Limbaje de Programare
Curs 6 – Funcții de intrare-ieșire

Dr. Casandra Holotescu

Universitatea Politehnica Timișoara

Ce discutăm azi...

1. Citire formatată
2. Citirea șirurilor de caractere
3. Citirea unor linii de text
4. Scriere formatată
5. Formatare. Specificatori de format

Citire formatată cu **scanf**

int scanf(const char* format, ...);

în format putem avea specificatorii:

- %c – char
- %d – întreg decimal cu semn
- %ld – long
- %u – întreg decimal fără semn
- %x – întreg în baza 16
- %f – real
- %lf – double
- %p – pointer (adresă)
- %s – șir de caractere

Citire formatată cu **scanf**

int scanf(const char* format, ...);

Restul parametrilor – adresele variabilelor în care se citește:
&nume-variabila

Atenție!
variabilele în care se citește trebuie să fie de
același tip cu tipul specificat în format!

verificarea potrivirii tipurilor este sarcina programatorului!
nu rezultă în eroare de compilare!

scanf returnează:
numărul variabilelor citite (atribuite) (NU valoarea!)
sau EOF (pt. eroare de intrare înainte de citirea primei var.)

Citire cu **scanf** – exemple

```
int n;
scanf("%d", &n);

long x, y, z;
scanf("%ld %ld %ld", &x, &y, &z);

float a[4];
scanf("%f", &a[2]);
```

Citire formatată cu **scanf**

scanf returnează:

numărul variabilelor citite
sau EOF

$\Rightarrow$ rezultatul returnat de scanf se poate folosi pentru a testa dacă s-au citit toate datele dorite

```
int n, m;
if (scanf("%d %d", &n, &m)==2) {
      /* s-au citit ambele valori*/
  }else {
      /* eroare */
  }
```

Atenție: Utilizatorul poate introduce **orice și oricât** $\Rightarrow$ la orice citire, **rezultatul returnat trebuie testat**!

Citire formatată cu **scanf**

Pe lângă specificatori, șirul de format poate conține și caractere obișnuite:

- la **printf**: se tipăresc
- la **scanf**: trebuie să apară în intrare

Exemplu pentru scanf:

```
unsigned z, l, a;
scanf("%u.%u.%u", &z, &l, &a);
// citeste o data in format cu punct
// de exemplu 15.12.2015 cu .
```

scanf citește **doar cât timp intrarea corespunde formatului**!

Citire formatată cu **scanf**

scanf citește **doar cât timp intrarea corespunde formatului**!
Când intrarea nu mai corespunde:

- funcția returnează nr. de variabile atribuite
- variabilele rămase necitite **nu se mai atribuie**
- caracterele necitite **rămân în bufferul de intrare**

scanf("%d%d", &x, &y);
intrare: *123A*
⇒ returnează 1
⇒ x=123, y: necitit, intrare: *A*

scanf("%d%x", &x, &y);
intrare: *123A*
⇒ returnează 2
⇒ x=123, y=0XA (10)

Citire formatată cu **scanf**

Deoarece când intrarea nu mai corespunde:

caracterele necitite **rămân în bufferul de intrare**

$\Rightarrow$ ele trebuie **eliminate din buffer**!

La eroare trebuie **consumată intrarea** înainte de a citi din nou!

```
int m, n;
printf("Introduceti doua numere: ");

while (scanf("%d %d", &m, &n) != 2) {
 // cat timp nu e corect
    while (getchar() != '\n');
    // consuma restul liniei
    printf("mai incercati o data: ");
} //dupa iesirea din while putem folosi m si n
```

Citirea șirurilor de caractere

Citirea mai multor caractere, într-un tablou, nedepășind limitele tabloului (obligatoriu **%lungime** în format!):

- un **cuvânt** (orice până la spațiu alb): adaugă '\0' la sfârșit, consumă și ignoră spațiile albe de la început

```
char t[33];
scanf("%32s", t);
```

- un **număr fix de caractere**: orice caractere, inclusiv spații albe, nu adaugă '\0' la sfârșit

```
char t[33];
scanf("%33c", t);
```

Citirea șirurilor de caractere

Citirea mai multor caractere – continuare:

- un șir dintr-o **mulțime de caractere**:
  %lungime[caract-permise]
  limitează caracterele permise la un subset (cu - pt. interval)
  adaugă '\0' la sfârșit

```
char t[33];
scanf("%32[A-Za-z!?]", t);
//litere mari, mici si !?
```

- un șir **cu excepția unor caractere**:
  %lungime[^caract-excluse]
  orice caracter ce nu e în caract-excluse, '\0' la sfârșit

```
char t[33];
scanf("%32[^0-9.,]", t);
//fara cifre, punct sau virgula
```

Citirea liniilor cu **fgets**

char *fgets(char *s, int size, FILE *stream);

Citește:

- până la linie nouă '\n' (inclusiv).
- maxim size-1 caractere
- pune linia în tabloul s
- adaugă '\0' la sfârșit

```
char tab[80];
fgets(tab, 80, stdin);
 //maxim 79 de caractere inclusiv '\0'
```

Parametrul al treilea: un **fișier**; pentru a citi de la **intrarea standard**, folosim identificatorul **stdin** (din stdio.h)

Citirea liniilor cu **fgets**

Atenție:

fgets returnează NULL dacă n-a citit nimic!
(a ajuns la sfârșit de fișier)

la succes: returnează chiar adresa primită parametru
(deci o adresă validă, nenulă).

Testăm rezultatul returnat! (e nul? nenul?)

```
char s[81];

//cat timp nu ajunge la sfarsitul intrarii
while (fgets(s, 81, stdin))
    printf("%s", s);
    //liniile mai lungi de 80 caract. se citesc
    //si afiseaza pe bucati
```

Atenție!

NU folosiți funcția gets pentru citirea de șiruri!

De ce?

Nu se poate specifica lungimea maxim disponibilă

$\Rightarrow$ se poate depăși zona de memorie alocată!

$\Rightarrow$ program abandonat, corupere de memorie, vulnerabilități de securitate

Din aceleași motive, nu folosiți nici **scanf("%s", s)** fără a limita numărul maxim de caractere ce pot fi citite!

La **scanf**, **lungimea** dupa % e **obligatorie** la citirea de șiruri!

scanf – particularități

- formatele **numerice** și **%s** sar peste spațiile albe inițiale:
  "%d%f" – întreg și real, cu oricâte spații albe înainte/între ele
- formatele **%c**, **%[ ]** și **%[^]** nu ignoră spațiile albe
- un număr între % și caracterul de format limitează numărul de caractere citite:
  "%4d" – întreg din cel mult 4 caractere
  (spațiile inițiale nu contează)

scanf – particularități

- un spațiu alb în format consumă oricâte spații albe consecutive din intrare:
  - scanf(“ ”); – consumă toate spațiile albe până la primul caracter diferit
  - “%c %c” – citește un caracter oarecare, consumă spațiile albe, citește următorul caracter (diferit de spațiu alb)
  - “%d %f” – același efect ca “%d%f” (spațiile sunt permise oricum)

Scriere formatată cu **printf**

int printf(const char* format, ...);

În format putem avea aceiași specificatori ca la **scanf**.

Restul parametrilor – **valorile** care se tipăresc (orice expresii)

Atenție!

valorile tipărite trebuie să fie de
același tip cu tipul specificat în format!

verificarea potrivirii tipurilor este sarcina programatorului!
nu rezultă în eroare de compilare!

printf returnează:
numărul de caractere tipărite

Specificatori pentru **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' la sfârșit.

Specificatori pentru **scanf**

%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 atribuite

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

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

%%: caracterul procent

Specificatori pentru **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 la început,
cu a-f pt. %x, A-F pt. %X

%c: caracter

%s: șir de caractere, până la '\0', sau numărul de caractere dat ca precizie

Specificatori pentru **printf**

%e, **%E**: real cu exponent; precizie implicită 6

%f, **%F**: real fără exponent; precizie implicită 6

%g, **%G**: real, ca %e, %E dacă exp. $<$ -4 sau $>$ precizia, altfel ca %f, %F

%a, **%A**: real hexazecimal

%p: pointer, în format dependent de implementare (tipic: hexazecimal)

%n: scrie în argument (int *) nr. de caractere scrise până în prezent;

%%: caracterul procent

Formatare: fanioane

Directivele de formatare pot avea opțional și alte componente:

%fanion dimensiune . precizie modificator tip

Fanioane:

- ***** (scanf): câmpul se citește, dar se ignoră
- **−** (printf): aliniere la stânga
- **+** (printf): + la număr pozitiv (tip cu semn)
- **spațiu** (printf): spațiu înainte de nr. pozitiv (tip cu semn)
- **0** (printf): completează cu 0 până la dimensiunea dată

Formatare: modificatori

%fanion dimensiune . precizie modificator tip

Modificatori:

- **hh**: la formate întregi ⇒ argumentul e pe un octet (deci **char**, dar interpretat numeric)
- **h**: la formate întregi, argumentul este **short**, ex: "%hd"
- **l**: la formate întregi ⇒ argumentul este **long**, ex. "%ld", la formate reale ⇒ argumentul este **double**, ex. "%lf"
- **ll**: la formate întregi ⇒ argumentul este **long long**
- **L**: la formate reale ⇒ argumentul este **long double**

Formatare: dimensiune

%fanion dimensiune . precizie modificator tip

Dimensiunea:

- la **scanf**: numărul maxim de caractere citite pentru argumentul respectiv
- la **printf**: numărul minim de caractere pe care se scrie argumentul

Formatare: precizie

%fanion dimensiune . precizie modificator tip

Precizia – doar la **printf**: punct . urmat de un întreg opțional (fără întreg ⇒ precizie 0):

- pt. formate întregi: numărul minim de cifre (completate cu 0)
- pt. formate reale: numărul de cifre zecimale

  printf(":%7.2f!", 15.234);

  ⇒ : 15.23! (2 zecimale, 7 caract. în total)
- pt. **%s**: numărul maxim de caractere de tipărit din șir

  char m[9]="ianuarie"; printf("%.3s", m);

  ⇒ ian

Obs.: la **printf**, în locul dimensiunii și/sau preciziei poate apărea *, caz în care se ia din argumentul următor:

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