Programiranje 1

Osnovni algoritmi na cijelim brojevima

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

11. prosinca 2023.

Cilj predavanja

Cilj: konstrukcija, **implementacija** i **analiza** jednostavnih (osnovnih) **algoritama**.

Struktura algoritama: jedna petlja i nekoliko **uvjetnih** naredbi.

Podaci: najjednostavniji tip podataka - **cijeli** brojevi.

Kasnije ćemo iste ili slične algoritme koristiti i na složenijim podacima: **nizovi** na ulazu, **polja**, **vezane liste** itd.

Algoritme ćemo realizirati kao **cijele** programe ili **odsječke** programa. Kasnije ćemo neke od tih algoritama **realizirati** kao **funkcije**.

Uvod

Ulazni podaci su **nenegativni** cijeli brojevi, tj. brojevi iz skupa $\mathbb{N}_0$ (osim ako specificiramo drugačije).

Za **prikaz** podataka koristimo `unsigned int` ili `int` ako nam raspon nije jako bitan ili koristimo brojeve iz skupa $\mathbb{Z}$.

Sve **algoritme** realiziramo u **cjelobrojnoj** aritmetici (realnu ne koristimo zbog mogućih grešaka zaokruživanja).

Algoritam treba raditi **korektno** za što **veći** skup **ulaznih** podataka, po mogućnosti za **svaki prikazivi** podatak.

Treba paziti na: a) skup **prikazivih** brojeva u računalu je **konačan**, b) **aritmetika** cijelih brojeva je **modularna** aritmetika.

Broj znamenki broja

Zadatak: program treba učitati **cijeli** broj n (tipa `int`) i naći **broj dekadskih znamenki** tog broja.

Koraci najjednostavnijeg algoritma: a) **brisanje** znamenki **straga**, b) **brojanje** obrisanih znamenaka.

Realizacija algoritma:

- **Inicijaliziramo** brojač na 0 (jer još nismo obrisali niti jednu znamenku),
- Znamenke brišemo **dijeljenjem** broja s bazom 10 (`n = n/10` ili `n/=10`),
- gornju naredbu ponavljamo u **petlji**, u svakoj iteraciji povećamo brojač za 1.
- Do kada **ponavljamo** postupak? Dok broj ima **bar jednu** znamenku koju još **nismo** obrisali ($n \neq 0$).

Broj znamenki broja

Primjer: $n = 123$, zadnja znamenka je n mod $10 = 3$.

- `n = n/10` daje $n = 12$, broj obrisanih znamenki 1,
- `n = n/10` daje $n = 1$, broj obrisanih znamenki 2,
- `n = n/10` daje $n = 0$, broj obrisanih znamenki 3.

Petlju možemo realizirati naredbom `while` ili `do-while` pošto **broj zamenki ne znamo unaprijed**. Može se i `for` petljom ali nije toliko **prirodno**.

Dogovorno uzimamo da $n = 0$ ima **nula** znamenki (ima smisla u **normaliziranom** prikazu broja u bazi) stoga koristimo `while` petlju.

Broj znamenki broja

Broj dekadskih znamenki cijelog broja.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5       int n, brZn=0;
6       printf("␣Upisi␣cijeli␣broj␣n:␣");
7       scanf("%d", &n);
8       while (n != 0) {
9            ++brZn;
10           n /= 10; /* Brisi zadnju znamenku. */
11      }
12     printf("␣Broj␣znamenki␣=␣%d\n", brZn);
13 return 0;
14 }
```

Ulaz: 12345, **izlaz:** `Broj znamenki = 5.`

Broj znamenki broja

Glavni dio programa za računanje broja dekadskih znamenki cijelog broja se može realizirati `for` petljom:

```
brZn = 0;
for(;n!=0;n/=10)
        ++brZn;
```

Ili ekvivalentno:

```
for(brZn = 0;n!=0;n/=10)
        ++brZn;
```

- Program radi za **sve** prikazive cijele brojeve,
- na kraju izvršavanja algoritma vrijedi $n = 0$,
- Složenost algoritma je jednaka broju prolaza kroz petlju, a to je **broj znamenaka** od n.

Broj znamenki broja u zadanoj bazi

Identičan algoritam radi korektno i u bilo kojoj drugoj bazi $b \geq 2$. Ako je $n \in \mathbb{N}$, onda prikaz u bazi b ima oblik:

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

Ovaj prikaz je **normaliziran**, tj. za znamenke vrijedi

$$a_0, \ldots, a_k \in \{0, 1, \ldots, b-1\}, \ a_k > 0$$

I dalje smatramo da $n = 0$ ima 0 znamenaka. U nastavku radimo s **nenegativnim** brojevima.

Oznaka za **čitanje** i **pisanje** nenegativnih brojeva tipa `unsigned int` je `%u`. Konstante se pišu sa sufiksom `u` (npr. `0u`).

U programima koji slijede je ispušten sufiks `u` zbog lakše čitljvosti. Konstanta tipa `int` se pri pridruživanju varijabli tipa `unsigned int` **pretvara** u `unsigned int`. Uz to bitni dio svih algoritama radi i u tipu `int`.

Broj znamenki broja u zadanoj bazi

Broj znamenki broja n u bazi b.

```
1  #include <stdio.h>
2  int main(void)
3  {
4      unsigned int b = 10, n, brZn=0;
5      printf(" Upisi nenegativni broj n: ");
6      scanf("%u", &n);
7      printf("\n Broj %u", n);
8      while (n > 0) {
9          ++brZn;
10         n /= b;/* Unistava n dijeljenjem.*/
11      }
12  printf(" ima %u znamenki u bazi %u\n", brZn, b);
13 return 0;
14 }
```

Zadatak: dodajte na kraj programa ispis **završne** vrijednosti varijable n i provjerite da je $n = 0$.

Broj znamenki broja u zadanoj bazi

Ako je $n \in \mathbb{N}$ i $n = a_k b^k + a_{k-1}^{k-1} b^{k-1} + \cdots + a_1 b + a_0$ **normalizirani** prikaz tog broja u bazi b, tj. vrijedi $a_0, \ldots, a_k \in \{0, 1, \ldots, b-1\}$, $a_k > 0$, tada broj znamenki $(k+1)$ možemo izračunati direktno preko **logaritma** $k + 1 = \lfloor log_b n \rfloor + 1$. Takvo računanje zahtjeva **realnu** aritmetiku a ona ima **greške zaokruživanja**.

U zaglavlju `<math.h>` postoje **dvije** funkcije za **logaritam**: `log = ln` i `log10 =` log_{10}. Pošto trebamo $log_b n$, to računamo kao:

$$log_b n = \frac{ln\ n}{ln\ b} = \frac{log_{10} n}{log_{10} b}$$

Najveće cijelo (za nenegativne brojeve) dobivamo **pretvaranjem** tipova **cast** operatorom `(int)` ili `(unsigned int)`.

Napomena: **izračunati** logaritam može imati **malu** grešku (**nadolje**) koja je dovoljna za **pogrešan** rezultat.

Broj znamenki broja u zadanoj bazi

Hoće li zaista doći do greške i za koje brojeve n i baze b ovisi o konkretnoj `C`-biblioteci koja stiže uz prevoditelj.

Na `Intelovom` prevoditelju na Windows operacijskom sustavu (uz Microsoft-ovu biblioteku):

- `log10` radi korektno u bazi 10, ali `log` **ne radi** za $n = 10^6$. $log(1000000)/log(10) = 5.999999999999999$, po tome $n = 10^6$ ima 6 a ne 7 znamenaka.
- `log` radi **korektno** u bazi 2,
- `log` **ne radi** u bazi 3, već za $n = 3^5 = 243$ $log(243)/log(3) = 4.999999999999999$, po tome $n = 3^5$ ima 5 a ne 6 znamenaka u bazi 3.

Broj znamenki broja u zadanoj bazi

Umjesto **dijeljenjem** broja s bazom, broj znamenki broja u bazi možemo dobiti i **potenciranjem** baze dok određeni uvjet nije zadovoljen.

Koji uvjet koristiti?
Iteriramo dok je **kvocijent** $n/potencija$ **veći** ili **jednak** od **baze**.

Uvjet n **veći li jednak** od potencije baze treba upotrebljavati **pažljivo**. Preporučljivo koristiti **prijašnji** uvjet (kvocijent).

Zadatak: Napišite program koji odgovara opisanom algoritmu traženja broja znamenki i pažljivo ga testirajte. Dodajte mu **pisanje** znamenki **sprijeda**.

Obrada znamenki broja

U nastavku slijedi niz varijacija na temu *obrade* znamenki broja u zadanoj bazi.

Ukoliko redoslijed obrade, odnosno poredak znamenki **nije** bitan, obrada se može odvijati kao i kod **brojanja** znamenki:

- **inicijaliziramo** rezultat (na odgovarajući način, ovisno o problemu),
- **izdvojimo zadnju** znamenku (broj % baza),
- **obradi** zadnju znamenku (ovisno o problemu),
- **obriši** zadnju znamenku (kao kod brojanja).

Suma znamenki broja

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i izračunati **zbroj znamenki** tog broja u zadanoj bazi $b = 10$.

Traženi rezultat je: $a_0 + a_1 + \cdots + a_k$.

Algoritamski: `rezultat = rezultat+`a_i, $i = 0, 1, \ldots, k$.

Inicijalizacija za zbrajanje je **neutral** za zbrajanje, `rezultat = 0`. Tako ćemo označavati i sumu **praznog** skupa.

```
1 printf("\n n = %u\n", n);
2 suma = 0;
3 while (n > 0) {
4     suma += n % b;
5     n /= b;
6 }
7 printf("Suma znamenki u bazi %u je %u\n",b,suma);
```

Umnožak znamenki broja

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i izračunati **umnožak znamenki** tog broja u zadanoj bazi $b = 10$.

Traženi rezultat je: $a_0 \cdot a_1 \cdot \dots \cdot a_k$.

Algoritamski: `rezultat = rezultat`$\cdot a_i$, $i = 0, 1, \dots, k$.

Inicijalizacija za množenje je **neutral** za množenje, `rezultat = 1`. Tako ćemo označavati i umnožak **praznog** skupa.

```
printf("\n n = %u\n", n);
p = 1;
while (n > 0) {
    p *= n % b;
    n /= b;
}
printf("Produkt znamenki u bazi %u je %u\n",b,p);
```

Najveća znamenka broja

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i izračunati **najveću znamenku** tog broja u zadanoj bazi $b = 10$. Traženi rezultat je: $max\{a_0, a_1, \ldots, a_k\}$.

Algoritamski: `rezultat = max{rezultat,` a_i`}`, $i = 1, \ldots, k$.

Inicijalizacija za maksimum je maksimum jednočlanog skupa $\{a_0\}$, `rezultat =` a_0.

Maksimum **nema neutral**, odnosno, maksimum **praznog skupa nije definiran**! Zato krećemo od $i = 1$, a ne od nule.

Najveća znamenka broja

```
1  if (n > 0) {
2          maxZn = n % b; /* Zadnja znamenka. */
3          n /= b;
4          while (n > 0) {
5              znam = n % b;
6              if (znam > maxZn) maxZn = znam;
7              n /= b;
8             }
9          printf("␣Najveca␣znamenka␣u␣bazi␣%u␣je"
10         "␣%u\n", b, maxZn);
11 }
12 else
13         printf("␣Nema␣znamenki\n");
```

Najveća znamenka broja

Način rješavanja koji treba **izbjegavati** jer je potrebno znati i moći reprezentirati najmanju moguću vrijednost skupa.

```
/* Najveca znamenka broja n u bazi b.
"Lazna" inicijalizacija na -1
ne moze u tipu unsigned, pa koristimo 0.
Unistava n dijeljenjem.
*/
maxZn = 0;
while (n > 0) {
    znam = n % b;
    if (znam > maxZn) maxZn = znam;
    n /= b;
}
printf(" Najveca znamenka u bazi %u je %u\n",
b, maxZn);
```

Najmanja znamenka broja

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i izračunati **najmanju znamenku** tog broja u zadanoj bazi $b = 10$. Traženi rezultat je: $min\{a_0, a_1, \ldots, a_k\}$.

Algoritamski: `rezultat = min{rezultat,` a_i`}`, $i = 1, \ldots, k$.

Inicijalizacija za minimum je minimum jednočlanog skupa $\{a_0\}$, `rezultat =` a_0.

Minimum **nema neutral**, odnosno, minimum **praznog skupa nije definiran**! Zato krećemo od $i = 1$, a ne od nule.

Najmanja znamenka broja

```
1  if (n > 0) {
2          minZn = n % b; /* Zadnja znamenka. */
3          n /= b;
4          while (n > 0) {
5              znam = n % b;
6              if (znam < minZn) minZn = znam;
7              n /= b;
8             }
9          printf(" Najmanja znamenka u bazi %u je"
10         " %u\n", b, minZn);
11 }
12 else
13         printf(" Nema znamenki\n");
```

Najmanja znamenka broja

Način rješavanja koji treba **izbjegavati** jer je potrebno inicijalizirati na vrijednost veću od svih unutar skupa (ili najveću). Pripaziti na slučaj $n = 0$.

```
/* Najmanja znamenka broja n u bazi b.
"Lazna" inicijalizacija na b.
Unistava n dijeljenjem.
*/
minZn = b;
while (n > 0) {
    znam = n % b;
    if (znam < minZn) minZn = znam;
    n /= b;
}
printf("␣Najmanja␣znamenka␣u␣bazi␣%u␣je␣%u\n",
b, minZn);
```

Provjere znamenki broja

Najjednostavnije **provjere** odgovaraju standardnim **kvantifikatorima** u matematici:

- **Postoji** ($\exists$) li objekt sa zadanim svojstvom?
- Ima li **svaki** ($\forall$) objekt zadano svojstvo?

Rezultat je **odgovor** na postavljeno pitanje, tj. rezultat ima **logički** tip (**DA/NE**, **istina/laž**, 1/0).

Postoji znamenka (svojstvo)

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i odgovoriti na pitanje: postoji li **znamenka** tog broja koja je jednaka 5 (u zadanoj bazi $b = 10$).

Traženi rezultat je: $(a_0 = 5) \ \vee \ (a_1 = 5) \ \vee \ \ldots \ \vee \ (a_k = 5)$.

Algoritamski:
`rezultat = rezultat` $\|$ $(a_i == 5)$, $i = 0, 1, \ldots, k$.

Inicijalizacija za postoji je **prazan** skup. **Inicijalizacija** za disjunkciju je **neutral** za disjunkciju: `rezultat = 0` (**laž**).

Postoji znamenka (svojstvo)

```
odgovor = 0; /* NE, laz. */
while (n > 0) {
      znam = n % b;
      odgovor = odgovor || (znam == trazena);
      n /= b;
}
if (odgovor) printf("␣Odgovor␣je␣DA\n");
else printf("␣Odgovor␣je␣NE\n");
```

```
odgovor = 0; /* NE, laz. */
while (n > 0) { //skracena varijanta
      znam = n % b;
if (znam == trazena) { //prekida petlju cim
     odgovor = 1;        //sazna odgovor
     break; }
    n /= b; }
```

Svaka znamenka (svojstvo)

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i odgovoriti na pitanje: je li svaka **znamenka** tog broja jednaka 5 (u zadanoj bazi $b = 10$).

Traženi rezultat je: $(a_0 = 5) \ \wedge \ (a_1 = 5) \ \wedge \ \ldots \ \wedge \ (a_k = 5)$.

Algoritamski:
`rezultat = rezultat &&` $(a_i == 5)$, $i = 0, 1, \ldots, k$.

Inicijalizacija za svaki je **prazan** skup. **Inicijalizacija** za konjunkciju je **neutral** za konjunkciju: `rezultat = 1` (**istina**).

Svaka znamenka (svojstvo)

```
odgovor = 1; /* DA, istina. */
while (n > 0) {
      znam = n % b;
      odgovor = odgovor && (znam == trazena);
      n /= b;
}
if (odgovor) printf("␣Odgovor␣je␣DA\n");
else printf("␣Odgovor␣je␣NE\n");
```

```
odgovor = 1; /* DA, istina. */
while (n > 0) {
     znam = n % b;
if (znam != trazena) {
    odgovor = 0;
    break;}
n /= b;}
```

Palindrom

Zadatak: program treba učitati **nenegativni** cijeli broj n (tipa `unsigned int`) i naći odgovor na pitanje: je li broj n **palindrom** (u zadanoj bazi $b = 10$), tj. čita li se n jednako s obije strane?

Primjer: 14741 **je** palindrom dok 14743 **nije**.

Ideja: umjesto provjere odgovarajućih znamenki, `prva == zadnja`, `druga == predzadnja` itd. napravimo broj s **obratnim** poretkom znamenki i usporedimo ga s **polaznim** brojem.

```
1 #include <stdio.h>
2 /* Provjera je li prirodni broj palindrom. */
3 int main(void)
4 {
5 unsigned int b = 10;
6 unsigned int n, m1, m2, palindrom;
7 printf(" Upisi nenegativni broj n: ");
```

Palindrom

```
scanf("%u", &n);
printf("␣Broj␣=␣%u\n", n)
m1 = n;
m2 = 0;
while (n > 0) {
      m2 = m2 * b + n % b;
      n /= b;
}
palindrom = m1 == m2 ? 1 : 0;
printf("␣Palindrom␣=␣%u\n", palindrom);
return 0;
}
```

Palindrom

Primjer: na **početku** je $m_1 = n = 14743$, $m_2 = 0$.

n	$m_2 \cdot b$	$n\%b$	m_2^{novi}	n^{novi}
14743	$0 \cdot 10$	$+3$	3	1474
1474	$3 \cdot 10$	$+4$	34	147
147	$34 \cdot 10$	$+7$	347	14
14	$347 \cdot 10$	$+4$	3474	1
1	$3474 \cdot 10$	$+1$	34741	0

Rezultat je $m_2 = 34741 \neq m_1 = 14743 \Rightarrow n$ nije palindrom.

Pitanje: radi li ovaj program **korektno** za **svaki** prikazivi ulaz?

- Je li **obratni** broj uvijek prikaziv?
- Možemo li **zato** dobiti **pogrešan** odgovor u bazi $b = 10$? (Odgovor je: NE. Dokažite!)

Probajte isto i za druge baze.

Palindrom

```
pot = 1; //Potencija baze uz najvisu znamenku.
while (n / pot >= b)
      pot *= b;
/* Moze i ovako, praznom for naredbom:
for (pot = 1; n / pot >= b; pot *= b);*/
palindrom = 1; /* DA, istina. */
/* Petlja za provjeru para znamenki. */
while (n >= b) { /* n ima bar dvije znam. */
      znam1 = n / pot; /* Prva znamenka. */
      znam2 = n % b; /* Zadnja znamenka. */
      if (znam1 != znam2) {
            palindrom = 0;
            break;}
n %= pot; /* Brisi prvu znamenku. */
n /= b; /* Brisi zadnju znamenku. */
pot = pot /b/b; /* Podijeli pot s b^2. */
}
```

Najveća zajednička mjera

Zadatak: program treba učitati dva **cijela** broja a i $b \neq 0$ te izračunati **najveću** zajedničku mjeru $M(a, b)$ **cijelih** brojeva a i b.

Algoritam se bazira na Euklidovom teoremu o dijeljenju:
$a = q \cdot b + r$ za neki $q \in \mathbb{Z}$, gdje je r **ostatak**, $0 < r < |b|$.

Ključni koraci:

- Ako $d|a$ i $d|b$, onda $d|r$, pa je $M(a, b) = M(b, r)$ (**smanjujemo** argumente po apsolutnoj vrijednosti).
- Ako je $r = 0$, tada $a = q \cdot b$ pa je $M(a, b) = b$ (**kraj**).

Test primjeri: $a = 48,\ b = 36,\ a = 21,\ b = 13$ (probajte i za negativne brojeve).

Najveća zajednička mjera

```
int a, b, ostatak, mjera;
...
while (1) {
   ostatak = a % b;
   if (ostatak == 0) {
     mjera = b;
     break;
    }
a = b;
b = ostatak;
}
```

Ovaj algoritam **radi** i za **negativne** brojeve a i b. U tom slučaju **mjera** može biti **negativna**.

Kod skraćivanja racionalnog broja a/b, zadanog brojnikom a i nazivnikom b, korisno je tražiti da je $M(a, b) > 0$.

Najveća zajednička mjera

Jedna mogućnost je da izračunamo $M(|a|, |b|)$. *C* funkcije za računanje **apsolutne** vrijednosti se zovu `abs` (za `int`) i `labs` (za `long int`), a deklarirane su u zaglavlju `<stdlib.h>`. U tom slučaju, na **početku** algoritma treba dodati `a = abs(a); b = abs(b);`

Jednostavnija opcija je vratiti $|M(a, b)|$. To radimo naredbom `mjera = abs(b)` umjesto `mjera = b`.

```
int a, b, ostatak, mjera;
...
while (b != 0) { /* Ne: b > 0. */
    ostatak = a % b;
    a = b;
    b = ostatak;
}
mjera = a; /* mjera = abs(a); za M(a,b)>0 */
```

Najveća zajednička mjera

Sporiji algoritam koristi samo **oduzimanje** (bez računanja ostatka).

```
1  int a, b, mjera;
2  ...
3  while (a != b)
4      if (a > b)
5          a -= b;
6      else
7          b -= a;
8  mjera = a; /* Moze i b. */
```

Ovaj algoritam radi samo za **prirodne** brojeve a i b. Zato je na početku dobro dodati `a = abs(a); b = abs(b);`

Potencija broja 2

Zadatak: program treba učitati **prirodni** broj n (tipa `unsigned int`) i naći odgovor na pitanje: je li broj n **potencija** broja $d = 2$, tj. može li se n prikazati kao d^k, gdje $k > 0$.

Za zadani faktor $d \geq 2$, **svaki** prirodni broj $n \in \mathbb{N}$ možemo **jednoznačno** prikazati u obliku:

$$n = d^k \cdot m, \;\; m \; mod \; d \neq 0$$

, tj. d **ne dijeli** m, gdje je $k \geq 0$ cijeli broj. **Dokažite** navedenu tvrdnju (slično rastavu na proste faktore, samo d **ne mora** biti **prost**).

Zadatak je pronaći k i m u tom rastavu.

Ideja rješenja:

- Sve dok je n **djeljiv** s faktorom d, **podijelimo** ga s d.
- **Broj** ovih dijeljenja je eksponent $k \geq 0$.
- Na **kraju** tog postupka ostaje nam m.

Potencija broja 2

n je potencija broja d ako i samo ako na kraju izvršavanja algoritma: $k > 0$ i $m = 1$.

Rezultat dijeljenja broja n s d spremamo u istu varijablu n. Zato je konačna vrijednost od $n = m$.

```
unsigned int n, d = 2, k, odgovor;
k = 0;
/* Sve dok je n djeljiv s d,
podijeli ga s d. */
while (n % d == 0) {
        ++k;
        n /= d;
}
/* Mora ostati n == 1. */
odgovor = n == 1 && k > 0;
```

Prikaz cijelog broja u računalu

Zadatak: program treba učitati **cijeli** broj n (tipa `int`) i napisati **prikaz** tog broja u računalu (kao niz **bitova**).

Broj bitova u prikazu **možemo** izračunati koristeći `sizeof` operator. Zato koristimo `for` petlju.

```
1  #include <stdio.h>
2  /* Prikaz cijelog broja u racunalu. */
3  int main(void)
4  {
5  int nbits, broj, bit, i;
6  unsigned mask;
7  /* Broj bitova u tipu int. */
8  nbits = 8 * sizeof(int);
9  /* Pocetna maska ima bit 1
10 na najznacajnijem mjestu. */
11 mask = 0x1 << nbits - 1;
```

Prikaz cijelog broja u računalu

```
1  printf("␣Upisi␣cijeli␣broj:␣");
2  scanf("%d", &broj);
3  printf("␣Prikaz␣broja␣%d:\n␣", broj);
4  for (i = 1; i <= nbits; ++i) {
5  /* Maskiranje odgovarajuceg bita. */
6  bit = broj & mask ? 1 : 0;
7  /* Ispis i blank nakon svaka 4 bita,
8  osim zadnjeg. */
9  printf("%d", bit);
10 if (i % 4 == 0 && i < nbits) printf("␣");
11 /* Pomak maske za 1 bit udesno. */
12 mask >>= 1;
13 }
14 printf("\n");
15 return 0;
16 }
```

Prikaz cijelog broja u računalu

Za ulaz 3 dobivamo:

```
Prikaz broja 3:
0000 0000 0000 0000 0000 0000 0000 0011
```

Za ulaz -3 dobivamo:

```
Prikaz broja -3:
1111 1111 1111 1111 1111 1111 1111 1101
```