

Programiranje (C)

7. predavanje

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

`www.math.hr/~singer`

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Ulaz i izlaz podataka (drugi dio):
 - Funkcija printf.
- Kontrola toka programa:
 - Izrazi i naredbe.
 - Uvjetne naredbe if, if-else, switch.
 - Petlje while, for, do-while.
 - Naredbe break i continue.
 - Naredba goto.

Bitno: 1. domaća zadaća je na Webu od ponedjeljka (3.4.).

- Riješite i prijavite se na vrijeme.

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)
```

gdje je `kontrolni_string` konstantan znakovni niz (string) koji sadrži informaciju o **formatiranju ispisa** vrijednosti argumenta `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 konverzije. To su grupe znakova koje počinju znakom %, a završavaju nekim znakom konverzije. Između ovih znakova može biti još znakova, s posebnim značenjima.

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 (int)
u	decimalni cijeli broj bez predznaka (unsigned int)
o	oktalni cijeli broj (int)
x	heksadecimalni cijeli broj (int)
e, f, g	broj s pomičnim zarezom (double)
c	jedan znak (char)
s	string (char *)
p	pokazivač (void *)
%	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.

Pri pozivu funkcije **printf** dolazi do konverzije tipova:

- argumenti tipa **char** i **short** konvertiraju se u tip **int**,
- a argumenti tipa **float** u **double**.

Zbog toga, znak konverzije:

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

Funkcija printf — primjer

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

Primjer: (v. **printf_1.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 uneseni u kontrolnom znakovnom nizu:

"**x=%f, y=%f\n**"

.

Funkcija printf — primjer

Znak možemo ispisati kao cijeli broj (%d) i kao jedan znak (%c).

Primjer:

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

ispisuje

```
c(int)=119, c(char)=w
```

ako računalo koristi ASCII skup znakova (broj 119 je ASCII kôd znaka “w”).

Oktalni i heksadecimalni ispis

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

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

Primjer: (v. `printf_2.c`)

```
short i = 64;  
...  
printf("i(okt)=%o: i(hex)=%x: i(dec)=%d\n",  
      i, i, i);
```

ispisuje

`i(okt)=100: i(hex)=40: i(dec)=64`

Oktalni i heksadecimalni ispis (nastavak)

Probajte isti program za `i = -3` (može tipa `short` ili `int`).

Dobijem (v. `printf_3.c` i `printf_4.c`):

- `i(okt) = 3777777775,`
- `i(hex) = ffffffd,`
- `i(dec) = -3.`

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.

Ispis brojeva tipa long

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

Primjer:

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

long i = LONG_MAX;
int main(void) {
    printf("i(okt) = %lo\n", i);
    printf("i(hex) = %lx\n", i);
    printf("i(dec) = %ld\n", i);

    return 0;
}
```

Ispis brojeva tipa long (*nastavak*)

Program, ovisno o računalu na kojem se izvršava, može ispisati:

i(okt) = 177777777777

i(hex) = 7fffffff

i(dec) = 2147483647

I ja dobijem isto na Intelovom compileru na IA-32
(v. [printf_5.c](#)).

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.

Za ispis brojeva tipa `long double` koristimo prefiks L.
Pripadne specifikacije konverzije su `%Le`, `%Lf`, `%Lg`.

Ispis realnih brojeva (nastavak)

Primjer: dio programa (v. [printf_6.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
```

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).
- **više** znakova od minimalne širine ispisa, bit će isписан sa **svim** potrebnim znakovima.

Minimalna širina ispisa (nastavak)

Primjer:

```
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 ispisa, kod realnih brojeva moguće je zadati i **preciznost ispisa**, tj.

- (najveći) broj decimala koje će biti ispisane.

Sintaksa:

- **%a.bf** ili **%a.bg** ili **%a.be**, 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 daje **šest decimala**.

Preciznost ispisa realnih brojeva (nastavak)

Primjer: ispis broja π na razne načine (v. [printf_7.c](#)).

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

int main(void) {
    double pi = 4.0 * atan(1.0);
    printf("%5f  %5.5f  %5.10f\n", pi, pi, pi);
    return 0;
}
```

Rezultat ispisa je ([zaokruživanjem](#) na [zadani](#) broj decimala):

3.141593 3.14159 3.1415926536

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**.

Dinamičko zadavanje širine i preciznosti (nast.)

Primjer: (v. `printf_8.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 ispisani.

Na primjer,

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

Kontrola toka programa

Izrazi i naredbe

Izraz je svaka kombinacija operatora i operanada koju jezik dozvoljava. Svaki izraz ima svoju vrijednost koja se dobiva

- izvršavanjem svih operacija u izrazu,
redoslijedom prema prioritetu.

Primjer:

```
x = 3      n++      printf(...)
```

Poziv funkcije je, također, izraz — čak i kad “odbacujemo” povratnu vrijednost (ako je ima).

Izrazi i naredbe (nastavak)

Program se, općenito, sastoji od niza naredbi.

- Naredbe završavaju znakom točka–zarez ;.

Svaki izraz iza kojeg slijedi točka–zarez postaje naredba (tzv. osnovna ili primitivna naredba).

Primjer:

```
x = 3;  
n++;  
printf(...);
```

Osim ovih, postoje još i složene naredbe, te posebne naredbe (s imenom) za kontrolu redoslijeda izvršavanja ostalih naredbi (tzv. naredbe za kontrolu toka).

Složena naredba

Složena naredba (blok, blok–naredba ili blok naredbi) je

- grupa deklaracija i naredbi, zatvorena u vitičaste zagrade { i }.

Primjer:

```
{x = 3;  
 n++;  
 printf(...);}
```

Uočiti da nema točka–zareza iza zatvorene zagrade }.

Složena naredba je sintaktički ekvivalentna jednoj naredbi, tj. može se pojaviti na istim mjestima gdje se može pojaviti i jednostavna naredba.

Uvjetno izvršavanje — if naredba

Najjednostavnija **if** naredba ima oblik:

```
if (uvjet) naredba;
```

gdje je **uvjet** aritmetički (ili pokazivački) izraz.

Redoslijed **izvršavanja**:

- Prvo se računa vrijednost izraza **uvjet**.
- Ako je ta vrijednost **različita od nule** (tj. **istina**) onda se **izvršava naredba**.
- Ako je ta vrijednost **jednaka nuli** (tj. **laž**), onda se **naredba ne izvršava** i program se nastavlja prvom naredbom **iza if** naredbe.

if naredba (nastavak)

Primjer:

```
int x;  
...  
if (x > 0) printf("\n x= %d \n", x);  
x++;
```

ispisuje samo **pozitivne** vrijednosti varijable **x**.

Ovaj oblik **if** naredbe je:

- **uvjetno izvršavanje jedne** naredbe.

Alternative su (ovisno o uvjetu): **izvrši** ili **ne**.

if-else *naredba*

if-else naredba ima oblik:

```
if (uvjet)
    naredba1;
else
    naredba2;
```

Ako izraz **uvjet**

- ima vrijednost **istine**, onda se **izvršava naredba1**,
- a u suprotnom **naredba2**.

Ovo je:

- uvjetno izvršavanje jedne od dviju naredbi.**

Alternative su (ovisno o uvjetu): izvrši **jednu ili drugu**.

if-else *naredba (nastavak)*

Primjer:

```
#include <stdlib.h>

...
if (!x) {
    printf("Djelitelj jednak nuli!\n");
    exit(-1);
}
else
    y /= x;
```

Uočiti: **!x** je **istina** ako i samo ako je **x == 0**.

Funkcija exit

Funkcija:

```
void exit(int status)
```

deklarirana je u datoteci zaglavlja `<stdlib.h>`. Ona

- zaustavlja izvršavanje programa
i vrijednost `status` predaje operacijskom sustavu.

Standardno, `status = 0` znači da je program uspješno
završen, a vrijednost različita od nule znači da se program
zaustavio zbog greške.

if naredba i uvjetni operator

Sljedeće dvije naredbe su ekvivalentne:

```
max = a >= b ? a : b;
```

i

```
if (a >= b)
    max = a;
else
    max = b;
```

Višestruki izbor if-else naredbama

Naredbe **if-else** mogu se **ugnijezditi**.

Primjer: dvije **if-else** naredbe, druga **iza else** od prve.

```
if (uvjet1)
    naredba1;
else if (uvjet2)
    naredba2;
else
    naredba3;
```

Primjer. Učitavaju se dva broja i jedan znak koji označava osnovnu računsku operaciju (**z**, **o**, **m**, **d**). U ovisnosti o učitanom znaku izvršava se jedna od četiri računske operacije (**+**, **-**, *****, **/**) na učitanim brojevima.

Višestruki izbor if-else naredbama (nastavak)

```
#include <stdio.h>

int main(void)
{
    float a, b;
    char operacija;

    printf("Upisati prvi broj: ");
    scanf(" %f", &a);
    printf("Upisati drugi broj: ");
    scanf(" %f", &b);
    printf("Upisati operaciju: z, o, m, d:\n");
    scanf(" %c", &operacija);
```

Višestruki izbor if-else naredbama (nastavak)

```
if (operacija == 'z')
    printf("%f\n", a + b);
else if (operacija == 'o')
    printf("%f\n", a - b);
else if (operacija == 'm')
    printf("%f\n", a * b);
else if (operacija == 'd')
    printf("%f\n", a / b);
else
    printf("Nedopustena operacija!\n");

return 0;
}
```

Kojem if pripada else?

Problem: Kad imamo ugniježđene **if** i **if-else** naredbe, kojem **if** pripada **else** — prvom ili drugom?

Pravilo: Svaka **else** naredba pripada **najbližoj** (prethodnoj) **if** naredbi. (Razlog: compiler u danom trenu uvijek “jede” **najdulju** moguću jezičku cjelinu.)

Primjer:

```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

```
if (n > 0)
    if (a > b) z = a;
    else /* LOS STIL */
        z = b;
```

Obje varijante, naravno, rade **isto**.

Kojem if priпада else? (nastavak)

Pripadnost mijenjamo grupiranjem u složenu naredbu, tj. korištenjem vitičastih zagrada.

Primjer:

```
if (n > 0) {           if (n > 0) {
    if (a > b)         if (a > b)
        z = a;          z = a;
}
else
    z = b;            }
else /* LOS STIL */
    z = b;
```

Obje varijante, opet, rade **isto**.

Višestruki izbor — switch naredba

Naredba **switch** slična je nizu ugniježđenih **if-else** naredbi. Opći oblik je:

```
switch (izraz) {  
    case konstanta_1: naredbe_1;  
                        /* moze vise naredbi! */  
    case konstanta_2: naredbe_2;  
    ...  
    case konstanta_n: naredbe_n;  
    default:           naredbe;  
}
```

Vrijednost izraza određuje ili selektira odgovarajući slučaj (**case**) i, eventualno, slučajeve ispod njega.

switch naredba (nastavak)

Osnovna pravila:

- izraz u switch naredbi mora imati cijelobrojnu vrijednost (tipovi char, int ili enum).
- Nakon ključne riječi case pojavljuju se cijelobrojne konstante ili konstantni izrazi, iza koji mora biti znak : (dvotočka).

Redoslijed izvršavanja u switch naredbi:

- Prvo se računa vrijednost izraza izraz.
- Zatim se provjerava je li dobivena vrijednost jednaka jednoj od konstanti: konstanta_1, ..., konstanta_n. Ove konstante moraju biti međusobno različite.

switch *naredba* (*nastavak*)

- Ako je **izraz** = **konstanta_i**
 - program **nastavlja** naredbama **naredbe_i** (može ih biti više, bez vitičastih zagrada),
 - i **svim naredbama** koje dolaze iza njih (u ostalim slučajevima **ispod** tog), sve do prve **break** naredbe (ako je ima) ili do kraja **switch** naredbe. Nakon toga program nastavlja **prvom naredbom** iza **switch** naredbe.
- Ako **izraz** **nije jednak** niti jednoj navedenoj konstanti,
 - program **izvršava naredbe** iza **ključne riječi default** (ako postoji), i **sve naredbe** iza njih, do **break** ili do kraja **switch** naredbe.

switch naredba (nastavak)

- Slučaj **default** ne mora nužno biti prisutan u **switch** naredbi. Ako **nije** i ako **nema** podudaranja izraza i konstanti,
 - program nastavlja **prvom** naredbom iza **switch** naredbe,
tj. **ne izvršava** niti jednu naredbu iz **switch**.
- Slučajevi oblika **case konstanta_i** i slučaj **default** (ako ga ima) mogu biti napisani **bilo kojim redom**.
 - Na primjer, **default** može biti i **prvi**, na samom početku **switch** naredbe.

Primjer. Program s izborom aritmetičke operacije (od malo prije) sad realiziramo **switch** naredbom (preglednije).

switch *naredba* (*nastavak*)

```
#include <stdio.h>

int main(void)
{
    float a, b;
    char operacija;

    printf("Upisati prvi broj: ");
    scanf(" %f", &a);
    printf("Upisati drugi broj: ");
    scanf(" %f", &b);
    printf("Upisati operaciju: z, o, m, d:\n");
    scanf(" %c", &operacija);
```

switch *naredba* (*nastavak*)

```
switch (operacija) {
    case 'z': printf("%f\n", a + b);
                break;
    case 'o': printf("%f\n", a - b);
                break;
    case 'm': printf("%f\n", a * b);
                break;
    case 'd': printf("%f\n", a / b);
                break;
    default: printf("Nedopustena operacija!\n");
}
return 0;
```

Ispuštanje break naredbe

Ispušteni **break** vodi na “propadanje kôda” u **niži case** blok.

Primjer: dio programa koji ispisuje **korektne** poruke!

```
int i;  
...  
switch (i) {  
    case 1:  
    case 2:  
    case 3: printf("i < 4\n");  
              break;  
    case 4: printf("i = 4\n");  
              break;  
    default: printf("i > 4\n");  
}
```

while *petlja*

while petlja ima oblik:

while (izraz) naredba;

naredba se izvršava sve dok je izraz istinit (različit od 0).

Primjer: sljedeći dio programa ispisuje brojeve 0, 1, . . . , 9.

```
i = 0;  
while (i < 10) {  
    printf("%d\n", i);  
    i++;  
}
```

while petlja najčešće se koristi kad se broj ponavljanja ne zna unaprijed, već je pod kontrolom uvjeta izraz.

while *petlja (nastavak)*

Primjer. Program čita **niz** brojeva **različitih** od **nule**, sve dok se ne upiše **nula**, i računa **srednju vrijednost** tog niza (bez zadnje nule).

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    double sum = 0.0, x;

    printf(" Upisite niz brojeva != 0,"
           " i nulu za kraj.\n");
    printf(" x[0]= ");
    scanf("%lf", &x);
```

while petlja (nastavak)

```
while (x != 0.0) {  
    sum += x;  
    printf(" x[%d]= ", ++i);  
    scanf("%lf", &x);  
}  
sum /= i;  
printf(" Srednja vrijednost = %f\n", sum);  
return 0;  
}
```

Oprez! Što se događa ako **odmah** upišemo **nulu** kao **prvi** broj, tj. imamo “**prazan**” niz?

- Dijeljenje s nulom!

Popravak: treba dodati test **if (i > 0) ...**

for petlja

for petlja ima oblik:

```
for (izraz_1; izraz_2; izraz_3) naredba;
```

i ekvivalentna je s

```
izraz_1;
while (izraz_2) {
    naredba;
    izraz_3;
}
```

Beskonačna petlja koja ne radi ništa (default izraz2 = 1):

```
for (;;);
```

for petlja (*nastavak*)

Primjer. Napišimo implementaciju funkcije `atoi` iz standardne biblioteke (zaglavje `<stdlib.h>`), koja prevodi niz znakova koji predstavlja **cijeli broj** u njegovu numeričku vrijednost. Funkcija treba:

- preskočiti sve početne **bjeline**,
- učitati **predznak** (ako ga ima),
- i redom pročitati sve **znamenke**,

te ih prevesti u **broj** tipa `int`.

```
#include <ctype.h>

int atoi(const char s[])
{
```

for petlja (*nastavak*)

```
int i, n, sign;

for (i = 0; isspace(s[i]); i++)
    /* preskace sve bjeline */
; /* prazna naredba */
sign = (s[i] == '-') ? -1 : 1;
if (s[i] == '+' || s[i] == '-') i++;
    /* preskoci predznak */
for (n = 0; isdigit(s[i]); i++)
    n = 10 * n + (s[i] - '0'); /* Horner */
return sign * n;
}
```

Ovdje koristimo funkcije `isspace` i `isdigit` koje su deklarirane u datoteci zaglavlja `<ctype.h>`.

Funkcije iz <ctype.h>

Funkcije iz <ctype.h> koje uzimaju jedan znak i vraćaju vrijednost tipa int različitu od nule ako je odgovarajući uvjet ispunjen, odnosno nulu ako nije.

Popis funkcija:

int isalnum(int c);	/* Alfanumericki */
int isalpha(int c);	/* Slovo */
int iscntrl(int c);	/* Kontrolni */
int isdigit(int c);	/* Numericki, znam. */
int isxdigit(int c);	/* Heksadec. znam. */
int islower(int c);	/* Malo slovo */
int isupper(int c);	/* Veliko slovo */
int isspace(int c);	/* Bjelina, tab, . . . */

Funkcije iz <ctype.h> (*nastavak*)

Popis funkcija (nastavak):

```
int isgraph(int c);    /* Ispisiv bez blanka */
int isprint(int c);   /* Ispisiv i blank */
int ispunct(int c);   /* Ispisiv bez blanka,
                      slova i brojki */
```

Funkcije konverzije za slova:

```
int tolower(int c);   /* Velika u mala */
int toupper(int c);  /* Mala u velika */
```
