

# *Programiranje 1*

## *9. predavanje*

Saša Singer

`singer@math.hr`

`web.math.pmf.unizg.hr/~singer`

PMF – Matematički odsjek, Zagreb

# Sadržaj predavanja

- Osnovni algoritmi na cijelim brojevima:
  - Uvod — što se hoće.
  - Broj znamenki cijelog broja.
  - Zbroj (suma) i umnožak (produkt) znamenki broja.
  - Najveća (najmanja) znamenka broja.
  - Provjere znamenki broja (postoji, svaka).
  - Palindrom.
  - Najveća zajednička mjera — Euklidov algoritam.
  - Potencija broja 2.
  - Binarni prikaz cijelog broja u računalu.

# Informacije — Praktični kolokvij

Praktični kolokvij (prvi krug) kreće krajem 9. tjedna nastave,

🕒 subota, 12. 12. — subota, 19. 12.

Razlog: da stignemo napraviti funkcije i srediti termine.

Termini će biti poznati do kraja 8. tjedna nastave.

Prijava za PK (zauzimanje termina) je onaj tjedan iza,

🕒 u 9. tjednu nastave, 7. 12. — 11. 12.,

🕒 preko aplikacije za zadaće!

Zapamtite: Vrijeme za rješenje je 45 minuta.

🕒 Zadaci su objavljeni na webu — pod Materijali, 3. točka (to su isti zadaci kao prošle godine, nema promjena).

🕒 Korisno je odmah pogledati i početi vježbati.

# Praktični kolokvij — upute

Barem dan–dva prije praktičnog kolokvija provjerite:

- da vam account “štima”,
- da se znate “logirati”,
- da znate naći IDE u kojem mislite raditi.

Uz svaki ponuđeni termin piše i praktikum u kojem se radi,

- tj. sami birate termin i praktikum.

Dakle, imate dovoljno vremena za isprobati (na licu mjesta) “je l’ vam sve radi”.

Ako vam account ne radi na praktičnom kolokviju, ili si ne znate password, ili ne znate naći/koristiti compiler, ...

- to je vaš problem. Što se nas tiče — to je pad.

# Informacije — 1. kolokvij i prijave

Službeni termin prvog kolokvija je:

- Prvi kolokvij: petak, 27. 11. 2015., u 15 sati.

Vezano uz prijave za zadaće:

- trenutni broj uspješno prijavljenih studenata je sasvim zadovoljavajući — oko 271, od očekivanih 298.

Nažalost, fali preko 25 studenata — ali to nije naš problem!

Lijepo molim, “ne šalite” se, uspješna prijava je

- nužan preduvjet za izlazak na kolokvij.

To pravilo se ne mijenja!

- Rok za prijavu je 7 dana = 168 sati prije kolokvija.

# Informacije — računi i prijava za zadaće

Ne zaboravite da treba:

- preuzeti korisnički račun u Računskom centru.
- Računi se “preuzimaju” svaki dan, od 12:30 do 14:30 sati.

Na papiru kojeg dobijete, piše i vaše korisničko ime.

Promijenite password (kako — piše pri dnu papira)!

Nadalje, treba:

- obaviti prijavu i, zatim, potvrditi prijavu u aplikaciji za domaće zadaće (“ku”), na internetskoj adresi

<http://degiorgi.math.hr/prog1/ku/>

## Informacije — korektna prijava, hrvatski znakovi

Kod **prve** prijave u aplikaciju, treba **popuniti 6** polja:

- **dva gore** = JMBAG (10 znamenki), lozinka,
- i još **četiri malo niže** = potvrda lozinke, ime, prezime, korisničko ime = **ono s papira** (bitno za potvrdu).

Čim **kliknete** na neko polje — **prije** no što išta stignete,

- uredno vam se pokaže **uputa što treba upisati**.

Zato, pažljivo **čitajte upute** — **prije popunjavanja** i slanja!

**Bitno:** Prilikom **prijave** u aplikaciju za “ku”,

- svoje **podatke** trebate upisati **korektno** — što znači i
- korištenje **hrvatskih** znakova u **imenu i prezimenu!**

# Informacije — potvrda prijave, ispravci

Ako je **taj** dio **uredno** prošao, nakon kraćeg vremena,

- trebate dobiti **e-mail** na vašu adresu na **studentu**,
- u kojem piše **kako potvrditi** prijavu.

Kad to **uspješno** napravite, tek onda je prijava **gotova**.

Studenti koji su upisali “**csz dj**” varijantu imena i prezimena, ili imaju **problema** s **potvrdom** prijave

- neka se jave **e-mailom** (sa **studenta**) **meni** na adresu

**singer@math.hr**

i napišu

- svoj **JMBAG** i **ispravno** ime i prezime.



# Osnovni algoritmi na cijelim brojevima

# Sadržaj predavanja

- Osnovni algoritmi na cijelim brojevima:
  - Uvod — što se hoće.
  - Broj znamenki cijelog broja.
  - Zbroj (suma) i umnožak (produkt) znamenki broja.
  - Najveća (najmanja) znamenka broja.
  - Provjere znamenki broja (postoji, svaka).
  - Palindrom.
  - Najveća zajednička mjera — Euklidov algoritam.
  - Potencija broja 2.
  - Binarni prikaz cijelog broja u računalu.

# Uvod — što je cilj?

Cilj je, zapravo, vrlo jednostavan:

- konstrukcija, implemenatcija i analiza jednostavnih (osnovnih) algoritama,
- sastavljenih od jedne petlje i nekoliko uvjetnih naredbi,
- na najjednostavnijim podacima — cijelim brojevima.

Kasnije ćemo iste ili slične algoritme koristiti na složenijim podacima:

- nizovi na ulazu, polja, vezane liste i sl.

Danas ćemo pisati cijele programe ili odsječke programa. Kad napravimo funkcije, onda ćemo

- neke od tih algoritama realizirati kao funkcije.

# Osnovne pretpostavke i dogovori

Ulazni podaci su:

- nenegativni cijeli brojevi, tj. brojevi iz skupa  $\mathbb{N}_0$ , osim ako nije drugačije rečeno.

Za prikaz podataka standardno koristimo

- tip `unsigned int`.

Može i “obični” `int`, ako nam raspon prikazivih brojeva nije jako bitan.

Katkad ćemo dozvoliti

- i negativne cijele brojeve, tj. brojeve iz skupa  $\mathbb{Z}$ .

Tada za prikaz koristimo tip `int`.

# Osnovne pretpostavke i dogovori (nastavak)

**Dogovor.** Sve algoritme realiziramo u cjelobrojnoj aritmetici. Realnu aritmetiku izbjegavamo zbog mogućih grešaka zaokruživanja.

**Oprez:** Neovisno o tipu kojeg koristimo za prikaz brojeva,

- skup prikazivih brojeva u računalu je konačan,
- a aritmetika cijelih brojeva je modularna aritmetika!

Na to treba paziti kod konstrukcije i izbora algoritma. Jedan od bitnih ciljeva je:

- algoritam treba raditi korektno za što “veći” skup ulaznih podataka.
- Po mogućnosti — za svaki prikazivi ulazni podatak!

## Broj znamenki broja

Primjer. Program treba učitati cijeli broj  $n$  (tipa `int`) i naći broj dekadskih znamenki tog broja.

Najlakši algoritam dobivamo jednostavnim

- “brisanjem” znamenki — i to “straga” (lakše je).

Usput, treba samo

- brojati obrisane znamenke!

Uzmimo da je  $n = 123$ . Zadnja znamenka je  $n \bmod 10 = 3$ .

Međutim, sama znamenka nam *ne treba*. Kako ćemo “obrisati” tu znamenku?

- Tako da broj *podijelimo* s bazom 10.

- Odgovarajuća naredba je:  $n = n / 10$  ili  $n /= 10$ .

## Broj znamenki broja (nastavak)

Ovo ponavljamo u **petlji**, s tim da

- svaki puta **povećamo** broj obrisanih znamenki za **1**.

Na **početku**, brojač **inicijaliziramo** na **0** — jer još nismo obrisali **niti jednu** znamenku!

Zadnje pitanje je “**kontrola**” petlje — **do kada** ponavljamo ovaj postupak?

- Sve dok broj ima **bar jednu** znamenku, koju još **nismo** obrisali.

A kad je to? **Sve dok** je  **$n \neq 0$** .

Drugim riječima,

- ponavljanje **prekidamo** kad obrišemo **sve** znamenke, tj. kad **n** postane **nula**.

## Broj znamenki broja (nastavak)

U našem primjeru, za  $n = 123$ , imamo redom:

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

Dakle, sve radi korektno!

Izbor naredbe za realizaciju petlje u programu:

- Broj znamenki **ne** znamo unaprijed (baš to tražimo), pa je **prirodno** koristiti **while** ili **do-while**.

Uz malo više iskustva u C-u, vidjet ćete da može i **for**.

Dogovorno uzimamo da  $n = 0$  ima **nula** znamenki! To ima smisla u **normaliziranom** prikazu broja u bazi.

Onda **koristimo while** petlju, a ne **do-while**.



## Broj znamenki broja (nastavak)

```
#include <stdio.h>

/* Broj dekadskih znamenki cijelog broja. */

int main(void)
{
    int n, broj_znam;

    printf(" Upisi cijeli broj n: ");
    scanf("%d", &n);
```

## Broj znamenki broja (nastavak)

```
broj_znam = 0;
while (n != 0) {
    ++broj_znam;
    n /= 10; /* brisi zadnju znamenku. */
}

printf(" Broj znamenki = %d\n", broj_znam);

return 0;
}
```

---

Za ulaz 12345, program ispisuje

---

Broj znamenki = 5

---

## Broj znamenki broja (nastavak)

Realizacija ključnog dijela programa `for` petljom, bazirana na vezi između `for` i `while` petlji — tipična za `C`:

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

Uočite da “inicijalizacije” **nema**, a “pomak” je upravo **brisanje** znamenki!

Može i ovako — **kratko**, ali nije baš lako za **pročitati**:

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

# Broj znamenki broja (nastavak)

Nekoliko pitanja.

- Za koje cijele brojeve  $n$  program radi korektno?
  - Odgovor: za sve prikazive, uključivo i negativne!
- Kolika je vrijednost broja  $n$  na kraju — nakon završetka algoritma?
  - Odgovor:  $n = 0$ .

Dakle, algoritam je “destruktivan” — uništava ulazni broj  $n$ . Ako to nećemo, treba napraviti kopiju od  $n$  u pomoćnu varijablu, na pr. `temp_n`, i nju “uništiti”.

- Kolika je složenost ovog algoritma?
  - Odgovor: Broj prolaza kroz petlju je upravo broj znamenki broja  $n$ .

# Broj znamenki broja u zadanoj bazi

Zanimljivo je da **isti** 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

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

s tim da je ovaj prikaz **normaliziran**, tj. za znamenke vrijedi

$$a_0, \dots, a_k \in \{0, 1, \dots, b-1\} \quad \text{i} \quad a_k > 0.$$

Dogovorno smatramo da  $n = 0$  **nema** znamenki!

U nastavku prelazimo na **nenegativne** brojeve, da nas **predznak** “ne smeta”.

# Broj znamenki broja u zadanoj bazi (nastavak)

Napomena.

- Oznaka konverzije za čitanje i pisanje nenegativnih brojeva tipa `unsigned int` je `%u`.
- Konstante se pišu s nastavkom (sufiksom) `u` — poput `0u`.

U algoritmima i programima koji slijede,

- namjerno je ispušten `u`, da se lakše čita.

Naime, bitni dio svih algoritama radi i u tipu `int`.

# Broj znamenki broja u zadanoj bazi (nastavak)

```
#include <stdio.h>

/* Broj znamenki broja n u bazi b.
   Unistava n dijeljenjem.
*/

int main(void)
{
    unsigned int b = 10, n, broj_znam;

    printf(" Upisi nenegativni broj n: ");
    scanf("%u", &n);
    printf("\n Broj %u", n);
}
```

## Broj znamenki broja u zadanoj bazi (nastavak)

```
broj_znam = 0;
while (n > 0) {
    ++broj_znam;
    n /= b;
}

printf(" ima %u znamenki u bazi %u\n",
       broj_znam, b);

return 0;
}
```

---

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



# Broj znamenki broja u zadanoj bazi — logaritam

Ako je  $n \in \mathbb{N}$  i ako je

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

normalizirani prikaz tog broja u bazi  $b$ , tj. vrijedi

$$a_0, \dots, a_k \in \{0, 1, \dots, b-1\} \quad \text{i} \quad a_k > 0,$$

onda broj znamenki  $= k + 1$  možemo izračunati i direktno — preko logaritma

$$k + 1 = \lfloor \log_b n \rfloor + 1.$$

Međutim, to zahtijeva realnu aritmetiku, a ona ima greške zaokruživanja.

# Broj znamenki broja — logaritam (nastavak)

U zaglavlju `<math.h>` postoje dvije funkcije za logaritam:

- `log` = `ln`,
- `log10` = `log10`.

Nama treba logaritam u bazi  $b$ . To dobijemo ovako, uz  $n > 0$ :

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

“Najveće cijelo” (za nenegativne brojeve) možemo dobiti pretvaranjem tipova:

- `cast` operatorom (`int`) ili (`unsigned int`).

**Oprez:** izračunati logaritam može imati malu grešku (nadolje) — koja je dovoljna za pogrešan rezultat!

## Broj znamenki broja — logaritam (nastavak)

Hoće li **zaista** doći do **greške** i za **koje** brojeve  $n$  i baze  $b$ ,

- ovisi o konkretnoj C–biblioteci koja stiže uz prevoditelj.

Na primjer, na **Intelovom** prevoditelju na Windowsima (biblioteka je **Microsoftova**):

- $\log_{10}$  radi **korektno** u bazi 10, ali **log ne radi** za  $n = 10^6$ :

- $\log(1000000) / \log(10) = 5.999999999999999999$ ,

pa izlazi da  $n = 10^6$  ima **6**, a ne **7** znamenki!

- $\log$  radi **korektno** u bazi 2,

- **log ne radi** u bazi 3, već za  $n = 3^5 = 243$ :

- $\log(243) / \log(3) = 4.999999999999999999$ .

Izlazi da  $n = 3^5$  ima **5**, a ne **6** znamenki u bazi 3!

# Broj znamenki broja — potenciranje baze

Umjesto “destruktivnog” brisanja znamenki *straga*, što je

- *dijeljenje* broja s bazom (odnosno, potencijom baze), može i ovako:

- *potenciraj bazu* (množenje potencije bazom), sve dok ...

**Oprez** s kriterijem *ponavljanja* ili *zaustavljanja*, ako želimo da algoritam radi za *sve* prikazive brojeve!

- **Ne valja:** ... sve dok je broj *n* *veći ili jednak* od *potencije* baze — na kraju je *manji*!

- **Bolje je:** ... sve dok je kvocijent *n/potencija* *veći ili jednak* od *baze*.

**Zadatak.** Napišite program koji odgovara ovom algoritmu i pažljivo ga testirajte! Dodajte mu *pisanje* znamenki *sprijeda*.

## Obrada znamenki broja — općenito

U nastavku ide hrpa varijacija na temu “obrade” znamenki broja u zadanoj bazi.

Ako redoslijed obrade (poredak znamenki) **nije** bitan, onda obrada može ići na **isti** način kao i **brojanje** znamenki:

- odgovarajuća **inicijalizacija** rezultata;
- **petlja** za obradu znamenki — sve dok “ima znamenki”
  - **izdvoji zadnju** znamenku (tj., straga) = modulo baza;
  - **obradi** ju;
  - **obriši** ju (kao kod brojanja).

I to je to!

# Zbroj ili suma znamenki broja

Primjer. Program treba učitati **nenegativni** cijeli broj **n** (tipa `unsigned int`) i naći

• **zbroj** znamenki tog broja u zadanoj bazi  $b = 10$ .

Traženi rezultat je

$$a_0 + a_1 + \dots + a_k.$$

Algoritamski:

$$\text{rezultat} = \text{rezultat} + a_i, \quad i = 0, 1, \dots, k.$$

**Inicijalizacija** za zbrajanje?

• **Neutral** za zbrajanje: `rezultat = 0`.

Ovo je i **dogovor** za sumu **praznog** skupa!

## Zbroj ili suma znamenki broja (nastavak)

Bitni odsječak programa izgleda ovako:

---

```
printf("\n n = %u\n", n);
```

```
suma = 0;
while (n > 0) {
    suma += n % b;
    n /= b;
}
```

```
printf(" Suma znamenki u bazi %u je %u\n",
      b, suma);
```

---

# Umnožak ili produkt znamenki broja

Primjer. Program treba učitati **nenegativni** cijeli broj **n** (tipa `unsigned int`) i naći

• **produkt** znamenki tog broja u zadanoj bazi  $b = 10$ .

Traženi rezultat je

$$a_0 \cdot a_1 \cdots a_k.$$

Algoritamski:

$$\text{rezultat} = \text{rezultat} \cdot a_i, \quad i = 0, 1, \dots, k.$$

**Inicijalizacija** za množenje?

• **Neutral** za množenje: `rezultat = 1`.

Ovo je i **dogovor** za produkt **praznog** skupa!



## Umnožak ili produkt znamenki broja (nastavak)

Bitni odsječak programa izgleda ovako:

---

```
printf("\n n = %u\n", n);
```

```
prod = 1;
while (n > 0) {
    prod *= n % b;
    n /= b;
}
```

```
printf(" Produkt znamenki u bazi %u je %u\n",
      b, prod);
```

---

# Najveća znamenka broja

Primjer. Program treba učitati **nenegativni** cijeli broj **n** (tipa `unsigned int`) i naći

• **najveću znamenku** tog broja u zadanoj bazi  $b = 10$ .

Traženi rezultat je

$$\max\{a_0, a_1, \dots, a_k\}.$$

Algoritamski:

$$\text{rezultat} = \max\{\text{rezultat}, a_i\}, \quad i = 1, 2, \dots, k.$$

**Inicijalizacija** za maksimum?

• Maksimum **jednočlanog** skupa  $\{a_0\}$ : **rezultat** =  $a_0$ .

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

## Najveća znamenka broja (nastavak)

```
if (n > 0) {
    max_znam = n % b; /* zadnja znamenka */
    n /= b;
    while (n > 0) {
        znam = n % b;
        if (znam > max_znam) max_znam = znam;
        n /= b;
    }
    printf(" Najveca znamenka u bazi %u je"
           " %u\n", b, max_znam);
}
else
    printf(" Nema znamenki\n");
```

## Najveća znamenka broja — Izbjegavati!

```
/* Najveća znamenka broja n u bazi b.
   ‘‘Lazna’’ inicijalizacija na -1
   ne može u tipu unsigned, pa stavim 0.
   Unistava n dijeljenjem.
*/
max_znam = 0;
while (n > 0) {
    znam = n % b;
    if (znam > max_znam) max_znam = znam;
    n /= b;
}
printf(" Najveća znamenka u bazi %u je %u\n",
       b, max_znam);
```

# Najmanja znamenka broja

Primjer. Program treba učitati **nenegativni** cijeli broj **n** (tipa `unsigned int`) i naći

• **najmanju znamenku** tog broja u zadanoj bazi  $b = 10$ .

Traženi rezultat je

$$\min\{a_0, a_1, \dots, a_k\}.$$

Algoritamski:

$$\text{rezultat} = \min\{\text{rezultat}, a_i\}, \quad i = 1, 2, \dots, k.$$

**Inicijalizacija** za minimum?

• Minimum **jednočlanog** skupa  $\{a_0\}$ : **rezultat** =  $a_0$ .

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

## Najmanja znamenka broja (nastavak)

```
if (n > 0) {
    min_znam = n % b; /* zadnja znamenka */
    n /= b;
    while (n > 0) {
        znam = n % b;
        if (znam < min_znam) min_znam = znam;
        n /= b;
    }
    printf(" Najmanja znamenka u bazi %u je"
           " %u\n", b, min_znam);
}
else
    printf(" Nema znamenki\n");
```

## Najmanja znamenka broja — Izbjegavati!

```
/* Najmanja znamenka broja n u bazi b.
   ‘‘Lazna’’ inicijalizacija na b.
   Unistava n dijeljenjem.
*/
min_znam = b;
while (n > 0) {
    znam = n % b;
    if (znam < min_znam) min_znam = znam;
    n /= b;
}

printf(" Najmanja znamenka u bazi %u je %u\n",
       b, min_znam);
```

# Provjere znamenki broja

Najjednostavniji primjeri “provjere” odgovaraju standardnim kvatifikatorima 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, ili 1/0.



## Postoji znamenka ... ?

**Primjer.** Program treba učitati **nenegativni** cijeli broj **n** (tipa **unsigned int**) i naći odgovor na pitanje

- **postoji** li **znamenka** tog broja koja je jednaka **5** (u zadanoj bazi  $b = 10$ ).

Traženi rezultat je

$$(a_0 = 5) \vee (a_1 = 5) \vee \dots \vee (a_k = 5).$$

**Algoritamski:**

$$\text{rezultat} = \text{rezultat} \ || \ (a_i == 5), \quad i = 0, 1, \dots, k.$$

**Inicijalizacija** za postoji? Prazan skup!

**Inicijalizacija** za disjunkciju (ili)?

- **Neutral** za disjunkciju: **rezultat = 0** (laž).

## Postoji znamenka ... ? (nastavak)

Bitni odsječak programa izgleda ovako:

---

```
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");
```

---

## Postoji znamenka ... ? (nastavak)

Skraćena varijanta koja **prekida** petlju čim **sazna** odgovor:

---

```
odgovor = 0; /* NE, laz */
while (n > 0) {
    znam = n % b;
    if (znam == trazena) {
        odgovor = 1;
        break;
    }
    n /= b;
}
```

---

# Svaka znamenka ... ?

Primjer. Program treba učitati **nenegativni** cijeli broj **n** (tipa `unsigned int`) i naći odgovor na pitanje

• je li **svaka** znamenka tog broja jednaka **5** (u zadanoj bazi  $b = 10$ ).

Traženi rezultat je

$$(a_0 = 5) \wedge (a_1 = 5) \wedge \dots \wedge (a_k = 5).$$

Algoritamski:

`rezultat = rezultat && (a_i == 5), i = 0, 1, ..., k.`

Inicijalizacija za svaki? Prazan skup!

Inicijalizacija za konjunkciju (i)?

• **Neutral** za konjunkciju: `rezultat = 1` (istina).

## *Svaka znamenka ... ? (nastavak)*

Bitni odsječak programa izgleda ovako:

---

```
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");
```

---

## Svaka znamenka ... ? (nastavak)

Skraćena varijanta koja **prekida** petlju čim **sazna** odgovor:

---

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

---

# Palindrom

**Primjer.** Program treba učitati **neneaktivni** 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 obje strane?

Na primjer, **14741 je** palindrom, a **14743 nije**.

**Trik:** umjesto provjere odgovarajućih znamenki,

- **prva = zadnja, druga = predzadnja, ...**  
(probajte to napisati tako da radi za **svaki** prikazivi **n** i **b**),
- napravimo broj s **obratnim** poretком znamenki i usporedimo ga s **polaznim** brojem!

# Palindrom (nastavak)

```
#include <stdio.h>

/* Provjera je li prirodni broj palindrom. */

int main(void)
{
    unsigned int b = 10;
    unsigned int n, m1, m2, palindrom;

    printf(" Upisi nenegativni broj n: ");
    scanf("%u", &n);

    printf(" Broj = %u\n", n);
```



## Palindrom (nastavak)

```
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 za $n = 14743$

Primjer. Na početku je  $m_1 = n = 14743$  i  $m_2 = 0$ .

$n$	pomak $m_2$ ulijevo (množenje s $b$ )	dodaj $n \% b$	novi $m_2$	novi $n$
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  $m_2 = 34741 \neq m_1 = 14743 \implies n$  nije palindrom.

## Palindrom — korektnost?

**Pitanje.** Radi li ovaj program **korektno** za **svaki** prikazivi ulaz?

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

**Probajte** i za druge **baze**!

**Zadatak.** Neka je baza  $b = 2^{30} + 1 = 1073741825$ , a broj je  $n = b + 5 = 1073741830$ . Je li odgovor ( $= 1 = \text{da}$ ) **točan**?

**Zadatak.** **Koliko** ima baza  $b$  (u tipu **unsigned** na 32 bita), u kojima ovaj algoritam daje **pogrešan** odgovor za **bar jedan**  $n$ ?

**Izazov** = **program** (s podosta matematike prije toga):  
ukupan broj nađenih baza = **715827889**, vrijeme = **0.23 s**!

# Najveća zajednička mjera

**Primjer.** Jedan od prvih algoritama u povijesti je **Euklidov** algoritam za nalaženje **najveće** zajedničke mjere  $M(a, b)$ , **cijelih** brojeva  $a$  i  $b$ , uz pretpostavku da je  $b \neq 0$ .

Algoritam se bazira na Euklidovom teoremu o dijeljenju

•  $a = q \cdot b + r$ , za neki  $q \in \mathbb{Z}$ , gdje je  $r$  **ostatak**,  $0 \leq r < |b|$ .

Ključni koraci:

• Ako  $d \mid a$  i  $d \mid b$ , onda  $d \mid r$ , pa je  $M(a, b) = M(b, r)$   
(“**smanjujemo**” argumente, po apsolutnoj vrijednosti).

• Ako je  $r = 0$ , onda je  $a = q \cdot b$ , pa je  $M(a, b) = b$  (**kraj**).

**Test-primjeri:**  $a = 48$ ,  $b = 36$  ili  $a = 21$ ,  $b = 13$ .

Probajte i za negativne brojeve!

## Najveća zajednička mjera (nastavak)

Dio programa koji računa  $M(a, b)$ :

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

## Najveća zajednička mjera (nastavak)

Ovaj algoritam radi i za negativne brojeve  $a$ ,  $b$ ,

- samo se može dogoditi da je mjera negativna.

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

Jedna mogućnost je da izračunamo  $M(|a|, |b|)$ . Funkcije za apsolutnu vrijednost zovu se `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);
```

---

Još jednostavnije je samo vratiti  $|M(a, b)|$ . Za to, u algoritmu treba staviti `mjera = abs(b)`, umjesto `mjera = b`.

## Najveća zajednička mjera (nastavak)

Može i ovako — s malo **manje** “teksta”:

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

Za  $M(a, b) > 0$ , na kraju treba staviti  **$mjera = abs(a)$** .

## Najveća zajednička mjera (nastavak)

**Sporija** varijanta za istu stvar, bez računanja ostataka, koristeći samo **oduzimanje**:

```
int a, b, mjera;
...
while (a != b)
    if (a > b)
        a -= b;
    else
        b -= a;
mjera = a;    /* moze i b. */
```

**Oprez!** Ovo **radi** samo za **prirodne** brojeve  $a$  i  $b$ .

Zato je, na početku, dobro dodati  $a = \text{abs}(a)$ ;  $b = \text{abs}(b)$ ;



## Potencija broja 2

**Primjer.** Program treba učitati **nenegativni** cijeli broj **n** (tipa `unsigned int`) i naći odgovor na pitanje

• je li broj **n** **potencija** broja  $d = 2$ ,

tj. može li se **n** prikazati u obliku

•  $n = d^k$ , s tim da je eksponent  $k > 0$ ?

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

$$n = d^k \cdot m, \quad m \bmod d \neq 0, \quad \text{tj. } d \text{ ne dijeli } m,$$

gdje je  $k \geq 0$  cijeli broj. **Dokažite!**

Slično rastavu na **proste** faktore, samo  $d$  **ne mora** biti **prost**.

## Potencija broja 2 (nastavak)

Treba **naći** brojeve  $k$  i  $m$  u tom rastavu

$$n = d^k \cdot m, \quad m \bmod d \neq 0, \quad \text{tj. } d \text{ ne dijeli } m.$$

Ideja:

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

Dakle,  $n$  je **potencija** broja  $d$ , ako i samo ako

- na **kraju** vrijedi:  $k > 0$  i  $m = 1$ .

Dijeljenje broja  $n$  s  $d$  radimo u **istoj** varijabli  $n$ , tako da je **konačna** vrijednost od  $n = m$ .

## Potencija broja 2 (nastavak)

Bitni odsječak programa izgleda ovako:

```
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

**Primjer.** Program treba učitati **cijeli** broj **n** (tipa **int**) i napisati **prikaz** tog broja u računalu — kao niz **bitova**.

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

---

```
#include <stdio.h>

/* Prikaz cijelog broja u racunalu. */

int main(void)
{
    int nbits, broj, bit, i;
    unsigned mask;
```

## Prikaz cijelog broja u računalu (nastavak)

```
    /* Broj bitova u tipu int. */  
    nbits = 8 * sizeof(int);  
  
    /* Pocetna maska ima bit 1  
       na najznacajnijem mjestu. */  
    mask = 0x1 << nbits - 1;  
  
    printf(" Upisi cijeli broj: ");  
    scanf("%d", &broj);  
    printf(" Prikaz broja %d:\n ", broj);
```

## Prikaz cijelog broja u računalu (nastavak)

```
for (i = 1; i <= nbits; ++i) {
    /* Maskiranje odgovarajućeg bita. */
    bit = broj & mask ? 1 : 0;
    printf("%d", bit);
    if (i % 4 == 0) printf(" ");
    /* Pomak maske za 1 bit udesno. */
    mask >>= 1;
}
printf("\n");

return 0;
}
```

---

## Prikaz cijelog broja u računalu (nastavak)

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

---