Zpravodaj Československého sdružení uživatelů TeXu

Peter Wilson; Jan Šustek
Mělo by to fungovat V – Cykly

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 26 (2016), No. 1-4, 121–127

Persistent URL: http://dml.cz/dmlcz/150258

Mělo by to fungovat V – Cykly

Peter Wilson

Abstrakt

Článek ukazuje, jak je možné v LaTeXu procházet řetězce a zpracovávat je znak po znaku. Dále článek popisuje LaTeXové makro `\@for` a jeho použití ukazuje na příkladu sazby tabulky.

Klíčová slova: LaTeX, řetězce, cykly, `\@for`.

> 'Tis better to be lowly born
> And range with humble livers in content
> Than to be perked up in a glist'ring grief
> And wear a golden sorrow.
>
> *Henry VIII*
> William Shakespeare

Cílem tohoto seriálu je ukázat čtenáři krátké kousky kódu, které mohou vyřešit některé z jeho problémů. Doufám, že situaci ještě více nezkomplikuji v důsledku mých chyb. Opravy, poznámky a návrhy na změny budou vždy vítány.

> To no one but the Son of Heaven does it belong to order ceremonies, to fix the measures, and to determine the written characters.
>
> *The Analects*
> Confucius

1. Práce s řetězci

Ve svém dřívějším článku (Wilson, 2005) jsem se zmínil, že se ještě později vrátím k práci s řetězci. Následující makra se mohou použít k postupnému zpracování všech znaků v jednoduchém řetězci.

```
\chardef\catcodeG=\catcode`\^^G
\catcode`\^^G=12
\newcommand*{\vsechnyznaky}[1]{%
  \def\argument{#1}\ifx\argument\@empty\else
  \vsechny@znaky#1^^G\fi}
```

Z anglického originálu *Glisterings* (Wilson, 2007) přeložil Jan Šustek.

doi: 10.5300/2016-1-4/121

```
\def\vsechny@znaky#1#2^^G{%
  \def\znak{#1}\def\dalsi{#2}%
  \ifx\znak\@empty\let\next\@gobble
  \else
    \zpracujznak{#1}%
    \ifx\dalsi\@empty \let\next\@gobble
    \else \let\next\vsechny@znaky \fi
  \fi
  \next#2^^G}
\catcode`\^^G=\catcodeG
```

V makrech jsem použil speciální znak `^^G` jako oddělovač konce řetězce. Za normálních okolností TEX bere tento znak jako neplatný (a některé balíčky jej mění na aktivní znak), ale zde jsem dočasně změnil jeho kategorii na „ostatní“ znak. Makro `\@gobble` je v LATEXu definováno jako makro, které vezme jeden argument a nic s ním neudělá. Makro `\vsechnyznaky` při své expanzi zavolá makro `\zpracujznak{`⟨*znak*⟩`}` pro každý znak v řetězci. Makro `\zpracujznak` můžeme definovat například následovně.

```
\newcommand*{\zpracujznak}[1]{\textit{#1}}
```

Potom makro `\vsechnyznaky` dá následující výsledky.

```
\vsechnyznaky{vsechnyznaky}                    vsechnyznaky
\vsechnyznaky{{\oe}rsted}                      œrsted
\vsechnyznaky{}
\vsechnyznaky{slova s mezerami}                slovasmezerami
```

Speciální případ prázdného argumentu řeší přímo makro `\vsechnyznaky`, zatímco ostatní případy jsou předány makru `\vsechny@znaky`. Toto makro volá samo sebe a přitom postupně načítá jednotlivé znaky původního řetězce. Tomuto procesu se říká *řetězová rekurze*, což znamená, že poslední věc, kterou makro udělá, je, že zavolá samo sebe, případně neudělá nic, pokud se má rekurze ukončit.

Připomínám, že pokud v LATEXu píšete zdrojový kód, ve kterém se vyskytují makra obsahující zavináč, pak je musíte psát uvnitř balíčku (v souboru .sty), nebo je musíte vložit mezi makra `\makeatletter` a `\makeatother`.

Nepříjemnou vlastností makra `\vsechnyznaky` je, že pohltí všechny mezery v zadaném řetězci. V následujícím řešení mezery zpracujeme ve dvou krocích. Nejdříve projdeme řetězec po jednotlivých slovech, kde za slovo považujeme posloupnost znaků oddělenou mezerou. Ve druhém kroku projdeme slovo po jednotlivých znacích.

Nejdříve si nastavíme potřebné věci a definujeme hlavní makro `\vsechnaslova`.

```
\newif\if@zacatekslova
\def\textrelax{\relax}
```

```
\chardef\catcodeG=\catcode`\^^G
\catcode`\^^G=12
\newcommand*{\vsechnaslova}[1]{%
  \vsechna@slova#1^^G}
\def\vsechna@slova#1^^G{%
  \nactislovo#1 ^^G}
```

Makro `\nactislovo` oddělí následující slovo od zbytku řetězce (argument je oddělený mezerou). Pak zavolá makro `\nactiznak`, jehož argumentem je ono slovo.

```
\long\def\nactislovo#1 {%
  \@zacatekslovatrue
  \nactiznak#1\relax}
```

Makro `\nactiznak` načte následující „znak“ slova. Pokud je tento „znak“ `^^G`, znamená to konec celého řetězce. Pokud je tento znak „`\relax`“, znamená to konec slova a musí se znovu zavolat makro `\nactislovo` na zbytek řetězce. Ve zbývajících případech se na tento znak aplikuje makro `\zpracujznak` a znovu se zavolá makro `\nactiznak`.

```
\def\nactiznak#1{%
  \def\znak{#1}%
  \if\znak^^G\let\next\relax
  \else
    \ifx\znak\textrelax
      \let\next\nactislovo
    \else
      \zpracujznak{#1}%
      \let\next\nactiznak
    \fi
  \fi
  \next}
\catcode`\^^G=\catcodeG
```

Makro `\nactiznak` je dalším příkladem použití řetězové rekurze.

Nyní makro `\zpracujznak` nadefinujeme trochu komplikovaněji. Makro otestuje, zda se znak nachází na začátku slova. Pokud ano, pak se znak převede na velké písmeno a vysází se kurzívou. Pokud ne, pak se znak vysází tučně. Tento příklad asi v praxi moc nevyužijeme, nicméně je to ukázka toho, jakým způsobem můžeme zpracovávat znaky v řetězci.

```
\newcommand*{\zpracujznak}[1]{%
  \if@zacatekslova
    \space\textit{\MakeUppercase{#1}}%
```

```
    \@zacatekslovafalse
  \else \textbf{#1}\fi}
```

Následuje několik příkladů.

```
\vsechnaslova{retezec s mezerami}
\vsechnaslova{{\oe}rsteduv zakon}
```

Retezec *S* *M*ezerami
Œrsteduv *Z*akon

Tato makra budou fungovat dobře pro jednoduché řetězce, ale velmi pravděpodobně budou dávat nesprávné výsledky, pokud řetězce budou obsahovat akcentované znaky nebo jiný materiál narušující plynulé čtení znaků. Na druhou stranu, asi by bylo jednodušší a rychlejší změnit řetězce ručně v textovém editoru.

Here we go loop de loop.
Here we go loop de li.
Here we go loop de loop
On a Saturday night.

Loop de loop
Johnny Thunder?

2. Cykly

Občas potřebujeme opakovaně provádět akci, která se netýká zpracování řetězců. V TEXu pro tyto účely existují makra `\loop`...`\repeat`. Používají se následovně

```
\loop
  ⟨příkazy⟩
\if ⟨podmínka⟩
  ⟨další příkazy⟩
\repeat
```

TEX nejdříve vykoná ⟨*příkazy*⟩ a pak vyhodnotí ⟨*podmínku*⟩ (u tohoto příkazu `\if` se nesmí vyskytovat `\else` ani `\fi`). Pokud je ⟨*podmínka*⟩ splněna, TEX provede ⟨*další příkazy*⟩, vrátí se zpět a pokračuje znovu ⟨*příkazy*⟩. Pokud ⟨*podmínka*⟩ není splněna, přeskočí TEX ⟨*další příkazy*⟩ a pokračuje za `\repeat`.

LATEX poskytuje mechanismus, který umožňuje procházet seznam položek oddělených čárkami (jako například seznam voleb třídy nebo balíčku). Obecná syntaxe je

```
\@for\promenna:=⟨seznam⟩\do{⟨příkazy⟩}
```

kde ⟨*seznam*⟩ obsahuje hodnoty oddělené čárkami. V cyklu se do makra `\promenna` postupně uloží jednotlivé hodnoty a s každou se provedou ⟨*příkazy*⟩.

Jak cyklus pracuje, si ukážeme na příkladu. Následuje očesaná verze maker z třídy memoir (Wilson, 2004). Makra umožňují zarovnat seznam věcí do tabulky, aniž bychom se zabývali ukončováním řádků. Hlavní makro

58 `\fillrows{`⟨*šířka*⟩`}{`⟨*počet*⟩`}{`⟨*seznam*⟩`}`

vytvoří vycentrovanou tabulku o celkové šířce ⟨*šířka*⟩, která má ⟨*počet*⟩ sloupců a v jednotlivých buňkách jsou položky ⟨*seznamu*⟩. Položky se do tabulky zapíšou v přirozeném pořadí, tj. zleva doprava a potom shora dolů. Nápad na makro `\fillrows` jsem dostal po přečtení *TEX for the Impatient* (Abrahams a kol., 1990), kde byla popsána verze pro TEX, která tabulku vyplňovala v pořadí shora dolů a potom zleva doprava.

Většina následujícího kódu je podobná kódu ukrytému v definici prostředí `tabular` a nebudu se snažit tento kód vysvětlovat.

Nejdříve nadeklarujeme čítače a délkové registry, které budeme potřebovat.

```
\newcount\CT@cols      % počet sloupců
\newcount\@cellstogo   % počet zbývajících sloupců
\newdimen\CT@col@width % šířka sloupce
\newtoks\crtok
  \crtok = {\cr}%
```

Nyní již můžeme definovat makro `\fillrows`. Makro má tři argumenty – celkovou šířku, počet sloupců a seznam položek. Na začátku makra nastavíme čítače podle počtu sloupců.

```
\newcommand{\fillrows}[3]{\par\begingroup
  \CT@cols=#2\relax
  \@cellstogo=\CT@cols
```

Další část kódu se provede po vložení každé položky do tabulky. Buď vloží `&`, nebo alternativu makra `\\`.

```
  \def\@endcolactions{%
    \global\advance\@cellstogo\m@ne
    \ifnum\@cellstogo<\@ne
      \global\@cellstogo=\CT@cols
      \the\crtok
    \else
      &
    \fi}%
```

Dále se spočítá šířka sloupců a nastaví se rozvržení tabulky.

```
  \CT@col@width=#1
  \divide\CT@col@width \CT@cols
  \penalty 10000\relax
  \noindent
  \vskip -\z@
  \def\@preamble{}%
```

```
81  \begingroup
82    \let\@sharp\relax
```

Nyní pomocí cyklu `\loop`...`\repeat` projdeme všechny sloupce kromě posledního a vždy přidáme k makru `\@preamble` vhodné mezery a znak `&`.

```
83    \ifnum\CT@cols>\@ne
84      \loop
85        \g@addto@macro{\@preamble}{%
86          \hb@xt@ \CT@col@width
87            {\strut\relax\@sharp\hfil} &}%
88        \advance\CT@cols\m@ne
89      \ifnum\CT@cols>\@ne
90      \repeat
91    \fi
```

Za posledním sloupcem znak `&` nebude.

```
92    \g@addto@macro{\@preamble}{%
93      \hb@xt@ \CT@col@width
94        {\strut\relax\@sharp\hfil}}%
95  \endgroup
```

(Výše uvedený kód nastaví šířku každého sloupce na hodnotu `\CT@col@width`. Pokud zakomentujeme řádky 86 a 93, bude šířka každého sloupce nastavena na přirozenou hodnotu podle jeho nejširší položky.[1]) Nyní již můžeme dokončit přípravné práce na tabulce.

```
96  \let\@sharp ##
97  \tabskip\fill
98  \halign to\hsize \bgroup
99    \tabskip\z@
100   \@preamble
101   \tabskip\fill\cr
```

Jednotlivé položky se do tabulky vloží cyklem `\@for`.

```
102   \@for\@tempa:=#3\do{%
103     \@tempa\unskip\space\@endcolactions}%
```

Na tomto místě již je tabulka hotová a můžeme ukončit definici makra `\fillrows`.

```
104   \the\crtok \egroup \endgroup \par}
```

Jako jednoduchý příklad vytvoříme následující tabulky.

[1] Potom samozřejmě bude první argument makra `\fillrows` nepodstatný, jako je tomu v příkladu na řádcích 107–109. (pozn. překl.)

```
\fillrows{0.5\textwidth}{4}{jedna,dvě,tři,čtyři,pět,%
  šest,sedm,osm,devět,deset}
```

jedna	dvě	tři	čtyři
pět	šest	sedm	osm
devět	deset		

```
\fillrows{0.5\textwidth}{7}{A,tady,je,výsledek,použití,makra,%
  \tt\char`\\fillrows,při,sazbě,do,sedmi,sloupců,nastavených,%
  na,svou,přirozenou,šířku.}
```

A	tady	je	výsledek	použití	makra	`\fillrows`
při	sazbě	do	sedmi	sloupců	nastavených	na
svou	přirozenou	šířku.				

Reference

Abrahams, Paul W., Berry, Karl, Hargreaves, Kathryn A. *TEX for the Impatient.* Boston, MA, USA : Addison-Wesley, 1990. 357 s. ISBN 0-201-51375-7. Dostupné na CTAN v adresáři `info/impatient/`.

Wilson, Peter. *The memoir class for configurable typesetting.* 2004. Dostupné na CTAN v adresáři `latex/macros/contrib/memoir/`.

Wilson, Peter. Glisterins [on-line]. *TUGboat*, 2005, **26**(3), s. 253–255. [cit. 2017-01-09]. (ISSN 0896-3207.) Dostupné na: `https://www.tug.org/TUGboat/tb26-3/tb84glister.pdf`.

Wilson, Peter. Glisterins [on-line]. *TUGboat*, 2007, **28**(1), s. 12–14. [cit. 2017-01-09]. (ISSN 0896-3207.) Dostupné na: `https://www.tug.org/TUGboat/tb28-1/tb88glister.pdf`.

Summary

This paper shows how to use process strings, character by character, in LaTeX. The paper also describes LaTeX macro `\@for` and shows its application for typesetting tables.

Keywords: LaTeX, strings, loops, `\@for`.

Peter Wilson, herries.press@earthlink.net
18912 8th Ave. SW
Normandy Park, WA 98166 USA