Programiranje 1

Ulaz i izlaz podataka — znakovi, stringovi, formatirani ulaz/izlaz

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

12. siječnja 2023.

Funkcije za ulaz/izlaz

U standardnoj ulazno-izlaznoj biblioteci (`<stdio.h>`) postoje sljedeće funkcije za **ulaz** (čitanje), odnosno **izlaz** (pisanje) podataka:

- `getchar`, `putchar` - za ulaz/izlaz **znakova**,
- `gets`, `puts` - za ulaz/izlaz **stringova**,
- `scanf`, `printf` - za **formatirani** ulaz/izlaz.

Funkcije za **ulaz** rade na standardnoj ulaznoj datoteci `stdin`, a funkcije za **izlaz** na standardnoj izlaznoj datoteci `stdout`. Zato se datoteka **ne zadaje** kao argument funkcije.

Funkcije getchar i putchar

Deklaracija (zaglavlje, prototip) ovih funkcija ima oblik:

```
int getchar(void);
int putchar(int c);
```

Funkcija getchar čita **jedan znak** sa standardnog **ulaza** (obično s tipkovnice) i **vraća** učitani znak. Učitava se **prvi** znak iz ulaznog spremnika koji **do tada** još **nije** bio pročitan. Ukoliko takav **ne postoji**, **čeka se** na ulaz.

Funkcija **nema** argumenata pa se poziva:

```
c_var = getchar();//učitani znak spremamo u c_var
```

Funkcija putchar piše **jedan znak** na standardni **izlaz** (obično na ekran). Ona uzima **jedan** argument (**znak** koji treba ispisati) i **vraća** cjelobrojnu vrijednost.

Funkcija se najčešće poziva uz ignoriranje povratne vrijednosti:

```
putchar(c);
```

Funkcije getchar i putchar

Zašto u deklaracijama imamo `int` umjesto `char`?
Kako prepoznati i **označiti kraj** podataka na ulazu, odnosno **grešku** prilikom izvođenja operacije?

Kada funkcija `getchar` naiđe na **kraj** ulaznih podataka, vraća vrijednost `EOF` (**End of File**). `EOF` je **simbolička** konstanta definirana u `<stdio.h>` koja označava **kraj** datoteke, odnosno kraj ulaznih podataka.

Konstanta `EOF` mora se **razlikovati** od **svih** znakova iz sustava koje računalo koristi. Zato `getchar` **ne** vraća vrijednost tipa `char` već vrijednost tipa `int`. `int` ima dovoljan raspon za kodiranje svih znakova i konstante `EOF` (obično se kodira s -1).

`putchar` uzima vrijednost tipa `int` i **vraća** vrijednost tipa `int`. **Vraćena** vrijednost je **znak** koji je ispisan, ili `EOF`, ako ispis znaka **nije uspio** (`EOF` - signal za grešku).

Funkcije `getchar` i `putchar`

Primjer: program koji kopira **znak-po-znak** s ulaza na izlaz i sva slova **pretvara** u velika.

Kod implementacije odgovarajućeg programa ćemo koristiti funkciju `int toupper(int c)` iz zaglavlja `<ctype.h>` koja pretvara **mala slova** u **velika** a sve druge znakove ostavlja na miru.

Napomena: kod **interaktivnog** ulaza s tipkovnice, **kraj ulaza** zadajemo tako da na tipkovnici prvo stisnemo i držimo tipku `Control` te dok je stisnuta još stisnemo i `Z`.

Funkcije getchar i putchar

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  int main(void) {
5  int c;
6
7  while ((c = getchar()) != EOF)
8       putchar(toupper(c));
9
10 return 0;
11 }
```

Što se događa ako umjesto `putchar(toupper(c));` imamo samo `putchar(c);`?

Funkcije getchar i putchar

Primjer: Funkcija čita **znakove** sa standardnog ulaza, **sve dok** ne naiđe na **prijelaz u novu liniju**, i **ispisuje** učitane znakove **obrnutim** redosljedom (**naopako**).

```
1  void naopako(void) {
2      int znak;
3
4      if ((znak = getchar()) != '\n') naopako();
5      putchar(znak);
6
7  return;
8  }
```

U svakom pozivu funkcije `naopako`, jedan znak u liniji se zapamti u lokalnu varijablu `znak`. Taj znak se ispiše na standardni izlaz tek nakon što se prekine rekurzivni poziv funkcije (mora biti `getchar() == '\n'`). Znakovi se ispisuju u obrnutom redosljedu.

Funkcije getchar i putchar

Glavni program (funkcija main).

```
1  int main(void) {
2
3     printf("␣Unesite␣niz␣znakova:␣");
4     naopako();
5
6  return 0;
7  }
```

Izvršavanje za **ulaz** Zdravo i tipku Enter daje **rezultat:**

```
Unesite niz znakova: Zdravo

ovardZ
```

Prvo se ispisuje **zadnji** učitani znak \n (nastao nakon pritiska tipke ENTER).

Osnovno o stringovima

String je **niz znakova** koji završava **nul-znakom** (`'\0'`).

```
1 char niz_znakova[5] = {'h', 'e', 'l', 'l', 'o'};
2 char string[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
3 char string[] = "hello";
```

U gornjem primjeru deklaracije u 2. i 3. retku su **ekvivalentne**. Deklaracija u prvom retku deklarira **običan niz znakova a ne string** (niz ne sadrži nul-znak).

Napomena: **ime polja** je **pokazivač** na **prvi** element polja (`string = &string[0]`).

Funkcije gets i puts

Deklaracija ovih funkcija ima oblik:

```
1 char *gets(char *s);
2 int puts(const char *s);
```

Funkcije gets i puts služe čitanju i pisanju **znakovnih nizova** (**stringova**).

Funkcija gets čita **znakovni niz** sa standardnog ulaza. Znak za kraj linije '\n' zamjenjuje s nul-znakom '\0'. Funkcija vraća **pokazivač** na char koji pokazuje na **učitani** znakovni niz ili je jednak NULL ukoliko je došlo do **greške** prilikom čitanja ili smo pri učitavanju došli do **kraja** ulaznih podataka.

Funkcija puts uzima kao argument **znakovni niz** i piše ga na standardni izlaz. Kod ispisa puts zamjenjuje nul-znak '\0' (s kraja stringa) znakom za prijelaz u novi red (kraj retka) '\n'.

Funkcije gets i puts

Funkcija **vraća** cijeli broj: a) **nenegativan** broj ako je ispis **uspio**, b) `EOF` ako **nije**.

Simbolička konstanta `NULL` definirana je u zaglavlju `<stdio.h>`.

Program koji kopira liniju po liniju ulaza na izlaz.

```
1  #include <stdio.h>
2
3  int main(void) {
4      char red[128];
5
6      while (gets(red) != NULL)
7          puts(red);
8
9  return 0;
10 }
```

Problem: što ako ulazna linija (s `'\n'`) ima $>$ 128 znakova?

Opasnost kod funkcije `gets`

Osnovni **nedostatak** funkcije `gets` je u tome što kod poziva **nije moguće** zadati **maksimalni** broj znakova koji će biti učitan. Ako je broj znakova na ulazu **veći** od duljine polja koje je argument funkcije `gets`, doći će do **prepunjenja** polja koje služi za spremanje **stringa**.
To prepunjenje se zove `buffer overflow` i često se koristi za računalne **viruse** i slične zlonamjerne svrhe.
Kod prepunjenja polja namijenjenog spremanju stringa i pristupa memoriji koja ne pripada tom polju, ne dolazi do **greške** dok ne prijeđemo granicu rezervirane memorije za **sve** podatke u programu.

Preporučljivo umjesto funkcije `gets` koristiti `fgets`. Standard `C11` izbacuje `gets` iz skupa standardnih funkcija.

Funkcija gets_s (C11)

Deklaracija ove funkcije ima oblik:

```
1 char *gets_s(char *s, rsize_t n);
```

Dodatni argument `n` (obzirom na `gets`) je **maksimalni** broj znakova u polju `s` koji se `smije` iskoristiti za spremanje učitanog stringa (**uključivo** nul-znak `'\0'`).
Funkcija učitava **najviše** $n - 1$ znak s ulaza u string `s`. Na ulazu mora biti $s \neq NULL$ i $n > 0$ (uvjeti izvršavanja).

Ako je linija **predugačka preskače** višak znakova (do kraja linije ili kraja datoteke). Preskakanje viška znakova se smatra **greškom** (prekršaj uvjeta izvršavanja). U slučaju greške se postavlja `s[0] = '\0'` i vraća `NULL`.

Funkcija gets_s (C11)

Program koji kopira liniju po liniju ulaza na izlaz.

```
1  #include <stdio.h>
2
3  int main(void) {
4      char red[128];
5
6      while (gets_s(red, sizeof(red)) != NULL)
7            puts(red);
8
9  return 0;
10 }
```

gets_s **provjerava** da svaka linija ima najviše 128 znakova, zajedno s krajem linije, inače vraća NULL.

Funkcija scanf

Funkcija scanf služi za **formatirano učitavanje** podataka sa standardnog ulaza (stdin). Opći oblik poziva funkcije je:

```
scanf(kontrolni_string, arg_1,arg_2, ..., arg_n)
```

Argument kontrolni_string je **konstantni** (funkcija ga ne mijenja) niz znakova (string) koji sadrži informacije o **tipovima vrijednosti** koje se učitavaju u argumente arg_1,...,arg_n. kontrolni_string može biti i **izraz**.

kontrolni_string sastoji se od znakova i posebnih **grupa znakova - specifikacija** (**oznaka konverzije** ili pretvaranja).

Svaka **oznaka** konverzije **pridružena** je po jednom **sljedećem** argumentu (redom kojim su definirani). **Svaka** oznaka konverzije **započinje** znakom postotka (%), a na **kraju** dolazi **znak konverzije**. On zadaje format zapisa i **tip** podatka koji se učitava (npr. %c, %d).

Funkcija scanf

Najčešće korišteni **znakovi konverzije** su:

znak	format i tip podatka koji se čita
d	decimalni cijeli broj (int)
i	decimalni, heksadecimalni ili oktalni cijeli broj (int)
u	decimalni cijeli broj bez predznaka (unsigned int)
o	oktalni cijeli broj (int)
x	heksadecimalni cijeli broj (int)
e,f,g	decimalni realni broj (float)
c	jedan znak (char)
s	string (char *)
p	pokazivač (void *)

Ispred znaka konverzije može doći **modifikator duljine** tipa: h - skraćuje tip (h = *half*), l ili L - **produljuje** tip (l = *long*).

Funkcija `scanf`

U pozivu funkcije `scanf`, argumenti `arg_1,...,arg_n` **moraju** biti **pokazivači** na varijable odgovarajućeg tipa. Ako podatak treba učitati u neku varijablu, onda `scanf` mora dobiti **adresu** te varijable.

Osnove rada naredbe `scanf`:

- Podaci koje `scanf` stvarno čita su **znakovi** koji dolaze sa standardnog ulaza (obično tipkovnica),
- Ako se unosi **više** podataka, oni **moraju** biti odvojeni **bjelinama** - uključuje i prijelaz u novi red.
- Kod čitanja **numeričkih** podataka, vodeće bjeline se **preskaču**, a podaci na ulazu **moraju** imati isti oblik kao i **numeričke konstante** pripadnog tipa u `C`-u (**bez sufiksa**).

Učitavanje cijelih brojeva

Cijeli brojevi mogu biti učitani kao **decimalni** (`%d`), ili kao **decimalni**, **oktalni** i **heksadecimalni** (`%i`). Znak konverzije `i` interpretira ulazni podatak kao **oktalni** ako mu prethodi **nula**, kao **heksadecimalni** ako mu prethodi `0x` ili `0X`.

```
1 int x, y, z;
2 ...
3 scanf("%i␣%i␣%i", &x, &y, &z);
```

U gornjem primjeru za ulaz: `13 015 0xd` u varijable `x`, `y` i `z` učitavamo istu vrijednost 13 (**decimalno**).

Cijeli brojevi u **oktalnom** i **heksadecimalnom** zapisu mogu se čitati i pomoću znakova konverzije `o`, odnosno `x`. Ti znakovi konverzije **ne zahtijevaju** da oktalna konstanta započinje **nulom**, a heksadecimalna s `0x` ili `0X`.

Učitavanje cijelih brojeva

```
1 int x, y, z;
2 ...
3 scanf("%d␣%o␣%x", &x, &y, &z);
```

U gornjem primjeru za ulaz: `13 15 d` u varijable `x`, `y` i `z` učitavamo istu vrijednost 13 (**decimalno**).

Podatak tipa `unsigned` učitavamo znakom konverzije `u`. Znakovi konverzije `d`, `i`, `o`, `u`, `x` **moraju** dobiti prefiks `h` kada je argument pokazivač na `short`, `l` kada je argument pokazivač na `long`.

```
1 int x;
2 short y;
3 long z;
4 ...
5 scanf("%d␣%hd␣%ld", &x, &y, &z);
```

Gornji primjer učitava tri decimalna cijela broja i sprema ih u varijable tipa `int`, `short` i `long`.

Učitavanje realnih brojeva

Znakovi konverzije `e`, `f` i `g` služe za učitavanje vrijednosti u varijablu tipa `float`. Ako se učitava vrijednost u varijablu tipa `double`, mora se koristiti prefiks `l` (`le`, `lf` ili `lg`).

```
1 float x;
2 double y;
3 ...
4 scanf("%f %lg", &x, &y);
```

Prefiks `L` se koristi za učitavanje realne vrijednosti u varijablu tipa `long double` (ako postoji).

Formatiranje i konverzija ulaza

Niz znakova na ulazu se **interpretira** i **pretvara** u **vrijednosti** na sljedeći način:

- Funkcija `scanf`, redom kako čita, dijeli ulazni niz znakova u **polja znakova** za konverzije u zadane tipove.
- Sljedeće polje počinje **prvim** dotad **nepročitanim** znakom, **nakon** eventualnog **preskakanja** vodećih bjelina — kod numeričkih konverzija i čitanja **stringova** (znak `s`).
- Nakon toga se, ovisno o **znaku za konverziju**, čita (i, po potrebi, konvertira) **najdulji dozvoljeni** niz znakova koji odgovara obliku ulaza za taj **znak konverzije**.

Kod čitanja **brojeva** i **stringova**, ulazno polje može ići **najdalje** do znaka **ispred prve sljedeće bjeline**.

Formatiranje i konverzija ulaza

```
1 scanf("%f%d", &x, &i);
```

Prva oznaka konverzije `%f` učitava i konvertira **prvo** polje znakova. Pritom se eventualne **bjeline** na početku preskaču. **Prvo** polje znakova završava **bjelinom** koju `%f` **ne** učitava.

Druga oznaka konverzije `%d` preskače sve **bjeline** koje odjeljuju **prvo** polje znakova od **drugog** i učitava (i konvertira) **drugo** polje znakova.

Napomena: Ulazni broj tipa `float` **ne smije** imati sufiks `f`. Za ulazni niz znakova `1.0f 2` - **prvo** polje znakova za `x` je **samo** 1.0, a znak `f` ostaje **nepročitan**. **Drugo** polje je **prazno** i **ne dobiva** vrijednost (nema konverzije).

Maksimalna širina ulaznog polja

Uz **svaki znak konverzije** može se zadati i **maksimalna širina** ulaznog polja koje će se učitati. To se radi tako da se **ispred znaka konverzije** i eventualnog **prefiksa** stavi odgovarajući **broj**.

```
%3d - učitava cijeli broj s najviše tri znamenke
%9s - učitava najviše 9 znakova stringa (bitno u praksi)
```

Ako podatak sadrži **manje** ili **jednako dozvoljenih** znakova od **zadane** maksimalne širine polja, čitanje staje **ispred** prvog **nedozvoljenog** znaka (npr. bjelina za brojeve).

Ako podatak ima **više dozvoljenih** znakova od **zadanog**, pripadno **polje znakova** za konverziju **skraćuje** se na **zadani** broj, a **višak** znakova ostaje za **naknadno** čitanje.

Razmaci (praznine) u kontrolnom stringu

Oznake konverzije mogu biti odijeljene **razmacima**:

```
1 scanf("%f␣%d", &x, &i);
```

Taj **razmak** (praznina) ima za posljedicu **preskakanje** svih **bjelina** na ulazu do **početka** novog ulaznog polja.

Kod čitanja vrijednosti **brojevnih** tipova i **stringova**, eventualne **bjeline** ispred polja se automatski **preskaču**.

U kontrolnom stringu, **pripadne** oznake konverzije možemo pisati **razdvojeno razmacima** (kao u primjeru `"%f %d"`), ili **nerazdvojeno** (kao `"%f%d"`). Oba zapisa rade **jednako**.

To ne vrijedi za znakove konverzije `c` i `[`. Tamo razmak **ispred** oznake **ima** značenje.

Drugi znakovi u kontrolnom stringu

U kontrolnom stringu se osim razmaka i oznaka konverzije mogu pojaviti i **drugi znakovi**. Njima **moraju** odgovarati **identični znakovi na ulazu** (vrši se **sparivanje**).

Ako realni broj `x` i cijeli broj `i` učitavamo naredbom:

```
scanf("%f,%d", &x, &i);
```

ispravan primjer ulaza je: `3.141, 18` (**bez bjeline** između **prvog broja** i **zareza**).

Sljedeća oznaka konverzije `%d` čita znakove **iza** zareza i preskače sve **bjeline** na ulazu ispred odgovarajućeg polja (drugog broja).

Formatiranje i konverzija ulaza

Ako se želi **dozvoliti** bjelina **prije** zareza, potrebno je koristiti naredbu:

```
scanf("%f␣,%d", &x, &i);
```

Razmak nakon oznake `%f` **preskače** sve eventualne bjeline na ulazu **ispred** zareza.

Za **razmak** u kontrolnom stringu također vrijedi **sparivanje**. Za razliku od ostalih znakova, s **razmakom** se na **ulazu** sparuje bilo koji uzastopni **niz bjelina** (6 dozvoljenih znakova), taj niz smije biti i **prazan**.

Učitavanje znakovnih nizova - %s

Znak konverzije s učitava **niz znakova** (string). Vodeće **bjeline** na ulazu se **preskaču**. Niz završava (najdalje) ispred **prve sljedeće bjeline** u ulaznom nizu znakova. **Iza posljednjeg** učitanog znaka, u string se automatski dodaje nul-znak (`'\0'`).

```
1 char string[128];
2 int x;
3 ...
4 scanf("%s %d", string, &x);
```

U gornjem primjeru se učitava jedna **riječ** (niz znakova) bez bjelina i jedan broj.

Ime **polja** je **sinonim** za **pokazivač** na **prvi** element polja. Zato se ispred varijable `string` **ne stavlja** adresni operator.

Učitavanje znakovnih nizova - `%[...]`

Oznakom konverzije `%s` **nije moguće** učitati niz znakova koji **sadrži bjeline** jer bjeline označavaju kraj polja.

Nizove znakova koji uključuju **bjeline** učitavamo koristeći **uglate zagrade** kao znak konverzije - oznaka `%[...]`.

- **Unutar** uglatih zagrada upisuje se niz znakova.
- Funkcija `scanf` će u pripadni argument učitati **najdulji** niz znakova s ulaza koji se sastoji **samo** od znakova **navedenih unutar** uglatih zagrada.
- Učitavanje završava **ispred prvog znaka** na ulazu koji **nije** naveden u uglatim zagradama. Na kraj učitanog niza dodaje se nul-znak (`\0`).
- Vodeće **bjeline** se **ne preskaču**.

Učitavanje znakovnih nizova - %[...]

```
1 char linija[128];
2 ...
3 scanf(" %[ ABCDEFGHIJKLMNOPRSTUVWXYZ]", linija);
```

Gornji dio programa učitava **najdulji** niz znakova sastavljen **samo** od velikih slova i razmaka. Razmak naveden **unutar** %[...] označava ' ', a **ne** i ostale **bjeline**.

Prije %[ostavljen je jedan **razmak** koji govori funkciji scanf da preskoči sve **bjeline ispred** znakovnog niza. To je **nužno** ako smo prije imali poziv funkcije scanf koji **nije učitao bjelinu** kojom završava prethodno polje u ulaznom nizu (npr. završni znak za prijelaz u novi red).

Učitavanje znakovnih nizova - %[...]

```
scanf("%[ ABCDEFGHIJKLMNOPRSTUVWXYZ]", linija);
```

Ukoliko bi eliminirali početni razmak iz `scanf`, čitanje bi započelo na znaku za prijelaz u novi red (`\n`). Budući da on **nije** naveden **unutar** uglatih zagrada, odmah bi **završila** čitanje ulaznih podataka. Broj učitanih podataka bio bi **nula**.

Posljedica: željena **linija ne bi** bila učitana, dobili bismo prazan string - sadrži samo `\0`.

Napomena: unutar uglatih zagrada `%[...]` smijemo koristiti **raspone** znakova (npr. `A-Z`), slično kao kod regularnih izraza (`regex`).

Učitavanje znakovnih nizova - %[^...]

S uglatim zagradama smijemo koristiti i sintaksu **negacije** - ^.

```
scanf(" %[^niz znakova]", linija);
```

U tom slučaju se u odgovarajući argument učitava **najdulji** mogući niz znakova sastavljen od **bilo kojih** znakova — **osim** onih koji se nalaze u uglatim zagradama **iza** znaka ^.

Primjer: cijelu liniju **bez** znaka za prijelaz u novi red (\n) možemo učitati naredbom:

```
scanf(" %[^\n]", linija);
```

Na kraj učitanog niza znakova bit će dodan \0 (kraj stringa). Ispred %[namjerno je ostavljen **razmak**, zato da se **preskoče** sve prethodne **bjeline**.

Opasnost kod čitanja znakovnih nizova

Kod formatiranog čitanja **stringova** postoji ista **opasnost** kao i kod funkcije `gets`. Ako **ne navedemo maksimalnu** širinu polja, može doći do **prepunjenja** stringa. Zato uvijek treba navesti maksimalnu širinu polja tako da u string stanu **svi** učitani znakovi i znak za kraj stringa \0.

```
1 char str_1[16], str_2[33], str_3[80];
2 ...
3 scanf("%15s", str_1);
4 scanf("%32[A-Z]", str_2);
5 scanf("%79[^\n]", str_3);
```

Učitavanje pojedinačnih znakova - %c

Znak konverzije c učitava **jedan** znak u varijablu, **bez obzira** je li on **bjelina** ili ne (**nema preskakanja bjelina**).

- Ako želimo **preskakanje bjelina** — primjerice, zato da preskočimo znak za prijelaz u novi red koji je ostao nakon prethodnog poziva funkcije `scanf`, treba staviti jedan **razmak** ispred oznake konverzije %c.
- Kontrolni niz `" %c%c%c"` učitava vrijednost tri znaka. Prvo **preskače** sve **bjeline** (zbog razmaka ispred prve oznake %c), a zatim čita **tri uzastopna** znaka. Zadnja dva znaka **mogu** biti **bjeline**.
- Ukoliko želimo čitati samo znakove **različite** od **bjelina**, treba koristiti `" %c %c %c"`.

Učitavanje polja znakova

Znaku konverzije c možemo zadati **maksimalnu** širinu polja. Tada se čita **uzastopni blok** od **točno** toliko znakova kolika je **zadana maksimalna** širina polja. Pripadni argument je **pokazivač** na **polje**, koje mora imati mjesta za bar **toliko** elemenata (nema kontrole).

Za razliku od %s i %[, na kraj učitanog niza znakova se **ne** dodaje nul-znak (**nije** string).

```
1 char kratica[4], drzava[3];
2 scanf("%4c", kratica);
3 scanf("%3c", &drzava[0]);
```

Prefiks *

Neki podatak u ulaznom nizu moguće je **preskočiti** i **ne pridružiti** ga odgovarajućoj varijabli. To radimo tako da znaku konverzije dodamo **prefiks * odmah** iza znaka %.

```
1  scanf(" %s %*d %d", ime, &n);
```

U gornjem primjeru, `scanf` korektno čita **drugi** podatak prema formatu %d. Zbog prefiksa *, **neće** se izvršiti pridruživanje te vrijednosti varijabli `n`. Taj podatak se **preskače** (zanemaruje).

Treći podatak bit će normalno pridružen varijabli `n`.

Svrha: preskakanje stupaca u tablicama.

Primjer - scanf

```
1 double x; char c; int i;
2 scanf("%lg%c%d", &x, &c, &i);
3 printf("x␣=␣%g,␣c␣=␣'%c',␣i␣=␣%d\n", x, c, i);
```

Gornji primjer **čita** i **ispisuje** podatke različitog tipa.

Za ulaz: `17.19x17` dobivamo izlaz:
`x = 17.19, c = 'x', i = 17.`

```
1 int i; float x; char s[50];
2 scanf("%2d%f%*d␣%[0-9]", &i, &x, s);
3 printf("i␣=␣%d,␣x␣=␣%f,␣s␣=␣%s\n", i, x, s);
```

U gornjem primjeru za ulaz: `56789 0123 56a72` dobivamo izlaz:
`i = 56, x = 789.000000, s = 56.`

Povratna vrijednost funkcije scanf

Funkcija `scanf` vraća (nenegativan) **broj uspješno učitanih podataka**, ili `EOF` ako je došlo do **greške** ili kraja datoteke prije čitanja **prvog** podatka.

```
int n;
...
while (scanf("%d",&n) == 1 && n >= 0) {
... /* Radi nesto s brojem n. */
}
```

`while` petlja se **prekida** ako čitanje broja **nije uspjelo** ili ako je učitan **negativan** broj (koristimo **skraćeno** izračunavanje).

Funkcija printf

Funkcija `printf` služi za **formatirani ispis** podataka na standardni izlaz (`stdout`). Opći oblik poziva funkcije je:

```
1  printf(kontrolni_string, arg_1, ..., arg_n)
```

Prvi argument `kontrolni_string` je **konstantni** (funkcija ga ne smije mijenjati) znakovni niz (string) koji sadrži informaciju o **formatiranju ispisa vrijednosti** argumenata `arg_1,...,arg_n`.

`kontrolni_string` smije biti **izraz**, ima **identičan oblik** i vrlo **sličnu funkciju** kao kod funkcije `scanf`. Ostali argumenti `arg_1,...,arg_n` su **izrazi**.

Kontrolni string sadrži **dvije** vrste objekata:

- **obične** znakove, **doslovno** se **prepisuju** u izlaznu datoteku,
- **specifikacije** (**oznake**) **konverzije** - **grupe** znakova koje **počinju** znakom %, a **završavaju znakom konverzije** (**između može** biti još znakova s **posebnim** značenjem).

Funkcija `printf`

Specifikacije konverzije vrše **pretvaranje** i **ispis sljedećeg** po redu (neispisanog) argumenta u tom pozivu `printf`. **Prvo** se izračuna **vrijednost** tog argumenta, a zatim se po pravilima konverzije ta vrijednost **pretvara** u **niz znakova** koji se zatim **ispisuje**.

Najčešće korišteni **znakovi konverzije** su:

znak	**format i tip podatka koji se čita**
`d`, `i`	decimalni cijeli broj (`int`)
`u`	decimalni cijeli broj bez predznaka (`unsigned int`)
`o`	oktalni cijeli broj (`int`)
`x`, `X`	heksadecimalni cijeli broj (`int`)
`e`,`f`,`g`	decimalni realni broj (`double`)
`c`	jedan znak (`char`)
`s`	string (`char *`)
`p`	pokazivač (`void *`)
`%`	nema konverzije, ispiše znak %

Funkcija `printf`

Znak % **ispisujemo** navođenjem %% unutar odgovarajućeg kontrolnog znakovnog niza.

Funkcija `printf` vraća **nenegativan broj ispisanih znakova** ili `EOF` ako je došlo do **greške**.

Funkcija `printf` koristi **prvi** argument (*format-string*) za određivanje **broja i tipova argumenata** koji slijede.

Ako je **argumenata premalo** ili su **pogrešnog** tipa, dobit ćete **pogrešne** rezultate. Ukoliko funkciji predamo **previše** argumenata s obzirom na specifikaciju u format-stringu, možemo dobiti **upozorenje**.

printf - primjer

```
int n = 13;
printf("%%10d\n", n);
```

Gornji primjer ispisuje **izlaz** od jednog retka teksta:
%10d

Do toga dolazi zato što %% **doslovno** ispisuje znak %. Stoga se **cijeli** format-string **doslovno** ispisuje. Uz to dobijemo upozorenje da format-string **završava prije** argumenta `n`.

Argumenti funkcije `printf` (iza format-stringa) su **izrazi** (mogu biti konstante, varijable ili složeniji izrazi s operatorima).

```
double x = 2.0;
printf("x=%f, y=%f\n", x, sqrt(x));
```

Svi znakovi koji nisu dio specifikacije konverzije ispisani su točno onako kako su napisani u format–stringu: `"x=%f, y=%f\n"`.
`x=2.000000, y=1.414214`

Konverzije tipova kod `printf`

Pri pozivu funkcije `printf` **može** doći do **konverzije** tipova:

- Argumenti tipa `float` uvijek se pretvaraju u `double`.
- Argumenti tipa `char` i `short` **uvijek** se pretvaraju u `int` a **mogu** se **ispisati** kao `char` ili `short` (odgovarajućom oznakom konverzije).

Zbog toga znak konverzije:

- `%f` - ispisuje vrijednosti tipa `float` i `double`,
- `%d` - **može** ispisati vrijednosti tipa `int`, `char` i `short`.

Slično vrijedi i za preostale **realne** odnosno **cjelobrojne** znakove konverzije (`e`, `g` odnosno `i`, `u`, `o`, `x`).

Znakovi konverzije se moraju koristiti oprezno i odgovarati tipu podataka, inače **može** doći do grešaka.

Ispis znaka

Jedan **znak** možemo ispisati na **dva** načina:

- kao **običan** znak %c (znak se reprezentira kao `int`, ispisuje kao znak),
- kao **cijeli broj** %d (znak se reprezentira kao `int`, ispisuje kao cijeli broj).

```
1 char c = '1';
2 printf("c(char) = %c, c(int) = %d\n", c, c);
```

Gornji primjer ispisuje: `c(char) = 1, c(int) = 49`.
Ukoliko računalo koristi `ASCII` skup znakova, broj 49 je `ASCII` kod znaka `'1'`.

Oktalni i heksadecimalni ispis

Pomoću znakova konverzije `%o` i `%x` (`%X`) ispisuju se **cijeli** brojevi u **oktalnom** i **heksadecimalnom** obliku **bez** predznaka i **bez** vodeće **nule** odnosno `0x` (ili `0X`).

Ako želimo da za broj **različit** od **nule oktalni** ispis **ima** vodeći znak 0, odnosno **heksadecimalni** ispis **ima** vodeće znakove `0x` (ili `0X`), moramo koristiti **alternativnu** formu ispisa. Dobiva se **zastavicom** #, koja se piše odmah iza znaka %.

```
1 int i = 64;
2 printf("i(dec) = %d = %i\n", i, i);
3 printf("i(oct) = %o = %#o\n", i, i);
4 printf("i(hex) = %x = %#x\n", i, i);
```

Gornji primjer ispisuje:

```
i(dec) = 64 = 64
i(oct) = 100 = 0100
i(hex) = 40 = 0x40
```

Oktalni i heksadecimalni ispis

Ukoliko isti primjer (bez `%i` i #) pokrenemo za $i = -3$ dobijemo ispis:

```
i(dec) = -3
i(oct) = 37777777775
i(hex) = fffffffd
```

Sadržaj varijable `i` se konvertira u oktalni, odnosno heksadecimalni zapis **bez** predznaka. Ispis je kao da interpretiramo odgovarajuću memorijsku lokaciju **po bitovima** (**binarno**), odnosno kao cijeli broj **bez** predznaka.

Modifikatori tipa za `short` i `long`

Promjena **duljine** osnovnog **tipa** zadaje se **modifikatorom tipa** u odgovarajućoj **oznaci** konverzije. Modifikator se piše **ispred** znaka konverzije (kao **prefiks**).

Za **cjelobrojne** tipove, modifikatori tipa su:

- `h` - označava da argument treba **ispisati** kao `short` ili `unsigned short`. Pri ulazu u `printf`, argument je već **pretvoren** u `int` ili `unsigned int`.
- `l` - označava da je argument tipa `long` ili `unsigned long`. Ovdje **nema** pretvaranja tipova (modifikator treba koristiti za ispis varijabli koje su zaista tipa `long int`, inače ispis neće biti dobar osim ako `int` i `long int` nisu isti u korištenom sustavu).

Na nekim sustavima postoji i `ll` za `long long` (ukoliko podržava taj tip).

Ispis brojeva skraćenjem tipa na short

```
1 short i = -3;
2 printf("i(dec) = %d\n", i);
3 printf("i(oct) = %o\n", i);
4 printf("i(hex) = %x\n", i);
5 printf("h(dec) = %hd\n", i);
6 printf("h(oct) = %ho\n", i);
7 printf("h(hex) = %hx\n", i);
```

Gornji primjer ispisuje isti broj `i` (koji se uvijek pretvara u `int`) kao `int` u prva tri poziva i **skraćenjem** na `short` (modifikator `h`) u zadnja tri poziva.

Ispis brojeva skraćenjem tipa na `short`

Odgovarajući ispis je:

```
i(dec) = -3
i(oct) = 37777777775
i(hex) = fffffffd
h(dec) = -3
h(oct) = 177775
h(hex) = fffd
```

U zadnja **dva** reda se **vidi** da su ispisana samo **donja** 2 byte-a (skraćenje na `short`), od vrijednosti tipa `int` koju dobije `printf`.

Za `short i=3` **ne** vidimo razliku (svi ispisi ispisuju 3) pošto se **vodeće nule** ne ispisuju.

Ispis brojeva tipa long

Izrazi tipa long ispisuju se pomoću **prefiksa** l.

```
1  #include <stdio.h>
2  #include <limits.h>
3
4  int main(void) {
5        long i = LONG_MAX;
6
7        printf("i(dec) = %ld\n", i);
8        printf("i(oct) = %lo\n", i);
9        printf("i(hex) = %lx\n", i);
10
11 return 0; }
```

Ispis ovisi o računalu na kojem se program izvršava.

Ispis brojeva tipa long

Na IA-32, s Intelovim C prevodiocem, rezultat je:

```
i(dec) = 2147483647
i(oct) = 17777777777
i(hex) = 7fffffff
```

Ovdje je int jednak long.

Simbolička konstanta LONG_MAX definirana je u datoteci zaglavlja <limits.h> i predstavlja najveći broj tipa long.

Ispis realnih brojeva

Brojeve tipa `float` i `double` možemo ispisati pomoću znakova konverzije `%f`, `%g` i `%e`.

Korištenjem znaka konverzije `%f` broj se ispisuje **bez** eksponenta, dok korištenjem `%e` ili `%E` se broj ispisuje **s** eksponentom (eksponent se označava slovom e ili E). `%g`, `%G` omogućava ispis s eksponentom ili bez njega **ovisno o vrijednosti** koja se ispisuje. Ako je eksponent **manji** od -4 ili dovoljno **velik** ($\geq$ od **preciznosti**), koristi se `%e`. U **protivnom** se koristi `%f`. Završne **nule iza** decimalne točke se **ne** ispisuju.

Za ispis brojeva tipa `long double` (ako tip postoji) koristimo **prefiks** (modifikator duljine) `L`. Pripadni formati konverzije su `%Le`, `%Lf`, `%Lg`.

Ispis realnih brojeva

```
1 double x = 12345.678;
2 printf("x(f) = %f\n", x);
3 printf("x(e) = %e\n", x);
4 printf("x(g) = %g\n", x);
```

Gornji primjer ispisuje:

```
x(f) = 12345.678000
x(e) = 1.234568e+04
x(g) = 12345.7
```

Kod %f i %e imamo ispis 6 **decimala**, a kod %g ispis 6 **vodećih** znamenki.

Minimalna širina ispisa

Uz **svaki** znak konverzije moguće je zadati **minimalnu širinu** ispisa (**minimalni** broj znakova u ispisu). To se radi **navođenjem broja ispred** znaka konverzije.

%3d - ispisuje cijeli broj s **najmanje** 3 znaka dok %9s ispisuje **najmanje** 9 znakova stringa.

Ako podatak treba **manje** znakova od zadane **minimalne** širine polja, bit će **slijeva dopunjen razmacima** do **zadane** širine (osim ako nije zadano drugačije dopunjavanje - *zastavicama*). Ako podatak treba **više** znakova od **minimalne** širine ispisa, bit će ispisan **sa svim potrebnim** znakovima.

```
1 int n = 54321;
2 printf("%10d,%5o,%5x\n", n, n, n);
```

Gornji primjer ispisuje:

```
     54321,152061, d431
```

Minimalna širina ispisa

Oktalni ispis ima svih 6 **potrebnih** znakova (**minimalna** širina pripadnog polja je 5).

Heksadecimalni ispis je dopunjen **jednim** razmakom.

```
1 double x = 1.2;
2 printf("%1g\n%3g\n%5g\n", x, x, x);
```

Gornji primjer ispisuje:

```
1.2
1.2
  1.2
```

Prva dva ispisa imaju točno 3 znaka u svom redu, dok **treći** ima točno **pet** znakova (ima **dva** vodeća razmaka).

Preciznost ispisa realnih brojeva

Pored minimalne širine, moguće je zadati i **preciznost** ispisa. Kod realnih brojeva je **preciznost najveći** broj **decimala** (za `%f` i `%e`) odnosno **vodećih** značajnih znamenki (za `%g`) koje će biti ispisane.

Preciznost se definira tako da se nakon minimalne širine ispisa (ukoliko je definirana) zapiše **točka** pa **broj**. `%a.bf`, `%a.be` ili `%a.bg` (`a` - minimalna širina ispisa, `b` - preciznost).

Primjer: `%7.3e` - označava ispis formatom `e` s **najmanje** 7 znakova, pri čemu su **najviše** 3 znamenke iza decimalne točke.

Ukoliko ne specificiramo **preciznost** ona je standardno jednaka 6.

Preciznost ispisa realnih brojeva

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(void) {
5      double pi = 4.0 * atan(1.0);
6      printf("%5f, %10.5f, %5.10f, %5.4f\n",
7              pi, pi, pi, pi);
8
9  return 0;
10 }
```

Rezultat ispisa gornjeg programa uz **zaokruživanje** na **zadani** broj decimala:

```
3.141593,    3.14159, 3.1415926536, 3.1416
```

Dinamičko zadavanje širine i preciznosti

Širinu i **preciznost** ispisa moguće je zadati i **dinamički**, u trenutku **izvođenja** programa tako da se **iznos širine** ili **preciznosti zamijeni** znakom *. Na **pripadnom** mjestu u listi argumenata koje **odgovara** znaku * mora biti **cjelobrojni izraz** (obično varijabla). **Trenutna vrijednost** tog argumenta određuje **širinu** odnosno **preciznost** (**uvrštava** se umjesto znaka *).

Vrijednost argumenta koji odgovara * se **ne ispisuje** već se ta vrijednost supstituira i obrađuje se **sljedeći** argument.

Dinamičko zadavanje širine i preciznosti

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(void) {
5       double pi = 4.0 * atan(1.0); int i = 10;
6       printf("%*f, %*.*f, %5.*f\n",
7              11, pi, 16, 14, pi, i, pi);
8  return 0;
9  }
```

Umjesto * se zapisuju `11, 16, 14` i `i = 10`, stoga je **ispis**:

```
3.141593, 3.14159265358979, 3.1415926536
```

Ispis znakovnih nizova

Znak konverzije %s služi za ispis **znakovnih nizova** (stringova). Ispisuje **sve** znakove u stringu, dok ne dode do nul-znaka \0, kojeg **ne ispisuje**.

```
1 char naslov[] = "Programski␣jezik␣C";
2 printf("%s\n", naslov);
```

Gornji primjer ispisuje:

`Programski jezik C`

Nakon toga dolazi do prijelaza u novi redak zbog `\n` nakon `%s`.

Minimalna širina polja i **preciznost** mogu se koristiti i kod `%s` konverzije. U tom slučaju, **preciznost** označava **maksimalni** broj znakova koji smije biti ispisan.

Primjer: `%5.12s` - specificira da će biti ispisano **minimalno** 5 znakova (dopunjenih vodećim razmacima do 5), a **maksimalno** 12 znakova (višak neće biti ispisan).

Ispis znakovnih nizova

Minimalna širina ispisa ne mora biti zadana.

```
1 char naslov[] = "Programski␣jezik␣C";
2 printf("%.16s\n", naslov);
```

Gornji primjer ispisuje:

`Programski jezik`

Zadnji znak `k` je **zadnji** znak u tom redu (16. znak iz stringa).

Ispis polja znakova (koje nije string)

Znak konverzije s, uz zadanu **preciznost** = **maksimalna** širina polja, može se koristiti i za ispis **polja** znakova koja **nisu** string (**ne** sadrže nul-znak). U tom slučaju se ispisuje **točno onoliko** znakova kolika je **zadana preciznost** (**ne** smije biti **veća** od broja elemenata u **polju** - nema kontrole).

```
1 char kratica[4] = { 'c', 'e', 'r', 'n' };
2 printf("%.4s, %.2s\n", kratica, &kratica[1]);
```

Gornji primjer ispisuje:

```
cern, er
```

Sažetak o zastavicama

Zastavice služe za **modificiranje** standardnog ponašanja **znakova konverzije**. Pišu se odmah iza znaka %, smije ih biti **više** i mogu biti napisane u **bilo kojem** međusobnom poretku.

- `-` označava **lijevo** pozicioniranje konvertiranog argumenta u polju za ispis.
- `+` označava da će **broj** biti uvijek ispisan s **predznakom**.
- `␣` (razmak ili praznina): ako prvi znak (nakon pretvorbe) **nije** predznak, **dodat** će se **razmak** (praznina, blank) na **početak**.
- `0` kod **numeričkih** konverzija, označava **dopunjenje** polja za ispis (do širine polja) **vodećim nulama**, a ne razmacima.
- `#` označava **alternativnu** formu ispisa za pojedine **znakove konverzije**. Za `o` - **prva** znamenka bit će **nula**, za `x` odnosno `X` **dodat** će znakove `0x` odnosno `0X` na **početak** rezultata različitog od **nula** (inače ne radi ništa). Za `e`, `E`, `f`, `g` i `G`, ispisani broj će uvijek imati **decimalnu točku**. Za `g`, `G` **nule** na kraju decimalnog broja će se **ispisati**.