Programiranje 1
9. predavanje

Saša Singer

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

PMF – Matematički odjel, 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.
  - Zadaci i varijacije.

Informacije

Termin prvog kolokvija je (službeno):

- utorak, 10. 11., u 12 sati.

Upozorenje — vezano uz prijave za zadaće:

- trenutni broj uspješno prijavljenih studenata još uvijek je zabrinjavajući — oko 230, od (barem) 274.

Pogledajte popis studenata na webu!

Lijepo molim, "ne šalite" se, uspješna prijava je

- nužan preduvjet za izlazak na kolokvij.

To pravilo se ne mijenja! Rok za prijavu je 48 sati prije kolokvija.

Informacije — Praktični kolokvij

Praktični kolokvij — prvi krug kreće tjedan dana nakon kolokvijskog dvotjedna. Preciznije,

- subota, 21. 11. — subota, 28. 11.,
- obje subote su namjerno uključene (prazni praktikumi).

Upisivanje za termine će biti odmah iza kolokvija,

- u 8. tjednu nastave, 16. 11. — 20. 11.

Popis s terminima će biti na zidu, desno od moje sobe, čim složimo termine.

Zapamtite: Vrijeme za rješenje je 45 minuta.

- Zadaci su objavljeni na webu.
- Korisno je odmah pogledati i početi vježbati (i prije redovitog kolokvija).

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.
  - Zadaci i varijacije.

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 ukoliko 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 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 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

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

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

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

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`.

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;
}
```

Broj znamenki broja u zadanoj bazi — logaritam

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

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

normalizirani prikaz tog broja u bazi b, tj. vrijedi

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

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

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

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

Broj znamenki broja — logaritam (nastavak)

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

- `log` = ln,
- `log10` = $\log_{10}$.

Nama treba logaritam u bazi b. To dobijemo ovako

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

"Najveće cijelo" možemo dobiti pretvaranjem tipova:

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

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

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 (suma) znamenki broja

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

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

Inicijalizacija za zbrajanje?

Zbroj 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 %d\n",
       b, suma);
```

Umnožak (produkt) znamenki broja

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

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

Inicijalizacija za množenje?

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 $b = 10$.

Inicijalizacija za maksimum?

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 (nastavak)

```
/* Najveca znamenka broja n u bazi b.
   ‘‘Lazna’’ inicijalizacija na -1
   ne moze 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(" Najveca 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 $b = 10$.

Inicijalizacija za minimum?

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 (nastavak)

```
/* 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. ima "logički" tip

- DA/NE, 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 $b = 10$).

Inicijalizacija za postoji? (Prazan skup!)

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:

```
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 $b = 10$).

Inicijalizacija za svaki? (Prazan skup!)

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:

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

Palindrom

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

- da li je broj `n` palindrom (u zadanoj bazi $b = 10$),

tj. da li se `n` "čita" jednako s obje strane?

Trik: umjesto provjere znamenki,

- napravimo broj s obratnim poretkom znamenki i usporedimo!

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;
}
```

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.

Algoritam se bazira na Euklidovom teoremu o dijeljenju

- $a = q \cdot b + r$, za neki $q \in \mathbb{Z}$, gdje je r ostatak.

Ključni koraci:

- Ako $d \mid a$ i $d \mid b$, onda $d \mid r$, pa je $M(a, b) = M(b, r)$ ("smanjujemo" argumente).
- Ako je $r = 0$, onda je $a = q \cdot b$, pa je $M(a, b) = b$ (kraj).

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

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)

Može i ovako — s malo manje "teksta":

```
int a, b, ostatak, mjera;
...
while (b > 0) {
    ostatak = a % b;
    a = b;
    b = ostatak;
}
mjera = 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. */
```

Potencija broja 2

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

- da li je broj `n` potencija broja $d = 2$,

tj. da li se `n` može prikazati u obliku $\texttt{n} = d^k$, s tim da je eksponent $k > 0$?

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 odgovarajuceg 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
```