TABLOURI DE POINTERI.
FUNCȚII SPECIFICE DE LUCRU PE ŞIRURI DE CARACTERE

1. Tablouri de pointeri

Declarația:

```
T1* tab[100];
```

defineşte un tablou de pointeri în care fiecare element este un pointer la tipul **T**.

Pentru vectorul de pointeri **tab** avem următoarea reprezentare schematică:

Fiecare element al tabloului fiind un pointer trebuie alocat spațiu de memorie pentru el (nu mai mult decât este necesar), iar la terminarea lucrului în zona de memorie alocată, memoria trebuie eliberată..

Tablourile de pointeri sunt folosite când se lucrează cu mai multe zone de memorie, de dimensiuni diferite, care trebuie să stocheze elemente de acelaşi tip (aşa cum se lucrează cu vectori de întregi, de exemplu, când trebuie prelucrate mai multe date de tip întreg).

Exemplu cel mai sugestiv este stocarea unui text cu un anumit număr de linii. În acest caz, fiecare linie poate avea o lungime diferită de a celorlalte linii. Deoarece stocarea unui şir de caractere se face prin intermediul unui vector de caractere (sau pointer la caracter), stocarea unui text (în care fiecare linie este privită ca un şir de caractere) se poate face prin intermediul unui tablou de pointeri la **char**.

2. Funcții 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 locației de memorie care conține codul ASCII al caracterului **A**

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

...

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

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

....

Declarația 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 operații cu şiruri de caractere se referă la :

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

Prototipurile funcțiilor 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 componența şirului respectiv (fără '\0' final). Prototipul funcției care returnează lungimea unui şir de caractere este:

unsigned strlen(const char *s);

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

Exemplu:

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

Se poate folosi şi : n=strlen("acesta este un sir");

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

Prototipul funcției care realizează copierea este:

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

Această funcție realizează copierea şirului de caractere a cărui adresă de început este conținută 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 funcția modifică zona de memorie corespunzătoare; în schimb **sursa** este declarată cu const. Funcția 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 octeți pentru destinație 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 funcția 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

Funcția 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. Funcția 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 funcția:

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 componența lor (comparația 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)

Funcții 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 funcția **strcmp**, dar nu se face diferența 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 comparație; în cazul în care **n** este mai mare decât minimul dintre cele două lungimi funcționează la fel cu **strcmp.**

2.4. Alte funcții utile de lucru cu şiruri de caractere

char *strrev(char *s)

Inversează caracterele din şirul de caractere cu excepția 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 spațiu obținut prin apel de genul malloc. În cazul în care funcția nu poate face alocare, ea returnează, ca şi **malloc**, un pointer nul.

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

De exemplu, secvența 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 apariții 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 apariție 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 direcție inversă, căutând ultima apariție a caracterului c (poate fi 0 sau '\0') în şirul s;

– returnează un pointer la ultima apariție 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 conține nici un caracter care face parte din al doilea şir de caractere (s2);

– returnează lungimea segmentului inițial 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 inițial 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 apariție 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 apariție a unui caracter din s2;

– returnează un pointer la prima apariție a vreunui caracter din s2 în s1, NULL dacă nu există;

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

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

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

– la primul apel al funcției strtok, aceasta întoarce un pointer pe primul subşir care nu are nici un caracter din s2, subşir căruia ți adaugă '\0' la sfârşit. La al doilea apel, funcția 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 obținute 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("abcxxabxxbcxaaabccde","xx");
printf("\nPrimul subşir: %s",s);
do
{
        s=strtok(NULL,"xx");
        printf("\nUrmatorul subşir:%s",s);
} while(s!=NULL);
```

Vor fi separate subşirurile:

```
abc
ab
bc
aaabccde
```

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

– funcția copie nr octeți de la sursa la dest, inversând octeții 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);

Atenție: ordinea octeților afectează organizarea câmpurilor de biți 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. Funcții 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 opțional de spații sau tab-uri

[sn] este un semn opțional + sau -

[ddd] un şir de cifre zecimale

– primul caracter care nu satisface conversia duce la terminarea funcției

– 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** (ATENȚIE: 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 numerație în care se consideră a fi acel întreg ; daca valoarea este negativa şi baza=10, primul caracter din şir va fi semnul '-'

– spațiul 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 octeți

– 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 funcții 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 împărțirea modulo 255);

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

– aceleaşi acțiuni 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 execuția fără a se închide fişierele, fără să golească buffer-ele de ieşire, şi fără să apeleze eventualele exit functions stabilite prin atexit();

– parametrul status are aceeaşi semnificație 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

Problema nr. 1

Scrieți un program care citeşte dintr-un fişier un text format din cel mult 100 de linii, depune liniile de text în zone de memorie alocate dinamic (alocând spațiul de memorie strict necesar), ordonează textul alfabetic sau după lungimea liniilor (în funcție de opțiunea utilizatorului) şi afişează textul ordonat. Exemplu (pentru ordonarea alfabetică) textul:

zzz
zabcs
abc
aaaaaaa

ordonat alfabetic este:

aaaaaaa
abc
zabcs
zzz

Problema nr. 2

Scrieți un program care citeşte de la tastatură un text format din cel mult 50 de linii, depune liniile în zone de memorie alocate dinamic (alocând spațiul de memorie strict necesar). Fiecare linie este formată din mai multe cuvinte (secvențe de caractere separate prin caractere albe, punct, virgulă, semnul întrebării, semnul exclamării).

Se cere să se afişeze numărul de cuvinte de pe fiecare linie, iar pentru o linie (indicată de utilizator) se cere afişarea tuturor cuvintelor din linia respectivă (fiecare pe câte o linie). Se va folosi funcția **strtok**.

Problema nr. 3

Scrieți doua funcții 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 citeşte un text format din cel mult 50 de linii de la tastatură, depune liniile în zone de memorie alocate dinamic (alocând spațiul de memorie strict necesar). Din fiecare linie din textul astfel citit se extrage un subşir folosind una din cele două funcții scrise la punctul a sau b (la alegerea utilizatorului) şi afişează subşirurile extrase de pe fiecare linie.

Problema nr. 4

Să se citească de la tastatură un şir de caractere ce reprezintă un număr întreg şi baza de numerație a acestuia (între 2 şi 16). Să se convertească acest şir într-un număr, reprezentat într-o altă bază de numerație 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.

Problema nr. 5

Se citeşte de la tastatură un şir de forma: [ddd][.][ddd]. Se cere să se transforme acest şir în numărul real corespunzător (baza 10)

Problema nr. 6

Scrieți şi testați o funcție:

char *elim(char *a, const char *b)

care elimină din şirul de caractere **a** toate subşirurile de forma **b**.

Problema nr. 7

Scrieți si testați o funcție :

int last(const char *a, const char *b)

care să returneze indicele ultimei apariții a şirului de caractere **b** ca subşir al lui **a** (funcția returnează valoarea -1 când şirul **b** nu apare deloc în şirul **a**).

Problema nr. 8

Scrieți un program care citeşte linii text de la tastatură, le depune în zone de memorie alocate dinamic după care afişează de câte ori apare un anumit şir de caractere (indicat de la tastatură) pe fiecare din liniile citite mai sus.

Se va scrie şi o funcție care să determine numărul de apariții ale unui subşir într-un şir de forma:

int nr_ap (char *şir, char *subşir)