

## Tablouri. Adrese. Siruri de caractere

7 aprilie 2009

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri. Adrese. Siruri de caractere  
**Exemplu: Calculul primelor numere prime** 3

```
#include <stdio.h>
#define MAX 100 // preprocesorul inlocuieste MAX cu 100
int main(void) {
    unsigned p[MAX] = {2}; // primul element initializat cu 2
    unsigned cnt = 1, n = 3; // avem un prim, 3 e urmatorul candidat
    do {
        for (int j = 0; n % p[j]; ++j) // cat timp nu am gasit divizor
            if (p[j]*p[j] > n) { // daca nu mai sunt altii e prim
                p[cnt++] = n; break; // il inregistram si iesim din ciclu
            }
        n += 2; // trecem la numarul impar urmator
    } while (cnt < MAX); // pana nu e plin tabloul
    for (int j = 0; j < MAX; ++j)
        printf("%d\n", p[j]); // tiparim cate un element pe rand
    return 0;
}
```

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri. Adrese. Siruri de caractere  
**Variabile și adrese** 5

Orice variabilă  $x$  are o adresă, la care e memorată valoarea ei

**Operatorul prefix &** dă adresa operandului:  $\&x$  e adresa variabilei  $x$   
 Operandul lui  $\&$ : orice *valoare* (destinație validă de atribuire): variabile,  
 elemente de tablou. NU au adrese: alte expresii, constantele

**Numele** unui tablou e chiar *adresa* tabloului. Ex: `int a[6];`

Numele a reprezentă *adresa* tabloului, NU toate elementele împreună  
 O adresă poate fi tipărită (în hexazecimal) cu formatul `%p` în `printf`  
`#include <stdio.h>`

```
int main(void) {
    double d; int a[6];
    printf("Adresa lui d: %p\n", &d); // folosim operatorul &
    printf("Adresa lui a: %p\n", a); // a e adresa, nu e nevoie de &
    return 0;
}
```

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri. Adrese. Siruri de caractere  
**Tablouri** 2

Tablou (vector) = o secvență de elemente de *aceiași tip* de date asociată o *valoare* ( $m$ ) cu un anumit *indice* ( $n$ ) (ca un șir matematic)

Declarație: *tip nume-tablou[nr-elem]*: `double x[20]; int mat[10][20];`

Inițializarea: între acolade, cu virgule: `int a[4] = { 0, 1, 4, 9 };`

**Numele** tabloului e *adresa* la care începe memorarea elementelor

**Dimensiunea** tabloului (nr. de elemente) = o *constantă* pozitivă

C99: și dimensiuni variabile, cu valoare cunoscută în momentul declarații

`int f(int n) { int tab[1]; /* la apelarea funcției știm n */ }`

**Element**: *nume-tab[indice]* indice: orice expresie cu valoare întreagă

Un element de tablou `x[3]` `a[1]` `t[2*1+1]` e folosit ca orice variabilă (are o valoare, poate fi folosit în expresii, poate fi atribuit)

**ATENȚIE!** În C, indicii de tablou încep de la *zero!*

`int a[4];` are elemente `a[0]`, `a[1]`, `a[2]`, `a[3]`, **NU există** `a[4]`

Sintaxa declarației: *tip a[dim]*; sugerează că *a[indice]* are tipul *tip*

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri. Adrese. Siruri de caractere  
**Tablouri multidimensionale (matrice)** 4

Sunt de fapt tablouri cu elemente care sunt la randul lor tablouri.

Decl.: *tip nume[dim1][dim2]...[dimN]*: `double m[6][8]; int a[2][4][3];`

`m`: tablou de 6 elemente, fiecare un tablou de 8 reali. Element: `m[4][3]`

Aceleași reguli: dimensiuni *constante* (C99: cunoscute la declarație)

`#define LIN 2 // definiții de constante pentru preprocesor`

`#define COL 5 // putem modifica ușor valorile într-un singur loc`

`int main(void) {`

`double a[LIN][COL] = { 10, 1, 2, 3, 4}, 5, 6, 7, 8, 9 };`

`// inițializare: cu acolade la fiecare linie, sau un singur șir`

`for (int i = 0; i < LIN; ++i) { // pe linii`

`for (int j = 0; j < COL; ++j) // pe coloane`

`printf("%f ", a[i][j]);`

`putchar('\n'); // gata o linie`

`return 0;`

`}`

Elementele: dispuse succesiv în memorie: `m[i][j]` e pe poziția `i*COL+j`

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri. Adrese. Siruri de caractere  
**Tablouri ca parametri la funcții** 6

Declarația unui tablou alocă și memorie pentru elementele sale dar *numele* reprezentă *adresa* sa și nu tabloul ca tot unta

⇒ numele tabloului NU poartă informații despre dimensiunea lui

excepție: *sizeof(numeTab)* este *nr-elem \* sizeof(tip-elem)*

La funcții trebuie transmis *numele* tabloului (*adresa*) **SI lungimea** sa

**NU** scriem lungimea între `[]` la parametru, nu e luată în considerare

```
#include <stdio.h>
void printTab(int t[], unsigned len) {
    for (int i = 0; i < len; ++i) printf("%d ", t[i]);
    putchar('\n');
}
int main(void) {
    int prim[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };
    printTab(prim, 10); // ATENȚIE: NU prim[10], NU prim[]
    return 0;
}
```

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri, Adrese, Șiruri de caractere  
**Tablouri ca parametri la funcții**

7

Transmiterea parametrilor în C se face *prin valoare*

⇒ un parametru tablou e transmis prin *valoarea adresei sale*

Având adresa, funcția poate accesa (*citi și scrie*) elementele tabloului

```
void sumvec(double a[], double b[], double r[], unsigned len) {
    for (unsigned i = 0; i < len; ++i) r[i] = a[i] + b[i];
}

#define LEN 3 // macro pt. constanta utilizata de mai multe ori
int main(void) {
    double a[LEN] = {0, 1.41, 1}, b[LEN] = {1, 1.73, 1}, c[LEN];
    sumvec(a, b, c, LEN);
    return 0;
}
```

### Inițializare

Tablourile neinițializate au elemente de valoare necunoscută.

Tablourile inițializate parțial au restul elementelor nule.

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri, Adrese, Șiruri de caractere  
**Tablouri multidimensionale ca parametri la funcții**

9

```
m[i][j] e pe poziția i*COL+j ⇒ trebuie cunoscut COL ⇒ la parametri
trebuie toate dims. în afară de prima. Ex:  $A_{lin \times 10} \times B_{10 \times 6} = C_{lin \times 6}$ 
void matmul(double a[][10], double b[][6], double c[][6], int lin) {
    for (int i = 0; i < lin; ++i) // functia e buna doar pentru
        for (int j = 0; j < 6; ++j) { // matrici cu dim. 10 si 6
            c[i][j] = 0;
            for (int k = 0; k < 10; ++k) c[i][j] += a[i][k]*b[k][j];
        }
}
```

} // pentru folosire vom scrie (de exemplu în main):

```
double m1[8][10], m2[10][6], m3[8][6]; // 1e dam apoi valori
matmul(m1, m2, m3, 8); // NU: m1[0][0], NU: m2[0][6], NU: m3[8][6]
În C99: parametri la funcții pot fi tablouri de dimensiuni variabile
(dar cunoscute în momentul apelului – dimensiunile sunt tot parametri)
void matmul(int lin, int n, int p, double a[][ln], double b[ln][p],
            double c[][p]); // n, p declarati inainte de folosire
```

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri, Adrese, Șiruri de caractere  
**Tipul pointer**

11

Rezultatul unei operații *adresă* are un tip, ca și orice expresie

Pentru o variabilă declarată *tip x*: *tipul adresei sale &x e tip \**

(citi: *pointer la tip*, adică: adresă unde se află un obiect de acel *tip*)

În particular, *numele* unui tablou are tipul pointer la tipul elementului  
int a[4]; a are tipul int \* char s[8]; s are tipul char \*

La declararea parametrilor funcției, void f(*tip a*[1]) înseamnă de fapt  
void f(*tip \*a*) (de aceea dimensiunea: void f(*tip a*[6]) *nu contează*)

Tipul unei constante șir de caractere "sir" este char \*:

adresa unde se găsește șirul în memorie

Valoarea specială NULL (0 de tip void \* ≡ adresă de tip neprecizat)

e folosit pentru a indica o adresă *invalidă*

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri, Adrese, Șiruri de caractere  
**Tablouri de dimensiune variabilă (C99)**

8

cu dimensiune cunoscută la declarare (ex. parametru la funcție)

```
#include <stdio.h>
void fractie(unsigned m, unsigned n) {
    int apare[n]; // dimensiune data de parametrul n
    for (int i = 0; i < n; ++i) apare[i] = 0; // init
    printf("%u. ", m/n); // calcul
    while (m % n) { // rest nenul
        if (apare[m]) { printf("%u...", 10*m/n); break; } // periodic
        m = 10; putchar(m/n + '0'); // umatoarea cifra
    }
    putchar('\n');
}

int main(void) {
    fractie(5, 28); // 5/28 = 0.178571428...
    return 0;
}
```

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri, Adrese, Șiruri de caractere  
**Tablouri și șiruri de caractere**

10

```
char cuvart[20]; // tablou de caractere neinițializat
char msg[] = "test"; // 5 octeti, terminat cu '\0'
char msg[] = {'t','e','s','t','\0'}; // același, scris altfel
char nume[3] = {'E', 'T', 'O'}; // nu are '\0' la sfârșit !
char sir[20] = "test"; // restul pana la 20 sunt '\0'
```

În C, termenii *șir de caractere* înseamnă un tablou de caractere încheiat  
în memorie cu caracterul/octetul '\0' (iafel *constanțele șir*: "salut\n")

(la memorare, nu în reprezentarea la intrare: nu cîm/țipăim '\0')

**ATENȚIE**: toate funcțiile standard pentru șiruri depind de acestai

nu au nevoie de parametru lungime, dar șirul trebuie terminat cu '\0'

La șiruri inițializate, dar fără dimensiune specificată (ex. msg mai sus)  
se alocă dimensiunea inițializatorului + 1 caracter '\0'

Programarea calculatoarelor. Curs 7

Marius Minea

Tablouri, Adrese, Șiruri de caractere  
**Funcții cu șiruri de caractere (string.h)**

12

```
size_t strlen(const char *s); // returneaza lungimea sirului s
char *strchr(const char *s, int c); // cauta caract. c in sirul s
// returneaza adresa unde l-a gasit sau NULL (0) daca nu-l gaseste
char *strcpy(char *dest, const char *src); // copiază src în dest
char *strcat(char *dest, const char *src); // concat. src la dest
// pentru ambele e necesar ca la dest sa fie loc suficient
int strcmp(const char *s1, const char *s2); // compara 2 siruri
// returneaza întreg < 0 sau = 0 sau > 0 dupa cum e s1 fata de s2
char *strncpy(char *dest, const char *src, size_t n);
// copiază cel mult n caractere din src în dest
char *strncat(char *dest, const char *src, size_t n);
// concatenează cel mult n caractere din src la dest
int strncmp(const char *s1, const char *s2, size_t n);
// compara sirurile pe lungime cel mult n caractere
size_t: tip întreg fără semn pentru dimensiuni
const: specificator de tip, indică că obiectul respectiv nu e modificat
```

Programarea calculatoarelor. Curs 7

Marius Minea