Citirea unei linii de text

`char *fgets(char *s, int size, FILE *stream);`
Citește până la (inclusiv) linie nouă, sau max. `size - 1` caractere, pune linia în tabloul `s`, adaugă `'\0'` la sfârșit
Exemplu: `char tab[80]; fgets(tab, 80, stdin);`
Parametrul al treilea: un *fișier* (detalii ulterior); pentru citirea de la intrarea standard, folosim identificatorul `stdin` (din `stdio.h`)

`fgets` returnează `NULL` (valoarea zero, invalidă pentru adrese) dacă nu poate citi nimic (a ajuns la sfârșit de fișier); în caz de succes returnează chiar adresa primită parametru

Exemplu: citirea linie cu linie până la sfârșitul intrării; liniile mai lungi de 40 caractere se afișează pe bucăți:
`char s[41]; while (fgets(s, 41, stdin)) printf("%s", s);`
ATENȚIE! Nu folosiți funcție `gets` ! Ea nu are ca parametru lungimea maxim disponibilă. Se poate depăși zona de memorie alocată șirului !

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

```
int printf(const char* format, ...); // tipărire formatată
```
restul parametrilor: *valorile* de tipărit (*expresii arbitrare*)
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 (sfârșit de fișier) *înainte* de citirea primei variabile

```
int b=4; printf("%d%f", 2*b+3, acos(0)); // tipăreşte întreg şi real
double a[4]; int n; scanf("%d%f", &n, &a[0]); //citeşte întreg, real
```

Tipul argumentelor trebuie să corespundă tipurilor indicate în format

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

Citirea unui cuvănt

Un *cuvânt* = secvenţă de *orice* caractere diferite de spaţii albe.
Exemplu: citirea unui cuvânt de maxim 20 caractere:

```
char c[21]; if (scanf("%20s", c) == 1) { /* s-a citit bine */ }
```

– litera de format pentru citirea unui cuvânt e `s` (string)
– la începutul citirii se sare peste oricâte spaţii albe iniţiale
– citirea se opreşte la primul spaţiu alb (rămâne în intrare) sau după numărul de caractere indicat între `%` şi `s`
– la citire cu succes, şirul e terminat automat cu `\0`
⇒ trebuie tablou cu 1 mai lung ca nr. de caractere permise la citire

Într-un program corect, e *obligatorie* specificarea între `%` şi `s` a unui număr maxim de caractere permise la citire ! Altfel, utilizatorul poate introduce mai multe caractere decât memoria alocată
⇒ corupe memoria, termină programul, probleme de securitate !
În general, utilizatorul poate introduce ORICE !
⇒ trebuie să ne protejăm de date (ne)intenţionat eronate.

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.)
– caracterele necitite rămân în tamponul de intrare
– variabilele necitite cu succes rămân neatribuite
Ex.: `scanf("%d:%d", &m, &n);` (doi `int` cu `:` între) intrare: `12 34\n`
⇒ citește 12 în `m` și `34\n` rămâne în intrare pt. 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("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: spații albe/formate numerice

– orice spațiu alb (v. `isspace`) din format consumă *toate* spațiile albe din intrare (dacă există) până la următorul caract. care nu e spațiu alb
– spațiile albe se consumă și ignoră înainte de formate numerice (`%d` `%f` etc.) și șir `%s` (nu la caracter `%c`) ⇒ nu e nevoie să scriem `"%d %d"`
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 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'`

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`: `float` (posibil cu exponent)

Atenție, pentru `double` trebuie prefix `l`, deci `%lf`

`%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 */}
```

– 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)` 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 */
```