Programarea calculatoarelor

Funcţii de intrare/ieşire

14 aprilie 2009

Citirea unei linii de text

`char *fgets(char *s, int size, FILE *stream);  //decl. in stdio.h`
Citeşte până la (şi inclusiv) linie nouă `\n`, dar max. `size-1` caractere
pune linia în tabloul `s`, adaugă `'\0'` la sfârşit
Exemplu: `char tab[80]; fgets(tab, 80, stdin);`
Parametrul al treilea: un *fişier* (discutăm ulterior); pentru a citi de la *intrarea standard*, folosim identificatorul `stdin` (din `stdio.h`).

ATENŢIE! `fgets` returnează `NULL` dacă n-a citit nimic (sfârşit de fişier)
la succes: returnează chiar adresa primită parametru (deci nenulă)
⇒ Testăm de rezultat nenul pentru a şti dacă s-a citit cu succes:

Exemplu: citire + afişare linie cu linie până la sfârşitul intrării
`char s[81]; while (fgets(s, 81, stdin)) printf("%s", s);`
liniile mai lungi de 80 de caractere se citesc/afişează pe bucăţi

ATENŢIE! NU folosiţi funcţia `gets` ! Nu se poate specifica lungimea maxim disponibilă ⇒ se poate depăşi zona de memorie alocată! ⇒ program abandonat, corupere de memorie, vulnerabilităţi de securitate

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

`int printf(const char* format, ...); // tipărire formatată`
returnează: nr. de caractere tipărite; în format: specificatori
`%c` char, `%d` decimal, `%f` float, `%p` pointer, `%s` sir, `%u` unsigned, `%x` heXa
restul parametrilor: *valorile* de tipărit (orice *expresii*)

`int scanf(const char* format, ...); // citire formatată`
restul parametrilor: *adresele* variabilelor de citit: `&` (mai puţin la şiruri)

Exemple: `int n; scanf("%d", &n);` `float a[4]; scanf("%f", &a[2]);`
Format: ca `printf`, cu unele diferenţe. Ex. `%f` float, `%lf` double
returnează: *numărul* variabilelor citite (atribuite) (NU valoarea!), sau `EOF` dacă apare o eroare de intrare *înainte* de citirea primei variabile
⇒ folosim rezultatul pentru a testa dacă s-au citit cele dorite:
`int n; if (scanf("%d", &n)==1) {/* s-a citit*/} else {/* eroare */}`

ATENŢIE! Utilizatorul poate introduce *orice* şi *oricât* ⇒ la orice citire (`fgets`, `getchar`, `scanf` etc.) *rezultatul returnat trebuie testat*!

Citirea formatată (`scanf`): tratarea intrării

Pe lângă specificatori, şirul de format poate avea *caractere obişnuite*
la `printf`: se tipăresc; la `scanf`: *trebuie să apară în intrare*
`unsigned z, l, a; scanf("%u.%u.%u", &z, &l, &a);  // 15.4.2008 cu .`

`scanf` citeşte până când intrarea *nu corespunde* formatului, apoi revine
Restul variabilelor nu se atribuie; caracterele necitite rămân în intrare
`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)

La eroare trebuie *consumată intrarea* înainte de a solicita din nou date:

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

Tipar uzual: `while (`*citit bine*`) `*prelucrează*` ; ` `while (fgets(...))...`
`while ((c = getchar()) != EOF)...` `while (scanf(...) == `*câte-cer*`)...`

Citirea de şiruri de caractere

Citirea unui caracter: `char c; scanf("%c", &c);` testaţi rezultatul (1?)
Mai simplu: `int c = getchar(); if (c != EOF) // valoare de char`

Citirea mai multor caractere: într-un tablou (şir), în limitele acestuia:
– un *cuvânt* (orice până la spaţiu alb) `char t[33]; scanf("%32s", t);`
consumă şi ignoră spaţii albe iniţiale; adaugă `'\0'` la sfârşit

– un *număr fix de caractere*: `char tab[80]; scanf("%80c", tab);`
orice caractere, inclusiv spaţii albe; NU adaugă `'\0'` la sfârşit

– şir dintr-o *mulţime de caractere* (- pt. intervale) %*lung*[*permise*]
`char a[33]; scanf("%32[A-Za-z_]", a);` max. 32 litere sau _

– şir *cu excepţia unor caractere* %*lung*[^*excluse*]
`char t[81]; scanf("%80[^,.0-9]", t);` max. 80, până la cifră , sau .

ATENŢIE! Numele de tablou *e o adresă*, nu mai trebuie pus `&`
E *obligatorie lungimea* între `%` şi `s` `[]` `[^]` (-1 faţă de tablou, pt. `'\0'`)
lipsa e o *eroare gravă* ⇒ corupere de memorie, atacuri de securitate
Formatul `s` citeşte cuvânt (până spaţii), nu linie! Formatul `[]` NU e ~~`[]s`~~

`scanf`: separatori, limitare

– formatele *numerice* şi `s` consumă (sar peste) spaţii albe iniţiale
`"%d%d"` doi întregi separaţi şi eventual precedaţi de spaţii albe
– formatele `c` `[ ]` `[^ ]` nu sar peste spaţii albe; ele nu au rol special
– un *spaţiu alb* în şirul de format consumă *oricâte* spaţii albe din intrare
`scanf(" ");` consumă toate spaţiile albe până la primul alt caracter
`"%c %c"` citeşte caracter arbitrar, consumă spaţii, citeşte alt caracter
`"%d %f"` acelaşi efect ca `"%d%f"` (spaţiile sunt permise oricum)
Atenţie! `"%d "` : spaţiu după număr consumă spaţii până la altceva!

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

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`; 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: $0\mathtt{x}h.hhh\mathtt{p}\pm d$
`%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: `*`: câmpul este 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: pune spațiu înainte de număr pozitiv de tip cu semn (`printf`)
`0`: completează cu `0` la stânga până la dimensiunea dată (`printf`)

Modificatori:
`hh`: argumentul este `char` (la format `diouxXn`)
`char c; scanf("%hhd", &c); // intrare: 123, c = 123 pe 1 octet`
dar: `char c; scanf("%c", &c); // intrare: 123, c = '1' (49 ASCII)`
`h`: argumentul este `short` (la format `diouxXn`), ex. `%hd`
`l`: arg. este `long` (format `diouxXn`), ex. `long n; scanf("%ld", &n);`
sau `double` (format `aAeEfFgG`), ex. `double x; scanf("%lf", &x);`
`ll`: argumentul este `long long` (la format `diouxXn`)
`L`: argumentul este `long double` (la format `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` / cifre semnificative pentru `Gg`
`printf("|%7.2f|", 15.234);` `|  15.23|` 2 zecimale, 7 caract. total
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 `*`
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 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("|% d|", 12);    | 12|
printf("|%-6d|", -12);  |-12   |   printf("|%06d|", -12);  |-00012|
printf("|%+6d|", 12);   |   +12|
```

Scriere pe total 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 (citit și ignorat cu `%*1[ ]`)

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

– citirea unui întreg cu exact 4 cifre: `unsigned n1, n2, x;`

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

(`%n` numără caracterele citite; stocăm contor în n1, n2, apoi scădem)

– citește/verifică un cuvânt care trebuie să apară în intrare `int nr=0;`

```
scanf("http://%n", &nr); if (nr == 7) { /*apare, s-a citit tot*/ }
else { /*nu ajunge la formatul %n, nr ramane cu val. dinainte 0*/ }
```

– ignoră până la (exclusiv) caracter anume (ex. `\n`): `scanf("%*[^\n]");`

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