Programarea calculatoarelor
Limbajul C

CURS 7

Şiruri de caractere

Şiruri de caractere

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

Şiruri de caractere

Un caracter din şir poate fi:

- un *simbol grafic*
- *o secvență escape*
- *un cod ASCII (în octal* sau hexazecimal, dupa \;).

Exemple:

- 'A' '\x41' '\101' (sunt echivalente)
- caracterele \ ' " : **‘\\’ ‘\‘’ ‘\“’**
- un simbol fara echivalent grafic, dat ca secventa escape: '\n' '\xa' '\12' (sunt echivalente).

Definire şiruri

Există două posibilități de definire a şirurilor:

- ca tablou de caractere:
  - char şir1[30];
  - char şir2[]="exemplu";
  - #define MAX_SIR 100
    char s[MAX_SIR];
- ca pointer la caractere:
  - char *sir3;

  Atenție la această definire!! Trebuie inițializare!

Definire şiruri

- Atenție! şir3 trebuie inițializat cu adresa unui şir sau a unui spațiu alocat pe heap (dinamic)
  - sir3=sir1;

  sir3 ia adresa unui şir static
  - Echivalent cu:
    - sir3=&sir1;
    - sir3=&sir1[0];
  - sir3=(char *)malloc(100*sizeof(char));

  se alocă dinamic un spațiu pe heap!
  - char *sir4="test";

  sir4 este inițializat cu adresa şirului constant

Funcții de prelucrare a şirurilor (din stdio.h)

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

Funcții de prelucrare a şirurilor (din stdio.h)

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

Observații

- Nu se recomandă citirea caracter cu caracter a unui şir (cu descriptorul “%c” sau cu funcția “getchar()”) decât după apelul funcției “fflush”
  - goleşte zona tampon de citire
  - în caz contrar se citeşte caracterul ‘\n’ (cod 10), care rămâne în zona tampon după citire cu “scanf(“%s”,..) 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 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 şirurilor (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 şirurilor (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 şiruri pe lungimea n
- char *strcpy(char *d, const char *s);
  - copiază şirul sursa s în şirul destinatie d;
  - returnează adresa şirului destinatie
- char* strncpy(char *d,const char *s,int n);
  - copiază maxim n caractere de la sursă la destinație;
  - returnează adresa şirului destinație

Funcții de prelucrare a şirurilor (din string.h)

- int strlen(const char *s);
  - returnează lungimea şirului fără a număra caracterul terminator
- char* strcat(const char *d, const char *s);
  - concatenează cele doua şiruri şi returnează adresa şirului rezultat
- char* strchr(const char *s,char c);
  - returnează poziția primei apariții a caracterului c în şirul s, respectiv NULL dacă c nu e în s
- char* strstr(const char *s, const char *ss);
  - returnează poziția primei apariții a şirului ss în şirul s, respectiv NULL dacă ss nu e în s.

- *Funcțiile standard "strcpy" şi “strcat" adaugă automat terminatorul zero la sfârsitul şirului produs de funcție!*
- *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.*

Funcții de prelucrare a şirurilor (din string.h)

- char *strdup(const char *s);
  - Alocă memorie la o altă adresă şi copiază în acea memorie şirul s
  - Intoarce adresa noului şir

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

const char*

- Argumentele ce reprezintă adrese de şiruri care nu sunt modificate de funcție
- Interpretat ca “pointer la un şir 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 şir dat
        char *p=s;              // avertisment la aceasta linie !
    …..
}
```

Observație

- Pentru realizarea unor noi operații cu şiruri 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)){
      char *aux=strdup(d+n);
     strcpy(d,aux);
   }
   return d;
 }

 // inserează şirul s la adresa d
void strins (char *d, char *s) {
  char aux[20];
  strcpy(aux,d);          // creare copie şir care începe la adresa d
  strcpy(d,s);            // adauga şirul s la adresa d
  strcat(d,aux);          // concatenează la noul şir vechiul şir
}
```

Extragere atomi lexicali

- Definiție cuvânt sau atom lexical (“token”):
  - 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.
- In 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
- Intre şiruri 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 şir dat s:

```
char* kword (const char* s, int k) {
   char word[256];
   while ( k >= 0) {
     sscanf(s,"%s",word);
     s=s+strlen(word)+1;
     k--;
   }
   return strdup(word);
}
```

Funcția strtok

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

- pentru extragere de cuvinte ce pot fi separate şi prin alte caractere (‘,’ sau ’;’ de ex.)
- 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.
- trebuie folosită cu atenție deoarece modifică şirul primit şi nu permite analiza în paralel a două sau mai multe şiruri

Exemplu strtok

```
int main ( ) {
    char linie[128], * cuv;          // adresa cuvant în 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
    }
     return 0;
  }
```

Erori posibile la utilizarea şirurilor

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

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

Erori posibile la utilizarea şirurilor

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

```
char a[50], b[50];              // aici se memoreaza doua siruri
scanf (%49s%49s", a,b);         // citire siruri a si b
if (a==b) printf("egale\n");    //gresit,rezultat zero
```

- Pentru comparare corectă de şiruri se va folosi funcția "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; ...}              // gresit !
if ( strcmp(nume,".") == 0 ) break;... }    // corect
```

Erori posibile la utilizarea şirurilor

- Atribuirea între pointeri cu intenția de copiere a unui şir 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”
```

Exemple

1. Scrieți variante de implementare a funcțiilor de bibliotecă strlen, strcmp, strcpy şi strcat.

2. Să se scrie un program care:
   - citeşte cuvinte tastate fiecare pe câte un rând nou, până la CTRL/Z (varianta: pana la introducerea unui cuvant vid)
   - afişează cuvantul cel mai lung
   - afişează cuvintele ce incep cu o vocală

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).

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 roşu 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 2

```
#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)++; */
}
```

Rezolvare 2

```
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 2

```
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 0;
}
```

Rezolvare 3

```
#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 3

```
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 0;
}
```

Argumentele liniei de comandă

- La lansarea în execuție a unui fişier executabil, în linia de comandă, pe lângă numele fişierului se pot specifica argumente, care se transmit ca parametrii funcției main.
- Argumentele = date inițiale pentru program: nume de fişiere folosite de program, opțiuni diverse de lucru.
- Antetul funcției main va fi:

  int main(int argc, char ** argv)

- argc - numărul de argumente
  argv - adresa vectorului de pointeri la şiruri, reprezentând argumentele

Argumentele liniei de comandă

- Sistemul de operare
  - analizează linia de comandă
  - extrage cuvintele din linie (siruri separate prin spații albe)
  - alocă memorie pentru aceste cuvinte
  - introduce adresele lor într-un vector de pointeri (alocat dinamic).
- Argumentele liniei de comandă vor fi şirurile:

  argv[0] - numele fişierului executabil
  argv[1]
  ...
  argv[argc-1]

Exemplu: tipărirea argumentelor liniei de comandă

```
#include<stdio.h>
int main( int argc, char** argv){
        int i;
        puts ("Argumente:");
        for (i=0; i<argc; i++)
          puts (argv[i]);
        return 0;
}
```

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