Programiranje 1
4. predavanje

Saša Singer

singer@math.hr

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

PMF – Matematički odsjek, 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 u računalu — IEEE standard:
  - Osnovni oblik "floating–point" prikaza — mantisa i eksponent.
  - IEEE standard — tipovi: single, double, extended.
  - Greške zaokruživanja u prikazu.
  - Pojam "jedinične greške zaokruživanja".

Sadržaj predavanja (nastavak)

- Realna aritmetika računala — IEEE standard:
  - Greške zaokruživanja osnovnih aritmetičkih operacija.
  - Relativna točnost izračunatog rezultata.
  - "Širenje" grešaka zaokruživanja.
- Primjeri širenja grešaka i izbjegavanja grešaka:
  - Parcijalne sume harmonijskog reda.
  - Opasno ili "katastrofalno" kraćenje.
  - Korijeni kvadratne jednadžbe.
- Greške u praksi:
  - Promašaj raketa Patriot.

Informacije — odrada, kolokviji

Termin odrade predavanja 21. 12. je sutra:

- subota, 29. 9., od 11–13 u (003).

Programiranje 1 je u kolokvijskom razredu F3.

Službeni termini svih kolokvija su:

- Prvi kolokvij: petak, 16. 11. 2012., u 15 sati.
- Drugi kolokvij: petak, 18. 1. 2013., u 15 sati.
- Popravni kolokvij: petak, 1. 2. 2013., u 15 sati.

Informacije — dodaci ovom predavanju

Na webu imate dva dodatka ovom predavanju:

- Cijeli i realni brojevi — binarni prikaz i prikaz u računalu (algoritmi i primjeri),
- Aritmetika realnih brojeva i širenje grešaka zaokruživanja.

Nije obavezno, ali isplati se pogledati — za dopunu znanja.

Informacije

Ne zaboravite da treba:

- preuzeti korisnički račun u Računskom centru.
- Računi se "preuzimaju" svaki dan, od 12:30 do 15 sati.

Promijenite password!

Nadalje, treba:

- obaviti prijavu i, zatim, potvrditi prijavu

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 znakova u imenu i prezimenu!

Studenti koji su upisali “czsdj” varijantu imena i prezimena neka se jave e-mailom asistentu Z. Bujanoviću na adresu

`zbujanov@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

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

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

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.

Tek u novije vrijeme postoji dogovoreni standard (tzv. C99)

- za proširenje operacija div i mod na $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$.

Dijeljenje cijelih brojeva — što je problem?

Primjer. Neka je $a = 5$ i $b = 3$. Onda je

- $5 = 1 \cdot 3 + 2$, pa je kvocijent $q = 1$ i ostatak $r = 2$.

Neka je sad $a = -5$ i $b = 3$. Za kvocijent q i ostatak r mora vrijediti $a = q \cdot b + r$. Ostaje izbor/restrikcija ostatka r.

U Euklidovom teoremu je uvijek $r \geq 0$, tj. $r \in \mathbb{Z}_b$. Onda je

- $-5 = -2 \cdot 3 + 1$, pa je $q = -2$ i $r = 1$.

To nema "nikakve veze" s prethodnim rezultatima za pripadne apsolutne vrijednosti!

Međutim, ako dozvolimo negativni ostatak $r \in -\mathbb{Z}_b$, onda je

- $-5 = -1 \cdot 3 - 2$, pa je $q = -1$ i $r = -2$.
- Apsolutne vrijednosti kvocijenta i ostatka ostaju iste! ■

Dijeljenje cijelih brojeva — izbor ostatka

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

Zato, u većini programskih jezika (uključivo i C) vrijedi da

- stvar radi očekivano, tj. prema standardnom 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.

Međutim, kod brojeva s predznakom imamo i negativne brojeve, pa (možda) ima smisla dozvoliti

- da i ostaci budu negativni, u nekim slučajevima.

Odluka, tj. izbor ostataka — ovisi o primjeni i željenim svojstvima rezultata!

Dijeljenje cijelih brojeva — izbor ostatka

Pripadni "Euklidov" teorem smije imati i kompliciraniju formulaciju, ako je to korisno u praksi.

- Pitanje je samo je li izbor ostataka propisan ili ne!!!

Nažalost, za negativne operande, ponašanje dijeljenja

- ne mora biti precizno definirano!

Na primjer, stari C90 standard (knjiga KR2) 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!

Srećom, novi C99 standard precizno propisuje izbor ostataka.

Dijeljenje cijelih brojeva s predznakom

Eksperiment:

- test–program `divmod.c` (v. sljedeća stranica),
- Intel C++ compiler, `gcc` compiler (Code::Blocks).

Rezultati $q = a \operatorname{div} b$ i $r = a \bmod b$ 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 $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$.

Test–program divmod.c

```
#include <stdio.h>

int main(void) {
    char s[] =
        " a = %2d, b = %2d, q = %2d, r = %2d\n";
    int a, b;

    a =  5; b =  3;  printf(s, a, b, a/b, a%b);
    a = -5; b =  3;  printf(s, a, b, a/b, a%b);
    a =  5; b = -3;  printf(s, a, b, a/b, a%b);
    a = -5; b = -3;  printf(s, a, b, a/b, a%b);

    return 0;
}
```

Veza cjelobrojnog i običnog dijeljenja (standard)

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:

- ako je $a \geq 0$, onda je $r \in \mathbb{Z}_b$, tj. $0 \leq r < |b|$,
- ako je $a < 0$, onda je $r \in -\mathbb{Z}_b$, tj. $-|b| < r \leq 0$.

Ovo je "Euklidov teorem" za cijele brojeve u C-u!

Dijeljenje cijelih brojeva (standard)

Novi C99 standard propisuje ovakav izbor ostataka, tj.

- ovakvo ponašanje cjelobrojnog dijeljenja za cijele brojeve s predznakom.

Zato zaboravite raniju definiciju, iako se "novi" mod ponaša drugačije nego u matematici.

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 (misli se i na ostale programske jezike).

Cijeli brojevi s predznakom — sažetak

Ako imamo n bitova za prikaz brojeva, onda je skup svih prikazivih cijelih brojeva s predznakom jednak

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

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 B 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, za svaki slučaj, treba provjeriti kako se dobiva proširenje ovih operacija s $\mathbb{N}_0 \times \mathbb{N}$ na $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$,
- tj. radi li compiler prema C99 standardu.

Cijeli brojevi u C-u — sažetak

U programskom jeziku C:

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

Ovaj tip postoji 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:

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

Napišimo C 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

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

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, $n!$. U traženoj potenciji 2^m, eksponent m je

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

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 $\ell = 16$ bitova (tip `short int`) i $\ell = 32$ bita (tip `int`), 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.

U bazi 10, uz dogovor da vodeći dio bude jednoznamenkast ispred točke, tj. 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. tako da vrijedi

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

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

- prikaziv je samo neki raspon realnih brojeva,
- neki brojevi unutar prikazivog raspona nisu prikazivi, jer im je 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 znamo da 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 je malo složeniji.

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 e

- 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 normalizirana mantisa obično se zove signifikand.

Stvarni prikaz realnih brojeva — IEEE–754

Stvarni prikaz realnih brojeva ima tri dijela i svaki od njih ima svoju duljinu — broj bitova predviđenih za prikaz tog dijela.

- predznak s — uvijek zauzima jedan bit, i to najviši;
- karakteristika k — zauzima sljedećih w bitova (w = engl. "width", širina pomaknutog eksponenta);
- signifikand m — zauzima sljedećih t bitova (t = engl. "trailing", završni ili razlomljeni dio od m).

Po starom standardu — ako se pamti vodeći (cjelobrojni) bit mantise, on je prvi (vodeći) u m, a duljina je $t + 1$.

Još se koristi i standardna oznaka

- preciznost $p := t + 1$ — to je ukupni broj vodećih značajnih bitova cijele mantise.

Stvarni prikaz realnih brojeva — IEEE–754

Karakteristika k se interpretira kao cijeli broj bez predznaka, tako da je $k \in \{0, \ldots, 2^w - 1\}$. "Rubne" vrijednosti za k označavaju tzv. posebna stanja:

- $k = 0$ — nula i denormalizirani brojevi,
- $k = 2^w - 1$ — beskonačno i nije broj.

Sve ostale vrijednosti $k \in \{1, \ldots, 2^w - 2\}$ koriste se za prikaz normaliziranih brojeva različitih od nule.

Veza između karakteristike k i stvarnog eksponenta e je:

$$k = e + bias, \qquad bias = 2^{w-1} - 1.$$

Dakle, dozvoljeni eksponenti e moraju biti između

$$e_{\min} = -(2^{w-1} - 2) \quad \text{i} \quad e_{\max} = 2^{w-1} - 1.$$

Standardni tipovi realnih brojeva — IEEE–754

Novi standard IEEE–754 standard ima sljedeće tipove za prikaz realnih brojeva:

`ime tipa`	`binary32`	`binary64`	`binary128`
duljina u bitovima	32	64	128
$t =$	23	52	112
$w =$	8	11	15
$u = 2^{-p}$	2^{-24}	2^{-53}	2^{-113}
$u \approx$	$5.96 \cdot 10^{-8}$	$1.11 \cdot 10^{-16}$	$9.63 \cdot 10^{-35}$
raspon brojeva $\approx$	$10^{\pm 38}$	$10^{\pm 308}$	$10^{\pm 4932}$

Broj u je tzv. jedinična greška zaokruživanja (v. malo kasnije).

Najveći tip `binary128` još uvijek ne postoji u većini procesora.

Standardni tipovi realnih brojeva — extended

Većina PC procesora još uvijek ima posebni dio — tzv. FPU (engl. Floating–Point Unit). On stvarno koristi

- tip `extended` iz starog IEEE standarda, koji odgovara tipu `extended binary64` u novom IEEE–754 standardu.

Dio primjera koje ćete vidjeti napravljen je baš u tom tipu!

`ime tipa`	`extended`
duljina u bitovima	80
$t + 1 =$	$63 + 1$
$w =$	15
$u = 2^{-p}$	2^{-64}
$u \approx$	$5.42 \cdot 10^{-20}$
raspon brojeva $\approx$	$10^{\pm 4932}$

Oznake

Oznake:

- Crveno — duljina odgovarajućeg polja u bitovima, bitove brojimo od 0, zdesna nalijevo (kao i obično),
- s — predznak: 0 za pozitivan broj, 1 za negativan broj,
- k — karakteristika,
- m — mantisa (signifikand).

1	w	t
s	k	m

- Najznačajniji bit u odgovarajućem polju je najljeviji, a najmanje značajan bit je najdesniji.

Stvarni prikaz tipa single (binary32)

“Najkraći” realni tip je tzv. realni broj jednostruke točnosti. U C-u se taj tip zove `float`. Savjet: ne koristiti u praksi!

On ima sljedeća svojstva:

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

1	8	23
s	k	m

- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- stvarni eksponent e broja, $e \in \{-126, \ldots, 127\}$,
- 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:

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

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

Realni broj nula ima dva prikaza:

- mantisa i karakteristika imaju sve bitove jednake 0,

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 bit mantise koji nije nula, onda se kao eksponent uzima -126 (najmanji dozvoljeni). 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:

$$s = 0$$

$$k = 0000\ 0000$$

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

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

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

- $s = 0$ — prikaz $+\infty$, skraćena oznaka `+Inf`,
- $s = 1$ — prikaz $-\infty$, skraćena oznaka `-Inf`.

Rezultat `Inf` (odnosno, `-Inf`) dobivamo ako

- pokušamo spremiti preveliki broj (tzv. "overflow"), ili
- ako nešto različito od nule podijelimo s nulom.

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

$$s = 0 \qquad (s = 1)$$

$$k = 1111\ 1111$$

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

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

Ako je $k = 255$ i postoji bar jedan bit mantise različit od nule, onda je to oznaka za

- tzv. "Not a Number" ("nije broj") ili, skraćeno, `NaN`.

Rezultat `NaN` je uvijek signal da se radi o pogrešci. Na pr.,

- dijeljenje nule s nulom,
- vađenje drugog korijena iz negativnog broja i sl.

Primjer.

$$s = 0$$

$$k = 1111\ 1111$$

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

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)

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

ima 25 znamenki mantise i ne može se egzaktno spremiti u realni broj jednostruke točnosti, odnosno, tip `float` u C-u, koji ima $23 + 1$ znamenki za mantisu. Što se onda zbiva?

Tada se pronalaze dva najbliža prikaziva susjeda B_-, B_+, broju B, takva da vrijedi

$$B_- < B < B_+.$$

Greške zaokruživanja (nastavak)

U našem primjeru je:

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

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

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

Nakon toga, zaokružuje se rezultat. Zaokruživanje može biti:

- prema najbližem broju (standardno, engl. "default", za sve procesore) — ako su dva susjeda jednako udaljena od B, izabire "parni" od ta dva broja (zadnji bit je 0),
- prema dolje, tj. prema $-\infty$,
- prema gore, tj. prema $+\infty$,
- prema nuli, tj. odbacivanjem "viška" znamenki.

Greške zaokruživanja (nastavak)

Standardno zaokruživanje u našem primjeru:

$$B = 1.0001\ 0000\ 1000\ 0011\ 1001\ 0111\,1$$

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

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

Ovdje su B_- i B_+ jednako udaljeni od B, pa je zaokruženi B jednak B_+, jer B_+ 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 (tj. $\leq \frac{1}{2}\,2^{-t} = 2^{-p}$), 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
s	k	m

Vrijednost broja je

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

Raspon tipa float

Najveći prikazivi pozitivni broj je

$$\texttt{FLT\_MAX} = (1 - 2^{-24}) \cdot 2^{128} \approx 3.402823466 \cdot 10^{38},$$

s prikazom

$$\begin{aligned} s &= 0 \\ k &= 1111\ 1110 \\ m &= 111\ 1111\ 1111\ 1111\ 1111\ 1111 \end{aligned}$$

Najmanji prikazivi normalizirani pozitivni broj je

$$\texttt{FLT\_MIN} = 2^{-126} \approx 1.175494351 \cdot 10^{-38},$$

s prikazom

$$\begin{aligned} s &= 0 \\ k &= 0000\ 0001 \\ m &= 000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{aligned}$$

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

$$
\begin{aligned}
s &= 0\\
k &= 0000\ 0000\\
m &= 000\ 0000\ 0000\ 0000\ 0000\ 0001
\end{aligned}
$$

Stvarni prikaz tipa double (binary64)

"Srednji" realni tip je tzv. realni broj dvostruke točnosti.
U C-u se taj tip zove `double`. Savjet: njega treba koristiti!

On ima sljedeća svojstva:

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

1	11	52
s	k	m

- u mantisi se ne pamti vodeća jedinica ako je broj normaliziran,
- stvarni eksponent e broja, $e \in \{-1022, \ldots, 1023\}$,
- 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
s	k	m

Vrijednost broja je

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

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

$$\texttt{DBL\_EPSILON} = 2u \approx 2.2204460492503131 \cdot 10^{-16}.$$

Najveći prikazivi pozitivni broj je

$$\texttt{DBL\_MAX} = (1 - 2^{-53}) \cdot 2^{1024} \approx 1.7976931348623158 \cdot 10^{308}.$$

Najmanji prikazivi normalizirani pozitivni broj je

$$\texttt{DBL\_MIN} = 2^{-1022} \approx 2.2250738585072014 \cdot 10^{-308}.$$

Tip extended

Stvarno računanje (na IA–32) se obično radi u "proširenoj" točnosti. U C-u je taj tip možda dohvatljiv kao `long double`.

On ima sljedeća svojstva:

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

1	15	1	63
s	k	i	m

- u mantisi se pamti vodeći bit i mantise,
- stvarni eksponent e broja, $e \in \{-16382, \ldots, 16383\}$,
- 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
s	k	i	m

Vrijednost broja je

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

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)

Realna aritmetika računala — standard

Realna aritmetika računala nije egzaktna!

Razlog:

- Rezultat svake operacije mora biti prikaziv,
- pa dolazi do zaokruživanja.

Standard IEEE–754 za realnu artimetiku računala propisuje da za sve četiri osnovne aritmetičke operacije vrijedi

- ista ocjena greške zaokruživanja kao i za prikaz brojeva,
- tj. da izračunati rezultat ima malu relativnu grešku.

Isto vrijedi i za neke matematičke funkcije, poput $\sqrt{\ }$, ali ne vrijedi za sve funkcije (na pr. za cos i sin u okolini nule).

Realna aritmetika računala — zaokruživanje

Neka je $\circ$ bilo koja od aritmetičkih operacija $+$, $-$, $*$, $/$, i neka su x i y prikazivi operandi (drugih, ionako, nema u računalu).

- Ako su x i y u dozvoljenom, tj. normaliziranom rasponu,
- i ako se egzaktni rezultat $x \circ y$, također, nalazi u normaliziranom rasponu (ne mora biti prikaziv),

za računalom izračunati (prikazivi) rezultat $f\ell(x \circ y)$ onda vrijedi

$$f\ell(x \circ y) = (1 + \varepsilon)\,(x \circ y), \qquad |\varepsilon| \le u,$$

gdje je u jedinična greška zaokruživanja za dani tip brojeva. Ova ocjena odgovara zaokruživanju egzaktnog rezultata!

Prava relativna greška ε ovisi o x, y, operaciji $\circ$, i stvarnoj realizaciji aritmetike računala.

Posljedice zaokruživanja u realnoj aritmetici

Napomena. Bez pretpostavki o normaliziranom rasponu, prethodni rezultat ne vrijedi — greška može biti puno veća!

Zbog zaokruživanja, u realnoj aritmetici računala, nažalost,

- ne vrijede uobičajeni zakoni za aritmetičke operacije na skupu $\mathbb{R}$.

Na primjer, za aritmetičke operacije u računalu

- nema asocijativnosti zbrajanja i množenja,
- nema distributivnosti množenja prema zbrajanju.

Dakle, poredak izvršavanja operacija je bitan!

Zapravo, jedino standardno pravilo koje vrijedi je

- komutativnost za zbrajanje i za množenje.

Širenje grešaka zaokruživanja — ukratko

Vidimo da gotovo svaki izračunati rezultat ima neku grešku.

Kad imamo puno aritmetičkih operacija, dolazi do tzv.

- akumulacije ili širenja grešaka.

Očekujemo da greške rastu, ali koliko: "pomalo" ili "brzo"?

Zapamtite: Jedina "opasna" operacija u aritmetici računala je

- oduzimanje bliskih brojeva — tzv. "kraćenje", tj. kad
- iz velikih brojeva, aditivnim operacijama dobivamo male.

To može drastično povećati grešku (v. primjer malo kasnije).

Više o greškama zaokruživanja — u Numeričkoj matematici.

- Možete pogledati i dodatak ovom predavanju (na webu).

Primjer:
Neasocijativnost zbrajanja

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 (tzv. Catalanovi brojevi). Bitno nam je samo da svi ti rasporedi, matematički, 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:

$$S_1 := 1,$$

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

- 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	14.357357978820801	2.45740E−0003
single S_2	14.392651557922363	5.22243E−0006
double S_1	14.392726722864781	6.54899E−0014
double S_2	14.392726722865775	−2.14449E−0015
extended S_1	14.392726722865723	1.91639E−0017
extended S_2	14.392726722865724	−1.08475E−0018

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

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. Pred kraj, mali dodani član jedva utječe na rezultat (tj. dobar dio znamenki pribrojnika nema utjecaj na sumu).

Primjer:
“Katastrofalno” kraćenje

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 broja) imamo $p = 4$ dekadske znamenke, a za eksponent 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 $p = 4$ znamenke

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

Primjer katastrofalnog kraćenja (nastavak)

Ovim zaokruživanjima napravili smo malu relativnu grešku u x i y (ovdje je $u = \frac{1}{2}\, b^{-p} = 5 \times 10^{-5}$).

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 polazni brojevi točni. Dakle, ovo oduzimanje je egzaktno i u aritmetici računala.

Konačni izračunati 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 i u aritmetici našeg "računala", ali rezultat je, svejedno, pogrešan.

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

- Uzrok su polazne greške u operandima.

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

- i dalje, naravno, dolazi do kraćenja,
- ali je rezultat (uglavnom, a po IEEE standardu sigurno) egzaktan,

pa se ovo kraćenje onda zove benigno kraćenje.

Primjer:
Kvadratna jednadžba

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. Rješavamo kvadratnu jednadžbu $x^2 - 56x + 1 = 0$.

U dekadskoj aritmetici s $p = 5$ značajnih znamenki 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

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

Apsolutno manji od ova dva korijena — x_1, ima samo dvije točne znamenke (kraćenje). Drugi je "savršeno" točan.

Kvadratna jednadžba — popravak

Prvo izračunamo većeg po apsolutnoj vrijednosti, po formuli

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

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:

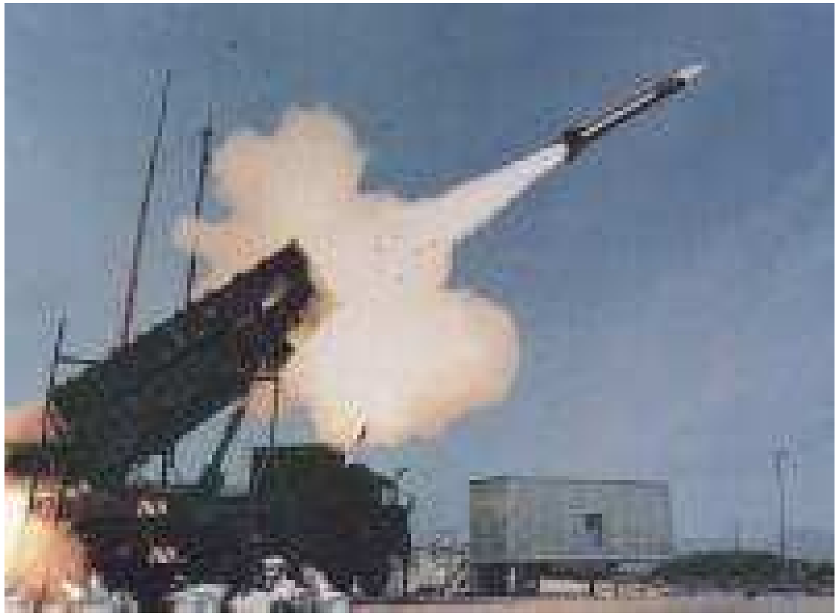
- "kvadriranje" pod korijenom — mogućnost za overflow. Rješenje — "skaliranjem".
- oduzimanje u diskriminanti (kraćenje) — nema jednostavnog rješenja. Naime, "krivac" nije aritmetika.
  - To je samo odraz tzv. nestabilnosti problema, jer tad imamo dva bliska korijena koji su vrlo osjetljivi na male perturbacije koeficijenata jednadžbe.
  - Na primjer, pomak c = pomak grafa "gore–dolje". Mali pomak rezultira velikom promjenom korijena!

Primjer “greške” iz prakse

Promašaj raketa Patriot

U prvom Zaljevskom ratu, 25. veljače 1991. godine, američke rakete Patriot nisu uspjele oboriti iračku Scud raketu iznad Dhahrana u Saudijskoj Arabiji.

- Scud raketa je pukim slučajem pala na američku vojnu bazu — usmrtivši 28 i ranivši stotinjak ljudi.



Promašaj raketa Patriot (nastavak)

Istraga otkriva sljedeće:

- Računalo koje je upravljalo Patriot raketama, vrijeme je brojilo u desetinkama sekunde proteklim od trenutka paljenja (uključivanja) sustava.
- Desetinka sekunde binarno

$$0.1_{10} = (0.0\dot{0}01\dot{1})_2.$$

- To računalo prikazivalo je realne brojeve korištenjem nenormalizirane mantise duljine 23 bita.
- Spremanjem broja 0.1 u registar takvog računala radi se (apsolutna) greška $\approx 9.5 \cdot 10^{-8}$ (sekundi).

Ne izgleda puno ... , a kamo li opasno.

Promašaj raketa Patriot (nastavak)

Detalji:

- Računalo je bilo u pogonu 100 sati, pa je ukupna greška zaokruživanja bila (stalno se zbraja, svakih 0.1 sekundi)

$$100 \cdot 60 \cdot 60 \cdot 10 \cdot 9.5 \cdot 10^{-8} = 0.34\,\mathrm{s}.$$

- Scud raketa putuje brzinom $\approx 1.6\,\mathrm{km/s}$, pa je "tražena" više od pola kilometra daleko od stvarnog položaja.
- Greška je uočena dva tjedna ranije, nakon 8 sati rada jednog drugog sustava. Modifikacija programa stigla je dan nakon nesreće.
- Posade sustava mogle su i dva tjedna ranije dobiti uputu "isključi/uključi računalo" svakih nekoliko sati — ali je nisu dobile.