

# Funcții de intrare/ieșire

23 noiembrie 2004

## Funcții 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 spațiu */
int islower(int c) ('a' <= c && c <= 'z')
int isprint(int c) /* caracter tipăribil, inclusiv spațiu */
int ispunct(int c) /* tipăribil și nu alnum și nu spațiu */
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 */
```

## Funcții de intrare-ieșire. Generalități

---

Funcțiile discutate: definite în `stdio.h`

- lucrează cu *intrarea/ieșirea standard*
- sunt universale și portabile, nu dependente de anumite periferice
- tipic: intrarea=tastatura, ieșirea=monitorul, dar pot fi redirectate (se pot lua ca intrare/ieșire orice fișiere)

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 execuția citirilor din program  
(chiar dacă programul citește un număr, utilizatorul poate introduce mai multe; restul vor fi citite ulterior)

```
scanf("%d", &x); scanf("%d", &y);    și    scanf("%d%d", &x, &y);  
au același efect (pentru int x, y;)
```

## Funcții de intrare/ieșire pe caractere

---

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

```
int getchar(void); /* citește un caracter de la intrare */  
    returnează caracterul citit sau EOF
```

Nu folosiți `char c = getchar();` Nu se poate compara cu EOF !

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

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

NU sunt standard C: `conio.h`, `getch()`, `getche()`, `clrscr()`

⇒ nu folosiți pentru operațiunile de intrare/ieșire 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 ieșire */
            } 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

Șirul de formatare are o structură similară pentru `printf` și `scanf`

Poate conține 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): generalități

---

- citește conform tiparului până când datele de intrare nu corespund (fie cu caracterele obișnuite 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`
- ⇒ citește `te` iar `xt\n` rămâne în intrare pentru următoarea citire
- ⇒ trebuie testată valoarea returnată pentru a ști că s-a citit corect
- ⇒ evtl. trebuie consumată intrarea înainte de a solicita din nou date

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

- *spațiile albe* (vezi `isspace()`) din intrare: separatori implicați; se ignoră înainte de formate numerice și șir `%s` (nu la caracter `%c`)  
⇒ formatele `"%d%f"` și `" %d %f"` etc. sunt echivalente
- orice spațiu alb din format consumă *toate* spațiile albe din intrare (dacă există) până la următorul caracter care nu e spațiu alb  
NU puneți spații la sfârșitul formatului: `"%d\n"` `"%c "` `"%f "` etc.  
obligă introducerea unui caracter diferit de spațiu alb (nu e consumat)
- orice alte caractere din format trebuie să corespundă exact în intrare
- un număr între `%` și caracterul de format limitează caracterele citite  
`%4d` întreg din cel mult 4 caractere (spațiile inițiale nu contează)

Format scanf	Intrare	Rezultat
<code>scanf("%d%d", &amp;m, &amp;n);</code>	12 34	12 34
<code>scanf("%2d%2d", &amp;m, &amp;n);</code>	12345	12 34
<code>scanf("%d.%d", &amp;m, &amp;n);</code>	12.34	12 34
<code>scanf("%f", &amp;x);</code>	12.34	12.34
<code>scanf("%d%x", &amp;m, &amp;n);</code>	123a	123 0xA



La citirea datelor de intrare: utilizatorul poate introduce ORICE !

⇒ trebuie să ne protejăm de date (ne)intenționat eronate.

Utilizatorul poate introduce mai multe caractere decât memoria alocată

⇒ corupe memoria, termină programul, probleme de securitate !

**NU folosiți `gets()` !**      **NU folosiți `scanf("%s", sir)` !**

Pentru o citire corectă și sigură, folosiți limitări în `scanf`

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

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

– un *număr fix de caractere*: `char s[80]; scanf("%80c", s);`

orice caractere, inclusiv spații albe; nu se adaugă automat `'\0'`

– un *cuvânt* (orice până la spațiu alb) `char s[80]; scanf("%79s", s);`

ignoră spații albe inițiale; adaugă `'\0'` la sfârșit

– o *linie de text*, până la `'\n'` `char s[80]; fgets(s, 80, stdin);`

citește max. 80-1 caractere, inclusiv `'\n'`, adaugă `'\0'`

`stdin`: identificator definit în `stdio.h` pt. fișierul standard de intrare

## Citirea formatată: aspecte avansate

---

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

– între [ și ] 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 punctuație

```
char id[33]; scanf("%1[A-Z_a-z]%31[0-9A-Z_a-z]", id, &id[1]);
```

citește un identificator de max. 32 de caractere, adaugă automat '\0'

```
char s[81]; scanf("%80[^\n]*1[\n]", s);
```

citește o linie de max. 80 caractere, și ignoră (vezi modificatorul \*) caracterul '\n' de la sfârșit, dar eșuează 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 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 în prezent; nu citește nimic; nu incrementează nr. de câmpuri convertite/atribuite

%[...]: șir de caractere din mulțimea indicată între paranteze

%[^...]: șir de caractere exceptând mulțimea 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`: șir 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  $\text{exp.} < -4$  sau  $\geq$  precizia; altfel ca `%f`.

Nu tipărește zerouri sau punct zecimal în mod inutil

`%a`, `%A`: real hexazecimal cu exponent zecimal de 2: `0xh.hhhhp±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

---

Directivile de formatare pot avea *optional* și alte componente:

*% fanion dimensiune . precizie modifier tip*

*Fanioane*: doar pentru printf, cu excepția lui \* (doar scanf)

\*: scanf: câmpul este citit, dar nu e atribuit (e ignorat)

-: aliniaza valoarea la stânga într-un câmp de dimensiune dată

+: pune + înainte de număr pozitiv de tip cu semn

*spațiu*: pune spațiu 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 și 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 și completat cu spații, sau conform modificatorilor)

**Precizie:** doar în `printf`; punct . urmat de un număr întreg opțional (dacă apare doar punctul, precizia se consideră 0)

numărul *minim* de cifre pentru `dioouxX` (completate cu 0)

numărul de cifre zecimale pentru `Eef`

numărul de cifre semnificative pentru `Gg`

numărul *maxim* de caractere de tipărit dintr-un șir (pentru `s`)

`char m[3]="ian"; printf("%.3s", m);` (util pt. șir neterminat în `'\0'`)

În `printf`, în locul dimensiunii și/sau preciziei poate apare `*`, caz în care valoarea se obține 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 și minute separate cu : între ele

```
unsigned h, m; if (scanf("%u:%u", &h, &m) == 2) { /* etc */ }
```

– două caractere separate de un singur spațiu

```
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 spațiilor: `scanf(" ");`

– ignorarea până la un caracter dat, ex. virgula: `scanf("%*[^,],");`

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

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

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

Pentru numere întregi, testați și depășirea, 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 */
```



## Fișiere standard de intrare/ieșire

---

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

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

Exemplu: pe linia de comandă

*prog* < *fișier*            citirea (intrarea) lui *prog* se face din *fișier*

*prog* > *fișier*            tipărirea (ieșirea) lui *prog* se face în *fișier*

*prog1* | *prog2*            ieșirea lui *prog1* se transmite direct la *prog2*

Exemplu: copiere pe caractere de la intrare la ieșire până la EOF

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

Se pot copia două fișiere: *nume-program* < *fișier-sursă* > *fișier-dest*