Programiranje 1
4. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Cijeli brojevi — prikaz i aritmetika (nastavak):
  - Dijeljenje cijelih brojeva s predznakom — cjelobrojni kvocijent i ostatak.
  - 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 jednadžbe.

Informacije — Odrada sutra

Odrađivanje zadnja dva predavanja završava sutra,

- u subotu, 3. 10., od 12–14 u (003).

Novi termin za konzultacije je (konačno):

- ponedjeljak, 14–16 sati.

Informacije

Ne zaboravite da treba:

- otvoriti korisnički račun u Računskom centru.
- Računi se “preuzimaju” u centru, utorkom i četvrtkom, od 12:30 do 15 sati.

Promijenite password!

Nadalje, treba:

- obaviti prijavu i dobiti potvrdu prijave

u aplikaciji za tzv. “domaće zadaće”, na web–adresi

`http://degiorgi.math.hr/prog1/ku/`

Informacije — nastavak

Bitno: Prilikom prijave za "ku",

- svoje podatke trebate upisati korektno,

što (između ostalog) znači i

- korištenje hrvatskih slova u imenu i prezimenu!

Studenti koji su upisali "czsdj" varijantu imena i prezimena neka se jave e-mailom asistentu V. Šegi na adresu

`vsego@math.hr`

i napišu

- svoj JMBAG i ispravno ime i prezime.

Napomene o sigurnosti — lozinka (password)

Tom prilikom, zaista nije nužno da (onako usput)

- pošaljete i svoju lozinku, tj. password!

Upravo suprotno: nemojte to raditi!

Za početak, za sve što se radi pod nekim korisničkim računom ili accountom,

- odgovoran je vlasnik tog računa!

Ne vrijedi isprika da vam se netko drugi logirao umjesto vas!

A sve što vam treba za logiranje na račun su dvije stvari:

- korisničko ime ili username — koji se vidi kad ga kucate,
- lozinka iliti password — koji se ne vidi, i to s razlogom.

Napomene o sigurnosti (nastavak)

Zapamtite: password vam je

- jedina zaštita od "nezvanih" korisnika.

Zato nemojte koristiti "početni" password, već ga

- promijenite na nešto "tajno", što zaista samo vi znate.

I, na kraju rada,

- uvijek treba uredno završiti rad, tj. "odlogirati" se.

Što mislite — kako funkcionira "on–line" kupovina!?

Prikaz cijelih brojeva s predznakom (nastavak)

Dijeljenje cijelih brojeva s predznakom

Prošli puta smo uveli operacije div (cjelobrojni kvocijent) i 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 div i mod definiramo relacijama

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

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 compilera.
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,

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

- ostatak ima isti predznak kao i a.

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

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. Noviji standard za C (tzv. C99) propisuje ovakvo ponašanje.

Cijeli brojevi s predznakom — sažetak

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

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

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

- nenegativni brojevi $B = 0, \ldots, 2^{n-1}-1$ imaju isti prikaz kao i bez predznaka,
- negativni brojevi $B = -1, \ldots, -2^{n-1}$ 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 div i 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 — dodjeljivanje

Primjer. Modularna aritmetika kod dodjeljivanja.

```
#include <stdio.h>

int main(void) {
    int broj;

    broj = 5;          printf(" broj = %d\n", broj);
    broj = 2000000000; printf(" broj = %d\n", broj);
    broj = 4000000000; printf(" broj = %d\n", broj);
    broj = 8000000000; printf(" broj = %d\n", broj);

    return 0;
}
```

Cijeli brojevi — dodjeljivanje (nastavak)

U tipu `int`, broj bitova za prikaz brojeva je $n = 32$.

Izlaz programa je:

```
broj = 5
broj = 2000000000
broj = -294967296
broj = -589934592
```

Cijeli brojevi — čitanje

Primjer. Modularna aritmetika kod čitanja.

```
#include <stdio.h>

int main(void) {
    int broj;

    scanf("%d", &broj);
    printf(" ucitani broj = %d\n", broj);

    return 0;
}
```

Za ulaz `4000000000`,
izlaz programa je: `ucitani broj = -294967296`.

Aritmetika cijelih brojeva: klasične greške

Cijeli brojevi — klasične greške

Primjer. Računanje $n!$ u cjelobrojnoj aritmetici.

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

$$
\begin{aligned}
1! &= 1, \\
n! &= n \cdot (n-1)!, \qquad n \geq 2.
\end{aligned}
$$

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

$$\begin{aligned} 50! = {} & 30414\ 09320\ 17133\ 78043\ 61260\ 81660 \\ & 64768\ 84437\ 76415\ 68960\ 51200\ 00000\ 00000, \end{aligned}$$

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

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

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

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

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

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

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

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

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 ℓ 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):

$$67800000000.0 \qquad 0.000002078$$

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:

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

Prikaz realnih brojeva

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

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

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

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

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) $\Longrightarrow$ 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

- 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 pozitivnog 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.

1	8	23
p	k	m

- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- stvarni eksponent broja e, $e \in \{-126, \ldots, 127\}$,
- karakteristika $k = e + 127$, tako da je $k \in \{1, \ldots, 254\}$,
- 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:

$$
\begin{aligned}
p &= 0 \\
k &= e + 127 = (130)_{10} = (2^7 + 2^1)_{10} = 1000\ 0010 \\
m &= 0100\ 1000\ 0000\ 0000\ 0000\ 000
\end{aligned}
$$

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:

$$\begin{aligned} p &= 0 \\ k &= 0000\;0000 \\ m &= 000\;0000\;0000\;0000\;0000\;1011 \end{aligned}$$

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

$$
\begin{aligned}
p &= 0 \qquad (p = 1) \\
k &= 1111\ 1111 \\
m &= 000\ 0000\ 0000\ 0000\ 0000\ 0000
\end{aligned}
$$

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`.

Primjer.

$$p = 0$$

$$k = 1111\ 1111$$

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

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 (u binarnom zapisu)

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

ima 25 znamenki mantise i ne može se egzaktno spremiti u realni broj jednostruke preciznosti `float` u C-u, koji ima $23 + 1$ znamenki za mantisu.

Procesor tada pronalazi dva najbliža prikaziva susjeda a_-, a_+, 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 (zadnji bit je 0),
- prema dolje, tj. prema $-\infty$,
- prema gore, tj. prema ∞,
- 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

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

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

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

gdje je ε relativna greška napravljena tim zaokruživanjem. Dakle, imamo vrlo malu relativnu grešku.

Prikaz brojeva jednostruke točnosti — sažetak

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

1	8	23
p	k	m

Vrijednost broja je

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

Raspon tipa float

Najveći prikazivi pozitivni broj je
`FLT_MAX` $\approx 3.402823466 \cdot 10^{38}$, s prikazom

$$p = 0$$

$$k = 1111\ 1110$$

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

Najmanji prikazivi normalizirani pozitivni broj je
`FLT_MIN` $\approx 1.175494351 \cdot 10^{-38}$, s prikazom

$$p = 0$$

$$k = 0000\ 0001$$

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

Raspon tipa float

Simboličke konstante `FLT_MAX`, `FLT_MIN` i još poneke vezane uz tip `float`, definirane su u datoteci zaglavlja `float.h` i mogu se koristiti u C programima.

Uočite:

- 1/`FLT_MIN` je egzaktno prikaziv (nađite prikaz),
- 1/`FLT_MAX` nije egzaktno prikaziv i zalazi u denormalizirane brojeve (tzv. “gradual underflow”).

Najmanji prikazivi denormalizirani pozitivni broj je $2^{-126} \cdot 2^{-23} = 2^{-149} \approx 1.4013 \cdot 10^{-45}$, s prikazom

$$p = 0$$

$$k = 0000\ 0000$$

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

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.

1	11	52
p	k	m

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

Prikaz brojeva dvostruke točnosti — sažetak

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

1	11	52
p	k	m

Vrijednost broja je

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

Jedinična greška i raspon tipa double

Jedinična greška zaokruživanja za `double` je

$$u = 2^{-53} \approx 1.11 \cdot 10^{-16}.$$

Broj $1 + 2u$ je najmanji prikazivi broj strogo veći od 1. Postoji

$$\texttt{DBL\_EPSILON} = 2u \approx 2.220446049250313 \cdot 10^{-16}.$$

Najveći prikazivi pozitivni broj je

$$\texttt{DBL\_MAX} \approx 1.797693134862315 \cdot 10^{308}.$$

Najmanji prikazivi normalizirani pozitivni broj je

$$\texttt{DBL\_MIN} \approx 2.225073858507201 \cdot 10^{-308}.$$

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.

1	15	1	63
p	k	i	m

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

Prikaz brojeva proširene točnosti — sažetak

IEEE tip extended:

1	15	1	63
p	k	i	m

Vrijednost broja je

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

Uočite da prva mogućnost uključuje:

- $+0$, -0 i denormalizirane brojeve (za $k = 0$),

jer se pamti vodeći "cjelobrojni" bit i mantise.

Realna aritmetika računala (IEEE standard)

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 ε 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

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

a zbrajanju "unatrag" odgovara raspored zagrada

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

Primjer neasocijativnosti zbrajanja (nastavak)

Koliko takvih rasporeda zagrada ima — bit će napravljeno u Diskretnoj matematici. 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 artimetike 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

$$S_1 := 1,$$

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

- unatrag

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

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

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	14.357357978820800**8**	$2.45740E{-}0003$
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 $t = 4$ dekadske znamenke, a za eksponent $s = 2$ znamenke (što nije bitno). Neka je

$$x = 8.8866 = 8.8866 \times 10^0,$$
$$y = 8.8844 = 8.8844 \times 10^0.$$

Umjesto brojeva x i y (koji nisu prikazivi), u "memoriju" spremamo brojeve $f\ell(x)$ i $f\ell(y)$, pravilno zaokružene na $t = 4$ znamenke

$$f\ell(x) = 8.887 \times 10^0,$$
$$f\ell(y) = 8.884 \times 10^0.$$

Primjer katastrofalnog kraćenja (nastavak)

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

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

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

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 $f\ell(x) - f\ell(y) = 3.000 \times 10^{-3}$.

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 $f\ell(x) - f\ell(y)$ je pogrešna, a relativna greška je 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 $f\ell(x) - f\ell(y)$ 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 vrijedi $a \neq 0$.

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 — problem

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

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

Manji od ova dva korijena — x_1, ima samo dvije točne znamenke (kraćenje).

Kvadratna jednadžba — popravak

Prvo izračunamo većeg po apsolutnoj vrijednosti, po formuli

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

a manjeg po apsolutnoj vrijednosti, izračunamo iz

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

(Vieta), tj. formula za x_1 je

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

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 jednadžbe.
  - Na primjer, pomak c = pomak grafa "gore–dolje".

Prikaz cijelih brojeva u računalu

Sadržaj

- Cijeli brojevi — prikaz u računalu:
  - Prikaz brojeva bez predznaka.
  - Prikaz brojeva s predznakom, komplementiraj i dodaj 1.

Zapis prirodnog broja u bazi 2

Neka je $B \in \mathbb{N}$ neki prirodan broj.

Tzv. pozicioni zapis broja B u bazi $b = 2$ ima sljedeći oblik:

$$B = b_k \cdot 2^k + \cdots + b_1 \cdot 2 + b_0, \quad b_i \in \{0, 1\},$$

gdje su b_i binarne znamenke ili bitovi broja B.

Da bismo dobili jednoznačnost prikaza, koristimo da je $B > 0$ i

- ne dozvoljavamo da vodeće znamenke budu nule,

tj. zahtijevamo da vodeća znamenka b_k mora biti pozitivna

$$b_k > 0.$$

Normalizirani zapis prirodnog broja u bazi 2

Prikaz oblika

$$B = b_k \cdot 2^k + \cdots + b_1 \cdot 2 + b_0, \quad b_i \in \{0, 1\}, \quad b_k > 0,$$

zovemo normalizirani pozicioni prikaz broja B u bazi $b = 2$.
Skraćeni zapis ovog prikaza je

$$B = (b_k b_{k-1} \ldots b_1 b_0)_2.$$

Broj $k \in \mathbb{N}_0$ jednoznačno je određen zahtjevom $b_k > 0$ i vrijedi

$$k = \lfloor \log_2 B \rfloor.$$

Broj binarnih znamenki ("duljina") broja B je

$$k + 1 = \lfloor \log_2 B \rfloor + 1.$$

Računanje znamenki broja u bazi 2

Binarne znamenke b_i zadanog broja B

$$B = b_k \cdot 2^k + \cdots + b_1 \cdot 2 + b_0 = (b_k b_{k-1} \ldots b_1 b_0)_2,$$

dobivamo dijeljenjem s bazom 2.

- Preciznije, koristimo cjelobrojni kvocijent i ostatak.

Kako to ide? Direktno iz Euklidovog teorema, zapisom

$$B = (b_k \cdot 2^{k-1} + \cdots + b_1) \cdot 2 + b_0 = \text{oznaka} = B_1 \cdot 2 + b_0.$$

Zadnju znamenku b_0 dobivamo kao ostatak pri dijeljenju broja B s bazom 2

$$b_0 = B \bmod 2.$$

Računanje znamenki broja u bazi 2 (nastavak)

Cjelobrojni kvocijent broja B s bazom 2 je "novi" broj

$$B_1 = B \text{ div } 2 = b_k \cdot 2^{k-1} + \cdots + b_1.$$

Njegov zapis u bazi 2 je

$$B_1 = (b_k b_{k-1} \ldots b_1)_2,$$

tj. dobiva se "brisanjem" znamenke b_0 iz zapisa broja B.

Znamenku b_0 smo upravo našli, pa je dovoljno naći binarni zapis broja B_1, a taj broj ima jednu znamenku manje.

Naravno, njegovu zadnju znamenku b_1 nalazimo

- ponavljanjem opisanog postupka, ali na broju B_1.

Računanje znamenki broja u bazi 2 (nastavak)

Čitav postupak možemo zapisati na sljedeći način.

Neka je, na početku, $B_0 := B$.

U općem — i–tom koraku, krećemo s brojem B_i i računamo:

- ostatak — izdvoji (trenutnu) zadnju znamenku u B_i

$$b_i = B_i \bmod 2,$$

- cjelobrojni kvocijent — "obriši" (trenutnu) zadnju znamenku u B_i

$$B_{i+1} = B_i \operatorname{div} 2,$$

tako da uvijek vrijedi

$$B_i = B_{i+1} \cdot 2 + b_i, \quad i = 0, 1, \ldots \ .$$

Računanje znamenki broja u bazi 2 (nastavak)

Pitanje je samo — kad treba stati (jer k ne znamo unaprijed)?

Odgovor: kad "obrišemo" sve znamenke iz broja B, ostaje nam nula.

Uočite da je

$$B_i = b_k \cdot 2^{k-i} + \cdots + b_i = (b_k b_{k-1} \ldots b_i)_2.$$

Za $i = k$ imamo $B_k = b_k > 0$. Nakon dijeljenja dobivamo

$$B_k = B_{k+1} \cdot 2 + b_k, \quad B_{k+1} = 0.$$

Dakle, postupak staje kad prvi put dobijemo kvocijent nula,

$$B_{k+1} = B_k \text{ div } 2 = 0.$$

Prikaz broja 1717 *u bazi* 2

Primjer. Prikaz broja 1717 u bazi 2 dobivamo ovako:

i	B_i	$B_{i+1} = B_i \text{ div } 2$	$b_i = B_i \bmod 2$
0	$1717 = 858 \cdot 2 + 1$	$1717 \text{ div } 2 = 858$	$1717 \bmod 2 = 1$
1	$858 = 429 \cdot 2 + 0$	$858 \text{ div } 2 = 429$	$858 \bmod 2 = 0$
2	$429 = 214 \cdot 2 + 1$	$429 \text{ div } 2 = 214$	$429 \bmod 2 = 1$
3	$214 = 107 \cdot 2 + 0$	$214 \text{ div } 2 = 107$	$214 \bmod 2 = 0$
4	$107 = 53 \cdot 2 + 1$	$107 \text{ div } 2 = 53$	$107 \bmod 2 = 1$
5	$53 = 26 \cdot 2 + 1$	$53 \text{ div } 2 = 26$	$53 \bmod 2 = 1$
6	$26 = 13 \cdot 2 + 0$	$26 \text{ div } 2 = 13$	$26 \bmod 2 = 0$
7	$13 = 6 \cdot 2 + 1$	$13 \text{ div } 2 = 6$	$13 \bmod 2 = 1$
8	$6 = 3 \cdot 2 + 0$	$6 \text{ div } 2 = 3$	$6 \bmod 2 = 0$
9	$3 = 1 \cdot 2 + 1$	$3 \text{ div } 2 = 1$	$3 \bmod 2 = 1$
10	$1 = 0 \cdot 2 + 1$	$1 \text{ div } 2 = 0$	$1 \bmod 2 = 1$

Prikaz broja 1717 *u bazi* 2 *(nastavak)*

Dobili smo da je $B_{11} = 0$, pa je $k = 10$.

Rješenje. Prikaz broja 1717 u bazi 2 je

$$(1717)_{10} = (11010110101)_2$$

i ima 11 binarnih znamenki.

Prikaz broja 1717 *kao* `int`

Primjer. Prikaz broja 1717 kao `int` i `unsigned int`.

Rješenje. Prikazu broja 1717 u bazi 2 samo treba dodati potreban broj vodećih nula, do broja bitova predviđenog za prikaz u odgovarajućem tipu.

- Za tipove `int` i `unsigned int`, broj bitova je $n = 32$.

Prikaz broja 1717 u tim tipovima je

```
0000 0000 0000 0000 0000 0110 1011 0101
```

Prikaz broja -1717 kao `int`

Primjer. Prikaz broja -1717 kao `int`.

Krećemo od prikaza broja 1717 kao `int`:

```
 0000 0000 0000 0000 0000 0110 1011 0101
```

Komplementiramo (bit–po–bit) i dodamo 1 (modulo 2^{32}):

```
 1111 1111 1111 1111 1111 1001 0100 1010
+                                       1
```

Rezultat je prikaz broja -1717 kao `int`:

```
 1111 1111 1111 1111 1111 1001 0100 1011
```

Binarni zapis broja 0.1

Primjer. Binarni zapis broja 0.1 je beskonačan

$$
\begin{aligned}
(0.1)_{10} &= (0.0\ 0011\ 0011\ 0011\dots)_2 \\
&= (1.1001\ 1001\ 1001\dots)_2 \cdot 2^{-4}.
\end{aligned}
$$

Prikaz broja 0.1 *kao* `float`

Primjer. Prikaz broja 0.1 kao `float`.

$$(0.1)_{10} = (1.1001\ 1001\ \ldots\ 1001\ 1001\ldots)_2 \cdot 2^{-4}.$$

Zaokruživanje iza 23 bita razlomljenog dijela daje

$$p = 0$$

$$k = e + 127 = -4 + 127 = (123)_{10} = 0111\ 1011$$

$$m = 1001\ 1001\ 1001\ 1001\ 1001\ 101$$

```
Prikaz broja      0.100 [float] u racunalu:
            0 01111011 10011001100110011001101
  1. rijec: 0011 1101 1100 1100 1100 1100 1100 1101
```

Prikaz broja 0.1 *kao* `double`

Primjer. Prikaz broja 0.1 kao `double`.

$$(0.1)_{10} = (1.1001\ 1001\ \ldots\ 1001\ 1001\ldots)_2 \cdot 2^{-4}.$$

Zaokruživanje iza 52 bita razlomljenog dijela daje

$$p = 0$$

$$k = e + 1023 = -4 + 1023 = (1019)_{10} = 011\ 1111\ 1011$$

$$\begin{aligned} m = {} & 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001 \\ & 1001\ 1001\ 1001\ 1001\ 1001\ 1010 \end{aligned}$$

```
Prikaz broja      0.100 u racunalu:
  1. rijec: 1001 1001 1001 1001 1001 1001 1001 1010
  2. rijec: 0011 1111 1011 1001 1001 1001 1001 1001
```
