Limbaje de programare

Tablouri. Șiruri de caractere. Matrice

6 noiembrie 2012

Tablouri: recapitulare

În C, numele unui tablou reprezintă *adresa* tabloului.
NU conține informație și despre dimensiunea alocată.

$\Rightarrow$ o funcție cu parametru tablou are nevoie și de lungimea sa
$\Rightarrow$ *programatorul e responsabil* pentru a evita depășirea

O funcție cu parametru tablou (= adresa lui) poate *folosi* (citi) sau *modifica* (scrie) elemente.

Calculul unei valori dintr-un tablou

```
double sum(double a[], unsigned len)
{
  double s = 0.;          // trebuie initializata!
  for (unsigned i = len; i--;) // de la sfarsit
    s += a[i];
  return s;
}

#define LEN 4

int main(void)
{
  double a[LEN] = { 1.0, 2.3, -5.6, 7 };
  printf("%f\n", sumtab(a, LEN));
  return 0;
}
```

Rezultatul acumulat (s) *trebuie inițializat!*

Sensul parcurgerii poate conta (sau nu), în funcție de problemă.

Prelucrarea selectivă a unor elemente

```
// media notelor studentilor promovati
double med_prom(double a[], unsigned len)
{
  double s = 0.;          // suma elementelor
  unsigned num = 0;       // numarul elementelor selectate
  for (unsigned i = len; i--;)
    if (a[i] >= 5) {      // doar pentru anumite elemente
      s += a[i];
      ++num;
    }
  return num ? s / num : 0;
}
```

Împărțirea la 0 ar genera valoarea `NAN` (not a number, `math.h`)
$\Rightarrow$ returnăm o valoare (0) care nu poate fi rezultat normal (≥ 5)

Căutarea unui element într-un tablou

Care este cel mai mic factor prim al lui n ≤ 1000 ?
(avem toate numerele prime p cu $p^2 \leq 1000$)

```
#define NP 11
unsigned prim[NP] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31};

unsigned factor(unsigned n)
{
  for (unsigned i = 0; i < NP; ++i)
    if (n % prim[i] == 0) return prim[i];   // factor prim
  return n;     // nu s-a gasit factor prim <= 31
}
```

Se iese din ciclu (cu `break`, sau cu `return` din funcție) când s-a găsit un element care satisface condiția (aici: divide numărul dat)
La ieșirea normală din ciclu (nu s-a găsit niciun element bun) trebuie returnată altă valoare care să indice acest lucru (aici: n)

Căutarea unui element într-un tablou

Când vrem să ieșim doar din ciclu, nu din funcție, folosim `break`;
Inițializăm rezultatul cu valoarea care indică eșecul căutării.

```
#include <stdio.h>

#define NP 11
unsigned prim[11] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31};

int main(void)
{
  unsigned n = 751, p = n;       // p = n inseamna n prim
  for (unsigned i = 0; i < NP; ++i)
    if (n % prim[i] == 0) { p = prim[i]; break; }
  if (p < n) printf("%u e cel mai mic factor prim al lui %u\n",
  else printf("%u e prim\n", n);
}
```

Memorarea frecvențelor de apariție

Câte caractere de fiecare fel apar într-un text ?

```
#include <stdio.h>

int main(void)
{
  int c;
  unsigned frq[256] = {0};      // indexat dupa caractere
  while ((c = getchar()) != EOF)
    ++frq[c];   // incr. nr. de la pozitia c (cod ASCII)
  for (unsigned car = 0; car < 256; ++car)
    if (frq[car] != 0)
      printf("%c a aparut de %u ori\n", car, frq[car]);
  return 0;
}
```

Tablouri și șiruri de caractere

```
char cuvant[20];    // tablou de caractere neinitializat
char nume[3] = { 'E', 'T', 'C' };   // exact 3 caractere
```

În C, termenul *șir de caractere* înseamnă un tablou de caractere încheiat în memorie cu caracterul `'\0'` (cod ASCII 0).

Constantele șir `"salut\n"` se termină tot cu `'\0'`

terminatorul `'\0'` (caraterul nul) nu se citește / tipărește

```
char msg[] = "test";    // 5 octeti, terminat cu '\0'
char msg[] = {'t','e','s','t','\0'};    // acelasi, scris altfel
char sir[20] = "test";  // restul pana la 20 sunt '\0'
```

La șiruri inițializate, dar fără dimensiune dată (ex. `msg` mai sus) se alocă dimensiunea inițializatorului + 1 pt. caracterul `'\0'`

Toate *funcțiile standard* au nevoie de *șiruri terminate cu* `'\0'`

Tipul pointer

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

Pentru o variabilă declarată *tip* `x;` *tipul adresei* sale `&x` e *tip* `*`
(*pointer la tip*, adică: adresă unde se află un obiect de acel *tip*)

Numele unui tablou are tipul pointer la tipul elementului

`int a[4];` `a` are tipul `int *`

`char s[8];` `s` are tipul `char *`

În antetul funcției, `void f(`*tip* `a[])` înseamnă `void f(`*tip* `*a)`
(de aceea dimensiunea: `void f(`*tip* `a[6])` *nu contează*)

Valoarea `NULL` (0 de tip `void *`, adică adresă de tip neprecizat) indică o *adresă invalidă* (folosită când trebuie returnată o valoare de tip adresă, dar funcția nu s-a încheiat cu succes)

Un șir de caractere este (are tipul) `char *`

În C, un șir de caractere e reprezentat prin *adresa* lui.
inclusiv *constantele* șir: `"sirul"`

Această adresă (și deci șirul) are tipul `char *`

ATENȚIE! `'a'` e un `char`, dar `"a"` e un șir (adresă, `char *`)

Un șir (constant sau nu) se termină cu caracterul nul `'\0'`

Funcțiile care lucrează cu șirul pot ști astfel unde se termină
(NU mai au nevoie de parametru lungime).

ATENȚIE! Comparăm șiruri cu `strcmp`, `strncmp`, NU cu ==
== compară *adrese* (UNDE se află șirurile), NU conținutul lor!

ATENȚIE! o *constantă* șir `"test"` NU poate fi modificată
(nu dați ca parametru la o funcție care modifică șirul primit)

Funcții cu șiruri de caractere (string.h)

```
size_t strlen(const char *s); // returneaza lungimea sirului s
char *strchr(const char *s, int c); // cauta caract. c in s
char *strstr(const char *big, const char *small); // cauta sir
// ambele returneaza adresa unde e gasit sau NULL daca nu exista

char *strcpy(char *dest, const char *src); // copie src in dest
char *strcat(char *dest, const char *src); // concat src la dest
// ambele necesită destul loc la dest, ATENŢIE LA DEPĂŞIRE!
int strcmp (const char *s1, const char *s2); // compara
// returneaza intreg < 0 sau 0 sau > 0 dupa cum e s1 fata de s2

char *strncpy(char *dest, const char *src, size_t n);
char *strncat(char *dest, const char *src, size_t n);
// copiază/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
```

`size_t`: tip întreg fără semn pentru dimensiuni

`const`: specificator de tip: obiectul respectiv nu e modificat

Tablouri multidimensionale (matrice)

Sunt de fapt tablouri cu elemente care sunt la randul lor tablouri.
Declarație: *tip nume*[*dim1*][*dim2*]...[*dimN*];

Exemple: `double m[6][8]; int a[2][4][3];`
`m`:tablou de 6 elemente, fiecare un tablou de 8 reali.
Element: `m[4][3]`

Și aici: dimensiuni *constante* (în C99: cunoscute la declarare)
Elementele tabloului sunt dispuse succesiv în memorie:
`m[i][j]` e pe poziția `i*COL+j`

Un exemplu cu matrice

```
#define LIN 2   // numarul de linii
#define COL 5   // numarul de coloane
int main(void) {
  double a[LIN][COL] = { {0, 1, 2, 3, 4}, 5, 6, 7, 8, 9 };
  // initializat cu acolade la fiecare linie, sau un singur şir

  for (int i = 0; i < LIN; ++i) {  // parcurge linii
    for (int j = 0; j < COL; ++j)  // parcurge coloane
      printf("%f ", a[i][j]);
    putchar('\n');                 // sfarsit de linie
  }
  return 0;
}
```

Tablouri de dimensiune variabilă (C99)

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;     // initializare
  printf("%u.", m/n);    // catul = partea intreaga
  while (m %= n) {       // rest nenul
    if (apare[m]) { putchar(10*m/n+'0'); break; }   // periodic
    apare[m] = 1;        // marcam ca apare
    m *= 10;             // deplaseaza cu o pozitie
    putchar(m/n + '0'); // catul = urmatoarea cifra
  }
  putchar('\n');
}
int main(void) {
  fractie(5, 28);        // 5/28 = 0.178571428...
  return 0;
}
```

Tablouri multidimensionale ca parametri la funcții

`m[i][j]` e pe poziția `i*COL+j` $\Rightarrow$ trebuie cunoscut `COL` $\Rightarrow$ trebuie *toate* dimens. 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; i < 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 in main):
double m1[8][10], m2[10][6], m3[8][6]; // le dam apoi valori
matmul(m1, m2, m3, 8); // NU: m1[][], NU: m2[][6], NU: m3[8][6]
```

C99 permite ca parametri tablouri de dimensiuni variabile, dar cunoscute în momentul apelului: și dimensiunile sunt parametri

```
void matmul(int l, int n, int p, double a[][n], double b[n][p],
       double c[][p]); // n, p declarati inainte de folosire
```