>
#include<conio.h>
void main()
{
     float a,b;
     char ch;
     clrscr();
     printf("Doriti : \n");
     printf("Adunare,Scadere,Multiplicare,Impartire? \n");
     do{
           printf("Introduceti prima litera : ");
           ch=getche();
           printf("\n");
     }while(ch!='A' && ch!='S' && ch!='M' && ch!='I');
     printf("Introduceti primul numar : ");
     scanf("%f",&a);
     printf("Introduceti al doilea numar : ");
     scanf("%f",&b);
     if(ch=='A')
        printf("%f",a+b);
     else if(ch=='S')
             printf("%f",a-b);
          else  if(ch=='M')
                   printf("%f",a*b);
                else if(b) printf("%f",a/b);
    getch();
}
```

Exemplul 3: Ciclul **do** este folositor în special când programul aşteaptă să se producă un anumit eveniment. De exemplu, următorul program aşteaptă ca utilizatorul să tasteze litera **q**. Programul conţine o singură apelare a funcţiei **getche()**.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     char ch;
     clrscr();
     do{
           ch=getche();
     } while(ch!='q');
     printf("\n am gasit pe q");
     getch();
}
```

Exemplul 4: Funcţia **kbhit()** din header-ul **conio.h** este foarte folositoare când dorim să permitem utilizatorului să întrerupă o rutină fără să îl forţăm să răspundă la un mesaj. De exemplu, programul următor afişează o tabelă cu procentul 5% aplicat unor valori care se incrementează cu 20. Programul continuă să afişeze tabela până când utilizatorul apasă o tastă sau până când a fost atinsă valoarea maximă.

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

```
void main()
{   double amount;
     amount=20.0;
     printf("Apasati o tasta pentru stop.\n");
     do{
          printf("valoarea:%lf,procent:%lf\n",amount,amount*0.05);
          if(kbhit()) break;
          amount=amount+20;
     }while(amount<10000);
}
```

5.9 Instrucţiunea continue

Această instrucţiune se poate utiliza numai în corpul unui ciclu. Ea permite abandonarea iteraţiei curente. Sintaxa ei este : **continue;**. Efectul instrucţiunii este următorul :

a) În corpul instrucţiunilor while şi do-while

La întâlnirea instrucţiunii **continue** se abandonează iteraţia curentă şi se trece la evaluarea **expresiei** care stabileşte continuarea sau terminarea ciclului respectiv (expresia inclusă între paranteze rotunde şi care urmează după cuvântul cheie **while**).

b) În corpul instrucţiunii for

La întâlnirea instrucţiunii **continue** se abandonează iteraţia curentă şi se trece la execuţia pasului de reiniţializare.

Instrucţiunea **continue** nu este obligatorie. Prezenţa ei măreşte flexibilitatea în scrierea programelor C. Ea conduce adesea la diminuarea nivelurilor de imbricare ale instrucţiunilor **if** utilizate în corpul ciclurilor.

Exemplul 1: Instrucţiunea **continue** este opusă instrucţiunii **break**. Ea forţează ca următoarea iteraţie a ciclului să aibă loc trecând peste instrucţiunile dintre ea şi testul de condiţie. De exemplu, următorul program nu va afişa nimic niciodată:

```
#include<stdio.h>
void main()
{
    int x;
    for(x=0;x<100;x++)
    {  continue;
       printf("%d",x);
    }
}
```

Instrucţiunea **continue** este folosită rareori, nu pentru că folosirea ei nu ar fi o practică bună, ci pentru că aplicaţiile în care ar putea fi utilizată sunt mai rare.

Exemplul 2: O bună utilizare a lui **continue** este aceea de a porni din nou o secvenţă de instrucţiuni atunci când apare o eroare. De exemplu, programul următor calculează suma totală a numerelor introduse de utilizator. Înainte de a aduna o valoare la suma totală, el verifică dacă numărul a fost introdus corect. Dacă numărul nu a fost introdus corect, porneşte din nou ciclul, folosind instrucţiunea **continue**.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i, total;
    char ch;
    clrscr();
    total=0;
    do{
           printf("numarul urmator(0 pentru stop): ");
           scanf("%d",&i);
           printf("Este %d corect ? (Y/N) ",i);
           ch=getche();
           printf("\n");
           if(ch=='N') continue;
```

```
            total+=i;
        }while(i);
        printf("Totalul este %d\n",total);
        getch();
}
```

5.10 Instrucţiunea break

Este înrudită cu instrucţiunea **continue**. Ea are formatul : **break;**
Poate fi utilizată în corpul unui ciclu. În acest caz, la întâlnirea instrucţiunii **break** se termină execuţia ciclului în al cărui corp este inclusă şi execuţia continuă cu instrucţiunea următoare instrucţiunii ciclice respective. Această instrucţiune, la fel ca şi instrucţiunea **continue**, măreşte flexibilitatea la scrierea programelor în limbajele C şi C++ .
Exemplul1: Să se scrie un program care citeşte două numere naturale şi pozitive, calculează şi afişează cel mai mare divizor comun al lor.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{  long m,n;
   long a,b;
   long r;
   clrscr();
   do
   {  printf("primul numar=");
      if(scanf("%ld",&m)==1 && m>0) break;
      printf("nu s-a tastat un intreg pozitiv\n");
   } while(1);
   do
   { printf("al doilea numar=");
      if(scanf("%ld",&n)==1 && n>0)
      break;
      printf("nu s-a tastat un intreg pozitiv\n");
   } while(1);
   a=m; b=n;
   do   /* algoritmul lui Euclid */
   {  r=a%b;
      if(r)
      {  a=b; b=r;}
   }while(r);
   printf("(%ld,%ld)=%ld\n",m,n,b);
   getch();
}
```

Exemplul 2: Instrucţiunea **break** permite ieşirea dintr-un ciclu, din oricare punct din interiorul său. Când instrucţiunea **break** este întâlnită în interiorul unui ciclu, acesta se va termina imediat, iar controlul va trece la instrucţiunea ce urmează după ciclu. De exemplu, programul următor afişează numai numerele de la 1 la 10.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    for(i=1;i<100;i++)
    {   printf("%d",i);
        if(i==10) break;
    }
    getch();
}
```

Exemplul 3: Instrucţiunea **break** poate fi folosită cu oricare din cele trei cicluri ale limbajului. Într-un ciclu pot exista oricâte instrucţiuni **break**. În general, se recomandă ca instrucţiunea **break** să fie utilizată pentru scopuri speciale şi nu ca ieşire normală din ciclu. Instrucţiunea **break** este utilizată în cicluri în care o condiţie specială poate cauza oprirea imediată a ciclului. Spre exemplu, în programul următor, apăsarea unei taste poate opri execuţia programului:

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int i;
     char ch;
     clrscr();
     for(i=1;i<10000;i++)
       if(!(i%6))
       {   printf("%d  Mai doriti ? (Y/N): ",i);
           ch=getche();
           if(ch=='N') break;
           printf("\n");
       }
      getch();
}
```

5.11 Instrucţiunea switch

Permite realizarea structurii selective. Aceasta este o generalizare a structurii alternative şi a fost introdusă de Hoare. Ea poate fi realizată prin instrucţiuni **if** imbricate. Utilizarea instrucţiunii **switch** face ca programul să fie mai clar decât dacă se utilizează varianta cu instrucţiuni **if** imbricate. Structura *selectivă*, în forma în care a fost ea acceptată de către adepţii programării structurate, se realizează în limbajul C cu ajutorul următorui format al instrucţiunii **switch** :

```
switch(expresie)
{ case c1:
      sir_1
      break;
  case c2:
     sir_2
      break;
…….
  case cn:
     sir_n
     break;
  default:
     sir
}
```

unde :

c1,c2,…,cn sunt constante
sir_1,sir_2,…..sir_n,sir sunt succesiuni de instrucţiuni

Instrucţiunea **switch** cu formatul de mai sus se execută astfel :

- Se evaluează expresia dintre parantezele rotunde
- Se compară pe rând valoarea expresiei cu valorile constantelor **c1,c2,…,cn**. Dacă valoarea expresiei coincide cu una din constante, să zicem cu **ci**, atunci se execută secvenţa **sir_i**, apoi se trece la instrucţiunea următoare instrucţiunii **switch**, adică la instrucţiunea aflată după acolada închisă care termină instrucţiunea. Dacă valoarea respectivă nu coincide cu nici una din constantele **c1,c2,….cn**, atunci se execută succesiunea de instrucţiuni **sir** din **default** şi apoi se trece la instrucţiunea următoare instrucţiunii **switch**.

Menţionăm că este posibil să nu se ajungă la instrucţiunea următoare instrucţiunii **switch** în cazul în care succesiunea de instrucţiuni selectată pentru execuţie (**sir_i** sau **sir**) va defini ea însăşi un alt mod de continuare a execuţiei programului (de exemplu, execuţia instrucţiunii de revenire dintr-o funcţie, saltul la o instrucţiune etichetată etc.). Succesiunile **sir_1,sir_2,…..,sir_n** se numesc *alternativele* instrucţiunii **switch**. Alternativa **sir** este opţională, deci într-o instrucţiune **switch** secvenţa

default : sir poate fi absentă. În acest caz, dacă valoarea expresiei nu coincide cu valoarea nici uneia dintre constantele **c1,c2,.. ..cn**, atunci instrucţiunea **switch** nu are nici un efect şi se trece la execuţia instrucţiunii următoare. Instrucţiunea **switch** de mai sus este echivalentă cu următoarea instrucţiune **if** imbricată :

```
if(expresie==c1)
  sir_1
else
  if(expresie==c2)
     sir_2
  else
     if(expresie==c3)
        sir_3
     else
        if
          ……..
                    else
                      if(expresie==cn)
                           sir_n
                      else
                           sir;
```

Instrucţiunea **break** de la sfârşitul fiecărei alternative permite ca la întâlnirea ei să se treacă la execuţia instrucţiunii următoare instrucţiunii **switch**. Se obişnuieşte să se spună că instrucţiunea **break** permite ieşirea din instrucţiunea **switch**.

Instrucţiunea **break** poate fi utilizată numai în corpurile ciclurilor şi în alternativele instrucţiunii **switch**. Prezenţa ei la sfârşitul fiecărei alternative nu este obligatorie. În cazul în care instrucţiunea **break** este absentă la sfârşitul unei alternative, după execuţia succesiunii de instrucţiuni din compunerea alternativei respective se trece la execuţia succesiunii de instrucţiuni din alternativa următoare a aceleaşi instrucţiuni **switch**. Adică, dacă o instrucţiune **switch** are formatul :

```
switch(expresie)
{  case c1:
        sir_1
   case c2:
    sir_2
}
```

atunci ea este echivalentă cu următoarea secvenţă :

```
if(expresie==c1)
{  sir_1
    sir_2
}else
    if(expresie==c2)
       sir_2;
```

Exemplul 1: Următorul program recunoaşte numerele 1,2,3 şi 4 şi afişează numele cifrei introduse.

```
#include<stdio.h>
void main()
{
    int i;
    printf("Introduceti un intreg intre 1 si 4 : ");
    scanf("%d",&i);
    switch(i)
    {
        case 1 : printf("unu"); break;
        case 2 : printf("doi"); break;
        case 3 : printf("trei"); break;
        case 4 : printf("patru"); break;
        default: printf("numar necunoscut");
    }
}
```

Exemplul 2: Instrucţiunile **switch** sunt deseori folosite pentru a procesa comenzi meniu. De exemplu, programul următor:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float a,b;
    char ch;
    clrscr();
    printf("Doriti :\n");
    printf("Adunare,Scadere,Multiplicare,
            Impartire?\n");
    do{
          printf("Introduceti prima litera : ");
          ch=getche();
          printf("\n");
    }while(ch!='A' && ch!='S' && ch!='M' && ch!='I');
    printf("Introduceti primul numar : ");
    scanf("%f",&a);
    printf("Introduceti al doilea numar : ");
    scanf("%f",&b);
    switch(ch)
    {
          case 'A': printf("%f",a+b);break;
          case 'S': printf("%f",a-b);break;
          case 'M': printf("%f",a*b);break;
          case 'I': if(b) printf(%f",a/b);break;
    }
    getch();
}
```

Exemplul 3: Instrucţiunile asociate unui **case** pot să lipsească. Aceasta permite ca două sau mai multe **case** să execute aceleaşi instrucţiuni fără să fie nevoie de duplicarea lor. Iată un program care clasifică literele în vocale şi consoane:

```
#include<stdio.h>
#include<conio.h>
void main()
{    char ch;
     clrscr();
     printf("Introduceti o litera : ");
     ch=getche();
     switch(ch)
     {   case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':  printf("\n este o vocala");break;
         default :  printf("\n este o consoana");
   }
}
```

5.12 Instrucţiunea goto

Nu este o instrucţiune absolut necesară la scrierea programelor în limbajul C. Cu toate acestea, ea se dovedeşte utilă în anumite cazuri, spre exemplu la ieşirea din mai multe cicluri imbricate. Astfel de situaţii apar adesea la întâlnirea unei erori. În astfel de situaţii, de obicei, se doreşte să se facă un salt în afara ciclurilor în care a intervenit eroarea, pentru a se ajunge la o secvenţă externă lor de tratare a erorii respective. Înainte de a indica formatul instrucţiunii **goto** să precizăm noţiunea de *etichetă*.

Prin *etichetă* se înţelege un nume urmat de două puncte **nume:** unde **nume** este numele etichetei respective. După etichetă urmează o instrucţiune. Se obişnuieşte să se spună că eticheta *prefixează* instrucţiunea care urmează după ea. Etichetele sunt locale în corpul funcţiei în care sunt

definite. Instrucţiunea **goto** are formatul : **goto nume;** unde **nume** este o etichetă definită în corpul aceleaşi funcţii în care se află eticheta **goto**. La întâlnirea instrucţiunii **goto**, se realizează salt la instrucţiunea prefixată de eticheta al cărei nume se află după cuvântul cheie **goto**.

Deoarece o etichetă este locală în corpul unei funcţii, rezultă că ea este nedefinită în afara funcţiei respective. În felul acesta, o instrucţiune **goto** poate realiza un salt numai la o instrucţiune din corpul aceleaşi funcţii în care este utilizată. Deci, o instrucţiune **goto** nu poate face salt din corpul unei funcţii la o instrucţiune din corpul altei funcţii.

Nu se justifică utilizarea abuzivă a acestei instrucţiuni. Se recomandă a fi utilizată pentru a simplifica ieşirea din cicluri imbricate.

Exemplu : Presupunem că într-un punct al programului, aflat în interiorul mai multor cicluri, se depistează o eroare şi se doreşte să se continue execuţia programului cu o secvenţă de tratare a erorii respective. În acest caz vom folosi o instrucţiune **goto** ca mai jos.

```
for(…)
{……..
     while(….)
     { ……..
          do
          {…….
              for(….)
              {……….
                   if(k==0)
                         goto divzero;
                   else x=y/k;
                    ……
                }
                ……..
            }while(……);
            ……….
          }
……….
}
 ……
/* secvenţa de tratare a erorii */
divzero:
printf(……..);
……….
```

În absenţa instrucţiunii **goto** se poate realiza acelaşi lucru folosind un indicator şi o serie de teste realizate asupra lui.

5.13 Funcţiile standard sscanf şi sprintf

Biblioteca standard a limbajelor C şi C++ conţine funcţiile **sscanf** şi **sprintf** care sunt analoge funcţiilor **scanf** şi **printf**. Ele au un parametru în plus în apel şi anume primul lor parametru este adresa unei zone de memorie (şir de caractere) în care se pot păstra caractere ale codului ASCII. Ceilalţi parametri sunt identici cu cei întâlniţi în corespondentele lor, **printf** şi **scanf**. Primul parametru al acestor funcţii poate fi numele unui şir de caractere, deoarece un astfel de nume are ca valoare chiar adresa de început a zonei de memorie care îi este alocată.

Funcţia **sprintf** se foloseşte, ca şi funcţia **printf**, pentru a realiza conversii ale datelor de diferite tipuri din formatele lor interne, în formate externe reprezentate prin succesiuni de caractere. Diferenţa constă în aceea că, de data aceasta caracterele citite nu se afişează la terminal, ci se păstrează în şirul de caractere definit ca prim parametru al funcţiei **sprintf**. Ele se păstrează sub forma unui şir de caractere şi pot fi afişate ulterior din zona respectivă cu funcţia **puts**. De aceea un apel al funcţiei **printf** poate fi întotdeauna înlocuit cu un apel al funcţiei **sprintf**, urmat de un apel al funcţiei **puts**. O astfel de înlocuire este utilă când dorim să afişăm de mai multe ori aceleaşi date. În acest caz se apelează funcţia **sprintf** o singură dată pentru a face conversiile necesare din format intern în format extern, rezultatul conversiilor păstrându-se într-un şir de caractere. În continuare, se pot afişa datele respective apelând funcţia **puts** ori de câte ori este necesară afişarea lor. Funcţia **sprintf**, ca şi funcţia **printf** returnează numărul octeţilor şirului de caractere rezultat în urma conversiilor efectuate.

Exemplu :

```
int zi,luna,an;
char data_calend[11];
….
sprintf(data_calend, "%02d/%02d/%02d",zi,luna,an);
puts(data_calend);
…….
puts(data_calend);
……
```

Funcţia **sscanf** realizează, ca şi funcţia **scanf**, conversii din formatul extern în format intern. Deosebirea constă că de data aceasta caracterele nu sunt citite din zona tampon corespunzătoare tastaturii, ci ele provin dintr-un şir de caractere definit de primul parametru al funcţiei **sscanf**. Aceste caractere pot ajunge în şirul respectiv în urma apelului funcţiei **gets**. În felul acesta, apelul funcţiei **scanf** poate fi înlocuit prin apelul funcţiei **gets** urmat de apelul funcţiei **sscanf**. Astfel de înlocuiri sunt utile când dorim să eliminăm eventualele erori apărute la tastarea datelor.

Funcţia **sscanf**, ca şi funcţia **scanf**, returnează numărul câmpurilor convertite corect conform specificatorilor de format prezenţi în sintaxă. La întâlnirea unei erori, ambele funcţii îşi întrerup execuţia şi se revine din ele cu numărul de câmpuri tratate corect. Analizând valoarea returnată, se poate stabili dacă au fost prelucrate corect toate câmpurile sau a survenit eroare. În caz de eroare se poate reveni pentru a introduce corect datele respective. În acest scop este necesar să se elimine caracterele începând cu cel din poziţia eronată. În cazul în care se utilizează secvenţa :

```
gets(....);  sscanf(....);
```

abandonarea caracterelor respective se face automat reapelând funcţia **gets**. În cazul utilizării funcţiei **scanf** este necesar să se avanseze până la caracterul "linie nouă" aflat în zona tampon ataşată tastaturii sau să se videze zona respectivă prin funcţii speciale.

În exerciţiile următoare vom folosi secvenţele formate din apelurile funcţiei **gets** urmate de apelurile lui **sscanf**. O astfel de secvenţă se apelează repetat în cazul în care se întâlnesc erori în datele de intrare.

Exemplul :

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

Amintim că funcţia **gets** returnează valoarea **NULL** la întâlnirea sfârşitului de fişier. Funcţiile **sscanf** şi **sprintf** au prototipurile în fişierul **stdio.h**.

Exemplul 1: Să se scrie un program care citeşte un întreg pozitiv de tip **long**, stabileşte dacă acesta este prim şi afişează un mesaj corespunzător.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{  long n;
   long i;
   int j;
   char tab[255];
   clrscr();
   do
   {  printf("tastati 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;
       /* se va relua ciclul deoarece j este diferit de zero */
      }
      j=0; /* ciclul se întrerupe deoarece s-a citit corect un întreg pozitiv */
```

```
    }while(j);
    for(j=1,i=2;i*i<=n && j;i++)
      if(!(n%i)) j=0; /* numărul nu este prim */
    printf("numarul : %ld",n);
    if(!j) printf(" nu");
    printf(" este prim\n");
    getch();
}
```

Observaţii: Utilizarea instrucţiunii **continue** se poate omite folosind o instrucţiune **if** cu alternativa **else**:

```
if(sscanf(….)!=1 || n<=0)
{ ….
   j=1;
}
else j=0;
```

Ciclul **for** continuă atât timp cât expresia **i*i<=n&&j** este adevărată. Această expresie se evaluează de la stânga spre dreapta şi din această cauză **i*i<=n** se evaluează şi atunci când **j=0**. De aceea expresia respectivă este mai eficientă sub forma : **j&&i*i<=n**. În acest caz pentru **j=0** nu se mai evaluează restul expresiei.

Exemplul 2: Să se scrie un program care citeşte măsurile **a**,**b**,**c** ale laturilor unui triunghi, calculează şi afişează aria triunghiului respectiv folosind formula lui Heron.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
void main()
{  double a,b,c,p;
   char tab[255];
   clrscr();
   do /* citeşte măsurile laturilor triunghiului */
   {   do /* citeşte pe a */
       {  printf("a=");
          if(gets(tab)==NULL)
          {  printf("s-a tastat EOF\n");
             exit(1);
          }
          if(sscanf(tab,"%lf",&a)==1 && a>0) break;
          /* se iese din ciclul pentru citirea lui a */
          printf("nu s-a tastat un numar pozitiv\n");
          printf("se reia citirea lui a\n");
       }while(1);
       do /* citeşte pe b */
       {  printf("b=");
          if(gets(tab)==NULL)
          {  printf("s-a tastat EOF\n");
             exit(1);
          }
          if(sscanf(tab, "%lf", &b)==1 && b>0) break;
          /* se iese din ciclul pentru citirea lui b */
          printf("nu s-a tastat un numar pozitiv\n");
          printf("se reia citirea lui a\n");
       }while(1);
       do /* citeşte pe c */
       {  printf("c=");
          if(gets(tab)==NULL)
          {  printf("s-a tastat EOF\n");
             exit(1);
          }
          if(sscanf(tab,"%lf",&c)==1 && c>0) break;
```

```
            /* se iese din ciclul pentru citirea lui c */
             printf("nu s-a tastat un numar pozitiv\n");
             printf("se reia citirea lui a\n");
          }while(1);
          p=(a+b+c)/2;
          if(p-a>0 && p-b>0 && p-c>0) break;
          /* a,b,c pot fi laturile unui triunghi */
          printf("a=%g\tb=%g\tc=%g\t",a,b,c);
          printf("nu pot fi laturile unui triunghi\n");
       }while(1);
       printf("aria=%g\n",sqrt(p*(p-a)*(p-b)*(p-c)));
       getch();
    }
```

5.14 Header-ul ctype.h

Header-ul **ctype.h** este specializat pentru prelucrarea datelor de tip caracter. El conţine numai funcţii şi macrouri (secvenţe de cod asemănătoare funcţiilor, la apelul cărora se face substituţia numelui funcţiei cu codul asociat) de verificare şi prelucrare a caracterelor. Astfel, pentru clasificarea caracterelor, avem următoarele macrodefiniţii :

Macro de verificare	**Valoarea 1 când caracterul este :**
isalnum(c)	o literă sau cifră
isalpha(c)	o literă
isdigit(c)	o cifră în baza 10
iscntrl(c)	un caracter de control
isascii(c)	un caracter valid ASCII
isprint(c)	un caracter tipăribil
isgraph(c)	un caracter tipăribil mai puţin spaţiul
islower(c)	o literă mică
isupper(c)	o literă mare
ispunct(c)	un caracter de punctuaţie
isspace(c)	spaţiu,tab,CR,LF,tab vertical,form-feed
isxdigit(c)	o cifră în baza 16
Funcţii conversie caractere	**Face conversia unui caracter :**
int toupper(int ch)	în literă mare. Spre deosebire de macroul _**toupper** care modifică orice caracter, dacă caracterul **ch** nu este literă mică, funcţia îl întoarce nemodificat
int tolower(int ch)	în literă mică. Spre deosebire de macroul _**tolower** care modifică orice caracter, funcţia întoarce caracter nemodificat dacă nu este literă mare.

Exemplul: Transformarea literelor unui şir în litere mari.

```
#include<stdio.h>
#include<ctype.h>
void main()
{    int i;
     char t[255];
     scanf("%s",t);
     for(i=0;s[i];i++)
          s[i]=toupper(s[i]);
     printf("%s\n",t);
}
```

5.15 Funcţii matematice uzuale

Funcţiile matematice sunt definite în header-ul **math.h**. De obicei, marea lor majoritate sunt definite pentru valori reale de tip **double** putând fi convertite fără probleme în alte tipuri.

Sintaxa funcţiei	Valoarea returnată
int abs(int x);	Macrouri care întorc modulul unui număr întreg de format normal, întreg de format lung şi real de tip double.
long int labs(long int x);	
double fabs(double x);	
double sqrt(double x);	Calculează rădăcina pătrată
double pow(double x, double y);	Funcţia putere $\mathbf{x^y}$. În cazul în care **x** este 0 şi **y** este negativ sau dacă **x** este negativ şi **y** nu este întreg se semnalează eroare.
double pow10(int p);	Funcţia putere când baza este 10.
double exp(double x);	Funcţia $\mathbf{e^x}$.
double log(double x);	Funcţia logaritm natural, **ln(x)** .
double log10(double x);	Logaritmul în baza 10.
double ldexp(double x, int exp);	Calculează $\mathbf{x*2^{exp}}$.
double fmod(double x, double y);	Calculează **x modulo y** .
double poly(double x, int n, double coef[]);	Evaluează o funcţie polinomială, unde : **x** – valoarea argumentului funcţiei , **n** – gradul funcţiei polinomiale, **coef** – tabloul de coeficienţi ai funcţiei polinomiale, **coef[0]** este termenul liber şi **coef[n]** este termenul de rang maxim
double floor(double x);	Rotunjire inferioară. Întoarce cel mai mare număr întreg mai mic sau egal cu **x**.
double ceil(double x);	Rotunjire superioară. Întoarce cel mai mic întreg mai mare sau egal cu **x**.

Exemplu: Calculul valorii unui polinom.

```
#include<stdio.h>
#include<math.h>
/* polinomul: x^5-2x^3-6x^2+15x-1 */
void main()
{    double a[]={-1.0,15,-6.0,-2.0,0,1.0};
     /* coeficienţii polinomului in ordinea crescătoare a puterilor */
     double x,rez;
     printf("x="); scanf("%lf",&x);
     rez=poly(x,5,a);
     printf(" val. polinomului pentru x=%lg este %lg\n",x,rez);
}
```

Funcţiile trigonometrice au argumente de tip **real** care trebuie specificate în **radiani**. Cele mai utilizate funcţii implementate sunt :

Sintaxa funcţiei	Numele funcţiei	Valoarea returnată
double sin(double x);	Sinus	Reală între –1 şi 1
double cos(double x);	Cosinus	Reală între –1 şi 1
double tan(double x);	Tangentă	Reală
double asin(double x);	Arc sinus	Reală între $-\pi/2$ şi $\pi/2$
double acos(double x);	Arc cosinus	Reală între 0 şi π
double atan9double x);	Arc tangentă	Reală între $-\pi/2$ şi $\pi/2$
double atan2(double y, double x);	Arc tangenta lui **y/x**	Reală între 0 şi π
double sinh(double x);	Sinusul hiperbolic	Reală
double cosh(double x);	Cosinusul hiperbolic	Reală
double tanh(double x);	Tangenta hiperbolică	reală

5.16 Exerciţii şi teste grilă

1. Care dintre următoarele secvenţe de instrucţiuni atribuie variabilei reale **x** cea mai mare dintre valorile variabilelor reale **a** şi **b** sau valoarea lor comună, în cazul în care acestea sunt egale ?

```
a) if(a<=b) x=b; else x=a;
b) if(a<=b) x=a; else x=b;
c) if(a==b) x=a; else if(b>a) x=b;
d) x=a; if(x<b) x=b;
e) nici una dintre secvenţele
   anterioare
```

2. Fie variabilele **a** şi **b**, ambele de tipul **int**, ale căror valori se presupun cunoscute. Scrieţi o secvenţă de programul pentru enunţul : „*dacă numerele x şi z sunt ambele impare, atunci tipăreşte valoarea 1*”.

```
a) if((x%2!=0)&&(y%2!=0))
                  puchar('1');
b) if(x%2==0||y%2==0) putchar('1');
c) if(x%2 && y%2) putchar('1');
d) if(!(x%2==0||y%2==0))
                  putchar('1');
e) if(!(x%2==0)&&!(y%2==0))
                  putchar('1');
```

3. Ce va afişa programul următor, dacă de la tastatură se introduc în ordine numerele 5, 7 şi 8?

```
#include<stdio.h>
void main()
{
      int x,y,z,m;
      scanf("%d %d %d",&x,&y,&z);
      m=(x+y+z)/3;
      switch(m) {
         case 1,2,3,4:
         {   printf("Corigent");
             break; }
         case 5,6:
         {   printf("Mediocru");
             break; }
         case 7,8,9:
         {   printf("Bine");
             break; }
         case 10:
         {   printf("Foarte bine");
             break; }
        default:
             printf("Eroare");
      }
}
```

a) Corigent b) Mediocru
c) Satisfăcător d) Foarte bine
e) Eroare

4. Precizaţi ce se va afişa în urma execuţiei secvenţei de program de mai jos pentru **n=5** (**s**,**n** şi **k** sunt variabile întregi).

```
s=0; k=1;
while(k<=n)
{
      s+=k;
      k+=2;
}
printf(„s=%d", s);
```

a) s=4 b) s=16 c) s=9
d) s=15 e)s=0

5. Care dintre secvenţele de program de mai jos calculează corect factorialul numărului natural **n** ?

```
1) p=1; for(i=1; i<=n; i++) p=p*i;
2) p=1; i=1; while(i<=n) p=p*i++;
3) p=1; i=1;
   do{ p*=i; i=i+1; }while(i<=n);
```

a) numai 1) b) numai 2)
c) numai 3) d) 1) şi 3)
e) toate

6. Care trebuie să fie valoarea variabilei întregi **m**, astfel încât următoarea secvenţă de program să afişeze exact un caracter ‘A’ ?

```
x=5;
do{
      putchar('A');
      x++;
}while(x>m);
```

a) 12 b)5 c)6
d) 4 e)1

7. Se consideră secvenţa de program de mai jos, în care toate variabilele sunt întregi. Pentru **n=3**, care va fi valoarea variabilei **p** după execuţia secvenţei ?

```
p=1;
for(i=1; i<=n; i++)
{     s=0;
       for(j=1; j<=i; j++) s+=j;
       p*=s;
}
```

a) 180 b) 18 c) 9
d) 216 e) 1

8. Precizaţi ce se va afişa în urma execuţiei programului următor pentru **x=179** ?

```
#include<stdio.h>
void main()
{
      int c,s;  long d,x;
      scanf(„%ld", &x);
      d=x; s=0;
      while(d)
```

```
      {    c=d%10; s+=c; d=d/10; }
      printf(„%d", s);
}
```
a) 16 b) 18 c)17
d) 0 e) 971

9. Considerăm programul următor :
```
#include<stdio.h>
void main()
{
     short int m,x;
     m=-1;
     while((scanf("%d",&x)==1)&& x)
         if(x>m) m=x;
     printf("%d", m);
}
```
Precizaţi ce valoare va afişa programul, dacă şirul de numere citit de la tastatură este **2, 5, -32000, 33000, 0.**
a) −1 b) 0 c) 33000
d) 2 e) 5

10. Pentru ce valoare a variabilei **m**, secvenţa de program de mai jos reprezintă o buclă infinită ?
```
int n=10, m;
do{
      while(n>0) n--;
}while(n!=m);
```
a) 10
b) orice valoare diferită de 10
c) 0
d) orice valoare diferită de 0
e) orice valoare întreagă

11. Ce valoare va afişa programul următor pentru **n=12** ?
```
#include<stdio.h>
void main()
{
      int i,n,s;
      scanf(„%d", &n);
      for(s=0,i=2; i<n/2;
            !(n%i)?s+=i++: i++);
      printf("%d",s);
}
```
a) 0 b) 9 c) 12 d) 78
e) programul conţine erori

12. Dacă de la tastatură se introduc, în ordine, numerele **2,7,3,8,5,5**, ce valoare va afişa secvenţa următoare ?
```
int a, b, nr=0;
do{
      scanf("%d %d", &a, &b);
}while((b!=a) ? ++nr : 0);
printf("%d", nr);
```
a) 0 b) 1 c) 2
d) 3 e) 4

13.
```
int i=0; int j=6;
   if(i!=0) && (j/i!=1)
   j=i; i+=4; j+=i;
```
Pentru codul de mai sus alegeţi comportamentul corect :
a) va genera eroare la rulare
b) j=4 c) j=0 d) j=10

14.
```
A) for(exp1;exp2;exp3)
        instructiune;
```
este echivalent cu
```
      exp1;
      while(exp2)
      {  instructiune;
         exp3;}
```
```
B) for( ;exp; ) instructiune;
```
este echivalent cu
```
      while(exp) instructiune;
```
```
C) for( ; ; ) instructiune;
```
este echivalent cu
```
      while(1) instructiune;
```
Care din echivalenţele de mai sus sunt eronate:
a) nici una b) A,B
c) B,C d) A,C

15. Fie secvenţa :
```
do{
        scanf("%c",&c);
        if (c>='a' && c<='z') i++;
} while(c!=EOF);
```
Care din următoarele afirmaţii este adevărată :
a) se numără câte caractere litere mici sunt citite
b) se numără câte caractere sunt citite
c) se numără câte caractere litere mari sunt citite
d) nici una

16. Se dă o secvenţă de program în care toate variabilele sunt de tip întreg. În urma execuţiei programului ce conţine această secvenţă, ce valori capătă variabilele **d** şi **s** ?
```
a=8;
b=c=1; d=s=0;
i=3;
do{
   i++;
   if(a>0)
     if(b>1)
        if(c>1) d=a;
     else d=a+b;
   else d=a+b+c;
   s+=i+d;
} while(i>5);
```
a) d=8 s=12 b) d=9 s=12
c) d=10 s=13 d) d=10 s=14

17. Se consideră secvenţa de program :

```
void main(void)
{  int x=1;
    float z, y=0.96;
    x+=y; z=sqrt(x);
    printf("%f",z);
}
```
Valoarea afişată este :
a) 1.46 b) 1.000000 c) 1
d) programul conţine erori de sintaxă

18. Fie secvenţa de cod prezentată mai jos :
```
i=1;
while(n)   i=i*(n--);
```
atunci aceasta :
a) calculează **n!**
b) calculează $\mathbf{i^n}$
c) calculează $\mathbf{n^i}$
d) se ciclează la infinit

19. Fie secvenţa de cod prezentată mai jos :
```
i=1;
while(n--)  i=i*2;
```
atunci aceasta :
a) calculează $\mathbf{2^n}$ b) calculează $\mathbf{i^2}$
c) calculează $\mathbf{n^2}$
d) se ciclează la infinit

20. Scrieţi o buclă care afişează secvenţa :
```
1
22
333
4444
55555
a) for(loop==1;loop<=5;looop++)
   {
for(loop1==1;loop1<=loop;loop1++)
           printf("%d",loop1);
          printf("\n");
   }
b) for(loop=1;loop<=5;loop++)
   {
   for(loop1=1;loop1<=loop;loop1++)
            printf("%d",loop);
      printf("\n");
   }
c) for(loop=1;loop<=5;loop++)
   {
   for(loop1=1;loop1<=loop;loop1++)
            printf("%d",loop1);
      printf("\n");
   }
d) for(loop=5;loop>0;loop--)
   {
   for(loop1=1;loop1<=loop;loop1++)
           printf("%d",loop1);
      printf("\n");
   }
```

21. Referitor la secvenţa de cod de mai jos, ce valoare va avea variabila **contor** după execuţia ei ?
```
int x=3, contor=0;
while((x-1)) {
   ++ contor;
    x--;
}
```
a) 0 b) 1 c) 2 d) 3

22. Ce realizează următoarea secvenţă :
```
scanf("%d",&x,&y,&z);
if(x<=y);
    x=x+z;
    y=y+z;
else z=x+y;
```
a) citeşte trei numere şi calculează suma lor
b) citeşte trei numere şi calculează produsul lor
c) este greşită
d) **z** devine minimul dintre **x** si **y**

23. Care secvenţă de program realizează o repetiţie la infinit :
```
I)   do while(1);
II)  do while(0);
III) do while(i%1<2);
```
a) doar I b) doar II
c) doar III d) doar I şi III

24. Se dă codul :
```
int x=4, a=2;
int b=4, c=8;
if(x==b) x=a;
           else x=b;
if(x!=b) c=c+b;
           else c=c+a;
printf("c=%d\n",c);
```
Ce se va afişa după execuţia codului de mai sus ?
a) c=4 b) c=8 c) c=10 d) c=12

25. Se dă următoarea secvenţă de cod :
```
int i, j=0;
for(i=1;i<11;i+=2)
{  j++; if(i==7) break; }
```
Care va fi valoarea finală a lui **j** ?
a) 3 b) 4 c) 5 d) 7

26. Se dă următoarea secvenţă de cod :
```
int i, j, k;  i=1; j=1; k=2;
while(i<6)
{       k=k+i;i++;
        j=j+k;
        if(k==3) j--;
        else if(j==8) break;
}
printf("%d--%d--%d",i,j,k);
```
Ce va afişa codul de mai sus ?

a) 1—1—2 b) 2—3—3
c) 3—8—5 d) 4—16—8

27. Se dă codul :
```
int a=3,  b=0;
while(a)
{       b=b++; a=b; }
```
Ce valoare va avea **b** ?
a) 1 b) 3 c) 0
d) nedefinită pentru că va fi o buclă infinită

28. Fie secvenţa de cod :
```
#include<stdio.h>
void main(void)
{  int i,y,x=6;
   y=x;
    while(1)
    {   y--; x=x*y;
        if(y==0) break;
    }
    printf("%d",x);
}
```
Ce număr va fi afişat pe ecran la terminarea execuţiei acestui cod ?
a) 720
b) programul va rula la infinit
c) 0 d) 6

29. Fie secvenţa de cod :
```
int x;
int y=1;
for(x=0;x<=30;x++)
{     y=y+1;
      if(x<5) continue;
      if(x>5) break;
      y=y+x;
}
```
După execuţia codului anterior, ce valoare va avea **y** ?
a) 486 b) 31 c) 13 d) 496

30. Fie următorul program :
```
int main()
{   int i;
     for(i=1;i<65535;i++);
      printf("i=%d\n",i);
}
```
a) valoarea afişată este i=65535
b) valoarea afişată este i=65534
c) valoarea afişată este i=32766
d) programul nu este corect deoarece este depăşit domeniul de valori

31. În urma execuţiei secvenţei de program de mai jos, pentru care dintre tripletele de valori ale variabilelor **a, b, c**, date mai jos, se va afişa valoarea 1?
```
x=1;
if(!(a<=b)||!(a<=c))
{  if(b>=c) printf("%d\n",-x); }
else
   if(b<c) printf("%d\n",x);
```
a) a=3, b=2, c=4 b) a=2, b=3, c=4
c) a=4, b=3, c=2 d) a=2, b=4, c=3
e) a=4, b=2, c=3

32. Se consideră programul următor:
```
#include<stdio.h>
void main()
{
      int a,b,c,d,i;
      scanf("%d %d", &a, &b);
      if(a>b)
      { c=a; a=b; b=c; }
      d=0;
      for(i=a;i<=b;i++)
        if(i%2==0) d++;
      printf("%d",d);
}
```
Ce valoare se afişează pentru **a=33** şi **b=18** ?
a) 8 b) 7 c) 0 d) 16 e) 33

33. Fie următorul program:
```
#include<stdio.h>
void main()
{
   int x,y,m,n,a,b;
   a=b=2;                          //(1)
   m=(x=a+3,y=b-1,y++,y+x);  //(2)
   if(a&&x>y) printf("%d",m);//(3)
   if(x-y>a&&x>y||!m)        //(4)
         putchar('1');
   else putchar('0');
   if((n=x>y)==0)||(--x==4)) //(5)
      printf("%d",x--);
}
```
În timpul execuţiei programului se pot spune următoarele:
a) atribuirea din linia (1) este eronată
b) instrucţiunea din linia (2) este eronată
c) în urma execuţiei liniei (3) nu se afişează nici o valoare
d) în urma execuţiei liniei (4) se afişează valoarea 1
e) în urma execuţiei liniei (5) se afişează valoarea 4

34. Dacă de la tastatură se introduce numărul 22, câte valori distincte va afişa programul următor?
```
#include<stdio.h>
#include<math.h>
void main()
{
   int x,n,i;
 for(scanf("%d",&n),i=1;;x=sqrt(i),
          printf("%d",x),i++)
```

```
   if(i>n) break;
}
```
a) nici una
b) 1 c) 2 d) 3 e) 4

35.Precizaţi de câte ori se va afişa valoarea 1 în timpul execuţiei programului următor, dacă **a=3,b=4** şi **x=5**.

```
#include<stdio.h>
void main()
{
      int a,b,x;
      scanf("%d%d%d",7a,&b,&x);
      if(!((x<=a)&&(x>=b)))
            putchar('1');
      if(!(x<=a||x>=b))
            putchar('1');
      if(!(x<=a)&&!(x>=b))
            putchar('1');
      if(!(x<=a)||!(x>=b))
            putchar('1');
}
```
a) nici o dată b) o dată
c) de două ori d) de trei ori
e) de patru ori

36. Dacă în timpul execuţiei programului de mai jos **n** va primi valoarea **232213**, care vor fi în final valorile variabilelor **f1**, **f2** şi **f3** ?

```
#include<stdio.h>
void main()
{
      long n;
      unsigned int f1,f2,f3,c;
      scanf("%ld",&n);
      f1=f2=f3=0;
      do{
         c=n%10; n=n/10;
         switch(c) {
           case 1: { f1++; break; }
           case 2: { f2++; break; }
           case 3: { f3++; break; }
          }
      }while(n!=0);
      printf("%u %u %u",f1,f2,f3);
}
```
a) f1=1,f2=1,f3=1 b) f1=1,f2=2,f3=2
c) f1=1,f2=2,f3=3 d) f1=2,f2=1,f3=3
e) f1=3,f2=2,f3=1

37.Pentru **n=7**, care dintre secvenţele de program de mai jos trebuie executată astfel încât, la finele execuţiei, valoarea variabilei **p** să fie 48 ?

```
a) p=1; i=2;
   while(i<=n) { p*=i;i+=2;}
b) p=1; i=1;
   while(i<n/2){i++;p=p*(2*i+1); }
c) p=1; i=1;
   while(i<=n/2) {p=p*(2*i);i++; }
d) p=1; i=0;
   while(i<n) { i+=2;p*=i; }
e) nici una dintre secvenţele
   anterioare
```

38. Precizaţi care dintre următoarele secvenţe de instrucţiuni atribuie variabilei întregi **x** valoarea $\mathbf{n^2}$, cu **n** număr natural.

```
a) x=1;
   for(j=1;j<3;j++) x*=n;
b) x=1;
   for(j=1;j<=n;j++) x*=2;
c) x=1; j=0;
   while(j<2) x*=n; j++;
d) x=1; j=0;
   do{ j++; x*=n;}while(j<2);
e) x=n*n;
```

39. Precizaţi care dintre următoarele secvenţe de instrucţiuni atribuie variabilei întregi **x** valoarea $\mathbf{10^n}$, cu **n** număr natural.

```
a) x=10;
   for(j=1;j<=n;j++)x*=i;
b) x=1;
   for(j=n;j>0;j--) x*=10;
c) x=1; j=1;
   do{ x*=10; j++; }while(j<n);
d) x=1; j=0;
   while(j<=n) { j++; x*=i;}
e) nici una dintre variantele
   anterioare
```

40. Deduceţi ce valoare se va afişa în urma execuţiei secvenţei de program de mai jos, dacă valorile variabilei **x** citite de la tastatură sunt în ordine **3,2,4,3,5,10,20,0**.

```
scanf("%d",&x);
nr=0;
do{
      y=n;
      scanf("%d",&x);
      if(x==2*y) nr++;
}while(x!=0);
printf("%d",nr);
```
a) 0 b) 1 c) 2 d) 3 e) 4

41. Care dintre următoarele secvenţe de înstrucţiuni atribuie variabilei întregi **u** valoarea primei cifre a numărului natural reprezentat de variabila **x** ?

```
a) u=x;
   while(u>=10) u=u%10;
b) while(x>=10) x=x/10;
   u=x;
c) u=x/10;
d) u=x%10;
e) nici una din variantele
   anterioare
```

42. Care dintre următoarele secvenţe de instrucţiuni atribuie variabilei întregi **u** valoarea ultimei cifre a numărului natural reprezentat de variabila **x** ?

```
a) while(x>=10) x=x/10; u=x;
b) u=x; while(u>=10) u=u%10;
c) u=x%10;
d) u=x/10;
e) toate variantele anterioare
```

43. Fie secvenţa de program următoare, în care **ok** este o variabilă de tipul int, iar **x** este un număr natural.

```
ok=0;
for(j=2;j<x;j++)
   if(x%j==0) ok=1;
printf("%d",ok);
```

Secvenţa afişează 1 dacă:

a) numărul **x** are cel puţin un divizor propriu
b) numărul **x** nu are nici-un divizor propriu
c) toate numerele naturale mai mici ca **n**, fără 0 şi 1, sunt divizori proprii ai lui **x**
d) numărul **x** are cel mult un divizor propriu
e) nici una dintre variantele de mai sus

44. Se consideră secvenţele de program de mai jos. Pentru **n=4**, precizaţi care dintre secvenţe afişează, în urma execuţiei, şirul de numere: **1,2,2,3,3,3,4,4,4,4** .

```
a) for(j=1;j<=n;j++)
     for(k=1;k<=n;k++)
        printf("%2d",j);
b) for(j=1;j<=n;j++)
     for(k=1;k<=j;k++)
        printf("%2d",j);
c) for(j=1;j<=n;j++)
     for(k=1;k<=n;k++)
        printf("%2d",k);
d) for(j=1;j<=n;j++)
     for(k=1;k<=j;k++)
        printf("%2d",k);
e) for(k=1;k<=n; k++)
     for(j=1;j<=n;j++)
        printf("%2d",j);
```

45. Fie secvenţa de program următoare:

```
s=0;
for(j=3;j<=n;j+=3) s+=j;
```

Se dau mai jos cinci triplete de numere, fiecare astfel de triplet reprezentând un set de valori pentru variabila de intrare **n**. Care dintre aceste triplete au proprietatea că pentru toate cele trei valori ale lui **n** din triplet se obţine aceeaşi valoare a lui **s** ?

a) (3,5,6) b) (6,7,8)
c) (10,11,12) d) (6,9,12)
e) (15,16,17)

46. Considerând că toate variabilele sunt întregi, ce valoare se afişează după execuţia secvenţei de mai jos ?

```
s=0;t=0;x=3;i=1;y=1;z=1;
do{
      if(x>0)
        if(y>1)
          if(z>2) t=x;
        else t=x+y;
      else t=x+y+z;
      s+=i+t; i++;
}while(i>7);
```

a) 1 b) 5 c) 6 d) 51 e) 63

47. Care dintre şirurile de valori date în variantele de răspuns trebuie introduse de la tastatură în timpul execuţiei programului următor, astfel încât să se declanşeze un ciclu infinit ?

```
#include<stdio.h>
void main()
{
  int x,y;
while(scanf("%d",&x)==1 &&
       scanf("%d",&y)==1 && (x||y))
   do{
      y--;
      printf("*%d *%d",x,y);
   }while(x!=y);
}
```

a) 2,7,3,8,0,0 b) 2,5,4,4,0,0
c) 1,3,6,2,0,0 d) 2,4,5,8,0,0
e) 0,0

48. Pentru programul următor, care dintre afirmaţiile de mai jos sunt adevărate ?

```
#include<stdio.h>
void main()
{
   int s,x;
   for(s=0,x=1;0;s+=x,scanf("%d",x)
       if(!x) break;
   printf("%d",s);
}
```

a) dacă de la tastatură se introduc, în ordine, numerele 2,3,4 şi 5, atunci programul va afişa suma numerelor citite, adică 14
b) dacă prima valoare introdusă de la tastatură este 0, atunci ciclul se încheie şi se afişează valoarea 1
c) ciclul este eronat: nu se poate face o citire în linia for
d) instrucţiunea if este eronată
e) din cauză că lipseşte expresia care dă condiţia de continuare,

ciclul for se va executa la infinit

49. Care dintre secvenţele de mai jos afişează corect şirul cifrelor impare **97531** în această ordine ?

a)
```
for(j=9;j>=1;j--)
     printf("%d",j--);
```
b)
```
for(j=0;j<=9;j++)
     printf("%d",9-j++);
```
c)
```
for(j=9;j-->=1;)
     printf("%d%d",j,j--);
```
d)
```
j=10;
while(j--) printf("%d",--j);
```
e)
```
j=1;
do{
     printf("%d",10-j++);
}while(j<=9?j++:0);
```

Cap.6 Tablouri

6.1 Declararea tablourilor

Un **tablou** reprezintă un tip structurat de date care ocupă o zonă de memorie continuă,cu elemente componente de acelaşi tip. În cadrul tabloului un element este în mod unic identificat prin poziţia ocupată în cadrul structurii. Această poziţie este definită prin unul sau mai mulţi **indici** sau **indecşi**, din acest motiv tablourile numindu-se **variabile indexate**.

Declararea unui tabou se face cu sintaxa :

tip nume[dim_1][dim_2]……..[dim_n];

unde :

- **tip** este tipul elementelor componente ale tabloului. Acesta poate fi un tip predefinit sau definit de utilizator
- **nume** este numele variabilei tablou
- **dim_1, dim_2,…..,dim_n** sunt numere întregi pozitive care exprimă dimensiunile tabloului

Pentru a utiliza un element din tablou se foloseşte sintaxa :

nume[index_1][index_2]……[index_n]

fiecare index respectând condiţia **index_i** $\in${**0,……,dim_i-1**}.

Un tablou unidimensional se numeşte **vector**, iar un tablou bidimensional se numeşte **matrice** .

Exemple:

```
char s[100];
int x[25];
long double a[10][15];
```

Observaţii:

- primul element dintr-un vector va avea indexul **0**, iar ultimul element stocat va avea indexul **dim-1**
- primul element dintr-o matrice va avea indexul **(0,0)**, iar ultimul va avea indexul **(dim_1-1,dim_2-1)**
- dimensiunile tabloului trebuie să fie expresii constante
- compilatorul nu face verificări pentru depăşirea dimensiunii tabloului
- pentru alocarea unui tablou sunt necesari **nr_elemente*sizeof(tip)** octeţi, unde **tip** este tipul de bază al tabloului
- atribuirea tablourilor nu poate fi făcută direct

Exemplu **:**

```
int x[10],y[10];
x=y;          /*operaţie ilegală*/
```

6.2 Iniţializarea tablourilor

Declaraţia unui tablou poate fi urmată de o secvenţă de iniţializare formată din una sau mai multe perechi de acolade între care se pun valori ale tipului de bază, separate prin virgulă.

Pentru tablourile unidimensionale este permisă omiterea numărului de elemente. El va fi egal cu numărul de valori folosite la iniţializare.

În cazul în care la iniţializarea tablourilor de tip numeric sunt folosite mai puţine elemente decât dimensiunea tabloului, restul elementelor sunt iniţializate cu 0.

Exemple**:**

```
float x[5]={1.2,3.4e3,-2,90.7E-8,-88.5e-7};
char s[100]="test tablou";
short y[]={0,1,2,3,4};              /* y are 5 elemente */
char s[]={'t','e','s','t','\0'};    /* char s[]="test"; */
int x[5]={1,2};                     /* x=(1, 2, 0, 0, 0) */
```

Iniţializarea unui tablou multidimensional se face asemănător cu cea a unui vector. Se poate face specificarea completă sau parţială a datelor, cele nedefinite fiind iniţializate cu 0. Este permisă doar omiterea primei dimensiuni (cea din stânga).

Exemple**:**

```
int a[3][2]={{1,2},{3,4},{5,6}};
```

echivalent cu :

```
int a[][2]={{1,2},{3,4},{5,6}};
```

sau chiar cu:

```
int a[3][2]={1,2,3,4,5,6};
```

iniţializarea

```
int m[2][3]={{1},{2}};
```

care este echivalentă cu :

```
int m[2][3]={{1,0,0} , {2,0,0}};
```

6.3 Prelucrări elementare ale vectorilor

Deoarece limbajul C nu verifică depăşirea dimensiunilor maxime declarate pentru tablourile utilizate în aplicaţii, pentru a evita scrierea accidentală a unor zone de memorie, programatorul trebuie să asigure validarea dimensiunilor reale (implicit a numărului de elemente) citite de la intrare. Uzual, un vector se declară în următoarea manieră:

```
#define MAX 100 /* dimensiunea maximă admisă */
……….
int a[MAX];
int n;       /* dimensiunea reală citită la intrare */
```

Secvenţa tipică de validare a dimensiunii unui vector este următoarea:

```
do{
      printf("n=");
      scanf("%d",&n);
      if(n<=0||n>MAX) printf("dimensiune incorecta\n");
}while(n<=0||n>MAX);
```

6.3.1 Citirea elementelor unui vector

După validarea dimensiunii, citirea elementelor unui vector se face în ordinea crescătoare a indicilor, uzual cu o instrucţiune for :

```
for(int i=0; i<n; i++)
{    printf("a[%d]=",i);
     scanf("%d",&a[i]);
}
```

Observaţie: Deşi tablourile sunt indexate în C începând de la 0, se pot utiliza elementele numai de la indexul 1 (ca în Pascal), elementul **a[0]** rămânând liber. În această situaţie secvenţa de citire (şi toate celelalte secvenţe de prelucrare) se vor modifica corespunzător, conform modelului de mai jos:

```
for(int i=1;i<=n;i++)
{   printf("a[%d]=",i);
    scanf("%d",&a[i]);
}
```

6.3.2 Determinarea elementului minim/maxim

Metoda tipică de determinare a elementului minim/maxim dintr-un vector este următoarea : se iniţializează minimul/maximul cu primul element din vector apoi se compară cu celelalte elemente din vector reţinându-se, pe rând, valorile mai mici/mai mari.

```
minim=a[0];          /* maxim=a[0]; */
for(i=1; i<n; i++)
  if(minim>a[i])     /*  (maxim<a[i])  */
    minim=a[i];      /* maxim=a[i]; */
```

6.3.3 Determinarea primului element cu o anumită proprietate

Pentru a determina primul element (de indice minim) cu o anumită proprietate, se parcurge vectorul de la stânga la dreapta până când găsim primul element cu proprietatea cerută sau până când

epuizăm elementele vectorului. De exemplu, determinarea primului element nul dintr-un vector se realizează cu secvenţa:

```
f=-1;
for(j=0;j<n;j++)
   if(!a[j])
   {    f=j; break; }
```

Verificând valoarea variabilei **f** decidem dacă în vectorul există cel puţin un element cu proprietatea cerută (**f**=indicele acestuia) sau nici unul (**f =-1**).

6.3.4 Determinarea ultimului element cu o anumită proprietate

Pentru a determina ultimul element (de indice maxim) cu o anumită proprietate, se parcurge vectorul de la dreapta spre stânga (în ordinea descrescătoare a indicilor) până când găsim primul element cu proprietatea cerută sau până când epuizăm elementele vectorului. De exemplu, determinarea ultimului element par dintr-un vector se realizează cu secvenţa:

```
f=-1;
for(j=n-1;j>=0;j--)
   if(!(a[j]%2))
   {    f=j; break; }
```

6.3.5 Eliminarea tuturor elementelor cu o anumită proprietate

Cea mai simplă metodă de a elimina dintr-un vector toate elementele cu o anumită proprietate este să creăm un nou vector în care se păstrează elementele care nu au proprietatea respectivă. De exemplu, pentru a elimina dintr-un vector toate elementele negative, putem utiliza secvenţa:

```
j=-1;
for(i=0;i<n;i++)
 if(a[i]>=0)          /* nu are proprietatea cerută */
   b[++j]=a[i];       /* păstram elementul în vectorul b */
n=j;                  /* actualizăm dimensiunea vectorului */
```

Metoda este ineficientă datorită consumului de memorie necesară pentru vectorul **b**. O metodă mult mai eficientă este să folosim acelaşi vector în care vom „îngrămădi" pe primele poziţii elementele care trebuie păstrate. Prin actualizarea dimensiunii vectorului, elementele de prisos nu vor mai fi luate în consideraţie în prelucrările ulterioare. Secvenţa care realizează această operaţie este următoarea:

```
j=-1;
for(i=0;i<n;i++)
   if(a[i]>=0)      /* nu are proprietatea cerută */
   a[++j]=a[i];     /* mutăm elementul la începutul vectorului */
n=j;                /* actualizăm dimensiunea vectorului */
```

6.3.6 Eliminarea elementului din poziţia k dată (1<=k<=n)

Prin eliminarea elementului din poziţia **k** dată (elementul de indice **k-1**), se observă că primele **k-1** elemente rămân neschimbate, în timp ce elementele din poziţiile **k+1**, **k+2**,……,**n** se deplasează cu o poziţie spre stânga pentru a "umple" golul rămas prin eliminarea elementului din poziţia **k**. Evident, dimensiunea vectorului scade cu o unitate :

```
for(j=k-1;j<=n-2;j++) a[j]=a[j+1];  /* deplasăm elementele spre stânga */
n--;      /* corectăm dimensiunea */
```

6.3.7 Inserarea unui element y în poziţia k dată (1<=k<=n)

Cum inserarea unui element se face fără a pierde vreun element din vectorul iniţial, elementele din poziţiile **k**, **k+1**,.....**n** trebuie să se deplaseze cu o poziţie spre dreapta pentru a face loc noii valori **y** introdusă în poziţia **k** (indice **k-1**). Dimensiunea vectorului creşte cu o unitate:

```
for(j=n;j>=k;j--) a[j]=a[j-1];      /* deplasăm elementele spre dreapta */
a[k-1]=y;   /* inserăm elementul y */
n++;        /* actualizăm dimensiunea */
```

6.3.8 Permutarea circulară cu o poziţie spre stânga

Prin acestă operaţie, elementele din poziţiile **2,3,……..,n** se deplasează cu o poziţie spre stânga şi elementul din prima poziţie ajunge în poziţia **n**. Vectorul nu îşi modifică dimensiunea:

```
aux=a[0];      /*salvăm temporar primul element */
for(j=0;j<=n-2;j++) a[j]=a[j+1];      /* deplasăm elementele spre stânga */
a[n-1]=aux;   /* mutăm elementul în ultima poziţie */
```

6.3.9 Permutarea circulară cu o poziţie spre dreapta

Prin această operaţie, elementele din poziţiile **1,2,……,n-1** se deplasează cu o poziţie spre dreapta, iar elementul din poziţia **n** ajunge în poziţia **1**. Vectorul nu îşi modifică dimensiunea:

```
aux=a[n-1];      /* salvăm temporar ultimul element */
for(j=n-1;j>=1;j--) a[j]=a[j-1];     /*deplasăm elementele spre dreapta */
a[0]=aux;        /* mutăm elementul în prima poziţie */
```

6.3.10 Sortarea vectorilor

Prin sortare se înţelege aranjarea elementelor unui vector în ordine crescătoare sau descrescătoare. Pentru rezolvarea acestei probleme au fost concepuţi diverşi algoritmi, mai mult sau mai puţin rapizi, mai simpli sau extrem de complicaţi. În acest moment vom aborda două dintre cele mai simple metode de sortare de complexitate $\mathbf{n^2}$.

A) Metoda bulelor (bubblesort)

Conform acestei metode, vectorul este parcurs de la stânga spre dreapta comparându-se perechi de elemente succesive (**a[j]** cu **a[j+1]**). Dacă cele două elemente nu sunt în ordinea cerută, se interschimbă şi, o variabilă iniţial egală cu 0, se incrementează. În acest fel, la prima parcurgere a vectorului, elementul maxim din şir (dacă se face ordonare crescătoare) se deplasează spre dreapta până când ajunge în ultima poziţie. La a doua parcurgere a vectorului , al doilea cel mai mare element ajunge în penultima poziţie etc. Parcurgerea vectorului se reia până când nu mai găsim nici-o pereche de elemente consecutive neordonate. La fiecare parcurgere, lungimea secvenţei care se verifică scade cu o unitate:

```
k=n;   /* iniţial verificăm tot vectorul */
do{
        f=0;     /* numără perechile neordonate */
        for(j=0;j<k-1;j++)
          if(a[j]>a[j+1])       /* pentru ordonare crescătoare */
          {   aux=a[j]; a[j]=a[j+1];
              a[j+1]=aux;       /* interschimbăm elementele */
              f++;              /* numărăm in f  */
          }
         k--;       /* lungimea secvenţei care se verifică scade */
}while(f);   /* repetă cât timp mai sunt perechi neordonate */
```

B)Sortarea prin selecţie directă

Conform acestei metode primul element (**a[0]**) se compară pe rând cu toate elementele de după el şi dacă ordinea de sortare nu este respectată, cele două elemente se interschimbă. După efectuarea tuturor comparaţiilor, în prima poziţie ajunge cel mai mic element din vector (în cazul ordonării crescătoare). Se compară apoi al doilea element cu toate elementele de după el etc. La ultimul pas se compară numai ultimele două elemente. Secvenţa corespunzătoare de program este :

```
for(i=0; i<n-1; i++)      /* elementul care se compară */
  for(j=i+1;j<n;j++)      /* elem. de după el cu care se compară */
     if(a[i]>a[j])        /* pentru ordonare crescătoare */
     {  aux=a[i]; a[i]=a[j];
        a[j]=aux;         /* interschimbăm elementele */
     }
```

6.3.11 Algoritmul de căutare binară

Se consideră un vector oarecare **A** cu **n** elemente şi o valoare **x**. Se cere să se verifice dacă **x** apare printre elementele vectorului sau nu. Dacă lucrăm cu un vector oarecare, se vor compara pe rând elementele acestuia cu valoarea căutată **x**. Sunt necesare cel mult **n** comparaţii în caz de succes (**x** apare în vector) şi exact **n** comparaţii în caz de eşec (**x** nu apare în vector). În cazul în care vectorul este ordonat crescător sau descrescător se poate folosi un algoritm de căutare mult mai eficient având complexitatea $\log_2 n$. Acest algoritm se numeşte "*algoritmul de căutare binară*" şi poate fi descris astfel : iniţial se consideră tot vectorul **A** şi se compară **x** cu elementul din mijlocul acestuia (fie el **a[mij]**). Dacă **x=a[mij]**, algoritmul se încheie cu succes; dacă **x<a[mij]**, vectorul fiind ordonat, căutarea va continua în prima jumătate, iar dacă **x>a[mij]** căutarea va continua în a doua jumătate. Procedeul se repetă până când fie găsim valoarea **x**, fie nu mai avem secvenţă validă de căutare, adică elemente neverificate. Secvenţa curentă în care se face căutarea elementului **x** este identificată prin indicele elementului din extrema stângă, respectiv indicele elementului din extrema dreaptă. Secvenţa de program este următoarea :

```
f=-1;  /* x nu a fost găsit în vectorul A */
st=0; dr=n-1;   /* secvenţa iniţială de căutare este întregul vector A */
while(st<=dr)    /* secvenţa curentă de căutare conţine cel puţin un element */
{   mij=(st+dr)/2;   /* indicele elementului din mijlocul secvenţei de căutare */
    if(a[mij]==x)
    {   f=mij;    /* memorăm poziţia în care apare */
        break;    /* căutare încheiată cu succes */
    }
    if(x<a[mij])
        dr=mij-1;       /* căutarea continuă în prima jumătate */
    else
        st=mij+1;       /* căutarea continuă în a doua jumătate */
}
```

6.3.12 Interclasarea vectorilor

Se consideră doi vectori, **A** cu **m** elemente şi **B** cu **n** elemente, ambii ordonaţi crescător sau descrescător. A interclasa cei doi vectori înseamnă a obţine un vector **C** cu **m+n** elemente, care conţine toate elementele din **A** şi din **B**, ordonate în acelaşi mod. Metoda de creare a vectorului rezultat **C** este foarte simplă: se compară elementul curent din **A** cu elementul curent din **B**, în **C** se introduce spre exemplu cel mai mic dintre ele (la ordonare crescătoare). Dacă elementul introdus a fost din vectorul **A** atunci se avansează la următorul element din **A**, altfel se avansează la următorul element din vectorul **B**. În momentul în care elementele unui vector sunt epuizate, elementele rămase în celălalt vector sunt copiate direct în vectorul C. Acest algoritm se simplifică dacă folosim două "*santinele*" care vor face ca cei doi vectori să se epuizeze simultan. O *santinelă* este un element care nu influenţează valoarea unui rezultat, scopul utilizării lui fiind numai obţinerea unui algoritm mai simplu şi mai eficient. În cazul algoritmului de interclasare, vom adăuga la sfârşitul vectorului **A** o santinelă mai mare decât cel mai mare element din **B**, iar la sfârşitul vectorului **B** o santinelă mai mare decât cel mai mare element din **A**. În acest caz, dacă presupunem că toate elementele utile din vectorul **A** au fost deja introduse în vectorul rezultat **C,** atunci, toate elementele utile din **B** care nu au fost încă verificate sunt mai mici decât santinela rămasă în **A** şi vor fi copiate automat, fără nici-o verificare suplimentară, în vectorul rezultat **C**. Secvenţa care realizează interclasarea cu santinele este:

```
a[m]=b[n-1]+1;  /* santinela din A mai mare decât cel mai mare element din B */
b[n]=a[m-1]+1;  /* santinela din B mai mare decât cel mai mare element din A */
i=0;            /* indicele elementului curent din A */
```

```
j=0;                /* indicele elementului curent din B */
for(k=0; k<m+n; k++)  /* k este indicele elementului curent din C */
  if(a[i]<b[j])  /* pentru ordonare crescătoare */
     c[k]=a[i++];         /* avansez în vectorul A */
  else
     c[k]=b[j++];         /* avansez în vectorul B */
```

6.4 Prelucrări elementare ale matricilor

Ca şi în cazul tablourilor unidimensionale, se pune problema depăşirii dimensiunilor maxime specificate în declaraţia unui tablou bidimensional (matrice). Matricile sunt de două tipuri: dreptunghiulare (numărul de linii diferit de numărul de coloane) şi pătratice (numărul de linii egal cu numărul de coloane).

Pentru matricile dreptunghiulare maniera uzuală de declarare şi validare a dimensiunilor este următoarea :

```
#define MAXLIN 7       /* numărul maxim de linii */
#define MAXCOL 5       /* numărul maxim de coloane */
int a[MAXLIN][MAXCOL];
int m,n;               /* numărul real de linii, respectiv coloane */
.....................................
do{
      printf("numarul de linii=");
      scanf("%d",&m);
      if(m<=0 || m>MAXLIN) printf("dimensiune eronata\n");
}while(m<=0 || m>MAXLIN);
do{
      printf("numarul de coloane=");
      scanf("%d",&n);
      if(n<=0 || n>MAXCOL) printf("dimensiune eronata\n");
}while(n<=0 || n>MAXCOL);
```

Pentru o matrice pătratică maniera uzuală de declarare şi validare a dimensiunii este următoarea:

```
#define DIM 8
int a[DIM][DIM];
int n;     /* dimensiunea reală a matricii */
.....
do{
      printf("dimensiunea matricii=");
      scanf("%d",&n);
      if(n<=0 || n>DIM) printf("dimensiune eronata\n");
}while(n<=0 || n>DIM);
```

Este bine de ştiut că memorarea matricilor se face pe linii (ordine lexicografică), adică compilatorul rezervă pentru matrice o zonă contiguă de memorie de dimensiune **MAXLIN*MAXCOL*sizeof(tip)**, unde **tip** este tipul de bază al matricii, adică tipul elementelor componente. Primele **MAXCOL** locaţii din această zonă sunt pentru elementele din prima linie (de indice 0), chiar dacă nu toate sunt folosite, următoarele **MAXCOL** locaţii sunt rezervate pentru elementele din a doua linie (de indice 1) etc. Practic, în memorie, matricea este liniarizată sub forma unui vector cu **MAXLIN*MAXCOL** elemente, unele din aceste elemente putând fi neutilizate. Raţionamentul se aplică identic pentru matricile pătratice, numărul de elemente fiind **DIM*DIM** (dimensiunea matricii).

În continuare, vom considera cazul general al matricilor dreptunghiulare, secvenţele de instrucţiuni trebuind modificate corespunzător pentru a lucra corect cu matricile pătratice (**m=n=dimensiunea matricii**).

6.4.1 Citirea elementelor unei matrici

Deoarece memorarea matricilor se face pe linii, elementele se citesc pe linii, adică indicele de coloană se modifică mai rapid decât indicele de linie. Secvenţa corespunzătoare de program este următoarea:

```
for(i=0;i<m;i++)          /* indicele de linie */
    for(j=0;j<n;j++)      /* indicele de coloană */
    {   printf("a[%d,%d]=",i,j);
        scanf("%d",&a[i][j]);
     }
```

6.4.2 Tipărirea elementelor unei matrici

Dacă dorim să tipărim matricea cu valorile de pe aceeaşi coloană aliniate spre exemplu la dreapta, vom folosi facilităţile de aliniere şi de specificare a numărului de zecimale (pentru valori reale) ale funcţiei **printf**. Spre exemplu, tipărirea unei matrici de numere reale cu valorile aliniate la dreapta şi trei zecimale exacte se poate realiza cu secvenţa:

```
for(i=0;i<m;i++)
{  for(j=0;j<n;j++)
      printf("%8.3f",a[i][j]);  /* tipăreşte elementele de pe o linie */
   printf("\n");       /* trece la o linie nouă */
}
```

6.4.3 Determinarea elementului maxim/minim

Metoda uzuală este următoarea: se iniţializează maximul/minimul cu primul element din matrice (**a[0][0]**), se compară apoi pe rând cu toate elementele din matrice şi se reţine valoarea mai mare/mai mică. Secvenţa de instrucţiuni care realizează acest lucru este următoarea:

```
maxim=a[0][0];       /* minim=a[0][0]  */
for(i=0;i<m;i++)
   for(j=0;j<n;j++)
       if(maxim<a[i][j])          /* minim>a[i][j] */
              maxim=a[i][j];       /*  minim=a[i][j] */
```

6.4.4 Identificarea elementelor specifice unei matrici pătratice

În cazul matricilor pătratice se identifică următoarele elemente specifice: diagonala principală (**DP**), diagonala secundară (**DS**), jumătatea superioară (**JS**) şi jumătatea inferioară (**JI**). Diagonalele corespund (geometric) diagonalelor unui pătrat.

Diagonala principală cuprinde elementele din colţul stânga sus până în colţul dreapta jos, adică mulţimea:

DP={a_{ij}| i=j, i=0,…..n-1}={a_{00}, a_{11}, a_{22}, ……,a_{n-1n-1}} unde **n** este dimensiunea reală a matricii.

Diagonala secundară cuprinde elementele din colţul dreapta sus până în colţul stânga jos, adică mulţimea:

DS={a_{ij}|i=0,1,…n-1 , j=n-1,n-2,…0, i+j=n-1}={a_{0n-1}, a_{1n-2},……..,a_{n-10}}

Jumătatea superioară cuprinde elementele de deasupra diagonalei principale (fără cele de pe **DP**), adică mulţimea:

JS={a_{ij} | i=0,1,……n-2 , j=i+1,……,n-1}

Jumătatea inferioară cuprinde elementele de sub diagonala principală (fără elementele de pe **DP**), adică mulţimea:

JI={a_{ij} | i=1,2,…….n-1 , j=0,1,……i-1}

Un element **a[i][j]** din **JS** are drept simetric elementul **a[j][i]** din **JI**. Astfel, transpusa unei matrici pătratice (matricea care are liniile drept coloane şi reciproc) se poate genera uşor interschimbând toate elementele din **JS** cu simetricele lor din **JI** aşa cum se vede în secvenţa de mai jos:

```
for(i=0; i<n-1; i++)
```

```
for(j=i+1; j<n; j++)
{  aux=a[i][j];
   a[i][j]=a[j][i];
   a[j][i]=aux;
}
```

6.5 Exerciţii şi teste grilă

1. Care dintre variantele de mai jos reprezintă o declaraţie corectă a unui vector **v** cu 20 de elemente numere întregi ?

a) `v[20]:integer;` **b)** `v[20] int;`
c) `int v[20];` **d)** `int :v[20];`
e) `integer v[20];`

2. Câte erori conţine programul de mai jos?

```
void main()
{
      int n,k;
      int v[n];
      n=4
      for(k=0;k<n;k++) v[k]= =0;
}
```

a) 0 b) 1 c) 2
d) 3 e) 4

3. Care dintre secvenţele de program de mai jos calculează corect suma primelor **n** elemente ale unui vector **s** ?

a)
```
s=0;
for(i=0;i<n;i++) s+=v[i];
```
b)
```
s=0; i=0;
while(i<n)
{ s+=v[i]; i++;}
```
c)
```
s=0; i=0;
do{
    s+=v[i]; i++;
}while(i<n-1);
```
d) toate **e)** nici una

4. Deduceţi care vor fi elementele vectorului **v** după execuţia secvenţei de program următoare:

```
int n,k,x,v[7]={5,14,-3,8,-1};
n=5; x=v[0];
for(k=1;k<n;k++) v[k-1]=v[k];
v[n-1]=x;
```

a) (-1,5,14,-3,8,0,0)
b) (14,-3,8,-1,0,0,5)
c) (14,-3,8,-1,5,0,0)
d) (0,0,5,-3,14,-1,8)
e) (0,0,-1,14,-3,8,5)

5. Câte elemente ale vectorului **v** vor avea valoarea 9 după execuţia programului de mai jos?

```
#include<stdio.h>
void main()
{
      int v[]={0,1,2,0,4,5,6};
      int i=0,x=9;
      do{
            v[i++]=x;
      }while(i<6 && v[i]);
}
```

a) nici unul b) unul
c) două d) trei e) toate

6. Fie programul următor :

```
void main()
{
      int i,j,m,n,p,a[10][10],b[6];
      m=2; n=3; p=6; i=0;
      while(i<p) b[i++]=i;
      for(i=0;i<m;i++)
        for(j=0;j<n;j++)
           a[i][j]=b[3*i+j];
}
```

În urma execuţiei sunt posibile următoarele situaţii:

a) programul nu funcţionează din cauză că declaraţia matricei este eronată
b) valorile vectorului **b** sunt 0,1,2,3,4,5
c) valorile vectorului **b** sunt 1,2,3,4,5,6
d) **a[1][0]** are valoarea 3
e) **a[0][2]** are valoarea 2

7. Se consideră o matrice **a** cu **n** linii*__n__ coloane şi un vector **v** cu **n** elemente. Precizaţi care vor fi elementele vectorului **v**, după execuţia secvenţei următoare:

```
int nr, n, i, j, x, b[20];
int a[3][3]={{7,1,7},{-7,7,0},
             {2,4,11}};
n=3; x=7;
for(i=0;i<n;i++)
{   nr=0;
    for(j=0;j<n;j++)
       if(a[i][j]==x) nr++;
    b[i]=nr;
}
```

a) nedefinite b) v=(0,0,0)
c) v=(1,2,3) d) v=(2,0,1)
e) v=(2,1,0)

8. Se consideră secvenţa de program următoare, în care **a** este o matrice cu **n** linii***n** coloane şi elemente numere întregi, iar **x** este o variabilă de tip întreg.

```
x=1;
for(i=1;i<=n-1;i++)
  for(j=0;j<=i-1;j++)
    if(a[i][j]!=0) x=0;
```

În urma execuţiei secvenţei, valoarea variabilei **x** va fi 1 dacă:

a) deasupra diagonalei principale există cel puţin un element egal cu 0
b) toate elementele de deasupra diagonalei principale sunt 0
c) toate elementele de sub diagonala principală sunt diferite de 0
d) toate elementele de sub diagonala principală sunt 0
e) sub diagonala principală există cel puţin un element diferit de 0

9. Fie următorul program:

```
#include<stdio.h>
void main()
{
   int v[20], i, n, E;
   scanf("%d", &n);
   for(i=0;i<n;i++) v[i]=i%2 ?i:-i;
   for(E=1,i=0;i<n; E*=v[i++]);
   E++;
   printf("%d",E);
}
```

În urma execuţiei sale sunt posibile următoarele situaţii:

a) expresia condiţională din primul ciclu **for** este eronată din punct de vedere sintactic
b) dacă variabila **n** primeşte prin citire valoarea 6, atunci elementele vectorului **v** vor fi, în ordine (0,1,-2,3,-4,5)
c) prezenţa caracterului ";" după al doilea ciclu **for** constituie o eroare
d) dacă variabila **n** primeşte prin citire valoarea 5, atunci programul afişează 32
e) programul funcţionează corect pentru orice valoare întreagă a lui **n** mai mică sau egală cu **MAXINT**

10. Care dintre secvenţele de program de mai jos afişează corect elementele **v[0], ...v[n-1]** ale unui vector de întregi ?

```
a) i=0;
   while(i<n)
   { printf("%d", v[i]); i++;}
b) i=0;
   while(i<n)
   { i++; printf("%d", v[i]); }
c) i=0;
   do{ i++;
       printf("%d", v[i]);
   }while(i<n);
d) i=0;
   do{ printf("%d", v[i]);
       i++
   }while(i<n);
e) nici una
```

11. Care dintre secvenţele de mai jos afişează corect produsul elementelor pare ale unui vector **v[0],.....,v[n-1]** cu **n** elemente întregi ?

```
a) p=1;
   for(j=1;j<=n;j++)
       if(v[j]%2==) p=p*v[j];
b) p=1;
   for(j=0;j<n;j++)
     if(v[j]/2==0) p=p*v[j];
c) p=0;
   for(j=0;j<n;j++)
     if(v[j]%2!=0) p=p*v[j];
d) p=1;
   for(j=0;j<n;j++)
     if(v[j]%2==0) p*=v[j];
e) p=0;
   for(j=0;j<n;j++)
     if(v[j]%2==0) p*=v[j];
```

12. Care dintre afirmaţiile de mai jos sunt adevărate pentru secvenţa de program următoare ?

```
p=0;
for(k=1;k<6;k++)
  if(v[k]>v[p]) p=k;
printf("%d", p);
```

a) secvenţa este corectă din punct de vedere sintactic
b) ciclul **for** are cinci paşi
c) dacă elementele vectorului sunt 5,4,-11,9,-12,1,atunci programul afişează valoarea 4
d) dacă elementele vectorului sunt 3,-2,8,6,11,4, atunci programul afişează valoarea 4
e) indiferent care ar fi elementele vectorului, secvenţa dată nu poate afişa valoarea 0

13. Pentru secvenţa de program următoare, precizaţi care dintre afirmaţiile de mai jos sunt adevărate:

```
int j=0 , v[5]={1,1,1,1,1};
while(j<5&&v[j]&&!v[j])
{v[j]=0;j++;}
```

a) expresia din **while** este eronată sintactic
b) declaraţia şi iniţializarea vectorului sunt corecte
c) după execuţia secvenţei toate elementele vectorului vor fi 1
d) după execuţia secvenţei toate elementele vectorului vor fi 0
e) execuţia secvenţei va produce un ciclu infinit

14. Precizaţi care va fi efectul secvenţei de program următoare, în care **v[0],.....,v[n-1]** este un vector cu **n** elemente întregi.

```
x=v[n-1];
for(k=n-1;k>0;k--) v[k]=v[k-1];
v[0]=x;
```

a) deplasează toate elementele vectorului cu o poziţie la dreapta
b) deplasează toate elementele vectorului cu o poziţie la stânga
c) şterge un element din vector prin deplasarea celor aflate înaintea lui
d) roteşte circular vectorul cu o poziţie
e) nici una din variantele anterioare

15. Fie secvenţa de program următoare, în care **v** este un vector cu **n** elemente întregi, iar **p** este o variabilă întreagă:

```
for(p=1,k=1;k<n;k++)
   if(v[k]==v[k-1]) p=0;
printf("%d", p);
```

Secvenţa afişează 0 dacă:

a) toate elementele sunt distincte două câte două
b) toate elementele sunt egale
c) există două elemente consecutive distincte
d) există două elemente consecutive egale
e) nici una din variantele de mai sus

16. Ce valoare va fi afişată în urma execuţiei programului următor?

```
#include<stdio.h>
void main()
{
      int v[]={0,1,2,0,4,5,6};
      int j=0,nr=0;
      do{
            if(j==v[j]) nr++;
      }while(j<6&&v[j++]);
      printf("%d",nr);
}
```

a) 0 b) 1 c) 3
d) 5
e) programul intră în buclă infinită

17. Deduceţi care vor fi, în ordine, de la stânga la dreapta, elementele nenule ale vectorului **a** la sfârşitul execuţiei secvenţei de program următoare:

```
int k, j=0;
int v[7]={0,2,7,3,4,8,5};
int a[7]={0,0,0,0,0,0,0};
for(k=0;k<7;k++)
  if((v[k]%2==0)&&(k%2!=0))
  {  a[j]=v[k]; j++; }
```

a) 2,4,8 b) 7,5 c) 2,8
d) 2,3,8 e) 7,3,5

18. Care parcurgere pe linii şi coloane a unei matrici **n*m** este corectă ?

```
I)    for(i=0;i<n;i++)
         for(j=0;j<m;j++)
           printf("%d",a[i][j]);
II)   for(j=1;j<m;j++)
         for(i=1;i<m;i++)
           printf("%d",a[i][j]);
III)  for(i=0;i<n;i++)
         for(i=0;j<m;i++)
           printf("%d",a[i][j]);
```

a) doar I b)doar II
c) doar I si II
d) doar I si III

19. Care din secvenţele de calcul a mediei unui şir de **n** întregi este corectă ?

```
I)    int i,n,s;
      float med;
      s=0;
      for(i=0;i<n;i++) s+=a[i];
      med=s/n;
II)   int i,n,s;
      float med;
      for(i=0,s=0;i<n;i++) s+=a[i];
      med=(float)s/n;
III)  int i,n,;
      float med, s;
      s=0;
      for(i=0;i<n;i++) s+=a[i];
      med=s/n;
```

a) doar I b)doar II
c) doar III d) doar II şi III

20. Calculul mediei geometrice a unui şir de **n** întregi este:

```
I)    int i,n, pr;
      float med;
      for(pr=1,i=0;i<n;i++)
         pr*=a[i];
      med=sqrt(pr);
II)   int i,n, pr;
```

```
      float med;
      for(pr=1,i=0;i<n;i++)
         pr*=a[i];
      med=pow(pr, 1/n);
III)  int i,n, pr;
      float med;
      for(pr=1,i=0;i<n;i++)
         pr*=a[i];
      med=pow(pr, 1./n);
```

a) doar I b) doar II
c) doar III d) doar II şi III

21. Ce secvenţă calculează corect maximul unui şir cu **n** valori?

```
I)    for(max=0, i=0;i<n;i++;)
         if(a[i]>max) max=a[i];
II)   max=a[0];
      for(i=0;i<n;i++)
         if(a[i]>a[i++]) max=a[i];
III)  max=a[0];
      for(i=0;i<n;i++)
         if(a[i]>max) max=a[i];
```

a) doar I b) doar II
c) doar III d) doar I şi II

22. Fie **X[1..n]** si **Y[1..n]** vectori de întregi. Care va fi valoarea lui **Y[n]** după execuţia secvenţei:

```
Y[1]=x[1];
for (i=2;i<n;i++) y[i]=y[i-1]+x[i];
```

a) x[n]+x[n-1] b) x[n]
c) x[1]+x[2]+……..+x[n]
d) nici una din valorile indicate

23. Fie **X[1..n]** si **Y[1..n]** vectori de numere reale. După execuţia secvenţei de program :

```
y[1]=-x[1];
for(i=2;i<n;i++) y[i]=y[i-1]*x[i];
```

elementul **Y[n]** exprimă :

a) x[1]*x[2]*……*x[n]
b) -x[1]*x[2]*….*x[n]
c) $(-1)^n$ x[1]*…..*x[n]
d) nici una din valorile indicate

24. Fie **V[1..n]** vector de intregi . Secvenţa de program :

```
i=1;
for(i=1;i<=n/2;i++)
{    j=n-i;
     aux=v[i]; v[i]=v[j];
     v[j]=aux;
}
```

are ca efect :

a) inversarea ordinii elementelor în vector
b) inversarea ordinii tuturor elementelor în vector numai când **n** este impar
c) inversarea ordinii tuturor elementelor în vector numai când **n** este par
d) nici una din variantele indicate

25. Se consideră matricea pătratică **A(mxm**) . Fie secvenţa de program :

```
x=a[0][m-1];
for(i=0;i<m;i++)
  if x<a[i][m-i]  x=a[i][m-i];
```

Variabila **x** calculată exprimă :

a) valoarea maximă de pe diagonala principală
b) valoarea maximă de pe diagonala secundară
c) valarea maximă din întreaga matrice
d) altă valoare decat cele indicate

26. Fie matricea pătratică **A(mxm)** şi secvenţa de program :

```
x=a[0][0];
for(i=0;i<m;i++)
{   if(x<a[i][i]) x=a[i][i];
    if(x<a[i][m-i-1])
        x=a[i][m-i-1];
 }
```

Variabila **x** calculată reflectă :

a) valoarea cea mai mare de pe diagonale
b) valoarea cea mai mare de pe diagonala principală
c) valoarea cea mai mare de pe diagonala secundară
d) altă valoare decat cele indicate

27. Fie matricea **A(mxm)** şi secvenţa de program:

```
s=1;
for(i=0;i<m;i++)
  for(j=0;j<i;j++) s*=a[i][j];
```

Valoarea calculată **s** reflectă :

a) produsul valorilor de sub diagonala principală
b) produsul valorilor de pe diagonala secundară
c) produsul valorilor din întreaga matrice
d) altă valoare decât cele indicate

28. Fie o matrice **A(nxn)** citită de la tastatură . Atunci secvenţa de cod :

```
i=0; m=a[0][0];
while(i<n) {
  for(j=0;j<n;j++)
    if(m<a[i][j])  m=a[i][j];
  i++;
}
```

a) calculează elementul maxim **m** dintre elementele matricii
b) calculează elementul minim **m** dintre elementele matricii
c) se ciclează la infinit

d) numără elementele matricii care sunt mai mari ca **m**

29. Fie secvenţa de cod de mai jos. Dacă considerăm ca date de intrare tabloul **V1** de dimensiune **n**, atunci tabloul **V0** va conţine :
```
for(i=0;i<n;i++)  v0[n-i-1]=v1[i];
```
a) elementele lui **V1** în ordine inversă
b) o parte din elementele lui **V1**
c) numai elem. impare ale lui **V1**
d) algoritmul depune elementele lui **V1** în ordine inversă numai dacă **n** este impar

30. Fie următoarea secvenţă de cod care primeşte o matrice **M** pătratică de dimensiune **n** la intrare :
```
p=1;
for(i=0;i<n;i++) p=p*m[i][i]-2;
```
atunci aceasta realizează :
a) produsul elementelor din matrice
b) produsul elementelor de pe diagonala principală a matricii **m**
c) o prelucrare a elementelor de pe diagonala principală a matricii **m**
d) se ciclează la infinit

31. Care este valoarea elementului **tab[2][3]** după execuţia secvenţei de cod :
```
int i,j;
int ctr=0;
int tab[4][4];
for(i=0;i<4;i++)
   for(j=0;j<4;j++)
   { tab[i][j]:=ctr; ++ctr; }
```
a) 7 b) 9 c) 11 d) 14

32. Ce face următoarea secvenţă ?
```
scanf("%d",&n);
for(i=0;i<n;i++) scanf("%d",&a[i]);
printf("\n%d",a[0]);
for(i=1;i<n;i++)
{  ex=0;
   for(j=0;j<i;j++)
     if(a[i]==a[j])    ex=1;
   if(!ex) printf("%d",a[i]);
}
```
a) afişează numerele dintr-un şir care sunt în mai multe exemplare
b) afişează numerele cu apariţie singulară în şir
c) afişează numerele dintr-un şir
d) afişează numerele impare dintr-un şir

33. Ce realizează următoarea secvenţă de program?
```
i=0;
do {
    i++;
    a[i]=nr%2;
    nr/=2;
}while(nr);
n=i;
for(i=n;i>0;i--) printf("%d",a[i]);
```
a) convertirea b10->b2 a unui număr fracţionar
b) convertirea b10->b2 a unui număr întreg
c) convertirea b10->b3 a unui număr întreg
d) convertirea b10->b2 a unui număr întreg pozitiv

34. Se dă următoarea secvenţă de cod :
```
int y[5]={3,4,5,6,0};
```
Ce valoare conţine **y[3]** ?
a) 3 b) 5 c) 6
d) codul nu compilează pentru că nu sunt destule valori

35. Se dă codul :
```
short testarray[4][3]=
          {{1},{2,3},{4,5,6}};
printf("%d\n",sizeof(testarray));
```
Presupunând că tipul "**short**" este de lungime 2 octeţi, ce va afişa codul de mai sus ?
a) nu va compila pentru că nu sunt daţi destui iniţializatori
b) 6 c) 12 d) 24

36. Fie secvenţa de cod :
```
int x,i,t;
int y[10]={3,6,9,5,7,2,8,10,0,3};
while(1)
{    x=0;
     for(i=0;i<9;i++)
       if (y[i]>y[i+1])
       {     t=y[i];
             y[i]=y[i+1];
             y[i+1]=t;
             x++;
       }
     if(x==0) break;
}
```
Cum va arăta vectorul după execuţia acestui cod ?
a) programul va rula la infinit
b) {0,2,3,3,5,6,7,8,9,10}
b) {10,9,8,7,6,5,3,3,2,0}
c) {3,6,9,5,7,2,8,10,0,3}

37. Există greşeli în secvenţa de calculare a mediei aritmetice?
```
#include<stdio.h>
#include<conio.h>
void main(void)
{  int a[30],i,n=20;
    int s=0; float ma;
```

```
    for(i=0;i<n;i++)
    {  printf("\na[%d]=",i);
       scanf("%d",&a[i]);
    }
    for(i=0;i<n;i++) s=s+a[i];
    ma=s/n;
    printf("\nRezultatul=%f",ma);
    getch();
}
```

a) nu, secvenţa este corectă
b) da, deoarece nu au fost citite toate elementele tabloului
c) da, deoarece va fi afişată doar partea întreagă a rezultatului
d) da, deoarece nu au fost citite corect toate elementele vectorului

Cap.7 Pointeri

7.1 Variabile pointer

În memoria internă, valorile variabilelor sunt stocate în locaţii care sunt referite prin numere numite adrese. O locaţie de memorie are asociată o adresă unică. Cea mai mică entitate de memorie adresabilă direct este bytul sau octetul. Să presupunem variabila de tip întreg **a** cu valoarea 7 ce ocupă în memorie doi octeţi. Valoarea variabilei **a**, care este 7, ocupă o locaţie de memorie care se poate referi direct prin numele variabilei sau indirect prin adresa ei, **adr(a)**.

O variabilă capabilă să stocheze adresa unei alte variabile se numeste pointer.

Limbajul C pune la dispoziţie doi operatori pentru lucru cu adrese :

- operatorul **&** cu rol de extragere a adresei unei variabile
- operatorul ***** cu rol de extragere a conţinutului zonei de memorie adresate de o variabilă pointer

Variabila pointer se defineşte în concordanţă cu un tip de dată. De exemplu, declaraţia **int *px;** precizează că **px** este un pointer spre întreg, adică variabila **px** este capabilă să stocheze adresa unui întreg. Operatorul ***** , în declaraţie, are rolul de a exprima faptul că nu **px** este de tip întreg ci conţinutul de la adresa memorată în **px** este de tip întreg. Deci **px** este un pointer la întreg.

Fie programul :

```
#include<stdio.h>
void main()
{
      int x=5,*px;
      px=&x;
      printf("\nx=%d",x);
      printf("\nx=%d",*px);
}
```

În urma execuţiei lui se va afişa de două ori valoarea variabilei **x**, adică 5. În primul caz s-a folosit adresarea directă a variabilei folosindu-se numele ei. În al doilea caz s-a folosit adresarea indirectă a variabilei, mai precis, s-a extras conţinutul de la adresa ei **(*px**). Se observă că, în prealabil, adresa variabilei **x** a fost stocată în pointerul **px**.

Pe lângă definirea de pointeri spre tipurile fundamentale cum ar fi :

```
float  *py;   - py este un pointer la float
char   *pc;   - pc este un pointer la caracter
```

există posibilitatea definirii acestora şi spre tipuri structurate de date. Astfel, declaraţiile :

`int (*px)[10];` exprimă faptul că **px** este un pointer la un vector cu 10 elemente de tip întreg

`int *pt[10];` exprimă faptul că **pt** este un vectori de pointeri la întregi.

Este important ca pointerul să stocheze adresa unei variabile ce are un tip bine definit, pentru ca expresii de genul ***pointer** să se poată evalua corect Conţinutul de la o adresă poate fi de dimensiuni diferite, minim un octet. Având bine precizat tipul de dată referit de pointer, conţinutul de la acea adresă este extras în mod corect în funcţie de mărimea în octeţi a tipului de dată respectiv. În exemplu, ***px** referă doi octeţi de la adresa stocată în **px** considerând că un întreg se memorează pe 2 octeţi. Există şi pointerul generic, adică un pointer fără tip de dată asociat (spre **void**) ce se declară sub forma **void *p**. Acest tip de pointer este folosit mai mult pentru transferul şi stocarea adreselor în cadrul programelor.

Un alt aspect important ce ţine de lucrul cu variabile pointer, care generează erori în munca de programare, se referă la faptul că se poate accesa conţinutul unei variabile pointer numai după ce pointerul referă o zonă de memorie ce a fost alocată în prealabil. Se mai spune că, în acest caz, pointerul conţine o adresă validă .

Prezentăm două moduri de încărcare a unui pointer :

a) <u>cu adresa unei variabile anterior definită</u>; în acest caz alocarea s-a făcut la momentul definirii variabilei :

```
int a=30,*pa;
pa=&a;
```

b) <u>prin alocare dinamică de memorie</u> ce se face în momentul execuţiei programului ; pentru realizarea unei astfel de operaţii se poate folosi operatorul **new**. Forma generală este :

pointer=new tip;

unde : - *pointer* – reprezintă variabila pointer ce urmează a se încărca
- *tip* – reprezintă un tip de date pentru care se face alocarea zonei de memorie; se foloseşte pentru a determina mărimea în octeţi a zonei de memorie ce urmează a se aloca şi care este egală cu **sizeof(tip)** .

Exemplu:

```
int *pi;
pi=new int;
*pi=5;
printf("\n%d",*pi);
```

În cazul în care se doreşte a se aloca o zonă de memorie care să stocheze mai mulţi întregi (**nr**) atunci se va folosi operatorul **new** în forma :

pointer=new tip[nr];

unde : - *tip* – este tipul de dată referit de pointer
- *nr* – reprezintă numărul de elemente de tipul *tip*

Mărimea în octeţi ce se va aloca se obţine conform relaţiei : **nr*sizeof(tip)** .
Legat de acest operator, limbajul C pune la dispoziţie şi operatorul **delete** pentru dealocarea unei zone de memorie alocată în prealabil cu operatorul **new**. Forma de utilizare a operatorului este :

delete pointer sau **delete [nr]pointer**

unde : - *pointer* – reprezintă variabila pointer
- *nr* – reprezintă numărul de elemente cu tipul referit de variabila *pointer*

Exemplu: Se va aloca dinamic o zonă de memorie capabilă să stocheze *n* valori double, după care spaţiul va fi dealocat.

```
double *p;
int n=5;
p=new double[n];
……..
delete [n]p;
```

Memoria alocată dinamic îşi păstrează conţinutul până când se dealocă în mod explicit în cadrul programului.

7.2 Aritmetica pointerilor

Aritmetica pointerilor reprezintă ansamblul operaţiilor care se pot efectua cu o variabilă pointer. Acestea sunt :

1) *Operaţia de extragere a conţinutului unei variabile pointer* care a fost exemplificată anterior **(*pointer**)
2) *Operaţia de extragere a adresei unei variabile pointer*. Având în vedere că pointerul este la rândul lui tot o variabilă şi acesteia i se poate extrage adresa. Variabila care memorează adresa unei alte variabile pointer se numeşte **pointer la pointer**.

Exemplu :

```
int a=15,*pa,**ppa;
/*  ppa este un pointer la pointer la întreg */
pa=&a;
ppa=&pa;
printf("\na=%d",**ppa);
```

Variabila **a** a fost afişată folosindu-se o dublă indirectare .

3) *Operaţia de atribuire între pointeri* este permisă doar dacă pointerii referă acelaşi tip. Fie secvenţa :

```
int a=7,*pa1,*pa2;
pa1=&a;
pa2=pa1;
/* ambele variabile pointer conţin adresa lui a */
printf("\n a=%d a=%d",*pa1,*pa2);
```

Se va afişa de două ori valoarea 7 indirect, prin intermediul celor doi pointeri care conţin aceeaşi adresă.

4) *Operaţia de incrementare, respectiv decrementare a unui pointer*, presupune obţinerea unei adrese mai mari, respectiv mai mici, în funcţie de dimensiunea tipului de dată referit de pointer.

Exemplu :

```
double a[]={5.3,2.1},*pa;
pa=&a[0];
```

```
/* adresa de început a tabloului , adică adresa elementului a[0] */
printf("\n Primul element=%lf",*pa);
pa++;
/*  conţine adresa următorului element din tablou , adică adresa lui a[1] */
printf("\n Al doilea element=%lf",*pa);
```

5) *Operaţia de adunare a unui întreg la un pointer*. Dacă se adună variabila **k** la un pointer spre tipul **TIP** se obţine o dresă mai mare cu **k*sizeof(TIP)**.

Exemplu :

```
double a[]={5.3,2.1,8.9,10},*pa;
int i;
pa=&a[0];
i=2;
printf("\n %lf",*(pa+i));
/* se va afişa al treilea element din vector adică 8.9 */
i=3;
print("\n %lf",*(pa+i));
/* se va afişa al patrulea element din vector adică 10 */
```

6) *Operaţia de conversie între pointeri*. Această operaţie se face cu ajutorul operatorului de conversie explicită (**cast**). Pentru exemplificare vom prezenta o funcţie ce are ca scop alocarea dinamică de memorie . Ea are prototipul : **void *malloc(int);** care se află în header-ul **alloc.h**. Parametrul are ca scop dimensionarea zonei de memorie în octeţi şi funcţia returnează adresa la care s-a alocat zona de memorie. Se observă că această funcţie returnează un pointer la *void* adică un pointer generic care nu poate fi folosit în cele mai multe operaţii uzuale asupra pointerilor. De aceea, el trebuie convertit într-un pointer spre un tip de dată bine precizat. Pentru dealocarea zonei de memorie ce a fost alocată anterior cu funcţia **malloc**, biblioteca limbajului pune la dispoziţie funcţia **free** care are prototipul : **void free(void *);**

Parametrul funcţiei reprezintă pointerul încărcat printr-un apel al funcţiei **malloc**. În secvenţa :

```
int *pa,n=5;
pa=(int*)malloc(n*sizeof(int));      /* conversie în pointer spre întreg */
.....
free(pa);
```

s-a alocat o zonă de memorie capabilă să stocheze 5 variabile de tip întreg. Adresa zonei a fost stocată în variabila pointer spre întreg **pa**. După ce s-au efectuat prelucrările necesare asupra zonei de memorie şi ea nu mai este necesară în program, atunci se eliberează spaţiul cu funcţia **free**.

7) *Operaţia de scădere a doi pointeri* necesită ca pointerii să fie spre acelaşi tip. Rezultatul reprezintă numărul de elemente de tipul referit de pointer ce se află între cele două adrese. În cazul în care tipul de dată referit de pointer este **TIP** atunci diferenţa dintre doi pointeri (p1 şi p2) se calculează după relaţia **(p1-p2)/sizeof(TIP)**.

Ca exemplu, se va scrie secvenţa de program care determină lungimea unui şir de caractere :

```
char  sir[50],*pc;
printf("\n Dati sirul : ");gets(sir);
pc=&sir[0];
while(*pc++);
printf("\n Lungimea sirului este %d",pc-sir-1);
```

8) *Operaţia de comparaţie dintre pointeri* se realizează folosind operatorii de egalitate şi cei relaţionali. Astfel, ca exemplu, se va prezenta secvenţa de traversare a unui vector în scopul afişării elementelor sale:

```
int a[]={5,4,3,2,7};
int n=sizeof(a)/sizeof(int); /* numărul de elemente din vector*/
int *p1,*p2,*pc;
for(p1=pc=&a[0],p2=&a[n-1];pc<=p2;pc++)
     printf("\n elementul %d",*pc);
```

7.3 Legătura pointer – tablou

Tabloul, prin definiţie, este o structură de date omogenă cu un număr finit şi cunoscut de elemente. Fiind definit tabloul **int a[50]**, numele lui reprezintă adresa primului element din tablou

(**a=&a[0]**). Având stabilită această legătură se pot defini mai multe modalităţi echivalente de accesare a unui element din tablou. Astfel, elementul de rang **i** se accesează folosind una din relaţiile :

```
a[i];              (1)
*(a+i);            (2)
*(&a[0]+i);        (3)
i[a];              (4)
```

c) relaţia (1) este forma obişnuită de accesare a elementului de rang **i** numită şi adresare **indexată**
d) relaţia (2) este forma de adresare a elementului de rang **i indirect**, pornind de la adresa acestuia
e) relaţia (3) este derivată din relaţia 2, doar că se pune în evidenţă în mod explicit că numele tabloului este adresa primului element din vector
f) relaţia (4) derivă din faptul că expresia **a[b]** este evaluată de compilator în forma ***(a+b)**. Deoarece adunarea este comutativă, pentru exemplul nostru **a[i]⇔ *(a+i) ⇔ *(i+a) ⇔ i[a].**

Acelaşi concept se aplică tablourilor cu două sau mai multe dimensiuni. De exemplu, presupunând că **a** este un tablou de întregi cu 10 linii şi 10 coloane (**a[10][10]**), următoarele două expresii sunt echivalente: **a şi &a[0][0]**. În continuare, elementul din linia 0 şi coloana 4 poate fi referit în două moduri: fie prin aplicarea de indecşi, **a[0][4]**, fie prin intermediul unui pointer ***(a+4)**. În general, pentru orice tablou bidimensional, **a[j][k]** este echivalent cu una din formele :

***(a+(j*numar_coloane)+k)**
***(a[j]+k)**
***(*(a+j)+k)**

Ca observaţie, numele unui vector este un pointer constant. O consecinţă este că aritmetica de pointeri ce se aplică acestui tip de pointer este mai restrânsă în sensul că, acest pointer nu-şi poate modifica valoarea. O altă particularitate constă în faptul că operatorul **sizeof** aplicat unui vector întoarce numărul de octeţi al întregii zone ocupată de vector.

Până acum s-a evidenţiat legătura dintre pointer şi tablou în sensul că s-a definit un tablou şi a fost exploatat prin intermediul pointerilor. Sensul este şi reciproc, adică definind un pointer, zona de memorie alocată poate fi utilizată ca şi cum ar fi numele unui tablou. Ca exemplu, se va defini o zonă de memorie capabilă a stoca **n** întregi, apoi se va încărca zona de memorie prin citire de la tastatură şi se va calcula suma elementelor introduse.

```
int *pa,n,s;
printf("\n Nr. de elemente=");
scanf("%d",&n);
pa=(int*)malloc(n*sizeof(int));
for(int i=0;i<n;i++)
{     printf("\n elem[%d]=",i+1);
      scanf("%d",&pa[i]);
}
for(i=s=0;i<n;i++) s+=pa[i];
printf("\n suma=%d",s);
```

Se observă că elementele zonei de memorie referite de pointerul **pa** au fost accesate folosindu-se adresarea indexată **pa[i]** care este specifică lucrului cu tablouri.

În limbajul C, pointerii şi tablourile sunt în strânsă legătură ei fiind de fapt interschimbabili. Dacă folosiţi numele unui tablou fără index, generaţi de fapt un pointer către primul element al tabloului. Iată de ce nu este necesar nici un index atunci când are loc citirea unui şir folosind funcţia **gets().** Ceea ce este transferat funcţiei **gets()** nu este un tablou, ci un pointer. De fapt, în C nu se poate transfera un tablou unei funcţii, ci numai un pointer către tablou. Funcţia **gets()** foloseşte un pointer pentru a încărca tabloul de caractere introdus de la tastatură.

Exemplul 1: Deoarece numele unui tablou fără index este un pointer către primul element al tabloului, această valoare poate fi asignată unui alt pointer şi prin aceasta devine posibilă accesarea elementelor tabloului folosind pointerul aritmetic. Să considerăm următorul program:

```
#include<stdio.h>
int a[10]={1,2,3,4,5,6,7,8,9,10};
void main()
{  int *p;
   p=a;    /* asignează lui p adresa de început a tabloului a */
   /* afişarea primelor trei elemente ale tabloului a */
   printf("%d %d %d",*p,*(p+1),*(p+2));
   /* afişează acelaşi lucru */
      printf("%d %d %d",a[0],a[1],a[2]);
```

```
}
```

În acest caz, ambele instrucţiuni **printf()** afişează acest lucru. Parantezele din expresii ca ***(p+2)** sunt necesare, deoarece operatorul ***** referitor la pointeri are o precedenţă mai mare decât operatorul **+.** Dacă folosiţi un pointer pentru a accesa un tablou bidimensional, trebuie să executaţi manual ceea ce compilatorul face automat. De exemplu, în tabloul `float balance[10][5];`fiecare rând are 5 elemente. Pentru a accesa **balance[3][1]** folosind un pointer de tip float, trebuie utilizat un fragment de forma: `*(p+(3*5)+1);`

În general, la tablourile multidimensionale este mai uşoară folosirea indexării numelui decât folosirea pointerului aritmetic.

Exemplul 2: Un pointer se poate indexa ca şi când ar fi un tablou. Următorul program este valid:

```
#include<stdio.h>
char str[]="Pointerii sunt puternici";
void main()
{   char *p;
    int j;
    p=str;
    for(j=0;p[j];j++)   /* ciclare până la întâlnirea  caracterului NULL */
         printf("%c",p[j]);
}
```

Exemplul 3: Un pointer poate fi indexat numai dacă el punctează un tablou. Deşi următorul program este corect din punct de vedere sintactic, executarea lui va conduce probabil la blocarea calculatorului:

```
char *p,ch;
int i;
p=&ch;
for(i=0;i<10;i++) p[i]='A'+i;
```

Întrucât **ch** nu este un tablou, pointerul **p** nu poate fi indexat. Deşi un pointer punctând un tablou poate fi indexat ca şi cum ar fi un tablou, rareori se va proceda astfel, deoarece, în general, prin folosirea pointerilor aritmetici se obţin programe mai rapide.

Exemplul 4: În biblioteca **ctype.h** există funcţiile **toupper()** şi **tolower()** care transformă literele mici în litere mari şi invers. Următorul program cere utilizatorului să introducă un şir de caractere şi apoi afişează şirul introdus mai întâi cu litere mari şi apoi cu litere mici. Versiunea următoare, pentru a accesa caracterele din şir, indexează numele tabloului.

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
void main()
{  char str[80];
    int i;
    clrscr();
    printf("Introduceti un sir : ");
    gets(sir);
    for(i=0;str[i];i++) str[i]=toupper(str[i]);
    puts(sir); /* afişarea cu litere mari */
    for(i=0;sir[i];i++) str[i]=tolower(str[i]);
    puts(sir);   /* afisarea cu litere mici */
    getch();
}
```

Exemplul 5: În varianta următoare este folosit un pointer pentru a accesa caracterele şirului şi este preferată de programatorii profesionişti deoarece *incrementarea unui pointer este mai rapidă decât indexarea unui vector*.

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
void main()
{   char str[80], *p;
    int i;
    clrscr(); printf("introduceti sirul: "); gets(str);
    p=str;
    while(*p) *p++=toupper(*p);
```

```
    puts(sir);  /* afişarea cu litere mari */
    p=str;
    while(*p) *p++=tolower(*p);
    puts(str);   /* afişarea cu litere mici */
    getch();
}
```

Exemplul 6: Un pointer poate fi şi decrementat. Următorul program foloseşte un pointer pentru a copia conţinutul unui şir în alt şir, în ordine inversă:

```
#include<stdio.h>
#include<string.h>
char str1[]="Pointerii sunt foarte utili";
void main()
{   char str2[80],*p1,*p2;
    /* p1 va puncta sfârşitul şirului str1 */
    p1=str1+strlen(str1)-1;
    p2=str2;      /* începutul şirului str2 */
    while(p1>=str1) *p2++=*p1--;
    *p2='\0';     /* caracterul null incheie sirul str2 */
    printf("%s  %s",str1,str2);
    getch();
}
```

Exemplul 7: Limbajul C permite utilizarea şirurilor constante (şiruri de caractere scrise între ghilimele). Când compilatorul întâlneşte un astfel de şir, îl depune în tabela de şiruri a programului şi generează un pointer către el. De exemplu, programul următor citeşte şiruri până când este tastat şirul **stop** :

```
#include<stdio.h>
#include<sting.h>
char *p="stop";
void main()
{
     char str[80];
     do{
            printf("introduceti un sir : ");
            gets(str);
     }while(strcmp(p,str));
}
```

Folosirea pointerilor către şirurile constante poate fi foarte utilă când aceste constante sunt foarte lungi. De exemplu, să presupunem că, în anumite puncte ale sale, un program afişează un mesaj. Pentru a tasta cât mai puţin, trebuie aleasă varianta de a iniţializa un pointer către un şir constant şi apoi, când trebuie afişat mesajul, să fie folosit acest pointer.

```
char *p="Insert disk into drive A then press ENTER";
…..
printf(p);
……
printf(p);
….
```

Un alt avantaj al acestei metode constă în faptul că, dacă mesajul trebuie modificat, schimbarea trebuie făcută o singură dată şi toate referirile la mesaj vor reflecta modificarea făcută.

Exemplul 8: Funcţia **gets()** citeşte caracterele introduse de la tastatură până când este apăsată tasta **Enter**. Dacă operaţia se termină cu succes, **gets()** returnează un pointer către începutul şirului. În cazul unei erori este returnat un pointer **null**. Programul următor arată cum poate fi folosit pointerul returnat de **gets()** pentru a accesa un şir care conţine informaţii de intrare. Înainte de a folosi şirul, programul verifică dacă nu au apărut erori de citire.

```
#include<stdio.h>
void main()
{    char str[80],*p;
     printf("Introduceti un sir :");
     p=gets(str);
     /* dacă p nu este null se afişează şirul */
    if(p)printf("%s  %s",p,str);
```

```
}
```

Dacă doriţi să fiţi siguri că funcţia **gets()** nu a lucrat cu erori, o puteţi plasa direct în instrucţiunea **if** ca în exemplul următor :

```
#include<stdio.h>
void main()
{   char str[80];
    printf("Introduceti un sir : ");
    /* dacă pointerul către începutul şirului nu este null se produce afişarea */
    if(gets(str)) printf("%s",str);
}
```

7.4 Exerciţii şi teste grilă

1. Fie declaraţia: **`int var,*pointer;`**. Verificaţi dacă expresia: **`(&pointer)`** are aceeaşi semnificaţie cu **`pointer`** şi **`&(var)`** are aceeaşi semnificaţie cu **`var`**. Cum explicaţi aceasta ?

2. Fie declaraţia: **`double d,*pd;`**. Să se decidă dacă variabilele **`d`** şi **`pd`** au aceeaşi dimensiune.

3. Considerăm declaraţiile :

```
int n, const int a=10 , *pci=&a;
```

Să se determine care dintre instrucţiuni sunt corecte :

```
n=a;  a=2;
n=*pci;  *pci=1;
pci++;
```

4. Fie declaraţiile :

```
int n=10;  const *cpi=&n;
```

Să se determine corectitudinea instrucţiunilor următoare :

```
*cpi=1;  cpi=&n;
cpi++;  n=cpi;
```

5. Precizaţi ce tipăreşte programul următor :

```
#include<stdio.h>
long
a[10]={10,11,12,13,14,15,16,17,
                        18,19};
void main()
{  long *pi;
   for(pi=&a[0]; pi<&a[10]; pi++)
       printf("\n%p:%ld", pi, *pi);
}
```

6. Precizaţi ce face programul următor:

```
#include<stdio.h>
int a[10]={10,11,12,13,14,15,16,17,
                              18,19};
void main()
{  int *pi;
   for(pi=a;pi<a+10;)
      printf("\n%p:%d",pi,++*pi++);
}
```

7. Fie declaraţia : **`int a[10][10];`**
Care din expresiile următoare sunt echivalente?

```
a) *a[i]    b) **(a+i)   c) *(*a+i)
d) a[0][i]  e) a[i][0]
```

8. Este corectă secvenţa **`char*s;gets(s);`** ?

9. Explicaţi care din instrucţiuni sunt greşite în secvenţa următoare :

```
char *s="Test C";
*s="Ansi C";
s="Ansi C";
```

10. Care dintre următoarele variante reprezintă o declaraţie corectă a unei variabile **x** de tipul "adresă a unei variabile întregi" ?

```
a) int x*;  b) int *x;  c) int x;
d) int &x;  e) int x&;
```

11. Se consideră declaraţia de variabile:

```
int m, *x,*y;
```

Care dintre următoarele atribuiri sunt corecte ?

```
a) x=m;     b) *x=*m;   c) *y=*x;
d) y=&m;    e) y=x;
```

12. Fie declaraţiile de variabile:

```
int a=2,b;  int *x,*y;
```

Precizaţi ce valori se vor afişa, în ordine, în urma execuţiei secvenţei de program de mai jos:

```
x=&a; a=5; printf("%d", *x);
b=a-2; y=&b; b+=(*y)+4;
printf("%d", b);
*y=*x; printf("%d", *y);
if(x==y) putchar('1');
else putchar('0');
```

```
a) 2,10,2,1 b) 2,10,2,0 c) 5,7,5,0
d) 5,10,5,0 e) 5,10,5,1
```

13. Se consideră următoarea secvenţă de program:

```
int *q,**p,a=5,b=3;
*p=&a;                // (1)
```

```
q=*p;                  // (2)
b+=*(&(**p));          // (3)
printf("%d %d",*q,b);
```

Ce puteţi spune despre atribuirile (1), (2) şi (3)?

a) nici una dintre atribuiri nu este corectă
b) numai atribuirea (1) este `corectă
c) numai atribuirile (1) şi (2) sunt corecte
d) toate sunt corecte şi secvenţa afişează de două ori numărul 5
e) toate atribuirile sunt corecte şi secvenţa afişează numerele 5 şi 8

14. Fie atribuirea : `*y=&(*(&z));` Cum trebuie scrise corect declaraţiile de variabile, astfel încât atribuirea să fie corectă ?

a) int *y,z; b) int y,*z;
c) int y,z d) int **y,z;**
e) int **y,*z;

15. Care dintre instrucţiunile (I),(II),(III),(IV) din programul următor sunt eronate? Precizaţi valorile obţinute în cazul instrucţiunilor corecte.

```
#include<stdio.h>
void main()
{
      const int x=3; int u,v;
      x=4;                  // (I)
      *(int*)&x=8;          // (II)
      u=x;                  // (III)
      v=*(int*)&x;          // (IV)
}
```

a) I b) II c) III
d) IV e) nici una

16. Alegeţi atribuirea corectă din programul de mai jos:

```
void main()
{
      int a;   void *p;
      p=(int*)&a;         // (I)
      p=&a;               // (II)
      p=(float*)&a;       // (III)
      p=&(int*)a;         // (IV)
}
```

a) I b) II c) III
d) IV e) nici una

17. Fie declaraţiile de variabile:

```
int a=2,b,c=5;  int *x,*y;
```

Precizaţi ce valori se vor afişa, în ordine, în urma execuţiei secvenţei de program de mai jos:

```
x=&c; a+=*x; printf("%d",a);
b=++a; y=&b; printf("%d",*y);
 x=y;  printf("%d",(*x)++;
```

a) 7,7,7 b) 7,8,9 c) 7,8,8
d) 7,7,8 e) 8,8,9

18. Fie un pointer **x** către întreg. Care dintre instrucţiunile de ma jos realizează corect alocarea dinamică a memoriei ?

```
a) x=(int)malloc(sizeof(int*));
b) x=(int*)malloc(sizeof(int*));
c) x=(int*)malloc(sizeof(int));
d) *x=(int*)malloc(sizeof(int));
e) *x=(int)malloc(sizeof(int*));
```

19. Fie următoarele declaraţii de variabile:

```
int **a,*b,c;
```

Care dintre expresiile de mai jos vor genera eroare la execuţie?

a) a=&(&c); b) b=&(a);**
c) *a=&c; d) **a=&b;
e) *b=a+c;**

20. Considerăm declaraţia: `int **p;`
şi atribuirea `p=&q;` Alegeţi varianta potrivită astfel încât atribuirea să aibă sens.

a) int q; b) int *q;
c) int ***q; d) int &q;
e) nici una

21. Precizaţi valoarea variabilei **a** ca urmare a execuţiei programului următor:

```
void main()
{
      int a; char b=1;
      a=*(int*)&b;
}
```

a) 1 b) 97 c) neprecizată
d) nici una e) programul este greşit

22. Precizaţi care dintre instrucţiunile de atribuire de mai jos face ca **x** să primească valoarea 0:

```
void main()
{
      int a=1,b=2;  float x;
      x=a/ *&b;          // (I)
      x=(float) a/b;     // (II)
}
```

a) I b) II c) ambele
d) nici una e) programul este gresit

23. Care dintre instrucţiunile de tipărire vor afişa aceeaşi valoare ?

```
#include<stdio.h>
void main()
{
      int a=2,*p=&a;
      printf("%d\n",*p+1);
      printf("%d\n",*&p+1);
      printf("%d\n",*(p+1));
      printf("%d\n",*(&p+1));
```

```
}
```
a) prima şi a doua
b) a doua şi a treia
c) a doua şi a patra d) nici una
e) programul este eronat

24. În programul următor, care dintre cele patru instrucţiuni va tipări valoarea 11?

```
#include<stdio.h>
void main()
{
      const int x=2,y=3;
      *(int*)&x=8;
      *(int*)&y=9;
      printf(„%d\n”,x+y);
      printf(„%d\n”,*(int*)&x+y;
      printf(„%d\n”,x+*(int*)&y;
      printf(„%d\n”,*(int*)&x+
                        *(int*)&y;
}
```
a) prima b) a doua c) a treia
d) a patra e) nici una

25. Fie programul următor:

```
#include<stdio.h>
void main()
{
      int m[9],i;
      for(i=0;i<9;i++) m[i]=i;
      while(i>0)
      {   i--; *(m+i)=-i;  }
}
```
Care dintre afirmaţiile de mai jos sunt adevărate ?
a) ambele cicluri sunt greşite
b) numai primul ciclu este corect
c) numai al doilea ciclu este corect
d) ambele cicluri sunt corecte
e) în cele două cicluri, elementele vectorului vor primi valori egale în modul, dar de semne opuse

26. Se consideră programul următor:

```
#include<stdio.h>
void main()
{
  int a=5,b=-12,c=7,*v[3];
  v[0]=&a;                    //(1)
  printf("%d\n",*v[0]);       //(2)
  *(v+1)=&b;                  //(3)
  printf("%d\n",*(*(v+1)));   //(4)
  2[v]=&c;                    //(5)
  printf("%d\n",*v[2]);       //(6)
}
```
a) declaraţia vectorului este eronată
b) atribuirile (1), (3) şi (5) sunt toate corecte
c) atribuirea (1) este corectă, iar (3) şi (5) sunt eronate
d) atribuirile (1) şi (3) sunt corecte, iar (5) este eronată
e) programul este corect şi afişează valorile 5, -12, 7

27. Ce va afişa programul următor?

```
#include<stdio.h>
void main()
{
      int (*v)[3];
      int u[]={10,11,12};
      v=&u;
      printf("%d",(*v)[1];
}
```
a) programul este eronat
b) o adresă de memorie oarecare, fără nici-o semnificaţie
c) valoarea întreagă 11
d) adresa de memorie la care se află valoarea întreagă 11
e) adresa începând cu care se găseşte vectorul **v** în memorie

28. Se consideră următoarea secvenţă de program:

```
int a[9][11],i,j;
for(i=0;i<9;i++)
   for(j=0;j<11;j++)
       if(i==j) (*(a+i)) [j]=0;
       else *(*(a+i)+j)=i*j;
```
Precizaţi care dintre afirmaţiile de mai jos sunt false:
a) **a[5][2]** este 10
b) **a[8][0]** este 6
c) ***(*(a+3)+3)** este 0
d) programul conţine erori de sintaxă
e) matricea **a** este simetrică faţă de diagonala principală

29. Se consideră următoarele declaraţii de variabile:

```
int q=6,d[3][4],(e[3])[4],v[4];
int *a[3][4],(*b)[3][4],
    (*c[3])[4];
```
Care dintre atribuirile de mai jos sunt corecte?

```
a) d[0][2]=e[1][3];
b) a[2][3]=&q;
c) b=&d;          d) c[2]=&v;
e) toate atribuirile anterioare
```

30. Precizaţi ce valoare va afişa programul următor:

```
#include<stdio.h>
void main()
{
      int a[20][20],i,j,n=4;
      for(i=0;i<n;i++)
```

```
         for(j=0;j<n;j++)
            *(*(a+i)+j)=(i>j)?
                  (j-i) : (j+i);
      int m=10;
      for(i=0;i<n;i++)
         for(j=0;j<n;j++)
             if(m>(*(a+i))[j])
                      m=a[i][j];
      printf("%d",m);
}
```

a) 10 b) 6 c) 3 d) −3
e) programul este eronat

31. Fie vectorul **y** cu patru elemente numere întregi:

```
      int y[4]={0,1,2,3};
```

Care dintre următoarele instrucţiuni declară şi iniţializează corect un pointer **ptr** către vectorul **y**?

a) `int *(ptr[4])=&y;`
b) `int (ptr*)[4]=&y;`
c) `int (*ptr)[4]=&y;`
d) `int ptr*[4]=&y;`
e) `int *ptr[4]=&y;`

32. Fie următorul program:

```
#include<stdio.h>
void main()
{
      int u[4]={1,2,3,4},
          v[4]={5,6,7,8},
          w[4]={0,0,0,0}, i;
      int (*x)[4]=&u, (*y)[4]=&v,
          (*z)[4]=&w;
      for(i=0;i<4;i++)
        printf("%3d",(*z)[i]=
                  (*x)[i]+(*y)[i]);
}
```

Care dintre afirmaţiile de mai jos sunt adevărate?

a) programul va afişa patru adrese de memorie
b) programul va afişa, în ordine, valorile 6,8,10,12
c) valoarea lui **(*x)[2]** este 3
d) valoarea lui **(*y)[4]** este 8
e) instrucţiunea de afişare din ciclu este eronată, din cauza folosirii operatorului de atribuire în funcţia **printf**

33. Fie următorul program:

```
#include<stdio.h>
void main()
{
      int x[4]={1,2,3},
          y[4]={4,5,6,7},z[7];
      int i,j;
      for(i=0;i<4;i++)
                *(z+i)=*(y+i);
      for(j=0;j<3;j++)
                *(z+i+j)=*(x+j);
      for(i=0;i<7;i++)
          printf("%d",*(z+i));
}
```

Care vor fi valorile afişate în urma execuţiei sale?

a) 1,2,3,4,5,6,7 b) 7,6,5,4,3,2,1
c) 3,2,1,7,6,5,4 d) 4,5,6,7,1,2,3
e) programul este eronat

34. Fie secvenţa de instrucţiuni:

```
   int x[]={10,20,30,40,50};
   int*ptr=x;
   printf("%d\n",*(ptr+2));
   printf("%d\n",*(ptr)+2);
```

Ce se va tipări după executarea codului de mai sus?

a) 30 30 b) 30 12
c) 12 12 d) 12 30

35. Fie secvenţa de instrucţiuni:

```
   int *array[3];
   int(*ptr)[]=array;
   int x=2,y=3,z=4;
```

Având în vedere codul de mai sus, cum veţi realiza atribuirea celui de al doilea pointer din şirul "**ptr**" ca să pointeze la valoarea lui **y** ?

a) `ptr[2]=&y;`
b) `(*ptr)[1]=y;`
c) `(*ptr)[1]=&y;`
d) `(*ptr)[2]=&y;`

36. Fie următoarea declaraţie de variabile :

```
      int *p;  int x,y;
```

Atribuirea `y=x+100;` este echivalentă cu secvenţa:

a) `p=&x; y=*p+100;`
b) `y=*p+100; p=&x;`
c) `p=&y; y=*p+100;`
d) `p=&x; y=&p+100;`

37. Fie următoarea declaraţie de variabile :

```
      int *p;  int x,y;
```

Atribuirea `x=y;` este echivalentă cu secvenţa :

a) `p=&x; *p=y;` **b)** `p=&y; *p=x;`
c) `*p=x; p=&y;` **d)** `*p=y; p=&x;`

38. Fie următoarea declaraţie de variabile :

```
      int *p;  int x,y;
```

Instrucţiunea `x++;` este echivalentă cu secvenţa :

a) `p=&x; (*p)++;` **b)** `p=*x; (&p)++;`
c) `p=&x; *(p++);` **d)** `p=&x; *p++;`

39. Fie următoarea declaraţie de variabile :

```
      int *p;  int x,y;  p=&x;
```

Atribuirea `y=x*(x+1);` este echivalentă cu secvenţa :

a) y=*p*((*p)++); b) y=*p*(*p++);
c) y=*pp++; d) y=(*p)*(*p++);**

40. Fie următoarea declaraţie de variabile :
int *p; int x=100,y; p=&x;
În urma atribuirii **y=*p*((*p)++);** **y** va avea valoarea :
a) 10100 b) 11000 c) 10001
d) 10000

41. Fie următoarea declaraţie de variabile :
int *p; int x=100, y; p=&x;
În urma atribuirii **y=*p+((*p)++);** **y** va avea valoarea :
a) 201 b) 102 c) 200 d) 202

42. Fie secvenţa :
```
int t[5]={20,30,40,50,10};
int *p;  int x;
```
Atribuirea **x=t[3];** este echivalentă cu :
a) p=t; x=*(p+3);
b) p=&t[0]; x=*(p+2);
c) p=*t; x=*p+3;
d) p=t; x=*p+3;

43. Fie secvenţa :
```
int t[5]={20,30,40,50,10};
int *p;  int x;
```
Atribuirea **x=*(&t[0]+3);** este echivalentă cu :
a) x=t[3]; b) x=t[4];
c) x=*(&t[2]); d) x=*(t+4);

44. Se consideră secvenţa de program :
```
void main(void)
{  int *p, *q;
    p=(int*)malloc(sizeof(int));
    q=(int*)malloc(sizeof(int));
    *p=5; *q=3; *p=*q;
    if(p==q) *p+=1;
    printf("%d",*p);
}
```
Care este valoarea afişată pentru **p** :
a) 5 b) 3 c)6 d)4

45. Se dă următoarea secventă de cod :
```
int a[5]={1,2,3,4,5};
int *aPtr;
aPtr=a;
printf("element=%d\n",*(aPtr+2));
```
Ce va afişa codul de mai sus după execuţie ?
a) element=1 b) element=2
c) element=3 d) element=4

46. Se dă codul:
```
int *ptr; int y[10];
int i;
for(i=0;i<10;i++) y[i]=i;
ptr=y; ptr+=8;
printf("ptr=%d\n", *ptr);
```
Ce se va afişa când codul este executat ?
a) ptr=0 b) ptr=9
c) ptr=7 d) ptr=8

Cap.8 Şiruri de caractere

8.1 Folosirea şirurilor

Cea mai comună utilizare a tabloului unidimensional în limbajul C este şirul (de caractere). Spre deosebire de multe alte limbaje de programare, C nu conţine tipul de dată **string**. În schimb, C permite utilizarea şirurilor folosind tablouri unidimensionale de tip **char**. Şirul este definit ca fiind *un tablou de caractere terminat prin caracterul null ('\0', codul ASCII 0)*. Faptul că şirul trebuie terminat prin caracterul *null* înseamnă că tabloul trebuie astfel definit încât să poată conţine un şir cu un octet mai lung decât cel mai lung şir ce va fi reprezentat vreodată în acest tablou, pentru a face loc caracterului *null*. **Un şir constant este automat terminat prin caracterul null de către compilator**.

Pentru a citi un şir de la tastatură, trebuie apelată funcţia **gets()**, care reclamă includerea header-ului **stdio.h**. Funcţia **gets()** trebuie apelată folosind numele tabloului de caractere fără nici un index. Funcţia **gets()** citeşte caractere până când este apăsată tasta **ENTER**. Tasta **ENTER** este înlocuită de compilator cu caracterul *null* care termină şirul. De exemplu, programul următor citeşte şi afişează un şir introdus de la tastatură :

```
#include<stdio.h>
void main()
{  char str[80];
   int j;
   printf("Introduceti un sir(<de 80 de caractere):\n");
   gets(str);
   for(j=0;str[j];j++)
        printf("%c",str[j]);
}
```

Să remarcăm că programul, pentru a controla ciclul care afişează şirul, utilizează faptul că *null* este *false*(0). Funcţia **gets()** nu efectuează verificări asupra limitelor în care variază indexul tabloului, aşa că este posibil ca utilizatorul să introducă mai multe caractere decât poate conţine şirul. Deci trebuie să fiţi siguri că aţi apelat funcţia **gets()** cu un tablou suficient de mare, care să poată conţine şi caracterele neaşteptate. Există un mod mult mai simplu de a afişa şirurile, folosind funcţia **printf()** sau **puts()** . Programul anterior, rescris cu ajutorul acestei funcţii este :

```
#include<stdio.h>
void main()
{  char str[80];
   int j;
   printf("Introduceti un sir (< de 80 de caractere):\n");
   gets(str);
   printf("%s",str);      /* puts(sir); */
}
```

Dacă după afişarea şirului se doreşte trecerea la o linie nouă, se poate afişa **str** după cum urmează :

```
printf("%s\n",str);
```

Această metodă foloseşte specificatorul de format **%s** urmat de caracterul linie nouă şi utilizează tabloul de caractere ca al doilea argument al funcţiei **printf()**.

8.2 Tablouri de şiruri

Tablourile de şiruri, numite deseori şi tabele de şiruri, sunt foarte des folosite în C. O tabelă bidimensională de şiruri se poate crea ca oricare alt tablou bidimensional. Totuşi modul în care trebuie gândit un tablou de şiruri este puţin diferit. Fie următoarea declaraţie : `char names[10][40];`
Această declaraţie specifică o tabelă care conţine 10 şiruri, fiecare având lungimea de până la 40 de caractere. Pentru a accesa un şir din tabelă, trebuie specificat numai primul index. De exemplu, pentru a citi de la tastatură al treilea şir al tabelei, trebuie folosită instrucţiunea : **gets(names[2]);**. Pentru a afişa primul şir al tabelei trebuie folosită instrucţiunea : **printf(name[0]);** .

8.3 Funcţii standard pentru prelucrarea şirurilor de caractere

În legătură cu şirurile de caractere se au în vedere operaţii de felul următor:

- calculul lungimii şirurilor de caractere
- copierea şirurilor de caractere
- concatenarea şirurilor de caractere
- compararea şirurilor de caractere
- căutarea în şiruri de caractere

Funcţiile standard prin care se realizează aceste operaţii au fiecare un nume care începe cu prefixul **str** şi au prototipul în fişierul header **string.h**.

8.3.1 Lungimea unui şir de caractere

Lungimea unui şir de caractere se defineşte prin numărul de caractere proprii care intră în compunerea şirului respectiv. Caracterul **NUL** este un caracter impropriu şi el nu este considerat la determinarea lungimii unui şir de caractere. Prezenţa lui este însă necesară, deoarece la determinarea lungimii unui şir se numără caracterele acestuia până la întâlnirea caracterului **NUL**. Funcţia pentru determinarea lungimii unui şir de caractere are prototipul : **unsigned strlen(const char *s);**

Exemplul 1:

```
char *const p="Acesta este un sir";
unsigned n;
…..
n=strlen(p);
```

Lui **n** i se atribuie valoarea 18, numărul caracterelor proprii ale şirului spre care pointează **p**.

Exemplul 2:

```
char tab[]="Acesta este un sir";
int n;
n=strlen(tab);
```

Variabila **n** primeste aceeaşi valoare ca în exemplul precedent.

Exemplul 3:

```
int n;
n=strlen("Acesta este un sir");
```

Observaţie : Parametrul funcţiei **strlen** este un pointer spre un şir constant deoarece funcţia **strlen** nu are voie să modifice caracterele şirului pentru care determină lungimea.

8.3.2 Copierea unui şir de caractere

Adesea este nevoie să se copie un şir de caractere din zona de memorie în care se află, în altă zonă. În acest scop se poate folosi funcţia : **char *strcpy(char *dest, const char *sursa);**

Funcţia copiază şirul de caractere spre care pointează ***sursa*** în zona de memorie a cărei adresă de început este valoarea lui ***dest***. Funcţia copiază atât caracterele proprii şirului, cât şi caracterul **NUL** de la sfârşitul şirului respectiv. Se presupune că zona de memorie în care se face copierea este destul de mare pentru a putea păstra caracterele copiate. În caz contrar se alterează datele păstrate imediat după zona rezervată la adresa definită de parametrul ***dest***. La revenire, funcţia returnează adresa de început a zonei în care s-a transferat şirul, adică chiar valoarea lui ***dest***. Această valoare este pointer spre caractere, deci tipul returnat de funcţie este : **char***. Se observă că parametrul ***sursa***, care defineşte zona în care se află şirul ce se copiază, este declarat prin modificatorul **const**. Aceasta deoarece funcţia **strcpy** nu are voie să modifice şirul care se copiază. În schimb, parametrul ***dest*** nu este declarat cu parametrul **const** deoarece funcţia **strcpy** modifică zona spre care pointează ***dest*** (în ea se copiază caracterele şirului).

Exemplul 1:

```
char tab[]="Acest sir se copiaza";
char t[sizeof(tab)];    /* are acelaşi număr de elemente ca şi tab */
.....
strcpy(t,tab);          /* şirul din tab se copiază în zona alocată lui t */
```

Exemplul 2:
```
char t[100];
strcpy(t,"Acest sir se copiaza");
```
Exemplul 3:
```
char *p="Acest sir se copiaza";
char t[100];
char *q;
q=strcpy(t,p);
```
Şirul păstrat în zona spre care pointează **p** se transferă în zona spre care pointează **t**. Valoarea lui **t** se atribuie lui **q**.

Pentru a **copia cel mult n caractere** ale unui şir dintr-o zonă de memorie în alta, se va folosi funcţia de prototip : **char *strncpy(char *dest , const char *sursa, unsigned n);**
Dacă **n>lungimea şirului** spre care pointează **sursa**, atunci toate caracterele şirului respectiv se transferă în zona spre care pointează **dest**. Altfel se copiază numai primele **n** caractere ale şirului. În rest, funcţia **strncpy** are acelaşi efect ca şi **strcpy**.
Exemplu:
```
char *p="Acest sir se copiaza trunchiat";
char t[10];
strncpy(t,p,sizeof(t));   /* se vor copia numai primele 10 caractere*/
```

Funcţia **strdup** copiază un şir într-o locaţie nou creată şi are prototipul : **char *strdup(consr char *s);**
Funcţia **strdup** face copierea unui şir în spaţiul obţinut prin apelul funcţiilor **calloc** sau **malloc**. Nu este deci necesară o apelare explicită a funcţiei de alocare. Dimensiunea spaţiului alocat este egală cu **lungimea şirului+1** (caracterul '\0'). Utilizatorul este responsabil pentru eliberarea spaţiului atunci când nu mai este nevoie de şir. Valoarea întoarsă este pointerul la şir în caz de succes sau pointerul **NULL** când spaţiul nu s-a putut aloca.
Exemplu :
```
#include<stdio.h>
#include<string.h>
#include<malloc.h>
void main()
{   char *dup_str , *string="Acest sir se copie";
    dup_str=strdup(string);
    printf("%s\n",dup_str);
    free(dup_str);
}
```
Observăm că nu a fost necesar apelul unei funcţii de alocare prealabilă a memoriei. Apelul funcţiei de eliberare, după ce şirul nu a mai fost utilizat, este necesar.

8.3.3 Concatenarea şirurilor de caractere

Bibliotecile limbajelor C şi C++ conţin mai multe funcţii care permit concatenarea unui şir de caractere la sfârşitul unui alt şir de caractere. Una dintre ele are prototipul :

char *strcat(char *dest, const char *sursa);

Această funcţie copiază şirul de caractere din zona spre care pointează ***sursa***, în zona de memorie spre care pointează ***dest***, imediat după ultimul caracter propriu al acestui şir. Se presupune că zona spre care pointează ***dest*** este suficientă pentru a păstra caracterele proprii celor două şiruri care se concatenează, plus caracterul **NUL** care termină şirul rezultat în urma concatenării. Funcţia returnează valoarea lui ***dest***.
Exemplu:
```
char tab1[100]="Limbajul C++";
char tab2[]="este C incrementat";
strcat(tab1,"  ");   /* se adaugă spaţiu după ultimul caracter + */
strcat(tab1,tab2);
```
Observaţie : Funcţia **strcat**, la fel ca funcţia **strcpy**, nu trebuie să modifice şirul de caractere spre care pointează ***sursa***.

O altă funcţie de bibliotecă utilizată la concatenarea de şiruri este funcţia **strncat** care are prototipul : **char *strncat(char *dest, const char *sursa, unsigned n);**

În acest caz se concatenează, la sfârşitul şirului spre care pointează ***dest***, cel mult **n** caractere ale şirului spre care pointează ***dest***. Dacă **n >lungimea** şirului spre care pointează ***sursa***, atunci se concatenează întregul şir, altfel numai primele **n** caractere.

Exemplu:

```
char tab1[100]="Limbajul E este mai bun decat ";
char tab2[]="limbajul C++ care este un superset a lui C";
strncat(tab1,tab2,12);
```

După revenirea din funcţie, tabloul **tab1** conţine succesiunea de caractere "*Limbajul E este mai bun decât limbajul C++*" .

8.3.4 Compararea şirurilor de caractere

Şirurile de caractere se pot compara folosind codurile ASCII ale caracterelor din compunerea lor. Fie **s1** şi **s2** două tablouri unidimensionale de tip caracter folosite pentru a păstra, fiecare, câte un şir de caractere.

Şirurile păstrate în aceste tablouri sunt *egale* dacă au lungimi egale şi **s1[j]=s2[j]** pentru toate valorile lui **j**.

Şirul **s1** este *mai mic* decât **s2**, dacă există un indice **j** astfel încât **s1[j]<s2[j]** şi **s1[k]=s2[k]** pentru **k=0,1,…,j-1**.

Şirul **s1** este *mai mare* decât **s2**, dacă există un indice **j** astfel încât **s1[j]>s2[j]** şi **s1[k]=s2[k]** pentru **k=0,1,…,j-1**.

Compararea şirurilor de caractere se poate realiza folosind funcţii standard de felul celor de mai jos. O funcţie utilizată frecvent în compararea şirurilor este cea de prototip :

int strcmp(const char *s1, const char *s2);

Notăm cu **sir1** şirul de caractere spre care pointează **s1** şi cu **şir2** şirul de caractere spre care pointează **s2**. Funcţia **strcmp** returnează :

- valoare negativă dacă **sir1<sir2**
- zero dacă **sir1=sir2**
- valoare pozitivă dacă **sir1>sir2**

O altă funcţie pentru compararea şirurilor este funcţia de prototip :

int strncmp(const char *s1, const char *s2, unsigned n);

Această funcţie compară cele două şiruri spre care pointează **s1** şi **s2** utilizând cel mult primele **n** caractere din fiecare şir. În cazul în care minimul dintre lungimile celor două şiruri este mai mic decât **n**, funcţia **strncmp** realizează aceeaşi comparaţie ca şi **strcmp**.

Adesea, la compararea şirurilor de caractere dorim să nu se facă distincţie între literele mici şi mari. Acest lucru este posibil dacă folosim funcţia de prototip :

int stricmp(const char *s1, const char *s2);

Această funcţie returnează aceleaşi valori ca şi funcţia **strcmp**, cu deosebirea că la compararea literelor nu se face disticţie între literele mari şi mici.

Exemple:

```
char *sir1="ABC";
char *sir2="abc";
int j;
```

Apelul : **j=strcmp(sir1, sir2);** returnează o valoare negativă, deoarece literele mari au codurile ASCII mai mici decât literele mici.

Aplelul : **j=stricmp(sir1, sir2);** returnează valoarea 0, deoarece ignorându-se diferenţa dintre literele mari şi mici, cele două şiruri devin egale.

Pentru a limita compararea a două şiruri de caractere la primele cel mult **n** caractere ale lor, la comparare ignorându-se diferenţa dintre literele mici şi mari, se va folosi funcţia de prototip :

int strincmp(const char *s1, const char *s2, unsigned n);

8.3.5 Căutarea în şiruri de caractere

Pentru *căutarea unui caracter într-un şir de caractere* sunt folosite funcţiile de prototip :

char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);

Când caracterul căutat nu se află în şir este întors pointerul **NULL**, altfel se întoarce un pointer la prima (**strchr**), respectiv ultima (**strrchr**) apariţie a caracterului în şirul **s**.

Pentru a *căuta prima apariţie a unui subşir într-un şir* se poate folosi funcţia de prototip :

char *strstr(const char *s1, const char *s2);

Funcţia caută prima apariţie a subşirului **s2** în şirul **s1** şi întoarce un pointer la şirul găsit sau pointerul **NULL** când nu s-a găsit nici o apariţie.

Pentru a *căuta o secvenţă de caractere într-un şir* putem folosi funcţia de prototip:

char *strpbrk(const char *s1, const char *s2);

Funcţia caută în şirul **s1** prima apariţie a unui caracter din şirul **s2**. Valoarea întoarsă este un pointer la prima apariţie sau pointerul **NULL** în cazul în care nici un caracter din **s2** nu apare în **s1**.

Pentru a *verifica apariţia caracterelor unui şir în alt şir* putem folosi funcţiile :

int strcspn(const char *s1, const char *s2);
int strspn(const char *s1, const char *s2);

Funcţia **strcspn** determină lungimea secvenţei de caractere de la începutul şirului **s1** care nu conţine nici un caracter din **s2**. Funcţia **strspn** determină lungimea secvenţei de caractere de la începutul şirului **s1** care conţine numai caractere din **s2**.

Funcţia **strtok** *caută prima parte din şirul **s1** care este diferită de orice subşir din **s2***. Este pus un caracter **NULL** la primul caracter din **s1** comun şirurilor şi se întoarce un pointer la subşirul găsit. Se poziţionează implicit adresa de căutare la adresa următoare caracterului găsit în **s1**. Următoarele apeluri cu primul parametru **NULL** vor continua căutarea de la adresa de căutare setată. Prototipul funcţiei este :

char *strtok(char *s1, const char *s2);

8.4 Exemple de utilizare a funcţiilor standard

1. Fiind dat un cuvânt să se afişeze toate sufixele acestuia. Literele fiecărui sufix vor fi separate prin două spaţii.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{  int n,i,j;
   char si[100];
   clrscr();
   printf("Introduceti cuvantul : "); gets(si);
   n=strlen(si);
   for(i=0;i<n;i++)
   {  for(j=i;j<n;j++)
      printf("%c ",*(si+j));
      puts("\n");
   }
   getch();
}
```

2. Să se determine prima şi ultima apariţie a unui caracter într-un şir de caractere.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{  char s[100],*ptr,c='e';
   clrscr();
   strcpy(s,"Test sir de caractere.");
   ptr=strchr(s,c);
   if(ptr)
   {  printf("Prima aparitie a car. %c este in pozitia
             %d\n",c, ptr-s);
      printf("Ultima aparitie a car.%c este pe pozitia
             %d\n",c, strrchr(s,c)-s);
   }
```

```
      else printf("Caracterul %c nu se afla in sir\n",c);
      getch();
   }
```

3. Programul următor ilustrează modul de folosire a funcţiei **strstr** :

```
   #include<stdio.h>
   #include<conio.h>
   #include<string.h>
   void main()
   {  char *str1="c:\\tc\\include ",*str2="\\include";
      char *ptr;
      clrscr();
      ptr=strstr(str1,str2);
      *ptr='\0';
      printf("Directorul de baza este : %s\n",str1);
      getch();
   }
```

4. Exemplu de folosire a funcţiei **strpbrk** .

```
   #include<stdio.h>
   #include<conio.h>
   #include<string.h>
   void main()
   {  char *string1="abcdefghijklmnopqrstuvwxyz";
      char *string2="12fmcd";
      char *ptr;
      clrscr();
      ptr=strpbrk(string1,string2);
      if(ptr)
         printf("strpbrk a gasit primul caracter : %c\n",
                     *ptr);
      else
         printf("strpbrk nu a  gasit nici un caracter in
                     sir\n");
      getch();
   }
```

5. Programul următor citeşte o secvenţă de caractere şi determină prima frază prin căutarea în şirul **text** a caracterelor din şirul **eop** .

```
   #include<stdio.h>
   #include<string.h>
   #include<conio.h>
   void main()
   {  char text[128];
      char *eop=".!?";
      int length;
      clrscr();
      printf("Introduceti un text : "); gets(text);
      length=strcspn(text,eop);
      text[length]='\0';
      printf("Prima fraza este : %s\n",text);
      getch();
   }
```

6. Programul următor găseşte toate cuvintele dintr-o linie. Presupunem că un cuvânt este delimitat de unul din caracterele : "**!,.:;?-!** " .

```
   #include<stdio.h>
   #include<conio.h>
   #include<string.h>
   void main()
   {  char in[100] , *p;
      clrscr();
      puts("Introduceti linia : "); gets(in);
     /* strtok plasează terminatorul NULL la începutul secvenţei de căutare,
```

```
    la poziţia găsită */
    p=strtok(in, ",.!:;-? ");
    if(p)
      printf("%s\n", p);
  /* următoarele apeluri utilizează pointerul NULL ca prim parametru */
    while(p)
    {  p=strtok(NULL,",.!:;-? ");
       if(p) printf("%s\n", p);
    }
    getch();
}
```

8.5 Funcţii pentru conversii de date

Pentru conversia datelor din format numeric în format şir de caractere şi invers se folosesc două seturi de funcţii şi macrouri. Primul set pentru conversii conţine funcţii şi macrouri care convertesc din format şir de caractere în format numeric (**atoi**, **atof**, **atol**) şi funcţiile inverse acestora de conversie şi macrouri din format numeric în şir de caractere (**itoa**, **ltoa**, **ultoa**).Primele trei macrouri au sintaxa:

int atoi(const char *s);
long atol(const char *s);
double atof(const char *s);

Primul macrou converteşte un şir într-un număr întreg. El întoarce valoarea obţinută în urma conversiei sau valoarea 0 dacă şirul nu poate fi convertit.Analog, macroul **atol** converteşte un şir într-un număr întreg de tip **long**, iar **atof** într-un număr real de tip **float**. În cazul apariţiei caracterelor nepermise, conversia se va opri la primul caracter incorect şi se va returna valoarea citită până atunci.

Exemplu: Ca exemplu de utilizare a funcţiei **atoi**, se consideră un şir care conţine caractere neadmise şi valori depăşind dimensiunea tipului întreg.

```
#include<stdlib.h>
#include<stdio.h>
void main()
{
      int j,n;
      char *s[]={"12345","12345.67","123456.78",
                 "123a45.67","a12345.67", "12345.a67"};
      for(j=0; j<6; j++)
      {  n=atoi(s[j]);
       printf("sir=%s\n val=%d",s[j],n);
    }
}
```

Valorile numerice tipărite sunt, în ordine:

- 12345 - a fost transformat întregul şir
- 12345 - partea zecimală nu a fost luată în consideraţie
- -7616 - numărul obţinut a fost întreg, dar nu s-a încadrat în limitele impuse de tipul **int**
- 123 - s-a citit până la primul caracter care nu era semn sau cifră
- 0 - primul caracter nu a fost cifră şi conversia s-a oprit aici
- 12345 - caracterul “a” nu a avut nici o influenţă fiind după punctul zecimal

Funcţiile de conversie inversă, din număr în şir de caractere , au sintaxele:

char *itoa(int valoare, char *sir, int baza);
char *ltoa)long valoare, char *sir, int baza);
char *ultoa(unsigned long valoare, char *sir, int baza);

Funcţiile întorc valoarea obţinută într-un şir terminat cu ‘\0’ şi stochează rezultatul în şir. Parametrul **valoare** reprezintă numărul întreg care urmează a fi convertit. El poate avea atât valori pozitive cât şi negative în cazul funcţiilor **itoa** sau **ltoa** şi numai valori pozitive pentru funcţia **ultoa**. Parametrul **sir** reprezintă adresa şirului de caractere în care se va obţine rezultatul conversiei. Parametrul **baza** reprezintă baza aleasă pentru conversie. Funcţia **itoa** poate întoarce un şir de până la 17 caractere, iar **ltoa** un şir de până la 33 de caractere (dacă baza are valoarea 2). În caz de succes, funcţiile întorc un pointer la şirul obţinut, în caz de eroare, este întors şirul vid.

Al doilea set de conversie între numere şi şiruri de caractere este mai performant în privinţa detectării erorilor decât primul. Funcţiile de conversie a unui număr real într-un şir de caractere au sintaxa:

char *fcvt(double val, int nr_cif, int *dec, int *semn);
char *ecvt(double val, int nr_cif, int *dec, int *semn);
char *gcvt(double val, int ndec, char *buf);

Parametrul **val** şi **nr_cif** reprezintă valoarea iniţială şi numărul de cifre care se doresc a fi obţinute. Funcţiile vor întoarce valoarea numărului în baza 10 stocat ca şir de caractere. Nu apare punctul zecimal. Pentru a obţine informaţii suplimentare despre conversie, variabilele **semn**, care reprezintă semnul numărului, şi **dec**, care reprezintă numărul de cifre zecimale, vor fi transmise printr-o referinţă la întreg. Pentru **ecvt**, variabila **nr_cif** reprezintă numărul total de cifre, în timp ce pentru **fcvt** variabila **nr_cif** reprezintă numărul de cifre ale părţii zecimale. Funcţia **gcvt** face suprimarea zerourilor inutile. Valoarea întoarsă de către cele trei funcţii este un pointer spre şirul de caractere.

Conversiile de la şiruri de caractere la numere se fac cu funcţiile:

double strtod(const char *s, char **endptr);
long strtol(const char *s, char **endptr, int baza);
long strtoul(const char *s, char **endptr, int baza);

Funcţia **strtod** converteşte un şir într-un număr real dublă precizie. Şirul iniţial **s** trebuie să aibă următoarea formă: **[ws] [s] [ddd] [.] [ddd] [fmt[s]ddd]** în care **ws** reprezintă spaţii, **s** este semnul, **ddd** sunt cifre zecimale, iar **fmt** este **e** sau **E** pentru forma exponenţială. Al doilea parametru **endptr** este un pointer la un şir de caractere şi are rolul de a se poziţiona la ultima poziţie de citire corectă din şir, pentru detectarea erorilor la conversie.

Exemplu:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
      char sir[80],*s;  double valoare;
      printf("numarul="); gets(sir);
      valoare=strtod(sir, &s);
      printf("sirul:%s   numarul:%lf\n",sir,valoare);
}
```

Funcţiile **strtol** şi **strtoul** convertesc un şir **s** care exprimă un număr într-o bază precizată de al treilea argument, **baza**, într-un număr întreg de tip **long**, respectiv de tip **unsigned long**. Este folosit deasemenea un pointer dublu la caractere pentru detectarea erorilor.

8.6 Exerciţii şi teste grilă

1. În programul următor, care dintre secvenţele de instrucţiuni (I), (II) realizează corect citirea unui şir de caractere de la tastatură şi afişarea acestuia ?

```
#include<stdio.h>
void main()
{
      char s1[10],s2[10];
      scanf("%s",s1);
      printf("s1=%s",s1);  //(I)
      scanf("%s",&s2);
      printf("%s",s2[10]); //(II)
}
```

a) numai (I) b) numai (II)
c) (I) şi (II) d) nici una

2. Analizaţi programul următor şi alegeţi răspunsul corect:

```
#include<stdio.h>
void main()
{
      char b[11],
           a[11]="abcdefghij";
      int i=0;
      while(a[i]%2)  b[i++]=a[i];
      b[i]=0;
}
```

a) programul are erori

b) şirul **b** conţine numai caracterul "a"

c) în urma execuţiei şirurile **a** şi **b** concid

d) şirul **b** conţine numai caracterele din şirul **a** ale căror coduri ASCII sunt numere pare

e) şirul **b** conţine numai caracterele de rang par din **a** (al doilea, al patrulea etc.)

3. Fie programul următor:

```
#include<stdio.h>
#include<string.h>
void main()
{
      char s1[10], s2[10],
           s3[10]="SB";
      gets(s1); gets(s2);
      puts(s1+s2);          // (1)
      if(strlen(s1)<strlen(s2))
           putchar('1');  // (2)
      if(s1>s3) putchar('1');
      else putchar('0');  // (3)
}
```

Presupunem că, în timpul execuţiei programului, se introduc de la tastatură şirurile **s1="BR"** şi **s2="122035"**. Precizaţi dacă sunt adevărate situaţiile de mai jos:

a) citirea de la tastatură este eronată
b) instrucţiunea (1) va afişa textul "BR122035"
c) instrucţiunea (2) va afişa valoarea 1
d) în linia (3) se compară şirurile **s1** şi **s3** din punct de vedere alfabetic
e) nici una dintre afirmaţiile anterioare

4. Pentru programul următor precizaţi care dintre afirmaţiile de mai jos sunt adevărate:

```
#include<stdio.h>
#include<string.h>
void main()
{
      char s[10]="-2B.2A5";
      int j,nr=0;
      for(j=0;j<strlen(s);j++)
         if(!(s[j]>='0'&&
                     s[j]<='9'))
         {  s[j]='0'; nr++; }
      printf("%d%s",nr,s);
}
```

a) declararea şirului este corectă
b) în ciclul **for** sunt parcurse corect caracterele şirului **s**
c) în ciclul **for** sunt înlocuite cu "0" cifrele din **s**
d) condiţia din linia **if** este eronată
e) programul afişează **40200205**

5. Ne propunem să definim un vector care să aibă două elemente, ambele de tip şir de caractere. Fiecare şir trebuie să conţină două caractere, primul "**ab**", iar al doilea "**cd**". Scrieţi declaraţia corectă.

```
a) char a[2][3]={"ab","cd"};
b) char a[2][2]={"ab","cd"};
c) char a[3][2]={"ab","cd"};
d) char a[3][3]={"ab","cd"};
e) char a[][3]={"ab","cd"};
```

6. Care dintre variantele de mai jos reprezintă o declaraţie corectă a unui şir de caractere?

```
a) char s[2];      b) char *s[20];
c) char *s;        d) char s;
e) char s[];
```

7. Pentru programul următor, analizaţi corectitudinea afirmaţiilor de mai jos:

```
#include<stdlib.h>
#include<stdio.h>
void main()
{
      char s1[4],s2[4];  long x;
      gets(s1); gets(s2);
      if(strcmp(s1,s2)<0)
                        x=atol(s1);
      else if(!strcmp(s1,s2)) x=0;
           else x=atol(s2);
      printf("%ld", x);
}
```

a) condiţiile din cele două linii **if** sunt greşite
b) apelurile funcţiei **atol** sunt corecte
c) dacă de la tastatură se introduc şirurile "98" şi "123", atunci se va afişa 98
d) dacă de la tastatură se introduc şirurile "123" şi "135", atunci programul va afişa şirul "123"
e) dacă de la tastatură se introduc şirurile "ab" şi "ac", atunci se va semnala un mesaj de eroare

8. Ştiind că, în conformitate cu standardul ASCII, codurile literelor mari sunt succesive începând cu 65, ce va afişa programul de mai jos?

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
void main()
{
      int x=20,e;
      char s[15]="ABC",t[15],u[15];
      e=s[1]+s[2];
      itoa(e,t,10);
      strcpy(u,t);
      strcat(s,u);
      puts(s);
}
```

a) nimic, şirul **s** fiind vid

b) ABC13 c) AB13 d) ABC133
e) ABC131

9. Ce şir de caractere va afişa secvenţa următoare?

```
char *s="abcdefg",*ptr;
ptr=s; ptr+=4;
puts(ptr);
```

a) "fg" b) "efg" c) "defg"
d) "cdefg" e) secvenţa este eronată

10. Ce va afişa programul următor?

```
#include<stdio.h>
void main()
{
      char *a[3]={"abc","def",
                          "ghi"};
      char *p=&a[0][0];
      printf("%s%c%c",a[1],a[2][1],
                          *(p+5));
}
```

a) abc d NULL b) abc d e
c) def h NULL d) def h e
e) programul va semnala eroare de compilare

11. Ce va afişa cea de-a doua instrucţiune de tipărire din programul de mai jos:

```
#include<stdio.h>
void main()
{
      char a[12]="abcdefghij";
      char *p=a; int j;
      for(j=0;j<12;j++)
                *(p+j)=a[j]^j;
      printf("%s\n",p);
      for(j=0;j<12;j++)
                *(a+j)=p[j]^j;
      printf("%s", p);
}
```

a) nimic b) textul "abcdefghij"
c) textul "jihgfedcba"
d) o succesiune de caractere imprevizibilă
e) programul conţine erori

12. Care dintre instrucţiunile programului de mai jos sunt eronate?

```
#include<stdio.h>
#include<string.h>
void main()
{
      char a[10],b[10];int k; //(1)
      scanf("%s %s",a,b);     //(2)
      k=strlen(a)/2;          //(3)
      a[k]='*';               //(4)
      printf("%d",strlen(a)<
                  strlen(b));//(5)
      b=a;                    //(6)
}
```

a) declaraţia de variabile din linia (1)
b) citirea şirurilor din linia (2)
c) atribuirile din liniile (3) şi (4)
d) afişarea din linia (5)
e) atribuirea din linia (6)

13. Precizaţi ce şir de caractere se va afişa în urma execuţiei programului următor:

```
#include<stdio.h>
#include<string.h>
void main()
{
      char s[20]="BorLanD C++ 3.1";
      int j;
      for(j=0;j<strlen(s);j++)
        if((s[j]>='A')&&
                        (s[j]<='Z'))
          s[j]-=('A'-'a');
      puts(s);
}
```

a) "BorLand C++ 3.1"
b) "bORlAND c++ 3.1"
d) "BORLAND C++ 3.1"
d) "borland c++ 3.1"
e) "Borland C++ 3.1"

14. Care dintre cele trei instrucţiuni **printf** de mai jos tipăresc şirul "**bd**"?

```
#include<stdio.h>
void main()
{
      char s[6][3]={"ab","ac",
           "ad","bc","bd","cd"};
      printf("%c%c",s[3][0],
                          s[2][1]);
      printf("%s",s[3][0]+
                          s[2][1]);
      printf("%s",s[5]);
}
```

a) toate b) numai prima
c) numai primele două
d) numai prima şi a treia
e) nici una

15. Ce va afişa programul de mai jos?

```
#include<stdio.h>
void main()
{
      char s[10]="AB6X92P3M",
           b[10];
      int j=0,k=0;
      while(s[j])
           if(j%2)  b[k]=s[j++];
      b[k]=0;  puts(b);
}
```

a) BX23 b) A69PM c) B
d) 3 e) nimic

16. Considerând declaraţiile:
```
char s[4]="123",t[4]; int x=123,y;
```
Care din expresiile de mai jos au valoarea 0?
```
a) atoi(s)!=x;
b) itoa(x,t,10)==s;
c) (y= =atoi(s))==x;
d) x= =(atoi(itoa(x,t,10)));
e) !strcmp(itoa(x,t,10),s);
```

17. Ce va afişa programul următor?
```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
void main()
{
      char s[12]="6789",t[12]="6",
           u[12]="89";
      long x=0;
      strcat(t,u);
      if(strcmp(s,t))  x=atol(t);
      else x=atol(s);
      if(strcmp(s,u)>0)  x=atol(u);
      printf("%ld", x);
}
```
a) 0 b) 6 c) 89
d) 689 e) 6789

18. Ce afişează programul următor?
```
#include<stdio.h>
void main()
{
      char *s1="EXEMPLU SIMPLU",
           *s2="SIMPLU";
      printf("\n%.8s%.6s",s1,s2);
}
```
a) "EXEMPLU" b) "EXEMPLU SIMPLU"
c) "EXEMPLU SIMPLU SIMPLU"
d) "EXEMPLUSIMPLU" e) "SIMPLU"

19. Ce afişează programul următor?
```
#include<stdio.h>
void main()
{
      char *s="123456789",*t,*u;
      u=&s[4],s+=3,t=&s[1];
      printf("%d%d%d",u==s,u==t,
                  s==t);
}
```
a) 000 b) 001 c) 010
d) 100 e) 111

20. Care dintre instrucţiunile (1),.....(5) de mai jos sunt eronate?
```
#include<stdio.h>
#include<string.h>
void main()
{
      char *s1,*s2,*s3;
      int x;
      s1="test";              //(1)
      scanf("%s",s2);         //(2)
      s3=&s1;                 //(3)
      printf("%s",s1+s2);     //(4)
      x=strlen(*s2);          //(5)
}
```
a) 2,3 ,4 b) 2,3,4,5 c) 4,5
d) 3,5 e) 3,4,5

21. Fie programul:
```
#include<stdio.h>
void main()
{
      char *s,*t,*u;
      int j,x;
      gets(s);
      for(x=0,j=0;s[j];t=&s[j],
      u=t+1,u[0]==t[0]? x=1:0,j++);
      printf(„%d", x);
}
```
În urma execuţiei programului, se va afişa valoarea 0, dacă:
a) toate caracterele şirului **s** sunt identice
b) în şirul **s** există cel puţin două caractere succesive diferite
c) în şirul **s** există cel mult două caractere succesive identice
d) în şirul **s** există cel puţin două caractere succesive identice
e) în şirul **s** nu există două caractere succesive identice

22. Considerăm următoarele noţiuni:
A)vector de doi pointeri către caracter
B)pointer către şir de două caractere
şi următoarele declaraţii de variabile:
```
I)   char *a[2];
II)  char (*b)[2];
```
Precizaţi corespondenţele corecte:
a) A) cu I) şi B) cu II)
b) A) cu II) şi B) cu I)
c) nu există corespondenţe
d) B) nu are corespondent
e) cele două declaraţii semnifică acelaşi lucru

23. Ce afişează programul de mai jos?
```
#include<stdio.h>
void main()
{
   char *s[5]={"012","345",
               "678","9AB","CDE"};
   char *t,*u; int j;
   t=&s[1][0];
   printf("%d",(*(t+5)==s[2][1]));
   u=&s[3][0]+1;
   j=0;
```

```
   while(u[j]) printf("%c",u[j++]);
}
```
a) 178 b) 1AB c) 078
d) 0AB e) 067

24. Ce afişează programul de mai jos?
```
#include<stdio.h>
#include<string.h>
void main()
{
      char *s[10]={"10","00","10",
                   "10","01","11"};
      char *t="10";
      int i=0,j=i-1;
      while(s[i])
         if(!strcmp(s[i++],t)) j=i;
      printf("%d", j);
}
```
a) -1 b) 0 c) 1
d) 3 e) 4

25. Se dau următoarele declaraţii
```
A)char *a[4][6];
B)char (*b[4])[6];
C)char (*c)[4][6];
D)char ((*d)[4])[6];
```
şi următoarele noţiuni:

N1. vector de 4 elemente, fiecare element este un pointer către un vector de 6 caractere
N2. pointer către matrice de caractere de 4 linii şi 6 coloane
N3. pointer către vector cu 4 elemente, fiecare fiind vector de 6 caractere
N4. matrice de 4 linii şi 6 coloane, fiecare element este pointer către caracter
Precizaţi corespondenţa directă:
a) (A,N1), (B,N2), (C,N3), (D,N4)
b) (A,N4), (B,N1), (C,N2), (D,N3)
c) (A,N4), (B,N1), (C,N3), (D,N2)
d) (A,N2), (B,N3), (C,N4), (D,N1)

26. Câte erori conţine programul următor ?
```
void main()
{
      char *(a[4][6]);
      char b;
      a[2][3]=*(b+2);
      a[3][2]=&b+3;
      *(4+a[2])=&b+1;
      *a[1][3]=b+3;
}
```
a) nici una b) una c) două
d) trei e) patru

Cap.9 Structuri

9.1 Definirea tipurilor structură

Limbajul C oferă programatorului posibilitatea de a grupa datele, date de tipuri diferite putând fi prelucrate atât individual cât şi împreună. Dacă tabloul conţine numai elemente de acelaşi tip, **o structură este formată din mai multe elemente de tipuri diferite**. La rândul lor, structurile definite pot constitui elemente componente pentru formarea de noi tipuri de date (tablouri, structuri, uniuni) . Elementele componente ale unei structuri se numesc **membri** sau **câmpuri** .

Sintaxa declarării unei structuri este următoarea :

```
struct [nume_tip_structura]
{       tip_1  lista_campuri_tip_1;
        tip_2  lista_campuri_tip_2;
                ……….
        tip_n  lista_campuri_tip_n;
} [lista_variabile_structura];
```

În declaraţie poate lipsi precizarea numelui pentru tipul structurii sau a listei de variabile însă nu ambele simultan. În cazul în care lipseşte numele tipului, spunem că variabilele au un tip anonim. Câmpurile componente ale unei structuri pot fi oricare din tipurile :

- predefinite (întreg, caracter, număr real)
- definite de utilizator : scalari (câmpuri de biţi , enumerare) sau compuse (tablouri, structuri, uniuni)

Exemplul 1: Definirea datelor de bază pentru o persoană specificând numele şi vârsta.

```
struct pers
{       char nume[35];
        int an;
};
```

S-a definit o structură de tip ***pers*** fără a fi folosită nici o variabilă de tipul definit.

Exemplul 2: Definirea unei structuri de tip ***angajat*** specificând pe lângă datele de bază (un câmp de tip structură ***pers***) adresa, salariul şi vechimea în muncă.

```
struct angajat
{       struct pers p;          /* structura pers a fost definită anterior */
        char adresa[50];
        long salariu;
        int vechime;
} persoana , firma[100];
struct angajat   p[50];
```

Observăm modul cum poate fi o structură folosită la rândul ei drept câmp al altei structuri. În acest caz, s-a definit o structură de tip ***angajat*** şi variabilele de tip ***angajat*** :

- variabila ***persoana***
- variabila ***firma*** de tip tablou de înregistrări

În a doua declaraţie, s-a folosit tabloul ***p*** de 50 de înregistrări de tipul ***angajat***.

Putem defini orice tip structură folosind **typedef**. Astfel, exemplul anterior se mai poate scrie :

```
typedef struct
{       struct pers p;
        char adresa[50];
        long salariu;
        int vechime;
} ANGAJAT;
ANGAJAT    persoana , firma[100];
```

Am definit un tip de structură cu aceleaşi componente ca mai sus şi variabilele ***persoana*** şi ***firma*** cu aceeaşi semnificaţie ca mai înainte. Se observă că nu mai este necesară precedarea tipului de structură de cuvântul cheie **struct**.

Exemplul 3: Se defineşte o structură care grupează informaţii, specificând datele necesare fişei medicale a unui bolnav.

```
typedef struct { int an , luna , zi}  DATA;
struct
```

```
{     struct pers   p;
      DATA  data_internarii;
      char adresa[50];
      char boli[10][30];
}  pacient[200];
```

În ultimul caz, nu s-a mai definit numele tipului structurii, ci doar tabloul de înregistrări ***pacient*** care conţine 200 de înregistrări de tipul fişelor medicale.

Exemplul 4: În cazul în care un tablou este definit ca ultim câmp al unei structuri este acceptată omisiunea primei dimensiuni din tablou. Astfel, este acceptată declaraţia :

```
struct vector
{     int n;
      int dim[];
};
```

Exemplul 5: În implementările limbajului C există diferite structuri predefinite. Spre exemplu, pentru utilizarea numerelor complexe este definită în header-ele **math.h** şi **complex.h** următoarea structură :

```
struct complex
{     double x;
      double y;
};
```

9.2 Iniţializarea structurilor

Datele componente ale unei structuri se pot iniţializa. În acest scop, la sfârşitul declaraţiei sau definiţiei structurii se pune caracterul egal şi se scriu între acolade, în ordine, valorile componentelor, delimitate prin virgulă. Iniţializarea unei structuri se face enumerând valorile, pentru fiecare din membrii săi.

Exemplul 1: O iniţializare corectă este următoarea :

```
struct pers{ char nume[3];int varsta;}s={"Marcel Dima", 35};
```

Se observă că valorile câmpurilor trebuie date în ordinea de definiţie a acestora în cadrul structurii.

Exemplul 2:

```
struct pers s={30,"Andronic Virgil"};
```

Structura este incorect iniţializată deoarece valorile nu sunt date în ordinea declarării câmpurilor din structură.

Exemplul 3: Ultimile valori ale componentelor pot lipsi. De exemplu, este corectă următoarea iniţializare, chiar dacă nu au fost iniţializate toate componentele :

```
struct s
{     int inf;
      char n[20];
      float x;
};
struct  s  s1={1};
```

Pentru variabila structurată ***s1*** de tip ***s***, se va iniţializa doar primul câmp **inf** cu valoarea 1. Celelalte două câmpuri vor fi iniţializate automat astfel :

- câmpul **n** cu valoarea şirului vid “”
- câmpul numeric **x** cu valoarea 0

Exemplul 4: În exemplul următor se iniţializează o variabilă de tipul **struct persoana**, precum şi a unui tablou cu elemente de acest tip.

```
struct persoana
{     char nume[32];
      int varsta;
      float salariu;
};
struct  persoana  pers={"Alex",28,1200000.0};
struct  persoana  grup[]=
      {   {"Gigi" , 32, 3000000.0} ,
          {"Mimi" , 19 , 1500000.0} ,
          {"Fred", 33 , 2950000.0}
      };
```

9.3 Operaţii permise asupra structurilor

Operaţia principală care poate fi efectuată asupra unei variabile de tip structură este selectarea unui câmp, utilizând operatorul de selecţie "**.**" conform sintaxei :

variabila_structura.camp_selectat

Câmpul selectat se comportă ca o variabilă şi i se pot aplica toate operaţiile care se pot aplica variabilelor de acel tip. Deoarece structurile se prelucrează frecvent prin intermediul pointerilor, a fost introdus un operator special, care combină operaţiile de indirectare şi selectare a unui câmp, anume **"→"**. Expresia **p→camp_selectat** este interpretată de compilator la fel ca expresia **(*p).camp_selectat**. Întotdeauna, unei variabile de tip structură i se pot aplica operatorii **&** (calculul adresei) şi **sizeof** (calculul mărimii zonei de memorie ocupate de variabilă).

În plus sunt permise următoarele operaţii :

a) unei variabile de tip structură i se poate atribui valoarea altei variabile de tip structură
b) variabilele de tip structură pot fi transmise ca parametri funcţiilor
c) o funcţie poate avea ca rezultat o valoare de tip structură

Exemplul 1: Fie structura :

```
struct pers
{     int ani;
      char nume[30];
} x,y,*ps;
```

Tipărirea valorilor elementelor componente se face astfel :

a) în cazul accesării câmpurilor pentru structura ***x*** care este de tip ***pers***, se va folosi **selecţia directă**. O tipărire corectă se poate face astfel :

```
printf("\nNumele:%s\nVarsta:%d\n",x.nume,x.ani);
```

b) în cazul accesării câmpurilor structurii indicate de pointerul la ***pers*** numit ***ps***, se va folosi **selecţia indirectă**. O secvenţă corectă de tipărire este :

```
printf("\nNumele:%s\nVarsta:%d\n",ps->nume,ps->ani);
```

Pentru atribuirea conţinutului structurii ***x*** la structura ***y*** se poate folosi una din secvenţele :

a) atribuirea câmp cu câmp : `y.ani=x.ani ; strcpy(y,nume,x.nume);`
b) atribuirea directă între două structuri de acelaşi tip : `y=x;`

Exemplul 2: Pentru structura de tip ***pers*** şi variabilele definite anterior sunt corecte atribuirile :

a) se atribuie unui pointer adresa structurii
b) se atribuie unei structuri conţinutul indicat de un pointer la o structură de acelaşi tip

```
ps=&x; /* adresa indicată de ps este egală cu adresa lui x */
y=*ps; /* conţinutul lui y devine egal cu conţinutul structurii de la adresa pointată de ps */
```

Atribuirea conţinutului unui pointer la o structură cu o structură de acelaşi tip se poate face astfel :

```
*ps=y;
```

Această atribuire este echivalentă cu secvenţa :

```
(*ps).inf=y.inf ;
strcpy((*ps).nume,y.nume);
```

Următoarele două atribuiri sunt echivalente cu cele două de dinainte :

```
ps->inf=y.inf;
strcpy(ps->nume,y.nume);
```

Folosirea expresiei de tipul **(*ps).inf** este corectă dar greoaie şi neuzuală. În locul ei este recomandată forma **ps->inf** .

Exemplul 3: Tablourile nu pot fi atribuite direct. Totuşi, dacă tabloul este membru al unei structuri, atunci atribuirea poate avea loc .

```
#include<stdio.h>
#include<stdlib.h>
struct  tablou  {int x[10]};
void main()
{     struct tablou a={{1,1,1,1,1,1,1,1,1,1}} ,
                     b={{2,2,2,2,2,2,2,2,2,2}};
      struct tablou *a1 , *b1;
      /* copierea directă a structurilor; are loc copierea directă a tabloului folosindu-l drept
      câmp al unei structuri */
      b=a;
      /* se alocă spaţiul necesar structurii şi adresa spaţiului alocat se stochează în pointerul
```

```
            la structură */
          a1=(struct tablou*)malloc(sizeof(struct tablou));
          *a1=a;
          b1=(struct tablou*)malloc(sizeof(struct tablou));
          /* se copiază conţinutul structurii de la adresa indicată de pointerul a1 peste conţinutul
            structurii indicate de pinterul b1 */
          *b1=*a1;
          /* sau : *(struct tablou)b1=*(struct tablou)a1 */
    }
```

Observăm că s-au putut face atribuirile directe cu tablouri care sunt câmpuri ale unei structuri, atât între structuri, cât şi între pointeri la structuri. În acest exemplu, toate structurile şi conţinutul pointerilor la structuri primesc valoarea structurii ***a***.

Exemplul 4: Se consideră o grupă de **n** studenţi (**n<=40**), pentru fiecare dintre ei cunoscându-se numele şi media anuală. Se cere să se afişeze studenţii în ordinea descrescătoare a mediilor.

```
#include<stdio.h>
#include<conio.h>
void main()
{   struct
    {  char nume[30];
       float media;
    } aux , grupa[40];
    int n,i,j;
    float media;
    clrscr();
    /* citirea datelor */
    printf("Numarul studentilor : ");scanf("%d",&n);
    puts("\n    Introduceti datele studentilor :\n");
    for(i=0;i<n;i++)
    {  getchar();
       printf("Numele :");gets(grupa[i].nume);
       printf("Media : ");scanf("%f",&media);
       grupa[i].media=media;
    }
    /* sortarea tabloului grupa descrescator dupa medie */
    for(i=0;i<n-1;i++)
      for(j=i+1;j<n;j++)
      if(grupa[i].media<grupa[j].media)
      {  aux=grupa[i];
         grupa[i]=grupa[j];
         grupa[j]=aux;
      }
    /* afisarea rezultatelor */
    clrscr();
    puts("Studentii ordonati desc. dupa medii :\n");
    for(i=0;i<n;i++)
      printf("%30s%5.2f\n",grupa[i].nume, grupa[i].media);
    getch();
}
```

9.4 Exerciţii şi teste grilă

1. Fie structura :

```
struct data {
     int zi, luna , an;
} d, *dl;
```

Cum se accesează membrul "**zi**" ?

a) d.zi respectiv dl.zi
b) d->zi respectiv dl->zi
c) d->zi respectiv dl.zi
d) d.zi respectiv dl->zi

2. Fie structura :

```
struct data {
```

```
     int zi, luna , an;
}*d;
```

Care este expresia logică a cărei valoare arată că anul este sau nu bisect ?

```
a) an%4==0 && an%100!=0
b) d.an%4==0 && d.an%100!=0
c) d.an%4==0 && d.an%100!=0 ||
            d.an%400==0
d) d->an%4==0 && d->an%100!=0 ||
            d->an%400==0
```

3. Linia de cod care declară o variabilă structură numită **total** de tip **sample** este :

```
a) type total: sample;
b) struct total;
c) struct sample total;
d) declare total as type sample;
```

4. Linia de cod care asignează valoarea 10 câmpului **loop** din structura **total** (de tip **sample**), este :

```
a) loop=10;
b) total.loop=10;
c) sample.total.loop=10;
d) sample.loop=10;
```

5. Linia de cod care afişează valoarea câmpului **word** din structura **total** (de tip **sample**), este :

```
a) printf("%s",total);
b) printf("%s",word);
c) printf("%s",total-word);
d) printf("%s",total.word);
```

6. Se dă urmatoarea secvenţă de cod :

```
struct computer {
       int cpuSpeed;
       char cpuType[10];
};
struct computer myComputer;
```

Referindu-ne la codul de mai sus, cum accesaţi primul caracter din **cpuType** ?

```
a) char c=myComputer.cpuType(0);
b) char c=myComputer.cpuType;
c) char c=myComputer[0].cpuType;
d) char c=myComputer.cpuType[0];
```

7. Fie secvenţa :

```
typedef struct
{ long cust_id;
  char custName[50];
  double balance;
} CUSTOMER_REC;
CUSTOMER_REC customer[50];
int i;
/*mai jos in program*/
for(i=0;i<50;i++)
{
    printf("%s\n",?????);
}
```

Ce ar trebui pus in locul **????** pentru a afişa fiecare element **custName** în codul anterior :

```
a) customer[i]->custName;
b) customer.custName[i];
c) customer[i].custName;
d) customer->custName[i];
```

8. Fie declaraţia:

```
struct computer{
          int cpuSpeed;
          char cpuType[10];
} comp[]=
  {
        {400,"Pentium"} ,
        {266,"PowerPC"} ,
        {333,"Sparc"},
   };
```

Se dă şirul de structuri de mai sus. Care dintre următoarele expresii va evalua numărul structurilor din şir (în cazul de faţă este 3)?

```
a) sizeof(*comp)/sizeof(comp)
b) sizeof(*comp)
c) sizeof(comp)
d) sizeof(comp)/sizeof(*comp)
```

Cap.10 Exploatarea fişierelor

10.1 Noţiunea de fişier

Prin **fişier** se înţelege o structură de date, cu componente numite **înregistrări**, ce pot avea o dimensiune fixă sau variabilă, cel de-al doilea caz impunând existenţa unor marcaje speciale numite *separatori de înregistrări*. Fişierele pot fi clasificate după mai multe criterii. Din punct de vedere al accesului la componente se împart în:

- **fişere cu acces secvenţial** ale căror înregistrări pot fi prelucrate numai în ordinea în care sunt stocate în fişier
- **fişiere cu acces direct** ale căror componente pot fi prelucrate în orice ordine. În cazul în care prelucrarea nu se face secvenţial, înainte de fiecare operaţie de citire/scriere trebuie furnizată informaţia necesară selectării componentei ce urmează a fi prelucrată

Din punct de vedere al conţinutului, fişierele se împart în două categorii:

- **fişiere text** care conţin numai caractere structurate pe linii
- **fişiere binare** în care informaţia este văzută ca o colecţie de octeţi

Biblioteca de funcţii **stdio.h** oferă posibilitatea operării cu fişiere printr-o structură numită **FILE**. Orice operaţie cu fişiere necesită o asemenea structură, care se iniţializează la deschiderea unui fişier şi al cărei conţinut devine nefolositor după închiderea sa.Gestionarea fişierelor se face printr-un pointer la structura predefinită **FILE**. Declararea unui astfel de pointer se face conform sintaxei:

FILE *identificator_fisier;

10.2 Deschiderea unui fişier

Se realizează cu ajutorul funcţiei **fopen** care are sintaxa de mai jos:

FILE *fopen(”nume_fisier”, ”mod_deschidere”);

în care : *nume_fişer* este numele complet (calea pe disc) a fişierului care se deschide, iar *mod_deschidere* precizează modul în care se deschide fişierul şi poate avea următoarele valori:

- ”**r**” pentru citirea unui fişier existent; se produce o eroare dacă fişierul nu există
- ”**w**” deschide un fişier pentru scriere; dacă fişierul există, îi distruge conţinutul
- ”**a**” se adaugă informaţie în fişier, la sfârşitul acestuia
- ”**r+**” în acelaşi timp citeste, respectiv scrie în fişier; acesta trebuie să existe
- ”**w+**” deschide un fişier pentru citire şi scriere; dacă acesta există, conţinutul este distrus
- ”**a+**” adăugare; dacă fişierul există, conţinutul este distrus
- ”**t**” fişierul este de tip text
- ”**b**” fişierul este binar

Se pot face combinaţii cu opţiunile de mai sus. De exemplu, deschiderea unui fişier text pentru citire se face cu opţiunea ”**rt**”. Crearea unui fişier binar pentru scriere este posibilă prin ”**wb**”. Nu contează ordinea în care sunt date literele în şirul *mod_deschidere*.

În caz de succes, funcţia **fopen** returnează un pointer la noul flux de comunicare deschis; altfel întoarce pointerul **NULL**. Operaţia de deschidere a unui fişier trebuie însoţită de verificarea reuşitei respectivei operaţii conform modelului de mai jos:

```
if(identificator_fisier)
   instructiuni_operatie_reusita
else
   instructiune_eroare_la_deschidere;
```

Exemple: Fie declaraţia **`FILE *f ;`** vom încerca următoarele deschideri de fişiere:

1) **`f=fopen(”test1.in”,”rt”);`** Am deschis fişierul ”**test1.in**” din directorul curent pentru citire în mod text. Litera ”**t**” nu era necesară, modul text fiind modul de deschidere implicit al fişierelor în C. Dacă fişierul nu există, atunci funcţia întoarce **NULL**. O deschidere mai riguroasă ar fi:
```
if((f=fopen (”test1.in”,”rt”))==NULL)
{ printf(“Eroare la deschiderea fisierului !”); exit(1); }
```
2) **`f=fopen(“test2.out”,”wb”);`** Este deschis/creat fişierul ”**test2.out**” din directorul curent în mod binar pentru scriere.
3) **`f=fopen(“c:\\tc\\test3.bin”,”ab+”);`** Este deschis fişierul ”**test3.bin**” din directorul ”**c:\tc**” în mod binar pentru citire/scriere fiind poziţionat la sfârşitul fişierului.

10.3 Închiderea unui fişier

Se poate face cu ajutorul funcţiilor:

- **int fclose(FILE *f);** care închide fişierul specificat şi returnează 0 în caz de succes sau **EOF** în cazul apariţiei unei erori
- **int fcloseall(void);** care închide toate fişierele deschise şi returnează numărul total de fişiere pe care le-a închis sau **EOF** la apariţia unei erori

10.4 Funcţia de verificare a sfârşitului unui fişier

Pentru a verifica dacă poziţia curentă de citire/scriere a ajuns la sfârşitul unui fişier se foloseşte funcţia **int feof(FILE *f);** care întoarce valoarea 0 dacă poziţia curentă nu este la sfârşitul fişierului şi o valoarea diferită de 0 dacă poziţia actuală indică sfârşitul de fişier.

10.5 Funcţii de citire/scriere caractere

Pentru citirea unui caracter dintr-un fişier text se foloseşte funcţia

int fgetc(FILE *f);

Dacă citirea a avut loc cu succes, se întoarce valoarea caracterului citit, iar în caz de eroare este întoarsă valoarea **EOF**.

Pentru scrierea unui caracter într-un fişier text se foloseşte funcţia

int fputc(int c,FILE *f);

Funcţia întoarce caracterul care s-a scris în caz de succes, respectiv **EOF** în caz de eroare.

Exemplul 1: Copierea unui fişier caracter cu caracter.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
      FILE *in,*out;
      // se incearca deschiderea fisierului sursa
      if((in=fopen("test.c","r"))==NULL)
      {   printf("fisierul nu poate fi deschis pentru citire");
          exit(1);
      }
      // se incearca deschiderea fisierului destinatie
      if((out=fopen("test2.bak","w"))==NULL)
      {   printf("fisierul nu poate fi deschis pentru scriere");
          exit(1);
      }
      // se copie un caracter din sursa si se scrie in destinatie
      while(!feof(in))  fputc(fgetc(in),out);
      fclose(in); fclose(out);
}
```

Exemplul 2: Se numără câte caractere de fiecare tip (litere, cifre şi neafişabile) sunt într-un fişier text.

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
void main()
{
      FILE *f; int l,c,g; char ch;
      l=c=g=0;
      if((f=fopen(„in.txt","r"))==NULL)
      {  printf(„nu se poate deschide fisierul pentru citire");
         exit(1);
      }
      do{
            ch=fgetc(f);
            if(isalpha(ch)) l++;
            if(isdigit(ch)) c++;
            if(!isprint(ch)) g++;   // caracterul nu este afisabil
```

```
        }while(ch!=EOF);
        printf("\n numarul de litere este %d",l);
        printf("\n numarul de cifre este %d",c);
        printf("\n numarul de caractere neafisabil este %d",g);
        fclose(f);
}
```

10.6 Funcţii de citire/scriere pe şiruri de caractere

Funcţia **char *fgets(char *s, int n, FILE *f);** citeşte un şir de caractere dintr-un fişier oarecare. Primul parametru, **s**, reprezintă zona de memorie în care se stochează şirul citit. Parametrul **n** indică numărul maxim de caractere care se vor citi din fişier. Dacă se detectează mai puţine caractere rămase pe linia curentă din fişier, citirea se opreşte la întâlnirea caracterului sfârşit de linie **'\n'**. În cazul unei citiri reuşite, funcţia întoarce un pointer la şirul citit, în caz de sfârşit de fişier sau de eroare se întoarce pointerul **NULL**.

Observaţie: Funcţia **fgets** inserează în linia citită şi caracterul **'\n'** generat la apăsarea tastei Enter. Pentru o prelucrare corectă a şirului de caractere citit din fişier, acest caracter trebuie eliminat utilizând spre exemplu instrucţiunea **linie[strlen(linie)-1]=0** care scrie terminatorul de şir **'\0'** peste caracterul **'\n'** (am considerat că şirul citit din fişier s-a depus în variabila **linie**).

Funcţia **int fputs(const char *s, FILE *f);** scrie şirul de caractere **s** (care nu se modifică) în fişierul **f**. În caz de eroare, funcţia întoarce valoarea **EOF**.

Exemplul 1: Copierea a două fişiere text, linie cu linie.

```
#include<stdio.h>
void main()
{
        FILE *f1,*f2;  char linie[80];
        f1=fopen("in.txt","r");
        f2=fopen("out.txt","w");
        do{
              if(feof(f1)) break;
              fgets(linie,80,f1); fputs(linie,f2);
        }while(1);
        fcloseall();
}
```

Exemplul 2: Afişarea unui fişier text pe ecran. La fiecare afişare a 22 de linii se va face o pauză, afişarea continuând la apăsarea unei taste.

```
#include<stdio.h>
void main()
{
        FILE *f;  char linie[80];
        long n=0;   // numara cate linii au fost citite
        f=fopen("in.txt","r");
        while(!feof(f))
        {  fgets(linie,80,f); printf("%s",linie);
           if(++n%22==0) getch();
        }
        fclose(f);
}
```

Exemplul 3: Fiind dat un fişier, să se determine numărul de linii, linia de lungime maximă şi lungimea fişierului (numărul de caractere utile).

```
        #include<stdio.h>
        #include<string.h>
        #include<stdlib.h>
        void main()
        {
              FILE *f;  char linie[120];
              int l,max=0;    // lungimea celei mai lungi linii din fisier
              long nr=0,n=0;  // nr-lungimea fisierului, n-numarul de linii
              f=fopen("in.txt","r");
              if(f==NULL)  {  printf("fisierul nu exista"); exit(1); }
              while(!feof(f))
              {   n++;
```

```
            fgets(linie,120,f);
            l=strlen(linie);
            if(max<l)  max=l;
            nr+=l;
        }
        printf("linia de lungime maxima :%d\n",max);
        printf("numarul de linii :%ld\n",n);
        printf("lungimea fisierului :%ld\n",nr);
        fclose(f);
}
```

Exemplul 4: Programul următor numără apariţiile unui cuvânt într-un fişier text. Citirea acestuia se face linie cu linie.

```
#include<stdio.h>
#include<string.h>
void main()
{
      typedef char STRING[256];
      STRING cuv,linie,
      FILE *f;
      char *p;
      int nr=0;
      f=fopen("in.txt","r");
      printf("cuvantul cautat : "); scaf("%s",cuv);
      while(!foef(f))
      {   fgets(linie,256,f);
          p=linie;
          while(p!=NULL)    // mai sunt caractere in linie
          {   p=strstr(p,cuv);
              // cauta cuvantul in sirul curent de caractere
              if(p)      // cuvantul apare in sirul curent)
              {  nr++;   // numarul aparitia cuvantului
                 p++;    // avansez la urmatorul caracter din sir
                         // pentru a repeta cautarea
              }
          }
      }
      printf("numarul de aparitii: %d",nr);
      fclose(f);
}
```

10.7 Funcţii de citire/scriere cu format

Pentru citirea valorilor unor variabile dintr-un fişier text se foloseşte funcţia

int fscanf(FILE *f, "specificatori_de_format", adrese_variabile);

Funcţia **fscanf** realizează următoarele:

- citeşte o secvenţă de câmpuri de intrare caractere cu caracter
- formatează fiecare câmp conform specificatorului de format corespunzător
- valoarea obţinută este stocată la adresa variabilei corespunzătoare

Valoarea întoarsă, în caz de succes, este numărul de câmpuri citite. Dacă nu a fost citit nici-un câmp, funcţia întoarce valoarea 0. Dacă funcţia citeşte sfârşitul de fişier, atunci valoarea întoarsă este **EOF**. Specificatorii de format pentru funcţia **fscanf** sunt aceeaşi cu cei ai funcţiei **scanf**.

Pentru scrierea cu format a datelor într-un fişier text se foloseşte funcţia:

int fprintf(FILE *f, "specificatori_de_format", expresii);

Funcţia **fprintf** realizează următoarele:

- acceptă o serie de argumente de tip expresie pe care le formatează conform specificatorilor de format corespunzători
- scrie datele formatate în fişierul specificat

Funcţia **fprintf** foloseşte aceleaşi formate ca şi funcţia **printf**.

10.8 Funcţii de citire/scriere a fişierelor pe blocuri de octeţi

Aceste funcţii realizează citirea şi scrierea datelor fără a face o interpretare sau conversie a acestora. Se folosesc doar în modul de acces binar şi sunt orientate pe zone compacte de octeţi. Funcţiile pot fi folosite şi pentru citirea/scrierea înregistrărilor.

Funcţia **fread** citeşte **n** înregistrări dintr-un fişier, fiecare înregistrare având **dim** octeţi. Funcţia întoarce numărul înregistrărilor citite şi are sintaxa:

int fread(void *pointer, int dim, int n, FILE *f);

Funcţia **fwrite** scrie într-un fişier **n** înregistrări a câte **dim** octeţi fiecare şi întoarce numărul total de înregistrări scrise. Funcţia are sintaxa:

int fwrite(void *pointer, int dim, int n, FILE *f);

Exemplul : Se arată modul în care trebuie scrisă o înregistrare într-un fişier binar.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct pers
{   int ani;
    char nume[20];
};
void main()
{
      FILE *f;  struct pers s;
      f=fopen("test.bin","wb");
      if(!f)
      {   printf("fisierul nu se poate deschide"); exit(1); }
      s.ani=40;
      strcpy(s.nume,"Mihai Popescu");
      fwrite(&s,sizeof(s),1,f);    // scrie structura s in fisier
      fclose(f);
}
```

10.9 Funcţii pentru aflarea poziţiei curente şi schimbarea ei

Funcţia cea mai folosită pentru determinarea poziţiei curente de citire/scriere este

long ftell(FILE *f);

Pentru poziţionarea în fişier se utilizează funcţiile :

- **int fseek(FILE *f, long nr, int origine);** care mută indicatorul de fişier cu un număr de **nr** octeţi faţă de punctul **origine**. Originea arată punctul faţă de care este măsurată deplasarea : 0 (sau SEEK_SET) faţă de începutul fişierului, 1 (sau SEEK_CUR) faţă de poziţia curentă, 2 (sau SEEK_END) faţă de sfârşitul fişierului.
- **void rewind(FILE *f);** mută poziţia curentă a fişierului la începutul său

Exemplu: Determinarea lungimii unui fişier.

```
#include<stdio.h>
void main()
{
      FILE *f;    f=fopen("in.txt","r");
      fseek(f,0L,SEEK_END);   // fseek(f,0L,2);
      printf("fisierul are %ld octeti",ftell(f));
      fclose(f);
}
```

10.10 Exerciţii şi texte grilă

1. Care dintre afirmaţiile de mai jos sunt adevărate ?

a) instrucţiunea care deschide fişierul "nr.txt" pentru citire şi returnează un pointer către fişierul deschis este **f=fopen("r", "nr.txt");**

b) pentru a putea citi din fişier folosim atribut "**r**" la deschidere, iar pentru a scrie în fişier îl deschidem cu atributul "**w**"

c) pentru a testa dacă nu s-a ajuns la sfârşitul fişierului referit de pointerul **f,** vom scrie **!feof(f)**

d) pentru a închide fişierul referit de pointerul **f** vom scrie **close(f)**
e) nici una dintre afirmaţiile de mai sus nu este adevărată

2. Se consideră un fişier definit prin pointerul **f**, şi care conţine următoarele valori pe primele două rânduri :

```
4  7  2.5  -6.23
#  8
```

Fie următoarele declaraţii de variabile:

```
FILE *f;
int x,y; float a,b,d; char c;
```

Care dintre secvenţele de instrucţiuni de mai jos trebuie executate astfel încât toate variabilele declarate să primească valori citite din fişierul **f** ?

```
a) fscanf(f,"%d %f %d %f\n",&x,
                        &a,&y,&b);
   fscanf(f,"%c %f",&c,&d);
b) fscanf(f,"%d %d %f %f\n",&x,
                        &y,&a,&b);
   fscanf(f,"%c %f",&c,&d);
c) fscanf(f,"%d %d %f %f\n",&x,
                        &y,&b,&a);
   fscanf(f,"%f %c",&d,&c);
d) fscanf("%f %f %d %d\n",&b,&a,
                         &y,&x,f);
   fsacnf("%c %f",&c,&d,f);
e) fscanf("%d %f %d %f\n",&x,&a,
                         &y,&d,f);
   fscanf("%f %c",&d,&c,f);
```

3. În timpul execuţiei programului următor sunt posibile următoarele situaţii :

```
#include<stdio.h>
void main()
{
      FILE *f;  int x=1,s=0;
      f=fopen("suma.txt","r");
      while(!feof(f) && x)
      {  fscanf(f,"%d",&x);
          if(x%2) s+=x;
      }
      fclose(f);
      printf("\ns=%d",s);
}
```

a) programul este corect sintactic
b) pentru a funcţiona citirea din fişier, acesta trebuie deschis în alt mod
c) programul va intra într-un ciclu infinit
d) dacă în fişier se găsesc, pe acelaşi rând separate prin câte un spaţiu, numerele 2 5 4 3 6 1 0 7, atunci programul va afişa s=16
e) modul în care este închis fişierul nu corespunde cu modul în care a fost deschis

4. Fie fişierul identificat prin descriptorul **f**, având următorul conţinut:

```
5
2  3  4  6  7  8
```

Care dintre secvenţele următoare de program S1, S2, S3 poate fi executată, astfel încât, în vectorul **v** să se citească corect toate numerele din fişier ?

```
    //secventa S1
fscanf(f,"%d",&n);
for(i=0;i<n;i++)
       fscanf(f,"%d",&v[i]);
   //secventa S2
j=0;
while(!feof(f))
{   fscanf(f,"%d",&v[j]); j++;  }
n=j;
   //secventa S3
j=0;
do{
     fscanf(f,"%d",&v[j]);j++;
}while(!feof(f));
n=j-1;
```

a) toate b) nici una
c) numai S1 şi S2 d) numai S2 şi S3
e) numai S1

5. Ce număr se va găsi pe al patrulea rând al fişierului "**4.txt**" după execuţia programului următor ?

```
#include<stdio.h>
void main()
{
      FILE *f;
      f=fopen("4.txt","w");
      int n=8,j=0,
          v[8]={1,3,8,5,0,6,7,4};
      while(v[j]%2) j++;
      while(j<n)
        if(v[j++])
           fprintf(f,"%d\n",v[j]);
      fclose(f);
}
```

a) 5 b) 0 c) 6
d) 7 e) 4

6. Fie programul:

```
#include<stdio.h>
#include<math.h>
void main()
{
      FILE *f,*g;  int e;
      char c1, c2;
      f=fopen("1.txt","r");
      g=fopen("2.txt","r");
      e=1;
```

```
      do{
            c1=fgetc(f);
            c2=fgetc(g);
            if(c1!=c2) e=0;
      }while(!(feof(f)||feof(g))
                              &&e);
      if(e)
        if(!(feof(f)&&feof(g)))e=0;
      fclose(f); fclose(g);
      printf("%d",e);
}
```

Programul de mai sus afişează valoarea 1 dacă:

a) cele două fişiere diferă prin cel puţin un caracter
b) cele două fişiere sunt identice
c) în cele două fişiere există şi caractere identice
d) cele două fişiere au acelaşi număr de caractere
e) nici unul dintre cazurile de mai sus

7. Precizaţi care va fi conţinutul fişierului **g** după execuţia programului următor, dacă fişierul **f** conţine pe fiecare linie o zi a săptămânii (luni,…..,duminica) :

```
#include<stdio.h>
#include<math.h>
void main()
{
      FILE *f,*g;  int j=1;
      char s[11],c1,c2;
      f=fopen("7.txt","r");
      g=fopen("7_2.txt","w");
      while(j++<4)  fgets(s,10,f);
      fprintf(g,"%d ",j-1);
      fputs(s,g);
      fclose(f); fclose(g);
}
```

a) 3 Miercuri b) 3 Joi
c) 4 Miercuri d) 4 Joi e) 5 Joi

8. Fie programul următor:

```
#include<stdio.h>
void main()
{
      FILE *f,*g;  int a,x,s;
      f=fopen("in.txt","r");
      g=fopen("out.txt","w");
      scanf("%d",&a);
      while(!feof(f))
      {  s=0;
         while(s<a && !feof(f))
         { fscanf(f,"%d",&x);
           s+=x;  }
         fprintf(g,"%d",s);
      }
      fclose(f); fclose(g);
      printf("\n s=%d",s);
}
```

Dacă de la tastatură se introduce valoarea 10, iar conţinutul fişierului "in.txt" este **4 6 3 2 6 15 1** (pe aceeaşi linie), câte numere va scrie programul în fişierul "out.txt" ?

a) nici unul b) unul c) două
d) trei e) patru

9. Câte numere se vor găsi în fişierul "**nr.txt**" după execuţia programului următor ?

```
#include<stdio.h>
void main()
{
    int v[9]={0,1,0,0,2,3,0,4,5},
        j;
    FILE *f;
    f=fopen("nr.txt","w");
    j=0;
    while(j<9)
    { while(v[j])
        fprintf(f,"%3d",v[j++];
      fprintf(f,"%3d",99);
      j++;
   }
   fclose(f);
}
```

a) 4 b) 5 c) 8
d) 9 e) 10

10. Deduceţi ce valoare va afişa programul următor, ştiind că în fişierul **f** se găsesc pe un rând, separate prin spaţii, numerele **1 3 0 0 2 -3 0 -4 -1**

```
#include<stdio.h>
#include<math.h>
void main()
{
      FILE *f;
      int s=1,j=0,a[20];
      f=fopen("nr.txt","r");
      while(!feof(f))
      {  j++;
         fscanf(f,"%d",&a[j]);
         if(a[j]) s*=abs(a[j]);
      }
      printf("\n%d",s); fclose(f);
}
```

a) 1 b) 72 c) -72
d) programul conţine erori de sintaxă
e) nu se pot citi corect numerele din fişier

11. Presupunând că toate liniile fişierului **g** conţin cel mult 100 de caractere, care este acţiunea programului următor ?

```
#include<stdio.h>
void main()
{
      FILE *f,*g;  char s[101];
```

```
      f=fopen("1.txt","a");
      g=fopen("2.txt","r");
      while(!feof(g))
      {  fgets(s,100,g);
         fputs(s,f); }
      fclose(f); fclose(g);
}
```

a) înlocuieşte conţinutul fişierului **g** cu conţinutul fişierului **f**
b) înlocuieşte conţinutul fişierului **f** cu conţinutul fişirului **g**
c) concatenează fişierul **g** la sfârşitul fişierului **f**
d) concatenează fişierul **f** la sfârşitul fişierului **g**
e) nici unul dintre cazurile anterioare

12. Deduceţi ce valoare va afişa programul de mai jos, dacă fişierul text are următorul conţinut:

```
3 3
1 2 3
4 5 6
7 8 9
```

```
#include<stdio.h>
void main()
{
      FILE *f;
      int i,j,m,n,s=0,a[20][20];
      f=fopen("c.txt","r");
      fscanf(f, "%d %d",&m,&n);
      for(i=0;i<m;i++)
       for(j=0;j<n;j++)
       {  fscanf(f,"%d",&a[i][j]);
          if((i+j)%2) s+=a[i][j];
       }
      fclose(f);  printf("%d",s);
}
```

a) 0 b) 8 c) 20
d) 25 e) programul este eronat

13. Se dă fişierul identificat prin descriptorul **f**, cu următorul conţinut:

```
33 1 -45 18 6
```

Ce instrucţiune trebuie scrisă în loc de "……." astfel încât programul următor să tipărească 85?

```
#include<stdio.h>
void main(()
{
      FILE *f;
      int x, y;
      f=fopen("v.txt","r");
      fseek(f,-6,2);
      fscanf(f,%d",&y);
      …….
      fscanf(f,"%d",&x);
      printf("\n%d%d",x,y);
      fclose(f);
}
```

a) fseek(f,11,0); b)fseek(f,-2,2);
c) fseek(f,3,1); d)fseek(f,2,1);
e) fseek(f,-3,2);

14. Precizaţi ce nume se va găsi pe al cincilea rând din fişierul "**p.txt**" după execuţia programului de mai jos:

```
#include<stdio.h>
#include<string.h>
void main()
{
      FILE *f;
      int i=0,j,k; char *aux;
      char *a[9]={"Marius",
       "Claudiu","3rei-Sud-Est",
       "Daniel","Vasile","Dan",
       "Sinacdu","2Pac"};
      while(a[i]) i++;
      for(j=0;j<i-1;j++)
         for(k=j+1;k<i;k++)
          if(strcmp(a[j],a[k])>0)
          {  aux=a[j];a[j]=a[k];
             a[k]=aux;}
      k=0; f=fopen("p.txt","w");
      while(a[k])
         fprintf(f,"%s\n",a[k++]);
      fclose(f);
}
```

a) 2Pac b) Claudiu c) Dan
d) Daniel e) Marius

15. Precizaţi care va fi conţinutul fişierului "**b.txt**" după execuţia programului următor, ştiind că fişierul "**a.txt**" are următorul conţinut:

```
11 2 13 4 15 6 17 8 19
```

```
#include<stdio.h>
#include<math.h>
void main()
{
      FILE *f,*g;  int v[10];
      f=fopen("a.txt","r");
      g=fopen("b.txt","w");
      fread(v,8,1,f);
      fwrite(v,6,1,g);
      fclose(f); fclose(g);
}
```

a) 11 2 13 4 15 6
b) 1 2 2 4 1 6
c) 11 2 13
d) 11 2 1
e) un alt conţinut decât cel indicat

Răspunsuri la testele grilă

1.5 Elemente de bază ale limbajului C

1)a 2)c 3)c 4)b 5)d 6)a 7)c 8)d

2.3 Tipuri fundamentale de date

1)b 2)a 3)d 4)b 5)b 6)c 7)c 8)c 9)a,b,c,d 10)e 11)a,b
12)c 13)b,c,d,e 14)a,c,d 15)d 16)c 17)b 18)c 19)c 20)a

3.8 Funcţii de intrare/ieşire standard

1)c 2)c 3)c 4)c 5)c 6)e 7)b 8)b 9)c 10)b 11)a,c 12)b
13)c 14)b,c 15)a 16)c 17)a 18)d 19)c 20)a 21)a 22)a 23)a 24)a
25)c 26)b 27)c 28)d 29)c 30)c 31)d 32)b 33)b 34)b 35)b 36)b
37)d 38)b

4.13 Operatorii limbajului C

1)b 2)a,d 3)b 4)a,c,d 5)b,c,e,g 7)a,c,f 10)b 11)a,b 13)c
14)a,b 19)c 20)b 21)d 22)b 23)d 24)d 25)a 26)d 27)b 28)a 29)b
30)c 31)b 32)c 33)b 34)d 35)c 36)c 37)a,d 38)d 39)b,c,e 40)c
41)c 42)b,c 43)b 44)b 45)d 46)a,e 47)b,e 48)e 49)e 50)c,e 51)a,c,e
52)a,e 53)c 54)c 55)c 56)a,c,d 57)a,e 58)e

5.16 Instrucţiunile limbajului C

1)a,d 2)a,c,d,e 3)b 4)c 5)e 6)a,c 7)b 8)c 9)e 10)d 11)b
12)c 13)d 14)a 15)a 16)d 17)b 18)a 19)a 20)b 21)c 22)c 23)d
24)d 25)b 26)c 27)d 28)c 29)c 30)d 31)b 32)a 33)d,e 34)e 35)c
36)c 37)a,c 38)a,d,e 39)b 40)d 41)b 42)b,c 43)a 44)b 45)b,e 46)c
47)b,c 48)b 49)a,b,e

6.5 Tablouri

1)c 2)d 3)a,b 4)c 5)d 6)b,d,e 7)e 8)d 9)b 10)a,d 11)d 12)a,b,d
13)b,c 14)d 15)b,d 16)b 17)c 18)a 19)d 20)c 21)c 22)c 23)b 24)d
25)d 26)a 27)a 28)a 29)a 30)c 31)c 32)b 33)d 34)c 35)d 36)b 37)c

7.4 Pointeri

7) b cu e, c cu d 10)b 11)c,d,e 12)d 13)e 14)d 15)a 16)a,b,c 17)c
18)c 19)a,d 20)b 21)c 22)a 23)d 24)b,c 25)d,e 26)b,e 27)c
28)a,c,e 29)e 30)d 31)c 32)b,c 33)d 34)b 35)b 36)a 37)a 38)a 39)a
40)a 41)a 42)a 43)a 44)b 45)c 46)d

8.6 Şiruri de caractere

1)c 2)b 3)c 4)a,b,e 5)a,d,e 6)a 7)b 8)d 9)b 10)d 11)b
12)e 13)d 14)b 15)e 16)a,b 17)d 18)b 19)c 20)e 21)d 22)a 23)b

24)d 25)b,c 26)b

9.4 Structuri

1)d 2)d 3)c 4)b 5)d 6)d 7)c 8)d

10.10 Exploatarea fişierelor

1)b,c 2)b 3)a 4)d 5)d 6)b 7)c 8)e 9)e 10)b 11)c 12)c
13)d,e 14)d 15)d

Bibliografie

1. *Herbert Schildt*
   - "C manual complet", Editura Teora, Bucureşti, 1998
2. *Liviu Negrescu*
   - "Limbajul C,vol.I, II", Editura Albastră, Cluj-Napoca, 1997
3. *Dorian Stoilescu*
   - "Manual de C/C++", Editura Radial, Galaţi, 1998
4. *Damian Costea*
   - "Iniţiere în limbajul C", Editura Teora, Bucureşti, 1996
5. *George-Daniel Mateescu, Pavel Florin Moraru*
   - "Limbajul C++, probleme şi teste grilă pentru liceu şi bacalaureat", Editura Niculescu, Bucureşti, 2001
6. *Claudia Botez, Dumitru Ilinca*
   - "Teste de informatică", Editura Universităţii Tehnice "Ghe. Asachi", Iaşi, 2001