

Programarea calculatoarelor

Limbajul C



CURS 7



Șiruri de caractere



Șiruri de caractere

- Limbajul C nu definește tipul de data șir (string în Pascal)
- Un șir este un vector de caractere
- Ultimul caracter din șir este caracterul nul ('\0' care are codul ASCII egal cu zero).
- O constantă șir de caractere se reprezintă între ghilimele.
- Ex: "Anul 2001" ocupă 10 octeți de memorie, ultimul fiind '\0'.

Definire șiruri

Există două posibilități de definire a șirurilor:

■ ca tablou de caractere:

- `char șir1[30];`
- `char șir2[]="exemplu";`
- `#define MAX_SIR 100`
`char s[MAX_SIR];`

■ ca pointer la caractere:

- `char *sir3;`

Atenție la această definire!!

Definire șiruri

- **Atenție!** șir3 trebuie inițializat cu adresa unui șir sau a unui spațiu alocat pe heap (dinamic)
 - `șir3=șir1;`
șir3 ia adresa unui șir static
 - Echivalent cu:
 - `șir3=&șir1;`
 - `șir3=&șir1[0];`
 - `șir3=(char *)malloc(100*sizeof(char));`
se alocă dinamic un spațiu pe heap
 - `char *șir4="test";`
șir4 este inițializat cu adresa șirului constant

Alocarea dinamică a memoriei

- Funcțiile de gestionare a memoriei au prototipurile în fișierele header `alloc.h` și `stdlib.h`:
- `void * malloc(size_t size);`
 - returnează un pointer la blocul alocat pe heap de dimensiune `size` (în octeți), dacă un astfel de bloc există, altfel returnează `NULL`.
- `void free(void * block);`
 - eliberează blocul alocat anterior cu `malloc`, având adresa `block` transmisă ca parametru.

Funcții de prelucrare a șirurilor (din stdio.h)

- `char * gets(char * s);`
 - citește caractere până la întâlnirea caracterului Enter; acestea nu se adaugă la șirul s;
 - plasează `'\0'` la sfârșitul lui s;
 - returnează adresa primului caracter din șir;
 - dacă se tastează CTRL/Z returnează NULL;
 - codul lui Enter e scos din buffer-ul de intrare
- `int puts(char * s);`
 - tipărește șirul s, trece apoi la rând nou
 - întoarce o valoare nenegativă sau EOF la insucces

Funcții de prelucrare a șirurilor (din stdio.h)

- `scanf("%s",s);`
 - citește caractere până la întâlnirea primului blanc sau Enter; acestea nu se adaugă la șirul s;
 - plasează '\0' la sfârșitul lui s;
 - dacă se tastează CTRL/Z returnează EOF;
 - codul lui blanc sau Enter *rămân* în buffer-ul de intrare
- `printf("%s",s);`
 - tipărește șirul s

Observații

- Nu se recomandă citirea caracter cu caracter a unui șir (cu descriptorul “%c” sau cu funcția “getchar()”) decât după apelul funcției “fflush”
 - golește zona tampon de citire
 - în caz contrar se citește caracterul ‘\n’ (cod 10), care rămâne în zona tampon după citire cu “scanf(“%s”,..)” sau cu getchar().
- Pentru a preveni erorile de depășire a zonei alocate pentru citirea unui șir se poate specifica o lungime maximă a șirului citit în funcția “scanf”.
 - Exemplu:

```
char nume[30];  
while (scanf (“%29s”,nume) != EOF)  
    printf (“%s \n”, nume);
```
 - numai primele 29 de caract vor fi citite, pentru ca al 30-lea caracter va fi ‘\0’

Funcții de prelucrare a șirurilor (din string.h)

- `int strcmp(const char *s1, const char *s2);`
- `char *strcpy(char *d, const char *s);`
- `char* strncpy(char *d, const char *s, unsigned n);`
- `char *strdup(const char *s);`
- `int strlen(const char *s);`
- `char *strcat(char *d, const char *s);`
- `char *strncat(char *d, const char *s, unsigned n);`
- `char *strchr(const char *s, int c);`
- `char *strrchr(const char *s, int c);`
- `char *strstr(const char *s, const char *subsir);`

Funcții de prelucrare a șirurilor (din string.h)

■ int **strcmp**(const char *s1, const char *s2);

Returnează:

- <0, dacă s1 < s2
- 0, dacă s1 = s2
- >0, dacă s1 > s2

■ int **strncmp** (const char *s1,const char *s2,int n);

- comparare a două șiruri pe lungimea n

■ char ***strcpy**(char *d, const char *s);

- copiază șirul sursa s în șirul destinație d;
- returnează adresa șirului destinație

■ char* **strncpy**(char *d,const char *s,int n);

- copiază maxim n caractere de la sursă la destinație;
- returnează adresa șirului destinație

Funcții de prelucrare a șirurilor (din string.h)

- `int strlen(const char *s);`
 - returnează lungimea șirului fără a număra caracterul terminator
- `char* strcat(const char *d, const char *s);`
 - concatenează cele două șiruri și returnează adresa șirului rezultat
- `char* strchr(const char *s, char c);`
 - returnează poziția primei apariții a caracterului c în șirul s, respectiv NULL dacă c nu e în s
- `char* strstr(const char *s, const char *ss);`
 - returnează poziția primei apariții a șirului ss în șirul s, respectiv NULL dacă ss nu e în s.
- *Funcțiile standard "strcpy" și "strcat" adaugă automat terminatorul zero la sfârșitul șirului produs de funcție!*
- *Funcțiile pentru operații pe șiruri nu pot verifica depășirea memoriei alocate pentru șiruri, deoarece primesc numai adresele șirurilor; cade în sarcina programatorului să asigure memoria necesară rezultatului unor operații cu șiruri.*

Funcții de prelucrare a șirurilor (din string.h)

- `char *strdup(const char *s);`
 - Alocă memorie la o altă adresă și copiază în acea memorie șirul s
 - Intoarce adresa noului șir

```
char * strdup ( char * adr) {  
    int len=strlen(adr);    // lungime șir de la adresa adr  
    char * rez = (char*) malloc((len+1)*sizeof(char);  
    // alocă memorie pentru șir și terminator  
    strcpy (rez,adr);    // copiaza șir de la adr la adresa rez  
    return rez;    // rezultatul este adresa duplicatului  
}
```

const char*

- Argumentele ce reprezintă adrese de șiruri care nu sunt modificate de funcție
- Interpretat ca “pointer la un șir constant (nemodificabil)”.
- Cuvântul cheie *const* în fata unei declarații de pointer cere compilatorului să verifice că funcția care are un astfel de argument nu modifică datele de la acea adresă.
- Exemplu de funcție care produce un mesaj de eroare la compilare:

```
void trim (const char*s) { // elimina toate spatiile dintr-un șir dat
    char *p=s;           // avertisment la aceasta linie !
    .....
}
```

Observație

- Pentru realizarea unor noi operații cu șiruri se vor folosi pe cât posibil funcțiile existente:

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

```
// inserează șirul s la adresa d
void strins (char *d, char *s) {
    int ls=strlen(s);
    strcpy (d+ls,d);           // deplasare sigura la dreapta șir d
    strncpy(d,s,ls);         // adauga șirul s la adresa d
}
```

Extragere atomi lexicali

- Definiție cuvânt sau atom lexical (“token”):
 - un șir de caractere separat de alte șiruri prin unul sau câteva caractere cu rol de separator între cuvinte (de exemplu, spații albe);
 - un șir care poate conține numai anumite caractere și este separat de alți atomi prin oricare din caracterele interzise în șir.
- În primul caz sunt puțini separatori de cuvinte și aceștia pot fi enumerați.
- Pentru extragerea de șiruri separate prin spații albe (‘ ‘, ‘\n’, ‘\t’, ‘\r’) se poate folosi o funcție din familia “scanf”
 - “fscanf” pentru citire dintr-un fișier
 - “sscanf” pentru extragere dintr-un șir aflat în memorie
- Între șiruri pot fi oricâte spații albe, care sunt ignorate.

Funcția sscanf

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

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

```
char* kword (const char* s, int k) {  
    char word[256];  
    while ( k >= 0) {  
        sscanf(s, "%s", word);  
        s=s+strlen(word);  
        k--;  
    }  
    return strdup(word);  
}
```


Funcția strtok

char ***strtok**(char **str1*, const char **str2*);

- pentru extragere de cuvinte ce pot fi separate și prin alte caractere (',' sau ';' de ex.)
- are ca rezultat un pointer la următorul cuvânt din linie și adaugă un octet zero la sfârșitul 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 își păstrează valoarea între apeluri succesive.
- trebuie folosită cu atenție deoarece modifică șirul primit și nu permite analiza în paralel a două sau mai multe șiruri

Exemplu strtok

```
int main ( ) {
    char linie[128], * cuv;           // adresa cuvant în linie
    char *sep=".,;\t\n "            // șir 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
    }
    getchar();
    return 1;
}
```

Quiz

1. Care dintre secvențele de mai jos copiază în mod corect șirul s1 în șirul s2?

```
char *s1="Curs C", *s2;
```

- s1=s2;
- strcpy(s2,s1);
- niciuna

2. Există vreun caracter care nu poate face parte dintr-un șir?

- Da
- Nu

3. Ce se tipărește pe ecran?

```
#define L 15  
char șir[L];  
strcpy(sir,"123\0456");  
printf("Sirul copiat este: %s și are lungimea: %d\n",sir, strlen(sir));
```

- Sirul copiat este 123\0456 și are lungimea 8
- Sirul copiat este 123 și are lungimea 3
- Sirul copiat este 123456 și are lungimea 6

Exemple

1. Scrieți variante de implementare a funcțiilor de bibliotecă `strlen`, `strcmp`, `strcpy` și `strcat`.
2. Să se scrie un program care:
 - citește cuvinte tastate fiecare pe câte un rând nou, până la CTRL/Z (varianta: pana la introducerea unui cuvânt vid)
 - afișează cuvântul cel mai lung
 - afișează cuvintele ce încep cu o vocală
3. Se citesc trei șiruri `s1`, `s2` și `s3`. Să se afișeze șirul obținut prin înlocuirea în `s1` a tuturor aparițiilor lui `s2` prin `s3`. (Observație: dacă `s3` este șirul vid, din `s1` se vor șterge toate subșirurile `s2`).

Funcția strlen

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

Funcția strcmp

```
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;
}
```

Funcția strcpy

```
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 roșu este echivalentă cu:

```
while(d[i]=s[i]) i++;
```

Funcția strcat

```
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;
}
```


Rezolvare

```
#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 și 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)++; */
}
```

Rezolvare

```
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);
}
```

Rezolvare

```
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]); */
}
```

Rezolvare

```
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);

    getchar();
    return 1;
}
```

Rezolvare

```
#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);
```

Rezolvare

```
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);

getchar();
return 1;
}
```

Erori posibile la utilizarea șirurilor

- Utilizarea unei variabile pointer neinițializate în funcția “scanf” (sau “gets”), datorită confuziei dintre vectori și pointeri.
- Poate cea mai frecventă eroare de programare
- Nu se manifestă întotdeauna ca eroare la execuție
- Exemplu greșit:

```
char * s; // corect este: char s[80]; 80= lung. maximă  
scanf ("%s",s); // citește la adresa conținută în "s"
```

Erori posibile la utilizarea șirurilor

- Compararea adreselor a două șiruri în locul comparației celor două șiruri
 - eroare frecventă (nedetectată la compilare)
 - Exemplu:

```
char a[50], b[50];           // aici se memoreaza doua siruri
scanf ("%50s%50s", a,b);   // citire siruri a si b
if (a==b) printf("egale\n"); //gresit,rezultat zero
```
- Pentru comparare corectă de șiruri se va folosi funcția “strcmp”.
 - Exemplu :

```
if (strcmp(a,b)==0) printf ("egale\n");
```
 - Aceeași eroare se poate face și la compararea cu un șir constant.
Exemple:

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


Erori posibile la utilizarea șirurilor

- Atribuirea între pointeri cu intenția de copiere a unui șir la o altă adresă
- O parte din aceste erori pot fi semnalate la compilare.
- Exemple:

```
char a[100], b[100], *c ;  
    // memorie alocata dinamic la adresa "c"  
c = (char*) malloc(100);  
a = b; // eroare la compilare  
c = a; // corect sintactic dar nu copiaza sir (modifica "c")  
strcpy (c,a); // copiaza sir de la adresa "a" la adresa "c"  
strcpy (a,b); // copiaza la adresa "a" sirul de la adresa "b"
```