Balíček apnum – výpočty v TEXu s libovolnou přesností

Petr Olšák

TEX na primitivní úrovni není vybaven příliš velkou škálou možností pro numerické výpočty. Jistě znáte příkazy `\advance`, `\multiply` a `\divide`, takže víte, že vydělit dvě desetinná čísla je značně komplikovaný problém. Ani přidání nových primitivních příkazů do eTEXu (`\numexpr` a `\dimexpr`) situaci nezlepšilo. Pochopitelně vznikly zejména pro LATEX nejrůznější makrobalíčky umožňující pohodlnější numerické výpočty. Žádný z nich[1]) mě ale nezaujal a vytvořil jsem si balíček vlastní: `apnum.tex`[2]). Je to balíček pokud možno bez kompromisů, takže umožní nastavit libovolnou přesnost výpočtu a umožní nejen sčítat, odčítat, násobit a dělit, ale také počítat odmocniny a hodnoty funkcí sin, cos, tan, arcsin, arccos, arctan, exp a ln. Vše se počítá s přesností na `\apFRAC` desetinných míst za desetinnou tečkou[3]), což si můžete nastavit na libovolnou hodnotu. Samozřejmě, pokud budete chtít řádově stovky či tisíce desetinných míst, dostanete typicky neužitečný výsledek, a navíc si budete muset na něj chvíli počkat. Přesto jsem se pokusil při implementaci brát zřetel na maximální efektivitu výpočtu, takže složitější výpočty mi `apnum` vyhodnotí řádově stokrát rychleji, než srovnatelné balíčky pro LATEX.

Dokumentace k balíčku, soubor `apnum.pdf`, obsahuje nejen uživatelskou část, ale také velmi podrobnou část implementační, kde si čtenář může přečíst jednak popis a smysl fungování všech interních maker balíčku, a také tam najde podrobné vysvětlení všech numerických algoritmů, podle kterých byly jednotlivé vlastnosti balíčku implementovány.

1 Zápis výrazů

Po `\input apnum` má uživatel k dispozici makro `\evaldef`, které funguje jako `\edef`, ale tělo definice neexpanduje, nýbrž je numericky vyhodnotí. Vstupem může být libovolný výraz, ve kterém se vyskytují operátory `+`, `-`, `*`, `/` a `^`. Operátory `*`, `/` mají přednost před `+` a `-` a operátor celočíselné mocniny `^` má největší prioritu. Dále lze použít kulaté závorky k vyjádření jiného pořadí vykonávání operací než podle implicitního pravidla „zleva doprava“ a dle priorit. Například:

```
\evaldef\A {2+4*(3+7)}
   % ... makro \A obsahuje výsledek výpočtu 42
\evaldef\B {\the\pageno * \A}
   % ... makro \B obsahuje 42násobek čísla strany
\evaldef\C {123456789000123456789 * -1234567891234567891234567891}
   % ... \C obsahuje -152415787654473413441975318499559530997501905210
\evaldef\D {1.23456789 + 12345678.9 - \A}
   % ... \D obsahuje 12345596.13456789
\evaldef\X {1/3}
   % ... makro \X obsahuje .33333333333333333333
```

Je možné si povšimnout, že poslední výsledek při dělení nemá absolutní přesnost, ale je limitován počtem míst za desetinnou tečkou, kterou má `apnum` implicitně nastavenu na hodnotu `\apFRAC=20`. Registr `\apFRAC` můžete změnit dle svého uvážení. Přesnost výpočtu je absolutní při operacích `+`, `-`, `*` a `^`, zatímco při dělení a při výpočtu hodnot matematických funkcí (jako např. $\sin x$) se použije zaokrouhlení na `\apFRAC` desetinných míst.

Operandy, které vstupují do operací, jsou čísla ve formátu

⟨*znaménko*⟩⟨*číslice*⟩`.`⟨*číslice*⟩

[1]) Například `xint`, `fltpoint`, `bnumexpr`, `minifp`, aritmetika v TikZ.
[2]) http://www.ctan.org/pkg/apnum.
[3]) Pomocí registru `\apTOT` lze také omezit celkový počet desetinných cifer.

kde ⟨*znaménko*⟩ je konečná posloupnost znaků `+` a `-` (třeba i prázdná), která vyjadřuje znaménko „mínus“ právě tehdy, když je v posloupnosti lichý počet znaků `-`. Tuto podivnost známe z klasického TEXu, tak jsem ji v balíčku `apnum` rovněž respektoval. Dále ⟨*číslice*⟩ je konečná posloupnost číslic, tj. znaků `0` až `9`. Před i za desetinnou tečkou může být libovolně mnoho číslic a mohou chybět zcela (ne v obou případech současně). Chybí-li číslice za tečkou, není nutné tečku psát.

Kromě tohoto zápisu čísla je možné číslo zapsat v semilogaritmickém tvaru jako

⟨*znaménko*⟩⟨*číslice*⟩`.`⟨*číslice*⟩`E`⟨*exponent*⟩

ovšem pak je ve výsledném makru jen mantisa výpočtu a ⟨*exponent*⟩ je možné si přečíst odděleně v registru `\apE`. Důvod je prostý: chcete-li vypočítat třeba `3E+2000 * 5E+1300`, pak se `apnum` nebude před výpočtem obtěžovat s převodem čísel na základní tvar, tím by mohl akorát zahltit paměť. Místo toho spočítá `3*5=15` a exponent `3300` hledejte v registru `\apE`. Podrobnější informace o možnosti počítání v semilogaritmickém tvaru lze dohledat v dokumentaci `apnum.pdf`. Pro pochopení příkladů v tomto článku je dostačující vědět, že `\apEnum`⟨*makro*⟩ předpokládá v makru mantisu výsledku a neudělá nic, pokud je registr `\apE` nulový (tj. mantisa je přímo rovna číslu v základní tvaru). Jinak převede ⟨*makro*⟩ na základní tvar čísla, tedy přidá nuly nebo posune desetinou tečku.

Místo čísel mohou ve výrazech jako operandy vystupovat také konstrukce typu `\the`⟨*registr*⟩ a `\number`⟨*registr*⟩, což umožňuje pracovat s hodnotami TEXových registrů. Dále je možné jako operand použít makro, které bylo dříve definováno (pomocí `\def` nebo `\evaldef`) jako číslo, což umožňuje používat ve výrazech „proměnné“. Konečně operandem může být tzv. *funkční makro*, což je makro s žádným nebo více parametry, které vrací nějaký numerický výsledek. Příkladem funkčního makra s jedním parametrem je `\EXP`, které má jeden parametr typu ⟨*výraz*⟩ a pracuje tak, že vrátí hodnotu funkce e^x pro $x = ⟨\textit{výraz}⟩$. Takže třeba:

```
\evaldef\X{.25}
\evaldef\A{\EXP{2*\X} - 1} % v makru \A je výsledek výpočtu e^{2X} - 1
```

Uživatel si může definovat vlastní funkční makra. O tom pojednává následující sekce. Balíček `apnum` nabízí mimo jiné tato předdefinovaná funkční makra: `\SQRT`, `\EXP`, `\LN`, `\SIN`, `\COS`, `\TAN`, `\ASIN`, `\ACOS`, `\ATAN`. Jejich význam je zřejmý z jejich názvu. Tato makra mají jeden parametr, kterým může být výraz (jako při `\evaldef`) a je nutné jej obklopit svorkami. Všimněte si, že se tedy ve výrazech mohou vyskytovat jednak kulaté závorky pro určení pořadí vyhodnocování operací `+`, `-`, `*`, `/`, `^` a také svorky pro obklopení parametrů funkčních maker, což mohou být zase vnořené výrazy:

```
\def\A{20}
\evaldef\B{ 30*\SQRT{ \SIN{\PI/6} + 1.12*\the\widowpenalty } / (4-\A) }
```

Dále poznamenejme, že mezery ve výrazech jsou ignorovány.

Vyhodnocení jednotlivých operací pracuje na úrovni expand procesoru i hlavního procesoru. Balíček `apnum` tedy (na rozdíl o podobných LATEXových balíčků) neumožňuje numericky vyhodnotit výraz jen na úrovni expand procesoru. Důvodem je optimalizace rychlosti a možnost balíček použít v klasickém TEXu, který není vybaven expandujícími eTEXovými primitivy `\numexpr` a `\dimexpr`. Další velká výhoda tohoto přístupu souvisí s možností uživatelů vytvářet funkční makra. Nemusejí se starat o to, aby jejich funkční makra pracovala jen na úrovni expand procesoru.

Osobně se domnívám, že šikovný makro-programátor nepotřebuje expandující vyhodnocování výrazů, protože kdykoli může použít místo `\edef\cosi{...`⟨*výraz*⟩`...}` konstrukci

```
\evaldef\V{⟨výraz⟩}\edef\cosi{...\V...}
```

Při použití výrazů v parametrech asynchronního `\write` je možné do souboru zapsat výraz bez expanze a vyhodnotit jej až při následném čtení tohoto souboru.

Makro `\evaldef\cosi{`⟨*výraz*⟩`}` uloží výsledek vyhodnocení výrazu nejen do `\cosi`, ale také do makra `\OUT`. Kromě toho registr `\apSIGN` obsahuje znaménko výsledku (to se hodí při následném testu na znaménko výsledku) a registr `\apE` obsahuje exponent výsledku (to se hodí při použití čísel v semilogaritmickém tvaru).

2 Funkční makra si tvoříme sami

Balíček `apnum` nabízí (kromě maker zmíněných výše) ještě funkční makra pro celočíselné dělení, zbytek po celočíselném dělení, faktoriál, binomické koeficienty a několik dalších. Také jsou připravena funkční

makra `\PI` a `\PIhalf` bez parametru, která vrátí hodnotu konstant π a $\pi/2$ na `\apFRAC` desetinných míst.

Při tvorbě vlastních funkčních maker je třeba dodržovat dvě zásady:

- Makro musí jako první token obsahovat `\relax`. Podle toho vyhodnocovač výrazů pozná, že makro neobsahuje konstantu. Vytvoří skupinu v rámci které budou všechna přiřazení lokální a v ní předá funkčnímu makru řízení (na úrovni hlavního procesoru).
- Funkční makro musí vložit výsledek své činnosti do makra `\OUT` a do registrů `\apSIGN` a `\apE`.

Následuje několik příkladů funkčních maker.

2.1 Hyperbolický sinus

Existuje mnoho matematických funkcí, které v balíčku `apnum` nejsou připraveny. Důvodem je můj pocit, že zapamatovat si název funkce, kterou je třeba použít, je někdy stejně náročné, jako pamatovat si vzorec, podle něhož je ta funkce definována. Přitom pamatovat si to druhé je daleko užitečnější a tedy výchovnější. Třeba hyperbolický sinus je možné definovat pomocí:

```
\def\SINH#1{\relax \evaldef\myE{\EXP{#1}}\evaldef\OUT{(\myE - 1/\myE)/2}}
```

což odpovídá vzorečku

$$\sinh x = \frac{e^x - e^{-x}}{2}.$$

V makru je nejprve povinné `\relax`, pak je vyhodnocena hodnota e^x a uložena do pomocného makra `\myE`. Nakonec je využita identita $e^{-x} = 1/e^x$. Protože závěrečné `\evaldef` nastaví hodnoty registrů `\apSIGN` a `\apE` správně, není potřeba se o to starat.

Možná může čtenáře napadnout i následující implementace:

```
\def\SINH#1{\relax \evaldef\OUT{(\EXP{#1} - \EXP{-(#1)})/2}}
```

To bude fungovat také, ale je to pomalejší. Náročnou operaci `\EXP` je v tomto případě nutné vykonat dvakrát, zatímco prvně navržené řešení si vystačilo s jediným voláním `\EXP` a taky s jediným vyhodnocením vstupního výrazu.

Pro úplnost dodávám, jak by se dala definovat inverzní funkce k hyperbolickému sinu:

```
\def\ASINH#1{\relax \LN{#1+\SQRT{(#1)^2+1}}}
```

Zde je využita identita:

$$\operatorname{arcsinh} x = \ln\left(x + \sqrt{x^2+1}\right).$$

V uvedeném příkladu nebylo nutné explicitně definovat `\OUT`, protože tuto práci udělá předdefinované funkční makro `\LN`.

Řešení otázky, jak implementovat hyperbolický kosinus, tangens a jejich inverze přenechám laskavému čtenáři.

2.2 Dekadický sinus, převod stupňů na radiány

Funkční makra `\SIN`, `\COS` a `\TAN`, která jsou připravena v `apnum.tex`, předpokládají argument v radiánech. Mají tedy periodu 2π neboli `2*\PI`. Někdy je ale užitečné pracovat s těmito funkcemi při interpretaci argumentu ve stupních. Jednoduchá možnost, jak definovat funkce `\SINdeg`, `\COSdeg` a `\TANdeg`, akceptující argument ve stupních, může vypadat takto:

```
\def\SINdeg#1{\relax \SIN{(\PI/180)*(#1)}}
\def\COSdeg#1{\relax \COS{(\PI/180)*(#1)}}
\def\TANdeg#1{\relax \TAN{(\PI/180)*(#1)}}
```

Všimněte si, že argument `#1` je schován v kulaté závorce, protože může být například ve tvaru součtu.

Dále je potřeba vyřešit problém, že údaje ve stupních jsou typicky udávány v šedesátkové soustavě necelé části stupňů, tj. v minutách a vteřinách. Vytvoříme funkční makro `\DEG`, které vezme údaj zapsaný v šedesátkové notaci a převede ho na stupně v desítkovém zápisu čísla. Tedy

```
\DEG{12;30'45.756''}% znamená 12 stupňů, 30 minut, 45.756 vteřin
                                % takže
\evaldef\a{\DEG{12;30'45.756''}}% vrátí 12.51271
\evaldef\a{\DEG{12;30'}}%       % vrátí 12.5
\evaldef\a{\DEG{12}}%           % vrátí 12
\evaldef\a{\DEG{12.5}}%         % vrátí 12.5 (převod se neprovede)
```

Všimněte si, že šedesátkový převod se provede v případě, že za celými stupni je středník, zatímco tečka znamená, že následuje desetinný zápis čísla. Symbol pro minutu (') a pro vteřinu ('') na úplném konci zápisu je nepovinný, tj. `12.5` je též výstupem `\DEG{12;30}`. Makro `\DEG` může být definováno třeba takto:

```
\def\DEG#1{\relax \DEGa#1;''\relax}
\def\DEGa#1;#2'#3'#4\relax{\evaldef\OUT{#1}%
   \ifnum\apSIGN<0 \def\tmps{-}\else\def\tmps{+}\fi
   \DEGb#2;\relax{\OUT+\tmp/60}%
   \DEGb#3;\relax{\OUT+\tmp/3600}%
}
\def\DEGb#1;#2\relax#3{\edef\tmp{#1}%
   \ifx\tmp\empty \else \edef\tmp{\tmps\tmp}\evaldef\OUT{#3}\fi
}
```

Makro přečte pomocí `\DEGa` do `#1` stupně, do `#2` minuty a do `#3` vteřiny. V parametru `#4` je případný zbylý balast. Prvním `\evaldef` vyhodnotíme stupně. Je-li výsledek záporný, budou se případné minuty a vteřiny odečítat, takže do `\tmps` vložíme znak `-`, jinak tam vložíme znak `+`. Makro `\DEGb` odstraní z parametru případný středník a další nevyužité tokeny a očištěný parametr uloží do `\tmp`. Pak `\evaldef` přidá k výsledku minuty a analogicky jsou přidány vteřiny.

Máme-li makro `\DEG` hotové, můžeme definovat makra `\SINd`, `\COSd` a `\TANd`, která ve svém parametru rovnou akceptují šedesátkový zápis stupňů, jako to dělá makro `\DEG`, tedy třeba `\SINd{12;30}`.

```
\def\SINd#1{\relax \SIN{(\PI/180)*\DEG{#1}}}
\def\COSd#1{\relax \COS{(\PI/180)*\DEG{#1}}}
\def\TANd#1{\relax \TAN{(\PI/180)*\DEG{#1}}}
```

Je ovšem potřeba mít na paměti, že makra `\SINd`, `\COSd`, `\TANd` nemohou mít v argumentu plnohodnotný výraz více údajů v šedesátkovém zápisu. Bez šedesátkového zápisu (tj. bez použití středníku) zpracují ovšem argument jako obvyklý výraz.

2.3 Maximum

Vytvoříme funkční makro `\MAX{`⟨*výraz*⟩`,`⟨*výraz*⟩`,...,`⟨*výraz*⟩`}`, které má ve svém parametru libovolný počet čárkami oddělených výrazů (aspoň jeden) a vrátí maximální hodnotu ze všech uvedených výrazů.

Pro porovnávání dvou hodnot nelze použít přímo primitivní `\ifnum` nebo `\ifdim`, protože hodnoty mohou být příliš velké nebo naopak malé, ale lišící se třeba až na padesátém desetinném místě. Dokumentace `apnum.d` proto doporučuje použít makro `\TEST{`⟨*výraz1*⟩`}`⟨*relace*⟩`{`⟨*výraz2*⟩`}\iftrue`, kde znak ⟨*relace*⟩ je jeden ze znaků `<`, `>`, `=`. Přitom `\iftrue` se stane skutečným `\iftrue`, pokud je podmínka splněna, jinak se promění v `\iffalse`. Implementace makra `\TEST` se opírá o odčítání porovnávaných hodnot po kterém zjistíme znaménko výsledku:

```
\def\TEST#1#2#3#4{\evaldef\tmp{#1-(#3)}\ifnum\apSIGN #2 0 }
```

Povšimněte si mezery za nulou na konci makra. Ta mezera tam není jen pro srandu králíkům. Ona ukončuje podmínku `\ifnum`. Kulatá závorka v parametru `\evaldef` taky není úplně zbytečná, protože druhý porovnávaný výraz může být třeba ve tvaru součtu.

Vlastní implementace makra `\MAX` může vypadat takto:

```
\newcount\mynum
\def\TEST#1#2#3#4{\evaldef\tmp{#1-(#3)}\ifnum\apSIGN #2 0 }
\def\MAX#1{\relax \MAXa#1,,}
\def\MAXa#1,{\evaldef\maxOUT{#1}\mynum=\apSIGN \MAXb}
\def\MAXb#1,{\ifx,#1,\let\OUT=\maxOUT \apSIGN=\mynum \else
   \evaldef\maxNEXT{#1}%
```

```
    \ifnum\apSIGN>\mynum \mynum=\apSIGN \let\maxOUT=\maxNEXT
    \else \TEST\maxNEXT>\maxOUT \iftrue \let\maxOUT=\maxNEXT \fi\fi
    \expandafter \MAXb \fi
}
```

Makro `\MAXa` připraví hodnotu prvního výrazu do `\maxOUT` a znaménko této hodnoty vloží do `\mynum`. Pak se opakovaně volá `\MAXb` na další výrazy až do doby, kdy je parametr prázdný. Je-li prázdný, činnost se ukončí tím, že se nastaví `\OUT` na `\maxOUT` a `\apSIGN` na `\mynum`. Není-li parametr prázdný, je třeba jej vyhodnotit (je uložen v `\maxNEXT`) a dále je třeba jej porovnat s `\maxOUT`. Nejprve se porovnávají jen znaménka. Je-li znaménko nově vyhodnoceného výrazu větší, obnoví se znaménko v `\mynum` i hodnota `\maxOUT`. Jinak se provede `\TEST`. Je-li `\maxNEXT` větší než `\maxOUT`, obnoví se hodnota `\maxOUT`.

Vytvoření analogického makra `\MIN` přenechám laskavému čtenáři. Upozorňuji, že není třeba opisovat celý kód znovu, stačí definovat makro `\mmREL`, které naplníte znakem `<` nebo `>` a v kódu makra `\MAXb` nahradíte výskyty znaku `>` makrem `\mmREL`.

2.4 Lineární interpolace

Vytvoříme makra `\setF...\endF`, kterými lze naplnit pomyslnou funkci `\F` tabulkou hodnot zhruba takto:

```
\setF
   \F{2} = 15 ;
   \F{3} = 10 ;
   \F{8} = 11 ;
\endF
```

Mezi `\setF` a `\endF` se opakují údaje `\F{`⟨*výraz*⟩`} =` ⟨*výraz*⟩`;` v libovolném množství. Tímto způsobem uživatel vymezí pro funkci `\F` konečně mnoho funkčních hodnot. Dále definujeme funkční makro `\F{`⟨*výraz*⟩`}` které vrátí hodnotu podle předem zadaných hodnot s užitím lineární interpolace. Například `\F{2.5}` po výše uvedené deklaraci vrátí hodnotu 12,5.

Vzhledem k tomu, že naším úkolem je mít v ukázce makro co nejjednodušší a nepředvádět zde implementaci algoritmu řazení podle velikosti, budeme předpokládat, že uživatel zadal vstupní hodnoty pro `\F` ve vzestupném pořadí. Například v předchozí ukázce to je splněno, protože $2 < 3 < 8$. Dále budeme předpokládat, že mimo krajní hodnoty (v našem příkladě mimo interval $[2, 8]$) je funkce `\F` nedefinovaná a při pokusu o vyhodnocení mimo definiční obor se vypíše hlášení „out of range“.

Makro `\setF` naplní `\Flist` seznamem za sebou jdoucích vstupních hodnot oddělených středníkem (v našem příkladě `2;3;8;`) a dále definuje makra `\F:1` s první funkční hodnotou, `\F:2` s druhou funkční hodnotou atd. Makro `\setF` je definováno takto:

```
\newcount\mynum
\def\setF{\mynum=0 \def\Flist{}\setFa}
\def\endF{konec setF}
\def\setFa#1{\ifx#1\endF \else \expandafter \setFb \fi}
\def\setFb#1#2#3;{\evaldef\X{#1}\apEnum\X \evaldef\Y{#3}\apEnum\Y
   \advance\mynum by1
   \expandafter\edef\csname F:\the\mynum\endcsname{\Y}%
   \edef\Flist{\Flist\X;}%
   \setFa
}
```

Funkční makro `\F` vyhodnotí vstupní parametr $x =$ `\X` a projde postupně `\Flist` až najde místo, kde je x mezi dvěma po sobě jdoucími hodnotami (nebo se přímo rovná první z těchto hodnot). V takovém případě jsou tyto hodnoty označeny $a =$ `\A`, $b =$ `\B`, jsou vyhodnoceny funkční hodnoty $F(a) =$ `\FA`, $F(b) =$ `\FB` a je použit vzorec pro lineární interpolaci

$$\texttt{\OUT} = F(x) = F(a) + \frac{(x-a)\,\big(F(b)-F(a)\big)}{b-a}.$$

```
\def\TEST#1#2#3#4{\evaldef\tmp{#1-(#3)}\ifnum\apSIGN #2 0 }
\def\F#1{\relax \evaldef\X{#1}\apEnum\X \expandafter\Fa\Flist;\endF}
```

```
\def\Fa#1;{%
   \TEST{#1}>\X \iftrue \Fe \expandafter \Fc \fi % out of range
   \def\A{#1}\mynum=0
   \Fb
}
\def\Fb#1;{%
   \advance\mynum by1
   \ifx;#1;\TEST\X=\A \iftrue \evaldef\OUT{\Xa}%
                      \else \Fe \fi % out of range
      \expandafter \Fc \fi
   \TEST{#1}>\X \iftrue
      \def\B{#1}\edef\FA{\csname F:\the\mynum\endcsname}%
      \advance\mynum by1 \edef\FB{\csname F:\the\mynum\endcsname}%
      \evaldef\OUT{\FA + (\X-\A)*(\FB-\FA) / (\B-\A)}% linear intetrpolation
      \expandafter \Fc
   \fi
   \def\A{#1}%
   \Fb
}
\def\Fc#1\endF{}
\def\Fe{\def\OUT{0}\apSIGN=0 \message{F{\X} OUT OF RANGE}}
```

3 Tisk a vyhodnocování vzorců ze společného zdroje

Balíček `apnum` vymezuje syntaxi pro výrazy, které pak umí vyhodnocovat. Tato syntaxe je popsána v prvním sekci tohoto článku. Kromě vyhodnocování umí `apnum` od verze 1.5 (z ledna 2016) tytéž výrazy tisknout v matematickém módu. Pro tisk je automaticky vyhledána nejvhodnější forma, jakou matematici běžně po staletí používají. Vzorec je tedy ze vstupní syntaxe konvertovnán do běžné matematické syntaxe a vytištěn v matematickém módu. K tomu slouží makro `\eprint{`⟨*výraz*⟩`}{`⟨*deklarace*⟩`}`. V ⟨*deklaraci*⟩ je možné mimo jiné lokálně předefinovat proměnné například na vhodná písmena.

Pro ilustraci této vlastnosti vytvoříme makro `\ep{`⟨*výraz*⟩`}`, které jednak vytiskne výraz a dále připojí rovnítko a přidá hodnotu výrazu. Předpokládá se užití proměnných `\X`, `\Y`, `\Z`.

```
\def\vars{\def\X{x}\def\Y{y}\def\Z{z}}     % písmena místo konstant
\def\ep#1{$\displaystyle \eprint{#1}\vars  % tisk výrazu
   \evaldef\OUT{#1}\ROUND\OUT6\corrnum\OUT % vyhodnocení výrazu
   \ifx\XOUT\empty =\else\doteq\fi \OUT$}  % tisk výsledku
```

Dříve, než začneme výrazy s proměnnými x, y, z vyhodnocovat, je třeba proměnné naplnit konkrétními hodnotami. Třeba

```
\def\X{0.51} \def\Y{-2.7}  \def\Z{17}
```

Nyní zkusíme zadat k tisku i vyhodnocení několik výrazů:

`\ep{(\X^2+1)/((\X+1)*(\X-2))}`	$\dfrac{x^2+1}{(x+1)\cdot(x-2)} \doteq -0{,}560069$
`\ep{-((\X^2-1)/((\X+1)*(\X-1)))}`	$-\dfrac{x^2-1}{(x+1)\cdot(x-1)} = -1$
`\ep{\SIN{\Y}^2 + \COS{\Y}^2}`	$\sin^2 y + \cos^2 y \doteq 0{,}999999$
`\ep{\ASIN{\X} + \ATAN{\X+1}}`	$\arcsin x + \operatorname{arctg}(x+1) \doteq 1{,}521041$
`\ep{\SIN{\PI/4}}`	$\sin\dfrac{\pi}{4} \doteq 0{,}707106$
`\ep{\SQRT{2}/2}`	$\dfrac{\sqrt{2}}{2} \doteq 0{,}707106$
`\ep{\PI}`	$\pi \doteq 3{,}141592$
`\ep{\FAC{\Z}}`	$z\,! = 355687428096000$

`\ep{\SQRT{\iFLOOR{\Y}^2+1}}`	$\sqrt{\lfloor y \rfloor^2 + 1} \doteq 3{,}162277$
`\ep{\iFLOOR{\Y} + \iFRAC{\Y}}`	$\lfloor y \rfloor + \{y\} = -2{,}7$
`\ep{\LN{\X/\Y^2}+1}`	$\ln \frac{x}{y^2} + 1 \doteq -1{,}659848$
`\ep{\LN{\X/\Y^2+1}}`	$\ln \left(\frac{x}{y^2} + 1 \right) \doteq 0{,}067620$
`\ep{-3*(\X+\Y)}`	$-3 \cdot (x + y) = 6{,}57$
`\ep{(\X+\Y)*-3}`	$(x + y) \cdot (-3) = 6{,}57$
`\ep{-3*-(\X+\Y)}`	$-3 \cdot (-(x + y)) = -6{,}57$
`\ep{\BINOM{5}{0}+\BINOM{5}{1}+\BINOM{5}{2}}`	$\binom{5}{0} + \binom{5}{1} + \binom{5}{2} = 16$
`\ep{\BINOM{5}{3}+\BINOM{5}{4}+\BINOM{5}{5}}`	$\binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 16$
`\ep{2^5/2}`	$\frac{2^5}{2} = 16$
`\ep{4^3^2}`	$4^{3^2} = 262144$
`\ep{(4^3)^2}`	$\left(4^3\right)^2 = 4096$
`\ep{\EXP{\LN{2}+\LN{3}}}`	$\mathrm{e}^{\ln 2 + \ln 3} \doteq 5{,}999999$

Povšimněte si, že `apnum` při tisku vzorců šetří v duchu tradičního matematického zápisu závorkami, ale někdy přidá závorku další, například `-3*-(\X+\Y)` vytiskne jako $-3 \cdot (-(x + y))$. Mocniny tiskne rovněž tam, kde má, tj. `\SIN{\X}^2` se promění na $\sin^2 x$.

Pokud výše uvedené makro `\ep` vyzkoušíte, shledáte, že tiskne ve výsledcích anglické desetinné tečky místo českých desetinných čárek. A jméno funkce arctg se vytiskne jako arctan. Pro účely české sezby jsem dále místo závěrečného `\OUT` v těle makra `\ep` napsal `\expandafter\dotcomma\OUT.\relax` a definoval jsem:

```
\def\dotcomma#1.#2\relax{#1\ifx\relax#2\relax\else{,}\dotcomma#2\relax\fi}
```

Toto makro vymění tečku za čárku. Dále jsem modifikoval názvy funkcí definováním makra `\apEPj` takto:

```
\def\apEPj{%
   \def\TAN{\apEPf{tan}}%
   \def\ATAN{\apEPf{arctg}}%
}
```

Makro `\apEPj` se spustí uvnitř skupiny v úvodu činnosti makra `\eprint` a umožní uživateli nebo spíše programátorovi maker deklarovat názvy funkcí a případná další nastavení.

4 Poznámky k implementaci apnum

Podrobný popis všech algoritmů, které balíček `apnum` používá, je v technické části dokumentace v souboru `apnum.pdf`. Tato sekce pouze shrnuje nejpodstatnější myšlenky.

Sčítání, odčítání, násobení a dělení probíhá v `apnum` stejně, jako to dělají žáci, když se učí počítat. Rozdíl je pouze v tom, že žáci počítají s ciframi desítkové soustavy a využívají znalosti malé násobilky do stovky a schopnosti sčítání do dvaceti. Balíček `apnum` ale pracuje s „ciframi“ v soustavě o základu 10000, tedy „cifra“ má typicky čtyři desítkové číslice. Využívám toho, že TEX zvládá „malou násobilku“ s těmito „ciframi“ jednoduše pomocí `\multiply`, neboť TEXové registry pojmou čísla do cca $2 \cdot 10^9$. Například k pronásobení dvou deseticiferných čísel (v desítkové soustavě) tedy nepotřebujeme vykonat 100 dílčích násobení, ale stačí jich jen 9.

TEX není schopen náhodného přístupu do své paměti jinak než pomocí definic nových maker. Testy ukázaly, že se nevyplatí pro každou „cifru“ alokovat nové makro (pomocí `\csname...\endcsname`), ale účelnější je data rozprostřít (expandovat) do vstupní fronty a odtud je zpětně odebírat. Problém ale je, že takto je možné číst sekvenčně jen jeden datový proud, zatímco v algoritmech sčítání resp. násobení

potřebujeme jednak číst cifry v opačném pořadí a také typicky ze dvou vstupních operandů. Makro tedy při rozkladu čísla na čtyřciferné „cifry“ a při konverzi těchto „cifer“ do opačného pořadí vytvoří speciální datový typ, ve kterém jsou „cifry“ vzájemně střídány z obou operandů. Taková data se pak vychrstnou do vstupního proudu a TEX je dokáže svižně přečíst a třeba sečíst.

Pokud se pracuje s příliš velkými čísly (mají mnoho cifer), pak činnost TEXu poněkud zdržuje opakované expanze těchto rozsáhlých dat do vstupní fronty. Proto jsem se snažil potřebu takových expanzí minimalizovat. Například je jednoduché načíst rozsáhlá data do parametru `#1` a tento parametr třeba šestkrát použít v pracovním makru v různých větvích vymezených pomocí `\if`, ale člověk si musí uvědomit, že se celé toto makro se šestkrát opakovanými rozsáhlými daty vyvrhne do vstupního proudu a to není optimální. Je tedy lepší v úvodu makra napsat `\def\tmp{#1}` a dále v těle makra šestkrát použít `\tmp`.

Dělení probíhá pomocí „ocásku dílčích zbytků“, jak to známe ze školy. Algoritmus v `apnum` optimalizuje dělení, při kterém je ve jmenovateli jediná „cifra“, tj. (do čtyř desítkových cifer). Pak je složitost algoritmu lineární vzhledem k počtu požadovaných cifer na výstupu. Při dělení rozsáhlými jmenovateli pak `apnum` vytváří speciální střídané datové typy, aby se co nejvíce ušetřil čas.

Pochopitelně naprosto nelze srovnávat v rychlosti výpočtu TEX (interpretr) s kompilovanými programy, jakými je například unixový program `bc`. Ten mi ostatně dobře posloužil při testech a ověřování správnosti výsledků. Na druhé straně jsem učinil několik srovnání výkonu balíčku `apnum` s podobnými makrobalíčky určenými pro LATEX a byl jsem potěšen, že mé algoritmy pracují typicky výrazně rychleji.

Už jako malý kluk jsem na MFF UK dělal semestrální práci na téma „dlouhá čísla“. Bylo to tehdy na sálovém počítači EC 1040, kam jsem opakovaně dodával děrné štítky a zhruba den jsem pak čekal, co z toho vyleze. Už tehdy jsem si říkal, že by bylo krásné rozšířit schopnost sčítání, odčítání, násobení a dělení velkých čísel na počítání v libovolné přesnosti typických matematických funkcí, které máme zadrátovány v kalkulačce (sqrt, sin, exp atd.). Technické možnosti tehdejší doby a zejména čas mě to ale neumožnily. Implementoval jsem tehdy jen výpočet odmocniny cifru po cifře podobným algoritmem, jakým probíhá „ocáskovité dělení“.

Jako ještě menší kluk (to mi bylo zhruba patnáct) jsem chodil do výpočetního střediska v místě svého bydliště v Ostravě na sálový počítač EC 1010, kde mě jeden inženýr seznámil s tajemstvím zásobníku a s tím, jak je možné zásobník použít na interpretaci matematických výrazů s operacemi různých priorit. Už tehdy jsem na to vytvořil program ve Fortranu. A tato zkušenost se mi hodila při tvorbě balíčku `apnum`. Ačkoli to výpočetní středisko už dávno odnesl čas a jméno toho inženýra jsem zapomněl, tímto mu děkuji. Interpret výrazů v `apnum` je částečně i jeho zásluha.

Jakmile jsem zprovoznil `apnum` se sčítáním, odčítáním, násobením, dělením a s interpretací výrazů, nabídla se mi druhá šance implementovat všechny funkce z kalkulačky. Nakonec proč ne, pravidelně ve výuce při zmínce o Taylorově polynomu říkám, že i ten je v kalkulačkách *nějak* použit, ale nikdy jsem přesně neměl rozmyšleno, jak. Nyní jsem si na to opravdu sáhnul. A byl jsem překvapen, že když se udělá několik málo prostých fíglů, je konvergence i na pomalém interpretru při relativně velkém množství cifer překvapivě rychlá.

Druhou odmocninu jsem v `apnum` implementoval pomocí Newtonovy metody. V ní je třeba najít první přiblížení k hodnotě $\sqrt{a}$. V něm spustit tečnu ke grafu funkce $x^2 - a$ a průsečík této tečny s osou x je druhé přiblížení. Tento krok se opakuje tak dlouho, až je přiblížení dostatečně přesné. Pro první přiblížení jsem se rozhodl použít lineární interpolaci funkce $\sqrt{x}$ na intervalu $[1, 100]$ se známými hodnotami v bodech $1, 4, 9, 16, \ldots, 81, 100$. Výpočet prvního přiblížení se provede pomocí klasických TEXových operací s registry typu $\langle dimen \rangle$. Lineární interpolace se nejvíc odchyluje od požadovaného výsledku na intervalu $(1, 4)$, takže tam jsem se rozhodl přidat ještě jeden „známý“ bod 2 s přibližně vypočtenou hodnotou $\sqrt{2}$. Při požadavku na přesnost 20 desetinných míst stačí při použití této metody provést zhruba 5 iterací. Při každé iteraci se počet přesně spočítaných cifer zhruba zdvojnásobí, takže na padesát cifer stačí zhruba 6 iterací. Je-li argument mimo interval $[1, 100]$, posuneme mu desetinou tečku o sudý počet desetinných míst M a tím ho převedeme na hodnotu v uvedeném intervalu. Ve výsledku pak posuneme desetinnou tečku zpět o $-M/2$ desetinných míst. Myšlenka využívá faktu, že $\sqrt{100} = 10$.

Na exponenciálu se hodí dobře známá Taylorova řada se středem nula. Je ale neúčelné do této řady dosazovat velké hodnoty vstupního argumentu x. Provedou se tedy nejprve následující úpravy argumentu. Je-li argument záporný, využije se identita $e^{-x} = 1/e^x$ a dále hledáme exponenciálu kladného argumentu. Je-li argument x větší než 4, vypočteme $d = \lfloor x/\ln 10 \rfloor$, použijeme identitu

$$e^x = e^{x - d\cdot\ln 10} \cdot 10^d$$

a nyní stačí vyhodnotit exponenciálu argumentu $x' = x - d \cdot \ln 10 \in [0, \ln 10] \subset [0, 4)$ a ve výsledku posunout desetinnou čárku o d desetinných míst. Je-li argument větší než 2, vydělíme jej dvěma. Je-li argument větší než 1, vydělíme jej dvěma. Nyní máme argument v intervalu $[0, 1)$ a můžeme použít Taylorovu řadu. Byl-li argument dělený dvěma, výsledek umocníme a byl-li dělený dvakrát dvěma, tak výsledek dvarát umocníme. To vyplývá z identity $e^{2x} = (e^x)^2$.

Taylorova řada díky faktoriálu ve jmenovateli konverguje rychle. Můžete namítnout, že přes dvacet sčítanců řady pro přesnost 20 cifer je mnoho (viz $20! \approx 10^{19}$), ale tyto sčítance lze spočítat rychle díky tomu, že každý další sčítanec řady lze odvodit z předchozího jediným násobením a jedním rychlým dělením (dělíme „jednociferným“ číslem, tj. číslem do velikosti 10000).

Na logaritmus je použita pomalu konvergentní Taylorova řada odvozená z inverze k hyperbolické tangentě

$$\ln x = 2 \operatorname{arg\,tanh} \frac{x-1}{x+1} = 2\left(\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \cdots\right).$$

Tato řada konverguje obstojně jen pro hodnoty x velice blízké jedné. Do blízkého okolí jedničky je ale možné argument dopravit jednoduchým fíglem. Vypočteme $A = x/\exp(\widetilde{\ln x})$, kde $\widetilde{\ln x}$ je přibližná hodnota logaritmu v x. Protože $\exp(\ln x) = x$, je A tím blíže jedné, čím přesnější je odhad $\widetilde{\ln x}$. Dále se použije výše uvedená řada pro argument A, tedy se spočítá přesně $\ln A$. Konečně platí $\ln x = \ln A + \widetilde{\ln x}$, protože $x = A \cdot \exp(\widetilde{\ln x})$ a $\ln(ab) = \ln a + \ln b$. Dále si povšimneme, že stačí umět spočítat $\ln x$ pro $x \in [1, 10]$. Pokud je x mimo tento interval, pak existuje $x' \in [1, 10)$ tak, že $x = x' \cdot 10^M$ a $\ln x = \ln x' + M \cdot \ln 10$. K vyhledání x' stačí posunout desetinnou tečku o M míst doleva, tj. nejsou nutné žádné náročné výpočty. Často používanou hodnotu $\ln 10$ má algoritmus `apnum` uloženu v paměti. Přibližnou hodnotu $\widetilde{\ln x}$ počítám lineární interpolací na intervalu $[1, 10]$. Ta se od přesného výsledku liší na druhém desetinném místě, totéž tedy platí pro odchylku hodnoty A od jedné. V každém sčítanci Taylorovy řady uvedené výše se přesnost výsledku zvýší aspoň o další čtyři místa, protože řada obsahuje jen liché mocniny.

Aby se člověk dočkal hodnot funkcí $\sin x$ a $\cos x$ pomocí známých Taylorových řad, je potřebné, aby byl vstupní argument menší než jedna. Pak řady konvergují rychle díky faktoriálu ve jmenovateli. Aby byl argument v intervalu $[0, 1)$, je třeba posunout argument o vhodný násobek periody (beze změny výsledku) nebo o půlperiodu (se změnou znaménka výsledku). Je-li argument mimo interval $[0, \pi/2)$, je možné ho tam přesunout užitím rovností $\sin x = \sin(\pi - x)$, $\cos x = -\cos(\pi - x)$. Také je možné argument přesunout do intervalu $[0, \pi/4)$ využitím identity $\cos x = \sin(\pi/2 - x)$, respektive $\sin x = \cos(\pi/2 - x)$. Nakonec tedy máme argument v intervalu $[0, \pi/4)$, je tedy menší než jedna a můžeme použít Taylorovu řadu.

Abychom mohli posunovat argument funkcí $\sin x$ a $\cos x$ v duchu předchozího odstavce, potřebujeme znát hodnotu konstanty π na požadovanou přesnost. Tuto konstantu má `apnum` předpočítánu na 30 desetinných míst a uloženu v paměti. Pokud někdo chce větší přesnost, spustí se automaticky v `apnum` nový výpočet konstanty π užitím Chudnovského řady, která má velmi dobrou konvergenci. Každým krokem se konstanta π zpřesní zhruba o 14 dalších desetinných míst. Jediným problémem je nutnost spočítat $\sqrt{640320}$ na potřebný počet desetinných míst. Proto má `apnum` uloženu hodnotu této odmocniny na 12 desetinných míst v paměti a tu použije jako výchozí pro start Newtonovy metody pro odmocninu. Tím se ušetří několik prvních kroků Newtonovy metody.

Funkce $\arctan x$ je implementována pomocí řady pro převrácenou hodnotu svého argumentu

$$\arctan \frac{1}{x} = \frac{x}{1+x^2} + \frac{2}{3}\frac{x}{(1+x^2)^2} + \frac{2}{3}\frac{4}{5}\frac{x}{(1+x^2)^3} + \frac{2}{3}\frac{4}{5}\frac{6}{7}\frac{x}{(1+x^2)^4} + \cdots$$

která konverguje dobře pro $x > 1$. Je-li $x \in (0, 1)$, lze užít rovnosti $\arctan x = \pi/2 - \arctan 1/x$ a pro záporný argument se využije toho, že je to funkce lichá.

Ostatní běžné matematické funkce se dají vyjádřit přímým vzorečkem obsahujícím už implementované funkce.

5 Závěrečná třešnička

Makrobalík `apnum` se opírá jen o primitivní příkazy klasického TEXu a o základní makro plainTEXu `\newcount`. Z toho důvodu funguje ve všech implementacích TEXu s libovolným formátem. To je pro vesměs všechna má TEXová makra obvyklé. Na druhé straně, pokud uživatel LATEXu vytvoří makro, často

ho znepřístupní ostatním tím, že na začátek svého souboru napíše `\NeedsTeXFormat{LaTeX2e}`. Někdy to bývá dost frustrující. Onehdy jsem se potýkal například s velice dobrým makrem `qrcode.sty` na vytváření QR kódů přímo pomocí TEXového `\vrule` a `\hrule`. Makra toho souboru obsahují z 90 % TEX-primitivní obraty a dále tam bylo asi 10 % LATEX-specifických věcí. Napsal jsem autorovi přání, aby svůj kód vyčistil od všeho LATEXového, že pak jeho makro bude fungovat pod jakýmkoli TEXem, což bude nesporně další výhoda tohoto makra. Neodpověděl. Tak jsem celý den strávil trasováním cizích maker a vykosťováním těch LATEX-specifických věcí. Vytvořil jsem odnož původního makrobalíku `qrcode.tex`, která dělá totéž co `qrcode.sty`, ale funguje všude. Tuhle práci jsem si mohl ušetřit, pokud by se uživatelé LATEXu neuzavírali zbytečně do své sekty tím, že míchají TEX-primitivní obraty s něčím, co je závislé na LATEXu. Například ve zmíněném `qrcode.sty` se můžeme setkat s mixem registrů alokovaných pomocí `\newcount` a pomocí `\newcounter`, s mixem užití primitivního `\advance` a LATEXového `\addtocounter` atd.

V návaznosti na tuto zkušenost jsem se rozhodl, že do svého makra `apnum` přidám na závěr tento kód:

```
\ifx\documentclass\undefined \else % please, don't remove this message
\message{WARNING: the author of apnum package recommends: Never use LaTeX.}\fi
```

Jinými slovy, pokud je `apnum` použito v LATEXu, pak se na terminál a do log souboru přidá navíc hlášení

```
WARNING: the author of apnum package recommends: Never use LaTeX.
```

Tedy: „VAROVÁNÍ: autor balíčku apnum doporučuje: nikdy nepoužívejte LATEX.“ LATEXoví uživatelé typicky nečtou pořádně text na terminálu ani v logu (protože toho tam mají velmi mnoho), takže jim jedna zpráva navíc nemusí vadit.