Tomáš Holan, dlouha.txt, Verse: 19. února 2006.

Dlouhá čísla

Už známe datové typy pro representaci celých čísel i typy pro representaci desetinných čísel. Co ale dělat, když nám žádný z dostupných datových typů nevyhovuje? Možná potřebujeme přesně počítat celá čísla o velikosti přesahující rozsah typů **integer** i **longint**, jindy zase můžeme potřebovat desetinná čísla s větším počtem platných číslic, než nabízejí **real**, **double** i **extended**.

V takovém případě nezbývá, než si vytvořit vlastní representaci čísel a naprogramovat vlastní provádění aritmetických operací na našem novém typu.

Při volbě způsobu represenatce bychom si měli položit a zodpovědět následující otázky:

- budeme reprezentovat **pouze nezáporná nebo i záporná** čísla?
- pokud záporná, budeme je ukládat **se znaménkem nebo v doplňkovém kódu**?
- budeme reprezentovat čísla **celá nebo i desetinná**?
- pokud desetinná, **s pevnou nebo pohyblivou řádovou čárkou**?
- **na kolik míst** budeme číslo ukládat?
- použijeme pro ukládání **pole nebo spojový seznam**?
- **jakou část čísla bude obsahovat jeden prvek** naší datové reprezentace?

Pokud máme řešit konkrétní úlohu, potřebujeme ještě rozhodnout, jaké operace budeme na našem datovém typu potřebovat — a ty potom naprogramovat. Ukažme si vše na konkrétním příkladu.

1 Příklad

Zadání Napište program, který spočte a vytiskne hodnotu Eulerovy konstanty e s přesností na 1000 desetinných míst. Pro výpočet použijte vzorce

$$e = \sum_{i=0}^{\infty} 1/i!$$

Začněme reprezentací dat a kladením a zodpovídáním otázek. Přijdeme na to, že nám stačí čísla **nezáporná**, **desetinná**, **s pevnou řádovou čárkou**.

U otázky počtu míst se zarazíme. Je zřejmé, že chceme-li znát výsledek s přesností na 1000 desetinných míst, nebude stačit provádět výpočty také s touto přesností, protože kdybychom každý prvek sčítané řady ořízli za 1000-tým desetinným místem, součet všech takto oříznutých hodnot by mohl způsobit, že hodnota na několika posledních desetinných místech vyjde jiná, než by měla vyjít. Budeme tedy muset počítat o něco přesněji. O kolik, to se pokusíme odhadnout (v praxi to ovšem pokaždé odhadnout nedokážeme a tedy se budeme muset

spokojit s tím, že zkusíme provést výpočet pro různý počet míst navíc a uvidíme, kam až sahají změny plynoucí ze změny počtu míst).

V tomto případě se ale o odhad můžeme pokusit.

Kdybychom věděli, kolik členů řady budeme sčítat, mohli bychom počítat s nejhorším případem, že totiž uříznutá část obsahuje samé devítky a vyšlo by nám, že pro jeden člen součtu bychom potřebovali jedno místo navíc, pro deset členů dvě místa, pro sto tři místa... neboli že počet míst, o která musíme zvýšit přesnost odpovídá logaritmu počtu prvků sčítané řady, zvětšenému o jedničku.

Hrubý odhad počtu prvků můžeme potom provést takto:

- stačí sčítat jen ty prvky, které v naší reprezentaci nebudou rovny nule
- i-tý prvek řady bude i-krát menší než jeho předchůdce
- pro i alespoň 10 to znamená, že další prvek bude alespoň desetkrát menší
- pro i alespoň 100 to znamená, že další prvek bude alespoň stokrát menší
- prvky pro i=10..99 tedy posunou nenulové hodnoty ve sčítanci alespoň o 90*1 číslic doprava (vydělí nejméně 10^{90})
- prvky pro i=100..999 tedy posunou nenulové hodnoty ve sčítanci alespoň o 900*2 číslic doprava (vydělí nejméně 10^{1800})
- 1000-tý prvek tedy nebude větší než $1/10^{1890}$
- 1000-tý prvek tedy bude jistě na prvních 1890 desetinných místech obsahovat nuly
- 1000-tý prvek nás tedy z hlediska součtu řady určitě nebude zajímat
- určitě tedy nebudeme sčítat více než 1000 prvků

Proto nám stačí, když pro výpočet budeme počítat a přesností na 1004 desetinných míst.

Čísla budeme representovat jako čísla s pevnou řádovou čárkou (všechna budou mít 1004 desetinných míst), před desetinnou čárkou nám stačí jediné místo (podvádíme, protože víme, že celý součet nekonečné řady nepřekročí číslo e, jehož hodnotu $2,7\ldots$ známe z matematiky.

Ještě se rozhodneme, že číslo budeme ukládat v poli a že každý prvek bude obsahovat pouze jedinou desítkovou číslici. Z hlediska paměti i (strojového) času to není příliš výhodné, jeden prvek pole zabírá nejméně jeden byte a do jednoho byte bychom mohli umístit hodnoty 0..99, případně do prvků typu **word** umístit hodnoty 0..9999 nebo do typu **longint** hodnoty 0..999999999 (ještě o jednu desítkovou číslici více než dvojnásobek). Protože nám ale teď záleží více na přehlednosti a pochopitelnosti programu, budeme v prvcích ukládat jednotlivé číslice.

Deklarace typu tedy bude vypadat takto:

```
const
   MAX = 1000+4;
type
   TCislo = array[0..MAX] of 0..9;
```

Nultý prvek bude obsahovat hodnotu odpovídající počtu celých jednotek, od prvního prvku začínají desetinná místa.

Intervalový typ 0..9 místo prostého **byte** použijeme proto, abychom se dozvěděli o případných chybách přetečení.

Další úkol, který nás čeká, je výběr a implementace aritmetických operací.

Na první pohled se může zdát, že budeme potřebovat operace jako sčítání, násobení, dělení nebo alespoň převrácenou hodnotu...

Když odoláme pokušení začít hned programovat, ušetříme si hodně práce, protože přijdeme na to, že každý další člen řady můžeme spočítat z předchozího členu a to vydělením hodnotou **i**. A dělit dlouhé číslo hodnotou typu **integer** je snazší (pro výpočet i pro programátora) než dělení dlouhým číslem.

Konečný seznam operací, které potřebujeme provádět potom bude:

- dosazení jedničky
- součet
- vydělení hodnotou typu **integer**

A mohli bychom přidat ještě test, jestli je dlouhé číslo rovno nule, zatím, než se zase zamyslíme.

Teď už můžeme napsat hlavní program:

```
var  Soucet, Prvek: TCislo;
     i: integer;
begin
   Dosad1( Prvek );
   Soucet := Prvek;   { ano, i dlouhá čísla můžeme dosazovat }
     { začneme od součtu rovneho 1/0! a od i=1 }

   i := 1;
   while ...JeNula( Prvek )... do
   begin
      Pricti( Prvek, Soucet );
      i := i+1;
      Vydel( Prvek, i )
   end;

   { uz jen tisk: }
   write( Soucet[0],'.' );
   for i:=1 to 1000 do
      write( Soucet[i] )
end.
```

Schází nám procedury **Dosad1**, **Pricti**, **Vydel** a otečkovaná funkce **JeNula** určující, zda je hodnota dlouhého čísla rovna nule.

Začněme procedurami **Dosad1** a **Pricti**:

```
procedure Dosad1( var Kam: TCislo );
var  i: integer
begin
   Kam[0] := 1;
   for i:=1 to MAX do
      Kam[i] := 0
end;

procedure Pricti( var Co, Kam: TCislo );
{ >Co< předáváme také odkazem, kvůli rychlosti }
var i: integer;
    x, prenost: integer;
begin
   prenos := 0;

   for i:=MAX downto 0 do
   begin
      x := Co[i] + Kam[i] + prenos;
      Kam[i] := x mod 10;
      prenos := x div 10
   end;

   { tady by mohla být kontrola, zda opravdu prenos=0 }
end;
```

Za koncem cyklu v proceduře **Pricti** bychom mohli kontrolovat, zda opravdu **prenos** obsahuje nulu. V této úloze víme, že součet přetéci nemůže, ale jistota...

Určení ukládané čislice a přenosu bychom mohli zapsat názorněji takto:

```
      x := Co[i] + Kam[i] + prenos;
      if x < 10 then
      begin
         Kam[i] := x;
         prenos := 0
      end
                else
      begin
         Kam[i] := x-10
         prenos := 1
      end
```

Varianta využívající **mod** a **div** je rychlejší a kratší, zvolte si sami.
Nezkoušejte ušetřit pomocnou proměnnou **x**!

Pokud jde o dělení, ještě jednou vyjadřuji radost nad tím, že nám stačí dělit hodnotou typu **integer** a konkrétně (na tom záleží!) hodnotou menší než 1000.

Při dělení budeme procházet dělené číslo odpředu, postupně si pomocí Hornerova schématu skládat jeho hodnotu (a později aktuální zbytek po dělení) a to, že víme, že budeme dělit hodnotou menší než 1000, nám zajišťuje, že dělená hodnota bude menší než desetinásobek — a tedy se nám také vejde do typu **integer**.

Algoritmus dělení tedy bude takový, že budeme postupně odleva procházet dělené dlouhé číslo, v proměnné **zbytek** si budeme počítat aktuální zbytek po dělení, v každém kroku ho vynásobíme deseti a přičteme další číslici (Hornerovo schema) a vydělíme dělitelem.

Celočíselný podíl (0..9) zapíšeme místo právě použité číslice děleného dlouhého čísla, zbytek uložíme do proměnné **zbytek** a tak dál:

```
procedure Vydel( var Co: TCislo; Cim: integer );
var i: integer;
    zbytek: integer;
begin
   zbytek := 0;

   for i:=0 to MAX do
   begin
      zbytek := 10*zbytek + Co[i];
      Co[i] := zbytek div Cim;
      zbytek := zbytek mod Cim
   end
end;
```

Zbývá nám už jen test, zda je číslo (v hlavním programu proměnná **Prvek**) rovno nule. To by šlo snadno, ale...

Podívejme se ještě na proceduru **Vydel**. Jak bude postupovat výpočet prvků řady, na začátku **Prvku** bude jistě přibývat nul. Upravme proceduru **Vydel** tak, aby nuly na začátku přeskočila:

```
procedure Vydel( var Co: TCislo; Cim: integer );
var i: integer;
    zbytek: integer;
begin
   zbytek := 0;

   i := 0;
   while (i<=MAX) and (Co[i]=0) do Inc( i );

   for i:=i to MAX do
   begin
      zbytek := 10*zbytek + Co[i];
      Co[i] := zbytek div Cim;
      zbytek := zbytek mod Cim
   end
end;
```

Tím určitě nic nezkazíme, protože každý krok tohoto přidaného cyklu nám ušetří jeden krok cyklu počítajícího násobení deseti a dělení a výpočet zbytku po dělení.

Pojďme ale ještě dál a uložme si pozici první nenulové číslice do zvláštní proměnné **PNC** (jako **P**rví**N**enulová**C**islice):

```
   ...
   i := 0;
   while (i<=MAX) and (Co[i]=0) do Inc( i );
   PNC := i;

   for i:=PNC to MAX do
   ...
end;
```

A už zbývá si jen všimnout, že hodnota **PNC** se mezi jednotlivými voláními procedury **Vydel** nikdy nemůže zmenšovat, protože vydělením čísla, které má na začátku **(PNC-1)** nul určitě dostaneme číslo, které bude mít počet nul na začátku stejný nebo větší.

Proměnná **PNC** tedy může být globální a první nenulovou číslici stačí hledat až od ní:

```
procedure Vydel( var Co: TCislo; Cim: integer );
var i: integer;
    zbytek: integer;
begin
   zbytek := 0;

   i := PNC;
   while (i<=MAX) and (Co[i]=0) do Inc( i );
   PNC := i;

   for i:=PNC to MAX do
   begin
      zbytek := 10*zbytek + Co[i];
      Co[i] := zbytek div Cim;
      zbytek := zbytek mod Cim
   end
end;
```

Tím (proměnná **PNC** musí na začátku dostat hodnotu **0**!) nejen uspoříme procházení nulových číslic při každém dělení, ale také získáme bezplatně příznak toho, zda **Prvek** ještě neobsahuje samé nuly!

Hlavní program tedy bude vypadat takto:

```
var  Soucet, Prvek: TCislo;
     i: integer;
```

```
     PNC: integer;
begin
   Dosad1( Prvek );
   Soucet := Prvek;   { ano, i dlouhá čísla můžeme dosazovat }
     { začneme od součtu rovneho 1/0! a od i=1 }

   PNC := 0;
   i := 1;
   while PNC <= MAX do
   begin
      Pricti( Prvek, Soucet );
      i := i+1;
      Vydel( Prvek, i )
   end;

   { uz jen tisk: }
   write( Soucet[0],'.' );
   for i:=1 to 1000 do
      write( Soucet[i] )
end.
```

2 Další...

Operace odčítání probíhá podobně jako operace sčítání, nezapomeňte ale na to, že při odečítání dvou čísel se stejným znaménkem (jinak se odečítání mění ve sčítání) musíme vždy nejprve porovnat absolutní hodnoty obou čísel a vždy odečítat číslo menší (v absolutní hodnotě) od čísla většího (v absolutní hodnotě).

Za úvahu stojí i převést odečítané číslo na desítkový doplněk (všechny číslice doplnit do devítky a nakonec přičíst **1**) a ten přičítat.

Algoritmus násobení všichni známe.

Dělení dvou dlouhých čísel probíhá podobně jako jsme viděli při dělení hodnotou typu **integer**, jenom zbytek bude dlouhé číslo a operace **div a mod** musíme provádět odhadem a/nebo postupným odečítáním.