# Programarea Calculatoarelor
# Curs 11

Iulian Năstac

Recap.

# Cap. **Matrici (Tablouri)**

- Tehnica tablourilor reprezintă o metodă comodă de stocare a matricilor în memoria unui sistem de calcul.
- **În programarea C,** o matrice (sau tablou) este o colecţie de variabile de acelaşi tip, oricare dintre elemente putând fi apelate cu acelaşi nume**.**
- C-ul utilizează principiul "Row Major“, adică elementele, de pe fiecare rând al unei matrici, sunt stocate succesiv în memorie (la adrese succesive raportate la dimensiunea tipului).

Recap.

# Declararea unei matrici

- Sintaxa:

  **tip Nume_matr [lim_1] [lim_2] … [lim_n];**

unde:

- *tip* → tipul elementelor matricei
- *lim_i* → limita superioară a indicelui al i-lea (ce poate avea valorile:0, 1, 2,… , lim_i – 1)

Recap.

# Legătura dintre pointeri şi matrici

- Numele unui tablou este un pointer deoarece el are ca valoare adresa primului său element.

- Există o diferenţă între numele unui tablou şi o variabilă de tip pointer. Astfel, unei variabile de tip pointer i se pot atribui valori la execuţie, în timp ce acest lucru nu este posibil să se realizeze pentru numele unui tablou (acesta are tot timpul ca valoare adresa primului său element).

- Se obişnuieşte să se spună că numele unui tablou este un ***pointer constant***.

Recap.

# Matrici unidimensionale (vectori)

- Vectorii reprezintă matricele (tablourile) cu o singură dimensiune.

- Format:

***tip nume_vect [lim];***

## Recap.

Dacă o funcţie primeşte o matrice unidimensională, parametrul său formal poate fi declarat în unul din următoarele 3 moduri:

- ca pointer;
- ca matrice cu dimensiune;
- ca matrice fără dimensiune.

Recap.

# Modificatorul *const*

- În limbajele *C* si *C++* există posibilitatea de a defini date constante folosind modificatorul *const* în declaraţii.
- Printr-o astfel de declaraţie, unui nume i se atribuie o valoare iniţială, care nu poate fi modificată ulterior printr-o altă expresie de atribuire (ca în cazul variabilelor).
- Deci o dată declarată cu ajutorul modificatorului *const,* o variabilă se consideră că este constantă.
- Reamintim că multe constante pot fi declarate cu ajutorul lui *#define*.

Recap.

# Şiruri de caractere în C

- Şirul de caractere reprezintă una din cele mai utilizate clase de aplicaţii în cadrul matricelor unidimensionale.

# Caracteristici ale şirurilor de caractere:

1) Un şir de caractere se păstrează într-o zonă de memorie organizată ca *tablou unidimensional* de tip ***char***.

2) Fiecare caracter se păstrează pe câte un octet prin codul său numeric.

3) Cel mai frecvent utilizat în acest scop este codul **ASCII**.

4) După ultimul caracter al şirului se păstrează caracterul **NUL** (sau *null*) care este '\0' ( şi are valoarea vidă).

5) Deoarece există *null* în poziţie finală, se declară matricele de tip caracter cu un caracter mai mult decât cel mai mare şir pe care îl conţine.

De exemplu, pentru a crea matricea sir care conţine cuvântul "hi", vom scrie:

```
char sir[3];

sir[0] = 'h';

sir[1] = 'i';

sir[2] = '\0';

printf("\n %s \n", sir);
```

# Caracteristici (continuare):

6) Deşi *C*-ul nu are explicit date de tip *şir*, el permite constante de tip *şir*. O constantă de tip *şir* este o listă de caractere închise între ghilimele (de exemplu “salut”).

7) Nu este necesar să fie introdus manual caracterul ***null*** la sfârşitul şirului de caractere (compilatorul o face automat).

De exemplu, dacă pentru formarea şirului se utilizează funcţia

*gets(şir);*

atunci **‘\0’** este introdus automat de către compilator.

8) Pentru a opera un şir de caractere se poate utiliza:

- *numele tabloului* (ca pointer constant spre şirul respectiv);
- *pointeri variabili* spre şirul de caractere.

# Exemplu:

*Dacă declarăm:*

***char tab[ ] =”Acesta este un sir”;***

*atunci:*

*tab* – adresa caracterului *A*
*tab+1* – adresa caracterului *c*
*tab+2* – adresa caracterului *e*
… *etc.*

sau:

tab[0] → codul ASCII al caracterului *A*
tab[1] → codul ASCII al caracterului *c*
… *etc.*

sau:

**tab* → codul ASCII al caracterului *A*
**(tab+1)* → codul ASCII al caracterului *c*
… *etc.*

# **Observaţie**:

Un efect similar se obţine cu declaraţia:

***char const *p =”Acesta este un sir”;***

*unde:*

***p[0]*** *sau* ****p*** *- conține codul ASCII al primului caracter (`A`)*

***p[1]*** *sau* ****(p+1)*** *- conține codul ASCII al celui de-al doilea caracter (`c`)*

*…*

# Funcţii standard utilizate în prelucrarea şirurilor de caractere

- Bibliotecile standard ale compilatoarelor C conţin o serie de funcţii care permit operaţii cu şiruri de caractere.

- Majoritatea acestor funcţii au prototipul în fişierul ***string.h***.

**Operaţiile pe care le execută funcţiile de prelucrare a şirurilor de caractere pot fi:**

- calculul lungimii şirurilor de caractere (ex.: *strlen*)
- copierea şirurilor de caractere (ex.: *strcpy*)
- concatenarea şirurilor de caractere (ex.: *strcat*)
- compararea şirurilor de caractere (ex.: *strcmp*)
- căutarea (identificarea) şirurilor sau caracterelor (ex.: *strcchr* sau *strstr*)

# 1. Calculul lungimii şirurilor de caractere

- Lungimea unui şir de caractere reprezintă numărul de caractere proprii care intră în compunerea şirului respectiv.

- Caracterul NULL nu este considerat la determinarea lungimii unui şir de caractere.

- Prezenţa lui *NULL* este necesară deoarece la determinarea lungimii unui şir se numără caracterele până la întâlnirea caracterului acestuia.

Sintaxa:

***unsigned strlen(const char *s);***

# Exemple:

1) *char * const p = ”Calcul lungime de sir”;*
*unsigned n;*
...
*n=strlen (p);*

Lui ***n*** i se va atribui valoarea 21.

2) *char tab [ ] = ”Calcul lungime de sir”;*
*int n;*
*n = strlen(tab);*

Parametrul ***n*** primeşte tot valoarea 21.

3) *int n;*
*n = strlen(”Calcul lungime de sir”);*

Variabila ***n*** primeşte aceeaşi valoarea 21.

# Observaţii:

- Cele trei exemple au rezultate identice.
- Parametrul formal al funcţiei ***strlen*** este un pointer spre o dată constantă.
- În consecinţă, funcţia ***strlen*** nu are voie să modifice caracterele şirului pentru care este determinată lungimea.

# 2. Copierea unui şir de caractere

- Uneori este necesar să se copieze un şir de caractere din zona de memorie în care se află, într-o altă zonă. Pentru aceasta se foloseşte funcţia ***strcpy*** care are prototipul:

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

- Funcţia copiază şirul de caractere spre care pointează **sursa** în zona de memorie a cărei adresă de început este valoarea lui ***dest***. Se copiază atât caracterele proprii şirului cât şi caracterul ***null*** de la sfarşitul şirului respectiv.
- Funcţia returnează adresa de început a zonei în care s-a transferat şirul (adică valoarea lui ***dest***).

# Exemple:

1) *char tab[ ] = “Sir de caractere”;*
*char t [ (sizeof tab)+1];* /*are acelaşi număr de elemente ca şi tab */
…
*strcpy (t, tab);*

2) *char t [100];*
*strcpy (t, “Sir de caractere”);*

3) *char * p=“Sir de caractere”;*
*char t [100];*
*char * q;*
*q=strcpy(t, p);*

# Variante ale funcției de copiere

- Pentru a copia cel mult n caractere ale unui şir dintr-o zonă în alta se foloseşte funcţia:

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

Dacă:

- *n* ≥ *lungimea şirului* - atunci toate caracterele şirului se transferă în zona spre care pointează *dest*;
- *n* < *lungimea şirului* - atunci se copiază numai primele n caractere ale şirului.

În rest funcţia este asemănătoare cu ***strcpy***.

# Exemplul 1

*char * p = “Transfer partial de sir”;*
*char t[9];*
*strncpy (t, p, (sizeof(t))-1);*

# Exemplul 2

- Scriem un program care citeşte o succesiune de cuvinte şi-l afişează în final pe cel mai lung dintre ele.

```
# include<stdio.h>
# include<string.h>
# define MAX 100     /* se presupune că un cuvânt nu are mai
mult de 100 caractere*/

int main( )
{
 int max=0, i;
 char cuvant[MAX+1];
 char cuvant_max[MAX+1];
 while (scanf("%100s", cuvant)!=EOF)
        if (max < (i = strlen(cuvant)))
              { max= i;
                 strcpy (cuvant_max, cuvant);
               }
 if (max) printf("\nCuvantul cel mai lung este %s  si are lungimea
%d \n",cuvant_max, max);
}
```

# 3. Concatenarea şirurilor de caractere

- Pentru concatenarea şirurilor de caractere se foloseşte funcţia ***strcat***.

- Formatul acestei funcţii este:

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

# Observaţii:

1) Funcţia copiază şirul de caractere din zona spre care pointează sursa, în zona de memorie care urmează imediat după ultimul *caracter propriu* al şirului spre care pointează *dest*.

2) Funcţia returnează valoarea lui *dest*.

3) Se presupune că zona spre care pointează *dest* este suficientă pentru a păstra caracterele proprii ale celor două şiruri care se concatenează plus caracterul NUL.

4) Funcţia ***strcat*** nu modifică şirul spre care pointează sursa (acesta fiind un pointer constant).

# Exemplu

```
…
char tab1[100] = ”limbajul C++”;
char tab2[ ] = ”este un superset al lui C”;
        …

strcat(tab1, ” ”);
strcat(tab1, tab2);
```

Pointerul *tab1* va face trimite la adresa de unde începe şirul *“Limbajul C++ este un superset al lui C”*.

# Variantă a funcției **strcat**

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

Utilizând această funcţie se concatenează la sfârşitul şirului spre care pointează ***dest*** cel mult ***n*** caractere ale şirului spre care pointează ***sursa***.

Dacă:

- *n ≥ lungimea şirului spre care pointează sursa* atunci se concatenează întregul şir.
- *n < lungimea şirului spre care pointează sursa* atunci se concatenează primele *n* caractere ale acesteia.

# Exemplu:

```
…
char tab[100] = ”Limbajul C este completat de ”;
char tab2[ ] = ”C++ este un superset al lui C”;
        …
strncat(tab, tab2, 3);
        …
```

Pointerul ***tab*** devine adresa către stringul: *“Limbajul C este completat de C++”.*

# 4. Compararea şirurilor de caractere

- Şirurile de caractere se pot compara folosind codurile ASCII ale caracterelor din compunerea lor.

- Pentru a înţelege mai bine acest mecanism considerăm $s_1$ şi $s_2$ două şiruri de caractere.

- Avem cazurile:

# două șiruri: **s**1 și **s**2

1. $s_1=s_2$ dacă au lungimi egale şi $s_1[i]=s_2[i]$ oricare ar fi $i$.

2. $s_1<s_2$ dacă există $i$ astfel încât $s_1[i]<s_2[i]$ şi $s_1[j]=s_2[j]$ oricare ar fi $j=0,\ldots,i-1$.

3. $s_1>s_2$ dacă există $i$ astfel încât $s_1[i]>s_2[i]$ şi $s_1[j]=s_2[j]$ oricare ar fi $j=0,\ldots,i-1$.

Ne putem gândi (spre comparație) la ordonarea cuvintelor în cadrul unui dicționar…

Există mai multe funcţii utilizate pentru compararea şirurilor de caractere. Una dintre cele mai utilizate este funcţia *strcmp*

# Funcția **strcmp**

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

Funcţia returnează:

- ***o valoare negativă*** dacă *şir1*< *şir2;*
- ***zero*** dacă şir1 = *şir2;*
- ***o valoare pozitivă*** dacă *şir1* > *şir2.*

- Funcţia nu modifică $s_1$ şi $s_2$.

Ca exemplu scriem un fragment de cod al unui program cu parolă:

```
...
char *p=”interzis”;
char nume[20];
…
printf(“\n introduceţi parola: “);
gets(nume);
if (strcmp(nume, p)) exit(1);
…
```

Există varianta de funcţie de comparare denumită ***strimcp*** care are formatul:

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

- Funcţia este asemănătoare cu *strcmp*, cu singura deosebirea ca *stricmp* nu face distincţie între literele mari şi mici.

*Observație: în unele compilatoare această funcție apare cu denumirea de* ***strcmpi***

# De exemplu pentru codul:

```
char *sir s1 = ”ABC”;
char *sir s2 = ”abc”;
int i;
...
```

Apelul:

```
i = strcmp(sir1, sir2);
```

produce o *valoare negativă* deoarece ”ABC”<”abc” (***A*** are codul 41h, iar ***a*** codul 61h).

În schimb:

```
i = stricmp(sir1, sir2);
```

produce *valoarea 0.*

Există o altă variantă de funcţie de comparare denumită ***strinmcp*** care are formatul:

***int strincmp( const char *s1, const char *s2, unsigned n);***

- Această funcţie limitează compararea primelor *cel mult n caractere*, ignorându-se diferenţa între literele mari şi mici. În rest ea acţionează la fel ca *stricmp*.

*Observație: putem avea variantele **strncmpi** sau **strncmp** în funcție de mediul de compilare utilizat.*

# Exemplu

- Ca exemplu, reluăm programul care citeşte o succesiune de cuvinte şi-l afişează de data asta pe cel mai mare (şi nu pe cel mai lung).

```
#include<stdio.h>
#include<string.h>
#define MAX 100

int main ()
{ char cuvant[MAX+1];
  char cuvant_max[MAX+1];
  cuvant_max [0]='\0';   /* cuvant_max se iniţializează
cu cuvântul nul */
  while (scanf("%100s", cuvant)!=EOF)
        { if (strcmp(cuvant,  cuvant_max) > 0)
                strcpy (cuvant_max, cuvant);
         }
printf("\n Cel mai mare cuvânt este %s", cuvant_max);
}
```

# 5. Identificarea şirurilor sau caracterelor dintr-un alt şir

- Pentru identificarea şirurilor sau caracterelor dintr-un alt şir avem funcţiile:

***strchr(s1, ch)*** → returnează un pointer la prima apariţie a caracterului ***ch*** în ***s1;***

***strstr(s1, s2)*** → returnează un pointer la prima apariţie a lui ***s2*** în ***s1.***

Dacă nu se regăsește caracterul sau subșirul căutat, aceste funcții întorc valoarea zero.

# Formatul funcţiilor este:

***char *strchr( const char *s1, const char ch);***

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

# Exemplu

```
#include<stdio.h>
#include<string.h>

int main ()
{
 char s[80];
 gets(s);        /* introducem de la tastatură ”Acesta este un test” */
...
 if (strchr(s, ’e’)) printf(”e este în şirul testat”);
 if (strstr(s, ”test”)) printf(”cuvântul test este în şirul testat”);
...
}
```

# Inițializarea vectorilor

- În C, un vector poate fi inițializat folosind o singură declarație după cum urmează :

```
double A[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

- Numărul de valori între acolade {} nu poate fi mai mare decât numărul de elemente pe care le declarăm pentru vector între parantezele pătrate [].

# Matrice bidimensionale

- Limbajul C admite utilizarea matricelor multidimensionale.
- Cea mai simplă formă de matrice multidimensională este cea bidimensională.
- O matrice bidimensională este un vector de vectori unidimensionali.