# Funcţii de intrare/ieşire

25 octombrie 2004





## Funcţii de clasificare a caracterelor (în `ctype.h`)

```
int isalnum(int c) (isalpha(c) || isdigit(c))
int isalpha(int c) ('A' <= c && c <= 'Z' || 'a' <= c && c <= 'z')
int isblank(int c) (c == ' ' || c == '\t')
int iscntrl(int c) /* caracter de control, valoare: 0 - 31 */
int isdigit(int c) ('0' <= c && c <= '9')
int isgraph(int c) /* caracter tipăribil, exceptând spaţiu */
int islower(int c) ('a' <= c && c <= 'z')
int isprint(int c) /* caracter tipăribil, inclusiv spaţiu */
int ispunct(int c) /* tipăribil şi nu alnum şi nu spaţiu */
int isspace(int c) (c == ' ' || c == '\t' || c == '\n' ||
                    c == '\v' || c == '\f' || c == '\r')
int isupper(int c) ('A' <= c && c <= 'Z')
int isxdigit(int c) /* cifră hexazecimală: 0-9, A-F, a-f */
int tolower(int c) /* A - Z -> a - z, restul neschimbat */
int toupper(int c) /* a - z -> A - Z, restul neschimbat */
```





## Funcţii de intrare-ieşire. Generalităţi

Funcţiile discutate: definite în `stdio.h`
– lucrează cu *intrarea/ieşirea standard*
– sunt universale şi portabile, nu dependente de anumite periferice
– tipic: intrarea=tastatura, ieşirea=monitorul, dar pot fi redirectate (se pot lua ca intrare/ieşire orice fişiere)

Citirea de la tastatură se face în mod linie, permite editarea/corectarea
⇒ caracterele introduse sunt stocate temporar în *tamponul de intrare*
apoi sunt preluate rând pe rând, la execuţia citirilor din program
(chiar dacă programul citeşte un număr, utilizatorul poate introduce mai multe; restul vor fi citite ulterior)
`scanf("%d", &x); scanf("%d", &y);` şi `scanf("%d%d", &x, &y);`
au acelaşi efect (pentru `int x, y;`)





## Funcţii de intrare/ieşire pe caractere

În aceste funcţii, caracterele apar ca şi `unsigned char` convertite la `int`
fie valoare 0 .. 255, fie `EOF` = sfârşit de fişier (definit ca -1)
`EOF` introdus de la tastatură: Ctrl-D (UNIX) sau Ctrl-Z (DOS)

`int getchar(void); /* citeşte un caracter de la intrare */`
returnează caracterul citit sau `EOF`
Nu folosiţi `char c = getchar();` Nu se poate compara cu `EOF` !

`int putchar(int c); /* tipăreşte un caracter la ieşire */`
returnează caracterul tipărit, sau `EOF` în caz de eroare

Citirea/scrierea caracter cu caracter şi cea formatată pot fi amestecate liber în program; fiecare continuă de unde s-a oprit precedenta.

NU sunt standard C: `conio.h, getch(), getche(), clrscr()`
⇒ nu folosiţi pentru operaţiunile de intrare/ieşire uzuale !!!





## Exemplu de prelucrare caracter cu caracter

Eliminarea comentariilor dintr-un program C citit de la intrare

```
#include <stdio.h>
int main(void)
{
  int c;
  while ((c=getchar()) != EOF)
    if (c != '/') putchar(c); /* în afara comentariului */
    else if ((c = getchar()) == '*') /* începe comentariul */
      do {
        while (getchar() != '*');
        while ((c = getchar()) == '*'); /* '*': posibilă ieşire */
      } while (c != '/'); /* iese dacă a apărut '/' după '*' */
    else { putchar('/'); putchar(c); } /* '/' fără '*' */
}
```

Obs: presupune că nu apare `EOF` în comentariu (se blochează altfel!)





## Citire/scriere formatată: `scanf()` / `printf()`

`int printf(const char* format, ...); /* tipărire formatată */`
restul parametrilor: *valorile* de tipărit (orice *expresii*)
returnează: numărul de caractere tipărite

`int scanf(const char* format, ...); /* citire formatată */`
restul parametrilor: *adresele* variabilelor de citit
returnează: numărul variabilelor citite (atribuite), sau
`EOF` dacă apare o eroare de intrare *înainte* de citirea primei variabile

Şirul de formatare are o structură similară pentru `printf` şi `scanf`
Poate conţine caractere arbitrare pe lângă directivele de formatare
(se tipăresc pt. `printf`; trebuie să apară în intrare pt. `scanf`)

Tipul argumentelor trebuie să corespundă precis tipurilor specificate în format (pentru `printf`, la nevoie, folosind conversii explicite)

## Citirea formatată (scanf): generalităţi

– citeşte conform tiparului până când datele de intrare nu corespund (fie cu caracterele obişnuite solicitate, fie cu formatul: `%d %f` etc.)
Restul variabilelor rămân neatribuite, iar caracterele necitite rămân în tamponul de intrare. Exemplu: `scanf("test");` intrare: `text\n`
⇒ citeşte `te` iar `xt\n` rămâne în intrare pentru următoarea citire
⇒ trebuie testată valoarea returnată pentru a şti că s-a citit corect
⇒ evtl. trebuie consumată intrarea înainte de a solicita din nou date

```
int m, n;
printf("Introduceţi două numere: ");
while (scanf("%d%d", &m, &n) != 2) {  /* amândouă corect ? */
  while (getchar() != '\n');  /* nu? consumă restul liniei */
  printf("mai încercaţi o dată: ");
}                             /* acum putem folosi m şi n */
```





## Citirea formatată (cont.)

– *spaţiile albe* (vezi `isspace()`) din intrare: separatori impliciţi;
se ignoră înainte de formate numerice şi şir `%s` (nu la caracter `%c`)
⇒ formatele `"%d%f"` şi `" %d %f"` etc. sunt echivalente

– orice spaţiu alb din format consumă *toate* spaţiile albe din intrare (dacă există) până la următorul caracter care nu e spaţiu alb
NU puneţi spaţii la sfârşitul formatului: `"%d\n"` `"%c "` `"%f "` etc.
obligă introducerea unui caracter diferit de spaţiu alb (nu e consumat)

– orice alte caractere din format trebuie să corespundă exact în intrare

– un număr între `%` şi caracterul de format limitează caracterele citite
`%4d` întreg din cel mult 4 caractere (spaţiile iniţiale nu contează)

| Format `scanf` | Intrare | Rezultat |
|---|---|---|
| `scanf("%d%d", &m, &n);` | `12 34` | `12 34` |
| `scanf("%2d%2d", &m, &n);` | `12345` | `12 34` |
| `scanf("%d.%d", &m, &n);` | `12.34` | `12 34` |
| `scanf("%f", &x);` | `12.34` | `12.34` |
| `scanf("%d%x", &m, &n);` | `123a` | `123 0xA` |





## Citirea de şiruri de caractere

La citirea datelor de intrare: utilizatorul poate introduce ORICE !
⇒ trebuie să ne protejăm de date (ne)intenţionat eronate.
Utilizatorul poate introduce mai multe caractere decât memoria alocată
⇒ corupe memoria, termină programul, probleme de securitate !

NU folosiţi `gets()` ! NU folosiţi `scanf("%s", sir)` !
Pentru o citire corectă şi sigură, folosiţi limitări în `scanf`

Citirea unui caracter: `char c;scanf("%c", &c);` Testaţi rezultatul (`EOF`!)

Citirea mai multor caractere: într-un tablou (şir), în limitele acestuia:

– un *număr fix de caractere*: `char s[80]; scanf("%80c", s);`
orice caractere, inclusiv spaţii albe; nu se adaugă automat `'\0'`

– un *cuvânt* (orice până la spaţiu alb) `char s[80]; scanf("%79s", s);`
ignoră spaţii albe iniţiale; adaugă `'\0'` la sfârşit

– o *linie de text*, până la `'\n'` `char s[80]; fgets(s, 80, stdin);`
citeşte max. 80-1 caractere, inclusiv `'\n'`, adaugă `'\0'`
`stdin`: identificator definit în `stdio.h` pt. fişierul standard de intrare





## Citirea formatată: aspecte avansate

Se poate limita citirea la caractere dintr-o mulţime: specificatorul `%[ ]`

– între `[` şi `]` se trec caracterele admise (cu `-` pentru intervale)
Exemplu: `"%32[A-Za-z]"` pentru maxim 32 de litere mari sau mici

– sau cu `^` după `[` se precizează caracterele *nepermise*. Exemplu: `"%80[^!,.:;?]"` pentru maxim 80 de caractere, nu semne de punctuaţie

`char id[33]; scanf("%1[A-Z_a-z]%31[0-9A-Z_a-z]", id, &id[1]);`
citeşte un identificator de max. 32 de caractere, adaugă automat `'\0'`

`char s[81]; scanf("%80[^\n]%*1[\n]", s);` citeşte o linie de max. 80 caractere, şi ignoră (vezi modificatorul `*`) caracterul `'\n'` de la sfârşit, dar eşuează cu `'\n'` necitit dacă se dă o linie goală ⇒ e preferabil `fgets`

Cu specificatorul `%n` se stochează într-o variabilă întreagă numărul caracterelor citite de la intrare ⇒ se pot face anumite verificări.

```
char s[81]; int n; if (scanf("%80[^\n]%n", s, &n) == 1)
                     printf("Linia are lungimea %d\n", n);
```





## Formatare în scanf: tipuri de conversii

`%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 spaţii (doar `" %c"`)
`%s`: şir de caractere, până la primul spaţiu 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 în prezent; nu citeşte nimic; nu incrementează nr. de câmpuri convertite/atribuite
`%[···]`: şir de caractere din mulţimea indicată între paranteze
`%[^···]`: şir de caractere exceptând mulţimea indicată între paranteze
`%%`: caracterul procent





## Formatare în printf: tipuri de conversii

`%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`: şir de caractere, până la `'\0'` sau nr. de caractere dat ca precizie
`%f`, `%F`: real fără exp.; precizie implicită 6 poz.; la precizie 0: fără punct
`%e`, `%E`: real, cu exp.; precizie implicită 6 poz.; la precizie 0: fără punct
`%g`, `%G`: real, ca `%e`, `%E` dacă exp. $< -4$ sau $\geq$ precizia; altfel ca `%f`.
Nu tipăreşte zerouri sau punct zecimal în mod inutil
`%a`, `%A`: real hexazecimal cu exponent zecimal de 2: $0xh.hhhhp\pm d$
`%p`: pointer, în format dependent de implementare (tipic: hexazecimal)
`%n`: scrie în argument (`int *`) nr. de caractere scrise până în prezent;
`%%`: caracterul procent

## Formatare: modificatori

Directivele de formatare pot avea *opţional* şi alte componente:

% *fanion dimensiune . precizie modificator tip*

*Fanioane*: doar pentru `printf`, cu excepţia lui `*` (doar `scanf`)
`*`: `scanf`: câmpul este citit, dar nu e atribuit (e ignorat)
`-`: aliniază valoarea la stânga într-un câmp de dimensiune dată
`+`: pune `+` înainte de număr pozitiv de tip cu semn
*spaţiu*: pune spaţiu înainte de număr pozitiv de tip cu semn
`#`: format alternativ (`0X/0x/0` pt. hex/octal, alte zecimale pt. reali)
`0`: completează cu 0 la stânga până la dimensiunea dată
*Modificatori*:
`hh`: argumentul este `char` (pt. `diouxXn`)
`h`: argumentul este `short` (pt. `diouxXn`)
`l`: argumentul este `long` (pt. `diouxXn`) sau `double` (pt. `aAeEfFgG`)
`ll`: argumentul este `long long` (pt. `diouxXn`)
`L`: argumentul este `long double` (pt. `aAeEfFgG`)





## Formatare: dimensiune şi precizie

*Dimensiune*: un număr întreg
`scanf`: numărul *maxim* de caractere citit pentru argumentul respectiv
`printf`: numărul *minim* de caractere pe care se scrie argumentul
(aliniat la dreapta şi completat cu spaţii, sau conform modificatorilor)

*Precizie*: doar în `printf`; punct . urmat de un număr întreg opţional
(dacă apare doar punctul, precizia se consideră 0)
numărul *minim* de cifre pentru `diouxX` (completate cu 0)
numărul de cifre zecimale pentru `Eef`
numărul de cifre semnificative pentru `Gg`
numărul *maxim* de caractere de tipărit dintr-un şir (pentru `s`)
`char m[3]="ian"; printf("%.3s", m);` (util pt. şir neterminat în '\0')

În `printf`, în locul dimensiunii şi/sau preciziei poate apare `*`, caz în care valoarea se obţine din argumentul următor. Exemplu:

```
printf("%.*s", max, s); /* scrie cel mult max caractere din s */
```





## Exemple de scriere formatată

Scriere de numere reale în diverse formate:

```
printf("%f\n", 1.0/1100); /* 0.000909 : 6 poz. 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 : fără 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("|%-6d|", -12);  /* |-12   | */
printf("|%+6d|", 12);   /* |   +12| */
printf("|% d|", 12);    /* | 12|    */
printf("|%06d|", -12);  /* |-00012| */
```





## Exemple de citire formatată

– ora şi minute separate cu : între ele
`unsigned h, m; if (scanf("%u:%u", &h, &m) == 2) { /* etc */ }`
– două caractere separate de un singur spaţiu
`char c1, c2; if (scanf("%c%*1[ ]%c", &c1, &c2) == 2) {/* etc */}`
– citirea unui întreg cu nr. fix de cifre (ex. 4): `unsigned n1, n2, x;`
`if (scanf(" %n%4u%n", &n1, &x, &n2) == 1 && n2 - n1 == 4) /* etc */`
– eliminarea spaţiilor: `scanf(" ");`
– ignorarea până la un caracter dat, ex. virgula: `scanf("%*[^,],");`

Testaţi după numărul dorit de variabile citite, nu doar număr nenul!
`if (scanf("%d", &n) == 1)` şi nu doar `if (scanf("%d", &n))`
`scanf` poate returna şi `EOF` care e diferit de zero !

Pentru numere întregi, testaţi şi depăşirea, folosind `extern int errno;`

```
#include <errno.h>
if (scanf("%d", &x) == 1))
  if (errno == ERANGE) { printf("număr prea mare"); errno = 0; }
  /* errno trebuie resetat după eroare */
```





## Fişiere standard de intrare/ieşire

Normal, într-un program: citirea de la tastatură, tipărirea pe ecran

Folosirea funcţiilor *standard* din `stdio.h` permite automat *redirectarea* intrării şi a ieşirii = efectuarea lor (d)in alt loc, precizat la rulare

Exemplu: pe linia de comandă
*prog* < *fişier* citirea (intrarea) lui *prog* se face din *fişier*
*prog* > *fişier* tipărirea (ieşirea) lui *prog* se face în *fişier*
*prog1* | *prog2* ieşirea lui *prog1* se transmite direct la *prog2*

Exemplu: copiere pe caractere de la intrare la ieşire până la `EOF`

```
#include <stdio.h>
void main(void)
{
  int c;
  while ((c = getchar()) != EOF) putchar(c);
}
```

Se pot copia două fişiere: *nume-program* < *fişier-sursă* > *fişier-dest*