

# *Programiranje 1*

## *5. predavanje*

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

PMF – Matematički odjel, Zagreb

# Sadržaj predavanja

- Cijeli brojevi — prikaz i aritmetika (nastavak):
  - Prikaz brojeva s predznakom — sustav ostataka.
  - Prikaz negativnih brojeva — komplementiraj i dodaj 1.
  - Tipične pogreške u korištenju cijelih brojeva.
- Prikaz realnih brojeva — “floating-point” standard:
  - Osnovni oblik “floating-point” prikaza — mantisa i eksponent.
  - Greške zaokruživanja u prikazu.
  - Pojam “jedinične greške zaokruživanja”.
  - IEEE standard — tipovi: *single*, *double*, *extended*.

# Sadržaj predavanja (nastavak)

- Greške zaokruživanja u aritmetici realnih brojeva:
  - Greške zaokruživanja osnovnih aritmetičkih operacija.
  - Opasno ili “katastrofalno” kraćenje.
  - “Širenje” grešaka zaokruživanja, stabilni i nestabilni algoritmi.
- Primjeri širenja grešaka zaokruživanja i izbjegavanja nestabilnosti:
  - Parcijalne sume harmonijskog reda.
  - Korijeni kvadratne jednačbe.

# Informacije — sutra = odrada za 2. 11.

Ne zaboravite da **sutra**

- **subota, 10. studenog,**

**odrađujemo** petak, 2. studenog, po sljedećem rasporedu:

- **predavanja** (S. Singer) — **10–12** u (003),

- **vježbe** (V. Šego) — **12–14** u (004).

# Prikaz cijelih brojeva s predznakom

# Cijeli brojevi s predznakom

Cijeli brojevi s predznakom modeliraju skup  $\mathbb{Z}$  cijelih brojeva.

Ako imamo  $n$  bitova na raspolaganju za prikaz, onda skup prikazivih brojeva ima  $2^n$  elemenata.

Među prikazivim brojevima moraju biti i (neki) negativni brojevi. Zgodno bi bilo da ih je podjednako mnogo kao i pozitivnih (odnosno, nenegativnih) brojeva.

**Standardni dogovor:** u računalu se prikazuje

- najveći mogući podskup uzastopnih brojeva iz  $\mathbb{Z}$  koji je “skoro” simetričan oko 0.

**Napomena:** “Prava” simetrija oko nule dala bi neparan broj prikazivih brojeva — što nema smisla.

## Cijeli brojevi s predznakom (nastavak)

Dakle, prikazivih negativnih brojeva ima podjednako mnogo kao i nenegativnih.

Ako želimo da točno polovina tih brojeva bude negativna, onda njih mora biti  $2^n/2 = 2^{n-1}$ . Nenegativnih brojeva je, naravno, isto toliko.

To znači da je skup svih prikazivih cijelih brojeva s predznakom jednak

$$\mathbb{Z}_{2^n}^- = \{ -2^{n-1}, -2^{n-1} + 1, \dots, -2, -1, \\ 0, 1, \dots, 2^{n-1} - 2, 2^{n-1} - 1 \}.$$

Brojeve izvan tog skupa ne možemo prikazati koristeći samo  $n$  bitova.

## Cijeli brojevi s predznakom (nastavak)

Najmanji i najveći prikazivi cijeli broj s predznakom su, redom:

$$-2^{n-1}, \quad 2^{n-1} - 1.$$

Tipične vrijednosti za ta dva “granična” broja su:

$n$	$-2^{n-1}$	$2^{n-1} - 1$
8	-128	127
16	-32 768	32 767
32	-2 147 483 648	2 147 483 647

Uočite da raspon prikazivih cijelih brojeva s predznakom nije jako velik, čak i za  $n = 32$  (što je standard). **Oprez!**

Zato se danas sve više koristi  $n = 64$  (želja za  $n = 128$ ).



# Prikaz cijelih brojeva s predznakom

Kako stvarno izgleda prikaz brojeva u tih  $n$  bitova?

Prikaz pojedinih (prikazivih) brojeva potpuno je određen s dva zahtjeva:

- nenegativni brojevi imaju isti prikaz kao u cijelim brojevima bez predznaka (dovoljno je to tražiti samo za jedan broj, na primjer, nulu),
- aritmetika za te prikaze mora (kao i ranije, za brojeve bez predznaka) dati “dobru” algebarsku strukturu na cijelim brojevima s predznakom, tj. na skupu  $\mathbb{Z}_n^-$ .

Prvi zahtjev je dosta jasan, ali što znači drugi?

## Veza između prikaza i aritmetike

Kod brojeva **bez predznaka**, prikaz binarnim znamenkama je očito odgovarao **aritmetici** — recimo, ovako:

- kad **prikazu** broja  $B$  (kao nizu bitova u bazi 2) **dodamo 1**, dobijemo točno **prikaz** broja  $B + 1$ .

(Hm, nismo baš puno razmišljali o tome).

**Modularna** aritmetika modulo  $2^n$  dodatno još “zatvara krug” na  $n$  bitova, tj. daje **zatvorenost** operacija, a onda i dobru strukturu **prstena s jedinicom** na  $\mathbb{Z}_{2^n} = \{0, 1, \dots, 2^n - 1\}$ .

Zato gore, umjesto običnog cjelobrojnog  $+$ , koristimo  $\oplus_{2^n}$ .

**Potpuno isto** mora vrijediti i za **prikaze** cijelih brojeva **s predznakom** iz  $\mathbb{Z}_{2^n}^-$  — inače nemamo jednostavnu realizaciju aritmetike i dobru strukturu.

# Aritmetika cijelih brojeva s predznakom

Dakle, **aritmetika** za cijele brojeve s predznakom **mora** i dalje biti ista, tj.

● **modularna aritmetika modulo  $2^n$** ,

samo je **interpretacija** sustava ostataka drugačija.

Skup svih prikazivih cijelih brojeva **s predznakom**

$$\mathbb{Z}_{2^n}^- = \{ -2^{n-1}, \dots, -1, 0, 1, \dots, 2^{n-1} - 1 \}$$

je, također, **potpuni** sustav ostataka modulo  $2^n$ .

Na tom skupu opet možemo definirati operacije **zbrajanja** i **množenja** (bez posebnih oznaka) preko odgovarajućih **ostataka** cjelobrojnih operacija  $+$  i  $\cdot$ . Kao i prije, dobivamo strukturu **prstena s jedinicom**, ali sad na skupu  $\mathbb{Z}_{2^n}^-$ .

## Prikaz cijelih brojeva s predznakom (nastavak)

A sad je lako dobiti prikaz svih negativnih brojeva.

Usporedimo skupove prikazivih brojeva bez predznaka i s predznakom, tj. pripadne sustave ostataka modulo  $2^n$ ,

$$\mathbb{Z}_{2^n} = \{0, 1, \dots, 2^n - 1\},$$

$$\mathbb{Z}_{2^n}^- = \{-2^{n-1}, \dots, -1, 0, 1, \dots, 2^{n-1} - 1\}.$$

Pravilo za prikaz je vrlo jednostavno:

- iste prikaze imaju oni brojevi (iz ta dva skupa) koji imaju isti “pravi” ostatak modulo  $2^n$ .

Zaključak: za  $B = 1, \dots, 2^{n-1}$ ,

- prikaz negativnog broja  $-B$  (s predznakom)
- jednak je prikazu broja  $2^n - B$  (bez predznaka).

## Prikaz cijelih brojeva s predznakom (nastavak)

Detaljnija argumentacija ovog zaključka ide na sljedeći način.

Prvo “poklopimo” zajednički dio **nenegativnih** brojeva

$$0, \dots, 2^{n-1} - 1$$

u ta dva skupa. Svaki od tih brojeva ima **isti prikaz** u oba skupa, pa ih “sparujemo”

$$B \in \mathbb{Z}_{2^n}^- \iff B \in \mathbb{Z}_{2^n},$$

za  $B = 0, \dots, 2^{n-1} - 1$ .

Uočite da

🔴 prikazi svih ovih brojeva imaju **vodeći bit** jednak 0.

## Prikaz cijelih brojeva s predznakom (nastavak)

Zatim “zatvaramo modularni krug” dodavanjem **jedan po jedan** modulo  $2^n$  i “sparujemo” odgovarajuće brojeve.

Prvi “dodaj **jedan**” modulo  $2^n$  daje

$$((2^{n-1} - 1) + 1) \bmod 2^n = 2^{n-1} \bmod 2^n = \begin{cases} 2^{n-1} \in \mathbb{Z}_{2^n}, \\ -2^{n-1} \in \mathbb{Z}_{2^n}^-, \end{cases}$$

tj. **zatvara krug** u  $\mathbb{Z}_{2^n}^-$ , pa “sparujemo”

$$-2^{n-1} \in \mathbb{Z}_{2^n}^- \iff 2^{n-1} \in \mathbb{Z}_{2^n}.$$

A dalje sve ide redom.

## Prikaz cijelih brojeva s predznakom (nastavak)

Kad “dodaj jedan” ponovimo  $k$  puta, tj. kad broju  $2^{n-1} - 1$  dodamo  $k$  modulo  $2^n$ , izlazi:

$$-2^{n-1} + (k - 1) \in \mathbb{Z}_{2^n}^- \iff (2^{n-1} - 1) + k \in \mathbb{Z}_{2^n},$$

za  $k = 1, \dots, 2^{n-1}$ . Supstitucijom  $-B = -2^{n-1} + (k - 1)$  za negativne brojeve dobivamo

$$-B \in \mathbb{Z}_{2^n}^- \iff 2^n - B \in \mathbb{Z}_{2^n},$$

i to vrijedi za  $B = 1, \dots, 2^{n-1}$ .

Dakle, spareni brojevi razlikuju se za točno  $2^n$ .

To je jasno, jer moraju imati isti “pravi” ostatak modulo  $2^n$ .

# Prikaz cijelih brojeva s predznakom (nastavak)

Zaključak: za  $B = 1, \dots, 2^{n-1}$ ,

- prikaz **negativnog** broja  $-B$  (s predznakom)
- jednak je prikazu broja  $2^n - B$  (bez predznaka).

Uočite da

- prikazi svih ovih brojeva imaju **vodeći bit** jednak **1**.



## Primjer — prikaz cijelih brojeva za $n = 3$

Primjer. Neka je  $n = 3$ , tj. imamo samo 3 bita za prikaz. Prikazivih brojeva ima  $2^3 = 8$ . Prikazi cijelih brojeva **bez predznaka** iz  $\mathbb{Z}_8$  i cijelih brojeva **s predznakom** iz  $\mathbb{Z}_8^-$  su:

$B \in \mathbb{Z}_8$	$B \in \mathbb{Z}_8^-$	prikaz
0	0	000
1	1	001
2	2	010
3	3	011
4	-4	100
5	-3	101
6	-2	110
7	-1	111

## Prikaz cijelih brojeva s predznakom — primjeri

**Primjer.** Prikaz broja  $-2^{n-1}$  dobivamo tako da uzmemo  $B = 2^{n-1}$  (koji sam **nije** prikaziv s predznakom) i pogledamo prikaz broja  $2^n - B$  bez predznaka:

$$-2^{n-1} \longleftrightarrow 2^n - 2^{n-1} = 2^{n-1} = [1\ 0\ 0\ \dots\ 0\ 0].$$

Ovdje koristimo skraćenu oznaku

$$[\text{bit}_{n-1}\ \text{bit}_{n-2}\ \text{bit}_{n-3}\ \dots\ \text{bit}_1\ \text{bit}_0]$$

za zapis vrijednosti svih  $n$  bitova u prikazu broja.

**Primjer.** Broj  $-1$  ima prikaz ( $B = 1$ )

$$-1 \longleftrightarrow 2^n - 1 = [1\ 1\ 1\ \dots\ 1\ 1].$$

# Prikaz cijelih brojeva s predznakom — primjeri

Primjer. Kad **najvećem** prikazivom broju (s predznakom)  $2^{n-1} - 1$  dodamo 1, rezultat je **najmanji** prikazivi broj (s predznakom), jer je

$$\begin{aligned}(2^{n-1} - 1) + 1 &= ((2^{n-1} - 1) + 1) \bmod 2^n \\ &= 2^{n-1} \bmod 2^n = -2^{n-1} \in \mathbb{Z}_{2^n}^-.\end{aligned}$$

Naravno, isti rezultat izlazi i običnim binarnim “zbrajanjem prikaza”

$$[0\ 1\ 1\ \dots\ 1\ 1] + [0\ 0\ 0\ \dots\ 0\ 1] = [1\ 0\ 0\ \dots\ 0\ 0].$$

Uočite da **nema** prijenosa.

## Prikaz cijelih brojeva s predznakom — primjeri

Primjer. Analogno, zbrajanju  $(-1) + 1 = 0$  odgovara binarno “zbrajanje prikaza”

$$[1\ 1\ 1\ \dots\ 1\ 1] + [0\ 0\ 0\ \dots\ 0\ 1] = 1 [0\ 0\ 0\ \dots\ 0\ 0].$$

Ovdje imamo prijenos, ali se on “modularno” ignorira.

# Prikaz cijelih brojeva s predznakom — komentar

Dakle, prikaz negativnih brojeva je posljedica one iste modularne aritmetike koja vrijedi i za cijele brojeve bez predznaka.

Tehnički gledano, tri osnovne operacije  $+$ ,  $-$  i  $\cdot$  na cijelim brojevima u računalu se izvršavaju

- **potpuno jednako** (istim “elektroničkim krugovima”) za cijele brojeve s predznakom i bez njega,

Važni su samo bitovi u prikazu, a ne i njihova interpretacija!

Za cjelobrojno dijeljenje to ne vrijedi (v. malo kasnije).

## Vodeći bit — predznak

U prikazu cijelih brojeva s predznakom

- svi **nenegativni** brojevi imaju **vodeći bit** jednak 0, a
- svi **negativni** brojevi imaju **vodeći bit** jednak 1.

Zato se vodeći bit obično zove i **bit predznaka** (engl. **sign**).

Taj naziv, nažalost, može zavesti na **pogrešnu** ideju o prikazu. Prikaz cijelih brojeva s predznakom **ne dobiva** se kao:

- bit **predznaka** i prikaz **apsolutne** vrijednosti broja (bez predznaka, s bitom manje).

Za razliku od ovog, kod “realnih” brojeva to **vrijedi**, što samo povećava mogućnost zabune.

# Prikaz suprotnog broja preko komplementa

Za zadani broj  $B > 0$ , kad tražimo prikaz broja  $-B$ ,

- broj  $2^n - B$  nije baš lako izračunati, čim je broj bitova  $n$  iole veći.

Srećom, ima i lakši način! Prikaz broja  $-B$  možemo naći i tehnikom “dvojnog komplementa”:

- prikaz zadanog broja  $B$  treba komplementirati, tj. pretvoriti nule u jedinice i obratno (tj.  $0 \leftrightarrow 1$ ),
- a zatim tom komplementu treba dodati 1 (modulo  $2^n$ ).

Kratko opravdanje: ako je  $\bar{B}$  broj (bez predznaka) kojem odgovara komplementirani prikaz broja  $B$ , onda je (očito)

$$B + \bar{B} = [1\ 1\ 1\ \dots\ 1\ 1] = 2^n - 1.$$

# Prikaz suprotnog broja preko komplementa

Prebacimo 1 s desne na lijevu stranu

$$B + \bar{B} + 1 = 2^n,$$

i zapišimo ovu relaciju “modularnim zbrajanjem”  $\oplus_{2^n}$  u  $\mathbb{Z}_{2^n}$

$$B \oplus_{2^n} (\bar{B} \oplus_{2^n} 1) = 0.$$

Dakle, **jedinstveni** suprotni element (inverz obzirom na zbrajanje) elementa  $B \in \mathbb{Z}_{2^n}$  jednak je  $\bar{B} \oplus_{2^n} 1$ .

Drugim riječima, **suprotni** broj dobivamo tako da

- **komplementiramo** broj (odnosno, njegov prikaz) i
- **dodamo 1** modulo  $2^n$ .

**Potpuno isto** “pravilo” vrijedi i za brojeve **s predznakom!**



## Prikaz cijelih brojeva s predznakom — primjer

**Primjer.** Uzmimo da je  $n = 8$  (da ne pretjeravamo), i pogledajmo zapis broja  $-120$ . Za početak, vrijedi

$$-2^7 = -128 \leq -120 \leq 127 = 2^7 - 1,$$

pa je  $-120$  prikaziv. Nadalje, binarni prikaz broja  $B = 120$  je

$$\begin{aligned} 120 &= 64 + 32 + 16 + 8 \\ &= 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 \\ &\quad + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0. \end{aligned}$$

Dakle, broj  $B = 120$  kao cijeli broj bez predznaka ima prikaz

$$120 \longleftrightarrow [0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0].$$

## Prikaz cijelih brojeva s predznakom — primjer

U zadanih  $n = 8$  bitova, broj  $B = 120$  (kao cijeli broj  $s$  predznakom ili bez njega) ima prikaz

7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	0

Komplementiranjem prikaza (bit po bit) dobivamo

7	6	5	4	3	2	1	0
1	0	0	0	0	1	1	1

Dodamo 1 (modulo  $2^8$ ). Prikaz broja  $-B = -120$  je

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

## Prikaz cijelih brojeva s predznakom — primjer

Istu stvar možemo dobiti i tako da nađemo prikaz broja

$$2^8 - B = 256 - 120 = 136$$

kao cijelog broja **bez predznaka**. Imamo redom

$$\begin{aligned} 136 &= 128 + 8 \\ &= 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 \\ &\quad + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0. \end{aligned}$$

Za  $n = 8$ , broj  $-120$  kao cijeli broj **s predznakom** ima isti prikaz kao i **136 bez predznaka**

$$-120 \longleftrightarrow 136 \longleftrightarrow [1\ 0\ 0\ 0\ 1\ 0\ 0\ 0].$$

## Dijeljenje cijelih brojeva s predznakom

Prošli puta smo uveli operacije  $\text{div}$  (cjelobrojni kvocijent) i  $\text{mod}$  (ostatak) na skupu  $\mathbb{Z} \times \mathbb{N}$ .

**Definicija.** Neka su  $a \in \mathbb{Z}$  i  $b \in \mathbb{N}$  bilo koji brojevi, i neka su  $q \in \mathbb{Z}$  (cjelobrojni kvocijent) i  $r \in \mathbb{Z}_b$  (ostatak) **jedinstveni** brojevi za koje vrijedi

$$a = q \cdot b + r.$$

Operacije  $\text{div}$  i  $\text{mod}$  **definiramo** relacijama

$$a \text{ div } b := q \in \mathbb{Z}, \quad a \text{ mod } b := r \in \mathbb{Z}_b.$$

Za početak, uočite da su **obje** operacije definirane na skupu  $\mathbb{Z} \times \mathbb{N}$ , a kodomene su različite.

## Dijeljenje cijelih brojeva s predznakom

Sad nam **treba** proširenje na skup  $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ ,

- kad **cjelobrojno dijelimo dva cijela broja** (što, naravno, ima smisla),

da dobijemo cjelobrojno dijeljenje za cijele brojeve **s predznakom**  $\mathbb{Z}_{2^n}^-$  (koji modeliraju  $\mathbb{Z}$ ).

Naravno, ideja je ista kao i kod brojeva bez predznaka.

Cjelobrojno dijeljenje ili **dijeljenje s ostatkom** cijelih brojeva **s predznakom** je naprosto

- **restrikcija** odgovarajućih operacija **div** i **mod**.

Dakle, treba nam proširenje tih operacija na  $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ .

Međutim, **ne postoji** dogovoreni standard za to proširenje.

## Dijeljenje cijelih brojeva s predznakom

U većini programskih jezika (uključivo i C) **vrijedi** da

- stvar radi očekivano (prema Euklidovom teoremu) **samo na skupu**  $\mathbb{N}_0 \times \mathbb{N}$ .

Dakle, za **nenegativne** brojeve **s predznakom** dobivamo očekivane (i korektne) rezultate u cjelobrojnom dijeljenju.

A za negativne operande? **Nije precizno definirano!**

Na primjer, C standard (knjiga Kernighan, Ritchie) kaže:

- ako je barem jedan od dva operanda **negativan**, rezultat **ovisi o implementaciji**.

Dakle, **nije predvidiv** — isti program može davati **različite** rezultate, ovisno o računalu i izboru C kompilera.

Zato — **čitajte upute** ili, naprosto, probajte!

# Dijeljenje cijelih brojeva s predznakom

## Eksperiment:

- test-program `divmod.c` (pokaži!),
- Intelov C++ compiler (verzija 9.1.032), na IA-32.

Rezultati  $q = a \operatorname{div} b$  i  $r = a \operatorname{mod} b$  za  $a = \pm 5$ ,  $b = \pm 3$ :

$a$	$b$	$q$	$r$
5	3	1	2
-5	3	-1	-2
5	-3	-1	2
-5	-3	1	-2

Operacije `div` i `mod` interpretiramo na  $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ .

# Veza cjelobrojnog i običnog dijeljenja

Ključ za interpretaciju:

- **kvocijent** se uvijek “zaokružuje” prema nuli,

$$q = \text{sign}\left(\frac{a}{b}\right) \cdot \left\lfloor \left| \frac{a}{b} \right| \right\rfloor,$$

- **ostatak** ima isti predznak kao i  $a$ .

$$r = \text{sign}(a) \cdot (|a| \bmod |b|).$$

Za ostatak  $r$  ovdje vrijedi:

- $0 \leq r < |b|$ , za  $a \geq 0$ ,
- $-|b| < r \leq 0$ , za  $a < 0$ .



## Veza cjelobrojnog i običnog dijeljenja

**Razlog:** standardno ograničenje na **ostatak**  $0 \leq r < b$ , tj.  $r \in \mathbb{Z}_b$ , odgovara cijelim brojevima **bez predznaka**.

Međutim, kod brojeva s predznakom imamo i negativne brojeve, pa (možda) ima smisla dozvoliti da i ostaci budu negativni (u nekim slučajevima).

**Prednosti** ovakve “definicije” operacija **div** i **mod** na skupu  $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ :

- bez obzira na predznake od  $a$  i  $b$ ,
- dobivamo iste **apsolutne** vrijednosti kvocijenta  $q$  i ostatka  $r$ ,

tj. samo predznaci od  $q$  i  $r$  ovise o predznacima od  $a$  i  $b$ .

Ovo je i **najčešća** realizacija cjelobrojnog dijeljenja u praksi.

## Cijeli brojevi s predznakom — sažetak

Ako imamo  $n$  bitova za prikaz brojeva, onda je skup svih prikazivih cijelih brojeva bez predznaka jednak

$$\mathbb{Z}_{2^n}^- = \{ -2^{n-1}, -2^{n-1} + 1, \dots, -2, -1, \\ 0, 1, \dots, 2^{n-1} - 2, 2^{n-1} - 1 \}.$$

Za prikaz broja  $B \in \mathbb{Z}_{2^n}^-$  vrijedi:

- nenegativni brojevi  $B = 0, \dots, 2^{n-1} - 1$  imaju isti prikaz kao i bez predznaka,
- negativni brojevi  $B = -1, \dots, -2^{n-1}$  imaju isti prikaz kao i brojevi  $2^n + B$  bez predznaka.

## Cijeli brojevi s predznakom — sažetak

Osim toga, prikaz suprotnog broja  $-B$  dobivamo tako da

- komplementiramo prikaz samog broja i dodamo 1 modulo  $2^n$ .

Aritmetika cijelih brojeva s predznakom je modularna aritmetika modulo  $2^n$  na sustavu ostataka  $\mathbb{Z}_{2^n}^-$ .

- To vrijedi za operacije  $+$ ,  $-$  i  $\cdot$ .

Operacije cjelobrojnog dijeljenja s ostatkom  $\text{div}$  i  $\text{mod}$  daju iste rezultate kao da dijelimo u  $\mathbb{Z}$ ,

- ali treba provjeriti kako se dobiva proširenje ovih operacija s  $\mathbb{N}_0 \times \mathbb{N}$  na  $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ .

# Cijeli brojevi u C-u — sažetak

U programskom jeziku C:

- cijelim brojevima s predznakom odgovara tip koji se zove `int`.

Ovaj tip postoje u nekoliko raznih veličina, a razlike su u broju bitova  $n$  predviđenih za prikaz. Na 32-bitnim arhitekturama računala imamo sljedeće tipove:

- standardni `int` ( $n = 32$ ),
- `short` ( $n = 16$ ),
- `long` ( $n = 32$ ), tj. isto kao standardni `int`,
- a katkad i druge, poput `long long` ( $n = 64$ ).

Zapis konstanti (vrijednost, navođenje tipa) — sljedeći put.

# Cijeli brojevi s predznakom u C-u — primjer 1

```
#include <stdio.h>

int main(void) {
    short int i = 32766; /* n = 16 za short */

    i += 1;
    printf("%d\n", i); /* 32767 */
    i += 1;
    printf("%d\n", i); /* -32768, a ne 32768 */

    return 0;
}
```

`SHRT_MAX = 32767` u zaglavlju `limits.h`. Ovdje je  $n = 16$ .

# Cijeli brojevi — klasične greške

**Primjer.** Računanje  $n!$  u cjelobrojnoj aritmetici.

Za prirodni broj  $n \in \mathbb{N}$ , funkciju **faktorijela** definiramo na sljedeći način:

$$1! = 1,$$

$$n! = n \cdot (n - 1)!, \quad n \geq 2.$$

Napišimo program koji računa broj  $50!$  u cjelobrojnoj aritmetici (tip `int`).

# Cijeli brojevi — klasične greške (nastavak)

```
#include <stdio.h>

int main(void) {
    int i, f50 = 1;    /* n = 32 za int */

    for (i = 2; i <= 50; ++i)
        f50 *= i;

    printf(" f50 = %d\n", f50);    /* f50 = 0 */

    return 0;
}
```

Izlaz programa je: **f50 = 0**. Zašto?

## Cijeli brojevi — klasične greške (nastavak)

Za početak,  $50!$  je **ogroman** broj. Točna vrijednost je

$$50! = 30414\ 09320\ 17133\ 78043\ 61260\ 81660 \\ 64768\ 84437\ 76415\ 68960\ 51200\ 00000\ 00000,$$

i ima **65** dekadskih znamenki. Dakle, sigurno **nije** prikaziv u cjelobrojnoj aritmetici.

Granice za tip `int` ( $n = 32$  bita) iz zaglavlja `limits.h` su

$$\text{INT\_MAX} = 2147483647, \quad \text{INT\_MIN} = (-\text{INT\_MAX} - 1).$$

Objasnimo još zašto je  $f50 = 0$  u našem programu.

Cjelobrojna aritmetika u kojoj računamo je **modularna** aritmetika — modulo  $2^{32}$ .



## Cijeli brojevi — klasične greške (nastavak)

To znači da naš program kaže da je

$$50! = 0 \pmod{2^{32}},$$

ili da  $2^{32}$  dijeli  $50!$ .

**Zadatak.** Nađite **najveću** potenciju broja  $2$  koja dijeli  $50!$ , ili, općenito  $n!$ . Rješenje:

$$m = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{2^2} \right\rfloor + \left\lfloor \frac{n}{2^3} \right\rfloor + \left\lfloor \frac{n}{2^4} \right\rfloor + \left\lfloor \frac{n}{2^5} \right\rfloor + \dots$$

Za  $n = 50$  imamo  $m = 25 + 12 + 6 + 3 + 1 = 47$ .

**Zadatak.** Nađite **najmanji** broj  $n$  za koji je  $n! = 0$  u cjelobrojnoj aritmetici s  $\ell$  bitova za prikaz brojeva.

## Tablica $n!$ za $\ell = 16$ i $\ell = 32$ bita

Usporedimo rezultate za  $n!$  dobivene u **cjelobrojnoj** aritmetici s  $\ell = 16$  i  $\ell = 32$  bita s (ispravnim) rezultatom dobivenim u **realnoj** aritmetici.

$n!$	$\ell = 16$	$\ell = 32$	realna aritmetika
7!	5040	5040	5040
8!	-25216	40320	40320
9!	-30336	362880	362880
10!	24320	3628800	3628800
11!	5376	39916800	39916800
12!	-1024	479001600	479001600

## Tablica $n!$ za $\ell = 16$ i $\ell = 32$ bita (nastavak)

$n!$	$\ell = 16$	$\ell = 32$	realna aritmetika
13!	-13312	1932053504	6227020800
14!	10240	1278945280	87178291200
15!	22528	2004310016	1307674368000
16!	-32768	2004189184	20922789888000
17!	-32768	-288522240	355687428096000
18!	0	-898433024	6402373705728000
19!	0	109641728	121645100408832000
20!	0	-2102132736	2432902008176640000

# Prikaz “realnih” brojeva u računalu — IEEE standard

# Uvod u prikaz realnih brojeva

Kako pohraniti “jako velike” ili “jako male” brojeve?  
Recimo (dekadski pisano):

67800000000.0      0.000002078

Koristimo tzv. **znanstvenu** notaciju u kojoj

- prvo pišemo vodeće značajne znamenke broja,
- a zatim pišemo faktor koji ima oblik baza na odgovarajući eksponent, tj. potenciju baze.

Uz dogovor da vodeći dio bude između 1 i 10 (strogo ispod),  
to izgleda ovako:

$6.78 \cdot 10^{10}$        $2.078 \cdot 10^{-6}$ .

# Prikaz realnih brojeva

U računalu se binarni zapis realnog broja pohranjuje u znanstvenom formatu:

$$\text{broj} = \text{mantisa} \cdot 2^{\text{eksponent}}.$$

**Mantisa** se uobičajeno (postoje iznimke!) pohranjuje u tzv. **normaliziranom obliku**, tj.

$$1 \leq \text{mantisa} < (10)_2.$$

I za pohranu **mantise** i za pohranu **eksponenta** rezervirano je **konačno** mnogo binarnih znamenki. Posljedice:

- prikaziv je samo neki **raspon** realnih brojeva,
- niti svi brojevi unutar prikazivog raspona **nisu prikazivi** (mantisa predugačka)  $\implies$  **zaokruživanje**.

# Prikaz realnih brojeva (nastavak)

Primjer: Znanstveni prikaz binarnih brojeva:

$$1010.11 = 1.01011 \cdot 2^3$$

$$0.0001011011 = 1.01011 \cdot 2^{-4}$$

Primijetite da se vodeća jedinica u normaliziranom obliku **ne mora** pamtiti (ako je broj  $\neq 0$ ).

- Taj bit se može upotrijebiti za pamćenje dodatne znamenke mantise.

Tada se vodeća jedinica zove **skriveni bit** (engl. hidden bit) — jer se **ne pamti**.

Ipak ovo je samo pojednostavljeni prikaz realnih brojeva.

## Stvarni prikaz realnih brojeva

Najznačajnija promjena obzirom na pojednostavljeni prikaz:

- **eksponent** se prikazuje u “zamaskiranoj” ili “pomaknutoj” formi (engl. biased form).

To znači da se stvarnom eksponentu se **dodaje konstanta** takva da je “pomaknuti” eksponent uvijek pozitivan.

Ta konstanta ovisi o broju bitova za eksponent i bira se tako da je prikaziva **recipročna vrijednost najmanjeg normaliziranog broja**.

Takav “pomaknuti” eksponent naziva se **karakteristika**, a normaliziranu mantisu neki zovu i **signifikand**.



# Oznake

## Oznake:

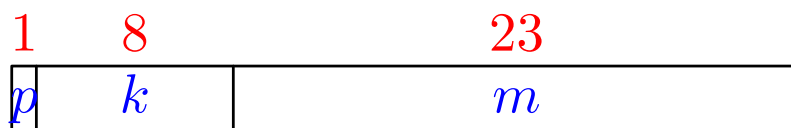
- **Crveno** — duljina odgovarajućeg polja (u bitovima), bitove brojimo od 0 zdesna nalijevo (kao i obično),
- $p$  — predznak: 0 za pozitivan broj, 1 za negativan broj,
- $k$  — karakteristika,
- $m$  — mantisa (signifikand).
- Najznačajniji bit u odgovarajućem polju je najljeviji.
- Najmanje značajan bit u odgovarajućem polju je najdesniji.

## Stvarni prikaz tipa single

“Najkraći” realni tip je tzv. realni broj **jednostruke** točnosti — u C-u poznat kao **float**.

On ima sljedeća svojstva:

- duljina: 4 byte-a (32 bita), podijeljen u **tri** polja.



- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- **stvarni eksponent** broja  $e$ ,  $e \in \{-126, \dots, 127\}$ ,
- **karakteristika**  $k = e + 127$ , tako da je  $k \in \{1, \dots, 254\}$ ,
- **karakteristike**  $k = 0$  i  $k = 255$  koriste se za “posebna stanja”.

## Stvarni prikaz tipa single (nastavak)

Primjer: Broj  $(10.25)_{10}$  prikažite kao broj u jednostrukoj točnosti.

$$\begin{aligned}(10.25)_{10} &= \left(10 + \frac{1}{4}\right)_{10} = (10 + 2^{-2})_{10} \\ &= (1010.01)_2 = 1.01001 \cdot 2^3.\end{aligned}$$

Prema tome je:

$$p = 0$$

$$k = e + 127 = (130)_{10} = (2^7 + 2^1)_{10} = 1000\ 0010$$

$$m = 0100\ 1000\ 0000\ 0000\ 0000\ 000$$

## Prikazi nule: $k = 0, m = 0$

Realni broj **nula** ima dva prikaza: mantisa i karakteristika su joj nula, a predznak može biti **0** “pozitivna nula” ili **1** “negativna nula”.

Ta dva prikaza nule su:

$$+0 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

$$-0 = 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

Smatra se da su vrijednosti ta dva broja jednake (kad se uspoređuju).

## Denormalizirani brojevi: $k = 0, m \neq 0$

Ako je  $k = 0$ , a postoji **barem jedan** znak mantise koji nije nula, onda se kao eksponent uzima  $-126$ . Mantisa takvog broja **nije normalizirana** i počinje s  $0.m$ .

Takvi brojevi zovu se **denormalizirani brojevi**.

**Primjer:** Kako izgleda prikaz realnog broja

$$0.000\ 0000\ 0000\ 0000\ 0000\ 1011 \cdot 2^{-126}?$$

Rješenje:

$$p = 0$$

$$k = 0000\ 0000$$

$$m = 000\ 0000\ 0000\ 0000\ 0000\ 1011$$

## Plus i minus beskonačno: $k = 255, m = 0$

Ako je  $k = 255$ , a mantisa jednaka 0, onda

•  $p = 0$  — prikaz  $+\infty$ , skraćena oznaka **+Inf**,

•  $p = 1$  — prikaz  $-\infty$ , skraćena oznaka **-Inf**.

Primjer: Prikaz broja  $+\infty$  ( $-\infty$ ) je

$$p = 0 \quad (p = 1)$$

$$k = 1111 \ 1111$$

$$m = 000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000$$

## Nije broj: $k = 255, m \neq 0$

Ako je  $k = 255$  i postoji bar jedan bit mantise različit od nule, onda je to signal da se radi o pogrešci (recimo dijeljenje s nulom, vađenje drugog korijena iz negativnog broja i sl.)

Tada se takva pogreška kodira znakom za Not a Number ili, skraćeno, s NaN:

$$p = 0$$

$$k = 1111 \ 1111$$

$$m = 000 \ 0000 \ 0000 \ 0101 \ 0000 \ 0000$$

# Greške zaokruživanja

Postoje realni brojevi koje **ne možemo egzaktno** spremiti u računalo, čak i kad su **unutar** prikazivog raspona brojeva. Takvi brojevi imaju **predugačku mantisu**.

**Primjer:** Realni broj

$$a = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$$

**ne može** se spremiti u realni broj jednostruke preciznosti **float** u C-u koji ima **23 + 1** znakova mantise, jer ima **25** znakova mantise.

Procesor tada pronalazi dva najbliža **prikaziva** susjeda broju  $a$  takva da vrijedi

$$a_- < a < a_+.$$



# Greške zaokruživanja (nastavak)

U našem primjeru je:

$$a = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$$

$$a_- = 1.0001\ 0000\ 1000\ 0011\ 1001\ 011$$

$$a_+ = 1.0001\ 0000\ 1000\ 0011\ 1001\ 100$$

Nakon toga, **zaokružuje** se rezultat. Zaokruživanje može biti:

- 🔴 prema najbližem broju (**standardno**, engl. **default**, za IA-32 procesore) — ako su dva susjeda jednako udaljena od  $a$ , izabire **parni** od ta dva broja,
- 🔴 prema dolje, tj. prema  $-\infty$ ,
- 🔴 prema gore, tj. prema  $\infty$ ,
- 🔴 prema nuli, tj. odbacivanjem “viška” znamenki.

## Greške zaokruživanja (nastavak)

Standardno zaokruživanje u našem primjeru:

$$a = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111$$

$$a_- = 1.0001\ 0000\ 1000\ 0011\ 1001\ 011$$

$$a_+ = 1.0001\ 0000\ 1000\ 0011\ 1001\ 100$$

Ovdje su  $a_-$  i  $a_+$  jednako udaljeni od  $a$ , pa je zaokruženi  $a$  jednak  $a_+$ , jer  $a_+$  ima **parni** zadnji bit (jednak je 0).

## Jedinična greška zaokruživanja

Dakle, ako je  $x \in \mathbb{R}$  unutar raspona brojeva prikazivih u računalu, onda se umjesto  $x$  sprema zaokruženi broj prikazivi  $fl(x)$ .

Time smo napravili **grešku zaokruživanja**  $\leq \frac{1}{2}$  “zadnjeg bita” mantise i taj broj se zove

☛ **jedinična greška zaokruživanja** (engl. unit roundoff).

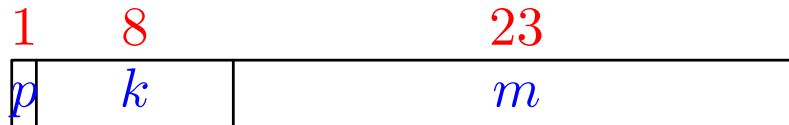
Standardna oznaka je  $u$ . Za **float** je  $u = 2^{-24} \approx 5.96 \cdot 10^{-8}$ .  
Vrijedi

$$fl(x) = (1 + \varepsilon)x, \quad |\varepsilon| \leq u,$$

gdje je  $\varepsilon$  relativna greška napravljena tim zaokruživanjem.  
Dakle, imamo **vrlo malu** relativnu grešku.

# Prikaz brojeva jednostruke točnosti — sažetak

IEEE tip `single` = `float` u C-u:



Vrijednost broja je

$$v = \begin{cases} (-1)^p * 2^{(k-127)} * (1.m) & \text{ako je } 0 < k < 255, \\ (-1)^p * 2^{(-126)} * (0.m) & \text{ako je } k = 0 \text{ i } m \neq 0, \\ (-1)^p * 0 & \text{ako je } k = 0 \text{ i } m = 0, \\ (-1)^p * \text{Inf} & \text{ako je } k = 255 \text{ i } m = 0, \\ \text{NaN} & \text{ako je } k = 255 \text{ i } m \neq 0. \end{cases}$$

## Raspon tipa float

Najveći prikazivi pozitivni broj je  $\text{FLT\_MAX} \approx 3.40282 \cdot 10^{38}$

$$p = 0$$

$$k = 1111\ 1110$$

$$m = 111\ 1111\ 1111\ 1111\ 1111\ 1111$$

Najmanji prikazivi normalizirani pozitivni broj je  
 $\text{FLT\_MIN} \approx 1.17549 \cdot 10^{-38}$

$$p = 0$$

$$k = 0000\ 0001$$

$$m = 000\ 0000\ 0000\ 0000\ 0000\ 0000$$

## Raspon tipa float

Najmanji prikazivi denormalizirani pozitivni broj je

$$\approx 1.4013 \cdot 10^{-45}$$

$$p = 0$$

$$k = 0000\ 0001$$

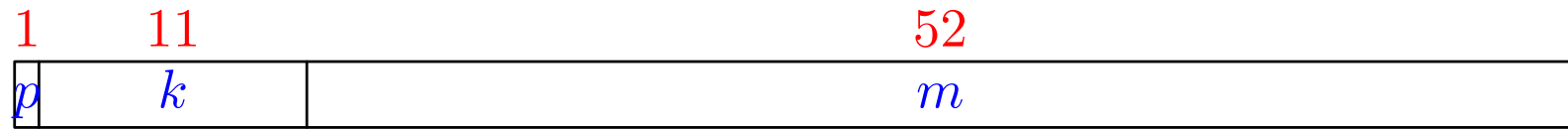
$$m = 000\ 0000\ 0000\ 0000\ 0000\ 0001$$

# Stvarni prikaz tipa double

“Srednji” realni tip je tzv. realni broj **dvostruke** točnosti — u C-u poznat kao **double**.

On ima sljedeća svojstva:

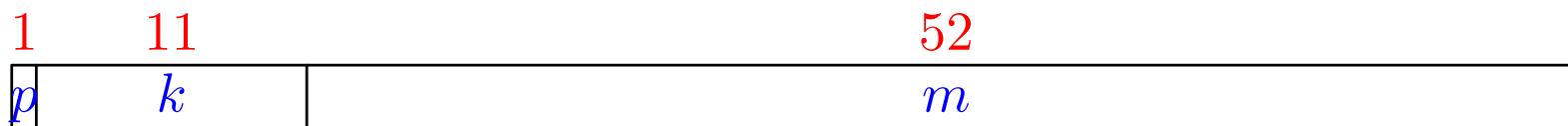
- Duljina: 8 byte-a (64 bita), podijeljen u **tri** polja.



- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- **stvarni eksponent** broja  $e$ ,  $e \in \{-1022, \dots, 1023\}$ ,
- **karakteristika**  $k = e + 1023$ , tako da je  $k \in \{1, \dots, 2046\}$ ,
- **karakteristike**  $k = 0$  i  $k = 2047$  — “posebna stanja”.

# Prikaz brojeva dvostruke točnosti — sažetak

IEEE tip `double` = `double` u C-u:



Vrijednost broja je

$$v = \begin{cases} (-1)^p * 2^{(k-1023)} * (1.m) & \text{ako je } 0 < k < 2047, \\ (-1)^p * 2^{(-1022)} * (0.m) & \text{ako je } k = 0 \text{ i } m \neq 0, \\ (-1)^p * 0 & \text{ako je } k = 0 \text{ i } m = 0, \\ (-1)^p * \text{Inf} & \text{ako je } k = 2047 \text{ i } m = 0, \\ \text{NaN} & \text{ako je } k = 2047 \text{ i } m \neq 0. \end{cases}$$

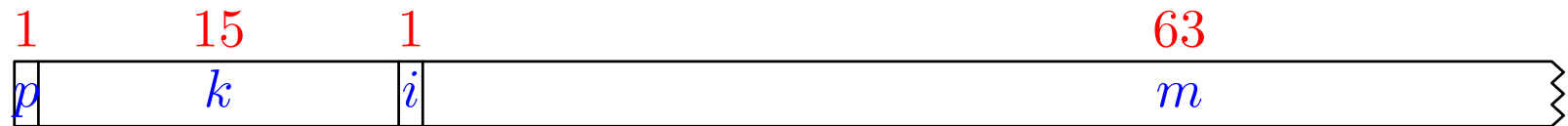


## Tip extended

Stvarno računanje (na IA-32) se obično radi u “proširenoj” točnosti — u C-u možda dohvatljiv kao **long double**.

On ima sljedeća svojstva:

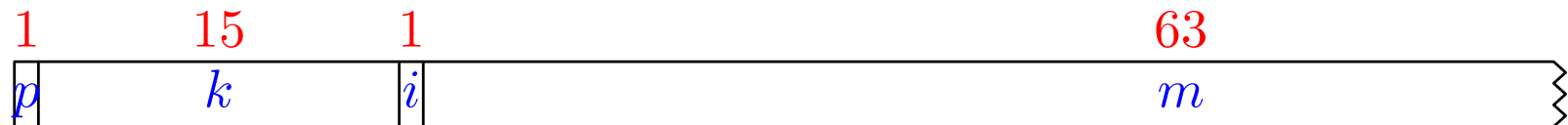
- Duljina: 10 byte-a (80 bita), podijeljen u četiri polja.



- u mantisi se pamti vodeći bit  $i$  mantise,
- stvarni eksponent broja  $e$ ,  $e \in \{-16382, \dots, 16383\}$ ,
- karakteristika  $k = e + 16383$ , tako da je  $k \in \{1, \dots, 32766\}$ ,
- karakteristike  $k = 0$  i  $k = 32767$  — “posebna stanja”.

# Prikaz brojeva proširene točnosti — sažetak

IEEE tip extended:



Vrijednost broja je

$$v = \begin{cases} (-1)^p * 2^{(k-16383)} * (i.m) & \text{ako je } 0 \leq k < 32767, \\ (-1)^p * \text{Inf} & \text{ako je } k = 32767 \text{ i } m = 0, \\ \text{NaN} & \text{ako je } k = 32767 \text{ i } m \neq 0. \end{cases}$$

Uočite da **prva** mogućnost uključuje:

•  $+0$ ,  $-0$  i **denormalizirane** brojeve (za  $k = 0$ ),  
jer se **pamti** vodeći “cjelobrojni” bit  $i$  mantise.

# Aritmetika računala

Aritmetika računala nije egzaktna. Za aritmetičku operaciju  $\circ$ , gdje je  $\circ$  jedna od operacija  $+$ ,  $-$ ,  $*$ ,  $/$ , zahtijeva se samo da ima malu relativnu grešku, tj.

$$fl(x \circ y) = (1 + \varepsilon)(x \circ y),$$

pri čemu je  $fl$  rezultat operacije dobiven računalom, a  $\varepsilon$  mali pozitivan broj.

Za aritmetiku računala **ne vrijedi**:

- asocijativnost zbrajanja i množenja,
- distributivnost množenja prema zbrajanju.

**Jedino** vrijedi:

- komutativnost za zbrajanje i množenje.

## Primjer neasocijativnosti zbrajanja

Primjer. **Asocijativnost** zbrajanja u računalu **ne vrijedi**.

Znamo (odn. uskoro ćete znati) da je tzv. **harmonijski** red

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{i} + \cdots$$

**divergentan**, tj. suma mu je “**beskonačna**”.

No, nitko nas ne spriječava da računamo konačne početne komade ovog reda, tj. **njegove parcijalne sume**

$$S_n := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}.$$

A kojim **redom** zbrajamo? (Zbrajanje je **binarna** operacija!)

## Primjer neasocijativnosti zbrajanja (nastavak)

U **realnim** brojevima je **potpuno svejedno** kojim poretkom zbrajanja računamo ovu sumu, jer vrijedi **asocijativnost**.

$$a + (b + c) = (a + b) + c = a + b + c.$$

Uostalom, sam zapis izraza **bez zagrada**

$$S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}$$

već “podrazumijeva” **asocijativnost**. U suprotnom, morali bismo **zagradama** naglasiti **poredak** operacija.

Ovdje imamo točno  $n - 1$  **binarnih operacija zbrajanja**, i možemo ih napraviti **kojim redom hoćemo**.

## Primjer neasocijativnosti zbrajanja (nastavak)

Drugim riječima, u prethodni izraz za  $S_n$

- možemo rasporediti zagrade na bilo koji način, samo da svi plusevi budu “binarni”, tj. zbrajaju dva objekta, a objekt je broj ili (podizraz u zagradama).

Na pr., zbrajanju “unaprijed” odgovara raspored zagrada

$$S_{n,1} := \left( \dots \left( \left( 1 + \frac{1}{2} \right) + \frac{1}{3} \right) + \dots + \frac{1}{n-1} \right) + \frac{1}{n},$$

a zbrajanju “unatrag” odgovara raspored zagrada

$$S_{n,2} := 1 + \left( \frac{1}{2} + \left( \frac{1}{3} + \dots + \left( \frac{1}{n-1} + \frac{1}{n} \right) \dots \right) \right).$$

## Primjer neasocijativnosti zbrajanja (nastavak)

Koliko takvih rasporeda zagrada ima — bit će napravljeno u Kombinatorici. Bitno je da svi daju **isti rezultat**.

**Komutativnost** nam uopće **ne treba**. Ako i nju iskoristimo, dobivamo još puno više načina za računanje ove sume, i svi, naravno, opet daju **isti rezultat**.

Izračunajmo **aritmetikom računala** navedene **dvije** sume

•  $S_{n,1}$  — unaprijed, i

•  $S_{n,2}$  — unatrag,

za  $n = 1\,000\,000$ , u **tri** standardne IEEE točnosti **single**, **double** i **extended**. Preciznije, koristimo ova tri tipa za prikaz brojeva, uz pripadne aritmetike za računanje.

## Primjer neasocijativnosti zbrajanja (nastavak)

Uz skraćene oznake  $S_1$  i  $S_2$  za varijable u kojima zbrajamo pripadne sume, odgovarajući algoritmi za zbrajanje su:

● unaprijed

$$S_1 := 1,$$

$$S_1 := S_1 + \frac{1}{i}, \quad i = 2, \dots, n,$$

● unatrag

$$S_2 := \frac{1}{n},$$

$$S_2 := \frac{1}{i} + S_2, \quad i = n - 1, \dots, 1.$$

Dakle, zaista ne koristimo komutativnost zbrajanja.



## Primjer neasocijativnosti zbrajanja (nastavak)

Dobiveni rezultati za sume  $S_1$ ,  $S_2$  i pripadne relativne greške su:

tip i suma	vrijednost	rel. greška
single $S_1$	14.3573579788208008	$2.45740E-0003$
single $S_2$	14.3926515579223633	$5.22243E-0006$
double $S_1$	14.3927267228647810	$6.54899E-0014$
double $S_2$	14.3927267228657545	$-2.14449E-0015$
extended $S_1$	14.3927267228657234	$1.91639E-0017$
extended $S_2$	14.3927267228657236	$-1.08475E-0018$

Slovo  $E$  u brojevima zadnjeg stupca znači “puta 10 na”, pa je, na pr.,  $-1.08475E-0018 = -1.08475 \times 10^{-18}$ .

## Primjer neasocijativnosti zbrajanja (nastavak)

Izračunate vrijednosti  $S_1$  i  $S_2$  su različite (u sve tri točnosti). Dakle, zbrajanje brojeva u aritmetici računala očito **nije asocijativno**.

Primijetite da, u sve tri točnosti, **zbrajanje unatrag**  $S_2$  daje nešto **točniji** rezultat. To **nije slučajno**.

Svi brojevi koje zbrajamo su **istog predznaka** pa zbroj stalno **raste**, bez obzira na poredak zbrajanja.

- Kad zbrajamo unatrag — **od manjih** brojeva **prema većim**, zbroj se **pomalo** “nakuplja”.
- Obratno, kad zbrajamo unaprijed — **od velikih** brojeva **prema manjim**, zbroj puno **brže naraste**. Onda mali dodani član **jedva utječe** na rezultat (tj. dobar dio znamenki pribrojnika nema utjecaj na sumu).

## Primjer katastrofalnog kraćenja

Zakruživanjem ulaznih podataka dolazi do male relativne greške. Kako ona može utjecati na konačan rezultat?

**Primjer.** Uzmimo realnu aritmetiku “računala” u bazi 10. Za mantisu (značajni dio) imamo  $t = 4$  dekadске znamenke, a za eksponent  $s = 2$  znamenke (što nije bitno). Neka je

$$\begin{aligned}x &= 8.8866 = 8.8866 \times 10^0, \\y &= 8.8844 = 8.8844 \times 10^0.\end{aligned}$$

Umjesto brojeva  $x$  i  $y$  (koji nisu prikazivi), u “memoriju” spremamo brojeve  $fl(x)$  i  $fl(y)$ , pravilno zaokružene na  $t = 4$  znamenke

$$\begin{aligned}fl(x) &= 8.887 \times 10^0, \\fl(y) &= 8.884 \times 10^0.\end{aligned}$$

## Primjer katastrofalnog kraćenja (nastavak)

Ovim zaokruživanjem smo napravili **malu** relativnu grešku (ovdje je  $u = 5 \times 10^{-5}$ ).

Razliku  $f_l(x) - f_l(y)$  računamo tako da **izjednačimo eksponente** (što već jesu), **oduzmemo** značajne dijelove (mantise), pa **normaliziramo**

$$\begin{aligned} f_l(x) - f_l(y) &= 8.887 \times 10^0 - 8.884 \times 10^0 \\ &= 0.003 \times 10^0 = 3.??? \times 10^{-3}. \end{aligned}$$

Kod normalizacije, zbog pomaka “**ulijevo**”, pojavljuju se

● **?** = znamenke koje više **ne možemo** restaurirati (ta informacija se izgubila).

Što sad?

## Primjer katastrofalnog kraćenja (nastavak)

Računalo radi **isto** što bismo i mi napravili:

🔴 na ta mjesta ? upisuje 0.

**Razlog:** da rezultat bude **točan**, ako su ulazni brojevi točni. Dakle, ovo oduzimanje je **egzaktno** i u aritmetici računala.

Konačni rezultat je  $fl(x) - fl(y) = 3.000 \times 10^{-3}$ .

Pravi rezultat je

$$\begin{aligned}x - y &= 8.8866 \times 10^0 - 8.8844 \times 10^0 \\ &= 0.0022 \times 10^0 = 2.2 \times 10^{-3}.\end{aligned}$$

Već **prva** značajna znamenka u  $fl(x) - fl(y)$  je **pogrešna**, a relativna greška **ogromna**! Uočite da je ta znamenka (**3**) ujedno i **jedina** koja nam je ostala — sve ostalo se skratilo!

## Primjer katastrofalnog kraćenja (nastavak)

Prava **katastrofa** se događa ako  $3.??? \times 10^{-3}$  uđe u naredna zbrajanja (oduzimanja), a onda se **skrati** i ta trojka!

Uočite da je **oduzimanje**  $fl(x) - fl(y)$  bilo **egzaktno** (a egzaktno je i u aritmetici računala), ali **rezultat je pogrešan**.

Krivac, očito, nije **oduzimanje** (kad je egzaktno).

- Uzrok su **polazne greške** u operandima.

Ako njih **nema**, tj. ako su operandi **egzaktni**,

- i dalje (naravno) dolazi do **kraćenja**,

- ali je rezultat (uglavnom, a po IEEE standardu sigurno) **egzaktan**,

pa se ovo kraćenje zove **benigno kraćenje**.

# Kvadratna jednadžba

Uzmimo da treba riješiti (realnu) kvadratnu jednadžbu

$$ax^2 + bx + c = 0,$$

gdje su  $a$ ,  $b$  i  $c$  zadani i  $a \neq 0$ .

Matematički gledano, problem je lagan: imamo 2 rješenja

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Numerički gledano, problem je mnogo izazovniji:

- ni uspješno računanje po ovoj formuli,
- ni točnost izračunatih korijena,

ne možemo uzeti “zdravo za gotovo”.

## Kvadratna jednadžba (nastavak)

Primjer:  $x^2 - 56x + 1 = 0$ . U aritmetici s 5 decimala dobijemo

$$x_1 = \frac{56 - \sqrt{3132}}{2} = \frac{56 - 55.964}{2} = 0.018000,$$

$$x_2 = \frac{56 + \sqrt{3132}}{2} = \frac{56 + 55.964}{2} = 55.982.$$

Točna rješenja su

$$x_1 = 0.0178628\dots \quad \text{i} \quad x_2 = 55.982137\dots$$

Manji od ova dva korijena ima samo dvije točne znamenke (kraćenje).



## Kvadratna jednadžba (popravak)

Prvo izračunamo većeg po apsolutnoj vrijednosti, po formuli

$$x_2 = \frac{-(b + \text{sign}(b)\sqrt{b^2 - 4ac})}{2a},$$

a manjeg po apsolutnoj vrijednosti, izračunamo iz

$$x_1 \cdot x_2 = \frac{c}{a}$$

(Vieta), tj.

$$x_1 = \frac{c}{x_2 a}.$$

Opasnog kraćenja za  $x_1$  više nema!

# Kvadratna jednadžba (nastavak)

Ovo je bila samo **jedna**, od (barem) **tri** “opasne” točke za računanje. Preostale dvije su:

- 🔴 “kvadriranje” pod korijenom — mogućnost za **overflow**.  
Rješenje — “skaliranjem”.
- 🔴 oduzimanje u diskriminanti (kraćenje) — nema jednostavnog rješenja.
- 🔴 To je odraz **nestabilnosti** problema, jer tad imamo **dva bliska korijena** koji su **osjetljivi** na male perturbacije koeficijenata (na pr. pomak  $c$  “gore–dolje”).