Limbaje de programare

Tablouri. Funcții de intrare/ieșire

7 noiembrie 2011

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; // init
  printf("%u.", m/n); // catul
  while (m %= n) {    // rest nenul
    if (apare[m]) { printf("%u...", 10*m/n); break; } //periodic
    apare[m] = 1; // marcam ca apare
    m *= 10; putchar(m/n + '0'); // 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]
```

În C99: parametri la funcții pot fi tablouri de dimensiuni variabile dar cunoscute în momentul apelului – ex. daca 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
```

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*)

Tipul unei constante șir de caractere `"text"` este `char *`
(adresa unde se găsește șirul în memorie)

ATENȚIE: o *constantă* șir NU poate fi modificată
(nu dați ca parametru la o funcție care modifică)
Valoarea `NULL` (0 de tip `void *`, adică adresă de tip neprecizat)
indică o adresă *invalidă*.

Citirea unei linii de text: `fgets`

Declarație:

```
char *fgets(char *s, int size, FILE *stream);
```

(toate funcțiile de intrare/ieșire sunt *declarate* în `stdio.h`)

Citește până la (inclusiv) linie nouă `\n`, max. `size-1` caractere, pune linia în tabloul `s`, adaugă `'\0'` la sfârșit.

Exemplu: `char tab[80]; fgets(tab, 80, stdin);`

Parametrul al treilea pentru `fgets` indică *fișierul* din care se citește: `stdin` (din `stdio.h`) indică *intrarea standard* (normal: tastatura)

ATENȚIE! La *orice citire* trebuie verificat dacă s-a efectuat corect:
`fgets` returnează `NULL` dacă n-a citit nimic (sfârșit de fișier),
la succes returnează chiar adresa primită parametru (deci nenulă)
⇒ Testăm că rezultatul e *nenul* pentru a ști dacă s-a citit cu succes

Citirea unei linii de text (cont.)

Exemplu: *citire și afișare linie cu linie până la sfârșitul intrării*

```
char s[81];
while (fgets(s, 81, stdin)) printf("%s", s);
```

O linie cu > 80 de caractere va fi citită (și afișată) pe bucăți

Putem testa dacă linia citită e incompletă (a fost trunchiată)

```
int c; char s[81];
if (fgets(s, 81, stdin))          // s-a citit linia
  if (strlen(s) == 80 && s[79] != '\n' // neterminata
    && ((c = getchar()) != EOF)    // n-am atins EOF
      printf("linie incompleta: %s\n", s);
      ungetc(c, stdin);
  } else printf("linie completa: %s\n", s);
```

ATENȚIE! NU folosiți funcția `gets` ! Nu limitează șirul citit

$\Rightarrow$ vulnerabilități grave de securitate prin depășiri de memorie

Scrierea formatată: `printf`

```
int printf(const char* format, ...);
```

Primul parametru (format): un *șir de caractere*; poate conține:
- caractere obișnuite (se tipăresc)
- *specificatori de format*: `%` și o literă:

`%c` char, `%d` decim., `%f` real, `%p` pointer, `%s` sir, `%u` unsigned, `%x` hex

Restul parametrilor: *expresii*, ale căror *valori* se tipăresc
- numărul și tipul trebuie să corespundă cu specificatorii de format

Rezultatul: numărul de caractere tipărite (de obicei ignorat)

Exemplu:

```
printf("radical din %d este %f\n", 3, sqrt(3));
```

Citirea formatată: `scanf`

```
int scanf(const char* format, ...);
```

Parametrul 1: un *șir de caractere*, cu specificatori de format
ca la `printf`, dar: `%f` e `float`, `%lf` e `double`

Restul parametrilor: *adresele* variabilelor de citit: `&`
La șiruri NU se pune `&`, numele șirului e chiar adresa lui.

Exemple:
```
int n; scanf("%d", &n);
float a[4]; scanf("%f", &a[2]);
char cuv[20]; scanf("%19s", cuv); //fara &, cuv e adresa
```

Returnează: *numărul* variabilelor citite (atribuite) (NU valoarea!)
sau `EOF` la eroare sau sfârșitul intrării *înainte* de prima citire

ATENȚIE! Verificați datele de intrare!

Utilizatorul poate introduce (sau nu) *orice* și *oricât*:
din greșeală, din neînțelegere, din rea intenție...

⇒ Verificați *obligatoriu* citirea corectă *testând rezultatul returnat*!

NU scriem niciodată doar: `int n; scanf("%d", &n);`

Nu putem ști dacă s-a introdus (și deci s-a citit) un număr!
Testăm valoarea returnată de funcția de citire (scanf, gets, etc.)

```
#include <stdio.h>
int main(void) {
  int m, n;
  if (scanf("%d%d", &m, &n) != 2) return -1; // evtl. mesaj
  printf("suma: %d\n, m + n);  // executie normala
  return 0;
}
```

Citirea cu `scanf`: potrivirea cu formatul

Șirul de format poate avea specificatori, sau *caractere obișnuite*
- la `printf`: se tipăresc;
- la `scanf`: *trebuie să apară în intrare*

```
unsigned z, l, a;
if (scanf("%u.%u.%u", &z, &l, &a) == 3)
  printf("s-a citit corect: z=%u, l=%u, a=%u\n", z, l, a);
else printf("eroare la introducerea datei\n");
```

introducem `15.4.2008` (cu puncte!) $\Rightarrow$ `z=15, l=4, a=2008`

`scanf` citește până când intrarea *nu corespunde* formatului
Caracterele care nu se potrivesc rămân necitite în intrare;
restul variabilelor nu se atribuie

`scanf("%d%d", &x, &y);` in: `12A` ret. 1; `x = 12`, `y`: necitit; rămas: `A`
`scanf("%d%x", &x, &y);` in: `12A` ret. 2; `x = 12`, `y = 0xA` (10)

Repetarea citirii la eroare

La eroare, *consumaţi intrarea* înainte de a cere din nou date:

```
int m, n;
printf("Introduceţi două numere: ");
while (scanf("%d%d", &m, &n) != 2) {  // cat timp nu e corect
  scanf("%*[^\n]"); getchar();  // consumă restul liniei
  // sau: while ((c = getchar()) != EOF && c != '\n');
  printf("mai încercaţi o dată: ");
}                                     // acum putem folosi m şi n
```

Tipar uzual de citire repetată: `while` (*citit bine*) *prelucrează*

```
while (fgets(...)) { /* prelucreaza */ }
while ((c = getchar()) != EOF)  { /* prelucreaza */ }
while (scanf(...) == nr_variabile) { /* prelucreaza */ }
```

Citirea unui cuvânt

cuvânt = șir de caractere *fără spații albe*. Citim cu formatul `s`
Tabloul în care citim cuvântul are o dimensiune limitată
⇒ trebuie specificată *lungimea maximă* (un număr) între % și `s`
(cu 1 mai puțin decât lungimea tabloului, ca să fie loc pentru `\0`)

```
char t[33]; if (scanf("%32s", t) == 1) ...
```

Cu formatul `s`, `scanf` consumă și ignoră spațiile albe inițiale
(`\t \n \v \f \r` și spațiu); adaugă `'\0'` la sfârșit

ATENȚIE! Numele de tablou *e o adresă*, nu mai trebuie pus `&`

E *obligatorie lungimea* între % și `s`. Lipsa e o *eroare gravă*!
⇒ corupere de memorie, atacuri de securitate

Formatul `s` citește un *cuvânt* (până spații), *nu o linie!*

Citirea de (șiruri de) caractere

Un caracter: `int c = getchar(); if (c != EOF) ...`
Sau: `char c; if (scanf("%c", &c) == 1) {/* citit bine */}`

Citirea unui *număr fix de caractere*:
`char tab[80]; if (scanf("%80c", tab) == 1) ...`
citește EXACT 80 de caractere, *orice* (inclusiv spații albe)
NU adaugă `'\0'` la sfârșit

Citirea unui șir din *caractere permise*: se trec între `[ ]`
(intervale: cu -); citirea se oprește la primul caracter nepermis

`char a[33]; if (scanf("%32[A-Za-z_]", a) == 1) ...`
max. 32 litere sau _
ATENȚIE! E *obligatorie* lungimea limită între % și `[ ]`

Citirea unui șir *cu excepția unor caractere*: la fel ca mai sus,
dar ^ după `[` specifică caracterele *nepermise*. (ex. cifră , sau .):
`char t[81]; if (scanf("%80[^,.0-9]", t) == 1) ...`

ATENȚIE! Formatul `[ ]` NU se scrie urmat de s: `%20[A-Z]s`

scanf: separatori, limitare

Formatele *numerice* și `s` consumă (sar peste) spații albe inițiale
`"%d%d"` doi întregi separați și eventual precedați de spații albe

Formatele `c [ ] [^ ]` nu sar peste spații albe

Un *spațiu alb* în șirul de format consumă *oricâte* spații albe
`scanf(" ");` consumă toate spațiile albe până la altceva
`"%c %c"` citește char oarecare, consumă spații, citește alt char
`"%d %f"` același efect ca `"%d%f"` (spațiile sunt permise oricum)
!!! `"%d "` : spațiu după număr consumă spații până la altceva!

Un număr între % și caracterul de format limitează caracterele citite
`%4d` întreg din cel mult 4 caractere (spațiile inițiale nu contează)

`scanf("%d%d", &m, &n);`	`12 34`	`m=12 n=34`
`scanf("%2d%2d", &m, &n);`	`12345`	`m=12 n=34 rest: 5`
`scanf("%d.%d", &m, &n);`	`12.34`	`m=12 n=34`
`scanf("%f", &x);`	`12.34`	`x=12.34`
`scanf("%d%x", &m, &n);`	`123a`	`m=123 n=0xA`

Specificatori de format în `scanf`

`%d`: întreg zecimal cu semn
`%i`: întreg zecimal, octal (`0`) sau hexazecimal (`0x`, `0X`)
`%o`: întreg în octal, precedat sau nu de 0
`%u`: întreg zecimal fără semn
`%x`, `%X`: întreg hexazecimal, precedat sau nu de `0x`, `0X`
`%c`: *orice* caracter; nu sare peste spații (doar `" %c"`)
`%s`: șir de caractere, până la primul spațiu alb. Se adaugă '\0'.
`%a`, `%A`, `%e`, `%E`, `%f`, `%F`, `%g`, `%G`: real (posibil cu exponent)
`%p`: pointer, în formatul tipărit de `printf`
`%n`: scrie în argument (`int *`) nr. de caractere citite până atunci
nu citește nimic; nu incrementează nr. de câmpuri citite
`%[···]`: șir de caractere din mulțimea indicată între paranteze
`%[^···]`: șir de caractere fără mulțimea indicată între paranteze
`%%`: caracterul procent

Specificatori de format în `printf`

`%d`, `%i`: întreg zecimal cu semn
`%o`: întreg în octal, fără 0 la început
`%u`: întreg zecimal fără semn
`%x`, `%X`: întreg hexazecimal, fără `0x`/`0X`; cu `a-f` pt. `%x`, `A-F` pt. `%X`
`%c`: caracter
`%s`: șir de caractere, până la '\0' sau nr. de caract. dat ca precizie
`%f`, `%F`: real fără exp.; implicit: 6 poz.; la precizie 0: fără punct
`%e`, `%E`: real, cu exp.; implicit 6 poz.; la precizie 0: fără punct
`%g`, `%G`: real, ca `%e`, `%E` dacă exp. $<$ -4 sau $\geq$ precizia; altfel ca `%f`.
Nu tipărește zerouri sau punct zecimal în mod inutil
`%a`, `%A`: real hexazecimal cu exponent zecimal de 2: `0x`h`.`$hhhh$`p`$\pm d$
`%p`: pointer (de obicei: hexazecimal)
`%n`: scrie în argument (`int *`) nr. de caract. scrise până în prezent;
`%%`: caracterul procent

Formatare: modificatori

Directivele de formatare pot avea *opțional* și alte componente:

% fanion dimensiune . precizie modificator tip

Fanioane: `*`: câmpul e citit, dar nu e atribuit (e ignorat) (`scanf`)

`-`: aliniază valoarea la stânga, la dimensiunea dată (`printf`)

`+`: pune + înainte de număr pozitiv de tip cu semn (`printf`)

spațiu: pune spațiu înainte de nr. pozitiv de tip cu semn (`printf`)

`0`: completează cu `0` la stânga până la dimensiunea dată (`printf`)

Modificatori:

`hh`: argumentul este `char` (la format `d i o u x X n`)

```
char c; scanf("%hhd", &c); // intrare: 123, c = 123 pe 1 octet
char c; scanf("%c", &c);   // intrare: 123, c = '1' (49 ASCII)
```

`h`: argumentul este `short` (la format `d i o u x X n`), ex. `%hd`

`l`: arg. e `long` (format `d i o u x X n`), ex. `long n; scanf("%ld", &n)`

sau `double` (fmt. `a A e E f F g G`), ex. `double x; scanf("%lf", &x)`

`ll`: argumentul este `long long` (la format `d i o u x X n`)

`L`: argumentul este `long double` (la format `a A e E f F g G`)

Formatare: dimensiune și precizie

Dimensiune: un număr întreg
`scanf`: numărul *maxim* de caractere pentru argumentul respectiv
`printf`: numărul *minim* de caractere pe care se scrie argumentul (aliniat la dreapta și completat cu spații, sau cf. modificatorilor)

Precizie: în `printf`; punct `.` urmat de un număr întreg opțional (dacă apare doar punctul, precizia se consideră 0)
numărul *minim* de cifre pentru `diouxX` (completate cu 0)
numărul de cifre zecimale pentru `Eef` / cifre semnificative pt. `Gg`
`printf("|%7.2f|", 15.234);` `|  15.23|` 7 char, 2 zecimale
numărul *maxim* de caractere de tipărit dintr-un șir (pentru `s`)
`char m[3]="ian"; printf("%.3s", m);` (util pt. șir fără `'\0'`)

În `printf`, în locul dimensiunii și/sau preciziei poate apare `*`
Atunci dimensiunea se obține din argumentul următor:
`printf("%.*s", max, s);` scrie cel mult max caractere din șir

Exemple de scriere formatată

Scriere de numere reale în diverse formate:

```
printf("%f\n", 1.0/1100);  // 0.000909 : 6 pozitii zecimale
printf("%g\n", 1.0/1100);  // 0.000909091 : 6 poz. semnificative
printf("%g\n", 1.0/11000); // 9.09091e-05 : 6 poz. semnificative
printf("%e\n", 1.0);       // 1.000000e+00 : 6 cifre zecimale
printf("%f\n", 1.0);       // 1.000000 : 6 cifre zecimale
printf("%g\n", 1.0);  // 1 : fara punct zecimal, zerouri inutile
printf("%.2f\n", 1.009);   // 1.01: 2 cifre zecimale
printf("%.2g\n", 1.009);   // 1: 2 cifre semnificative
```

Scriere de numere întregi în formă de tabel:

```
printf("|%6d|", -12);  |   -12|  printf("|% d|", 12);   | 12|
printf("|%-6d|", -12); |-12   |  printf("|%06d|", -12); |-00012|
printf("|%+6d|", 12);  |   +12|
```

Scriere pe 20 de poziții (`printf` returnează nr. de caractere scrise)

```
int m, n, len = printf("%d", m); printf("%*d", 20-len, n);
```

Exemple de citire formatată

două caractere separate de unic spațiu (citit și ignorat cu `%*1[ ]`)

```
char c1,c2; if (scanf("%c%*1[ ]%c", &c1, &c2) == 2) {/*etc*/}
```

citirea unui întreg cu exact 4 cifre: `unsigned n1, n2, x;`

```
if (scanf(" %n%4u%n", &n1, &x, &n2) == 1 && n2 - n1 == 4) ...
```

(`%n` numără caracterele citite; stocăm contor în n1, n2, scădem)

citește/verifică un cuvânt care trebuie să apară în intrare `int nr=0;`

```
scanf("http://%n", &nr); if (nr == 7) { //a citit "http://"}
else { /* nu ajunge la formatul %n, nr ramane 0 */ }
```

ignoră până la (exclusiv) un caracter (ex. `\n`): `scanf("%*[^\n]");`

Testați după numărul dorit de variabile citite, nu doar număr nenul!

`if (scanf("%d", &n) == 1)` nu doar `if (scanf("%d", &n))`

`scanf` poate returna și `EOF` care e diferit de zero !

Testați depășirea la citirea de întregi, cu `extern int errno;`

```
#include <errno.h>  // declară errno + coduri de eroare
if (scanf("%d", &x) == 1) // test errno pt. depasire; resetam
  if (errno == ERANGE) { printf("numar prea mare"); errno = 0; }
```