Limbaje de programare

Funcții de intrare/ieșire

12 noiembrie 2012

Orice citire trebuie verificată!

Un program nu va primi tot timpul datele pe care le cere.

Utilizatorul poate *greși*, sau poate fi **rău intenționat!**
$\Rightarrow$ programul *trebuie să verifice* că datele au fost citite corect

Evitați **depășirea** la *citirea șirurilor de caractere* și tablourilor
(ne *oprim* din citire când am ajuns la lungimea tabloului)

Depășirile de tablouri corup memoria (și datele din program)
și fac sistemul *vulnerabil* la **atacurile intrușilor**
$\Rightarrow$ printre cele mai **periculoase și costisitoare** erori

Un program prost scris
Un programator ignorant
fac *mult* mai mult rău decât bine.

Cum umplem un tablou evitând depășirea

Adesea un tablou trebuie umplut până la o condiție:

- citire de la intrare până la un anumit caracter (punct, `\n`, etc)
- copiere din alt șir de caractere sau tablou

Trebuie să nu scriem în tablou dincolo de lungimea lui!

```
for (int i = 0; i < len; ++i) { // limitam la lungimea tabloului
   tab[i] = ...;          // pune elementul in tablou
   if (conditie normala de terminare) break/return;
}
// aici se poate testa daca s-a ajuns la limita lungimii
// si semnala daca e cazul
```

Citirea unei linii de text, caracter cu caracter

```
#include <stdio.h>
int rdline(char line[], unsigned size) {
  --size;                              // pastram loc pentru '\0'
  for (int c, i = 0; i < size; ++i) {   // doar pana la size
    if ((c = getchar()) == EOF) { line[i] = '\0'; return i; }
    if ((line[i] = c) == '\n') { line[++i] = '\0'; return i; }
  }
  line[size] = '\0'; return -1;          // linie trunchiata
}
#define LEN 82
int main(void) {
  char s[LEN]; int res;
  if (res = rdline(s, LEN)) {   // nenul, s-a citit ceva
    printf("%s", s);            // tipareste sirul citit
    if (res == -1) puts("\nlinie lunga trunchiata");
    else if (s[res-1] != '\n') puts("\nEOF fara \\n");
  } else puts("EOF, nu s-a citit nimic");
  return 0;
}
```

Citirea unei linii de text: `fgets`

```
char tab[80];
if (fgets(tab, 80, stdin)) { /* s-a citit o linie */ }
else { /* EOF, nu s-a citit nimic */ }
```

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.

Al treilea parametru la `fgets` indică *fișierul* din care se citește: `stdin` (din `stdio.h`) e *intrarea standard* (normal: tastatura)

ATENȚIE! NU facem nicio citire fără verificare!

`fgets` returnează `NULL` dacă n-a citit nimic (sfârșit de fișier),
la succes returnează chiar adresa primită parametru (deci nenulă)

$\Rightarrow$ Testăm că rezultatul e *nenul* pentru a ști dacă s-a citit cu succes

Exemple: citirea liniilor de text

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);         // pune inapoi pe c
   } else printf("linie completa: %s\n", s);
```

Standardul C11 a eliminat funcția ~~gets~~: nu limita citirea

$\Rightarrow$ depășiri, corupere de memorie, vulnerabilități grave de securitate

Scrierea unui șir

```
puts("text urmat de linie noua");
```

Declarație: `int puts(const char *s);`
tipărește șirul `s` urmat de o linie nouă `\n`

```
fputs("text fara linie noua", stdout);
```

`fputs(s, stdout);` e la fel ca `printf("%s", s);`
tipărește șirul `s` ca atare, fără linie nouă suplimentară
`stdout` reprezintă *ieșirea standard* (normal: ecranul)

Declarație: `int fputs(const char *s, FILE *stream);`

`puts` și `fputs` returnează `EOF` la eroare, altfel un nr. natural (≥ 0)

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, %i decimal, %e, %f, %g real, %o octal, %p pointer,
%s șir (cuvânt), %u unsigned, %x heXazecimal

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 cu format: `scanf`

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

Primul parametru: un *șir de caractere*, cu specificatori de format ca la `printf`, dar: *ATENȚIE!* `%f` e `float`, `%lf` e `double`

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

Returnează numărul variabilelor citite (atribuite) (NU valoarea!)
sau `EOF` la eroare sau sfârșitul intrării *înainte* de a citi ceva

ATENȚIE! Orice citire trebuie verificată!

```
double x; float y;
if (scanf("%lf%f", &x, &y) == 2) { /* ok, foloseşte x, y */ }
else { /* eroare: aici o tratam */ }
```

ATENȚIE! În format trebuie dată lungimea șirului!

```
char cuv[30];}
if (scanf("%29s", cuv) == 1) { /* bine: a citit cuvântul */ }
else { /* eroare: aici o tratam */ }
```

NU folosiți niciodată ~~`%s`: `scanf("%s",...)`~~. Duce la depășire.

Tratarea erorilor la citire

Cel mai simplu: *ieșirea din program*
primitiv, dar *mai bine decât continuarea cu eroare!*

Funcția `void exit(int status)` din `stdlib.h` termină execuția.

Putem scrie o funcție care tipărește un mesaj și apelează `exit()`

```
#include <stdlib.h>
void fatal(char *msg)
{
  fputs(msg, stderr);   // fişierul de eroare, normal: ecranul
  exit(EXIT_FAILURE);   // sau exit(1): eroare
}
```

Putem folosi apoi funcția la fiecare citire:

```
if (scanf("%d", &n) != 1) fatal("eroare la citirea lui n\n");
// ajuns aici: ok, folosim pe n
```

Tratarea erorilor la citire

Adesea vrem să citim și prelucrăm repetat ceva. Un tipar simplu:

`while (`*s-a citit bine*`)` *prelucrează*

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

La ieșirea din ciclu se poate testa: `EOF` (sfârșit normal) sau eroare.

`scanf` se oprește când intrarea diferă de format, și NU mai citește
$\Rightarrow$ *consumați intrarea eronată* 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 bine
  for (int c; (c = getchar()) != '\n';) // pana la linie noua
    if (c == EOF) exit(1);              // sfarsitul intrarii
  printf("mai încercaţi o dată: ");
}
// acum putem folosi m şi n
```

Citirea unui cuvânt

Cu formatul `s` citim un *cuvânt* (șir de caractere fără spații).

Tabloul în care citim cuvântul are o dimensiune limitată
$\Rightarrow$ E obligatorie lungimea maximă (un număr) *între* % și `s`
cu 1 mai puțin decât lungimea tabloului, lasă loc pentru `\0`

```
char cuv[33];
if (scanf("%32s", cuv) == 1)
  printf("Cuvantul citit: %s\n", cuv);
```

`scanf` cu formatul `s` 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 se mai pune `&`

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

Citirea anumitor caractere

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

`char a[33]; scanf("%32[A-Za-z_]", a);` max. 32 litere și _
`char num[81]; scanf("%80[0-9]", num);` șir de cifre

ATENȚIE! E obligatorie lungimea limită între % și `[ ]`

Citirea unui șir *cu excepția unor caractere*:
la fel ca mai sus, dar `^` după `[` specifică caracterele *nepermise*

`char t[81]; scanf("%80[^\n.]", t);` până la `.` sau linie nouă

ATENȚIE! Formatul este `[ ]`, NU e urmat de `s`: ~~`%20[A-Z]s`~~

Citirea unui număr fix de caractere

Un caracter:

```
int c = getchar(); if (c != EOF) { /* s-a citit */}
```

sau

```
int c; if ((c = getchar()) != EOF) { /* s-a citit */}
```

Cu `scanf` (putem declara normal ca și `char`)

```
char c; if (scanf("%c", &c) == 1) {/* s-a citit */}
```

Citirea unui *număr fix de caractere*:

```
char tab[80]; if (scanf("%80c", tab) == 1) { /* citit */ };
```

citește EXACT 80 de caractere, *orice* (inclusiv spații albe)
NU adaugă `'\0'` la sfârșit $\Rightarrow$ nu știm dacă s-au putut citi toate

Verificăm dacă s-a ajuns la EOF inițializând și testând lungimea:

```
char tab[81] = "";
scanf("%80c", tab);
int len = strlen(tab); // va fi între 0 şi 80
```

Citirea cu `scanf`: potrivirea cu formatul

În format avem: specificatori cu %, sau *caractere obișnuite*

la `printf`: se tipăresc;

la `scanf`: *trebuie să apară în intrare*

Exemplu: citirea unei date calendaristice în format `zz.ll.aaaa`

```
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 nepotrivite nu se citesc; acele variabile nu se atribuie

`scanf("%d%d", &x, &y);` in: `123A` ret. `1`; `x = 123`, `y`: necitit; rămas: `A`

`scanf("%d%x", &x, &y);` in: `123A` ret. `2`; `x = 123`, `y = 0xA` (10)

Tratarea spațiilor în `scanf`

Formatele *numerice* și `s` consumă și ignoră spații albe inițiale
  `"%d%d"` doi întregi separați și eventual precedați de spații albe

Formatele `c [ ] [^ ]` nu ignoră spații albe (sunt caract. normale)

Un *spațiu alb* în format consumă *oricâte* ≥ 0 spații albe din intrare
`scanf(" ");` consumă spații albe până la primul caracter diferit

`"%c %c"` citește caracter, consumă ≥ 0 spații, citește alt caracter
`"%d %f` e la fel ca `"%d%f"` (spațiile sunt permise oricum)

Atenție! `"%d "` : spațiu după număr consumă toate spațiile după
  (*inclusiv* linii noi!)

Consumăm spații albe, dar nu linie nouă `\n`:
`scanf("%*[\t\v\f\r ]");`

Consumă și ignoră cu `scanf`

Pentru a sări peste (citi fără a folosi) date cu un format dat:
Punem * după %, și nu mai dăm o adresă unde să fie citit
$\Rightarrow$ `scanf` citește după tiparul dat, dar nu pune niciunde datele
și nu se numără ca variabilă citită

Exemplu: text care conține trei note și media, vrem doar media.

```
int media;
if (scanf("%*d%*d%*d%d", &media) == 1) { /* foloseşte */ }
else { /* raportează date în format greşit */ }
```

Exemplu: consumă restul liniei

```
scanf("%*[^\n]");        // consumă până la \n, fără \n
if (getchar() == EOF)    { /* s-a terminat intrarea */ }
// altfel getchar() a citit \n, continuă prelucrarea
```

Precizarea de limite în `scanf`

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 acum
nu citește nimic; nu se numără ca și variabilă citită
`%[···]`: șir de caractere din mulțimea indicată între paranteze
`%[^···]`: șir de caractere exceptând mulțimea dintre 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`; `%x`: cu `a-f`, `%X`: cu `A-F`
`%c`: caracter
`%s`: șir de caractere, până la '\0' sau număr dat ca precizie
`%f`, `%F`: real fără exp.; implicit 6 cifre după . precizie 0: fără punct
`%e`, `%E`: real, cu exp.; implicit 6 cifre după . precizie 0: fără punct
`%g`, `%G`: real, ca `%e`, `%E` dacă exp. $<$ -4 sau $\geq$ precizia; altfel ca `%f`.
Nu tipărește inutile zerouri sau punct zecimal.
`%a`, `%A`: real hexazecimal cu exponent zecimal de 2: `0x`h`.`$hhhh$`p`$\pm d$
`%p`: pointer, uzual în hexazecimal
`%n`: scrie în argument (`int *`) nr. de caractere scrise până acum
`%%`: 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: spațiu înainte de număr pozitiv cu semn (`printf`)
`0`: completează cu `0` la stânga până la dimensiunea dată (`printf`)

Modificatori:
`hh`: argumentul e `char` (la format `d i o u x X n`) (1 octet)

```
char c; scanf("%hhd", &c); // 123 -> c = 123 pe 1 octet
```

`h`: argumentul este `short` (la format `d i o u x X n`), ex. `%hd`
`l`: arg. `long` (format `d i o u x X n`) sau `double` (fmt. `a A e E f F g G`)

```
long n; scanf("%ld", &n); 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 citit pentru acel argument
`printf`: numărul *minim* de caractere pe care se scrie argumentul, aliniat la dreapta și completat cu spații sau conform modificatorilor

Precizie: doar în `printf`; punct `.` urmat opțional de un întreg
(dacă apare doar punctul, precizia se consideră 0)
numărul *minim* de cifre pentru `diouxX` (completate cu 0)
numărul de cifre zecimale (la `Eef`) / cifre semnificative (la `Gg`)
`printf("|%7.2f|", 15.234);` `|  15.23|` 2 zecimale, 7 total
numărul *maxim* de caractere de tipărit dintr-un șir (pentru `s`)
`char m[3]="ian"; printf("%.3s", m);` (util la 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 si 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 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 un singur spațiu (consumat cu %*1[ ])
`char c1, c2; if (scanf("%c%*1[ ]%c", &c1, &c2) == 2) ...`
Citește un 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, apoi scădem
Citește/verifică un cuvânt care trebuie să apară: `int nr=0;`
`scanf("http://%n", &nr); if (nr == 7) { /* apare */ }`
`else { /*nu ajunge la %n, nr ramane cu val. 0 */ }`
Ignoră până la (exclusiv) un caracter (\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 !

Pentru numere întregi, testați depășirea cu `extern int errno;`

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