# Funcții de intrare/ieșire

26 octombrie 2005

# Funcţii de intrare-ieşire. Generalităţi

Toate 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: *în mod linie* $\Rightarrow$ permite editarea/corectarea
$\Rightarrow$ caracterele introduse sunt stocate temporar în *tamponul de intrare* apoi sunt preluate rând pe rând, la execuţia citirilor din program
$\Rightarrow$ daca pe o linie se introduce mai mult decat se cere la prima citire restul datelor vor fi preluate de apeluri de citire ulterioare

```
printf("x = "); scanf("%d", &x); printf("y = "); scanf("%d", &y);
```

daca se introduc `4 5` pe aceeasi linie, apare:

```
x = 4 5
y =
```

(lui `y` se atribuie `5` deja introdus, programul continua)

# 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` *nu e un caracter* ci o valoare *diferita* de orice caracter
valoarea $\neq$ caracterul tastat din terminal: `Ctrl-D (UNIX)/Ctrl-Z (DOS)`

```
int getchar(void); /* citeşte un caracter de la intrare */
```

returnează caracterul citit (convertit la `int 0 .. 255` sau `EOF`
NU folosiţi `char c = getchar();` `char` nu se compara corect cu `EOF` !
– un `unsigned char` nu va da niciodata `== -1 (EOF)`
– pt. `signed char`, caracterul (obisnuit) 255 e convertit la `-1 == EOF`

```
int putchar(int c); /* tipăreşte un caracter la ieşire */
```

returnează caracterul tipărit, sau `EOF` în caz de eroare

NU sunt standard C: `conio.h, getch(), getche(), clrscr()`
$\Rightarrow$ nu folosiţi pentru intrare/ieşire in programe obisnuite, portabile !!!

# 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`
(la eroare de intrare *înainte* de a incepe citirea primului camp)

Şirul de formatare: structură similară pentru `printf` şi `scanf`
conţine *specificatori de formatare* (cu %) dar si caractere obisnuite
(se tipăresc pt. `printf`; *trebuie să apară* în intrare pt. `scanf`)

Tipul argumentelor trebuie *să corespundă* precis tipurilor din format
(verificarea e sarcina utilizatorului; compilatorul eventual avertizeaza)

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

# Citirea formatată (`scanf`): generalităţi

– citeşte conform tiparului *până când caract. introduse nu corespund* (fie cu caracterele obişnuite solicitate, fie cu formatul: `%d` `%f` etc.)
– caracterele necitite *rămân în tamponul de intrare* (nu se sare peste!)
– restul variabilelor *ramân neatribuite*

Exemplu: `scanf("test");` intrare: `text\n`
⇒ citeşte `te` iar `xt\n` rămâne în intrare pentru următoarea citire
⇒ trebuie testată valoarea returnată pentru a şti dacă 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 citite corect ?
  while (getchar() != '\n');          // nu? consumă restul liniei
  printf("mai încercaţi o dată: "); // altfel ne putem bloca
}                                     // la iesire putem folosi m şi n
```

# Citirea formatată (cont.)

– *spațiile albe* (vezi `isspace()`) din intrare: separatori impliciți;
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 |
|---|---|---|
| `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 șiruri de caractere

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 numărul caracterelor citite de la intrare
în variabila întreagă cu adresa data ⇒ se pot face diverse verificări

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

# 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 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, 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 *opțional* și alte componente:

% *fanion dimensiune . precizie modificator tip*

*Fanioane*: doar pentru `printf`, cu excepția 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

*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 `diouxX` (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 spaţiu

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

– un întreg cu numar fix de cifre (ex. 4): `unsigned n1, n2, x;`

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

– eliminarea spaţiilor albe: `scanf(" ");`

– ignorarea până la un caracter dat, ex. `\n`: `scanf("%*[^,],");` Atentie: `while (getchar() != '\n')` se poate bloca la `EOF` !

Testaţi `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, testaţi 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*

# Testarea sfârșitului de fișier (standard de intrare?)

1. În orice punct de program: `feof(stdin)` mai general: `int feof(FILE *fp)` returnează nenul (adevărat) dacă s-a atins sfârșitul intrarii; 0 dacă nu

2. După valoarea returnată de funcțiile de intrare:

`getchar()` returneaza `EOF` (`-1`, valoare diferita de orice caracter)
Pt. test corect, valoarea trebuie atribuita unui `int`, nu `char`

`scanf` returnează `EOF` dacă intrarea se termina inainte de a citi ceva
$\Rightarrow$ Testati citirea corecta cu `if (scanf(...) == nr_campuri_dorite)`
nu doar `if ((scanf(...))` (și `EOF` e nenul)

`fgets` returnează `NULL` dacă intrarea se termină înainte de a citi ceva
Exemplu: prelucrarea unui fișier linie cu linie

```
char lin[128]; while (fgets(lin, 128, stdin)) /* prelucrează lin */
```

# Citirea anumitor categorii de caractere

*Atenţie*: Funcţiile din `ctype.h` returnează *nenul* pentru un caracter de felul dorit și 0 în caz contrar (*NU* neapărat 1 și 0)
nu scrieți niciodată `if (isalpha(c) == 1)` ci doar `if (isalpha(c))`

*Atenţie* la cicluri infinite pentru sfârșit de fișier:

```
int c; while (isdigit(c = getchar())) /* ceva */
```

va ieși din ciclu când `c` nu e cifră, inclusiv la `EOF` (nu e cifră)

```
int c; while (!isdigit(c = getchar())) /* ceva */
```

se va bloca la `EOF`, pentru că nu e cifră (nici `isalpha`, `isspace`, etc.)
Corect: cu test suplimentar de `EOF`

```
while (!isdigit(c = getchar()))
  if (c == EOF) break; /* sau ce vrem să facem la EOF */
  else /* restul prelucrării */
```

# Citirea și prelucrarea până la sfârșit de fișier

Un fisier = sir de octeti ⇒ `EOF`: nu mai sunt octeti de citit
`EOF` e rezultatul unor functii, NU un fanion fizic la sfarsit de fisier!
Indicatorul de sfârșit de fișier e poziționat doar când se încearcă citirea *dincolo* de sfârșitul fișierului, nu când s-a citit ultimul caracter.
⇒ după ultima citire cu succes, `feof()` poate fi adevărat sau nu
Ex: pt. un fișier de întregi separați prin spații, `feof()` e poziționat după citirea ultimului doar dacă nu e urmat de altceva (ex. spațiu, `\n`)
Ex: la citirea linie cu linie, `feof()` e poziționat după citirea ultimei linii doar dacă ea nu se termină cu `\n`.
⇒ dacă `feof()` e fals, fișierul poate să mai conțină un element sau nu

In general, nu e suficient testul de `feof()` nici inainte, nici dupa o citire ci trebuie testat *rezultatul corect* al citirii !

# Citirea si prelucrarea pana la sfarsit de fisier (cont.)

```
int n, s = 0;
while (!feof(stdin)) {
  scanf("%d", &n);
  s += n;
}
```

```
for (;;) {
  scanf("%d", &n);
  if (feof(stdin)) break;
  s += n;
}
```

Varianta 1: duplica ultimul numar daca apar spatii albe inainte de `EOF` (`feof()` da fals, ultima citire esueaza, numarul ramane nemodificat)
Varianta 2: pierde ultimul numar daca e urmat direct de `EOF`

Trebuie testată citirea corectă (`getchar() != EOF`, `fgets(...) != NULL`, valoarea lui `scanf`), și tratat cazul de eroare (ex. ieșirea din buclă)

```
int n, s = 0;
while (scanf("%d", &n) == 1)
  s += n;
```

```
for (;;) {
  if (citeste() != CORECT) break;
  prelucrează();
}
```

# Refacerea ultimului caracter citit

Uneori sfarsitul a ceea ce dorim sa citim poate fi detectat doar citind primul caracter *nedorit*. De exemplu:
– un numar: la primul caracter care nu e cifra
– un cuvant: la primul caracter spatiu alb
⇒ adesea, am vrea sa prelucram doar ulterior caracterul citit in plus

```
ungetc(c, stdin); // pune caracterul c înapoi in intrare
```

in general: `int ungetc(int c, FILE *stream);`
– de fapt, caracterul e pus inapoi intr-o locatie speciala
de unde e returnat apoi de urmatoarea citire (`getchar()`, `scanf()`, etc.)
⇒ nu poate fi apelata repetat consecutiv, are loc doar un caracter
(decat dupa alta citire intermediara)

# Citirea/scrierea din/in siruri de caractere

Putem folosi aceleasi facilitati de formatare pentru prelucrari de siruri:

```
int sprintf(char *s, const char *format, ...);
int sscanf(const char *s, const char *format, ...);
```

Diferente de retinut:

La `sprintf`, poate apărea problema depășirii tabloului în care se scrie, dacă acesta nu e dimensionat corect (suficient). Se recomandă:

```
int snprintf(char *str, size_t size, const char *format, ...);
```

în care scrierea e limitată la `size` caractere ⇒ variantă sigură

ambele termina sirul cu \0 ⇒ `snprintf` scrie max. `size-1` caractere utile

ambele returneaza nr. de caractere (care ar fi fost) scrise fara limitare ⇒ `snprintf` nu a trunchiat ⇔ val. returnata e pozitiva si `< size`

`sscanf` nu modifica sirul prelucrat, si nu indica direct avansarea in el (spre deosebire de citirea de la intrare, unde se consuma caractere) ⇒ pentru a afla lungimea prelucrata (pentru a continua din acel punct), folosim formatul `%n` (numarul de caractere citite)