

# *Programiranje (C)*

## *8. predavanje*

Saša Singer

singer@math.hr  
web.math.hr/~singer

PMF – Matematički odjel, Zagreb

# *Sadržaj predavanja*

- Tipovi i složene deklaracije:
  - Pokazivač i polja. Pokazivači i funkcije.
  - Primjeri tipova.
  - Složene deklaracije.
  - Deklaracija tipova — `typedef`.
- Strukture (prvi dio):
  - Deklaracija strukture. Strukture i `typedef`.
  - Rad sa strukturama. Operator točka.
  - Strukture i funkcije.
  - Strukture i pokazivači. Operator strelica (`->`).
  - Unije.

# Tipovi i složene deklaracije

# Sadržaj

- Tipovi i složene deklaracije:
  - Pokazivači i polja (ponavljanje).
  - Pokazivači i višedimenzionalna polja.
  - Pokazivač na funkciju.
  - Primjeri tipova.
  - Složene deklaracije.
  - Deklaracije tipova — `typedef`.

# *Polje pokazivača*

Polje pokazivača ima deklaraciju:

---

```
tip_pod *ime[izraz];
```

---

Napomena: Primarni operator [ ] ima viši prioritet od unarnog operatora \*.

Primjer. Razlikujte polje pokazivača (ovdje 10 pokazivača):

---

```
int *ppi[10];
```

---

od pokazivača na polje (ovdje od 10 elemenata):

---

```
int (*ppi)[10];
```

---

# *Pokazivač na funkciju*

Pokazivač na funkciju deklarira se kao:

---

```
tip_pod (*ime)(tip_1 arg_1, ..., tip_n arg_n);
```

---

Ovdje je **ime** varijabla tipa — pokazivač na funkciju koja

- uzima **n** argumenata tipa **tip\_1** do **tip\_n**
- i vraća vrijednost tipa **tip\_pod**.

Slično kao i u prototipu funkcije, ne treba pisati imena argumenata **arg\_1** do **arg\_n**.

Primjer:

---

```
int (*pf)(char c, double a);
int (*pf)(char, double);
```

---

# *Pokazivač na funkciju (nastavak)*

U deklaraciji pokazivača na funkciju — zgrade su nužne.

- Primarni operator ( ) — “poziva” ili argumenata funkcije, ima viši prioritet od unarnog operatora \*.

Primjer. Razlikujte funkciju koja vraća pokazivač na neki tip (ovdje na double):

---

```
double *pf(double, double);
double *(pf(double, double)); /* Isto */
```

---

od pokazivača na funkciju koja vraća vrijednost nekog tipa (ovdje double):

---

```
double (*pf)(double, double);
```

---

## *Pokazivač na funkciju — primjeri*

Primjeri pokazivača na funkciju iz integraciju (prošli put):

---

```
double integracija(double, double,  
                  double (*)(double));
```

```
double integracija(double a, double b,  
                  double (*f)(double)) {  
    return 0.5 * (b - a) * ( (*f)(a) + (*f)(b) );  
}
```

---

ili:

---

```
double integracija(double, double, int,  
                  double (*)(double));
```

...

---

## *Složene deklaracije*

Pri interpretaciji deklaracije uzimaju se u obzir prioriteti pojedinih operatora. Ti prioriteti mogu se promijeniti upotrebom zagrada.

Primjeri: p je:

---

```
int *p[10];          /* polje od 10 ptr na int */
int *p(void);        /* funkcija koja nema arg i
                      vraca pokazivac na int */
int p(char *a);      /* funkcija koja uzima ptr na
                      char i vraca int */
int *p(char *a);    /* funkcija koja uzima ptr na
                      char i vraca ptr na int */
int (*p)(char *a); /* ptr na funkciju koja uzima
                      ptr na char i vraca int */
```

## *Složene deklaracije (nastavak)*

```
int (*p(char *)) [10]; /* funk. uzima ptr na char  
                      i vraca ptr na polje  
                      od 10 elt tipa int */  
int p(char (*a) []);    /* funk. uzima ptr na polje  
                      znakova i vraca int */  
int (*p)(char (*a) []); /* ptr na funk. koja uzima  
                      ptr na polje znakova i  
                      vraca int */  
int *(*p)(char (*a) []); /* ptr na funk. koja uzima  
                      ptr na polje znakova i  
                      vraca ptr na int */  
int *(*p[10])(char *a); /* polje 10 ptr na funk.  
                      koja uzima ptr na char  
                      i vraca ptr na int */
```

# Deklaracije tipova — `typedef`

# Ključna riječ `typedef`

Korištenjem ključne riječi `typedef`

- postojećim tipovima podataka dajemo nova imena (ne kreiramo nove objekte ili nove tipove!).

Jednostavni oblik `typedef` deklaracije je:

---

```
typedef tip_podatka novo_ime_za_tip_podatka;
```

---

To znači da:

- `novo_ime_za_tip_podatka` postaje sinonim za `tip_podatka`

i smije se tako koristiti u svim kasnijim deklaracijama — tamo gdje smijemo napisati jedno, smijemo napisati i drugo, i to s istim značenjem.

# **Jednostavne *typedef* deklaracije**

Primjer. Deklaracijom

---

```
typedef double Masa;
```

---

identifikator **Masa** postaje **sinonim** za **double**.

Nakon toga, varijable tipa **double** možemo deklarirati i kao:

---

```
Masa m1, m2, *pm1;
```

```
Masa elementi[10];
```

---

Uočite da je

- **pm1** — pokazivač na **double**,
- **elementi** — polje od 10 elemenata tipa **double**.

Međutim, nije baš jasno što smo s tim “**dobili**”!

## *Svrha deklaracije tipova*

Zaista, kod ovako **jednostavnih** deklaracija — svrha se **ne** vidi odmah.

Stvarna **svrha** deklaracije ili **imenovanja tipova** je:

- lakše **razumijevanje** (čitanje) kôda i
- **dokumentiranje** programa.

To postaje **vrlo korisno** kod **složenijih** tipova podataka — kad u programo koristimo

- čitavu **hijerahiju** tipova — koji se grade jedni iz drugih.

Korist će se vidjeti vrlo **skoro**, kad dođemo na

- **strukture** i **samoreferencirajuće strukture** (vezane liste, binarna stabla i sl.).

# *Primjer jednostavne deklaracije tipova*

Korist od deklaracije **tipova** može se vidjeti i na jednostavnim primjerima — ako dobro izaberemo ime za **tip**.

Primjer.

---

```
typedef int Metri, Kilometri;  
Metri duljina, sirina;  
Kilometri udaljenost;
```

---

Ideja (ili svrha): ovdje **ime tipa** sugerira **jedinice** u kojima su izražene određene vrijednosti!

No, stvarna **korist** od **typedef** je tek kod **složenijih** tipova.

- Kako se pišu takve deklaracije?

## *Složenije typedef deklaracije*

Sasvim općenito, deklaracija imena za složeniji tip:

- počinje s **typedef**, a
- dalje ima **isti** oblik kao i deklaracija **variable** tog **imena** i tog **tipa**.

Sve je isto, osim što tada

- ime nije varijabla tog **tipa** (ne dobiva memorijski prostor i adresu), već
- ime postoji **sinonim** za **tip** kojeg “bi imala” takva varijabla.

---

```
typedef deklaracija_za_tip_podatka;
```

---

## *Primjer složenije deklaracije tipa — za polja*

Primjer. Uvedimo imena tipova za vektore i matrice odgovarajućih dimenzija (recimo,  $n = 10$ ).

---

```
#define n 10
typedef double skalar;
typedef skalar vektor[n];
typedef vektor matrica[n];
```

---

Zadnje dvije deklaracije daju imena poljima:

- **vektor** je ime tipa za polje od  $n$  (10) skalara (**double**),
- **matrica** je ime tipa za polje od  $n$  (10) vektora, tj.
- **matrica** je dvodimenzionalno polje skalara, ili sinonim za tip **double[n][n] = double[10][10]**.

## *typedef i polja (nastavak)*

Funkciju za računanje produkta  $y = Ax$ , kvadratne matrice  $A$  i vektora  $x$ , možemo i ovako napisati:

```
void prod_mat_vek(matrica a, vektor x, vektor y)
{
    int i, j;
    for (i = 0; i < n; ++i) {
        y[i] = 0;
        for (j = 0; j < n; ++j)
            y[i] += a[i][j] * x[j];
    }
}
```

**Napomena.** Ovdje je **n** fiksan —  $n = 10$ . Popravite funkciju tako da stvarni red matrice i vektora bude argument funkcije.

## *Primjer deklaracije tipa — stringovi*

Primjer. Kod obrade stringova možemo uvesti deklaraciju

---

```
typedef char *string;
```

---

Ovdje je **string** sinonim za **pokazivač** na **char** (tip **char \***), s **očitom** svrhom:

- taj **pokazivač interpretiramo** kao pokazivač na **prvi element** u **polju znakova**,
- a to **polje znakova** obrađujemo kao **string** (do nul-znaka)!

Funkcija **strcmp** za **uspoređivanje stringova** smije se ovako deklarirati:

---

```
int strcmp(string, string);
```

---

## *typedef i pokazivači (nastavak)*

Primjer. Pokazivač na **double** nazvat ćemo **Pdouble**.

---

```
typedef double *Pdouble;
```

---

**Pdouble** postaje **pokazivač** na **double**, pa smijemo pisati:

---

```
Pdouble px;      /* = double *px */

void f(Pdouble, Pdouble);
/* = void f(double *, double *); */

px = (Pdouble) malloc(100 * sizeof(Pdouble));
```

---

## *typedef i deklaracije funkcija*

Općenito, **typedef** koristimo za kraće zapisivanje složenih deklaracija.

Primjer. Pokazivač na funkciju.

---

```
typedef int (*PF)(char *, char *);
```

---

PF postaje ime za pokazivač na funkciju koja uzima dva pokazivača na **char** i vraća **int**. Umjesto deklaracije:

---

```
void f(double x, int (*g)(char *, char *)) { ... }
```

---

možemo pisati:

---

```
void f(double x, PF g) { ... }
```

---

# Strukture

# Sadržaj

- Strukture (prvi dio):
  - Deklaracija strukture. Strukture i `typedef`.
  - Rad sa strukturama. Operator točka.
  - Strukture i funkcije.
  - Strukture i pokazivači. Operator strelica (`->`).
  - Unije.

# Što je struktura?

Struktura je složeni tip podataka, kao i polje. Za razliku od polja, koje služi

- grupiranju podataka istog tipa,  
struktura služi
- grupiranju podataka različitih tipova.

Može i ovako — malo detaljnije.

- Svi elementi polja imaju isti tip i zajedničko ime, a razlikuju se po indeksu. To se vidi i u deklaraciji polja.
- Elementi (ili članovi) strukture mogu, ali ne moraju, biti različitog tipa i svaki element ima svoje posebno ime.

Zato u deklaraciji strukture moramo navesti ime i tip svakog člana. Tip strukture možemo deklarirati na dva načina.

# Deklaracija strukture — bez `typedef`

Prvi način — bez `typedef`. Tip strukture deklarira se ovako:

```
struct ime {  
    tip_1 ime_1;  
    tip_2 ime_2;  
    ...  
    tip_n ime_n;  
};
```

Ovdje je `struct` rezervirana riječ, a `ime` je ime strukture.

Stvarni `tip` strukture je

- `struct` `ime` — dvije riječi (i to je nezgodno!).

Unutar vitičastih zagrada popisani su `članovi` strukture.

## *Definicija varijabli tipa strukture — bez typedef*

Napomena. Kao i kod polja, članovi strukture

- smješteni su u memoriji **jedan za drugim**.

Kod ovakve deklaracije **tipa** strukture, **varijable** tog **tipa**, općenito, definiramo ovako:

---

```
mem_klasa struct ime var_1, var_2, ..., var_n;
```

---

- **var\_1, var\_2, ..., var\_n** su varijable **tipa struct ime**.

## **Primjer — struktura za točke**

Primjer. Struktura **tocka** definira točku u ravnini. Uzmimo da točka ima cijelobrojne koordinate, poput **pixela** na ekranu.

---

```
struct tocka {  
    int x;  
    int y;  
};
```

---

Varijable tipa strukture **tocka** možemo definirati na (barem) **dva** načina.

Nakon gornje deklaracije **strukture tocka** (kao tipa), napišemo “običnu” definiciju **varijabli**:

---

```
struct tocka t1, t2;
```

---

## **Primjer — struktura za točke (nastavak)**

Deklaraciju **tipa** strukture **ne** moramo napisati **posebno**.

Možemo ju napisati i **u sklopu** definicije **varijabli** tog tipa:

---

```
struct tocka {  
    int x;  
    int y;  
} t1, t2;
```

---

**Prvi način je pregledniji!**

Međutim, postoji i **bolji** način deklaracije **tipa strukture**, koji olakšava i definiciju **varijabli** tog tipa — preko **typedef**.

**Prednost:** tako možemo **izbjegći** stalno navođenje riječi **struct** u deklaracijama varijabli.

# *Deklaracija strukture — preko `typedef`*

Drugi način — preko `typedef`.

Tip strukture deklarira se ovako:

---

```
typedef struct ime {  
    tip_1 ime_1;  
    tip_2 ime_2;  
    ...  
    tip_n ime_n;  
} ime_tipa;
```

---

Ovdje smo još, na kraju deklaracije, cijelom **tipu** strukture dali ime `ime_tipa`. Stvarni **tip** strukture je onda i

- `ime_tipa` — kao **sinonim** za `struct ime`.

Sve ostalo je isto kao i prije.

## *Definicija varijabli tipa strukture — uz typedef*

Napomena. “Prvo” **ime** strukture (odmah iza **struct**) smijemo i **ispustiti**, ako ga nigdje nećemo koristiti! (Uvijek bi trebalo pisati **struct** ispred tog imena.)

Kod ovakve deklaracije **tipa** strukture, **variable** tog **tipa**, općenito, definiramo ovako:

---

```
mem_klasa ime_tipa var_1, var_2, ..., var_n;
```

---

- **var\_1, var\_2, ..., var\_n** su varijable **tipa** **ime\_tipa**, što je **sinonim** za **struct** **ime**.

## **Primjer — struktura za točke**

**Primjer.** Umjesto ranije definicije strukture za točku u ravnini, možemo uvesti tip **Tocka** za **cijelu** strukturu.

```
typedef struct {  
    int x;  
    int y;  
} Tocka;  
  
...  
Tocka t1, t2, *pt1;
```

Identifikator **Tocka** je ime **tipa** za cijelu strukturu, a **t1** i **t2** su varijable **tipa Tocka**. Što je **pt1**?

Uočite da ovdje nismo napisali **ime** strukture iza **struct**, jer ga nećemo ni koristiti.

## Inicijalizacija strukture

Varijablu tipa **struktura** možemo inicijalizirati pri definiciji (kao i svaku drugu varijablu):

---

```
mem_klasa struct ime_var = {v_1, ..., v_n};  
mem_klasa ime_tipa    var = {v_1, ..., v_n};
```

---

- Konstante **v\_1**, **v\_2**, ..., **v\_n** pridružuju se navedenim **redom** odgovarajućim članovima strukture **var** (član po član).

# Inicijalizacija strukture — primjer

Primjer. Ako je definirana **struktura**

---

```
struct racun {  
    int broj_racuna;  
    char ime[80];  
    float stanje;  
};
```

---

onda **varijablu** **kupac** možemo **inicijalizirati** ovako:

---

```
struct racun kupac = { 1234, "Pero Bacilova",  
                      -12345.00 };
```

---

## *Inicijalizacija polja struktura*

Primjer. Slično se može **inicijalizirati** i čitavo **polje struktura**:

```
struct racun kupci[] = { 2234, "Goga",     456.00,
                        1235, "Josip",   -234.00,
                        436,  "Martina", 0.00 };
```

# *Struktura definirana pomoću strukture*

Strukture mogu sadržavati druge strukture kao članove.

Primjer. Pravokutnik paralelan koordinatnim osima možemo zadati parom dijagonalno suprotnih vrhova — na pr. donjim lijevim (pt1) i gornjim desnim (pt2). Vrhovi su točke.

---

```
struct pravokutnik {  
    struct tocka pt1;      /* ili Tocka pt1; */  
    struct tocka pt2;      /* ili Tocka pt2; */  
};
```

---

Deklaracija strukture **tocka** mora prethoditi deklaraciji strukture **pravokutnik**.

U različitim strukturama mogu se koristiti ista imena članova.

# *Rad sa strukturama — pristup članovima*

Primjer. Pristup članovima strukture.

---

```
struct tocka {  
    int x;      /* prvi član strukture */  
    int y;      /* drugi član strukture */  
};  
struct tocka ishodiste;
```

---

Imena objekata i značenje:

- **ishodiste** je **varijabla** tipa **struct tocka**,
- **ishodiste.x** je **prvi** član (ili prva komponenta) varijable **ishodiste**,
- **ishodiste.y** je **drugi** član (ili druga komponenta) varijable **ishodiste**.

# *Pristup članovima strukture (nastavak)*

Primjer. Ako je

---

```
struct racun {  
    int broj_racuna;  
    char ime[80];  
    float stanje;  
} kupac = { 1234, "Pero Bacilova", -12345.00 };
```

---

tada je, redom:

---

```
kupac.broj_racuna = 1234,  
kupac.ime = "Pero Bacilova",  
kupac.stanje = -12345.00.
```

---

# *Operator točka — pristup članu strukture*

Članovima strukture može se individualno pristupiti korištenjem primarnog operatora točka (.).

- Operator točka (.) separira ime varijable i ime člana strukture.

Ako je var varijabla tipa strukture koja sadrži član memb, onda je

---

var.memb

---

član memb u strukturi var.

Napomena. Ime člana je lokalno za svaku strukturu. Zato smije isto ime člana u raznim strukturama.

# *Prioritet operatora točka*

Operator **točka** (.)

- spada u **najvišu** prioritetnu grupu (primarni operatori) i ima asocijativnost  $L \rightarrow D$ .

Zbog **najvišeg** prioriteta vrijedi:

- `++varijabla.clan;  $\iff$  ++(varijabla.clan);`
- `&varijabla.clan;  $\iff$  &(varijabla.clan);`

# *Polje kao član strukture*

Kada struktura sadrži polje kao član strukture, onda se elementima polja (zovimo ga clan) pristupa izrazom:

---

varijabla.clan[izraz]

---

Koristi se asocijativnost  $L \rightarrow D$  za primarne operatore

- točka ( . ) i
- indeksiranje polja ( [] ).

# *Polje kao član strukture — primjer*

Primjer.

---

```
typedef struct {  
    int broj_racuna;  
    char ime[80];  
    float stanje;  
} Racun;  
Racun kupac = { 1234, "Pero Bacilova",  
                -12345.00 };  
  
...  
if (kupac.ime[0] == 'P') puts(kupac.ime);
```

---

# **Polje struktura**

Ako imamo **polje struktura**, onda za pojedini **element** polja, **članu** pripadne strukture pristupamo izrazom

---

**polje[izraz].clan**

---

Asocijativnost je bitna, jer su svi operatori istog prioriteta.

**Primjer.**

---

```
struct tocka {  
    int x;  
    int y;  
} vrhovi[1024] ;  
...  
if vrhovi[17].x == vrhovi[17].y ...
```

---

# **Strukture i funkcije**

Operacije nad strukturuom kao cjelinom su:

- Pridruživanje.
- Uzimanje adrese, primjena **sizeof** operatora.
- Struktura **može** biti argument funkcije. Funkcija tada dobiva kopiju strukture kao argument.
- Funkcija **može** vratiti strukturu.

Primjer. Funkciji **suma** argumenti su 2 strukture tipa **tocka**, a vraća sumu argumenata (tipa **tocka**).

---

```
struct tocka {  
    int x;  
    int y;  
} t, ishodiste = {0, 0};
```

## *Strukture i funkcije (nastavak)*

```
struct tocka suma(struct tocka p1,
                   struct tocka p2) {
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}

...
t = ishodiste /* t i ishodiste moraju
                 biti istog tipa */
printf("%d\n", sizeof(t));
```

---

Napomena: Nije dozvoljeno uspoređivanje struktura.

# *Strukture i funkcije — kompleksni brojevi*

Primjer. Biblioteka funkcija za osnovne operacije s kompleksnim brojevima.

```
typedef struct {
    double re;      /* ili x */
    double im;      /* ili y */
} complex;

double zabs(complex a) /* cabs vec postoji! */
{
    return sqrt( a.re * a.re + a.im * a.im );
}
```

# *Strukture i pokazivači*

Pokazivač na strukturu definira se kao i pokazivač na osnovne tipove varijabli.

```
struct tocka {  
    int x;  
    int y;  
} p1, *pp1;  
  
...  
pp1 = &p1;  
(*pp1).x = 13;  
(*pp1).y = 27;  
*pp1.x = 13; /* GRESKA */  
/* *pp1.x je isto sto i *(pp1.x) */
```

# *Operator strelica (->)*

- Primarni operator strelica ( $\rightarrow$ ) nudi jednostavan način dohvaćanja člana strukture korištenjem pokazivača.
- Asocijativnost operatora  $\rightarrow$  je  $L \rightarrow D$ .

Ako je **ptvar** pokazivač na strukturu, a **clan** jedan član strukture, onda je:

$$\text{ptvar} \rightarrow \text{clan} \iff (*\text{ptvar}).\text{clan}$$

Primjer.

---

```
struct tocka p1, *pp1 = &p1;
pp1->x = 13;
pp1->y = 27;
```

---

## *Složeni izrazi*

```
struct pravokutnik {  
    struct tocka pt1;  
    struct tocka pt2;  
} r, *pr = &r;
```

Sljedeći su izrazi ekvivalentni:

```
r.pt1.x          /* operatori . i -> */  
pr->pt1.x        /* imaju isti prioritet */  
(r.pt1).x        /* i asocijativnost */  
(pr->pt1).x     /* L -> D */
```

## *Složeni izrazi (nastavak)*

Primjer. Što će ispisati sljedeći program?

---

```
struct {  
    int n;  
    char *ch;  
} t[10], *pt = &t[0]; /* globalne var. */  
  
int main(void) {  
    int i;  
    char tmp[10];  
  
    for (i = 0; i < 10; i++) { /* inic. polja */  
        t[i].n = i;  
        t[i].ch = (char *)malloc(10);
```

## *Složeni izrazi (nastavak)*

```
    sprintf(tmp, "%c", 'a' + i);
    strcat(tmp, "++");
    strcpy(t[i].ch, tmp);
}
printf("%s \n", (++pt) -> ch);
printf("%s \n", pt++ -> ch);
/* pt++ -> ch <=> (pt++) -> ch */
/* zbog razlicite asoc. operatora */
printf("%c \n", *pt -> ch++);
printf("%c \n", *pt++ -> ch);
printf("%s \n", pt -> ch);
return 0;
}
```

# *Složeni izrazi (nastavak)*

Ispis programa:

b++

b++

c

+

d++

# Unija

Unija je složeni tip podataka sličan strukturi, jer sadrži

- članove različitog tipa.

Gdje je razlika?

Članovi strukture su

- smješteni u memoriji jedan za drugim.

Za razliku od toga, svi članovi unije

● počinju na istom mjestu u memoriji — na istoj lokaciji, tj. dijeli jedan zajednički dio memorije, ovisno o veličini članova unije.

Ukupna rezervirana memorija za varijablu tipa unije

- bit će dovoljno velika da u nju stane “najveći” član unije.

# *Svrha unije i rad s unijama*

Ideja: taj **zajednički** dio memorije možemo interpretirati

- na **razne** načine — kao vrijednost **različitih** tipova.

Zato i ime — **unija** tipova!

**Napomena.** Osnovna svrha unije **nije**

- **ušteda** memorijskog prostora,  
iako se može koristiti i za to.

Osim navedene **razlike** između **unija** i **struktura** u **rezervaciji** memorije, sve ostalo u **C-u** je potpuno **isto**, samo

- umjesto ključne riječi **struct** za **strukture**,
- pišemo ključnu riječ **union** za **unije**.

# Deklaracija unije

Deklaracija **tipa unije** ima isti oblik kao i za **tip strukture** — umjesto **struct**, pišemo **union**.

---

```
union ime {  
    tip_1 ime_1;  
    ...     ...  
    tip_n ime_n;  
};
```

---

Kao i kod struktura, **bolje** je koristiti **typedef** za deklaraciju tipa unije.

Varijable **x** i **y** tipa ove unije mogu se deklarirati ovako:

---

```
union ime x, y;
```

---

# *Unija (nastavak)*

Primjer.

---

```
union pod {  
    int i;  
    float x;  
} u, *pu;
```

---

Ovdje su:

- `u.i` i `pu->i` — variable tipa `int`.
- `u.x` i `pu->x` — variable tipa `float`.

## *Unija (nastavak)*

**Primjer.** Uniju možemo iskoristiti kako bismo ispisali “binarni” zapis broja s pomičnim zarezom.

---

```
u.x = 0.234375;  
printf("0.234375 binarno = %x\n", u.i);
```

---