Zpravodaj Československého sdružení uživatelů TeXu

Jan Šustek

Načítání souboru s argumenty v TeXu

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 25 (2015), No. 1-2, 86–94

Persistent URL: http://dml.cz/dmlcz/150230

Načítání souboru s argumenty v TEXu

Jan Šustek

Abstrakt: Cílem článku je simulovat v TEXu načítání souboru s argumenty z terminálu. Bude představeno několik řešení tohoto problému, od jednodušších po složitější. Přitom si čtenář připomene některá pravidla TEXu, s nimiž se tak často nesetká. Jako ukázku vytvoříme soubor brozura.tex, který po zavolání z terminálu `pdftex brozura kniha` načte dokument kniha.pdf a přerovná jeho strany do souboru brozura.pdf. Ten pak lze vytisknout jako brožuru.

Klíčová slova: TEX, argumenty, brožura, scanargs.

1 Popis problému

Běžně program TEX voláme s argumentem, kterým je název vstupního souboru. Když zavoláme

```
1 tex vstup
```

bude TEX zpracovávat vstupní soubor vstup.tex. Cílem článku je nasimulovat načítání vstupního souboru s „jeho“ argumenty, například

```
2 tex vstup arg1 arg2
```

Ukažme si nejprve, co se stane, když TEX zavoláme jako na řádku 2. Znaky

```
3 vstup arg1 arg2
```

jsou TEXu poslány jako nultý vstupní řádek. Input procesor tyto znaky načte jako[1]

```
4 \input vstup arg1 arg2§
```

kde § je znak s hodnotou `\endlinechar`.

Vstupní soubor obvykle obsahuje makro `\end{document}` nebo `\bye`, případně jinou ukončovací značku. Když hlavní procesor TEXu dostane příkaz k ukončení, přestane číst další vstupní text a po nezbytných formalitách ukončí svoji činnost. Ke znakům

```
5 arg1 arg2§
```

se tak TEX nedostane.

[1] Příkaz `\input` se provede pouze interně a token `input` ve čtecí frontě ve skutečnosti nebude.

 DOI: 10.5300/2015-1-2/86

2 Návrh řešení

Aby se TEX k uvedeným znakům dostal, nesmí se hlavní procesor při čtení vstupního souboru dostat k žádnému ukončovacímu příkazu. V takovém případě se vstupní soubor celý přečte a čtecí fronta bude pokračovat znaky na řádku 5. TEX se ale nějak ukončit musí, proto do vstupního souboru musíme ukončovací příkaz vložit trikově a musíme zajistit, aby se k němu hlavní procesor dostal až po přečtení znaků na řádku 5.

Základní myšlenka je jednoduchá – napsat na konec vstupního souboru řádky

```
6 \catcode\endlinechar=2
7 \afterassignment\bye%
8 \def\ARGS{%
```

Nastavení `\catcode` způsobí, že se znak `§` na řádku 5 promění na token $\boxed{\S}_2$ a bude se chovat jako `}`. Nastavení `\afterassignment` způsobí, že se po přiřazení `\def\ARGS` vloží do čtecí fronty token $\boxed{\texttt{bye}}$. Pokud na konec vstupního souboru napíšeme řádky 6–8, bude čtecí fronta po zavolání řádku 2 a provedení řádku 8 vypadat následovně:

```
9 \def\ARGS{arg1 arg2}\bye
```

A máme, co jsme chtěli. Dostali jsme se k argumentům a trikově jsme ukončili běh TEXu.[2]

Problém je, že toto jednoduché řešení nemůže fungovat. Důvodem je konec vstupního souboru uvnitř definice makra `\ARGS`. TEX totiž při ukončení načítání souboru testuje mimo jiné, zda právě nenačítá definici makra. Pokud k tomu dojde, TEX předpokládá, že je to chyba uživatele, který zapomněl napsat uzavírací závorku za definicí. V tom případě TEX nahlásí chybu a ukončí načítání definice.

Řešením je ukončit načítání souboru před zahájením definice. K tomu se použije příkaz `\noexpand` na úplném konci souboru. Příkaz `\noexpand` načte ze čtecí fronty jeden token. Jelikož okamžitě za příkazem končí soubor, je čtení souboru ukončeno a čtecí fronta pokračuje znaky na řádku 5. Příkaz `\noexpand` tak přečte token $\boxed{\texttt{a}}_{11}$. Je víceméně vedlejší, že se tím tokenu přiřadí příznak *no_expand_flag*. Důležité je, že k tomu dojde před zahájením definice. TEX pak čte tokeny $\boxed{\texttt{def}}$ $\boxed{\texttt{ARGS}}$ $\boxed{\{}_1$ $\boxed{\texttt{a}}_{11}$ z paměti a během čtení definice nenarazí na konec souboru. Zbývá zajistit, aby se TEX k příkazu `\noexpand` skutečně dostal dříve než k příkazu `\def`. K tomu poslouží příkaz `\expandafter`. Funkční řešení na posledních řádcích vstupního souboru vypadá následovně.

```
10 \catcode\endlinechar=2
11 \afterassignment\bye%
12 \expandafter\def\expandafter\ARGS\expandafter{\noexpand%
```

[2] Řádky 7 a 8 je nutné zakončit procenty, aby token procesor nevytvořil token $\boxed{\S}_2$ i na koncích těchto řádků.

Jako příklad vytvoříme soubor vypis.tex, který po zavolání

```
13 tex vypis nejake argumenty
```

vypíše na terminál

```
14 Argumenty jsou nejake argumenty.
```

Rozdíl oproti řádkům 10–12 bude v tom, že se po přiřazení neukončí běh TEXu, ale že se zavolá makro `\RUN`, které vypíše řádek 14 a až poté ukončí běh TEXu. Připomínám, že `\bye` je `\outer` makro, a proto se token `bye` nesmí vyskytnout uvnitř definice jiného makra.

```
15 \def\RUN{\message{Argumenty jsou \ARGS.}
16   \csname bye\endcsname}
17 \catcode\endlinechar=2
18 \afterassignment\RUN%
19 \expandafter\def\expandafter\ARGS\expandafter{\noexpand%
```

Všimněme si, že pokud budou argumenty obsahovat nějaká makra, vypíšou se tato po úplné expanzi. Čtenář si lehce domyslí, jak tuto expanzi při výpisu potlačit.

3 Využití registrů `\toks`

Řádky 17–19 nevypadají z uživatelského pohledu příliš přívětivě. Lépe by vypadalo, kdybychom si nadefinovali makro `\SCAN` a uživatel by pak na konci souboru psal pouze

```
20 \SCAN
```

Zde ale narazíme na další problém, a sice na nepárovou závorku na řádku 19, která se uvnitř definice makra `\SCAN` nesmí vyskytnout. Problém lze vyřešit pomocí registru `\toks`, který místo otevírací závorky umožňuje použít token `bgroup`, jenž nepodléhá kontrole na párování závorek. V následujícím řešení je navíc nastavení `\catcode` umístěno do skupiny, aby nemohlo způsobit další komplikace v případě uživatelovy neopatrnosti.[3]

```
21 \def\SCAN{\begingroup \catcode\endlinechar=2
22   \afterassignment\RUN
23   \toks0=\expandafter\bgroup\noexpand}
24 \def\RUN{\xdef\ARGS{\the\toks0}\endgroup
25   \message{Argumenty jsou \ARGS.}
26   \csname bye\endcsname}
27 \SCAN%
```

[3] Na konci řádku 22 už procento být nemusí, protože ke změně `\catcode` dojde až na řádku 27, kdy už je konec řádku 22 dávno zpracován.

Použití `\xdef` na řádku 24 neprovede úplnou expanzi argumentů, ale úplnou expanzi `\the\toks0`. Jejím výsledkem jsou neexpandované tokeny přesně tak, jak je zadal uživatel na terminálu.

Vadou na kráse ovšem stále zůstává znak `%` na řádku 27, který je v tuto chvíli nutný, aby se potlačil znak konce řádku 27, ze kterého by vznikl token §$_2$. Tím by se ale ukončilo načítání `\toks0` a k argumentům z terminálu bychom se nedostali.

V následujícím řešení tohoto faktu využijeme a odstraněním procenta ukončíme přiřazení do `\toks0`. Na toto místo totiž můžeme vložit další token pomocí `\afterassignment`. A expanzí tohoto tokenu analogicky načteme argumenty z terminálu. Takto dostáváme nový soubor vypis.tex, který už uvedenou vadu na kráse nemá.

```
\def\SCAN{\begingroup \catcode\endlinechar=2
  \afterassignment\SCANc
  \toks0=\bgroup}
\def\SCANc{\afterassignment\RUN
  \toks2=\expandafter\bgroup\noexpand}
\def\RUN{\xdef\ARGS{\the\toks2}\endgroup
  \message{Argumenty jsou \ARGS.}
  \csname bye\endcsname}
\SCAN
```

Pro pochopení se podrobně podívejme, jak se bude postupně měnit čtecí fronta na řádku 36, pokud zavoláme řádek 13. Text na následujících řádcích, který není v rámečcích, je text již přečtený input procesorem, ale zatím nedotčený token procesorem.

```
\SCAN§nejake argumenty§
SCAN §nejake argumenty§
begingroup catcode endlinechar =12 212 ␣10
    afterassignment SCANc toks 012 =12 bgroup §nejake argumenty§
toks 012 =12 bgroup §2 nejake argumenty§
SCANc nejake argumenty§
afterassignment RUN toks 212 =12
                  expandafter bgroup noexpand nejake argumenty§
toks 212 =12 bgroup n11 e11 j11 a11 k11 e11 ␣10
                                a11 r11 g11 u11 m11 e11 n11 t11 y11 §2
RUN
```

4 Příprava brožury

V dalším příkladu, inspirovaném makry Petra Olšáka (2016), vytvoříme užitečný soubor brozura.tex, který po zavolání

```
pdftex brozura kniha
```

načte dokument kniha.pdf a přeuspořádá jeho strany do souboru brozura.pdf, jehož oboustranným vytištěním a přeložením dostaneme brožuru.

Základ souboru je stejný jako v předcházejících příkladech. Rozdíl je jen v makru `\RUN`. V něm se nejprve příkazem `\pdflastximagepages` zjistí počet stránek souboru kniha.pdf a pak se v cyklu po jednoduchém výpočtu vloží příslušné stránky do souboru brozura.pdf. Je použit příkaz `\shipout`, kterým se obejde algoritmus stránkového zlomu i output rutina a bez nutnosti nastavování záhlaví a zápatí se příslušný box přímo vysází do souboru brozura.pdf.

Soubor brozura.tex vypadá následovně.

```
\def\SCAN{\begingroup \catcode\endlinechar=2
  \afterassignment\SCANc
  \toks0=\bgroup}
\def\SCANc{\afterassignment\RUN
  \toks2=\expandafter\bgroup\noexpand}
\newcount\aL \newcount\aR \newcount\bL \newcount\bR
\def\RUN{\xdef\nazev{\the\toks2}\endgroup
  \pdfpagewidth=297mm \pdfpageheight=210mm
  \pdfhorigin=0pt \pdfvorigin=0pt
  \pdfximage{\nazev.pdf}
  \aL=\pdflastximagepages
  \advance\aL by3 \divide\aL by4 \multiply\aL by4
  \aR=1 \bL=2 \bR=\aL \advance\bR by-1
  \loop
    \shipout\hbox{\strana\aL \strana\aR}
    \shipout\hbox{\strana\bL \strana\bR}
    \advance\aR by2 \advance\aL by-2
    \advance\bR by-2 \advance\bL by2
  \ifnum\aL>\aR
  \repeat
  \csname end\endcsname}
\def\strana#1{%
  \ifnum#1>\pdflastximagepages
    \hbox to.5\pdfpagewidth{\hss}%
  \else
    \pdfximage width.5\pdfpagewidth page#1 {\nazev.pdf}%
    \pdfrefximage\pdflastximage
  \fi}
\SCAN
```

Pokud se soubor brozura.tex umístí do adresářové struktury TEXu, může uživatel pohodlně psát řádek 45 přímo v adresáři, v němž se nachází soubor kniha.pdf a v němž se pak vytvoří soubor brozura.pdf.

Soubor brozura.tex by se samozřejmě mohl vylepšit. Mohl by například testovat, zda náhodou uživatel nezadal název souboru včetně přípony .pdf, zda zadaný soubor existuje, a podobně. To ale není náplní tohoto příkladu.

5 Řešení v balíčku

Cílem této části je uložit dříve definovaná makra do balíčku scanargs.tex, aby mohl uživatel zprovoznit načítání argumentů z terminálu pouhým načtením balíčku.

```
... \input scanargs ...
\bye
```

Naše řešení spočívá v tom, že se zevnitř balíčku scanargs.tex načte zbytek hlavního souboru až po závěrečné `\bye`, pak se uloží argumenty z terminálu a vrátí se k načtenému textu. Problém ale je, že pokud text jednou načteme, není pak možné v něm měnit kategorie znaků, což občas uživatelé potřebují. Dále není možné načíst text, který obsahuje nějaké `\outer` makro, například `\newcount`.

Toto lze obejít dvouprůchodovým načítáním hlavního souboru. Balíček scanargs.tex změní kategorie speciálních znaků na 12 a zbytek hlavního souboru přeskočí. Přeskakované znaky bude načítat jako verbatim a jejich separátorem bude `\bye`, konkrétně tokeny $\boxed{\backslash}_{12}$ $\boxed{\texttt{b}}_{11}$ $\boxed{\texttt{y}}_{11}$ $\boxed{\texttt{e}}_{11}$. Dále se načtou a uloží argumenty z terminálu. Pak se znovu načte hlavní soubor a opět jako verbatim se přeskočí začátek souboru až po text `\input scanargs`, konkrétně po tokeny $\boxed{\backslash}_{12}$ $\boxed{\texttt{i}}_{11}$ $\boxed{\texttt{n}}_{11}$ $\boxed{\texttt{p}}_{11}$ $\boxed{\texttt{u}}_{11}$ $\boxed{\texttt{t}}_{11}$ $\boxed{\sqcup}_{12}$ $\boxed{\texttt{s}}_{11}$ $\boxed{\texttt{c}}_{11}$ $\boxed{\texttt{a}}_{11}$ $\boxed{\texttt{n}}_{11}$ $\boxed{\texttt{a}}_{11}$ $\boxed{\texttt{r}}_{11}$ $\boxed{\texttt{g}}_{11}$ $\boxed{\texttt{s}}_{11}$. Tím činnost balíčku scanargs.tex končí a zbytek hlavního souboru se zpracuje obvyklým způsobem, včetně koncového `\bye`.

V následujícím výpisu souboru scanargs.tex se navíc na řádcích 87–88 a 91–105 zjišťuje počet argumentů oddělených mezerou a po jednom je ukládá do paměti. Počet argumentů je pak uložen v registru `\ARGSCOUNT` a i-tý argument lze získat zavoláním `\ARG[`i`]`. Při počítání argumentů je jako pomocný separátor použit token $\boxed{\texttt{\^{}\^{}A}}_{8}$, jehož použití v argumentech je velmi nepravděpodobné.

```
\def\SCAN{\begingroup
  \def\do##1{\catcode`##1=12}\dospecials\SCANa}
{\catcode`\|=0 \catcode`\\=12 |long|gdef|SCANa#1\bye{|SCANb}}
\def\SCANb{\endgroup
  \begingroup\catcode\endlinechar=2
  \afterassignment\SCANc
  \toks0=\bgroup}
```

```
\def\SCANc{\afterassignment\RUN
  \toks2=\expandafter\bgroup\noexpand}
\def\RUN{\xdef\ARGS{\the\toks2}\endgroup
  \expandafter\expandafter\expandafter\SAVE
    \expandafter\ARGS\space^^A%
  \begingroup\def\do##1{\catcode`##1=12}\dospecials
  \expandafter\SKIP\input\jobname\relax}
\newcount\ARGSCOUNT
\def\SAVE#1 #2^^A{%
  \def\next{#1}%
  \ifx\next\empty
  \else
    \advance\ARGSCOUNT1
    \expandafter\def\csname ARG[\the\ARGSCOUNT]\endcsname{#1}%
  \fi
  \def\next{#2}%
  \ifx\next\empty
    \let\next\relax
  \else
    \def\next{\SAVE#2^^A}%
  \fi
  \next}
\def\ARG[#1]{\csname ARG[#1]\endcsname}
{\catcode`\|=0 \catcode`\ =12 \catcode`\\=12
|long|gdef|SKIP#1\input scanargs{|endgroup}}
\expandafter\SCAN\noexpand
```

Balíček scanargs.tex lze načíst víceméně na libovolném místě před prvním použitím makra `\ARG` nebo `\ARGSCOUNT`. V případě, že by uživatel definoval makro se stejným jménem, jaké má nějaké pomocné makro v balíčku scanargs.tex, nevadí to, pokud makro definuje až po načtení balíčku.

Balíček scanargs.tex bude fungovat, jestliže bude hlavní soubor končit řádkem 76 s nepotlačeným koncem řádku. Bude fungovat i v případě, že za řádkem 76 budou prázdné řádky s komentářem. Pokud ale za řádkem 76 bude následovat prázdný řádek bez komentáře, vznikne z něj token $\boxed{\S}_2$, a argumenty se z terminálu nenačtou.

6 Ukázka použití balíčku

V následujícím jednoduchém příkladu si ukážeme použití balíčku scanargs.tex. Vytvoříme dokument, ve kterém budou vypsány jednotlivé argumenty, a to jak v expandovaném, tak neexpandovaném tvaru.

```
\input opmac
\input scanargs
\rightskip0ptplus1fil
\def\bezzacatku#1>{}
Počet argumentů: \the\ARGSCOUNT\par
\tmpnum=0
\loop
  \ifnum\tmpnum<\ARGSCOUNT \advance\tmpnum1
  Argument~\the\tmpnum: \ARG[\the\tmpnum]\csname par\endcsname
  Argument~\the\tmpnum\ bez expanze: {\tt
   \expandafter\expandafter\expandafter\expandafter
   \expandafter\expandafter\expandafter\bezzacatku
   \expandafter\expandafter\expandafter\meaning
     \ARG[\the\tmpnum]}
   \csname par\endcsname
\repeat
\bye
```

7 Problém nula argumentů

Nyní se zaměříme na související problém, který se autorovi tohoto článku nepodařilo vyřešit. Týká se všech dříve uvedených způsobů načítání argumentů.

Předpokládejme, že uživatel na terminálu načte soubor vypis.tex a nezadá argumenty, tj. zavolá TEX pouze

```
tex vypis
```

Rozeberme si podrobně tuto situaci. Čtecí fronta se bude vyvíjet následovně.

```
\input vypis§
```

Příkaz `\input` si vyžádá tokeny, ze kterých poté složí název souboru.

v$_{11}$ ypis§
v$_{11}$ y$_{11}$ pis§
v$_{11}$ y$_{11}$ p$_{11}$ is§
v$_{11}$ y$_{11}$ p$_{11}$ i$_{11}$ s§
v$_{11}$ y$_{11}$ p$_{11}$ i$_{11}$ s$_{11}$ §

Znak `\endlinechar` má v tuto chvíli (na začátku běhu TEXu) kategorii 5, proto se znak § promění na token ␣$_{10}$.

v$_{11}$ y$_{11}$ p$_{11}$ i$_{11}$ s$_{11}$ ␣$_{10}$

Token s ASCII-hodnotou 32 slouží jako oddělovač názvu souboru a je hlavním procesorem pohlcen. Čtecí fronta tak zůstane prázdná a důležitý token §$_2$ se

nevytvoří. Pak se načte soubor vypis.tex. Makra na konci tohoto souboru očekávají, že ve čtecí frontě ještě nějaké znaky jsou. To však není pravda, a proto TEX vypíše na terminál hvězdičku a tím uživatele vyzve ke vložení znaků.

Uživatel pak píše znaky úplně stejně, jako by je psal na konci řádku 127. To znamená, že se za tyto znaky automaticky vloží token $\boxed{\S}_2$. Tím se ukončí načítání `\toks2`, za nímž se již pak makro `\RUN` vykoná obvyklým způsobem.

Pokud na výzvu TEXu uživatel žádné znaky nenapíše a pouze stiskne enter, pak bude ve čtecí frontě pouze znak `§` vložený input procesorem. Z tohoto znaku se stane token $\boxed{\S}_2$ a vše bude opět v pořádku. Registr `\toks2` a makro `\ARGS` budou prázdné.

Aby uživatel nebyl zmaten hvězdičkou vypsanou na terminálu, bylo by dobré, kdyby existoval způsob, jak za běhu TEXu zjistit, zda je čtecí fronta prázdná, nebo zda ještě něco obsahuje. Autorovi článku se však takový způsob najít nepodařilo.

Literatura

Olšák, P. *TEX pro pragmatiky.* Brno: CSTUG, v tisku. 150 s.
ISBN 978-80-901950-1-1.

Summary: Reading Files with Arguments in TEX

The paper shows a possibility in TEX how to input a file with arguments from the terminal. There are simple solutions and also more complicated solutions with several advanced rules of TEX explained. As an application there is file brochure.tex. The user inputs this file by `pdftex brochure book`. Then TEX reads the document book.pdf and reorders its pages into file brochure.pdf so that it can be printed as a brochure.

Key words: TEX, arguments, brochure, scanargs.

Jan Šustek, jan.sustek@osu.cz
Ostravská univerzita, Přírodovědecká fakulta, Katedra matematiky
30. dubna 22, CZ-701 03 Ostrava, Czech Republic