PROGRAMOVÁNÍ PRO FYZIKY

LEKCE 1

Jan Schee

Ústav fyziky, Filozoficko-přírodovědecká fakulta,
Slezská univerzita v Opavě

- E-mail: jan.schee@fpf.slu.cz
- Kancelář: přízemí, číslo dveří 121
- Podmínky pro udělení zápočtu:
  - odevzdat vypracované domácí úkoly do konce zápočtového týdne

- GNU Compiler Collection
  - gcc – překladač jazyka **C**
  - g++ - překladač jazyka **C++**
  - gfortran - překladač jazyka fortran
  - … a další
- GNU make
  - nástroj pro automatické sestavování větších projektů
- BASH
  - sh-kompatibilní interpretr příkazů, spouští příkazy načtené ze standartního vstupu nebo ze souboru

- Cygwin
  - soubor nástrojů, které poskytují prostředí pro Windows s vlastnostmi Linuxu.
  - http://www.cygwin.com/
- MinGW a Msys
  - "Minimalist GNU for Windows", je minimalistické vývojové prostředí pro “Windows-ovské” aplikace
  - "Minimal SYStem", je interpreter příkazového řádku - Bourne Shell. Alternativa k “cmd.exe”
  - http://mingw.org/

- IDE
  - anjuta
    - je vývojové prostředí pro GNOME desktop
    - Zvýraznění syntaxe, správa projektů, application wizards, interaktivní debuger, ...
  - geany
    - malé a rychlé vývojové prostředí
    - nezávislé na konkrétním desktopu, závisí pouze na **gtk2** knihovnách
    - zvýraznění syntaxe, autocompletion, správa projektu,..
- Textové editory
  - Emacs - podpora maker, vysoce konfigurovatelný
  - Vim – podpora maker, vysoce konfigurovatelný

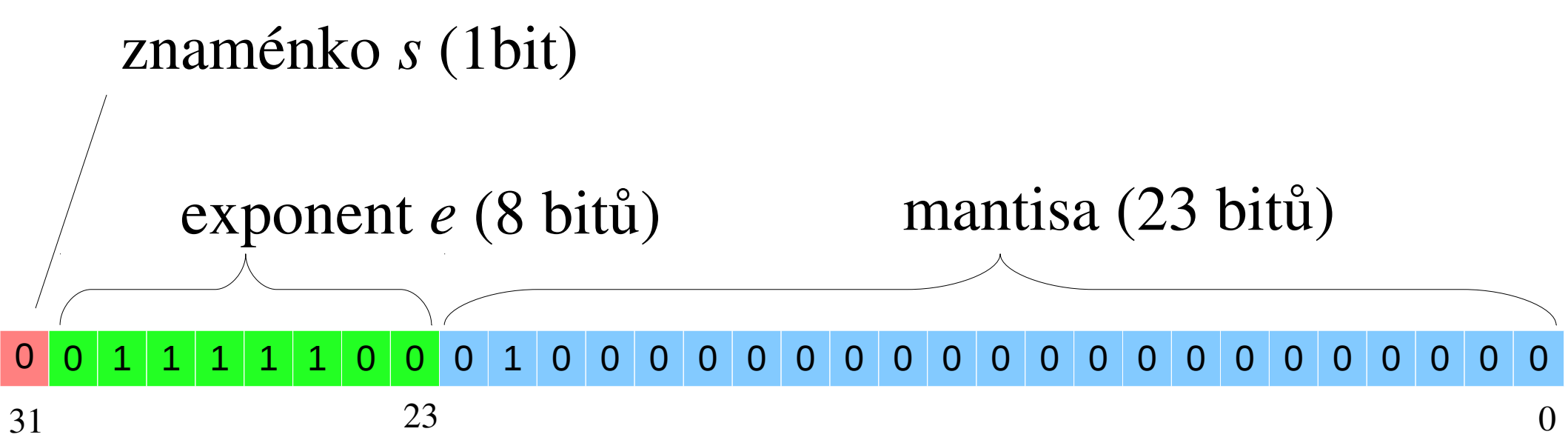
- **Řešení soustavy lineárních algebraických rovnic**
  - Gaussova-Jordanova eliminace
  - SU dekompozice
- **Interpolace a extrapolace funkcí**
  - Polynomiální interpolace a extrapolace
  - Kubická splinová interpolace
- **Integrace funkcí**
  - Rombergova integrace
  - Integrace pomocí Gaussových kvadratur
- **Hledání kořenů**
  - Polynomické rovnice
  - Transcendentní rovnice
- **Integrace obyčejných diferenciálních rovnic**
  - Eulerova metoda
  - Metoda Runge-Kutha s fixním krokem
  - Metoda Runge-Kutha s adaptivním krokem

Čísla v počítači

- Číslo v **celočíselné reprezentaci** je exaktní, v paměti je uloženo do pole bitů dané velikosti (1byte, 2 byte, 4 byte,...) a vzájemné operace opět vedou k exaktním číslům.
- V **reprezentaci plovoucí desetiné** čárky je číslo definováno následovně

$$s \times M \times B^{e-E}$$

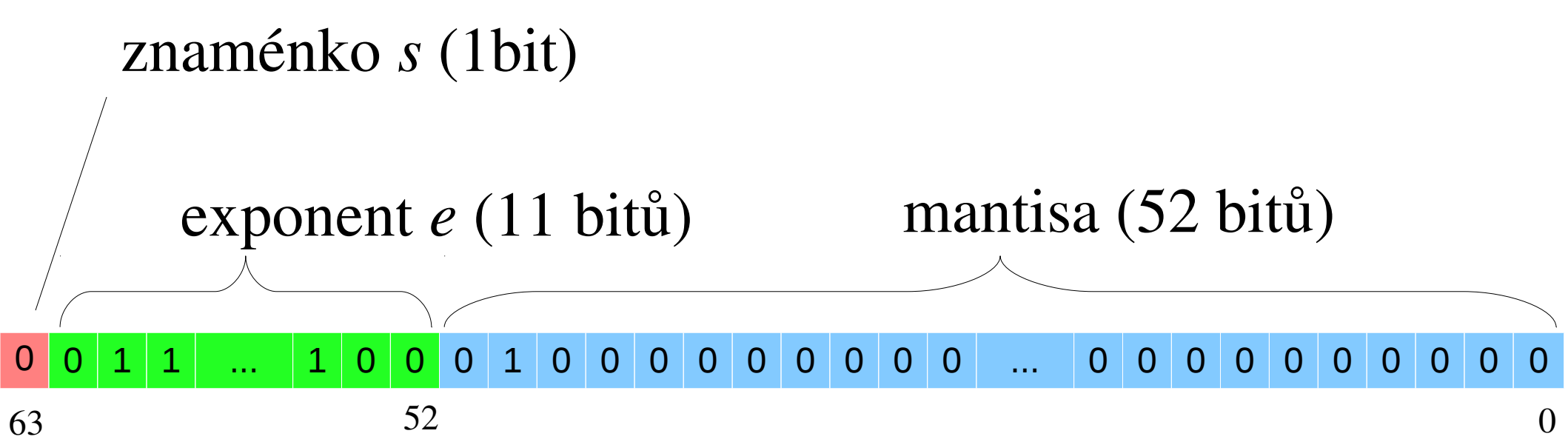
kde je *s* znaménko, *M* mantisa (exaktní, kladné, celé číslo) , *B* báze reprezentace (v počítači je *B*=2) a *E* bias exponentu *e*.



$$0.15625$$

$$0.15625 = (-1)^s \left(1 + \sum_{i=1}^{23} b_{-i} 2^{-i}\right) \times 2^{e-127}$$

$$= (-1)^0 \left(1 + 2^{-2}\right) \times 2^{124-127} = 1.25 \times 2^{-3}$$



$$0.15625$$

$$0.15625 = (-1)^s \left(1 + \sum_{i=1}^{52} b_{-i} 2^{-i}\right) \times 2^{e-1023}$$

$$= (-1)^0 \left(1 + 2^{-2}\right) \times 2^{1020-1023} = 1.25 \times 2^{-3}$$

$$a = (12.375)_{10} = (12)_{10} + (0.375)_{10}$$

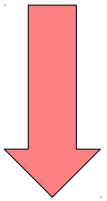
$$\begin{aligned}(12)_{10} &= b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 \\ &= 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 \\ &= (1100)_2\end{aligned}$$

$$\begin{aligned}(0.375)_{10} &= b_{-1} 2^{-1} + b_{-2} 2^{-2} + b_{-3} 2^{-3} + b_{-4} 2^{-4} \\ &= 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} + 1 \cdot \frac{1}{8} + 0 \cdot \frac{1}{16} \\ &= (0.011)_2\end{aligned}$$

$$a = (12.375)_{10} = (1100.011)_2$$

Normalizace

$$a = (1.100011 \times 2^3)_2$$



$$a = (1.100011 \times 2^{(e-127)})_2$$

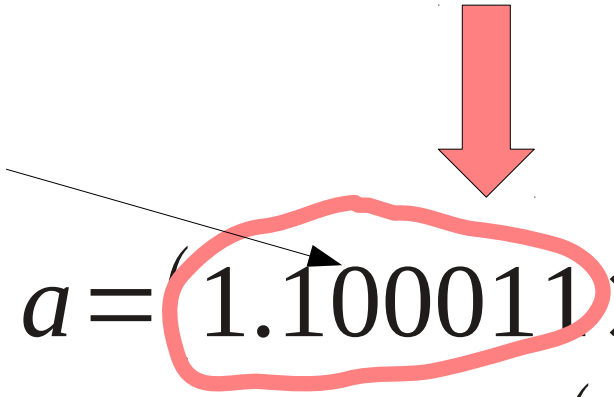
$$\Rightarrow e = 130_{10} = (1000\,0011)_2$$

$$a = (12.375)_{10} = (1100.011)_2$$

Normalizace

$$a = (1.100011 \times 2^3)_2$$

Mantisa



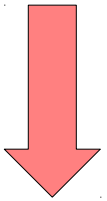
$$a = (1.100011 \times 2^{(e-127)})_2$$

$$\Rightarrow e = 130_{10} = (1000\,0011)_2$$

$$a = (12.375)_{10} = (1100.011)_2$$

Normalizace

$$a = (1.100011 \times 2^3)_2$$



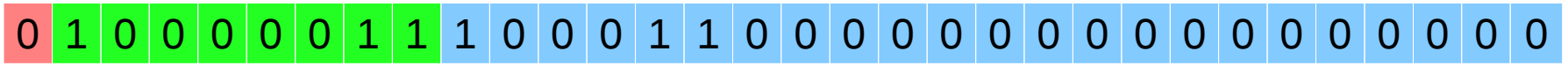
$$a = (1.100011 \times 2^{(e-127)})_2$$

$$\Rightarrow e = 130_{10} = (1000\,0011)_2$$

Biasovaný exponent

FLOAT

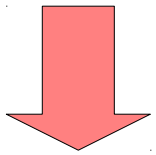
$$a = (-1)^0 \left(1.100011 \times 2^{(10000011 - 01111111)}\right)_2$$



$$(0.1)_{10} = b_{-1}2^{-1} + b_{-2}2^{-2} + b_{-3}2^{-3} + b_{-4}2^{-4} + \cdots$$

$$= 0\cdot\frac{1}{2} + 0\cdot\frac{1}{4} + 0\cdot\frac{1}{8} + 1\cdot\frac{1}{16} + 1\cdot\frac{1}{32} +$$

$$0\cdot\frac{1}{64} + 0\cdot\frac{1}{128} + 0\cdot\frac{1}{256} + 1\cdot\frac{1}{512} + \cdots$$



$$(0.1)_{10} = (0.0001100110011001100\ldots)_2 = (0.11001100\ldots\times 2^{-3})_2$$

Konečný binární rozvoj mají pouze racionální čísla se jmenovatelem který je mocninou čísla 2 (např. ½, 3/16, …)!

Ostatní radionální čísla mají binární rozvoj nekonečný !

- Přesnost dané reprezentace určuje výraz

$$p = m \times \mathrm{Log}_{10}(2)$$

kde je m počet bitů v mantise.

- Přesnost vybraných číselných typů:
  - p(**float**) ≐ 6.9
  - p(**double**) ≐ 15.6
- **Desetinná čísla nejsou přesně reprezentována!**

- **Strojová přesnost**, ε_m, je číslo s plovoucí desetinou čárkou, které **po přičtení** k číslu 1.0 dá číslo s plovoucí desetinou čárkou **různé** od 1.0
- Pro typický počítač s B=2 a číselný datový typ float (32 bitů) je

$$\epsilon_m = 3 \times 10^{-8}$$

- POZOR: ε_m **není** nejmenší reprezentovatelné číslo v dané reprezentaci. Např. Pro datový typ float je interval reprezentovatelných čísel $[\pm 1.18 \times 10^{-38} - \pm 3.4 \times 10^{38}]$

- Přetečení (Overflow)
  - Nastává v okamžiku, kdy exponent přesáhne maximální hodnotu (308 v případě typu double)
- Podtečení (Underflow)
  - Nastává v okamžiku, kdy začne být exponent menší než minimální hodnota (-308 pro typ double)

- Chyba zaokrouhlení (Roundoff error)
  - jak jsme ukázali ne všechna čísla lze přesně reprezentovat binárním rozvojem
  - Konečný binární rozvoj některých čísel může být delší než je počet bitů v mantise

  Taková čísla jsou pak **zaokrouhlena** a výpočty jsou zatíženy **chybou zaokrouhlování**

- Tato chyba roste s rostoucím počtem výpočtů. Provádíme-li N výpočtů pak celková chyba zaokrouhlouvání může být řádově $\varepsilon_m \sqrt{N}$.

- Ztráta přesnosti (Loss of precision)

  Máme spočítat součet $1+10^{-8}$ a máme k distpozici datový typ **float**.

  Při sčítání počítač nejprve srovná exponenty sčítanců (tak aby se sobě rovnaly).

  Pro 10^{-8} to znamená dostat exponent 10^{0} a tedy bity v mantise se musí posunout doprava. Nakonec jsou všechny bity v mantise rovny **0** !

  Takže výsledek bude $\mathbf{1+10^{-8} = 1}$ (a to je zlé)

- Ilustrace nevhodné volby algoritmu. Zlatý řez Z je dán výrazem

$$Z \equiv \frac{\sqrt{5}-1}{2} \approx 0.61803398$$

- Mocniny Z^n splňují jednoduchou rekurzní relaci

$$Z^{n+1} = Z^n - Z^{n-1}$$

Tento algoritmus je nestabilní. Pro datový typ float dává chybný výsledek už pro n=16.

- Dále ilustrujme ztrátu přesnosti na příkladu výpočtu funkce exp(-x) řadou:

$$\exp(-x) = \sum_{\infty}^{n=0} (-1)^n \frac{x^n}{n!}$$

- Výpočet faktoriálu

```
double fact(int n)
{
    Int i;
    double f;

    f=1.0
    for(i=1;i<=n;i++) f=f*(double)i;

    return f;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define TRUNC 1.0e-14

double fact(int n);

int main()
{
    int i;
    double x,term,s,e;

    printf("     x            exp(-x)      series     num\n"
           "=========================================\n");
    for(x=0.0;x<=100.0;x=x+10.0)
    {
        term=1.0;
        i=0;
        e=0.0;
        while(fabs(term)>TRUNC)
        {
            s=(double)(1-2*(abs(i) % 2));
            term=s*pow(x,(double)i)/fact(i);
            e=e+term;
            i++;
        }

        printf("%.6le %.6le %.6le %d\n",x,exp(-x),e,i-1);
    }
    return 0;

}
```

Výsledek výpočtu exp(-*x*) řadou.

```
     x              exp(-x)       series      num
=========================================================
0.000000e+00 1.000000e+00 1.000000e+00 1
1.000000e+01 4.539993e-05 4.539993e-05 50
2.000000e+01 2.061154e-09 4.992638e-09 79
3.000000e+01 9.357623e-14 -4.820864e-06 107
4.000000e+01 4.248354e-18 4.881300e+00 135
5.000000e+01 1.928750e-22 4.864373e+04 163
6.000000e+01 8.756511e-27 1.120641e+08 171
7.000000e+01 3.975450e-31 -nan 171
8.000000e+01 1.804851e-35 -nan 171
9.000000e+01 8.194013e-40 -nan 171
1.000000e+02 3.720076e-44 -nan 171
$
```

- Problému výpočtu faktoriálu se vyhneme využitím rekurzivní formule

$$\exp(-x) = \sum_{\infty}^{n=0} s_n$$

kde je

$$s_n = -s_{n-1}\frac{x}{n}$$

```c
/*
    curing factorial problem of series.c code
    using recursive formula to avoid factorial calculation
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define TRUNC 1.0e-10


int main()
{
    int i;
    double x,term,s,e;

    printf("     x           exp(-x)     series    num\n"
           "=========================================\n");
    for(x=0.0;x<=100.0;x=x+10.0)
    {
        term=1.0;
        i=1;
        e=1.0;
        while(fabs(term)>TRUNC)
        {
            term *=-x/(double)i;
            e += term;
            i++;
            //fprintf(stderr,"%d %lf\n",i,term);
        }

        printf("%.6le %.6le %.6le %d\n",x,exp(-x),e,i-1);
    }
    return 0;

}
```

Výsledek výpočtu exp(-*x*) řadou a rekurzivní formulí.

```
      x              exp(-x)         series       num
=============================================================
0.000000e+00 1.000000e+00 1.000000e+00 1
1.000000e+01 4.539993e-05 4.539994e-05 44
2.000000e+01 2.061154e-09 6.164185e-09 72
3.000000e+01 9.357623e-14 6.103055e-06 100
4.000000e+01 4.248354e-18 3.116952e-01 127
5.000000e+01 1.928750e-22 2.041833e+03 155
6.000000e+01 8.756511e-27 7.227457e+08 182
7.000000e+01 3.975450e-31 4.594081e+12 209
8.000000e+01 1.804851e-35 2.450820e+17 237
9.000000e+01 8.194013e-40 -5.865799e+21 264
1.000000e+02 3.720076e-44 8.144653e+25 291
$
```

Výsledek výpočtu exp(-*x*) řadou a rekurzivní formulí.

```
      x              exp(-x)         series       num
=============================================================
0.000000e+00 1.000000e+00 1.000000e+00 1
1.000000e+01 4.539993e-05 4.539994e-05 44
2.000000e+01 2.061154e-09 6.164185e-09 72
3.000000e+01 9.357623e-14 6.103055e-06 100
4.000000e+01 4.248354e-18 3.116952e-01 127
5.000000e+01 1.928750e-22 2.041833e+03 155
6.000000e+01 8.756511e-27 7.227457e+08 182
7.000000e+01 3.975450e-31 4.594081e+12 209
8.000000e+01 1.804851e-35 2.450820e+17 237
9.000000e+01 8.194013e-40 -5.865799e+21 264
1.000000e+02 3.720076e-44 8.144653e+25 291
$
```

- Určíme nejprve, předchozí metodou $\exp(x)$

  a následně už snadno určíme

$$\exp(-x) = \frac{1}{\exp(x)}$$

```c
/*
    curing factorial problem of series.c code
    using recursive formula to avoid factorial calculation
    and calculating exp(x) first and then take exp(-x)=1/exp(x)
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define TRUNC 1.0e-10


int main()
{
    int i;
    double x,term,s,e;

    printf("     x           exp(-x)     series    num\n"
           "=========================================\n");
    for(x=0.0;x<=100.0;x=x+10.0)
    {
        term=1.0;
        i=1;
        e=1.0;
        while(fabs(term)>TRUNC)
        {
            term *=x/(double)i;
            e += term;
            i++;
            //fprintf(stderr,"%d %lf\n",i,term);
        }

        printf("%.6le %.6le %.6le %d\n",x,exp(-x),1.0/e,i-1);
    }
    return 0;

}
```

Výsledek výpočtu exp(-*x*) řadou a rekurzivní formulí a 1/exp(x) .

```
      x                exp(-x)         series       num
=============================================================
0.000000e+00 1.000000e+00 1.000000e+00 1
1.000000e+01 4.539993e-05 4.539993e-05 44
2.000000e+01 2.061154e-09 2.061154e-09 72
3.000000e+01 9.357623e-14 9.357623e-14 100
4.000000e+01 4.248354e-18 4.248354e-18 127
5.000000e+01 1.928750e-22 1.928750e-22 155
6.000000e+01 8.756511e-27 8.756511e-27 182
7.000000e+01 3.975450e-31 3.975450e-31 209
8.000000e+01 1.804851e-35 1.804851e-35 237
9.000000e+01 8.194013e-40 8.194013e-40 264
1.000000e+02 3.720076e-44 3.720076e-44 291
$
```

- Další problém, který souvisí s kumulací roundoff chyby ilustruje příklad výpočtu součtu řad

$$s_1 = \sum_{n=1}^{N} \frac{1}{n} \qquad s_2 = \sum_{n=N}^{1} \frac{1}{n}$$

- V případě reálné analýzy, samozřejmě, platí $s_1 = s_2$
- Numerický výpočet pro N=10 a N=100 a datový typ **float** dá výsledek

	N=10	N=**100**
s_1	2.92896843	5.18737793
s_2	2.92896843	5.18737698

- Další problém, který souvisí s kumulací roundoff chyby ilustruje příklad výpočtu součtu řad

$$s_1 = \sum_{n=1}^{N} \frac{1}{n} \qquad s_2 = \sum_{n=N}^{1} \frac{1}{n}$$

- V případě reálné analýzy, samozřejmě, platí $s_1 = s_2$
- Numerický výpočet pro N=100 a N=10^8 a datový typ **double** dá výsledek

	N=100	N=$\mathbf{10^8}$
s_1	5.1873775176396206	18.997896413852554 8
s_2	5.1873775176396215	18.997896413853446 5

- Numerické výpočty jsou zatíženy chybou z důvodu konečné přesnosti kterou nám poskytuje zvolená reprezentace reálného čísla v počítači.
- Tato chyba má tendenci se kumulovat.
- Její kumulaci lze uvést do rozumných mezí vhodnou volbou algoritmu výpočtu a zvolením reprezentace s vyšší přesností.

- Napište program, který řeší následující kvadratickou rovnici

$$0.00000001\,x^2+2\,x-5=0$$

- Standartně

$$a\,x^2+b\,x+c=0\Rightarrow x_{\pm}=\frac{-b\pm\sqrt{D}}{2\mathrm{a}},\ D=b^2-4\,a\,c$$

- Numericky korektně

$$a\,x^2+b\,x+c=0\Rightarrow x_1=\frac{q}{a},\ x_2=\frac{c}{q},\ q=\frac{1}{2}\left(b+sgn\left(b\right)\sqrt{D}\right)$$

A to je konec …