

# *Programiranje 1*

## *12. predavanje*

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

PMF – Matematički odjel, Zagreb

# Sadržaj predavanja

- Ulaz i izlaz podataka:
  - Funkcije getchar i putchar.
  - Funkcije gets i puts.
  - Funkcija scanf.
  - Funkcija printf.

# Informacije

Na mom [webu](#), pod **dodatnim** materijalima za **Prog1** i **Prog2**, nalazi se tekst

- 📄 [in\\_out.pdf](#) (7 stranica, 58 kB),

koji sadrži **detaljan** opis funkcija

- 📄 za **formatirani ulaz** i **izlaz** podataka.

Najveći dio teksta govori o funkcijama **fprintf** i **fscanf**.

U imenima ovih funkcija,

- 📄 **prvo** slovo **f** dolazi od riječi “file” (datoteka), a

- 📄 **zadnje** slovo **f** dolazi od “formatted” (formatirani).

Ove funkcije, u principu, rade za **bilo koju** datoteku, a

- 📄 **izlazna** ili **ulazna** datoteka se **zadaje** kao **argument**.

# Informacije — nastavak

Funkcije `printf` i `scanf` (bez prvog slova `f`)

- rade na **standardnim** datotekama za **izlaz**, odnosno **ulaz**, i zato se datoteka **ne zadaje**. To je **jedina** razlika!

Veza između **osnovne** funkcije (s prvim slovom `f`) i funkcije za **standardnu** datoteku dana je na **kraju** opisa osnovne funkcije.

Osnova za tekst je

- **Dodatak B** iz knjige **KR2**.

Tamo je opis **svih** funkcija iz standardne **C** biblioteke.

Međutim, “prijevod” **nije** doslovan. Neki dijelovi su prošireni i

- popravljen je opis **fscanf** (original nije skroz korektan).

Lijepo molim, ako uočite “**tipfelere**” — **javite** mi!

# Ulaz i izlaz podataka

# Sadržaj

- Ulaz i izlaz podataka:
  - Funkcije getchar i putchar.
  - Funkcije gets i puts.
  - Funkcija scanf.
  - Funkcija printf.

# Funkcije za ulaz/izlaz

U standardnoj ulazno–izlaznoj biblioteci postoje sljedeće funkcije za **ulaz/izlaz** podataka (za standardne ulazne, odnosno, izlazne datoteke **stdin, stdout**):

- **getchar, putchar** — za **znakove**,
- **gets, puts** — za **stringove**,
- **scanf i printf** — za **formatirani** ulaz/izlaz.

Program koji koristi neku od tih funkcija

- **mora** uključiti datoteku zaglavlja **<stdio.h>**.

# Funkcije *getchar* i *putchar*

Deklaracija ovih funkcija (zaglavlje, prototip) ima oblik:

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

Funkcija *getchar* čita **jedan znak** sa standardnog **ulaza** (tipično tipkovnice).

Funkcija **nema** argumenata pa je sintaksa poziva:

```
c_var = getchar();
```



## Funkcije `getchar` i `putchar` (nastavak)

Funkcija `putchar` šalje (tj. piše) **jedan znak** na standardni **izlaz** (tipično ekran).

Ona uzima **jedan** argument (**znak** koji treba ispisati) i **vraća** cjelobrojnu vrijednost.

Poziv funkcije, najčešće, ima oblik

---

```
putchar(c_var);
```

---

pri čemu se vraćena vrijednost **ignorira**.

**Pitanje:** Zašto u deklaracijama piše tip `int`, a **ne** tip `char`?

**Problem:** Kako prepoznati i “označiti” **kraj** podataka, odnosno, **grešku**?

## Funkcije `getchar` i `putchar` (nastavak)

Kada funkcija `getchar` naiđe na **kraj** ulaznih podataka — **vraća** vrijednost `EOF` (skraćeno od engl. `End of File`).

`EOF` je **simbolička** konstanta definirana u `<stdio.h>` koja signalizira **kraj** datoteke, odnosno, kraj ulaznih podataka (ulaz je tretiran kao datoteka `stdin`).

Konstanta `EOF` mora se **razlikovati** od znakova iz sustava znakova koje računalo koristi. Stoga funkcija `getchar` ne vraća vrijednost tipa `char`, već vrijednost tipa `int`, što daje dovoljno prostora za kôdiranje konstante `EOF` (obično `-1`).

Isto tako `putchar` uzima vrijednost tipa `int` i **vraća** vrijednost tipa `int`. **Vraćena** vrijednost je **znak** koji je ispisan ili `EOF`, ako ispis znaka **nije uspio**.

## Primjer za getchar i putchar

Primjer. Program koji **kopira znak po znak** s ulaza na izlaz i pritom sva slova **pretvara** u velika.

U datoteci zaglavlja `<ctype.h>` deklarirana je funkcija **toupper** koja **pretvara mala** slova u **velika**, a sve druge znakove ostavlja na miru.

Ova funkcija radi **isto** što i funkcija `malo_u_veliko` s prošlog predavanja.

## Primjer za getchar i putchar (nastavak)

```
#include <stdio.h>
#include <ctype.h>

int main(void) {
    int c;

    while ((c = getchar()) != EOF)
        putchar(toupper(c));

    return 0;
}
```

Pitanje: Što se događa ako piše samo `putchar(c);` ?

# Primjer rekurzivne funkcije — naopako pisanje

**Primjer.** Funkcija čita znakove sa standardnog ulaza, sve dok ne naiđe na prijelaz u novu liniju, i ispisuje učitane znakove obrnutim redosljedom, tj. naopako, pa se tako i zove.

```
void naopako(void) {
    char znak;
    if ((znak = getchar()) != '\n') naopako();
    putchar(znak);
    return;
}
```

**Rekurzija** služi pamćenju učitanih znakova u lokalnoj varijabli znak. Ispis nakon rekurzivnog poziva daje ispis unatrag (kako se vraćamo iz rekurzije).

# Naopako pisanje (nastavak)

Glavni program — funkcija `main` (v. `naopako.c`):

---

```
int main(void) {  
    printf(" Unesite niz znakova: ");  
    naopako();  
    return 0;  
}
```

---

Izvršavanjem s ulazom: `Zdravo`, dobit ćemo ovaj rezultat:

---

```
Unesite niz znakova: Zdravo
```

```
ovardZ
```

---

Prvo je ispisan je **zadnji** učitani znak `\n`.

## Kratko o stringovima

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

**Primjer.** Primijetite razliku između sljedećih deklaracija:

---

```
char niz_znakova[5] = {'h', 'e', 'l', 'l', 'o'};
```

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

---

pri čemu u prvoj na kraj niza znakova **ne** dolazi simbol `'\0'`.

**Zapamtiti:** Ime polja je **pokazivač** na **prvi** element polja.

Na primjer, `string = &string[0]`.

# Funkcije `gets` i `puts`

Deklaracija ovih funkcija ima oblik:

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

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

Funkcija `gets` čita **znakovni niz** sa standardnog ulaza. Kad naiđe na kraj linije '`\n`', zamjenjuje ga nul-znakom '`\0`'.

Funkcija vraća **pokazivač** na `char` koji pokazuje na **učitani** **znakovni niz** ili `NULL`, ako se došlo do **kraja** ulaznih podataka ili se javila **greška**.

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



## Funkcije `gets` i `puts` (nastavak)

Funkcija `puts` uzima kao argument **znakovni niz** koji će biti ispisan na standardnom izlazu.

Kod ispisa, `puts` zamjenjuje nul-znak `'\0'` (na kraju stringa) znakom `'\n'` za kraj reda.

Funkcija **vraća** cijeli broj (tipa `int`). Ta vrijednost je:

- broj ispisanih znakova (nenegativan) ako je ispis **uspjao**,
- a `EOF`, ako **nije**.

## Primjer za gets i puts

Primjer. Program koji kopira liniju po liniju ulaza na izlaz.

---

```
#include <stdio.h>

int main(void) {
    char red[128];

    while (gets(red) != NULL)
        puts(red);

    return 0;
}
```

---

## Primjer za gets i puts (nastavak)

Osnovni nedostatak funkcije `gets` je u tome što

- nije moguće odrediti maksimalni broj znakova koji će biti učitani.

Ako je broj znakova na ulazu veći od dimenzije polja koje je argument funkcije `gets`, doći će do greške (“gazimo” po memoriji).

Stoga je bolje, umjesto `gets`, koristiti funkciju `fgets` (bit će opisana kasnije, kod datoteka).

# Funkcija scanf

# Funkcija scanf

Funkcija `scanf` služi **formatiranom** učitavanju podataka sa standardnog ulaza. Opća forma poziva funkcije je

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

`kontrolni_string` je **konstantni** znakovni niz (string) koji sadrži informacije o vrijednostima koje se učitavaju u argumente `arg_1, ..., arg_n`.

Sastoji se od individualnih **grupa znakova** konverzije, tzv. **specifikacija konverzije**.

Svaka **specifikacija** konverzije **pridružena** je po jednom **sljedećem** argumentu — redom kako pišu.

## Funkcija scanf (nastavak)

Svaka grupa znakova konverzije **započinje** znakom postotka (%), a na **kraju** dolazi **znak konverzije** koji upućuje na **tip** podatka koji se učitava. Na pr. %c ili %d.

Najčešće korišteni znakovi konverzije su:

znak konverzije	tip podatka koji se učitava
c	jedan znak (char)
d	decimalni cijeli broj (int)
e, f, g	broj s pomičnim zarezom (float)
h	kratak cijeli broj (short)
:	:

## Funkcija scanf (nastavak)

znak konverzije	tip podatka koji se učitava
:	:
i	decimalni, heksadecimalni ili oktalni cijeli broj ( <code>int</code> )
o	oktalni cijeli broj ( <code>int</code> )
u	cijeli broj bez predznaka ( <code>unsigned int</code> )
x	heksadecimalni cijeli broj ( <code>int</code> )
s	string ( <code>char *</code> )
p	pokazivač ( <code>void *</code> )

## Funkcija scanf (nastavak)

- Unutar kontrolnog niza znakova grupe kontrolnih znakova mogu se nastavljati jedna na drugu **bez razmaka** ili mogu biti odvojene **bjelinama** (prazno mjesto, tabulator, prijelaz u novu liniju). Bjeline će u ulaznim podacima biti učitane i ignorirane.
- Argumenti funkcije **scanf** mogu biti samo **pokazivači** na varijable. Ako podatak treba učitati u neku varijablu, onda **scanf** uzima kao argument adresu te varijable.
- Podaci koje **scanf** čita dolaze sa standardnog ulaza, što je tipično tipkovnica. Ako se unosi više podataka oni **moraju** biti separirani **bjelinama**, što uključuje i prijelaz u novi red (koji se računa kao bjelina). Numerički podaci na ulazu **moraju** imati isti oblik kao i **numeričke konstante**.



## Učitavanje cijelih brojeva

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

**Primjer.** Komad programa

```
int x, y, z;  
...  
scanf("%i %i %i", &x, &y, &z);
```

učitava ulaznu liniju:

```
13  015  0Xd
```

onda je u **x**, **y** i **z** učitana vrijednost **13** (decimalno).

## Učitavanje cijelih brojeva (nastavak)

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

Primjer.

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

ispravno čita ulazne podatke:

```
13 15 d
```

i **svim varijablama** pridružuje vrijednost **13** (decimalno).

## Učitavanje cijelih brojeva (nastavak)

Podatak tipa `unsigned` učitavamo znakom konverzije `%u`.

Znakovi konverzije `d`, `i`, `o`, `u`, `x` mogu dobiti prefiks `h` ako je argument pokazivač na `short` te prefiks `l` ako je argument pokazivač na `long`.

Primjer.

---

```
int x;  
short y;  
long z;  
...  
scanf("%d %hd %ld", &x, &y, &z);
```

---

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

# Učitavanje realnih brojeva

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

Primjer.

---

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

---

Prefiks **L** koristi se ako je argument pointer na **long double**.

# Formatiranje i konverzija ulaza

- Funkcija `scanf` dijeli niz znakova na ulazu u **polja znakova** odvojena **bjelinama**.
- Polje znakova (u principu) **ne sadrži** bjeline.
- Svako **polje znakova** interpretira se prema odgovarajućem znaku **konverzije** i pripadna **vrijednost** se upisuje u varijablu na koju pokazuje odgovarajući argument funkcije.
- Svaki **znak konverzije** (u principu) **učitava jedno** ulazno polje.

## Maksimalna širina ulaznog polja

Uz svaki kontrolni znak može se zadati **maksimalna širina** ulaznog polja koje će se učitati — tako da se ispred kontrolnog znaka stavi **broj** koji određuje širinu polja.

**Primjer.**

**%3d** učitava cijeli broj s najviše tri znamenke.

**%11c** učitava najviše 11 znakova.

- Ako podatak sadrži manje znakova od zadane maksimalne širine polja on se učita samo do prve bjeline.
- Ako podatak ima više znamenaka od maksimalne širine polja, “višak” znamenaka bit će učitani sljedećim konverzijskim znakom ili sljedećom **scanf** funkcijom.

# Formatiranje i konverzija ulaza (nastavak)

Primjer.

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

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

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

## Bjeline u kontrolnom stringu

Znakovi konverzije mogu biti odijeljeni **bjelinama**:

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

Ta bjelina ima za posljedicu **preskakanje** svih bjelina na ulazu do početka novog ulaznog polja.

Stoga je pisanje znakova konverzije u kontrolnom znakovnom nizu razdvojeno bjelinama (kao u primjeru "%f %d") ili nerazdvojeno (kao "%f%d") posve ekvivalentno.

To **ne vrijedi** za znakove konverzije %c i [ (v. malo kasnije).



## Drugi znakovi u kontrolnom stringu

U kontrolnom znakovnom nizu mogu se pojaviti i **drugi znakovi** osim bjelina i znakova konverzije. Njima **moraju** odgovarati posve **isti znakovi na ulazu**.

**Primjer.** Ako realan i cijeli broj učitavamo naredbom

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

onda ulazni podaci **moraju** biti oblika, na pr.

```
1.456, 8
```

**bez bjeline** između prvog broja i zareza.

Tek sljedeći znak konverzije **%d** preskače sve eventualne **bjeline** na ulazu ispred “svog polja” (drugog broja).

## Formatiranje i konverzija ulaza (nastavak)

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

---

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

---

u kojoj **bjelina** nakon **%f** preskače sve eventualne bjeline na ulazu **ispred** zareza.

## Učitavanje znakovnih nizova — %s

Znak konverzije `%s` učitava niz znakova (string). Niz završava **prvom bjelinom** u ulaznom nizu znakova. Iza posljednjeg učitanoog znaka automatski se dodaje nul-znak (`\0`).

Primjer.

---

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

---

Budući da se svako polje kao argument funkcije interpretira kao pokazivač na prvi element polja, ispred varijable `string` **ne stavlja** se adresni operator.

## Učitavanje znakovnih nizova — %[...]

Znakom konverzije **%s** nije moguće učitati niz znakova koji sadrži bjeline, jer bjeline služe kao oznaka za kraj polja.

Za učitavanje nizova znakova koji uključuju i bjeline koristimo uglate zagrade kao znak konverzije %[...].

- Unutar uglatih zagrada upisuje se niz znakova.
- Funkcija **scanf** će učitati u pripadni argument najveći niz znakova s ulaza koji se sastoji od znakova navedenih unutar uglatih zagrada.
- Učitavanje završava prvi znak na ulazu koji nije naveden u uglatim zgradama i na kraj učitano g niza dodaje se nul-znak (**\0**).
- Vodeće bjeline se ne preskaču.

# Učitavanje znakovnih nizova — %[...]

Primjer. Naredba

```
char linija[128];  
...  
scanf(" %[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]", linija);
```

učitava najveći niz znakova sastavljen od velikih slova i razmaka.

- Prije %[ ostavljen je jedan razmak koji govori funkciji `scanf` da preskoči sve bjeline koje prethode znakovnom nizu.
- To je **nužno** ako smo već imali poziv `scanf` funkcije, jer ona ostavlja završni znak prijelaza u novi red u ulaznom nizu (**ne učitava**).

## Učitavanje znakovnih nizova — %[...]

Naredba

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

bi pročitala prethodni znak prijelaza u novi red i budući da on **nije** unutar uglatih zagrada, završila bi čitanje ulaznih podataka i **linija ne bi bila učitana**.

## Učitavanje znakovnih nizova — %[<sup>^</sup>...]

S uglatim zagradama možemo koristiti sintaksu

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

Sada se u odgovarajući argument učitava najveći mogući niz znakova sastavljen od svih znakova **osim** onih koji se nalaze u uglatim zagradama.

**Primjer.** Cijelu liniju bez znaka za prijelaz u novi red možemo učitati pomoću naredbe

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

Na kraj učitano g niza znakova bit će dodan `\0`, a ispred `%[` mora biti ostavljeno prazno mjesto kako bi bile preskočene sve prethodne bjeline.

## Učitavanje pojedinačnih znakova

Znak konverzije `c` učitava **jedan** znak u varijablu **bez obzira** je li on **bjelina** ili ne.

- Ako je prvi znak konverzije `c` potrebno je ispred njega staviti jednu bjelinu kako ne bi pročitao znak za prijelaz u novi red koji je ostao nakon prethodnog poziva funkcije `scanf`.
- Kontrolni niz "`%c%c%c`" čita tri znaka. Počet će s prvim znakom koji nije bjelina (zbog bjeline ispred prvog `%c` znaka) i pročitat će tri uzastopna znaka bili oni bjeline ili ne.
- Ako želimo čitati samo znakove **bez bjelina** treba koristiti "`%c %c %c`" ili `%c` zamijeniti s `%1s`.



## Prefiks \*

Neki podatak u listi moguće je **preskočiti** i **ne pridružiti** ga odgovarajućoj varijabli. To se radi tako da se znaku konverzije doda prefiks **\***.

Primjer.

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

**neće** izvršiti pridruživanje **drugog** podatka varijabli **n**. On će biti preskočen!

**Treći** podatak bit će normalno pridružen varijabli **n**.

**Svrha:** Preskakanje “kolona” u tablicama!

## Primjer za scanf

Primjer. Dio programa koji čita i piše podatke (v. [p\\_sc\\_02.c](#)).

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

Za ulaz:

56789 0123 56a72

dobivamo izlaz:

i = 56, x = 789.000000, s = 56

## Povratna vrijednost funkcije scanf

Funkcija `scanf` vraća **broj** uspješno učitanih podataka ili **EOF**, ako je do greške došlo **prije** nego što je uspješno učitani **prvi** podatak.

**Primjer.** Učitavanje brojeva većih ili jednakih od nule.

```
int n;

while (scanf("%d",&n) == 1 && n >= 0)
{
    // radi nesto s brojem
}
```

`while` petlja se prekida ako je učitani negativan broj ili ako unos broja nije uspio.

# Funkcija printf

# Funkcija printf

Funkcija `printf` služi za **formatirani ispis** podataka na standardnom izlazu (`stdout`). Opća forma poziva funkcije je

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

`kontrolni_string` je **konstantni** znakovni niz (string) koji sadrži informaciju o **formatiranju ispisa vrijednosti** argumenata `arg_1, ..., arg_n`.

**Kontrolni string** (ili “format-string”) ima posve **istu formu** i vrlo **sličnu funkciju** kao kod funkcije `scanf`.

Ostali argumenti `arg_1, ..., arg_n` su, općenito, **izrazi**.

# Funkcija printf (nastavak)

Kontrolni string sadrži dvije vrste objekata:

- obične znakove, koji se doslovno prepisuju (kopiraju) pri ispisu na izlaznu datoteku,
- specifikacije ili oznake konverzije. To su grupe znakova koje počinju znakom %, a završavaju nekim znakom konverzije. Između ovih znakova može biti još znakova, s posebnim značenjima (v. malo kasnije).

Svaka specifikacija konverzije vrši pretvaranje i ispis sljedećeg po redu (još 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 onda ispisuje.

# Funkcija printf (nastavak)

Najčešće korišteni **znakovi konverzije** su:

znak konverzije	tip podatka koji se ispisuje
d, i	decimalni cijeli broj ( <code>int</code> )
u	decimalni cijeli broj bez predznaka ( <code>unsigned int</code> )
o	oktalni cijeli broj ( <code>int</code> )
x	heksadecimalni cijeli broj ( <code>int</code> )
e, f, g	broj s pomičnim zarezom ( <code>double</code> )
c	jedan znak ( <code>char</code> )
s	string ( <code>char *</code> )
p	pokazivač ( <code>void *</code> )
%	nema konverzije, ispiši znak %

## Funkcija printf (nastavak)

**Uočiti:** ako treba **ispisati** znak %, onda unutar kontrolnog znakovnog niza **na tom mjestu** treba staviti %.

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

Funkcija **printf** koristi **prvi** argument (“format-string”)

- za određivanje **broja argumenata** koji slijede i **njihovih tipova!**

Ako je **argumenata** **premalo** ili su **pogrešnog tipa**,

- **printf** će se “**zbuniti**” i dobit ćete **pogrešne** rezultate.

Katkad dobijete **upozorenje** i kad je argumenata **previše** (obzirom na specifikacije u format-stringu).



## Funkcija printf — primjer

Primjer. Dio programa (v. `p_pr_00.c`)

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

ispisuje **izlaz** od **jednog** reda teksta:

```
%10d
```

Razlog: **%%** nije “prava” oznaka konverzije, već

• “nalog” za **doslovni** ispis **jednog** znaka **%**.

Dakle, **cijeli** format-string se “**doslovno**” ispisuje (**nema** oznaka konverzija). I još dobijem **upozorenje** od prevoditelja da

• format-string **završava** prije argumenta **n**.

## Doslovni ispis i konverzije — primjer

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

**Primjer.** Dio programa (v. `p_pr_01.c`)

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

ispisuje **jedan** red teksta (znak **x** je prvi znak u redu):

```
x=2.000000, y=1.414214
```

Svi znakovi koji **nisu** dio **specifikacije konverzije** ispisani su **točno** onako kako su napisani u format-stringu:

```
"x=%f, y=%f\n".
```

## 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` mogu biti pretvoreni u tip `int`, ako tako piše u oznaci konverzije (primjeri slijede).

Zbog toga, znak konverzije:

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

Slično vrijedi i za ostale “realne”, odnosno, “cjelobrojne” znakove konverzije.

Zato, oprez s tipovima.

- Nemojte ignorirati upozorenja prevoditelja, jer ne mora raditi dobro.

# Ispis znaka

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

- kao “običan” **znak** — **%c**, i
- kao **cijeli broj** — **%d** (uz pretvaranje tipova).

**Primjer.** Dio programa (v. **p\_pr\_02.c**)

---

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

---

ispisuje

---

```
c(char) = 1, c(int) = 49
```

---

ako računalo koristi **ASCII** skup znakova — broj **49** je ASCII kôd znaka **'1'**.

## Oktalni i heksadecimalni ispis

Pomoću znakova konverzije `%o` i `%x` ispisuju se cijeli brojevi u **oktalnom** i **heksadecimalnom** obliku, i to:

- **bez** predznaka i **bez** vodeće **nule**, odnosno, **0X**.

Ako želimo da za broj **različit** od **nule**

- **oktalni** ispis **ima** vodeći znak **0**, odnosno,

- **heksadecimalni** ispis **ima** vodeće znakove **0x**,

onda treba koristiti tzv. **alternativnu** formu ispisa.

- Dobiva se “**zastavicom**” (engl. “flag”) **#**.

# Oktalni i heksadecimalni ispis — primjer

Primjer. Dio programa (v. `p_pr_03.c`)

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

ispisuje

```
i(dec) = 64  
i(oct) = 100  
i(hex) = 40
```

## Oktalni i heksadecimalni ispis — primjer (nast.)

Primjer. Ako istu stvar napravimo za `i = -3` (v. `p_pr_04.c`), dobivamo

• `i(dec) = -3,`

• `i(oct) = 37777777775,`

• `i(hex) = ffffffff.`

**Objašnjenje:** sadržaj lokacije `i` na kojoj je spremljen `-3` konvertira se u **oktalni**, odnosno, **heksadecimalni** zapis, ali **bez predznaka**.

Ispis je isti kao da tu lokaciju

• interpretiramo **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, koji se piše **ispred** znaka konverzije (tj. kao **prefiks**).

Za **cjelobrojne** tipove modifikatori tipa su:

- **h** — označava da je argument tipa **short** ili **unsigned short**. Ako ga **ne** napišemo, dolazi do pretvaranja tipa u **int** ili **unsigned int**.
- **l** — označava da je argument tipa **long** ili **unsigned long**. Ovdje **nema** pretvaranja tipova, tj. treba napisati modifikator tipa (osim ako su **int** i **long** **isti**, pa stvar radi **slučajno**).

Na nekim sustavima postoji i **ll** za **long long** (kad ga ima).



## Ispis brojeva tipa short

Primjer. Dio programa (v. `p_pr_05.c`)

---

```
short i = -3;

printf("i(dec) = %d\n", i);
printf("i(oct) = %o\n", i);
printf("i(hex) = %x\n", i);
printf("h(dec) = %hd\n", i);
printf("h(oct) = %ho\n", i);
printf("h(hex) = %hx\n", i);
```

---

ispisuje isti broj tipa `short`

- pretvaranjem u `int` i
- bez pretvaranja — modifikator tipa `h`.

## Ispis brojeva tipa short (nastavak)

Ispis je

---

i(dec) = -3

i(oct) = 37777777775

i(hex) = ffffffff

h(dec) = -3

h(oct) = 177775

h(hex) = fffd

---

Probajte sami sa `short i = 3` (ništa se `ne` vidi).

## Ispis brojeva tipa long

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

Primjer. Cijeli program (v. `p_pr_06.c`)

---

```
#include <stdio.h>
#include <limits.h>

int main(void) {
    long i = LONG_MAX;

    printf("i(dec) = %ld\n", i);
    printf("i(oct) = %lo\n", i);
    printf("i(hex) = %lx\n", i);

    return 0; }
```

---

## Ispis brojeva tipa long (nastavak)

Ispis ovisi o računalu na kojem se program izvršava.

Na IA-32, s Intelovim C compilerom, rezultat je

---

i(okt) = 17777777777

i(hex) = 7fffffff

i(dec) = 2147483647

---

Tu je `int = 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 ispisivati pomoću znakova konverzije `%f`, `%g` i `%e`.

- `%f` — broj se ispisuje **bez** eksponenta.
- `%e` — broj se ispisuje **s** eksponentom.
- `%g` — način ispisa (s eksponentom ili bez njega) **ovisi o vrijednosti** koja se ispisuje.
  - Ako je eksponent **manji** od `-4` ili dovoljno **velik**, koristi se `%e`. U **protivnom**, koristi se `%f`.
  - Završne **nule iza** decimalne točke se **ne** ispisuju.

Za ispis brojeva tipa `long double` (ako postoje) koristimo **prefiks** (modifikator duljine) `L`.

- Pripadne specifikacije konverzije su `%Le`, `%Lf`, `%Lg`.

## Ispis realnih brojeva (nastavak)

Primjer. Dio programa (v. [p\\_pr\\_07.c](#))

```
double x = 12345.678;

printf("x(f) = %f\n", x);
printf("x(e) = %e\n", x);
printf("x(g) = %g\n", x);
```

ispisuje

```
x(f) = 12345.678000
x(e) = 1.234568e+004
x(g) = 12345.7
```

Za `%f` i `%e` imamo 6 decimala, a `%g` daje 6 vodećih znamenki.

# Minimalna širina ispisa

Uz **svaki** znak konverzije moguće je zadati **minimalnu širinu** ispisa, tj. **minimalni broj znakova** u ispisu, tako da se

- **ispred** znaka konverzije stavi odgovarajući **broj**.

Primjer.

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

Ako podatak treba:

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

## Minimalna širina ispisa (nastavak)

Primjer. Dio programa (v. `p_pr_08.c`)

```
int n = 54321;
```

```
printf("%10d,%5o,%5x\n", n, n, n);
```

ispisuje

```
54321,152061, d431
```

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



## Minimalna širina ispisa (nastavak)

Primjer. Dio programa (v. `p_pr_09.c`)

```
double x = 1.2;
```

```
printf("%1g\n%3g\n%5g\n", x, x, x);
```

ispisuje

```
1.2
```

```
1.2
```

```
1.2
```

Prva dva ispisa imaju točno 3 znaka u svom redu, dok treći ima točno pet znakova, tj. ima dvije vodeće bjeline.

# Preciznost ispisa realnih brojeva

Pored minimalne širine, moguće je zadati i **preciznost** ispisa. Kod realnih brojeva, **preciznost** je

- (najveći) broj **decimala** (za **%f** i **%e**), odnosno, **vodećih** znamenki (za **%g**), koje će biti ispisane.

Sintaksa:

- **%a.bf** ili **%a.be** ili **%a.bg**, gdje je
  - **a** — minimalna širina ispisa,
  - **b** — preciznost.

Primjer.

- **%7.3e** — znači ispis u **e** formatu s **najmanje 7** znakova, pri čemu su **najviše 3** znamenke iza decimalne točke.

Ispis **bez** specificirane **preciznosti**  $\iff$  **preciznost** = **6**.

# Preciznost ispisa realnih brojeva (nastavak)

Primjer. Ispis broja  $\pi$  na razne načine (v. `p_pr_10.c`)

```
#include <stdio.h>
#include <math.h>

int main(void) {
    double pi = 4.0 * atan(1.0);
    printf("%5f, %10.5f, %5.10f, %5.4f\n",
           pi, pi, pi, pi);
    return 0;
}
```

Rezultat ispisa je (zaokruživanjem na **zadani** broj decimala):

3.141593,      3.14159,    3.1415926536,    3.1416

## Dinamičko zadavanje širine i preciznosti

Širinu i preciznost ispisa moguće je odrediti **dinamički** — u trenutku **izvođenja** programa, tako da se

- **iznos širine** ili **preciznosti** u formatu **zamijeni** znakom **\***.

Na **pripadnom** mjestu u listi argumenata, koje **odgovara** tom znaku **\***, mora biti

- **cjelobrojni izraz** — obično, varijabla.

**Trenutna vrijednost** tog argumenta određuje **širinu**, odnosno, **preciznost**, tj.

- **“uvrštava”** se, tog trena, umjesto znaka **\***.

Vrijednost tog argumenta se **ne ispisuje** (argument se **“potroši”** na supstituciju umjesto **\***) i ide se dalje, na **sljedeći** argument.

## Dinamičko zadavanje širine i preciznosti (nast.)

Primjer. Opet, ispis broja  $\pi$  na razne načine (v. [p\\_pr\\_11.c](#))

```
#include <stdio.h>
#include <math.h>

int main(void) {
    double pi = 4.0 * atan(1.0);  int i = 10;
    printf("%*f, %*.*f, %5.*f\n",
           11, pi, 16, 14, pi, i, pi);
    return 0;
}
```

ispisuje

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 dođe do nul-znaka `\0`, kojeg **ne ispisuje**.

Primjer.

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

ispisuje

Programski jezik C

i prelazi u novi red, zbog `\n` iza `%s`.

## Ispis znakovnih nizova (nastavak)

Minimalna širina polja i preciznost mogu se koristiti i kod %s konverzije.

- Preciznost je maksimalni broj znakova koji smije biti ispisan.

Na primjer,

- %5.12s — specificira da će biti ispisano minimalno 5 znakova (dopunjenih bjelinama ako treba), a maksimalno 12 znakova.
- Ako string ima više od 12 znakova, “višak” neće biti ispisan (već samo prvih 12 znakova).

# Ispis znakovnih nizova (nastavak)

Primjer.

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

ispisuje

Programski jezik

Zadnji znak **k** je i **zadnji** znak u tom redu.