Capitolul IB.08. Şiruri de caractere. Biblioteci standard

Cuvinte cheie
Şiruri de caractere, funcţii de intrare/ieşire şiruri de caractere,
biblioteca string, extragere atomi lexicali, funcţii stdlib, funcţii ctype,
erori, argumente în linia de comandă

IB.08.1. Şiruri de caractere în C

În limbajul C nu există un tip de date *şir de caractere*, deşi există constante şir. Reamintim că o constantă şir de caractere se reprezintă între ghilimele.

Definiţie:
Un şir de caractere este un vector de caractere terminat cu caracterul '\0'.

Există însă anumite particularităţi în lucrul cu şiruri faţă de lucrul cu alţi vectori.

Prin natura lor, şirurile pot avea o lungime variabilă în limite foarte largi, iar lungimea lor se poate modifica chiar în cursul execuţiei unui program ca urmare a unor operaţii cum ar fi concatenarea a două şiruri, ştergerea sau inserţia într-un şir .

Pentru simplificarea listei de parametri şi a utilizării funcţiilor pentru operaţii cu şiruri s-a decis ca fiecare şir memorat într-un vector să fie terminat cu un caracter numit *terminator de şir*, caracterul *\0* ce are codul ASCII egal cu zero, şi astfel să nu se mai transmită explicit lungimea şirului. Multe funcţii care produc un nou şir precum şi funcţiile standard de citire adaugă automat un octet terminator la şirul produs (citit), iar funcţiile care prelucrează sau afişează şiruri detectează sfârşitul şirului la primul octet zero.
Exemplu: Şirul de caractere *"Anul 2012"* ocupă 10 octeţi, ultimul fiind \0.

Există două posibilităţi de definire a şirurilor:

- **ca tablou de caractere;**

Exemple:

```
char sir1[30];
char sir2[10]="exemplu";

#define MAX_SIR 100
char s[MAX_SIR];
```

- **ca pointer la caractere (iniţializat cu adresa unui şir sau a unui spaţiu alocat dinamic). La definire se poate face şi iniţializare:**

Exemple:

```
char *sir3;  /* sir3 trebuie iniţializat cu adresa unui şir sau a unui
                spaţiu alocat pe heap */
sir3=sir1;                      // sir3 ia adresa unui şir static
sir3=sir1; ⇔ sir3=&sir1;  ⇔ sir3=&sir1[0];     // sunt echivalente
sir3=(char *)malloc(100);       // se aloca dinamic un spatiu pe heap
char *sir4="test";              / * sir2 este initializat cu adresa
                               sirului constant */
```

IB.08.2. Funcţiile de intrare/ieşire pentru şiruri de caractere sunt:

Funcţie	Exemple utilizare	Descriere	Rezultat
char * gets(char * s);	char sir[10]; gets(sir);	Citeşte caractere până la întâlnirea lui *Enter*; acesta nu se adaugă la şirul s. Plasează */0* la sfârşitul lui s. Obs: codul lui *Enter* e scos din buffer-ul de intrare.	Returnează adresa primului caracter din şir. Dacă se tastează CTRL+Z returnează *NULL*.
int puts(const char * s);	char sir[10]; puts(sir);	Tipăreşte şirul primit ca parametru, apoi *NewLine*, cursorul trecând la începutul rândului următor	Returnează codul ultimului caracter din şir sau *EOF* la insucces
int scanf("%s", char* s);	char sir[10]; scanf("%s",sir);	Citeşte caractere până la întâlnirea primului blanc sau *Enter*; acestea nu se adaugă la şirul *s*. Plasează *\0* la sfârşitul lui *s*. Codul lui blanc sau *Enter* rămân în buffer-ul de intrare	Returnează numărul de valori citite sau EOF în cazul în care se tastează *CTRL/Z*
int printf("%s", char*s);	char sir[10]; printf("%s",sir);	Tipăreşte şirul *s*.	Returnează numărul de valori tipărite sau EOF în cazul unei erori.

Citirea unei linii care poate include spaţii albe se va face cu *gets*. Citirea unui cuvânt (şir delimitat prin spaţii albe) se va face cu *scanf*.
Exemplu de citire şi afişare linii de text, cu numerotare linii:

```
char lin[128];                                  // linii de maxim 128 car
int nl=0;
while ( gets (lin) != NULL) {                   // citire linie in lin
     printf ("%4d ",++nl);                      // mareste numar linie
     printf (%d: %s \n", nl, lin);       // scrie numar şi conţinut linie
}
```

Nu se recomandă citirea caracter cu caracter a unui şir, cu descriptorul *%c* sau cu funcţia *getchar*, decât după apelul funcţiei *fflush*, care goleşte zona tampon de citire. În caz contrar se citeşte caracterul *\n* (cod 10), care rămâne în zona tampon după citire cu *scanf* sau cu *getchar*.

Pentru a preveni erorile de depăşire a zonei alocate pentru citirea unui şir se poate specifica o lungime maximă a şirului citit în funcţia *scanf*.
Exemplu:

```
char nume[30];
while(scanf("%30s",nume) != EOF) //numai primele 30 de caractere citite
  printf ("%s \n", nume);
```

Din familia funcţiilor de intrare-ieşire se consideră că fac parte şi funcţiile standard *sscanf* şi *sprintf*, care au ca prim argument un şir de caractere ce este analizat (*scanat*) de *sscanf* şi respectiv produs de *sprintf* (litera *s* provine de la cuvântul *string*). Aceste funcţii se folosesc fie pentru conversii interne în memorie, după citire sau înainte de scriere din/în fişiere text, fie pentru extragere de subşiruri dintr-un şir cu delimitatori diferiţi de spaţii.

Exemplu:

```
char d[]="25-12-1989";
int z, l, a;
sscanf(d, "%d-%d-%d", &z, &l, &a);       //z=25, l=12, a=1989
printf ("\n %d, %d, %d \n", z, l, a);    //25, 12, 1989
```

IB.08.3. Funcții standard pentru operații cu șiruri

Funcțiile standard pentru şiruri de caractere sunt declarate în fişierul *string.h*. Urmează o descriere puțin simplificată a celor mai folosite funcții:

Funcție	Descriere
• **char* strcpy(char *s1, const char *s2)** • **char* strncpy(char *s1, const char *s2, unsigned int n)**	• Copiază la adresa s1 tot şirul de la adresa *s2* (inclusiv terminator şir) • Copiază primele *n* caractere de la s2 la s1 Returnează s1
• **int strcmp(const char *s1, const char *s2)** • **int strncmp(const char *s1, const char *s2)**	• Compară şirurile de la adresele *s1* şi *s2* • Compară primele *n* caractere din şirurile *s1* şi *s2* Au următorul rezultat: • 0 dacă şirurile comparate conțin aceleaşi caractere (sunt identice) • < 0 dacă primul şir (*s1*) este inferior celui de al doilea şir (*s2*) • > 0 dacă primul şir (*s1*) este superior celui de al doilea şir (*s2*)
unsigned int strlen(const char *s)	Returnează lungimea sirului s, excluzând '\0'. char *msg="Hello"; strlen(msg); //5
• **char* strcat(char *s1, const char *s2)** • **char* strncat(char *s1, const char *s2, unsigned int n)**	• Adaugă şirul s2 la sfârşitul şirului *s1* • Adaugă primele *n* caractere de la adresa *s2* la şirul *s1*
char* strtok(char *s1, const char *s2)	Împarte s1 în tokeni – atomi lexicali, s2 conține delimitatorii
• **char * strchr (char *s, char c);** • **char *strrchr (char *s, char c);** • **char * strstr (char *s1, char *s2);**	• Returnează adresa lui *c* în şirul s (prima apariție a lui *c*) • Returnează adresa ultimei apariții a lui *c* în şirul s • Returnează adresa în şirul s1 a şirului *s2*
char * strdup (char *s);	Crearea unei copii a unui şir (alocă memorie şi copiază conținutul lui *s)*

Observații:

- Rezultatul funcției de comparare nu este doar -1, 0 sau 1 ci orice valoare întreagă cu semn, deoarece comparația de caractere se face prin scădere.

- Funcțiile standard *strcpy* şi *strcat* adaugă automat terminatorul zero la sfârşitul şirului produs de funcție! Funcția *strncpy* nu adaugă subşirului copiat la adresa *d* terminatorul de şir dacă *n* < *strlen(s)* dar subşirul este terminat cu zero dacă *n* > *strlen(s)*.

- Funcțiile pentru operații pe şiruri nu pot verifica depăşirea memoriei alocate pentru şiruri, deoarece primesc numai adresele şirurilor; cade în sarcina programatorului să asigure memoria necesară rezultatului unor operații cu şiruri.

- Funcțiile de copiere şi de concatenare întorc rezultatul în primul parametru (adresa şirului destinație) pentru a permite exprimarea mai compactă a unor operații succesive pe şiruri.
  Exemplu:

```
char fnume[20], *nume="test", *ext="cpp";
strcat( strcat( strcpy( fnume,nume),"."),ext);
```

- Utilizarea unor şiruri constante în operații de copiere sau de concatenare poate conduce la erori prin depăşirea memoriei alocate (la compilare) şirului constant specificat ca şir destinație. Exemplu:

```
     strcat ("test",".cpp");            // efecte nedorite !
```

- Funcția *strdup* alocă memorie la o altă adresă, copiază în acea memorie şirul *s* şi întoarce adresa noului şir. Se foloseşte pentru crearea de adrese distincte pentru mai multe şiruri citite într-o aceeaşi zona buffer.
  Exemplu:

```
int main () {
   char buf[40];
   int i=0, j;
   char* tp[100];       //tp este un tabel de pointeri la şiruri
   while (gets(buf))    //gets are rezultat zero la sfârşit de fişier
     tp[i++]= strdup(buf);   //copiaza şirul citit şi memoreaza adresa
   for (j=0; j<i; j++)  //afişarea şirurilor folosind tabelul de pointeri
     puts(tp[j]);
      return 0;
}
```

IB.08.4. Extragerea atomilor lexicali

Un cuvânt sau un atom lexical (*token*) se poate defini în două feluri:
- **un şir de caractere separat de alte şiruri prin unul sau câteva caractere cu rol de separator între cuvinte (de exemplu, spații albe);**
- **un şir care poate conține numai anumite caractere şi este separat de alți atomi prin oricare din caracterele interzise în şir.**

În primul caz sunt puțini separatori de cuvinte şi aceştia pot fi enumerați.

Pentru **extragerea de şiruri separate prin spații albe** (‘ ‘, ’\n’, ’\t’, ’\r’) se poate folosi o funcție din familia “scanf”:
- “fscanf” pentru citire dintr-un fişier
- “sscanf” pentru extragere dintr-un şir aflat în memorie

Între şiruri pot fi oricâte spații albe, care sunt ignorate.

int sscanf(const char **str*, const char **format*, ...);

Exemplu de funcție care furnizează cuvântul k dintr-un şir dat s:

```
char* kword (const char* s, int k) {
   char word[256];
```

```
   while ( k >= 1) {
     sscanf(s,"%s",word);       // extrage un cuvant din linie in word
     s=strstr(s, word)+ strlen(word);   // trece peste cuvantul extras
     k--;
   }
   return strdup(word);           // returneaza o copie a cuvantului k
}
```

Pentru **extragere de cuvinte separate prin câteva caractere** se poate folosi funcția de biblioteca *strtok*:

char *strtok(char *_str1_, const char *_str2_);

Exemplu care afişează atomii lexicali dintr-o linie de text:

```
char linie[128], * cuv;            // adresa cuvant in linie
char *sep=".,;\t\n "               // şir de caractere separator
gets(linie);                       // citire linie
cuv=strtok (linie,sep);            // primul cuvant din linie
while ( cuv !=NULL) {
     puts (cuv);                   // scrie cuvant
     cuv=strtok(0,sep);            // urmatorul cuvant din linie
 }
```

Funcția ***strtok*** are ca rezultat un pointer la următorul cuvânt din linie şi adaugă un octet zero la sfârşitul acestui cuvânt, dar nu mută la altă adresă cuvintele din text. Acest pointer este o variabilă locală statică în funcția *strtok*, deci o variabilă care îşi păstrează valoarea între apeluri succesive. Extragerea de cuvinte este o operație frecventă, dar funcția *strtok* trebuie folosită cu atenție deoarece modifică şirul primit, iar primul apel este diferit de următoarele apeluri.

In *stdlib.h* sunt declarate câteva funcții folosite pentru extragerea unor numere dintr-un text (dintr-un sir de caractere) şi care ilustrează o altă soluție pentru funcții care primesc o adresă într-un şir, extrag un subşir şi modifică adresa în cadrul şirului (după subşirul extras). Este vorba de funcțiile:

- ***double strtod*** (char *s, char **p); - string to double
- ***long strtod*** (char *s, char **p); - string to long

care extrag, la fiecare apel, un subşir care reprezintă un număr şi au două rezultate: valoarea numărului în baza zece şi adresa imediat următoare numărului extras în şirul analizat.

Exemplu de utilizare:

```
int main () {
            char * str =" 1   2.2   3.333      44e-1 ";
            char * p = str;
            double d;
            do {
                  d = strtod (p,&p);
                  printf ("%lf\n", d);
            } while (d != 0);
     return 0;
}
```

IB.08.5. Alte funcţii de lucru cu şiruri de caractere

Din biblioteca **stdlib:**

Funcţie	Descriere
`int atoi(char* s)`	Transformă un şir într-un int
`double atof(char* s)`	Transformă un şir într-un double

Din biblioteca **ctype:**

Funcţie	Descriere

`int isalpha(int ch)`
`int isdigit(int ch)`
`int isalnum(int ch)`
	Returnează non-zero (true) dacă ch este literă sau cifră;
	sau zero (false) altfel

`int isspace(int ch)`	Verifică dacă ch este spaţiu alb (Space, CR, LF, Tab, FF, VTab)
`int isupper(int ch)` `int islower(int ch)`	Verifică dacă ch este literă mare/mică
`int toupper(int ch)` `int tolower(int ch)`	Converteşte la literă mare/mică

IB.08.6. Erori uzuale la operaţii cu şiruri de caractere

O primă eroare posibilă este aceea că numele unui vector este un pointer şi nu mai trebuie aplicat operatorul & de obţinere a adresei în funcţia *scanf*, aşa cum este necesar pentru citirea în variabile simple.
O eroare de programare (şi care nu produce întotdeauna erori la execuţie) este utilizarea unei variabile pointer neiniţializate în funcţia *scanf* (sau *gets*).
Exemplu greşit:

```
char * s;              // corect este: char s[80];  80= lungime maxima
scanf("%s",s);         // citeşte la adresa conţinuta in s
```

O altă eroare frecventă (nedetectată la compilare) este compararea adreselor a două şiruri în locul comparaţiei celor două şiruri.
Exemplu:

```
char a[50], b[50];           // aici se memoreaza doua şiruri
scanf (%50s%50s", a,b);      // citire şiruri a şi b
if (a==b) printf("egale\n"); //greşit,rezultat zero
```

Pentru comparare corectă de şiruri se va folosi funcţia *strcmp*.
Exemplu:

```
if (strcmp(a,b)==0)  printf ("egale\n");
```

Aceeaşi eroare se poate face şi la compararea cu un şir constant.
Exemple:

```
if ( nume == "." ) break; ...}                  // greşit !
if ( strcmp(nume,".") == 0 ) break;... }        // corect
```

Din aceeaşi categorie de erori face parte atribuirea între pointeri cu intenţia de copiere a unui şir la o altă adresă, deşi o parte din aceste erori pot fi semnalate la compilare. *Exemple*:

```
char a[100], b[100], *c ;
a = b;      // eroare semnalata la compilare
c = a;      // corect sintactic dar nu copiaza şir (doar modifica  c)
c=strdup(a); // copiaza şir de la adresa a la adresa c, aloca mem pt c
strcpy (a,b); // copiaza la adresa a şirul de la adresa b
```

IB.08.7. Definirea de noi funcţii pe şiruri de caractere

Funcţiile standard pe şiruri de caractere din C lucrează numai cu adrese absolute (cu pointeri) şi nu folosesc ca rezultat sau ca argumente adrese relative în şir (indici întregi). De exemplu, funcţia *strstr* care caută într-un şir un subşir are ca rezultat un pointer: adresa în şirul cercetat a subşirului găsit. Parametrii de funcţii prin care se reprezintă şiruri se declară de obicei ca pointeri dar se pot declara şi ca vectori.

La definirea unor noi funcţii pentru operaţii pe şiruri programatorul trebuie să fie sigur de adăugarea terminatorului de şir la rezultatul funcţiei, pentru respectarea convenţiei şi evitarea unor erori. Pentru realizarea unor noi operaţii cu şiruri se vor folosi pe cât posibil funcţiile existente.

Deoarece nu există funcţii care să elimine un caracter dintr-un şir, care să insereze un caracter într-un şir sau care să extragă un subşir dintr-o poziţie dată a unui şir, le vom defini în continuare:

```
// şterge n caractere de la adresa "d"
char * strdel ( char *d, int n) {
   if ( n < strlen(d)) {
     char *aux=strdup(d+n);
      strcpy(d,aux);
   }
   return d;
}
```

```
// inserează şirul s la adresa d
void strins (char *d, char *s) {
   char *aux=strdup(d);
   strcpy(d,s);
   strcat(d,aux);
}
```

În general, nu se recomandă funcţii care au ca rezultat adresa unei variabile locale, deşi erorile de utilizare a unor astfel de funcţii apar numai la apeluri succesive (în cascadă).

Precizări la declararea funcţiilor standard sau nestandard pe şiruri :

- Parametrii sau rezultatele care reprezintă lungimea unui şir sunt de tip *size_t* (echivalent de obicei cu *unsigned int*) şi nu *int*, pentru a permite şiruri de lungime mai mare.
- Parametrii prin care se reprezintă adrese de şiruri care nu sunt modificate de funcţie se declară ***const* (*const char * str*)**, interpretat ca ***pointer la un şir constant (nemodificabil)***.

Exemplu:

```
size_t  strlen (const char * s);
```

Cuvântul cheie *const* în fata unei declaraţii de pointer cere compilatorului să verifice că funcţia care are un astfel de argument nu modifică datele de la acea adresă. Toate funcţiile de bibliotecă cu pointeri la date nemodificabile folosesc declaraţia *const*, pentru a permite verificări în alte funcţii scrise de utilizatori dar care folosesc funcţii de bibliotecă.

Exemple

1. Variante de implementare a funcţiilor de bibliotecă *strlen, strcmp, strcpy* şi *strcat*.

```
int strlen( char *s){
   int lg=0;
      while (s[lg]!='\0')
            lg++;
    return lg;
}

int strcmp( char *s1, char *s2){
    int i;
    for(i=0; s1[i] || s2[i]; i++)
       if (s1[i] < s2[i])
           return -1;
       else
         if (s1[i] > s2[i])
           return 1;
    return 0;
}

char *strcpy( char *d, char *s){
    int i=0;
    while(s[i]){
       d[i]=s[i];
             i++;
    }
    d[i]='\0';   // sau d[i]=0;
    return d;
}

// secvența ce cuprinde liniile cu verde este echivalentă cu:
//    while(d[i]=s[i]) i++;


char *strcat(char *d, char *s){
   int i=0,j=0;

   while(d[i]) i++;
   /*  la  iesirea  din  while,  i  este  indicele  caracterului  terminator*/
   while(d[i++]=s[j++]);
   return d;
}
```

2. Program care:

- citeşte cuvinte tastate fiecare pe câte un rând nou, până la CTRL/Z (varianta: până la introducerea unui cuvânt vid)
- afişează cuvântul cel mai lung
- afişează cuvintele ce încep cu o vocală

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define LUNG 81    //lungime maxima cuvant
#define NR 15       // nr max de cuvinte citite

void citire_cuv ( char tab_cuv[][LUNG], int *nr_cuv ) {
      printf( "Se introduc maxim %d cuvinte, terminate cu CTRL/Z:\n", NR );
      while(*nr_cuv<NR && gets ( tab_cuv[*nr_cuv] ) )
                  (*nr_cuv)++;
  /* la CTRL/Z gets returneaza NULL (= 0) */

  /* citirea se poate face şi cu scanf:
        while(*nr_cuv<NR && scanf("%s",tab_cuv[*nr_cuv])!=EOF) (*nr_cuv)++; */
```

```
  /* dacă terminarea se face cu un cuvant vid:
        while(*nr_cuv<NR && strcmp("",gets(tab_cuv[*nr_cuv]))) (*nr_cuv)++; */
}

void cuv_max ( char tab_cuv[][LUNG], int nr_cuv ){
  int i, lung_crt, lung_max=0;
  char * p_max;
  /* pointerul spre cuvantul maxim */
  /* se poate memora indicele cuvantului maxim: int i_max;
  sau memora cuvantul maxim intr-un şir: char c_max[LUNG]; */

  for(i=0;i<nr_cuv;i++)
    if ( ( lung_crt = strlen(tab_cuv[i]) ) > lung_max){
      p_max = tab_cuv[i];
      lung_max = lung_crt;
    }
  printf ("Cuvantul de lungime maxima %d este: %s\n", lung_max, p_max);
}

void cuv_vocale ( char tab_cuv[][LUNG], int nr_cuv ){
  int i;
  puts("Cuvintele ce incep cu vocale:");
  for(i=0;i<nr_cuv;i++)
      switch(toupper(tab_cuv[i][0])){
      case 'A': case'E': case 'I': case 'O': case 'U':
                  puts(tab_cuv[i]);
      }
   /* în loc de switch se putea folosi:
  char c;
  if(c=toupper(tab_cuv[i][0]),c=='A' || c=='E' || ...) puts(tab_cuv[i]); */
}

int main(){
  char tab_cuv[NR][LUNG];     //vectorul de cuvinte
  int nr_cuv=0;               // numarul cuvintelor introduse
  citire_cuv(tab_cuv,&nr_cuv);
  cuv_max(tab_cuv,nr_cuv);
  cuv_vocale(tab_cuv,nr_cuv);
  return 1;
}
```

3. Se citesc trei şiruri: s1, s2 şi s3. Să se afişeze şirul obținut prin înlocuirea în s1 a tuturor aparițiilor lui s2 prin s3. (Observație: dacă s3 este şirul vid, din s1 se vor şterge toate subşirurile s2).

```
#include <stdio.h>
#include <string.h>
#define N 81

int main(void){
  char s1[N],s2[N],s3[N],rez[N];
  char *ps1=s1,*pos, *r=rez;

  puts("sirul s1:"); gets(s1);
  puts("subsirul s2:"); gets(s2);
  puts("s3:"); gets(s3);

  while ( pos=strstr(ps1,s2) ){
    while(ps1<pos) *r++=*ps1++;          //copiez în r din s1 pana la pos
    strcpy(r,s3);                         //copiez în r pe s3
    r+=strlen(s3);                        //sar peste s3 copiat în r
```

```
    ps1+=strlen(s2);                    //sar în s1 peste s2
  }
  strcpy(r,ps1);                        //adaug ce a mai ramas din s1
  puts("sirul rezultat:");
  puts(rez);
  return 1;
}
```

IB.08.8. Argumente în linia de comandă

Funcția *main* poate avea două argumente, prin care se pot primi date prin linia de comandă ce lansează programul în execuție. Sistemul de operare analizează linia de comandă, extrage cuvintele din linie (şiruri separate prin spații albe), alocă memorie pentru aceste cuvinte şi introduce adresele lor într-un vector de pointeri (alocat dinamic).

Primul argument al funcției *main* este dimensiunea vectorului de pointeri (de tip *int*), iar al doilea argument este adresa vectorului de pointeri (un pointer).

Primul cuvânt, cu adresa în *argv[0]*, este chiar numele programului executat (numele fişierului ce conține programul executabil), iar celelalte cuvinte din linie sunt date inițiale pentru program: nume de fişiere folosite de program, opțiuni de lucru diverse.
Programele care preiau date din linia de comandă se vor folosi în acelasi mod ca şi comenzile predefinite ale sistemului (*DIR, TYPE*, etc.), deci extind comenzile utilizabile în linie de comandă.
Exemplu de afişare a datelor primite în linia de comandă:

```
// fisierul se numeste listare.c
int main ( int argc, char * argv[]) {   // sau :  char** argv
  int i;
  for (i=0;i<n;i++)                 // nu se afişeaza şi argv[0]
    printf ("%s ", argv[i]);
}
```

O linie de commandă de forma:
listare iata patru argumente
va lansa în execuție listare.exe, care va tipări pe ecran:
listare.exe
iata
patru
argumente