POINTERI ŞI STRUCTURI.
FUNCŢII SPECIFICE DE LUCRU PE ŞIRURI DE CARACTERE.

1. Pointeri şi structuri

Dacă în fişierul header avem declarată o structură:

```
struct   _NUMES {
      // declaraţii de variabiler;
};
typedef struct   _ NUMES NUMES;
```

atunci în funcţii (inclusiv în funcţia **main**) putem folosi pointeri la structura de tip **numeS** definind astfel variabilele de acest tip:

```
struct   _ NUMES *ps=0;
```

sau

```
NUMES *ps = 0;
```

Accesul la membrii structurii NUMES. în cazul în care folosim pointer la structură. se face folosind operatorul **->** (operatorul este format din semnul **–** (minus) şi semnul **>**).

Atenţie! Şi pentru un pointer la o structură trebuie făcută alocare dinamică de memorie.

Exemplu:

Fie structura de tip PUNCT asociată unui punct din plan:

```
typedef struct   _PUNCT {
      double x, y;
}      PUNCT;
```

Într-o funcţie din program avem definiţia unui pointer la această structură

```
PUNCT *A = 0;
/* Am definit variabila A ca fiind pointer la structura PUNCT */
```

Se face alocarea dinamică de memorie:

```
A = (PUNCT *)xmalloc(sizeof(PUNCT));
```

Atribuirea de valori pentru abscisa, respectiv ordonata corespunzătoare punctului **A** din plan se face astfel:

```
A->x = 3.5;      // pentru abscisă (primul membrul al
structurii
```

```
          A->y = 5.0;      // pentru ordonată (al doilea membru al
structurii
```

Când structura care este definită prin intermediul unui pointer are ca membri pointeri, trebuie făcută alocare dinamică şi pentru aceşti pointeri.

Exemplu 1

Fie structura PERSOANA:

```
typedef struct   _PERSOANA {
      char *mume;
      char *prenume;
      int varsta;
      double salariu;
} PERSOANA;
```

şi definirea unui pointer la această structură:

```
PERSOANA *pp=0;
```

Pentru a putea lucra cu acest pointer trebuie făcută alocare dinamică de memorie:

```
pp = (PERSOANA *)xmalloc(sizeof(PERSOANA));
```

şi apoi, alocare dinamică pentru fiecare membru al structurii de tip pointer:

```
pp->nume = (char *)xmalloc(lung*sizeof(char));
```

presupunând că şirul de caractere reprezentând numele are **lung** de caractere (în care am inclus şi caracterul terminator de şir).

Dealocarea memoriei se face în sens invers alocării, adică mai întâi se dealocă memoria alocată pentru membri şi apoi cea alocată pentru structura în ansamblu:

```
      free(pp->nume);
      pp->nume = 0;
.....................................
      free(pp);
      pp = 0;
```

Valorile unui membru al unei structuri definite prin intermediul unui pointer se folosesc în funcţia **scanf** sau în expresii ca orice altă variabilă:

```
scanf( "%lf", &pp->salariu);
impozit = 0.16 * pp->salariu;
```

Exemplu 2:

În programe putem folosi o structură asociată unui vector (ştiut fiind că pentru un vector trebuie să cunoaştem numărul de elemente şi valorile elementelor sale):

```
typedef struct   _VECTORInt {
      int n;          // Numărul de elemente
```

```
        int *a;    // Valorile elementelor vectorului
} VECTORInt;
```

Programul (funcţia **main**) care citşte şi afişează un vector având ca elemente numere întregi poate fi de următoarea formă (vectorul este definit prin intermediul unui pointer la structura VECTORInt.

```
int main (void)
{
      VECTORInt *v = 0;

      v = citireVector();
      afisareVector();

      free(v->a);
      v->a=0;

      free(v);
      v = 0;
      return 0;
}
```

iar definiţiile funcţiilor folosite sunt:

```
VECTORInt *citireVector(void)
{
      VECTORInt v = 0;
      int i;

      v = (VECTORInt *)xmalloc(sizeof(VECTORInt));

      printf("n = ");
      scanf("%d", &v->n);

      v->a = (int *)xmalloc(v->n * sizeof(int));
      for(i=0; i < v->n; i++)
      {
            printf("a(%d)=",i);
            scanf("%d", &v->a[i]);
      }
  return v;
}

void afisareVector(VECTORInt v)
{
      int i;
      for(i=0; i < v->n; i++)
      {
            printf("%d ", v->a[i]);
      }
      printf("\n");
}
```

În cazul în care trebuie să scriem funcţii în să modificăm valorile membrilor unei structuri avem de ales între:

1) a returna structura ai cărei membri vrem să îi modificăm sau

2) a transmite ca parametru un pointer la structura respectivă.

Exemplu: pentru lucrul cu numere complexe avem declarată structura COMPLEX:

```
struct _COMPLEX {
      double real, imag;
};

typedef struct _COMPLEX COMPLEX;
```

Dacă trebuie să citim un număr complex. funcţia care realizează acest lucru poate fi scrisă în două moduri (corespunzând celor două situaţii de mai sus):

1) returnăm o structură

```
COMPLEX citesteComplex1(void)
  {
      COMPLEX c;
      printf("Partea reala a numarului complex: ");
      scanf("%lf", &c.real);
      printf("Partea imaginară a numarului complex: ");
      scanf("%lf", &c.imag);
      return c;
  }
```

iar folosirea în funcţia **main** este:

```
int main(void)
{
      COMPLEX c;
      ......................
      c = citesteComplex1();
      ........................
      return 0;
}
```

sau

2) folosim un pointer la structură

```
void citesteComplex2(COMPLEX *c)
  {
      printf("Partea reala a numarului complex: ");
      scanf("%lf", &c->real);
      printf("Partea imaginară a numarului complex: ");
      scanf("%lf", &c->imag);
  }
```

iar folosirea în funcţia **main** este:

```
int main(void)
{
      COMPLEX c;
      ......................
      citesteComplex2(&c);
      ........................
      return 0;
}
```

2. Funcţii specifice de lucru pe şiruri de caractere

Un şir de caractere se păstrează într-o zonă de memorie organizată sub forma unui tablou unidimensional de tip char. Fiecare caracter se păstrează pe un octet prin codul sau numeric (codul ASCII). După ultimul caracter al şirului se păstrează caracterul '\0'.

Exemplu:

```
char tab[]="Acesta este un sir";
```

în exemplul de mai sus:

tab este un tablou (pointer constant). iar primul element din zona de memorie rezervată are adresa locaţiei de memorie care conţine codul ASCII al caracterului **A**

tab+1 conţine adresa caracterului **c** (este un pointer care conţine adresa locaţiei de memorie în care se găseşte codul ASCII al caracterului **c**).

...

tab[0] sau *tab conţine codul ASCII al caracterului **A**

tab[1] sau *(tab+1) conţine codul ASCII al caracterului **c**

....

Declaraţia de mai sus pentru tab poate fi făcuta şi astfel:

```
const char *p="Acesta este un şir";
      p pointează pe adresa lui 'A'
      p+1 pointează pe adresa lui 'c'
      ...
      p[0] sau *p este caracterul 'A'
      p[1] sau *(p+1) este caracterul 'c'
```

Principalele operaţii cu şiruri de caractere se referă la :

- lungime
- copiere
- concatenare
- comparare
- conversie
- căutare caractere /subşiruri în şiruri

Prototipurile funcţiilor ce lucrează cu şiruri de caractere se găsesc în fişierul header **string.h**.

2.1. Lungimea unui şir de caractere

Lungimea unui şir de caractere este numărul de caractere ce intră în componenţa şirului respectiv (fără '\0' final). Prototipul funcţiei care returnează lungimea unui şir de caractere este:

unsigned strlen(const char *s);

Parametrul formal al funcţiei este un pointer spre o dată constantă deoarece funcţia nu are voie sa modifice şirul căruia îi calculează lungimea.

Exemplu:

```
char *tab="acesta este un sir";
int n;
n=strlen(tab);
```

sau

```
n=strlen("acesta este un sir");
```

2.2. Copierea unui şir de caractere dintr-o zonă de memorie în altă zonă de memorie

Prototipul funcţiei care realizează copierea este:

char *strcpy(char *dest, const char *sursa);

Această funcţie realizează copierea şirului de caractere a cărui adresă de început este conţinută în pointerul **sursa** în zona de memorie care începe de la adresa dată de valoarea pointerului **dest**. Se copie şi caracterul final '\0'. Se presupune că pentru zona de memorie indicată de **dest** (în care se copie) s-a făcut deja alocarea de memorie necesară pentru stocarea întregului şir de caractere (inclusiv terminatorul de şir) de la adresa indicată de **sursa**. **dest** nu e declarată cu const deoarece funcţia modifică zona de memorie corespunzătoare; în schimb **sursa** este declarată cu const. Funcţia returnează un pointer a cărui valoare este adresa zonei în care s-a făcut copierea, adică **dest**.

Exemple:

```
char *tab="acest sir se copie";
int n = strlen(tab)+1;
/ * numărul minim de octeţi pentru destinaţie este n. include
şi un octet pentru terminatorul final '\0', pe care strlen() nu îl
contorizează */
char *t;
char *q;
t = (char *)malloc(n*sizeof(char));
if(t == 0) {
     fprintf(stderr. "Memorie insuficenta\n");
     exit(EXIT_FAILURE);
}
strcpy(t,tab); /* copierea lui tab în t      */
/* Sau putem scrie */
q=strcpy(t,tab);
puts(q);        /* afişare şir */
```

Dacă se doreşte copierea a cel mult **n** caractere din şirul sursă atunci se foloseşte funcţia cu prototipul:

char * strncpy(char *dest, const char *sursa, unsigned n) ;

Dacă n > lungimea sursei se copie toate caracterele. altfel numai primele n. Şirul rezultat nu are '\0' la final.

2.3. Concatenarea a două şiruri de caractere

Funcţia care realizează acest lucru are prototipul:

char * strcat(char *dest, const char *sursa)

Copie şirul de caractere din zona de memorie a cărei adresă de început este dată de valoarea pointerului **sursa** în zona de memorie care urmează după ultimul caracter din şirul de caractere care are primul caracter memorat la adresa indicată de pointerul **dest**. Se presupune că pentru zona de memorie indicată de pointerul sursa s-a alocat suficientă memorie (prin alocare dinamică sau prin rezervare statică printr-un tablou de caractere) pentru a memora caracterele din cele 2 siruri şi caracterul '\0' de la final. Funcţia returnează un pointer a cărui valoare este adresa de început a zonei de memorie unde se face copierea, adică **dest**.

Exemplu:

```
char tab1[100]="primul sir";
char tab2[]="al doilea sir";
strcat(tab1, " ");
strcat(tab1, tab2);
```

În acest moment tab1 va deveni: "primul sir al doilea sir".

Dacă se doreşte să se ia doar primele n caractere de la sursă se foloseşte funcţia:

char * strncat(char *dest, const char *sursa, unsigned n)

Şirul rezultat nu are ‘\0’ la final.

2.4. Compararea a două şiruri de caractere

Şirurile de caractere se pot compara pe baza codurilor ASCII ale caracterelor ce intră în componenţa lor (comparaţia alfabetică sau lexicografică).

Fie :

char *s1,*s2;

s1=s2 dacă au lungimi egale şi s1[i]==s2[i] oricare ar fi i=0 ... strlen(s1)
s1<s2 dacă exista un i pentru care s1[i]<s2[i] si s1[j]==s2[j] oricare ar fi j=0 ... (i-1)
s1>s2 dacă exista un i pentru care s1[i]>s2[i] si s1[j]==s2[j] oricare ar fi j=0 ... (i-1)

Funcţii de comparare:

int strcmp(const char *s1. const char *s2)

Returnează o valoare <0 dacă s1<s2 (s1 este înaintea lui s2 la o ordonare alfabetică). valoarea 0 dacă s1=s2 (cele două şiruri coincid şi o valoarea >0 dacă s1>s2 (s1 este după s2 la o ordonare alfabetică).

Exemplu:

strcmp("ab", "ab") returnează 0
strcmp("aab", "abb") returnează un număr negativ
strcmp("za", "z") returnează un număr pozitiv
strcmp("a", "A") returnează un număr pozitiv

deoarece codul ASCII al lui ‘a’ > codul ASCII al lui ‘A’

int stricmp(const char *s1, const char *s2)

Are acelaşi efect ca la funcţia **strcmp**, dar nu se face diferenţa între litere mari şi mici.

Exemplu:

stricmp("a","A") returnează 0

int strncmp(const char *s1, const char *s2, unsigned n)

Foloseşte doar **n** caractere pentru comparaţie; în cazul în care **n** este mai mare decât minimul dintre cele două lungimi funcţionează la fel cu **strcmp.**

2.4. Alte funcţii utile de lucru cu şiruri de caractere

char *strrev(char *s)

Inversează caracterele din şirul de caractere cu excepţia caracterului terminator de şir '\0'. Returnează un pointer la şirul inversat.

char *strlwr(char *s)

Converteşte literele mari în litere mici. celelalte caractere rămân neschimbate. Returnează pointer la şirul modificat.

char *strupr(char *s)

Converteşte literele mici în litere mari. celelalte caractere rămân neschimbate. Returnează pointer la şirul modificat.

char *strdup(const char *s)

Copie un şir de caractere într-un spaţiu obţinut prin apel de genul malloc. În cazul în care funcţia nu poate face alocare. ea returnează. ca şi **malloc**. un pointer nul.

Spaţiul alocat are dimensiunea (strlen(s)+1) (unde **s** este şirul care trebuie copiat) şi este responsabilitatea utilizatorului să elibereze spaţiul alocat de funcţia **strdup**.

De exemplu. secvenţa următoare:

```
char *s="Sir exemplu";
char *x;
x = strdup(s);
if(x == 0) {
      fprintf(stderr. "Memorie insuficienta\n");
      exit(EXIT_FAILURE):
}
```

are acelaşi efect cu:

```
char *s="Sir exemplu";
char *x;
x=(char*)malloc((strlen(s)+1)*sizeof(char));
if(x == 0) {
      fprintf(stderr. "Memorie insuficienta\n");
      exit(EXIT_FAILURE):
}
strcpy(x,s);
```

char *strchr(const char *s, int c)

– parcurge un şir de caractere (s) în căutarea primei apariţii a unui caracter (c);

– '\0' se consideră ca făcând parte din şir, deci strchr(s, 0) returnează un pointer la caracterul terminator de şir al lui s;

– returneaza un pointer la prima apariţie a caracterului c în s. NULL dacă caracterul căutat nu există în şir.

char *strrchr(const char *s, int c)

– parcurge un şir de caractere în direcţie inversă, căutând ultima apariţie a caracterului c (poate fi 0 sau '\0') în şirul s;

– returnează un pointer la ultima apariţie a caracterului c în s. NULL dacă nu există.

size_t strcspn(const char *s1, const char *s2)

– parcurge un şir de caractere (s1) pentru a separa segmentul de la început care nu conţine nici un caracter care face parte din al doilea şir de caractere (s2);

– returnează lungimea segmentului iniţial din s1 compus din caractere ce nu fac parte din s2;

size_t strspn(const char *s1, const char *s2)

– parcurge un şir de caractere (s1) atât timp cât fiecare caracter face parte şi din s2;

– returnează lungimea segmentului iniţial din s1 compus din caractere ce fac parte şi din s2;

char *strset(char *s, int ch)

– toate caracterele din s vor deveni ch;

– returneaza s;

char *strstr(const char *s1, const char *s2)

– caută prima apariţie a şirului s2 în interiorul lui s1;

– returnează un pointer la elementul din s1 unde începe s2. NULL dacă s2 nu apare ca subşir al s1;

char *strpbrk(const char *s1, const char *s2)

– caută în şirul s1 prima apariţie a unui caracter din s2;

– returnează un pointer la prima apariţie a vreunui caracter din s2 în s1 și zero dacă nu există;

char *strtok(char *s1, const char *s2)

– funcţia se foloseşte de regulă pentru descompunerea şirului s1 în subşiruri delimitate de caractere din şirul s2;

– funcţia interpretează şirul s1 ca fiind alcătuit din subşiruri, delimitate de caractere (indiferent care din ele) conţinute în şirul s2;

– la primul apel al funcţiei strtok, aceasta întoarce un pointer pe primul subşir care nu are nici un caracter din s2, subşir căruia i se adaugă '\0' la sfârşit. La al doilea apel, funcţia trebuie sa aibă primul argument NULL (în locul lui s1). Ea continuă separarea a ceea ce a rămas din s1, întorcând pointeri pe subşirurile astfel obţinute si punând '\0' la sfârşitul lor, până la epuizarea lui s1. Dacă şirul s2 este ales convenabil, ca un delimitator pentru subşirurile din s1, descompunerea este bună.

Exemplu:

```
char *s;
s=strtok("abc  ab, bc.aaabccde"," .,");
printf("\nPrimul subşir: %s",s);
do
{
        s=strtok(NULL," .,");
        printf("\nUrmatorul subşir:%s",s);
} while(s!=NULL);
```

Vor fi separate subşirurile:

```
abc
ab
bc
aaabccde
```

void swab(char *sursa, char *dest. int nr);

– funcţia copie nr octeţi de la sursa la dest, inversând octeţii de la adresele pare cu cei de la cele impare;

– nr trebuie sa fie par;

– este utilă pentru transferarea datelor între două calculatoare cu memorare diferită (octetul cel mai semnificativ dintr-un cuvânt la adresa superioară sau la cea inferioară);

Atenţie: ordinea octeţilor afectează organizarea câmpurilor de biţi care depăşesc un octet !

Exemplu:

```
...
char sir[]="1234567890";
char s[15];
lung=strlen(sir);
swab(sir.s.lung)
s[lung]='\0';
puts(s);
...
```

Pe monitor va apare:2143658709

3. Funcţii de conversie a şirurilor de caractere

Au prototipul în stdlib.h

int atoi (const char* şir)

– converteşte un şir de caractere într-un număr întreg

– şirul de caractere trebuie să fie de forma:

[ws][sn][ddd]

unde :

[ws] este un şir opţional de spaţii sau tab-uri

[sn] este un semn opţional + sau -

[ddd] un şir de cifre zecimale

– primul caracter care nu satisface conversia duce la terminarea funcţiei

– dacă rezultă un număr care iese din gama pentru tipul **int** rezultatul este nedefinit;

– returnează rezultatul conversiei sau 0 dacă şirul de caractere nu poate fi convertit într-un număr corespunzător tipului de data **int** (ATENŢIE: nu se poate separa cazul corect al şirului de caractere "0" de eroare !)

long atol(const char * şir)

– converteşte un şir de caractere (de forma prezentată mai sus) într-un număr long;

– în caz de depăşire rezultatul este nedefinit

– conversia se opreşte la primul caracter ilegal

– returnează valoarea în care a fost convertit şirul de intrare sau 0 dacă numărul nu poate fi convertit într-un număr corespunzător tipului **long**.

double atof(const char* şir)

– converteşte şirul de caractere într-un număr real dublă precizie (tipul double)

– şirul trebuie să fie de forma :

[whitespaces][sign][ddd][.][ddd][e|E[sign]ddd]

– primul caracter necunoscut duce la sfârşitul conversiei

– recunoaşte +INF . -INF (+/- infinit) şi +NAN sau -NAN (not a number)

– returnează valoarea convertită a şirului de intrare; dacă se produce depăşire returnează +/- HUGE_VAL

char *itoa(int valoare. char* şir, int baza)

– converteşte un întreg (valoare) într-un şir de caractere

– baza specifica baza de numeraţie în care se consideră a fi acel întreg ; daca valoarea este negativa şi baza=10, primul caracter din şir va fi semnul '-'

– spaţiul alocat pentru şir trebuie sa fie suficient de mare pentru a memora şirul rezultat inclusiv '\0' final; itoa poate produce un şir de până la 17 octeţi

– returnează un pointer la char (deci un şir)

char *ltoa(long valoare, char *şir, int baza)

– converteşte un long (valoare) în şir de caractere

– lucrează asemănător cu itoa; şirul rezultat poate avea până la 33 de caractere

char *ultoa(unsigned long valoare, char *şir, int baza)

– converteşte un unsigned long într-un şir de caractere (poate rezulta de maxim 33 octeti)

Alte funcţii de conversie sunt:

– ecvt, fcvt, gcvt (pentru conversie din număr real în şir de caractere)

– strtod, strtol, strtoul (conversie din şir în double, long respectiv unsigned long)

4. Terminarea Programelor

void exit(int status);

– terminarea programului curent;

– status poate avea valori cuprinse între 0 şi 255 (dacă se indică valori mai mari. se face împărţirea modulo 255);

– înainte de terminare. toate fişierele sunt închise şi bufferele golite

– aceleaşi acţiuni se petrec şi la terminarea normală a programului chiar dacă nu se apelează exit. codul de retur fiind însă o valoare aleatoare.

void _exit(int status);

– termină execuţia fără a se închide fişierele, fără să goleasca buffer-ele de ieşire şi fără să apeleze eventualele exit functions stabilite prin atexit();

– parametrul status are aceeaşi semnificaţie ca la exit().

void abort(int test)

– scrie un mesajul de sfârşit: **Abnormal program termination** la dispozitivul standard de erori (stderr) după care abandonează programul apelând _exit(3).

TEMA 1

Problema nr. 1.1.

Folosind structura COMPLEX dată în referat să se scrie funcţii pentru:

- adunarea a două numere complexe (funcţia primeşte ca parametri doi pointeri la structura COMPLEX şi returnează un pointer la structura COMPLEX);
- înmulţirea a două numere complexe (funcţia primeşte ca parametri doi pointeri la structura COMPLEX şi returnează un pointer la structura COMPLEX);
- conjugatul unui număr complex (funcţia primeşte ca parametru un pointer la structura COMPLEX şi returnează un pointer la structura COMPLEX);
- modulul unui număr complex (funcţia primeşte ca parametru un pointer la structura COMPLEX şi returnează un real în dublă precizie);

Să se scrie un program **C** care citeşte de la tastatură două numere complexe z1 şi z2 şi calculează expresiile:

c1 = 2*z1 + 3.2*z2;

c2 = (3.5*z1+ 1)*z2;

c3 = z1 / z2;

Indicaţie: dacă $z_1 = a_1 + ib_1$. $z_2 = a_2 + ib_2$ şi $z = a + ib$ atunci

$$z_1 + z_2 = (a_1 + a_2) + i(b_1 + b_2)$$

$$z_1 \cdot z_2 = (a_1 a_2 - b_1 b_2) + i(a_1 b_2 + a_2 b_1)$$

$\bar{z} = a - ib$ (conjugatul numărului complex *z*)

$$|z| = \sqrt{a^2 + b^2}$$

$$\frac{1}{z} = \frac{\bar{z}}{|z|^2}$$

Problema nr. 1.2

Să se scrie un program care testează doua funcţii proprii de conversie a datei:

- dacă se dă ziua. luna şi anul să se calculeze ziua din an (a câta zi din an este) funcţia **f1**

- dacă se dă ziua din an şi anul. să se determine ziua şi luna care îi corespund.- funcţia **f2**

Indicaţie:
Se va folosi o structura de forma

```
typedef struct   _DATA {
   int zi, luna, an, zian;
 } DATA;
```

Se va folosi, de asemenea, un tablou pentru numărul de zile din cele 12 luni ale anului. Se va ţine cont şi dacă anul este bisect (se pot declara 2 tablouri, pentru an bisect /nebisect).

Funcţiile vor folosi ca argument un pointer la strcura DATA. Prototipurile celor două funcţii sunt :

```
void f1(DATA *d);
void f2(DATA *d);
```

Schematic funcţia **main** va arăta astfel:

```
int main(void)
{
      DATA d;
      // Atenţie: in main structura NU este declarată ca pointer
      ..................
      f1(&d);
      ..................
      f2(&d);
      ..................
      return 0;
}
```

citirea informaţiilor şi afişarea rezultatelor se vor face în funcţia **main** şi NU în funcţiile **f1** sau **f2**.

Problema nr. 1.3

Să se definească tipul de dată VECTORR (asociat unui vector cu elemente reale) ca o structură care cuprinde:

- numele vectorului memorat prin intermediul unui pointer (numele este format din mai multe caractere);
- numărul de elemente din vector;
- elementele vectorului (de tip real) stocate într-o zonă de memorie alocată dinamic (printr-un pointer)

Să se scrie un program care pentru un vector

1. citeşte de la tastatură numele vectorului (unul sau mai multe caractere);
2. stochează toate informaţiile referitoare la vector într-o structură de tip VECTORR. citind de la tastatură numărul de elemente şi elementele vectorului;
3. afişează vectorul citit (fiecare element cu câte 3 zecimale);

4. afişează un meniu care dă posibilitatea utilizatorului să aleagă una din următoarele prelucrări:

a) ordonarea crescătoare a elementelor vectorului în funcţie de valoare

b) ordonarea crescătoare a elementelor vectorului în funcţie de suma cifrelor părţii întregi a modulului fiecărui element (de exemplu dacă vectorul are elementele

19.3. 42.2. 75.7. -111.9. 22.0

atunci rezultatul acestei ordonări este 111 (are suma 3). 22 (cu suma 4). 42 (cu suma 6). 19 (cu suma 10) 75 (cu suma 12)

Programul poate face o singură prelucrare în funcţie de opţiunea utilizatorului şi va trebui scris astfel încât să permită prelucrarea mai multor seturi de date.

Barem de notare

1.Încărcarea informaţiilor într-o structură de tip VECTORR (funcţia nu primeşte nici un parametru şi returnează o structură de tip VECTORR)	1.0
2. Alocarea corectă de spaţiu de memorie pentru pointeri (nume şi vector)	1.0
3. Citirea elementelor vectorului (într-o funcţie care primeşte ca parametru un număr întreg şi returnează un pointer la real)	1.0
4. Afişarea pe monitor a vectorului citit (funcţia are un parametru – un pointer la o structură de tip VECTORR şi nu returnează nimic)	0.5
5. Scrierea meniului de prelucrare (funcţia nu are nici un parametru şi returnează un număr care reprezintă numărul opţiunii de prelucrare)	0.5
6. posibilitatea de reluare a programului (prelucrarea mai multor seturi de date)	0.5
7. Folosire proiect (corect)	0.5
8. Fişier header (corect şi complet)	0.4
8a. Definirea corectă a structurii de tip VECTORR	0.1
9. Funcţia main (complet – inclusiv eliberarea corectă a zonelor de memorie folosite)	1.0
10. Interschimbarea valorilor a două numere reale (funcţia primeşte ca parametri doi pointeri la real şi nu returnează nimic) – funcţie folosită în funcţiile de ordonare	0.5
11. Ordonarea crescătoare (prin metoda bulelor) (funcţia primeşte ca parametru un pointer la o structură de tip VECTORR şi nu returnează nimic)	1.0
12. Ordonarea (prin metoda bulelor) în funcţie de suma cifrelor elementelor (funcţia primeşte ca parametru un pointer la o structură de tip VECTORR şi nu returnează nimic)	1.0
13. Calcularea sumei cifrelor unui număr întreg (funcţia primeşte ca parametru numărul întreg pozitiv şi returnează un număr întreg pozitiv)	1.0
TOTAL TABEL	**10** p

TEMA 2

Problema nr. 2.1

Scrieţi doua funcţii care să extragă un subşir dintr-un şir:

a) una cu prototipul:

char * substr(char *sir. int start. int stop)

Exemplu:

substr("abcdef".0.2) să returneze "abc"

substr("abcdef".1.5) să returneze "bcdef"

b) şi cealaltă cu prototipul:

char *str(char *sir.int start. int nr_car)

Exemplu:

str("abcdef".0.2) să returneze "ab"

str("abcdef".2.3) să returneze "cde"

Scrieți un program care să integreze cele două funcții.

Problema nr. 2.2

Scrieţi un program care citeşte de la tastatură o linie de text formată din mai multe cuvinte (secvenţe de caractere separate prin caractere albe. punct. virgulă. semnul întrebării. semnul exclamării).

Se cere să se afişeze numărul de cuvinte de pe linie şi afişarea tuturor cuvintelor din linia respectivă (fiecare pe câte o linie). Se va folosi funcţia **strtok** (și pentru exercițiu modelul dat îin referat – pag. 9).

Problema nr. 2.3

Se consideră un tablou de structuri de tip PERSONAL:

```
struct PERSONAL{
    char *nume;
    char *prenume;
    int virsta;
}tab_pers[20];
```

Să se scrie un program care realizează următoarele:

- citeşte datele pentru n persoane; n cerut de la tastatură;
- afişează datele pentru aceste n persoane intr-un tabel;
- afişează toate persoanele care au vârsta sub 30 ani;
- afişează toate persoanele ordonate alfabetic după nume.

Problema nr. 2.4

Să se citească de la tastatură un şir de caractere ce reprezintă un număr întreg şi baza de numeraţie a acestuia (între 2 şi 16), notată **baza_1**. Să se convertească acest şir

într-un număr reprezentat într-o altă bază de numeraţie (notată **baza_2**) cerută tot de la tastatură.

Exemplu:

Se introduc:

-sirul:"1111". baza_1 este 2

-baza_2 sa fie 16

Rezultatul trebuie să fie F. Dacă baza_2 este 10 rezultatul va fi 15. dacă baza_2 va fi 8 rezultatul va fi 17.

Sugestie: se poate trece mai întâi din baza_1 în baza 10. iar apoi din baza 10 în baza _2.