

Adrese. Tablouri. Şiruri de caractere

Marius Minea

11 aprilie 2006

Variabile și adrese

În limbajul C, orice variabilă are o *adresă*: o valoare numerică; indică locul din memorie unde e memorată valoarea variabilei

Operatorul prefix & dă adresa operandului: `&x` e adresa variabilei `x`
Operandul: orice *lvalue* (destinație validă în stânga unei atribuiri); nu se poate lua adresa unei expresii arbitrare

O adresă poate fi tipărită (în hexazecimal) cu formatul `%p` în `printf`

```
#include <stdio.h>
int main(void)
{
    double d; int n;
    printf("Adresa lui d: %p\n", &d); // de ex. 0xbfbb8840
    printf("Adresa lui n: %p\n", &n); // de ex. 0xbfbb883c
    return 0;
}
```

Tablouri

Tablou (vector) = concept corespunzător *șirului* din matematică
= o secvență de elemente de *același tip* de date
(șir x_n = funcție de la *indice* natural n la *valoare* de anumit tip)

Sintaxa declarației: *tip nume-tablou[nr-elem];* (eventual inițializat)

Exemple: `int a[4];` `double x[20];`

- *numele* tabloului e *adresa* la care începe memorarea elementelor
- *dimensiunea* tabloului (nr. de elemente) = o *constantă pozitivă*
- un element: dat de *numele* tabloului și un indice întreg: `x[3]`
(ca indice se poate folosi orice *expresie* de valoare întreagă)
- un element de tablou poate fi folosit ca orice variabilă individuală
(are o valoare, și poate primi una nouă, în stânga unei atribuirii)
- **ATENȚIE!** în C, numerotarea elementelor începe de la zero!!
tabloul `a` are elemente `a[0]`, `a[1]`, `a[2]`, `a[3]`, **NU există** `a[4]` !!

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

Exemplu: șirul lui Fibonacci

```
#include <stdio.h>
int main(void)
{
    unsigned f[20]; // tablou pentru elementele șirului;
    unsigned i;
    f[0] = f[1] = 1; // cazul de bază
    for (i = 2; i < 20; ++i)
        f[i] = f[i-1] + f[i-2]; // completăm tabloul
    for (i = 0; i < 20; ++i)
        printf("%u ", f[i]); // tipărim tabloul
    putchar('\n');
    return 0;
}
```

Tablouri multidimensionale

Declararea tablourilor de mai multe dimensiuni: `int m[10][7];`

Interpretare: tablou de 10 elemente, fiecare un tablou de 7 întregi

Elementele: dispuse succesiv în memorie, în ordinea liniilor din tablou

Un element de tablou e accesat cu sintaxa `m[indice1][indice2]`

```
#define LIN 10 // definitii de simboluri pentru preprocesor
#define COL 7 // se substituie textual in prima faza de compilare
int i, j, m[LIN][COL];
```

Elementul `m[i][j]`, este pe poziția $i*COL+j$ din tablou.

⇒ ca și un tablou liniar, `int m[LIN*COL];`, accesat cu `m[i*COL+j]`

Inițializarea tablourilor

Pentru variabilele de tip tablou, inițializatorii se scriu între acolade

– nivelele de acolade indică sub-obiectele inițializate

```
int m[2][3] = { { 1, 0, 0 }, { 0, 1, 0 } };
```

– pt. inițializator mai mic ca dimensiunea, restul e inițializat pe zero

– dacă dimensiunea nu e dată explicit, se deduce din inițializator

```
int prime[] = { 2, 3, 5, 7, 11 }; (5 elemente)
```

– se poate specifica elementul de inițializat, se continuă apoi în ordine:

```
int t[10] = { 1, 2, 3, [8] = 2, 1 }; /* t[3]-t[7] nespecificate */
```

Tablouri ca parametri la funcții

Declarația unui tablou are două efecte *distincte*

- declară un nume pentru tablou (adresa acestuia)
- alocă memorie pentru elemente

Numele (adresa) NU poartă informații despre dimensiunea tabloului !

⇒ La o funcție, trebuie transmisă adresa tabloului, ȘI dimensiunea sa!

```
#include <stdio.h>
void printtab(int t[], int len) {
    int i;
    for (i = 0; i < len; ++i) printf("%d ", t[i]);
    putchar('\n');
}
int main(void) {
    int a[5] = { 1, 3, 5, 7, 9 };
    printtab(a, 3); // tiparim doar primele 3
    return 0;
}
```

Tablouri ca parametri la funcții

Transmiterea parametrilor în C se face *prin valoare*

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

Având adresa, funcția poate accesa fiecare element, fie la citire, fie la scriere!

```
void sqrtab(int t[], int len) {
    int i;
    for (i = 0; i < len; ++i) tab[i] = i*i;
}

int main(void) {
    int a[5];
    sqrtab(a, 5); // va completa cu 0, 1, 4, 9, 16
    return 0;
}
```


Tablouri multidimensionale parametri la funcții

Fie declarația `tip a[DIM1][DIM2];` (cu DIM1, DIM2 *constante*)

Atunci `a[i][j]` e elementul `j` din tabloul de DIM2 elemente `a[i]`

⇒ are $i * DIM2 + j$ elemente înainte sa

⇒ pentru compilarea expresiei `a[i][j]` e necesară cunoașterea lui DIM2

⇒ în declarația unei funcții cu parametri tablou trebuie precizate toate dimensiunile în afară de prima (irelevantă): `void f(int m[][5]);`

În C99, se pot specifica parametri tablou de dimensiuni variabile:

```
void f(int m, int n, int a[m][n]);      sau
```

```
void f(int m, int n, int a[][n]);
```

Şiruri de caractere

= un caz particular de tablouri, cu tip caracter

```
char cifre[20]; // tablou de caractere neinițializat
char msg[] = "test"; // constantă şir, 5 octeti, terminat cu '\0'
char msg[] = {'t','e','s','t','\0'}; // acelaşi lucru scris altfel
char nume[3] = { 'U', 'P', 'C' }; // tablou inițializat, fara '\0'
char msg[20] = "test"; // constantă şir, 20 octeti, restul '\0'
```

– în memorie, sfârşitul unui şir e indicat cu caracterul special '\0' (nul)

Atenție: toate funcțiile care lucrează cu şiruri depind de acest lucru !

(dar convenția nu are legătură cu aspectul în text, de ex. la citire)

– dacă şirul e declarat fără a explicita dimensiunea (vezi `msg`), se alocă dimensiunea inițializatorului (şirului dat) + 1 (pt. '\0')

Tipul pointer

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

Dacă o variabilă e declarată *tip* *x*;, tipul adresei sale *&x* e *tip **
(citit: pointer la tip)

În particular, tipul numelui unui tablou e pointer la tipul elementului:
pentru `int a[4]`; *a* e o adresa de tipul `int *`

La declararea de parametri la funcție, e echivalent dacă scriem
`void f(tip a[])` sau `void f(tip *a)`

Tipul unei constante șir de caracter (între " ") este `char *`
(adresa unde compilatorul a depus constanta șir)

Funcții cu șiruri de caractere (string.h)

```
size_t strlen(const char *s); // lungimea șirului s
char *strcpy(char *dest, const char *src); // copiaza src in 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 intreg < 0 sau == 0 sau > 0 dupa cum e s1 fata de s2
char *strncpy(char *dest, const char *src, size_t n);
// copiaza cel mult n caractere din src in dest
char *strncat(char *dest, const char *src, size_t n);
// concateneaza 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
```