1. Opakování teorie

1.1. Reprezentace čísel v počítači

Celá čísla (přesné výpočty, velmi omezený rozsah):

- INTEGER => 2 byty = 16 bitů => 2^{16} čísel <-32768, 32767>
- LONGINT => 4 byty = 32 bitů => 2^{32} čísel <-2^{31}, $-2^{31}-1$>

Reálná čísla - čísla v pohyblivé desetinné tečce :

- $\pm$ 5.423687 . $10^{\pm 23}$

Znaménko

Exponent

Mantisa

Reálná čísla jsou v počítači uložena jako (dvojková soustava): $s \times m \times 2^e$.

Délka MANTISY - tj. počet bitů na mantisu určuje přesnost čísla (tj. počet čísel mezi 1 a 2).

Interval mezi čísly mezi 1 a 2 je rovnoměrný - do paměti se mohou ukládat jenom čísla 1, $1+\varepsilon$, $1+2\varepsilon$, ... , $2-\varepsilon$ (viz. kap. 1.2).

Čím více bitů na mantisu, tím menší $\varepsilon \rightarrow$ menší chyby při zaokrouhlování (u mezivýsledků je v registrech procesoru přesnost vyšší).

Při změnách exponentů se krok mezi čísly zvýší úměrně $2^{exponent}$ (relativní chyba čísla se ale nemění).

Různé typy reálných čísel se liší počtem bitů použitých na uložení exponentu, velikostí konstanty a počtem bitů použitých na uložení mantisy. Zatímco exponent je zodpovědný za to, jak veliké nebo malé číslo můžeme uložit, mantisa je zodpovědná za to, jak přesné číslo máme.

Délka EXPONENTU - tj. počet bitů na exponent - určuje rozsah

Datový typ	**Velikost v paměti**	**Rozsah**
	Celočíselné typy	
Boolean	1 bit (ačkoliv obvykle uložen jako 1 bajt)	0 až 1
Byte	8 bitů (= 1 bajt)	0 až 255
Word	2 bajty	0 až 65 535
Double Word	4 bajty	0 až 4 294 967 295
Integer	4 bajty	–2 147 483 648 až 2 147 483 647
Double Integer	8 bajtů	–9 223 372 036 854 775 808 až 9 223 372 036 854 775 807
	Typy s plovoucí čárkou	
Real	4 bajty	1E-37 až 1E+37 (6 desetinných míst)
Double Float	8 bajtù	1E-307 až 1E+308 (15 desetinných míst)

1.2. Strojové epsilon

Každé reálné číslo je v počítači uloženo s omezeným počtem platných cifer, tedy s omezenou přesností. Ve skutečnosti se jedná o omezený počet cifer ve dvojkové soustavě, což je pro běžného uživatele špatně představitelné. Proto se často jako míra této přesnosti užívá tzv. *strojové epsilon*. Jedná se o nejmenší číslo, které když přičteme k jedničce (stejného datového typu) získáme číslo odlišné od jedničky. Každé menší číslo je po přičtení a zkrácení na patřičný počet platných cifer je totožné s jedničkou.

Čím více platných cifer budeme mít k dispozici, tím menší bude strojové epsilon. Můžeme tedy očekávat, že reálné typy zabírající v paměti více místa budou mít menší strojové epsilon.

Odhad strojového epsilon můžeme získat tak, že zvolíme počáteční odhad (libovolně vysoký) a postupně jej dělíme například dvěma tak dlouho, dokud po přičtení k jedničce dostaneme číslo větší než jedna.

```
{Program pro odhad strojoveho epsilon. Tedy nejmensiho cisla, ktere po
pricteni k 1 jeste neda opet 1. Pocitat budeme pro typy real, single a
double}
{$N-} {prepinac zakazující pouzivani koprocesoru a vylepsenych schopnosti}
program StrojoveEpsilon;
var
        jednickaReal,epsilonReal:real;
begin
        jednickaReal := 1.0; {pouzivame proto, aby jednicka i epsilon melo
stejny typ}
        epsilonReal := 0.1; {uvodni odhad epsilon}
        while jednickaReal + epsilonReal > jednickaReal do
        begin
                epsilonReal := epsilonReal / 2.0;
        end;

        write('Typ REAL zabira v pameti bytu: ');
        writeln(sizeof(jednickaReal));
        write('Odhad strojoveho epsilon typu real je: ');
        writeln(epsilonReal);
end.
```

1.3. Minimální kladné číslo

V každém reálném datovém typu existuje minimální kladné číslo. Každé menší už by bylo uloženo jako nula. Takové číslo by mělo maximální možný záporný exponent a v mantise samé nuly (neboť první jednička je explicitně v mantise vždy).

Odhad minimálního kladného čísla můžeme získat tak, že zvolíme počáteční odhad (libovolně vysoký) a postupně jej dělíme například dvěma tak dlouho, dokud nezískáme nulu. Číslo předcházející tomuto poslednímu je náš odhad.

```
{Program pro odhad nejmensioho nenuloveho cisla.}
{$N-} {prepinac zakazující pouzivani koprocesoru a vylepsenych schopnosti}
program MinimalniCislo;
var
        minCislo,predchoziMinCislo:real;
begin
        minCislo := 0.1;
        while minCislo > 0.0 do
        begin
```

```
                predchoziMinCislo := minCislo;
                minCislo := minCislo / 2.0;
        end;

        write('Typ REAL zabira v pameti bytu: ');
        writeln(sizeof(minCislol));
        write('Odhad nejmensiho nenuloveho cisla: ');
        writeln(predchoziMinCislo);
end.
```

1.4. Počítačová aritmetika

Z principu uložení reálných čísel v počítači lze dospět k několika pravidlům, které je třeba mít stále na mysli. Některé zřejmé rovnosti či nerovnosti z matematiky platí i v počítači, některé však nikoli.

Platí:

- $1 \times x = x$
- $x \times y = y \times x$
- $x + x = 2 \times x$

Nemusí platit:

- $x \times x^{-1} = 1$
- $(1 + x) – 1 = x$
- $(x + y) + z = x + (y + z)$

Že neplatí asociativnost sčítání se můžeme snadno přesvědčit na jednoduchém příkladu. Mějme řadu definovanou jako $x_n = 1 / n^{-1.1}$. Při sečtení prvních 20 členů v jednom směru a v druhém směru dostaneme poněkud odlišný výsledek, ačkoli jsme sčítali naprosto stejná čísla!

```
{Program pro testovani rozdilu poradi scitani}
{$N+$E+}
program TestRazeniScitani;
var
   n:integer;

(*takto nadefinujeme radu, v podstate se jedna o i^{-1.1} *)
function rada(i:integer):real;
begin
     rada := exp(-ln(1.1)*i);
end;

(* tato funkce nam secte cleny vzestupne *)
function soucetPrvnichNVzestupne(n:integer):real;
var
   i:integer;
   mezisoucet:real;
begin
     mezisoucet := 0;
     for i := 0 to n do begin
         mezisoucet := mezisoucet + rada(i);
     end;
     soucetPrvnichNVzestupne := mezisoucet;
end;
```

```
(* a tato sestupne *)
function soucetPrvnichNSestupne(n:integer):real;
var
   i:integer;
   mezisoucet:real;
begin
     mezisoucet := 0;
     for i := n downto 0 do begin
         mezisoucet := mezisoucet + rada(i);
     end;
     soucetPrvnichNSestupne := mezisoucet;
end;

begin
     write('Zadejte pocet scitancu n: ');
     readln(n);

     write('Soucet vzestupne: ');
     writeln(soucetPrvnichNVzestupne(n));

     write('Soucet sestupne:  ');
     writeln(soucetPrvnichNSestupne(n));

     readln;
end.
```

1.5. Zaokrouhlovací chyba

Zdroje chyb:

- Chyby vstupních dat (např. chyby měření, chyby modelu reality)
- Chyby metody (Truncation errors) - v důsledku převedení matematické úlohy na numerickou
- Zaokrouhlovací chyby (Roundoff errors) - v důsledku zaokrouhlování při výpočtech s čísly o konečné délce

1.6. Zaokrouhlovací chyba při aritmetických operacích

Absolutní chyba:

$$A(x) = |x_1 - x_2| \leq a(x)$$

kde:

- x_1 přesná hodnota
- x_2 přibližná hodnota
- $a(x)$ odhad absolutní chyby

Relativní chyba:

$$R(x) = \frac{A(x)}{|x|} \leq r(x)$$

kde:

- r(x) odhad relativní chyby

1.7. Úprava výrazů

Při výpočtech derivace, integrálu apod. nahrazujeme nekonečně krátký krok *dx* konečným krokem *h*.

Pozor: Tento typ chyby nijak nesouvisí se zaokrouhlováním.

1.8. Korektnost a podmíněnost úlohy

Korektnost úlohy - definice:

Nechť úlohou je najít řešení $\vec{y} \in N$ (N je množina možných řešení) pro zadaný vektor $\vec{x} \in M$ (M je množina vstupních dat). Pak úloha je korektní právě tehdy, jsou-li splněny následující dvě podmínky :

1. Existuje právě jedno řešení $\vec{y}$ pro $\forall \vec{x} \in M$.

2. Řešení spojitě závisí na vstupních datech, tj. jestliže pro $\forall n$ z množiny přirozených čísel je $\vec{y}_n$ řešení pro vstupní data $\vec{x}_n$, a jestliže $\vec{y}$ je řešení pro vstupní data $\vec{x}$, nechť dále ρ je norma v množině vstupních dat a σ je norma v množině možných řešení, pak platí:

$$x_n \xrightarrow{\rho} x \Rightarrow y_n \xrightarrow{\sigma} y$$

V praxi se řeší i nekorektní úlohy, ale 1. krok řešení spočívá v nalezení vhodného způsobu, jak převést úlohu na úlohu korektní (např. podmínkou na výsledek; interpretací vstupních dat; vhodnou volbou normy v prostoru řešení apod.)

Podmíněnost úlohy - definice:

Podmíněnost úlohy C_p je daná poměrem relativní změny výsledku ku relativní změně vstupních dat, tj.:

$$C_p = \frac{\frac{\|\delta y\|}{\|y\|}}{\frac{\|\delta x\|}{\|x\|}} \approx \frac{r(y)}{r(x)}$$

Pokud $C_p \sim 1$, říkáme, že úloha je dobře podmíněná, pokud $C_p > 100$, úloha je špatně podmíněná.
Pokud je přesnost použitého typu čísel ε $(r(x) = \varepsilon)$, pak úloha s $C_p > \varepsilon^{-1}$ není v rámci dané přesnosti řešitelná.

Často se pro špatně podmíněné úlohy používají speciální metody, které omezují růst zaokrouhlovacích chyb.

Příklad:

Soustava lineárních rovnic s maticí blízkou k singulární (špatně podmíněná matice). Nechť je dána úloha:

$$x + \alpha y = 1$$

$$\alpha x + y = 0$$

Nechť vstupem je hodnota α a výstupem hodnota x. Pak

$$x = \frac{1}{1-\alpha^2} \qquad \text{a} \qquad C_p = \frac{\dfrac{\|\delta x\|}{\|x\|}}{\dfrac{\|\delta\alpha\|}{\|\alpha\|}} \approx \left|\frac{\alpha}{x}\frac{dx}{d\alpha}\right| = \left|\frac{\alpha}{\dfrac{1}{1-\alpha^2}}\frac{2\alpha}{\left(1-\alpha^2\right)^2}\right| = \frac{2\alpha^2}{\left|1-\alpha^2\right|}$$

Při $\alpha^2 \to 1$ je úloha špatně podmíněná.