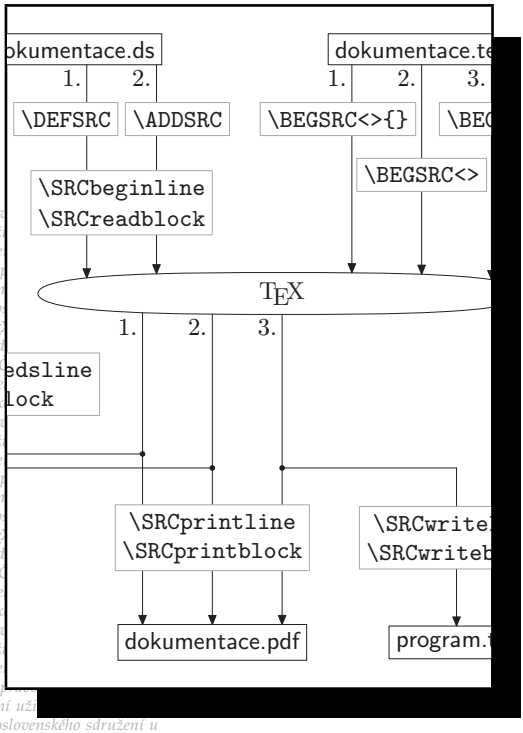
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

3–4

2023

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 33

OBSAH

Vít Starý Novotný: Úvodník 61

Vít Starý Novotný: CSTUG na konferenci TUG 2023 63

Jan Šustek: Generování dokumentovaného zdrojového souboru po blocích v TEXu 66

Jan Šustek: Jak umožnit stránkový zlom uvnitř vložených obrázků 102

Vít Starý Novotný: Markdown 3: Co je nového a co se chystá? 111

Ondřej Sojka, Petr Sojka, Jakub Máca: A Roadmap for Universal Syllabic Segmentation 125

Barbara Beeton: Co by každý (LA)TEXový nováček měl znát 139

Vít Starý Novotný: Sazba textu české lidové písně „Když jsem já sloužil“ pomocí modulu l3seq jazyka expl3 153

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Vydaná čísla Zpravodaje v elektronické podobě (PDF) jsou bezodkladně veřejně vystavena na webové adrese `https://www.cstug.cz/`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor `(.zip, .arj, .tar.gz)`, na e-mailovou adresu `bulletin@cstug.cz`. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live).

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Úvodník

Vít Starý Novotný

Milé čtenářky a čtenáři, TEXistky a TEXisté!

Vítejte u druhého letošního dvoučísla Zpravodaje CSTUGu, ve kterém pokračujeme ve snaze poskytovat cenné informace a poznatky ze světa TEXu. 45 let od svého vzniku si TEX udržuje své nezastupitelné místo jako nástroj pro kvalitní sazbu dokumentů a my jsme hrdí na to, že můžeme být součástí jeho rozvoje.

Číslo Zpravodaje, které držíte v rukou, obsahuje řadu zajímavých článků:

- *CSTUG na konferenci TUG 2023* od autora tohoto úvodníku popisuje letní účast členů CSTUGu na konferenci TUG 2023 v Bonnu.
- *Generování dokumentovaného zdrojového souboru po blocích v TEXu* od Honzy Šustka přináší nový pohled na tvorbu dokumentovaných zdrojů. Článek zpracovává téma, o kterém Honza přednášel na konferenci OSS Conf 2019 v Žilině a na TUGu 2023. Honza bude přednášet také na valném shromáždění CSTUGu, které se koná v sobotu 16. prosince 2023 v Brně [1].
- *Jak umožnit stránkový zlom uvnitř vložených obrázků*, také od Honzy Šustka, ve kterém se autor zabývá řešením specifických sazebních výzev.
- *Markdown 3: Co je nového a co nás čeká?* od autora tohoto úvodníku pojednává o nejnovějších trendech v jazyce Markdown. Článek zpracovává téma, o kterém jsem přednášel na TUGu 2023.
- *Plán pro univerzální slabičnou segmentaci* Ondřeje Sojky, Petra Sojky a Jakuba Máci ukazuje možnosti univerzálního dělení slov nejen v TEXu. Článek zpracovává téma, o kterém trojice samozvaných *sylabických univerzalistů* přednášela na TUGu 2023.
- *Co by každý (LA)TEXový nováček měl znát* od Barbary Beeton v českém překladu Honzy Šustka poskytuje užitečné rady pro začátečníky.
- *Sazba textu české lidové písně „Když jsem já sloužil“ pomocí modulu l3seq jazyka expl3* od autora tohoto úvodníku ukazuje využití vysokoúrovňového programovacího jazyka expl3 na příkladu sazby textu české lidové písně.

Závěrem s radostí oznamuji, že konference TUG 2024 se uskuteční v Praze, což je historická událost, neboť od EuroTEXu ’92 se podobná akce v České republice nekonala. Konferenci pořádá TUG a lokální organizaci zajišťuje Tom Hejda za společnost Overleaf z pozice hlavního sponzora. Aktuální informace o konferenci najdete na webových stránkách TUGu [2] a v poštovním seznamu `cstug-members@cstug.cz`, jehož součástí jsou všichni členové CSTUGu. Do poštovního seznamu směrujte také nabídky pomoci s organizací.

DOI: 10.5300/2023-3-4/61

Odkazy

1. STARÝ NOVOTNÝ, Vít. *Valná hromada CSTUG 2023* [online]. 2023-11-12. [cit. 2023-11-27]. Dostupné z: `https://www.cstug.cz/informace/zpravy/2023-11-12-valna-hromada-2023/`.
2. *TeX Users Group* [online]. [cit. 2023-11-27]. Dostupné z: `https://tug.org/`.

Summary: Editorial

The editorial presents an overview of the articles from this issue and announces TUG 2024, which will be held in Prague.

Vít Starý Novotný, witiko@mail.muni.cz

CSTUG na konferenci TUG 2023

Vít Starý Novotný

Od pátku 14. července do neděle 16. července se v německém Bonnu konala konference TUG 2023. Konference se s podporou sdružení CSTUG zúčastnili tři jeho členové: předseda Petr Sojka, technický redaktor Zpravodaje Vítek Starý Novotný a šéfredaktor Zpravodaje Honza Šustek, vizte Obrázek 1.

V pátek ráno představil Honza Šustek svůj makrobalík pro literární programování pomocí maker OPmac v přednášce nazvané *On generating documented source code by blocks in TEX* [2], vizte Obrázek 4. Během přednášky položil Honza obecenstvu několik otázek. Tři šťastní řešitelé si po přednášce mohli připravit bylinný Čaj pro Ostravu. Článek z Honzovy přednášky začíná na straně 66.

Téhož dne večer představili Petr Sojka, Ondřej Sojka a Jakub Máca (vizte Obrázek 3) novinky z projektu univerzálních vzorů dělení slov v přednášce nazvané *Universal syllabic pattern generation* [3; 4]. V první části představil Ondřej Sojka technické a jazykové aspekty automatického dělení slov. V druhé části popsal Petr Sojka výsledky přípravy univerzálních vzorů pro devět jazyků a technická omezení programu Patgen, která zamezují generování vzorů pro více jazyků. V závěrečné části představil Jakub Máca datovou strukturu Judy, které se věnoval ve své bakalářské práci [5] a kterou lze použít místo trií pro urychlení generování vzorů. Článek z přednášky Petra, Ondřeje a Jakuba začíná na straně 125.

V sobotu ráno představil Vítek Starý Novotný třetí verzi makrobalíku pro přípravu dokumentů ve značkovacím jazyce Markdown v přednášce nazvané *Markdown 3: What's new, what's next?* [6; 7]. V první části přednášky Vítek představil nové funkce, které do balíku přibyly během posledních dvou let. V druhé části přednášky se zaměřil na nedostatky, které je třeba vyřešit před vydáním stabilní verze 3.0.0, a na nové funkce, které jsou plánovány po vydání verze 3.0.0. Článek z Vítkovy přednášky začíná na straně 111.

V neděli dopoledne proběhlo na závěr konference slosování výherců třetího vydání dvousvazkového díla *The LATEX Companion* [8]. Šťastným výhercem se stal i Honza Šustek, který si nechal knihy podepsat od přítomných členů LATEXového týmu, načež pod tíhou dystopických 1984 stran LATEXu odspěchal na vlak.

Odkazy

1. STARÝ NOVOTNÝ, Vít. *CSTUG na konferenci TUG 2023* [online]. 2023-07-16. [cit. 2023-11-12]. Dostupné z: `https://www.cstug.cz/informace/zpravy/2023-07-16-tug-2023/`.

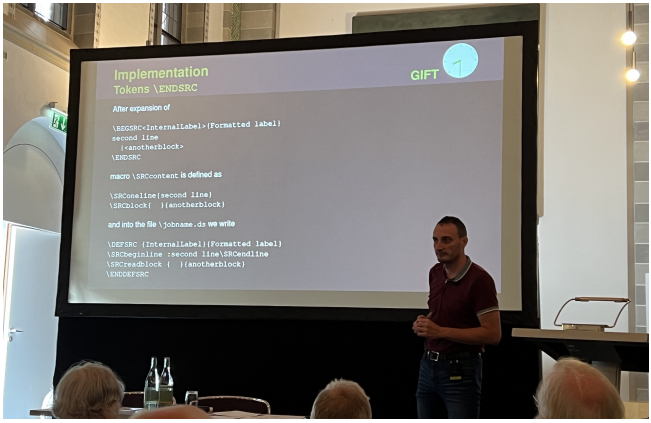
Tento článek vznikl rozšířením zprávy na webu sdružení CSTUG [1].

DOI: 10.5300/2023-3-4/63

Obrázek 1: Petr Sojka (vlevo), Vítek Starý Novotný (uprostřed) a Honza Šustek na recepci konferenčního hotelu TUGu 2023

Obrázek 2: TEXoví maskoti

Obrázek 3: Ondřej Sojka (vlevo), Petr Sojka (uprostřed) a Jakub Máca



Obrázek 4: Honza Šustek přednáší o literárním programování.

2. ŠUSTEK, Jan. *On generating documented source code by blocks in TEX* [online]. 2023-07-14. [cit. 2023-11-12]. Dostupné z: `https://tug.org/tug2023/files/fr-03-sustek-literate/sustek-literate-slides.pdf`.
3. SOJKA, Ondřej; SOJKA, Petr; MÁCA, Jakub. *A roadmap for universal syllabic segmentation* [online]. 2023-07-14. [cit. 2023-11-12]. Dostupné z: `https://tug.org/tug2023/files/fr-14-maca-syllabic/maca-syllabic-slides.pdf`.
4. SOJKA, Ondřej; SOJKA, Petr; MÁCA, Jakub. A roadmap for universal syllabic segmentation. *TUGboat*. 2023, **44**(2), 289–296.
5. MÁCA, Jakub. *Judy* [online]. Brno, 2023 [cit. 2023-11-12]. Dostupné z: `https://is.muni.cz/th/kru3j/`. Bakalářská práce. Fakulta informatiky Masarykovy univerzity. Vedoucí práce Petr SOJKA.
6. STARÝ NOVOTNÝ, Vít. *Markdown 3: What's new, what's next?* [online]. 2023-07-15. [cit. 2023-09-15]. Dostupné z: `https://tug.org/tug2023/files/sa-03-novotny-markdown3/novotny-markdown3-slides.pdf`.
7. STARÝ NOVOTNÝ, Vít. *TUG 2023 – Markdown 3: Co je nového a co se chystá?* [online]. 2023-07-15. [cit. 2023-09-15]. Dostupné z: `https://youtu.be/U8XjTOhJkg0`.
8. MITTELBACH, Frank; FISCHER, Ulrike. *The LATEX Companion*. 3. vyd. Addison-Wesley Professional, 2023.

Summary: CSTUG at the TUG 2023 Conference

This article reports on the participation of CSTUG members at TUG 2023 in Bonn.

Vít Starý Novotný, witiko@mail.muni.cz

Generování dokumentovaného zdrojového souboru po blocích v TEXu

Jan Šustek

Článek popisuje problematiku psaní programů a dokumentace k nim. Ukazuje autorův balíček gensrc nad OPmacem, který umožňuje psát program i dokumentaci k němu v jediném TEXovém souboru. Jsou ukázány další možnosti a aplikace tohoto balíčku.

Klíčová slova: dokumentace, literární programování, OPmac, gensrc

1 Úvod

Představme si, že píšeme počítačový program, který je delší než několik řádků. Potom nutně musíme k jednotlivým částem programu psát také komentáře. V opačném případě bude program špatně čitelný a špatně pochopitelný, a to jak pro druhou osobu, tak i pro nás samotné, pokud se na program podíváme po nějakém čase.

Zejména u složitějších programů používajících hlubší myšlenky pak musí být dokumentace k programu důkladnější. V následující ukázce je část zdrojového kódu programu[1] a příslušná část souboru s dokumentací.

```
Procedure PridejDoMenu;
  Begin
  inc(Podnabidek[M]);
  if Poz=Nakonec then
    begin
    Nabidka[M,Podnabidek[M]]:=Text;
    if M<>Hlavni then Proc[M,Podnabidek[M]]:=Procedura;
    end
  else
    begin
    ...
    end;
  End;
```

[1] Z důvodu jednoduššího vyjadřování budeme nadále namísto pojmu „zdrojový kód programu“ psát pouze „program“. Přitom vůbec není nutné, aby výsledkem byl nějaký program – může se jednat o libovolný text, který chceme vygenerovat.

 DOI: 10.5300/2023-3-4/66

4.3.18PridejDoMenu

Deklarace: `Procedure PridejDoMenu(M,Poz:Byte;Text:String;Procedura:TProc)`
Procedura přidá do `M`-tého menu na `Poz`-tou pozici položku s názvem `Text`, po jejímž spuštění se spustí procedura `Procedura`. Hlavnímu menu přísluší `M` rovno nule. Pro vkládání na konec daného menu může být argument `Poz` roven nule.
Přímo z hlavního menu nelze spouštět procedury, proto je přiřazení proměnné `Procedura` do proměnné `Proc` prováděno pouze pro podmenu. Výjimku tvoří ta podmenu, která obsahují právě jednu položku. Ta je vybrána a příslušná procedura spuštěna již při zvolení na hlavním menu.

Na problém narazíme v případě, že se program vyvíjí. Potom musíme udržovat dokumentaci programu kompatibilní s programem samotným. Každá změna programu se musí projevit v dokumentaci. To znamená, že změnu musíme evidovat na dvou místech – v souboru s programem a v souboru s dokumentací. Přitom je vhodné a výhodné, když se každá změna eviduje pouze na jednom místě.

V tomto článku jsou nejprve ukázána některá existující řešení uvedeného problému, konkrétně v sekci 2 je to řešení od Donalda Knutha, které použil při psaní samotného TEXu, a v sekci 3 je to řešení pro LATEX. V sekcích 4 až 6 je popsán autorův balíček gensrc pracující nad OPmacem. Implementaci balíčku popsanou v sekci 6 ocení spíše programátoři maker než běžní uživatelé TEXu. Sekce 7 pak ukazuje reálné situace, ve kterých se balíček využil.

Cílem balíčku gensrc nebylo kopírovat nebo napodobovat některá existující řešení. Nebylo ani cílem, aby byl balíček dokonalý a uměl vše. Balíček naopak dodržuje všechny tři hlavní zásady OPmacu [1]:

- V jednoduchosti je síla.
- Makra nejsou univerzální, ale jsou čitelná a srozumitelná.
- Uživatel si makra může snadno předefinovat k obrazu svému.

Původním cílem balíčku bylo plnit pracovní úkoly. A při plnění těchto úkolů se TEX a jeho makra ukázaly jako účinní a užiteční pomocníci.

Pro názornost článek obsahuje větší množství ukázek programů a dokumentace k nim. Aby ukázky byly dostatečně graficky odlišeny od textu článku, jsou vloženy do rámečků.[2] Pokud není uvedeno jinak, jsou v ukázkách použity části autorových vlastních programů.

Článek navazuje na autorovy přednášky na konferenci OSS Conf 2019 v Žilině a na konferenci TUG 2023 v Bonnu.

[2] O sazbě těchto rámečků se čtenář dočte v následujícím článku tohoto Zpravodaje, který začíná na straně 102.

2 DEK a WEB

Začátkem 80. let přišel Donald Knuth [2] s myšlenkou, že by se měly programy psát jiným způsobem, než bylo do té doby obvyklé. Namísto psaní programu jako posloupnosti instrukcí pro počítač navrhl Knuth psát program tak, jako bychom druhému člověkovi vysvětlovali, co chceme, aby počítač prováděl.

Pro psaní programů Knuth vyvinul jazyk WEB. Vstupní soubor jazyka WEB v sobě obsahuje dokumentaci proloženou částmi programu. Tento soubor se

- zpracuje programem weave, čímž se vygeneruje TEXový soubor s dokumentací, který se pak může TEXem vysázet. Dále se vstupní soubor
- zpracuje programem tangle, čímž se vygeneruje program v jazyce Pascal, který se poté může zkompilovat a spustit.

Původní varianta jazyka WEB byla vytvořena pro jazyky TEX a Pascal. Později byly vytvořeny varianty i pro jiné kombinace dokumentačního a programovacího jazyka.

Pro ukázku použití Knuthova literárního programování jsem použil dokumentovaný zdrojový kód TEXu[3] [3], konkrétně kód jeho nejdůležitější procedury main_control. Tato procedura řídí celý běh TEXu a v *TEXbooku naruby* [4] se označuje jako „hlavní procesor“. Procedura se spouští ihned po spuštění TEXu po úvodních formalitách (inicializace interních proměnných, inicializace tabulky primitivů, kontrola konzistence atd.). Po dokončení této procedury už následují pouze formality závěrečné (kontrola neuzavřených skupin při ukončení, závěrečný zápis do souboru \jobname.log, uzavření souborů atd.) a samotné ukončení TEXu.

1030. We shall concentrate first on the inner loop of *main_control*, deferring consideration of the other cases until later.

```
  define big_switch = 60   { go here to branch on the next token of input }
  define main_loop = 70   { go here to typeset a string of consecutive characters }
  define main_loop_wrapup = 80   { go here to finish a character or ligature }
  define main_loop_move = 90   { go here to advance the ligature cursor }
  define main_loop_move_lig = 95   { same, when advancing past a generated ligature }
  define main_loop_lookahead = 100   { go here to bring in another character, if any }
  define main_lig_loop = 110   { go here to check for ligatures or kerning }
  define append_normal_space = 120   { go here to append a normal space between words }
⟨ Declare action procedures for use by main_control 1043 ⟩
⟨ Declare the procedure called handle_right_brace 1068 ⟩
procedure main_control;   { governs TEX's activities }
  label big_switch, reswitch, main_loop, main_loop_wrapup, main_loop_move, main_loop_move + 1,
        main_loop_move + 2, main_loop_move_lig, main_loop_lookahead, main_loop_lookahead + 1,
        main_lig_loop, main_lig_loop + 1, main_lig_loop + 2, append_normal_space, exit;
  var t: integer;   { general-purpose temporary variable }
  begin if every_job ≠ null then begin_token_list(every_job, every_job_text);
big_switch: get_x_token;
reswitch: ⟨ Give diagnostic information, if requested 1031 ⟩;
```

[3]Autor tohoto článku předpokládá, že by se uživateli TEXu mohlo zdát zajímavé vidět vnitřnosti svého oblíbeného programu. :-)

```
case abs(mode) + cur_cmd of
hmode + letter, hmode + other_char, hmode + char_given: goto main_loop;
hmode + char_num: begin scan_char_num; cur_chr ← cur_val; goto main_loop; end;
hmode + no_boundary: begin get_x_token;
  if (cur_cmd = letter) ∨ (cur_cmd = other_char) ∨ (cur_cmd = char_given) ∨ (cur_cmd = char_num)
      then cancel_boundary ← true;
  goto reswitch;
  end;
hmode + spacer: if space_factor = 1000 then goto append_normal_space
  else app_space;
hmode + ex_space, mmode + ex_space: goto append_normal_space;
⟨ Cases of main_control that are not part of the inner loop 1045 ⟩
end; { of the big case statement }
goto big_switch;
main_loop: ⟨ Append character cur_chr and the following characters (if any) to the current hlist in the
    current font; goto reswitch when a non-character has been fetched 1034 ⟩;
append_normal_space: ⟨ Append a normal inter-word space to the current list, then goto big_switch 1041 ⟩;
exit: end;
```

Pro zpřehlednění logické struktury procedury jsou použity ⟨bloky⟩, které jsou definovány v dalších sekcích dokumentace. Procedura `main_control` je definována v sekci 1030 a v jejím těle se používají některé další procedury, které musejí být (dle syntaxe jazyka Pascal) definovány dříve než procedura `main_control`. Proto bylo nutné blok ⟨Declare ... 1043⟩ vložit před definici procedury `main_control`. Na druhou stranu: kvůli logickým návaznostem v dokumentaci mohou být tyto další procedury definovány až v sekci 1043.

Uvedený blok není definován celý najednou, ale postupně v 57 částech, jedna procedura po druhé. Jedna z těchto částí je v sekci 1075 a jak je vidět, i uvnitř bloku se mohou vyskytovat další vnořené bloky.

1075. The *box_end* procedure does the right thing with *cur_box*, if *box_context* represents the context as explained above.

```
⟨ Declare action procedures for use by main_control 1043 ⟩ +≡
procedure box_end(box_context : integer);
  var p: pointer; { ord_noad for new box in math mode }
  begin if box_context < box_flag then
    ⟨ Append box cur_box to the current list, shifted by box_context 1076 ⟩
  else if box_context < ship_out_flag then ⟨ Store cur_box in a box register 1077 ⟩
    else if cur_box ≠ null then
        if box_context > ship_out_flag then ⟨ Append a new leader node that uses cur_box 1078 ⟩
        else ship_out(cur_box);
  end;
```

Tyto vnořené bloky jsou pak definovány v dalších sekcích dokumentace.

Použitý mechanismus má navíc výhodu, že bloky mohou být použity opakovaně na více místech. Například blok ⟨Get ... 404⟩ je použit na 8 místech programu.

1077. ⟨ Store *cur_box* in a box register 1077 ⟩ ≡

```
if box_context < box_flag + 256 then eq_define(box_base − box_flag + box_context, box_ref, cur_box)
else geq_define(box_base − box_flag − 256 + box_context, box_ref, cur_box)
```

This code is used in section 1075.

```
1078.  ⟨ Append a new leader node that uses cur_box 1078 ⟩ ≡
  begin ⟨ Get the next non-blank non-relax non-call token 404 ⟩;
  if ((cur_cmd = hskip) ∧ (abs(mode) ≠ vmode)) ∨ ((cur_cmd = vskip) ∧ (abs(mode) = vmode)) then
    begin append_glue; subtype(tail) ← box_context − (leader_flag − a_leaders);
    leader_ptr(tail) ← cur_box;
    end
  else begin print_err("Leaders␣not␣followed␣by␣proper␣glue");
    help3("You␣should␣say␣`\leaders␣<box␣or␣rule><hskip␣or␣vskip>´.")
    ("I␣found␣the␣<box␣or␣rule>,␣but␣there´s␣no␣suitable")
    ("<hskip␣or␣vskip>,␣so␣I´m␣ignoring␣these␣leaders."); back_error; flush_node_list(cur_box);
    end;
  end
This code is used in section 1075.
```

V ukázce sekce 1030 můžeme vidět také definice maker jazyka WEB. Knuth do jazyka WEB zavedl mechanismus maker kvůli zpřehlednění a zjednodušení dokumentace, ale také kvůli odstínění některých nedokonalostí tehdejších kompilátorů jazyka Pascal. Makra jazyka WEB mohou mít nula, nebo jeden parametr.

Soubor, z něhož byla programem weave vygenerována výše uvedená dokumentace, má název tex.web. V následující ukázce je zdrojový kód sekce 1075. Jsou zvýrazněny příkazy pro práci s bloky a sekcemi.

```
@ The |box_end| procedure does the right thing with |cur_box|, if
|box_context| represents the context as explained above.

@<Declare act...@>=
procedure box_end(@!box_context:integer);
var p:pointer; {|ord_noad| for new box in math mode}
begin if box_context<box_flag then @<Append box |cur_box| to the current list,
    shifted by |box_context|@>
else if box_context<ship_out_flag then @<Store \(c)|cur_box| in a box register@>
else if cur_box<>null then
  if box_context>ship_out_flag then @<Append a new leader node that
      uses |cur_box|@>
  else ship_out(cur_box);
end;
```

Zpracováním souboru tex.web programem tangle se vygeneruje program tex.pas. V ukázce je zvýrazněna ta část programu, která odpovídá sekci 1075.

```
{:1070}{1075:}procedure boxend(boxcontext:integer);var p:halfword;
begin if boxcontext<1073741824 then{1076:}begin if curbox<>0 then begin
mem[curbox+4].int:=boxcontext;
if abs(curlist.modefield)=1 then begin appendtovlist(curbox);
if adjusttail<>0 then begin if 29995<>adjusttail then begin mem[curlist.
tailfield].hh.rh:=mem[29995].hh.rh;curlist.tailfield:=adjusttail;end;
adjusttail:=0;end;if curlist.modefield>0 then buildpage;
end else begin if abs(curlist.modefield)=102 then curlist.auxfield.hh.lh
:=1000 else begin p:=newnoad;mem[p+1].hh.rh:=2;mem[p+1].hh.lh:=curbox;
```

```
curbox:=p;end;mem[curlist.tailfield].hh.rh:=curbox;
curlist.tailfield:=curbox;end;end;
end{:1076}else if boxcontext<1073742336 then{1077:}if boxcontext<
1073742080 then eqdefine(-1073738146+boxcontext,119,curbox)else
geqdefine(-1073738402+boxcontext,119,curbox){:1077}else if curbox<>0
then if boxcontext>1073742336 then{1078:}begin{404:}repeat getxtoken;
until(curcmd<>10)and(curcmd<>0){:404};
if((curcmd=26)and(abs(curlist.modefield)<>1))or((curcmd=27)and(abs(
curlist.modefield)=1))then begin appendglue;
mem[curlist.tailfield].hh.b1:=boxcontext-(1073742237);
mem[curlist.tailfield+1].hh.rh:=curbox;
end else begin begin if interaction=3 then;printnl(262);print(1065);end;
begin helpptr:=3;helpline[2]:=1066;helpline[1]:=1067;helpline[0]:=1068;
end;backerror;flushnodelist(curbox);end;end{:1078}else shipout(curbox);
end;{:1075}{1079:}procedure beginbox(boxcontext:integer);label 10,30;
```

3 Program docstrip

Pro LaTeX je k dispozici program docstrip [5], který umožňuje do jednoho zdrojového souboru psát program i dokumentaci k němu. Podobně jako v případě jazyka WEB i zde můžeme jedním zpracováním zdrojového souboru vygenerovat program a druhým zpracováním vysázet dokumentaci. Balíček má i další dovednosti.

3.1 Základní použití

Program a dokumentace se zapisují do textového souboru, který mívá příponu dtx. V tomto souboru jsou řádky programu vloženy do prostředí macrocode, které musí začínat a končit na řádcích, které na začátku mají znak procenta a přesně čtyři mezery. Řádky mimo toto prostředí tvoří dokumentaci a musí začínat znakem procenta. Pokud si odmyslíme tato procenta, píše se dokumentace stejně jako libovolný jiný LaTeXový dokument.[4]

```
% Makro |\radeknabody| zpracuje řádek a jednotlivé položky uloží
% do příslušných maker a čítačů.
%    \begin{macrocode}
\def\radeknabody#1;#2;#3;#4;#5;{%
  \def\no{#1}\def\bodyA{#2}\def\bodyB{#3}\def\bodyC{#4}\def\bodyD{#5}%
  \nacislo#2 \bodycA \nacislo#3 \bodycB
  \nacislo#4 \bodycC \nacislo#5 \bodycD}
%    \end{macrocode}
```

[4] V ukázkách je použita část programu používaného při Mezinárodní matematické soutěži Vojtěcha Jarníka. Soutěž je pořádána Ostravskou univerzitou a bližší informace k ní lze nalézt na stránkách https://vjimc.osu.cz/.

```
% Pokud je některý soutěžící diskvalifikován, do tabulky s~počtem bodů
% se mu zapíše z~prvního příkladu 666~bodů. Jeho součet bodů se nastaví
% na nulu a v~souboru \soub{results.srt?} bude uveden až na konci.
%    \begin{macrocode}
\def\nvzpracujradek{%
  \ea\radeknabody\radek
...
%    \end{macrocode}
```

Pro vygenerování souboru pdf s dokumentací vytvoříme nový LaTeXový soubor. V něm mimo jiných použijeme balíček doc. Soubor dtx pak načteme makrem `\DocInput`. LaTeXový soubor zpracujeme běžným způsobem LaTeXem.

`\nacislo` Makro `\nacislo` nastaví čítač `#2` na hodnotu `#1` nebo na hodnotu `#1` − 100, pokud `#1` ∈ [100; 200). Pokud `#1` = -, nastaví `#2` na nulu.

```
\def\nacislo#1 #2{\ifx-#1#20\else
    #2#1 \ifnum#2>99 \ifnum#2<200 \advance#2-100\fi\fi\fi}
```

`\radeknabody` Makro `\radeknabody` zpracuje řádek a jednotlivé položky uloží do příslušných maker a čítačů.

```
\def\radeknabody#1;#2;#3;#4;#5;{%
  \def\no{#1}\def\bodyA{#2}\def\bodyB{#3}\def\bodyC{#4}\def\bodyD{#5}%
  \nacislo#2 \bodycA \nacislo#3 \bodycB
  \nacislo#4 \bodycC \nacislo#5 \bodycD}
```

`\nvzpracujradek` Pokud je některý soutěžící diskvalifikován, do tabulky s počtem bodů se mu zapíše z prvního příkladu 666 bodů. Jeho součet bodů se nastaví na nulu a v souboru results.srt? bude uveden až na konci.

```
\def\nvzpracujradek{%
  \ea\radeknabody\radek
  \sumac\bodycA \advance\sumac\bodycB
  \advance\sumac\bodycC \advance\sumac\bodycD
  \ifnum\bodycA=\bodyDQ
    \counta99 \sumac0
```

Pro vygenerování programu vytvoříme další LaTeXový soubor. V něm mimo jiné použijeme balíček docstrip. Dále použijeme makro `\generateFile`, jímž načteme vstupní soubor dtx a vygenerujeme program s daným názvem. LaTeXový soubor stačí jednou zpracovat LaTeXem.

```
\def\nacislo#1 #2{\ifx-#1#20\else
    #2#1 \ifnum#2>99 \ifnum#2<200 \advance#2-100\fi\fi\fi}
\def\radeknabody#1;#2;#3;#4;#5;{%
  \def\no{#1}\def\bodyA{#2}\def\bodyB{#3}\def\bodyC{#4}\def\bodyD{#5}%
  \nacislo#2 \bodycA \nacislo#3 \bodycB
  \nacislo#4 \bodycC \nacislo#5 \bodycD}
```

```
\def\nvzpracujradek{%
  \ea\radeknabody\radek
  \sumac\bodycA \advance\sumac\bodycB
  \advance\sumac\bodycC \advance\sumac\bodycD
  \ifnum\bodycA=\bodyDQ
    \counta99 \sumac0
```

Detaily jednotlivých LaTeXových souborů a maker se čtenář dozví v dokumentaci [6; 7].

3.2 Další funkce

V souboru dtx mimo prostředí `macrocode` lze využít další funkce balíčku docstrip. Prostředí `macro` označuje, že část programu a dokumentace se týká definice konkrétního příkazu.[5] Makro `\changes` označuje, že na daném místě došlo v dané verzi programu k určité změně.

```
% \begin{macro}{\nvzpracujradek}
% Pokud je některý soutěžící diskvalifikován, do tabulky s~počtem bodů
% se mu zapíše z~prvního příkladu 666~bodů. Jeho součet bodů se nastaví
% na nulu a v~souboru \soub{results.srt?} bude uveden až na konci.
% \changes{100908}{100908}{Pročištění}
% \changes{140314}{140314}{Logika diskvalifikací}
% \changes{140314}{140314}{Přidáno top competitors}
%    \begin{macrocode}
\def\nvzpracujradek{%
  \ea\radeknabody\radek
  \sumac\bodycA \advance\sumac\bodycB
  \advance\sumac\bodycC \advance\sumac\bodycD
  \ifnum\bodycA=\bodyDQ
...
%    \end{macrocode}
% \end{macro}
```

Při použití výše uvedených maker lze v dokumentaci makrem `\PrintChanges` vygenerovat seznam všech změn v jednotlivých verzích programu. Jestliže ke změně došlo uvnitř prostředí `macro`, přiřadí se změna přímo ke konkrétnímu příkazu. Pro zaznamenávání změn je třeba v preambuli souboru použít makro `\RecordChanges`.

`\spoctiporadi`: Přidáno spoctiporadi 79
General: Dokumentace 72, 73
Název souboru do chybové hlášky . . 4
`\jedenradektop`: Přidáno top competitors 66
`\jedentop`: Přidáno top competitors . 66

[5] Balíček docstrip je uzpůsoben ke generování LaTeXových balíčků a zřejmě proto jsou pro účely tohoto balíčku části programu označeny jako „macro“. Zde, opět z důvodu srozumitelnosti, budeme používat pojem „příkaz“.

Přidáno mezuspesnych 86
Rozdělení diplomů na části . . . 78, 82
Umožnění Plainu i LaTeXu . . . 72, 73
140314
\nvzpracujradek: Logika diskvalifikací 60
Přidáno top competitors 60
\spoctiporadi: Logika diskvalifikací . 79
\zpracujkategorii: Logika zpracování kategorií 68
General: Dokumentace 85
Logika diskvalifikací 61
Logika zpracování kategorií 71
Přidáno top competitors 65
140315
\celkovamrizka: Označení makra . . . 53
\meztop: Přidáno meztop 2
\nactisouborIRD: Dokumentace 5
\nastavcasnated: Přidáno casX 25
\round: Přidáno top competitors 66
\zapisradektop: Přidáno top competitors 67
\zapistop: Přidáno top competitors . 67
\zpracujtop: Přidáno top competitors 66
General: Dokumentace 84
Přidán pomocný registr 1
Přidáno meztop 87
140317
\ifzobrazporadi: Přidáno potvrzení pořadí 33
\potvrzeni: Přidání potvrzení pořadí 32

Podobně lze v dokumentaci makrem `\PrintIndex` vytvořit rejstřík všech příkazů[6] použitých v programu. V rejstříku je u konkrétního příkazu možné zobrazit číslo stránky nebo řádku,[7] na nichž je příkaz použit. Podtržení čísla znamená, že na daném místě je příkaz definován. Pro umožnění tvorby rejstříku je třeba v preambuli souboru použít makro `\PageIndex` nebo `\CodelineIndex`.

\pgI 1404, 3045–3048, 3086–3089
\phantom 2203, 2210, 3017, 3019, 3025, 3028, 3034, 3037, 3046–3048, 3063, 3066, 3069, 3078, 3087, 3089
\pocetD 192, 200, 1343
\pocetdiplomu . . . 21, 2981
\pocetI . . 192, 198, 204, 883, 884, 904, 946, 1341, 1384, 1546, 1548, 1709, 1751, 1954
\pocetII 192, 199, 893, 894, 909, 962,
\pozvanka 1344, 1347, 1640
\predchozisuma . . 2106, 2149, 2150, 2865, 2868, 2890
\predseda . . 36, 2880, 2990
\predsedajmeno 36, 2854, 2880
\prefix 793, 886, 888, 897, 900, 907, 911, 930, 951, 953, 968, 972, 977, 981, 993, 996, 1006, 1066, 1153, 1154, 1174, 1175, 1545
\prefixMr 1066, 1174, 1191
\prefixMs 1174, 1191
\prevedvysledky 2003, 2496

R

\radek 64, 66, 68, 107, 2014, 2022, 2029, 2049, 2063, 2091
\radeknabody . 2058, 2063
\radekvysledku 2119, 2122
\radekvysledkuA 2124, 2126
\raise . . . 311, 318, 394, 400, 1017, 1021, 1076, 1169, 1362, 1370, 1397, 1407, 2823, 2843, 2848, 2879
\rboxA 743, 746
\re 286
\read 107, 871, 2014, 2022, 2049, 2091

[6] Jako příkazy se v programu vyhledají všechny názvy začínající znakem \, případně jiným nastaveným znakem.

[7] Čísla řádků v rejstříku odpovídají číslům řádků v dokumentaci. Tato čísla se však mohou lišit od čísel řádků ve vygenerovaném programu.

4 Jednoduché řešení v OPmacu

Autor článku namísto LaTeXu raději pracuje v Plainu s balíkem OPmac [1]. Cílem této sekce je vytvořit nad OPmacem soubor gensrc.tex[8] a v něm nadefinovat makra

- `\SRCFILENAME` udávající název vygenerovaného programu,
- `\BEGSRC` a `\ENDSRC` ohraničující řádky programu.

S balíčkem gensrc.tex se pak v jediném průchodu TeXem vytvoří jak program, tak dokumentace. Jak bude tento program a dokumentace vypadat, to si zajisté čtenář snadno domyslí.[9]

```
\input opmac
\input gensrc
\SRCFILENAME style.tex
\activettchar"
Format of the page is~A4 with~2cm~margins.
The basic font size is set to 12\,pt.
\BEGSRC
\margins/1 a4 (2,2,2,2)cm
\typosize[12/14]
\ENDSRC
Macro "\safedef" defines a control sequence which is not yet defined.
If it is already defined then its new definition is ignored.
\BEGSRC
\def\safedef#1{\ifdefined#1\begingroup\afterassignment\endgroup\fi\def#1}
\ENDSRC
Sections are defined in the same way as in~\OPmac.
\BEGSRC
\safedef\section{\sec}
\ENDSRC
\bye
```

4.1 Základní použití

Součástí dokumentace jsou i části programu. Proto není divu, že základem souboru gensrc.tex bude makro pro sazbu verbatim textu. K tomu je v OPmacu následujícím způsobem definováno makro `\begtt`. Detailní popis jednotlivých v něm použitých triků čtenář najde v *TeXbooku naruby* [4, sekce 1.3].

```
\def\begtt{\par\ttskip\bgroup \wipeepar
  \setverb\adef{ }{ }
```

[8] Soubor gensrc.tex tvoří řádky 17 až 45 na následujících stranách, přičemž je třeba vymazat šedý text. Navíc je možné při použití řádků 39 až 45 dále vymazat řádky 29 až 32.

[9] Zvídavému TeXistovi doporučuji si prozkoumat, jak funguje definice makra `\safedef` uvedená v ukázce.

```
  \ifx\savedttchar\undefined \else \catcode\savedttchar=12 \fi
  \parindent=\ttindent \vskip\parskip \parskip=0pt
  \tthook\relax
  \ifnum\ttline<0 \else
    \tenrm \thefontscale[700]\let\sevenrm=\thefont
    \everypar={\global\advance\ttline by1
      \llap{\sevenrm\the\ttline\kern.9em}}\fi
  \def\par##1{\endgraf\ifx##1\egroup\else
    \penalty\ttpenalty\leavevmode\fi ##1}
  \obeylines \startverb}
\def\setverb{\frenchspacing\def\do##1{\catcode`##1=12}%
  \dospecials \catcode`\*=12 }
{\catcode`\|=0 \catcode`\\=12
|gdef|startverb#1\endtt{|tt#1|egroup|par|ttskip|testparA}}
```

Jednoduchými úpravami výše uvedených maker lze vytvořit makro `\BEGSRC`. Konkrétně se jedná o následující kosmetické úpravy, přičemž pouze první dva body jsou opravdu potřebné.

- Řídicí sekvence `\begtt`, `\setverb`, `\tthook`, `\ttline`, `\startverb`, `\endtt` jsou přejmenovány.
- Registr `\everypar` se nastaví nezávisle na podmínce `\SRCline<0`. Toto nastavení je lokální uvnitř prostředí `\BEGSRC...\ENDSRC`.[10]
- Makro `\wipeepar` globálně promazává registr `\everypar`. Toto není potřeba, a proto zde makro `\wipeepar` není nutné.
- Je použito `\csname` pro případ, že by háček `\SRChook` nebyl definován.
- Namísto makra `\leavevmode` je použit primitiv `\quitvmode`, který v daném místě neexpanduje registr `\everypar`. Navíc příkaz `\expandafter` zajistí, že TEX na token `\fi` narazí ještě ve vertikálním módu.

Další úpravou je přidání jedné definice.

- Makro `\SRCFILENAME` nemá analogii v případě makra `\begtt`. Makro otevře pro zápis soubor `\SRCfile`. V případě, že je makro použito podruhé (tj. z jednoho zdrojového souboru začínáme generovat další program), je aktuální soubor nejprve uzavřen. Tento krok v případě prvního použití makra nenahlásí chybu.

A samozřejmě je nutná i jedna funkční úprava.

- Na začátku každého řádku programu je zavoláno makro `\SRCgetline`. Makro načte argument až po znak konce řádku[11] (tj. načte jeden řádek), tento argument vysází a zapíše jej do souboru.

Více úprav není třeba. Všechny výše uvedené úpravy jsou barevně vyznačeny.

[10] Pro přehlednost budeme dále používat text „prostředí `\BEGSRC`“.

[11] V makru `\BEGSRC` je použito makro `\obeylines`, které nastaví znak konce řádku jako aktivní. Proto je nutný onen trik s vlnovkou na řádcích 33 a 34.

```
\def\BEGSRC{\par\ttskip\bgroup \wipeepar
  \setSRC\adef{ }{ }
  \ifx\savedttchar\undefined \else \catcode\savedttchar=12 \fi
  \parindent=\ttindent \vskip\parskip \parskip=0pt
  \csname SRChook\endcsname\relax
  \ifnum\ttline<0 \else
    \tenrm \thefontscale[700]\let\sevenrm=\thefont
    \everypar={\global\advance\SRCline by1
      \llap{\sevenrm\the\SRCline\kern.9em}\SRCgetline}\fi
  \def\par##1{\endgraf\ifx##1\egroup\else
    \penalty\ttpenalty\expandafter\quitvmode\fi ##1}
  \obeylines \startSRC}
\def\setSRC{\frenchspacing\def\do##1{\catcode`##1=12}%
  \dospecials \catcode`\*=12 }
{\catcode`\|=0 \catcode`\\=12
|gdef|startSRC#1\ENDSRC{|tt#1|egroup|par|ttskip|testparA}}
\begingroup\lccode`\~13
  \lowercase{\endgroup\def\SRCgetline#1~}%
    {#1\immediate\write\SRCfile{#1}\par}
\def\SRCFILENAME{\immediate\closeout\SRCfile
  \immediate\openout\SRCfile=}
\newcount\SRCline \newwrite\SRCfile
```

4.2 Snadná vylepšení

Změnou definice makra `\startSRC` umožníme uživateli použít háček `\endSRChook`, který se expanduje v místě `\ENDSRC`. Pomocí `\csname` opět umožníme bezchybnou expanzi v případě, že háček není definován.

```
{\catcode`\|=0 \catcode`\\=12
  |gdef|startSRC#1\ENDSRC{|tt#1|egroup|par|ttskip
    |csname endSRChook|endcsname|testparA}}
```

Některé textové editory odsazují řádky pomocí tabulátoru, jiné pomocí mezer. To například v jazyce Python může činit problém. Definicí aktivního tabulátoru expandujícího se na posloupnost několika mezer tento problém vyřešíme. Při definování musíme opatrně měnit kategorie znaků tabulátoru a mezery.

```
{\catcode9=12
  \gdef\setSRC{\def\do##1{\catcode`##1=12}\dospecials
    \catcode`\*=12 \adef{^^I}{\SRCtab}}}
{\catcode32=13 \gdef\SRCtab{    }}
```

5 Generování po blocích

V této sekci si naplánujeme vylepšení souboru gensrc.tex po vzoru sekce 2. Navíc součástí tohoto vylepšení bude práce s lokálně aditivním odsazením řádků, což velmi pomůže při generování programů například v jazyce Python. Oproti sekci 2 se omezíme na situaci, kdy na řádku s vkládaným blokem není kromě bloku a odsazení žádný další text.

V této sekci pouze popíšeme, jaká makra budeme používat a co tato makra budou dělat. Implementace těchto maker bude následovat v sekci 6. Doporučuji čtenáři, aby si po přečtení sekce 6 znovu přečetl podsekce 5.3 a 5.4.

Nový soubor gensrc.tex tvoří řádky 46 až 207 na stranách 83 až 92. Soubor je ke stažení online [8].

5.1 Syntaxe

Makro `\BEGSRC` je možné volat následujícími způsoby.[12]

1. `\BEGSRC<InterniNazev>{Zobrazený název}`
2. `\BEGSRC<InterniNazev>`
3. `\BEGSRC`

Přesná syntaxe je patrná z následující ukázky. Každý blok má svůj (jednoduchý) interní název a dále název, který se zobrazí v dokumentaci. V této ukázce je navíc použit háček `\SRChook`, který způsobí, že řádky programu budou v dokumentaci vysázeny červeně.

```
\input gensrc
\def\SRChook{\longlocalcolor\Red}
\SRCFILENAME program.txt
Here we define a block which will be inserted to another place.
\BEGSRC<InternalLabel>{Displayed label}
second line
  |<InnerBlock>
\ENDSRC
A block can be defined by parts.
\BEGSRC<InternalLabel>
fourth line
\ENDSRC
And here we insert the block.
\BEGSRC
first line
    |<InternalLabel>
\ENDSRC
```

[12] V dalším textu se často budeme na uvedené číslování odkazovat.

```
Finally we define the inner block.
\BEGSRC<InnerBlock>{Inner block}
third line
\ENDSRC
\bye
```

Pro správné vysázení dokumentace jsou nutné dva průchody TEXem, protože v prvním průchodu ještě nejsou známy zobrazené názvy bloků.

Here we define a block which will be inserted to another place.

⟨*Displayed label*⟩ ≡
```
1 second line
2   ⟨Inner block⟩
```

A block can be defined by parts.

⟨*Displayed label*⟩ +≡
```
3 fourth line
```

And here we insert the block.

```
4 first line
5     ⟨Displayed label⟩
```

Finally we define the inner block.

⟨*Inner block*⟩ ≡
```
6 third line
```

Z podobného důvodu jsou nutné dva průchody TEXem pro správné vygenerování programu. Tyto dva průchody však již máme za sebou, když jsme vysázeli dokumentaci. Výsledek je dle očekávání následující:

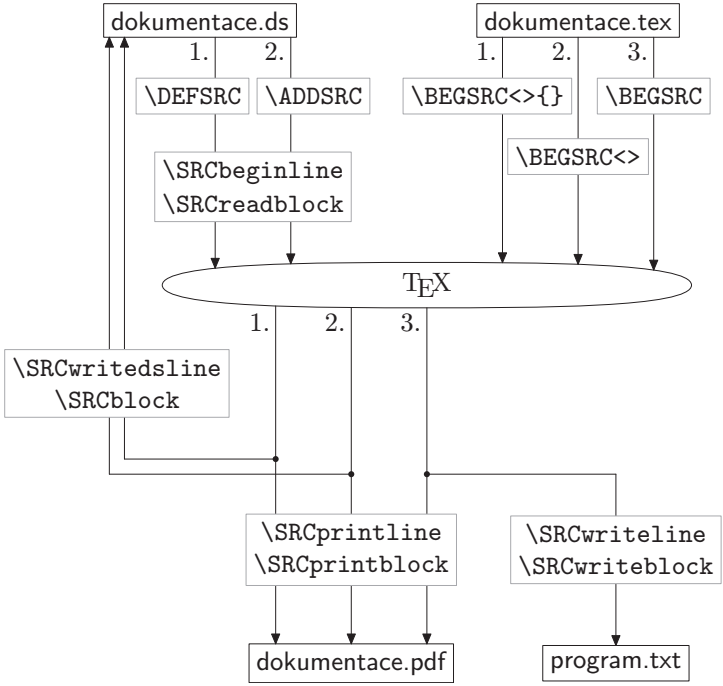
```
first line
    second line
      third line
    fourth line
```

5.2 Přenos informací

Dále v textu budeme předpokládat, že máme vstupní soubor dokumentace.tex, ze kterého chceme vytvořit dokumentaci v souboru dokumentace.pdf a vygenerovat program v souboru program.txt. Makra budou využívat soubor `\jobname`.ds,[13] v našem případě tedy soubor dokumentace.ds.[14] Graf na straně 80 ukazuje, jak se u jednotlivých způsobů volání makra `\BEGSRC` přenáší informace mezi soubory. V rámečcích u šipek jsou uvedena makra, která příslušnou informaci přenáší.

[13] Písmena „ds“ vznikla ze slov „dočasný soubor“.

[14] Pro zpřehlednění textu budeme dále jednotlivé soubory označovat pouze jako „tex“, „pdf“, „ds“, „txt“, přičemž budeme vynechávat i slovo „soubor“.



5.3 První průchod

V ukázce v podsekci 5.1 je čtyřikrát použito prostředí `\BEGSRC`. Vnitřek jednotlivých prostředí se ukládá do makra `\SRCcontent`. Toto makro má v místech `\ENDSRC` postupně následující obsah:

```
\SRConeline{second line}
\SRCblock{  }{InnerBlock}
```

```
\SRConeline{fourth line}
```

```
\SRConeline{first line}
\SRCblock{    }{InternalLabel}
```

```
\SRConeline{third line}
```

Makro `\doSRC` definuje příslušná makra a do `ds` postupně zapíše následující text:

```
\DEFSRC {InternalLabel}{Displayed label}
\SRCbeginline :second line\SRCendline
\SRCreadblock {  }{InnerBlock}
\ENDDEFSRC
\ADDSRC {InternalLabel}
\SRCbeginline :fourth line\SRCendline
\ENDDEFSRC
\DEFSRC {InnerBlock}{Inner block}
\SRCbeginline :third line\SRCendline
\ENDDEFSRC
```

Makra `\csname SRCtit:\SRCid\endcsname` obsahují zobrazené názvy bloků a ty v prvním průchodu ještě nejsou známy, proto tato makra expand procesor interpretuje jako `\relax` a v dokumentaci se namísto názvů bloků zobrazí prázdné závorky ⟨⟩:

Here we define a block which will be inserted to another place.

⟨*Displayed label*⟩ ≡
```
1 second line
2   ⟨⟩
```

A block can be defined by parts.

⟨⟩ +≡
```
3 fourth line
```

And here we insert the block.

```
4 first line
5     ⟨⟩
```

Finally we define the inner block.

⟨*Inner block*⟩ ≡
```
6 third line
```

Podobně makra `\csname SRCcon:\SRCid\endcsname` obsahují text jednotlivých bloků, který ale také v prvním průchodu není znám. Onen `\relax` způsobí, že se do `txt` příslušné řádky nezapíšou a že v `txt` bude pouze následující obsah:

```
first line
```

5.4 Druhý průchod

Na začátku druhého průchodu se načte `ds`. Makra z podsekce 6.7 způsobí, že se interně provedou následující definice:

```
\expandafter\def\csname SRCtit:InternalLabel\endcsname{%
  Displayed label}
```

```
\expandafter\def\csname SRCcon:InternalLabel\endcsname{%
  \SRConeline{second line}%
  \SRCblock{  }{InnerBlock}%
  \SRConeline{fourth line}}
\expandafter\def\csname SRCtit:InnerBlock\endcsname{%
  Inner block}
\expandafter\def\csname SRCcon:InnerBlock\endcsname{%
  \SRConeline{third line}}
```

Druhý průchod poté pokračuje stejně jako první průchod, až na dva rozdíly. Prvním rozdílem je, že po těchto definicích již TEX zná vše potřebné a dokumentaci vysází správně, jak je na straně 79. Druhým rozdílem je, že se správně vygeneruje i txt. Podívejme se, jak probíhá ono generování.

Pouze ve třetím případě je makro `\BEGSRC` voláno třetím způsobem, tj. způsobem, kdy se zapisuje do txt. Makro `\SRCcontent` se postupně expanduje takto:

```
\SRCcontent
```

```
\SRConeline{first line}
\SRCblock{    }{InternalLabel}
```

```
\SRCwriteline{first line}
\SRCwriteblock{    }{InternalLabel}
```

```
\csname SRCwritehook\endcsname
\immediate\write\SRCfile{\SRCodsazeni first line}
\begingroup
\addto\SRCodsazeni{    }
\csname SRCcon:InternalLabel\endcsname
\endgroup
```

```
\csname SRCwritehook\endcsname
\immediate\write\SRCfile{\SRCodsazeni first line}
\begingroup
\addto\SRCodsazeni{    }
\SRConeline{second line}
\SRCblock{  }{InnerBlock}
\SRConeline{fourth line}
\endgroup
```

Po několika dalších expanzích se dostaneme na následující příkazy:

```
\csname SRCwritehook\endcsname
\immediate\write\SRCfile{\SRCodsazeni first line}
\begingroup
\addto\SRCodsazeni{    }
\csname SRCwritehook\endcsname
\immediate\write\SRCfile{\SRCodsazeni second line}
\begingroup
\addto\SRCodsazeni{  }
\csname SRCwritehook\endcsname
\immediate\write\SRCfile{\SRCodsazeni third line}
\endgroup
\csname SRCwritehook\endcsname
\immediate\write\SRCfile{\SRCodsazeni fourth line}
\endgroup
```

Tyto příkazy způsobí, že se do txt zapíše správný text, který je uveden na straně 79.

6 Implementace

6.1 Začátek

Protože se v souboru gensrc.tex opakovaně používají makra OPmacu, je OPmac načten přímo v tomto souboru.[15]

```
\input opmac
```

Pro označení bloků kódu v pdf je použito makro `\SRCangle` a uvnitř něj je použit přepínač fontu `\sl`. Přitom je nutné font `\tensl` registrovat, aby se správně škálovala jeho velikost.

```
\regfont\tensl
\def\SRCangle#1{{$\langle$\sl#1\/$\rangle$}}
```

Stejně jako dříve se název generovaného souboru zadá makrem `\SRCFILENAME`. Interně bude tento soubor reprezentován řídicí sekvencí `\SRCfile`. Při opakovaném použití `\SRCFILENAME` se soubor nejdříve zavře a poté se začne začne generovat nový.

[15] Na konferenci TUG 2023 zazněl námět, aby makra fungovala také v LaTeXu. Jak je vidět v následujícím článku tohoto Zpravodaje na straně 106, je možné vytvořit balíček, který by fungoval jak v Plainu, tak v LaTeXu, pokud byl balíček původně naprogramován v Plainu. V tom případě by se v LaTeXové větvi samozřejmě OPmac nenačítal, musela by se ale nadefinovat všechna používaná OPmacová makra a na všech místech, kde LaTeX používá jiné mechanismy (například práce s fonty), by se musela použít makra, jejichž expanze by závisela na použitém formátu TeXu.

```
\newwrite\SRCfile
\def\SRCFILENAME{\immediate\closeout\SRCfile
  \immediate\openout\SRCfile=}
```

Prostředí `\BEGSRC` interně používá přepínače `\ifsaveSRC`, `\ifaddingSRC` a `\ifprintSRC`, aby se v různých situacích chovalo správně.

Pokud je nastaveno `\saveSRCtrue`, znamená to, že se definuje blok a ten se zapíše do `ds`. Pokud je nastaveno `\saveSRCfalse`, pak se zapisuje přímo do `txt`.

Pokud je nastaveno `\addingSRCtrue`, znamená to, že se přidávají řádky k již definovanému bloku. Pokud je nastaveno `\addingSRCfalse`, je příslušný blok vynulován a řádky jsou uloženy přidáním k takto vynulovanému bloku.

Pokud je nastaveno `\printSRCtrue`, bude se aktuální prostředí `\BEGSRC` sázet do `pdf`. V případě, že je nastaveno `\printSRCfalse`, bude se sázet do boxu `\nonprintbox`, který se nevytiskne. Implicitní hodnota je `\printSRCtrue`.

```
\newif\ifsaveSRC
\newif\ifaddingSRC
\newif\ifprintSRC \printSRCtrue
\newbox\nonprintbox
```

Makro `\SRCensurehmode\token` zajistí, že se ve vertikálním módu bude `\token` chovat tak, že bezpečně přejde do módu horizontálního a tam zůstane neexpandovaný.

```
\def\SRCensurehmode#1{\def#1{\quitvmode\noexpand#1}}
```

Uvnitř prostředí `\BEGSRC` má znak `|` kategorii 0. Pro jeho sazbu a export se použije řídicí sekvence `||`.

```
\def\|{|}
```

Může se stát, že prostředí `\BEGSRC` zavoláme uvnitř prostředí `\begitems`, v němž je hvězdička aktivním znakem. Proto ji musíme na všech potřebných místech deaktivovat. To uděláme nejjednodušeji tak, že ji zařadíme mezi znaky, které OPmac interně považuje za speciální.

```
\addto\dospecials{\do\*}
```

6.2 Makro `\BEGSRC`

Makro `\BEGSRC` nejdříve přes `\futurelet` zjistí, jak je voláno, a nastaví příslušné přepínače. Pak se uloží (vizte podsekci 6.4) jednotlivé řádky prostředí `\BEGSRC` dovnitř maker `\SRConeline` a `\SRCblock` a všechny řádky se takto postupně vloží do makra `\SRCcontent`. V místě `\ENDSRC` (vizte podsekci 6.5) se pak v závislosti na jednotlivých přepínačích nadefinují makra `\SRConeline` a `\SRCblock` a expanduje se makro `\SRCcontent`.

Již v makru `\BEGSRC` a jeho větvích jsou opatrně měněny kategorie znaků, aby se případný první token prostředí načetl do `\futurelet` s kategorií platnou uvnitř prostředí, ale aby uvnitř argumentů maker `\BEGSRCb` a `\BEGSRCd` byly kategorie ještě původní.

```
\def\BEGSRC{\ifprintSRC\par\ttskip\fi
  \bgroup
  \catcode`\|=0 \catcode`\\=12
  \futurelet\BEGSRCt\BEGSRCa}
```

Makro `\BEGSRCa` zjistí, kterým způsobem je makro `\BEGSRC` voláno, a podle toho buď zavolá další makro `\BEGSRCb` (první dva způsoby), nebo zavolá přímo makro `\BEGSRCz` (třetí způsob). Zároveň nastaví přepínač `\ifsaveSRC`.

```
\def\BEGSRCa{\ifx<\BEGSRCt
    \saveSRCtrue
    \catcode`\|=12 \catcode`\\=0
    \expandafter\BEGSRCb
  \else
    \saveSRCfalse
    \expandafter\BEGSRCz
  \fi}
```

Makro `\BEGSRCb` uloží `InterniNazev` bloku pro další použití do makra `\SRCid` a zavolá rozhodovací makro `\BEGSRCc`.

```
\def\BEGSRCb<#1>{\def\SRCid{#1}%
  \catcode`\|=0 \catcode`\\=12
  \futurelet\BEGSRCt\BEGSRCc}
```

Makro `\BEGSRCc` zjistí, kterým způsobem je makro `\BEGSRC` voláno, a podle toho buď zavolá další makro `\BEGSRCd` (první způsob), nebo zavolá přímo makro `\BEGSRCz` (druhý způsob). Zároveň nastaví přepínač `\ifaddingSRC`.

```
\def\BEGSRCc{\ifx\bgroup\BEGSRCt
    \addingSRCfalse
    \catcode`\|=12 \catcode`\\=0
    \expandafter\BEGSRCd
  \else
    \addingSRCtrue
    \expandafter\BEGSRCz
  \fi}
```

Makro `\BEGSRCd` uloží `Zobrazený název` bloku pro další použití do makra `\csname SRCtit:\SRCid\endcsname` a zavolá makro `\BEGSRCz`.

```
82 \def\BEGSRCd#1{\sdef{SRCtit:\SRCid}{#1}\BEGSRCz}
```

Makro `\BEGSRCz` je analogií makra `\BEGSRC` z řádků 17 až 28, přičemž již respektuje nastavené přepínače. Na začátku prostředí `\BEGSRC` je možné použít háček `\SRChook`, na konci prostředí háček `\endSRChook`. Na začátku každého řádku je možné použít háček `\SRClinehook`. Každý řádek je načten a zpracován makrem `\SRCgetline`. Makro `\<` mimo `\SRCgetline` nedělá nic a v horizontálním módu na něj expand procesor nenarazí. Je však třeba makro `\<` nadefinovat, ať ve vertikálním módu bezpečně přejde do módu horizontálního a v něm ať zůstane neexpandované.

```
83 \def\BEGSRCz{\ifprintSRC\else\setbox\nonprintbox\vbox\bgroup\fi
84   \ifsaveSRC
85     \noindent\SRCangle{\csname SRCtit:\SRCid\endcsname}%
86       ${}\ifaddingSRC\mathrel{{+}{\equiv}}\else\equiv\fi$%
87     \par\nobreak
88   \fi
89   \def\SRCcontent{}%
90   \setSRC
91   \SRCensurehmode\<%
92   \SRCensurehmode\label
93   \parindent\ttindent
94   \csname SRChook\endcsname
95   \tenrm\thefontscale[700]\let\sevenrm\thefont
96   \everypar{\SRCgetline}%
97   \def\par##1{\endgraf\ifx##1\egroup\else\penalty\ttpenalty
98     \fi##1}%
99   \obeylines\startSRC}
```

6.3 Makro `\setSRC`

Se sazbou znaků uvnitř verbatim prostředí, které je vybaveno nějakou přidanou hodnotou, nutně souvisejí následující problémy.

- Co nejvíce znaků musí být přímo tisknutelných, tj. musejí mít kategorii 11 nebo 12. To zajistí makro `\dospecials` a definice makra `\do`.
- Kdyby měla mezera kategorii 10 (jako obvykle), pak by více mezer za sebou splynulo do jedné. Výhodné je nastavit mezeru jako aktivní znak a ten pak expandovat na normální mezeru.
- Některý znak musí mít kategorii 0 nebo 13, aby uvnitř prostředí `\BEGSRC` bylo možné zavolat makro, které zařídí onu přidanou hodnotu. V našem případě to je znak `|`, který bude uvozovat řídicí sekvence.
- Znak uvedený v předchozím bodě musí být možné také vytisknout. To jsme již zajistili na řádku 57.

Uvnitř prostředí `\BEGSRC` se navíc tabulátor expanduje na posloupnost mezer. A jestliže uživatel makrem `\activettchar` nastavil znak pro přechod do odstavcového verbatim módu, nastaví se tomuto znaku kategorie 12.

```
{\catcode9=12
  \gdef\setSRC{\def\do##1{\catcode`##1=12}\dospecials
    \ifx\savedttchar\undefined\else\catcode\savedttchar=12 \fi
    \adef{ }{\ }%
    \catcode`\|=0 \adef{^^I}{\SRCtab}}}
{\catcode32=13 \gdef\SRCtab{    }}
```

6.4 Makro `\SRCgetline`

Makro `\SRCscan` zjistí, zda argument makra `\SRCgetline` obsahuje `\<popisek>` a jaké je před ním odsazení.[16] Oboje uloží do příslušných maker. V argumentu mají mezery kategorii 13 a vlnovky kategorii 12. V níže použitém testu mají vlnovky kategorii 13, proto nemůže dojít k chybnému testu. Jestliže je odsazení nulové, platí `#1=\noexpand` dle definice makra `\SRCensurehmode`. V tom případě se `\SRCods` opraví na nic.

```
\def\SRCnoex{\noexpand}
\def\SRCscan#1\<#2>#3\SRCend{%
  \ifx~#3~%
    \let\SRCnaz\relax
  \else
    \def\SRCods{#1}%
    \ifx\SRCods\SRCnoex
      \def\SRCods{}%
    \fi
    \def\SRCnaz{#2}%
  \fi}
```

Makro `\SRCgetline` načte a vytiskne jeden řádek.

- Pokud řádek obsahuje `\<popisek>`, uloží se `popisek` do makra `\SRCnaz` a odsazení před ním do makra `\SRCods`. Případný další text za `\<...>` se ignoruje. Do makra `\SRCcontent` se přidá `\SRCblock{odsazení}{popisek}`.
- V opačném případě se do makra `\SRCcontent` přidá `\SRConeline{#1}`.

Oddělovačem makra `\SRCgetline` je token $\boxed{\texttt{\^{}\^{}M}}_{13}$, který se vytvoří díky makru `\obeylines` použitému na řádku 99.

```
\begingroup\lccode`\~13
    \lowercase{\endgroup\def\SRCgetline#1~}{%
```

[16] Jako odsazení se chápe text (zpravidla posloupnost mezer) před prvním výskytem `\<...>`.

```
\SRCscan#1\<>\SRCend
\ifx\SRCnaz\relax
  \addto\SRCcontent{\SRConeline{#1}}%
  \SRCprintline{#1}%
\else
  \expandafter\expandafter\expandafter\addto
    \expandafter\expandafter\expandafter\SRCcontent
    \expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter\SRCblock
      \expandafter\expandafter\expandafter{%
      \expandafter\SRCods\expandafter}\expandafter{\SRCnaz}}%
  \SRCprintblock{\SRCods}{\SRCnaz}%
\fi\par}
```

Makro `\SRCprintline` vysází jeden řádek programu. Čísla vysázených řádků jsou uložena v registru `\SRClinenum`. Pokud je `\SRClinenum<0`, pak se číslo nevysází. Na začátku každého řádku je možné použít `\SRClinehook`.

```
\newcount\SRClinenum
\def\SRCprintline#1{\SRCprintlinenum#1\par}
\def\SRCprintlinenum{\ifprintSRC
    \ifnum\SRClinenum<0 \else
      \global\advance\SRClinenum1
    \fi
  \fi
  \csname SRClinehook\endcsname
  \quitvmode
  \ifnum\SRClinenum<0 \else
    \llap{\sevenrm\the\SRClinenum\kern.9em}%
  \fi}
```

Makro `\SRCprintblock` vysází odkaz na blok. Mezery v odsazení mají kategorii 13 a chovají se jako ostatní mezery uvnitř prostředí `\BEGSRC`.

```
\def\SRCprintblock#1#2{\SRCprintlinenum
  #1\SRCangle{\csname SRCtit:#2\endcsname}\par}
```

6.5 „Makro“ `\ENDSRC`

Makra definovaná v této podsekci se budou provádět v místě použití `\ENDSRC`. Proto má podsekce tento název. Ve skutečnosti ale žádná sekvence `\ENDSRC` není definována. Načtení řádků až po tokeny $\boxed{\backslash}_{12}\,\boxed{\mathtt{E}}_{11}\,\boxed{\mathtt{N}}_{11}\,\boxed{\mathtt{D}}_{11}\,\boxed{\mathtt{S}}_{11}\,\boxed{\mathtt{R}}_{11}\,\boxed{\mathtt{C}}_{11}$ řeší makro `\startSRC`. Pokud je nastaveno `\printSRCfalse`, pak je teoreticky možné uvolnit paměť nastavením `\nonprintbox` na prázdný vbox. Nicméně toto

realizováno není, což umožňuje pomocí háčku `\endSRChook` obsah `\nonprintbox` dále zpracovat.

Samotné vysázení jednotlivých řádků je vyřešeno expanzí `|tt#1` na řádku 147, kdy se prostřednictvím `\everypar` (vizte řádek 96) volá makro `\SRCgetline` a to se expanduje na `\SRCprintline` nebo na `\SRCprintblock`.

```
{\catcode`\|=0 \catcode`\\=12
  |gdef|startSRC#1\ENDSRC{|tt#1|doSRC
    |ifprintSRC
      |egroup|par|ttskip
    |else
      |egroup|egroup
    |fi
    |csname endSRChook|endcsname|testparA}}
```

Makro `\doSRC` zajistí, že řádky a bloky budou dále zpracovány v závislosti na způsobech volání makra `\BEGSRC`.

- U prvního způsobu se do `ds` zapíše řádek
  `\DEFSRC {InterniPopisek}{Formátovaný popisek}`
- U druhého způsobu se do `ds` zapíše řádek
  `\ADDSRC {InterniPopisek}`
- V obou případech se pak zavolá háček `\SRCdshook`, v němž se například mohou lokálně nadefinovat další makra. Dále se jednotlivé řádky obsažené v `\SRCcontent` zapíšou do `ds` makrem `\SRCwritedsline` a bloky makrem `\SRCreadblock`.[17] Po ukončení bloku se do `ds` zapíše token `\ENDDEFSRC`.
- U třetího způsobu se pak pomocí maker `\SRCwriteline` a `\SRCwriteblock` zapíše obsah `\SRCcontent` přímo do `txt`.

```
\def\doSRC{\everypar{}%
  \def\ { }%
  \ifsaveSRC
    \ifaddingSRC
      \immediate\write\SRCdsfile{\noexpand\ADDSRC{\SRCid}}%
    \else
      \immediate\write\SRCdsfile{\noexpand\DEFSRC
        {\SRCid}{\csname SRCtit:\SRCid\endcsname}}%
    \fi
    \let\SRConeline\SRCwritedsline
    \let\SRCbeginline\relax
    \let\SRCendline\relax
```

[17] Trik na řádcích 166 a 167 způsobí, že se token `\SRCblock` zapíše do `ds` jako neexpandovaný token `\SRCreadblock`, za nímž budou následovat původní „argumenty“ makra `\SRCblock` vložené na řádku 129.

```
    \def\SRCblock{\SRCreadblock}%
    \let\SRCreadblock\relax
    \csname SRCdshook\endcsname
    \immediate\write\SRCdsfile{\SRCcontent}%
    \immediate\write\SRCdsfile{\noexpand\ENDDEFSRC}%
  \else
    \let\SRConeline\SRCwriteline
    \let\SRCblock\SRCwriteblock
    \SRCcontent
  \fi}
```

6.6 Cesta z tex do ds

Makro `\SRCwritedsline` zapíše do ds jeden řádek ve tvaru

```
\SRCbeginline : ... \SRCendline
```

Uvědomme si, že při zápisu do souboru se za název řídicí sekvence skládající se z písmen automaticky vkládá mezera. Tato mezera a všechny bezprostředně navazující mezery jsou poté při načítání souboru pohlceny token procesorem. Znak dvojtečka způsobí, že případné mezery na začátku řádku programu (tj. za touto dvojtečkou) nebudou při načítání souboru pohlceny.

```
\def\SRCwritedsline#1{\SRCbeginline:#1\SRCendline}
```

Jak již bylo řečeno v poznámce 17, blok programu se do ds zapíše ve tvaru

```
\SRCreadblock {   }{...}
```

6.7 Cesta z ds do expand procesoru

Již víme, že uvnitř ds jsou uloženy jednotlivé bloky programu v následujícím tvaru.

```
\DEFSRC {InterniPopisek}{Formátovaný popisek}
  \SRCbeginline :...\SRCendline ...\SRCreadblock {   }{...}...
\ENDDEFSRC
\ADDSRC {InterniPopisek}
  \SRCbeginline :...\SRCendline ...\SRCreadblock {   }{...}...
\ENDDEFSRC
```

Makro `\DEFSRC#1#2` uloží argument `#1` do makra `\SRCid` a argument `#2` do makra `\csname SRCtit:\SRCid\endcsname`. Další obsah až po `\ENDDEFSRC` se postupně uloží do makra `\csname SRCcon:\SRCid\endcsname`.

```
\def\DEFSRC#1#2{\def\SRCid{#1}\sdef{SRCtit:#1}{#2}%
  \sdef{SRCcon:#1}{}}
```

Podobně makro `\ADDSRC#1` uloží argument `#1` do makra `\SRCid` a další obsah až po `\ENDDEFSRC` se do makra `\csname SRCcon:\SRCid\endcsname` přidá postupně.

```
179 \def\ADDSRC#1{\def\SRCid{#1}}
```

Makro `\SRCbeginline:` načte verbatim text až po tokeny $\boxed{\backslash}_{12}\ \boxed{S}_{11}\ \boxed{R}_{11}\ \boxed{C}_{11}\ \boxed{e}_{11}\ \boxed{n}_{11}\ \boxed{d}_{11}\ \boxed{l}_{11}\ \boxed{i}_{11}\ \boxed{n}_{11}\ \boxed{e}_{11}$ a tento text přidá k aktuálnímu obsahu makra `\csname SRCcon:\SRCid\endcsname` jako argument makra `\SRConeline`. Všechny mezery se načtou s kategorií 13 (díky makru `\setSRC` a jeho řádku 103) a pak se uvnitř makra `\doSRC` (díky řádku 155) postupně expandují na mezery kategorie 10. Uvnitř definice `\SRCbeginlineA` je třeba s mezerami pracovat opatrně.

```
180 \def\SRCbeginline:{\bgroup\setSRC \catcode`\|=12 \SRCbeginlineA}
181 {\catcode`\|=0 \catcode`\\=12
182   |gdef|SRCbeginlineA#1\SRCendline{|egroup
183     |expandafter|addto|csname SRCcon:|SRCid|endcsname
184       {|SRConeline{#1}}%
185     |ignorespaces}}
```

Makro `\SRCreadblock#1#2` přidá `\SRCblock{#1}{#2}` k aktuálnímu obsahu makra `\csname SRCcon:\SRCid\endcsname`. Je nutné zajistit, aby se mezery z odsazení načítaly s kategorií 13. V souboru `ds` je za tokenem `\SRCreadblock` mezera a ta je na řádku 187 při načítání čísla pohlcena. Poté se pomocí maker `\SRCreadblockA` a `\SRCreadblockB` načtou jednotlivé argumenty.

```
186 \def\SRCreadblock{\begingroup\afterassignment\SRCreadblockA
187   \catcode32=13}
188 \def\SRCreadblockA#1{\gdef\SRCods{#1}\endgroup\SRCreadblockB}
189 \def\SRCreadblockB#1{\expandafter\addto
190   \csname SRCcon:\SRCid\expandafter\endcsname\expandafter{%
191     \expandafter\SRCblock\expandafter{\SRCods}{#1}}}
```

Makro `\ENDDEFSRC` nedělá nic a je použito pouze proto, aby byl obsah souboru `\jobname.ds` čitelnější.

```
192 \let\ENDDEFSRC\relax
```

Soubor `ds` se načítá na začátku dokumentu v průběhu načítání souboru `gensrc.tex`. Pokud soubor `ds` neexistuje, nic se nestane. Pro test existence je použito makro `\softinput` z *TEXbooku naruby* [4, sekce 7.1].

```
193 \newread\testin
194 \def\softinput #1 {\let\next=\relax \openin\testin=#1
195   \ifeof\testin \message{Warning: the file #1 does not exist}%
```

```
196   \else \closein\testin \def\next{\input #1 }\fi
197   \next}
198 \softinput\jobname.ds
```

Interně bude ds reprezentován řídicí sekvencí \SRCdsfile. Soubor se po svém načtení ihned otevře pro zápis.

```
199 \newwrite\SRCdsfile
200 \immediate\openout\SRCdsfile\jobname.ds
```

6.8 Cesta z expand procesoru do txt

V makru \SRCodsazeni je uloženo aktuální odsazení celého bloku, které je uvnitř vnořených \SRCwriteblock lokálně aditivní. Na začátku je odsazení prázdné.

```
201 \def\SRCodsazeni{}
```

Makro \SRCwriteline zapíše do generovaného programu jeden řádek. Na začátku každého zapsaného řádku je možné použít \SRCwritehook.

```
202 \def\SRCwriteline#1{\csname SRCwritehook\endcsname
203   \immediate\write\SRCfile{\SRCodsazeni#1}}
```

Makro \SRCwriteblock lokálně přidá nové odsazení k aktuální hodnotě makra \SRCodsazeni. Mezery v odsazení mají kategorii 13. S tímto odsazením se pak expanduje obsah aktuálního bloku.

Jelikož expand procesor provádí úplnou expanzi, jsou takto na řádku 206 expandovány i případné vnořené bloky programu.

Navíc díky použití \csname nedojde k chybě v případě, že blok s daným názvem není definován, například v prvním průchodu TEXem.

```
204 \def\SRCwriteblock#1#2{\begingroup
205   \addto\SRCodsazeni{#1}%
206   \csname SRCcon:#2\endcsname
207   \endgroup}
```

7 Aplikace

7.1 Imitování podprogramů

V balíčku gensrc jsou bloky programu interně definovány jako makra. Proto je možné stejný blok použít na více místech a do txt jej exportovat vícekrát. Takto je možné celkem přímočaře imitovat volání podprogramů v programovacích jazycích, které volání podprogramů neumožňují.

Jak se ale vyrovnat s podprogramy, které mají argumenty? Použijeme globální proměnné, které si rezervujeme pouze pro použití uvnitř daného „podprogramu“.

V našem příkladě to budou proměnné SDRin1 a SDRin2, jejichž hodnotu nastavíme před voláním „podprogramu“ a které budou sloužit jako jeho argumenty, a proměnná SDRout, do níž „podprogram“ uloží svou výstupní hodnotu.

V ukázkách je část zdrojového souboru. Jak vypadá vygenerovaný program, si lze snadno domyslet.

```
\BEGSRC<SDR>{Concat solutions}
SDRout=SDRin2.clone();
if (SDRin1.max > SDRin2.max) {SDRout.max = SDRin1.max;}
else {SDRout.max = SDRin2.max;}
SDRout.position = SDRin1.position.concat(SDRin2.position);}
\ENDSRC
```

```
\BEGSRC
SDRin1=mainsolution.clone();
SDRin1.position=mainsolution.position.clone();
SDRin2=ZPRout.clone();
SDRin2.position=ZPRout.position.clone();
|<SDR>
mainsolution=SDRout.clone();
mainsolution.position=SDRout.position.clone();
\ENDSRC
```

Tímto způsobem sice není možné imitovat rekurzivní volání podprogramů, nicméně toto v rámci řešení původního úkolu nebylo potřeba.

7.2 Koordinace více programů

Představme si situaci, kdy píšeme několik programů, třeba i v různých programovacích jazycích, a tyto programy mají vzájemně spolupracovat. Například si programy mohou předávat data, kdy výstup některé funkce jednoho programu je použit pro vstup příslušné funkce druhého programu. Při vývoji programů pak potřebujeme, aby tyto funkce byly spolu kompatibilní. Pro lepší údržbu a pochopení je výhodnější, když jsou ve zdrojovém souboru a v dokumentaci tyto funkce poblíž sebe.

Z pohledu balíčku gensrc se jedná o situaci, kdy průběžně generujeme dva soubory txt, přičemž se dokonce může stát, že některé části kódu budou společné. Soubory txt můžeme generovat každý se svým blokem a tyto bloky poté definovat průběžně.

```
\SRCFILENAME programA.txt
\BEGSRC
|<A>
\ENDSRC
```

```
\SRCFILENAME programB.txt
\BEGSRC
|<B>
\ENDSRC
\BEGSRC<A>{Program A}
|<firstpart>
in the first
|<secondpart>
\ENDSRC
\BEGSRC<B>{Program B}
|<firstpart>
in the second
|<secondpart>
\ENDSRC
\BEGSRC<firstpart>{First part}
This will be
\ENDSRC
\BEGSRC<secondpart>{Second part}
program.
\ENDSRC
```

Může být užitečné v dokumentaci odlišit barvou, ke kterému generovanému souboru příslušná část kódu patří. K tomu lze využít háček `\SRChook`.

V následující ukázce navíc háčky nastavíme tak, aby jejich změna byla lokální a aby se po ukončení prostředí `\BEGSRC` předefinovaly zpět na předchozí hodnotu. Rozbor ukázky ponechávám na čtenáři.

```
\def\switchSRC#1#2{%
  \sdef{#1BEGSRC}{%
    \let\exSRChook\SRChook
    \def\SRChook{\longlocalcolor#2%
      \global\let\SRChook\exSRChook}%
    \BEGSRC}}
\switchSRC{a}{\Red}
\switchSRC{b}{\Green}
```

```
\SRCFILENAME programA.txt
\aBEGSRC
|<A>
\ENDSRC
\SRCFILENAME programB.txt
\bBEGSRC
|<B>
\ENDSRC
```

```
\aBEGSRC<A>{Program A}
|<firstpart>
in the first
|<secondpart>
\ENDSRC
\bBEGSRC<B>{Program B}
|<firstpart>
in the second
|<secondpart>
\ENDSRC
\BEGSRC<firstpart>{First part}
This will be
\ENDSRC
\BEGSRC<secondpart>{Second part}
program.
\ENDSRC
```

Příslušná dokumentace pak vypadá následovně.

```
 1 ⟨Program A⟩

 2 ⟨Program B⟩

⟨Program A⟩ ≡
 3 ⟨First part⟩
 4 in the first
 5 ⟨Second part⟩

⟨Program B⟩ ≡
 6 ⟨First part⟩
 7 in the second
 8 ⟨Second part⟩

⟨First part⟩ ≡
 9 This will be

⟨Second part⟩ ≡
10 program.
```

7.3 Křížové odkazy

V balíčku gensrc není implementován mechanismus maker jako v jazyku WEB. Pokud si ale uvědomíme, že uvnitř prostředí \BEGSRC má znak | kategorii 0, tj. uvozuje název řídicí sekvence, pak není problém

```
v programu použít jednoduché |makro nebo dokonce
použít |Makro(s jedním)(i více) argumenty,
```

pokud si řídicí sekvence \makro a \Makro předem nadefinujeme. Pouze si musíme dát pozor na to, aby v okamžiku definování měly znaky, které oddělují argumenty, kategorii 12, případně si musíme správné načtení argumentů zajistit jiným způsobem.

V následujících ukázkách budeme používat také makra, která nebudeme definovat globálně, ale využijeme mechanismus háčků k tomu, abychom v různých okamžicích jejich definici měnili lokálně.

Když autor tohoto článku začínal s programováním, měl počítač Commodore 64 s jazykem BASIC. Po třech desetiletích objevil na internetu emulátor tohoto počítače [9] a z nostalgie si v něm trochu zaprogramoval.

Emulátor má jako počítač Commodore 64 stejně nízkou rychlost, stejně nízkou přesnost reálné aritmetiky a stejně nízké rozlišení obrazovky. Tyto nevýhody autor přetavil ve výhody a vytvořil názorný program používaný pro řešení úloh v jím vyučovaném předmětu Numerická matematika.

Pro jazyk BASIC je charakteristické, že

- program tvoří posloupnost příkazů, která nijak neodpovídá logické struktuře programu.
- Na začátku každého řádku je uvedeno číslo tohoto řádku. Bývá zvykem, že posloupnost čísel řádků má krok 10.
- Program obsahuje skoky na řádky s daným číslem.

Tyto problémy jsou dobře vidět na následující ukázce části programu.

```
1190 let ra=15: gosub 2080: input a
1200 let ra=16: gosub 2080: input b
1210 if b<a then let c=a: let a=b: let b=c
1220 let x=a: gosub 3100: let d=f
1230 let x=b: gosub 3100
1240 if sgn(f)*sgn(d)<0 then 1270
1250 let ra=17: gosub 2270
1260 goto 1390
```

S balíčkem gensrc je možné uvedené problémy vyřešit následovně.

- Logickou strukturu programu lze vytvořit a udržovat pomocí bloků.
- Číslo řádku je známo až po úplné expanzi všech bloků při zápisu do souboru txt. Na tom místě lze využít háček `\SRCwritehook`.
- Pro skoky lze použít mechanismus křížových odkazů. Přitom číslo řádku v dokumentaci je obecně jiné než číslo řádku v programu. V dokumentaci pak křížové odkazy mohou být aktivní.

V ukázce je příslušná část dokumentace.

3.2.4 Bisekce

Na začátku bisekce uživatel zadá krajní body do proměnných `A`, `B`.

⟨*Bisekce*⟩ ≡

```
370 LET RA=15: GOSUB 139: INPUT A
371 LET RA=16: GOSUB 139: INPUT B
```

Ze slušnosti je zařízeno, aby platilo `A < B`.

⟨*Bisekce*⟩ +≡
```
372 IF B<A THEN LET C=A: LET A=B: LET B=C
```

Pokud nejsou v krajních bodech různá znaménka, vypíše se chybové hlášení a metoda se ukončí.

⟨*Bisekce*⟩ +≡
```
373 LET X=A: GOSUB 173: LET D=F
374 LET X=B: GOSUB 173
375 IF SGN(F)*SGN(D)<0 THEN 378
376 LET RA=17: GOSUB 158
377 GOTO 390
```

Podívejme se, jak lze uvedený mechanismus čísel řádků a křížových odkazů implementovat.

Pro křížové odkazy budeme používat řídicí sekvence `\LL(#1)` a `\RR(#1)`, kde oddělovače argumentů mají kategorii 12. Tyto sekvence budou na grafu na straně 80 cestovat všemi šipkami. Pomocí háčků budeme na jednotlivých místech definovat, jak se mají tyto sekvence expandovat nebo neexpandovat.

Při sazbě dokumentace se makro `\LL(#1)` chová stejně jako `\label[S:#1]`, přičemž se odkazuje na číslo řádku `\SRClinenum`. Makro `\RR(#1)` se chová jako `\ref[S:#1]` se změnou fontu dle řádku 142.

```
216 \def\SRChook{\def\LL(##1){%
217     \immediate\write\reffile
218       {\string\Xlabel{S:##1}{\the\SRClinenum}}%
219     \dest[ref:S:##1]}%
220   \def\RR(##1){{\sevenrm\ref[S:##1]}}}
```

Při zápisu do `ds` se makra pouze přepíšou s uvozujícím znakem `|`. Ten má při zápisu do `ds` kategorii 12 a při opětovném načtení `ds` kategorii 0. Obě makra tak budou i nadále jako makra fungovat.

```
221 \def\SRCdshook{\def\LL(##1){|LL(##1)}%
222   \def\RR(##1){|RR(##1)}}
```

Nejsložitější je zápis do `txt`, protože při zápisu do souboru není možné provádět příkazy hlavního procesoru. V našem případě potřebujeme příkazy pro

- zvýšení čísla řádku a
- zápis čísla řádku do souboru `\jobname.ref`, který se v OPmacu používá pro realizaci křížových odkazů.

Háček `\SRCwritehook` nadefinujeme tak, že namísto zápisu řádku do `txt` tento řádek načte,[18] dále zpracuje a sám zajistí zápis. Nejdříve řádek vysází do pomocného `\box0`, přičemž uvnitř tohoto boxu lokálně nadefinuje `\LL(#1)` jako

[18] V ukázce na straně 83 je vidět, kdy přesně se háček `\SRCwritehook` volá a jakým způsobem lze pomocí něj načíst argument příkazu `\write`.

`\label[B:#1]`, zatímco `\RR(#1)` ignoruje. Poté řádek zapíše do txt, přičemž na začátek řádku vloží jeho číslo, `\LL(#1)` ignoruje a `\RR(#1)` lokálně nadefinuje jako `\ref[B:#1]`.

```
\newcount\cislo
\def\SRCwritehook#1\SRCfile#2{\advance\cislo10
  {\setbox0=\hbox{%
    \def\LL(##1){\immediate\write\reffile
      {\string\Xlabel{B:##1}{\the\cislo}}}%
    \def\RR(##1){}%
    \lowercase{#2}}}%
  {\def\LL(##1){}%
    \def\RR(##1){\csname lab:B:##1\endcsname}
    \lowercase{\immediate\write\SRCfile{\the\cislo\space#2}}}}
```

Následuje ukázka příslušné části zdrojového kódu s využitím výše uvedených maker.

```
\seccc Bisekce

Na začátku bisekce uživatel zadá krajní body do proměnných "A", "B".
\BEGSRC<bisekce>{Bisekce}
|vypisretezec(15): INPUT A |LL(bisekce)
|vypisretezec(16): INPUT B
\ENDSRC
Ze slušnosti je zařízeno, aby platilo "A"${}<{}$"B".
\BEGSRC<bisekce>
IF B<A THEN LET C=A: LET A=B: LET B=C
\ENDSRC
Pokud nejsou v~krajních bodech různá znaménka, vypíše se chybové hlášení a
metoda se ukončí.
\BEGSRC<bisekce>
LET X=A: GOSUB |RR(deffce): LET D=F
LET X=B: GOSUB |RR(deffce)
IF SGN(F)*SGN(D)<0 THEN |RR(bis1)
|vypisretezecln(17)
GOTO |RR(bis0)
\ENDSRC
```

Vygenerovaný program lze programem petcat [10] převést do souboru s příponou prg, ve kterém je po bajtech uložen obsah paměti počítače Commodore 64, jaký by byl po načtení programu do počítače. Soubor prg lze spustit v emulátoru [9]. Programem prg2wav [11] lze dále vytvořit soubor s příponou wav, který odpovídá zvuku nahranému na magnetofonovou kazetu.[19] Soubor wav lze dále na

[19] Dříve narozenému čtenáři se zajisté okamžitě vybavilo ono charakteristické „chrrr píp píííp píip chrrrrr píííp“. :-)

kazetu nahrát, nicméně autorovi těchto řádků se nepodařilo jej nahrát tak, aby jej počítač Commodore 64 načetl správně.[20]

7.4 Program ve více jazycích

V této podsekci se zaměříme na situaci, kdy potřebujeme vytvořit stejný program v několika podobných programovacích jazycích. V této situaci s úspěchem využijeme TEXová makra, která se v závislosti na přepínači budou expandovat různě.

V ukázkách se bude jednat o dva dialekty jazyka SQL, přičemž aktuální dialekt bude určovat makro `\version`. Podle něj se nastaví přepínač `\ifOR` a ten poté bude rozhodovat o správné expanzi maker.

```
\def\versionOR{OR}
\unless\ifdefined\version \let\version\versionOR \fi
\newif\ifOR
\ifx\version\versionOR \ORtrue \fi
```

Vytvoříme makro `\variant#1#2#3`, které nadefinuje makro `#1` s jedním argumentem, aby se expandovalo jako `#2`, nebo jako `#3`.

```
\def\variant#1#2#3{\ifOR \sdef{#1}##1{#2}%
  \else \sdef{#1}##1{#3}\fi}
```

Dále nadefinujeme samotná makra s jejich expanzí v jednotlivých dialektech.

```
\variant{integer}{number(38)}{decimal(38)}
{\catcode`\_=12 \globaldefs=1
  \variant{rownum}{rownum}
    {(row_number() over (order by #1))}}
```

Uvnitř prostředí `\BEGSRC` můžeme makra volat s uvozujícím znakem `|`. Nicméně ukážeme si, jak je možné pro volání maker využít znak kategorie 13. Znak `@` nadefinujeme tak, aby načítal název makra až po další znak `@` a pak načítal argument makra až po třetí znak `@`. Znak musí být aktivní v okamžiku definování i uvnitř prostředí `\BEGSRC`. Případné vysázení znaku `@` se provede pomocí `@@@`.

```
{\catcode`\@=13 \gdef@#1@#2@{\csname#1\endcsname{#2}}}
\sdef{}#1{@}
\def\SRChook{\catcode`\@=13 }
```

S těmito definicemi již můžeme psát části programu.

[20] Pokud někdo ze čtenářů má s nahráváním souboru `wav` na kazetu zkušenosti, bude autor za předání informací rád.

```
\SRCFILENAME program\version.txt
\BEGSRC
select cast(NN/p as @integer@@), st+1
from FindN1,
    (select @rownum@p@ r, p from SmallFactors) a
where st=r
\ENDSRC
```

Příslušný vygenerovaný program a dokumentace při nastavení `\ORtrue` jsou následující:

```
select cast(NN/p as number(38)), st+1
from FindN1,
    (select rownum r, p from SmallFactors) a
where st=r
```

```
1 select cast(NN/p as number(38)), st+1
2 from FindN1,
3     (select rownum r, p from SmallFactors) a
4 where st=r
```

Podobně pro `\ORfalse`.

```
select cast(NN/p as decimal(38)), st+1
from FindN1,
    (select (row_number() over (order by p)) r, p from SmallFactors) a
where st=r
```

```
1 select cast(NN/p as decimal(38)), st+1
2 from FindN1,
3     (select (row_number() over (order by p)) r, p from SmallFactors) a
4 where st=r
```

Jak ale zajistit, abychom vygenerovali program najednou v obou dialektech a nemuseli přitom ručně nastavovat `\version`? Stačí si vytvořit řídicí soubor, kterým TEX spustíme dávkově.

```
pdfcsplain soubor
pdfcsplain soubor
mv soubor.pdf souborOR.pdf
pdfcsplain '\def\version{MS}\input soubor'
pdfcsplain '\def\version{MS}\input soubor'
mv soubor.pdf souborMS.pdf
```

Odkazy

1. OLŠÁK, Petr. *OPmac: Rozšiřující makra plain TEXu* [online]. [cit. 2023-11-13]. Dostupné z: https://petr.olsak.net/opmac.html.
2. KNUTH, Donald E. *Literate Programming* [online]. 1983-09. [cit. 2023-11-13]. Dostupné z: http://www.literateprogramming.com/knuthweb.pdf. Submitted to The Computer Journal.
3. KNUTH, Donald E. *TEX: The Program*. Sv. B. Reading, MA: Addison-Wesley, 1986. Computers & Typesetting. Dostupné také z: https://tug.ctan.org/systems/knuth/dist/tex/tex.web.
4. OLŠÁK, Petr. *TEXbook naruby*. 2. vyd. Konvoj, 2001. Dostupné také z: http://petr.olsak.net/ftp/olsak/tbn/tbn.pdf.
5. THE LATEX PROJECT TEAM; MITTELBACH, Frank. docstrip*: Remove comments from file* [online]. 2022-09-03. [cit. 2023-11-13]. Dostupné z: https://ctan.org/pkg/docstrip.
6. MITTELBACH, Frank. *The* doc *and* shortvrb *Packages* [online]. 2023-11-01. [cit. 2023-11-13]. Dostupné z: https://mirrors.ctan.org/macros/latex/base/doc.pdf.
7. MITTELBACH, Frank; DUCHIER, Denys; BRAAMS, Johannes; WOLIŃSKI, Marcin; WOODING, Mark. *The* DocStrip *program* [online]. 2023-11-01. [cit. 2023-11-13]. Dostupné z: http://mirrors.ctan.org/macros/latex/base/docstrip.pdf.
8. ŠUSTEK, Jan. gensrc.tex [online]. 2023-11-07. [cit. 2023-11-13]. Dostupné z: https://github.com/jsustek/gensrc/blob/main/gensrc.tex.
9. *C64 online emulator* [online]. [cit. 2023-11-13]. Dostupné z: https://virtualconsoles.com/online-emulators/c64/.
10. *VICE: The Versatile Commodore Emulator* [online]. [cit. 2023-11-13]. Dostupné z: https://vice-emu.sourceforge.io/.
11. TOM THE DRAGON. *prg2wav: C64 PRG to Turbo Tape 64* [online]. [cit. 2023-11-13]. Dostupné z: https://github.com/tomdwaggy/prg2wav.

Summary: On Generating Documented Source Code by Blocks in TEX

This paper concerns writing programs and their documentation. We show author's package gensrc running on OPmac, which allows to write both program and its documentation in one TEX file. We also show more possibilities and applications of this package.

Keywords: documentation, literate programming, OPmac, gensrc

Jan Šustek, jan.sustek@seznam.cz

Jak umožnit stránkový zlom uvnitř vložených obrázků

Jan Šustek

V článku jsou definována a popsána makra TEXu pro vložení objektů, které jsou natolik vysoké, že komplikují stránkový zlom. Může se jednat o obrázky, vysázený text nebo obecně o obsah boxu. Makra vloží objekt na aktuální místo sazby a umožní uvnitř objektu provést stránkový zlom.

Klíčová slova: rámečky, obrázky, řezání, stránkový zlom

1 Úvod

Když autor psal předchozí článek tohoto Zpravodaje začínající na straně 66, narazil na problém, že článek obsahuje velké množství ukázek, které bylo nutné opticky oddělit od okolního textu. Autor se rozhodl ukázky vložit do šedých rámečků. Rámečky však musejí být vysoké, jinak by ukázky nemohly ukázat vše potřebné. To samozřejmě velmi komplikuje stránkový zlom. Přitom využití plovoucích objektů by zhoršilo plynulost textu a odkazy na plovoucí objekty by zhoršily srozumitelnost vět.

Proto se autor rozhodl umisťovat rámečky přímo k příslušnému textu a v rámečcích umožnit stránkový zlom. Ten ale nelze provést v libovolném místě rámečku. Mohlo by se totiž stát, že se zlom provede uprostřed řádku. Ukázky přitom pocházejí z vnějších souborů a do těch autor článku často nemohl snadno zasáhnout.

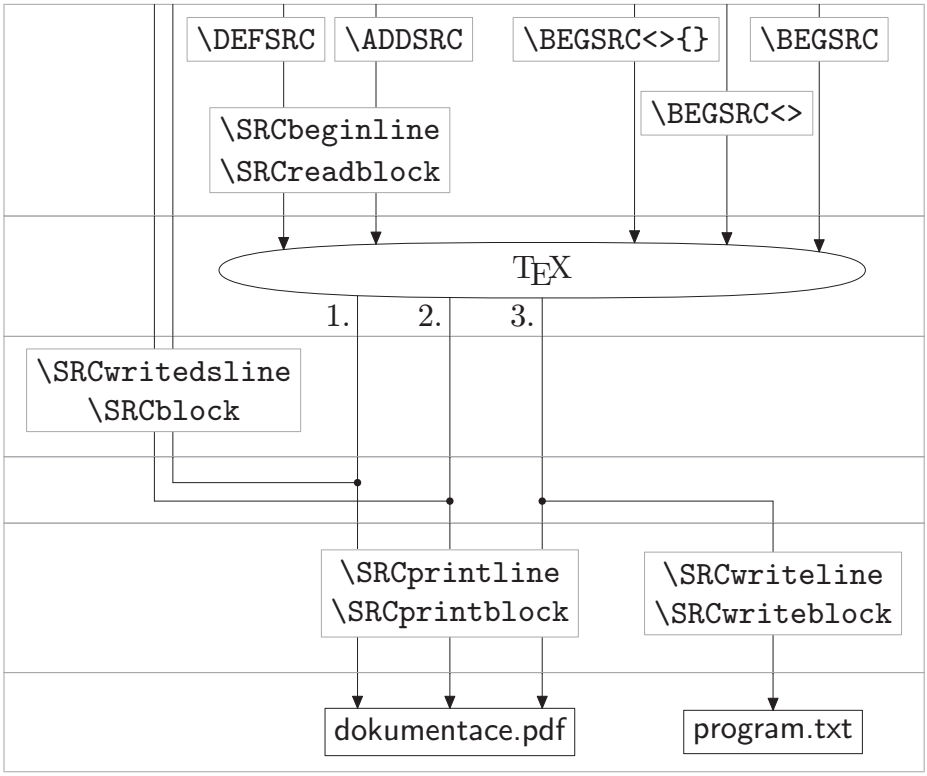
Níže definovaná makra umožní do rámečku vložit soubor s obrázkem[1] nebo obsah TEXového boxu a nastavit seznam míst, v nichž je možné provést stránkový zlom. Interně se provede to, že pomocí operátorů jazyka pdf se vložený obrázek ořeže v daných místech na nevysoké části, tyto části se poskládají pod sebe a mezi nimi se umožní stránkový zlom.

2 Použití

Použití maker ukážeme na grafu ze strany 80 tohoto Zpravodaje:

[1] Podporovány jsou všechny formáty obrázků, které lze vložit primitivem `\pdfximage`, tj. jbig2, jpeg, pdf a png.

DOI: 10.5300/2023-3-4/102



Graf jsme vložili pomocí následujících příkazů:

```
\input rezani
\ladenitrue
...
\ram{095,35,495,64,72,9}{graf-1.pdf}
```

Parametry určují, že ve vkládaném souboru graf-1.pdf umožníme stránkový zlom na souřadnicích 9,5 %, 35 %, 49,5 %, 64 %, 72 % a 90 % (měřeno od horního okraje obrázku). Při nastaveném `\ladenitrue` se navíc zobrazí místa možných řezů, abychom snadněji mohli určit jejich souřadnice.

Uvedená makra je také možné použít pro stránkový zlom uvnitř `\vboxu`, v němž může být vysázen například verbatim výpis. Použijeme ukázku ze strany 78 tohoto Zpravodaje. Ukázka je v OPmacu, ale s ohledem na šablonu Zpravodaje použijeme pro sazbu ukázky LaTeX. Přitom je použito prostředí `Verbatim` z balíčku fancyvrb, které umožňuje uvnitř verbatim výpisu volat makra, například k barevnému zvýraznění textu.

```
\def\zelena#1{\textcolor[cmyk]{1,0,1,0}{#1}}
\def\fialova#1{\textcolor[cmyk]{0,1,0,0}{#1}}
\nastavobrbox
\vskip-\medskipamount
\vskip0pt
\begin{Verbatim}[commandchars=/(),numbers=none]
\input gensrc
\def\SRChook{\longlocalcolor\Red}
\SRCFILENAME program.txt
Here we define a~block which will be inserted to another
  place.
/zelena(\BEGSRC<InternalLabel>{Displayed label})
second line
  /fialova(|<InnerBlock>)
/zelena(\ENDSRC)
...
\end{Verbatim}
\vskip-\medskipamount
\endnastavobrbox
\dily23 \ram{3,5,9,13,18}{}
```

Makra `\nastavobrbox` a `\endnastavobrbox` jsou definována v sekci 3 a pomocí nich se vytvoří `\vbox`, do nějž je pak vložen obsah prostředí `Verbatim`. Tento box pak rozřežeme na 23 stejných dílů a umožníme stránkový zlom po 3, 5, 9, 13 a 18 dílech. Přitom musíme odstranit mezeru `\medskipamount`, kterou prostředí `Verbatim` vkládá nad a pod svůj obsah.

```
\input gensrc
\def\SRChook{\longlocalcolor\Red}
\SRCFILENAME program.txt
```

```
Here we define a~block which will be inserted to another
  place.
```

```
\BEGSRC<InternalLabel>{Displayed label}
second line
  |<InnerBlock>
\ENDSRC
```

```
A~block can be defined by parts.
\BEGSRC<InternalLabel>
fourth line
\ENDSRC
```

```
And here we insert the block.
\BEGSRC
first line
    |<InternalLabel>
\ENDSRC
```

```
Finally we define the inner block.
\BEGSRC<InnerBlock>{Inner block}
third line
\ENDSRC
\bye
```

3 Syntaxe

Je možné použít některou z následujících syntaxí:

- `\ram{}{obrázek}` vloží obrázek do rámečku, který se nezlomí.
- `\ram{1,23,4567,89}{obrázek}` vloží obrázek do rámečku a umožní stránkový zlom na souřadnicích 10 %, 23 %, 45,67 % a 89 % (měřeno od horního okraje obrázku).
- `\dily11 \ram{3,5,7,8}{obrázek}` podobně umožní stránkový zlom na souřadnicích $\frac{3}{11}$, $\frac{5}{11}$, $\frac{7}{11}$ a $\frac{8}{11}$. Tato syntaxe je výhodná, když víme, že v ukázce je zdrojový kód, který má ekvidistantní řádky, v tomto případě 11.
- `\dily11 \ud \ram{obrázek}` umožní stránkový zlom na stejných souřadnicích jako `\dily11 \ram{3,4,5,6,7,8}{obrázek}`. Jinými slovy místa řezů nastaví rovnoměrně,[2] přičemž vynechá řezy u horního a dolního okraje obrázku.
- `\Ram{}{obrázek}{strana}` u vícestránkových obrázků vloží do rámečku danou stranu obrázku.
- Podobně se pro makro `\Ram` použijí i zbývající výše uvedené syntaxe.
- `\setbox\obrbox\vbox{...} \ram{1,23,4567,89}{}` nevloží do rámečku obrázek, ale obsah boxu `\obrbox`.[3] V tomto případě se samozřejmě název souboru nezadává, i tak ale složené závorky ohraničující druhý argument makra `\ram` musejí zůstat.
  Pokud bychom chtěli mít vnitřek rámečku s `\obrbox`em stejně široký jako u rámečků s obrázkem, musíme si zajistit, že uvnitř `\vbox`u bude správně nastavena šířka sazby, například pomocí následujících maker:

```
\def\nastavobrbox{\setbox\obrbox\vbox\bgroup
  \hsize\dimexpr\hsize-2\hodstup\relax}
```

[2] Název makra `\ud` vznikl ze zkratky „u.d.“ = „uniformly distributed“.

[3] Aby mohl uživatel do rámečku vložit také verbatim výpisy nebo jiný delší text, je praktičtější i přehlednější si tento obsah rámečku předem připravit do boxu a ten pak použít.

```
\def\endnastavobrbox{\egroup}
```

a použitím následující konstrukce:

```
\nastavobrbox
\begin{verbatim}
...
\end{verbatim}
\endnastavobrbox
\dily11 \ud \ram{}
```

- Také v případě `\setbox\obrbox` je možné použít všechny výše uvedené syntaxe.

Pokud bychom kolem obrázku nechtěli kreslit rámeček, stačí předem provést následující nastavení:

```
\hodstup=0pt
\vodstup=0pt
\tloustkaramu=0pt
```

Makra jsou definována a odladěna v OPmacu. Přitom byl soubor s makry beze změny použit v předchozím článku tohoto Zpravodaje, který je vysázen v LaTeXu. Soubor se jmenuje rezani.tex a je ke stažení online [1].

4 Implementace

Nejdříve nadefinujeme některá OPmacová makra pro případ, že používáme LaTeX.

```
\unless\ifdefined\OPmac
  \csname newcount\endcsname\tmpnum
  \def\sdef#1{\expandafter\def\csname#1\endcsname}
  \def\sxdef#1{\expandafter\xdef\csname#1\endcsname}
  {\lccode`\?=`\p \lccode`\!=`\t
    \lowercase{\gdef\ignorept#1?!{#1}}}
  \def\localcolor{}
  \def\Grey{\color[gray]{0.5}}
\fi
```

Definujeme základní parametry, které budou postupně vysvětleny dále v textu.

```
\newdimen\hodstup \hodstup=3mm
\newdimen\vodstup \vodstup=2mm
\newdimen\tloustkaramu \tloustkaramu=0.4pt
\newskip\okoliramu \okoliramu\medskipamount
\def\barvaramu{\localcolor\Grey}
```

Přepínač `\ifladeni` určí, zda se v rámečcích mají zobrazovat místa řezů.

```
\newif\ifladeni
```

Makro `\svisleokraje` vycentruje svůj argument do `\hboxu` šířky `\hsize` a kolem něj vysází svislé linky.

```
\def\svisleokraje#1{\hbox to\hsize{\svisla\hss#1\hss\svisla}%
  \nointerlineskip}
\def\svisla{{\barvaramu\vrule width\tloustkaramu}}
\def\vodorovna{{\barvaramu\hrule height\tloustkaramu}}
\def\svislekousky{\svisleokraje{\vbox to\vodstup{\vss}}}
```

Makro `\eitd` provede aritmetický výpočet, jehož výsledkem je délková hodnota v jednotkách `pt`, a na úrovni expand procesoru tuto hodnotu bez jednotky `pt` vrátí.

```
\def\eitd#1{\expandafter\ignorept\the\dimexpr#1\relax}
```

Makro `\nactirezy` načte seznam souřadnic oddělených čárkou a jednotlivá čísla uloží do maker `\csname rez:`n`\endcsname`. Formát souřadnic je následující:

- Jestliže je `\dily` $= 0$, pak jsou hodnoty dány jako zlomkové části čísel z intervalu $(0, 1)$, a to bez uvedení desetinné tečky.
- Jestliže je `\dily` > 0, pak jsou hodnoty dány jako čitatelé zlomků, jejichž jmenovatel je roven `\dily`.

```
\def\nactirezy#1,#2\konec{%
  \advance\pocetrezu1
  \ifnum\dily>0
    \sxdef{rez:\the\pocetrezu}{\eitd{#1pt/\dily}}%
  \else
    \sxdef{rez:\the\pocetrezu}{0.#1}%
  \fi
  \ifx:#2:%
    \sdef{rez:\the\numexpr\pocetrezu+1}{1}%
    \let\next\relax
  \else
    \def\next{\nactirezy#2\konec}%
  \fi\next}
```

Makro `\ram{seznam}{obrázek}` vysází rámeček s daným obrázkem. Rámeček je na celou šířku `\hsize`. Horizontální a vertikální odstup rámečku od obrázku jsou dány registry `\hodstup` a `\vodstup`. V seznamu jsou čárkou oddělené hodnoty souřadnic, v nichž je možné provést stránkový zlom.

```
\def\ram#1#2{\ramA{#1}{}{#2}}
```

Makro `\Ram{seznam}{obrázek}{strana}` funguje zcela stejně, přičemž se do rámečku vloží daná strana obrázku.

```
\def\Ram#1#2#3{\ramA{#1}{page#3}{#2}}
```

Makro `\ramA{seznam}{parametr}{obrázek}` je společným jádrem maker `\ram` a `\Ram`. V registrech `\vyskaobrazkuBP` a `\sirkaobrazkuBP` je uvedena hodnota příslušného rozměru obrázku v jednotkách `bp` pronásobená jednotkou `pt`. V makru se obrázek načte do registru `\obrbox`, pokud je tento registr prázdný. Pokud je neprázdný, pouze se zajistí, že `\obrbox` je `\hbox`. Pak se v cyklu spočítají hodnoty potřebné uvnitř příkazů jazyka pdf, které obrázek ořežou a posunou na správné místo. V jednotlivých místech řezů je umožněn stránkový zlom s penaltou 100.

```
\newcount\pocetrezu
\newbox\obrbox
\newdimen\vyskaobrazkuBP
\newdimen\sirkaobrazkuBP
\newcount\dily
\def\ramA#1#2#3{\relax
  \pocetrezu=0
  \ifx@#1@%
    \nactirezy0,\konec
  \else
    \nactirezy0,#1,\konec
  \fi
  %\vypisrezy
  \dily=0
  \ifvoid\obrbox
    \pdfximage width \dimexpr\hsize-2\hodstup\relax #2{#3}%
    \setbox\obrbox\hbox{\pdfrefximage\pdflastximage}%
  \else
    \ifvbox\obrbox
      \setbox\obrbox\hbox{\box\obrbox}%
    \fi
  \fi
  \vyskaobrazkuBP=0.996264\ht\obrbox
  \sirkaobrazkuBP=0.996264\wd\obrbox
  \vskip\okoliramu
  \vodorovna
  \svislekousky
  \tmpnum=0
  \ht\obrbox=0pt \dp\obrbox=0pt \wd\obrbox=0pt
```

```
\loop
  \ifnum\tmpnum<\pocetrezu \advance\tmpnum1
  \edef\ramXB{\eitd{\sirkaobrazkuBP} }%
  \edef\ramYA{\eitd{\vyskaobrazkuBP-
    \csname rez:\the\numexpr\tmpnum+1\endcsname
      \vyskaobrazkuBP} }%
  \edef\ramYB{\eitd{\vyskaobrazkuBP-
    \csname rez:\the\tmpnum\endcsname
      \vyskaobrazkuBP} }%
  \svisleokraje{\hbox{%
    \vrule height \dimexpr\ramYB bp-\ramYA bp\relax width0pt%
    \pdfliteral{q 1 0 0 1 0 -\ramYA cm q
      0 \ramYA m \ramXB \ramYA l \ramXB \ramYB l 0 \ramYB l
      h W n}%
    \copy\obrbox
    \pdfliteral{Q Q}%
    \hskip\ramXB bp}}
  \ifnum\tmpnum<\pocetrezu
    \ifladeni \nobreak \vskip-\tloustkaramu \vodorovna \fi
    \penalty100
    \ifladeni \vskip-\tloustkaramu \vodorovna \fi
  \fi
\repeat
\svislekousky
\vodorovna
\vskip\okoliramu
{\setbox0\hbox{\box\obrbox}}}
```

Makro `\ud#1` se expanduje na `#1{seznam}`, kde v závorce je seznam přirozených čísel od (`\udpreskoc` + 1) do (`\dily` − `\udpreskoc` − 1).

```
\newcount\udpreskoc \udpreskoc2
\def\udbezcarky#1,\udbc{#1}
\def\ud#1{\def\poud{#1}%
  \def\seznamud{}%
  \tmpnum\udpreskoc
  \loop
    \ifnum\tmpnum<\numexpr\dily-\udpreskoc-1\relax
    \advance\tmpnum1
    \edef\seznamud{\seznamud\the\tmpnum,}%
  \repeat
  \expandafter\expandafter\expandafter\poud
    \expandafter\expandafter\expandafter{%
```

```
    \expandafter\udbezcarky\seznamud\udbc}}
```

Makro `\vypisrezy` vypíše seznam jednotlivých řezů. Toto makro je použito pouze pro účely ladění.

```
\def\vypisrezy{{\tmpnum=0
  \loop
    \ifnum\tmpnum<\pocetrezu \advance\tmpnum1
    \csname rez:\the\tmpnum\endcsname,
  \repeat
  \csname rez:\the\numexpr\pocetrezu+1\endcsname}}
```

Odkazy

1. ŠUSTEK, Jan. rezani.tex [online]. 2023-11-07. [cit. 2023-11-13]. Dostupné z: https://github.com/jsustek/rezani/blob/main/rezani.tex.

Summary: How to Enable Page Breaks in Embedded Images

The paper defines and describes TEX macros for inserting objects which are so high that the page breaking is difficult. These objects can be images, text examples or generally a content of a box. The macros insert the object on the current position and they allow page break in the middle of the object.

Keywords: frames, images, cutting, page break

Jan Šustek, jan.sustek@seznam.cz

Markdown 3: Co je nového a co se chystá?

Vít Starý Novotný

TEXový balíček Markdown poskytuje posledních osm let rozšiřitelný a formátově agnostický značkovací jazyk. V článku představuji třetí verzi balíčku Markdown a změny, které přináší ve srovnání s verzí 2.10.0. Zaměřuji se na tři hlavní skupiny uživatelů balíčku Markdown. Spisovatelé se dozvědí o nových prvcích, které mohou používat ve svých dokumentech, TEXperti se naučí určovat styl prvků jazyka markdown v různých formátech TEXu a vývojáři nahlédnou pod pokličku správy a vývoje balíčku Markdown a naučí se přidávat podporu pro nové prvky jazyka markdown. Článek je překlad mé přednášky na TUGu 2023 [1; 2].

Klíčová slova: Markdown, CommonMark, Lua, TEX, LATEX, OpTEX, Pandoc

1 Úvod

Děkuji a dobré ráno všem! Jmenuji se Vít Starý Novotný a přijel jsem sem ze sousední České republiky oslavit s vámi třetí hlavní verzi balíčku Markdown pro TEX. Kdykoliv mě někdo požádá, abych mu vysvětlil balíček Markdown, přijde mi užitečné zaměřit se na různé cílové skupiny, pro které je balíček určen.

1.1 Motivace pro spisovatele

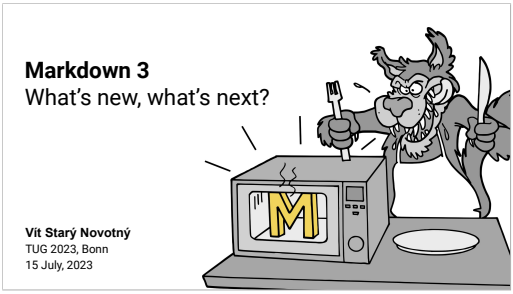
První a nejdůležitější skupinou jsou autoři.

Během svého bakalářského, magisterského a doktorského studia informatiky jsem používal LATEX pro všechno možné od psaní poznámek a domácích úkolů, prezentací až po psaní krátkých knih, článků v časopisech a svých závěrečných prací. Avšak pouze nejjednodušší texty jsou napsány naráz. U většiny textů je třeba první verzi upravit, znovu si ji přečíst, přemyšlet o ní, přeorganizovat ji, a to často několikrát, než je připravena pro veřejnost.

Během let jsem zjistil a jiní uživatelé mi také potvrdili [3], že LATEX je sice relativně snadný pro zápis, ale je těžko čitelný a obtížně se o něm přemýšlí. V akademickém světě se s tímto tématem pojí určitá míra „siláctví“ ze strany autorů, kteří jsou hrdí na svou schopnost číst LATEX. Avšak podle mého se hodně úsilí promarní na zbytečnou kognitivní zátěž způsobenou čtením a úpravami kódu místo toho, aby se autoři soustředili na to, co by mělo být jejich hlavním cílem: samotný text dokumentu.

Během svého magisterského studia jsem začal učit univerzitní kurz o digitální produkci dokumentů, což je široké a fascinující téma, jak asi všichni souhlasíme,

DOI: 10.5300/2023-3-4/111



Obrázek: Přednáškové slajdy (vlevo) [1] a videozáznam přednášky (vpravo) [2].

a zamiloval jsem se do odlehčených značkovacích jazyků, které používají interpunkci k označení struktury a které jsou snadno čitelné i zapisovatelné. Markdown byl významným odlehčeným značkovacím jazykem, široce používaným pro blogování, v „kecálcích“, na online diskusních fórech a pro dokumentaci softwaru. Díky svému zájmu o markdown, stejně jako svým dalším zájmům o LuaTEX nebo návrh kompilátorů a díky výjimečnému množství volného času jsem se rozhodl podat své univerzitě návrh projektu, který by umožnil autorům používat markdown místo LATEXu.

Projekt byl schválen a dostal jsem grant, který mi umožnil pracovat na balíčku Markdown pro TEX během studia. Od té doby uplynulo osm let a balíček Markdown dozrál do bodu, kdy autoři mohou s důvěrou používat markdown k psaní složitých textů obsahujících tabulky, obrázky, seznamy, poznámky pod čarou a další prvky, a vytvářet tak krásné dokumenty pomocí TEXu.

V této prezentaci se autoři dozvědí o nových prvcích jazyka markdown, které mohou používat ve svých dokumentech.

1.2 Motivace pro TEXperty

Dokonce i v LATEXu, kde mnoho autorů přepíná mezi dvěma rolemi autora a programátora, často dochází k dělbě práce, kdy vydavatel najme TEXperta, který připraví šablonu podle designové specifikace, a tuto šablonu pak autoři používají k vytváření dokumentů pro vydavatele.

Pro TEXperta je balíček Markdown pro TEX důležitým nástrojem, který zvyšuje jeho hodnotu na trhu práce. Je to proto, že balíček Markdown byl navržen jako bílá skříňka, která umožňuje definovat všechny aspekty markdownu pomocí TEXových maker až po jednotlivé prvky, jako jsou nadpisy, hypertextové odkazy a poznámky pod čarou [4]. Tím pádem může expert vytvořit šablonu a poté přidat další kód, který zajistí, že z markdownu vznikne výstup, který je v souladu se šablonou.

Balíček Markdown pro TEX navíc podporuje nejen LATEX, ale také plain TEX, ConTEXt a další formáty. To dále zvyšuje význam balíčku Markdown pro TEX jako společného rozhraní, podobně jako je tomu u TikZu, explu a dalších rozhraní, která jsou k dispozici v různých formátech TEXu.

V této prezentaci se TEXperti dozvědí, jak mohou stylovat prvky jazyka markdown v různých formátech TEXu.

1.3 Motivace pro softwarové vývojáře

Kromě autorů a TEXpertů jsou zde také vývojáři softwaru, kteří přispívají do svobodných softwarových projektů, jako je balíček Markdown, aby se vzdělávali a zkoušeli sami sebe, sbírali odměny za nevyřešené problémy a účastnili se správy a údržby projektů.

Pro vývojáře představuje balíček Markdown pro TEX jedinečnou výzvu. Heterogenní a vrstvený kód ve více než šesti různých programovacích jazycích obsahuje lekce ze systémové architektury, návrhu kompilátorů a digitální typografie s několika rozhraními a s vazbami na externí nástroje, jako jsou Pandoc, Graphviz a další. Vývojáři se buďto mohou soustředit na jeden z mnoha různých modulů, které odpovídají jejich kvalifikaci, nebo mohou kombinovat své dovednosti a čelit složitějším výzvám, které ovlivňují větší část heterogenního celku, jímž je balíček Markdown.

V této prezentaci se vývojáři dozvědí o implementačních detailech balíčku Markdown a jeho správě. Dále se také dozvědí, jak mohou vyvíjet syntaktická rozšíření, která přidávají podporu pro nové prvky jazyka markdown požadované autory a vydavateli [5, sekce 2.2].

1.4 Obsah

V první části své prezentace se zaměřím na nové funkce balíčku Markdown, které byly vyvinuty v posledních dvou letech. Mezi ně patří: 1) implementace standardu CommonMark, která usnadní export dokumentů napsaných pro balíček Markdown do různých procesorů markdownu, 2) prvky jazyka markdown podporované balíčkem Markdown, 3) externí vazby a nové formáty TEXu, které nyní podporujeme, 4) nové balíčky pro TEX a jazyk Lua, které byly odděleny od balíčku Markdown, protože jsou obecně užitečné i mimo balíček Markdown. První část uzavřu informacemi o správě balíčku Markdown.

Ve druhé části své prezentace se zaměřím na budoucnost balíčku Markdown pro TEX. Nejprve představím plán směřující k stabilní verzi 3.0 balíčku Markdown v následujících měsících a poté posluchače seznámím s plánovanými funkcemi a správou balíčku po verzi 3.0.

2 Co je nového?

Nejprve se podívejme na některé z nových funkcí balíčku Markdown, které můžete používat již dnes.

2.1 Podpora standardu CommonMark

Jazyk markdown byl vyvinut v roce 2004 Johnem Gruberem jako program v Perlu, který převádí text v jazyce markdown na webové stránky v jazyce HTML. Tento program obsahoval textový popis jazyka markdown.

Když jazyk nabral na popularitě, vznikly další implementace včetně balíčku Markdown pro TEX, které se chovaly odlišně v některých specifických případech, jež nebyly pokryty Gruberovým popisem jazyka markdown a které dávaly matoucí výsledky v původní implementaci v Perlu.

Aby se předešlo další fragmentaci a aby se zlepšila interoperabilita mezi nástroji, standardizační snaha vedená Jeffem Atwoodem a Johnem MacFarlanem vedla ke vzniku standardu CommonMark, který jednoznačně definuje jazyk markdown.

Ve své bakalářské práci můj student Andrej Genčur z Masarykovy univerzity v České republice implementoval standard CommonMark do balíčku Markdown [6]. Implementace je k dispozici v alfa verzi 3.0 balíčku Markdown, kterou si můžete stáhnout v digitálním repozitáři balíčku Markdown; zatím ne na CTANu, o tom řeknu více později.

Tato změna umožňuje autorům vytvářet text v markdownu, který lze snadno používat napříč různými implementacemi CommonMarku nebo ho převádět na jiné dialekty markdownu a na další značkovací jazyky, které může požadovat vydavatel.

2.2 Nová syntaktická rozšíření

Ačkoli CommonMark poskytuje silný základ pro psaní dokumentů v markdownu, mnohé užitečné prvky, jako jsou tabulky, poznámky pod čarou a citace, v něm nejsou zahrnuty. Proto balíček Markdown pro TEX poskytuje syntaktická rozšíření, která autorům umožňují tyto prvky zapisovat.

2.2.1 Metadata

Zatímco markdown lze použít k napsání hlavního textu dokumentu, není vhodný pro označení strukturovaných metadat, jako je název, jména autorů nebo abstrakt dokumentu.

YAML je formát pro serializaci dat, který je nadmnožinou JSON a jenž poskytuje způsob, jak označit metadata dokumentu, který je snadno čitelný a zároveň

snadno zapisovatelný. Několik implementací markdownu včetně balíčku Markdown podporuje propojení markdownu s YAMLem, což umožňuje autorům kombinovat metadata dokumentu a text dokumentu v jednom zdrojovém souboru.

TEXperti mohou definovat význam různých prvků YAMLu, jako je název, autoři dokumentu a rok vydání [7, sekce 2.1; 5, sekce 2.1]. To umožňuje vydavatelům specifikovat, jaká metadata by měla být autory poskytnuta a jak by měla být formátována.

2.2.2 Atributy

V těžkopádnějších značkovacích jazycích, jako je XML, mohou autoři připojit k prvkům atributy s doplňujícími informacemi, jako jsou například identifikátory sekcí pro křížové odkazy, třídy, které poskytují různé formátování pro stejný XML prvek, nebo hodnoty, které specifikují rozměry obrázku.

Několik implementací markdownu včetně balíčku Markdown umožňuje autorům připojit k prvkům jazyka markdown atributy. TEXperti mohou opět definovat formátování různých atributů na základě požadavků vydavatele a autorů [8].

2.2.3 Hybridní značkování

V komplexních dokumentech autoři často napíšou 90 % textu v markdownu, ale pro zbývajících 10 % potřebují použít TEX, například pro složité tabulky, obrázky a matematiku. Proto balíček Markdown vždy umožňoval autorům ukončit blok textu v jazyce markdown a v libovolném bodě se přepnout do TEXu.

Aby bylo možné upravit dělení slov nebo rozestupy a zapisovat TEXová makra uvnitř odstavců, balíček Markdown vždy poskytoval možnost zapnout tzv. *hybridní režim*, který umožňuje autorům volně kombinovat TEX s markdownem. Zapnutí hybridního režimu ale mění jazyk markdown, což činí dokumenty méně interoperabilními. Hybridní režim je ale natolik užitečný, že je ve většině šablon zapnutý, což považujeme za problém. Abychom autory odradili od používání hybridního režimu, připravili jsme tři syntaktická rozšíření, která neovlivňují interoperabilitu.

Autoři mohou označit tzv. *surové bloky* a také *surové úryvky*, které jsou interpretovány jako kód v TEXu. Další možností je umístit velké bloky kódu v TEXu do samostatného souboru a zahrnout je do dokumentu v jazyce markdown jako externí zdroj. [9] A pro matematiku mohou autoři také povolit syntaktické rozšíření, které jim umožní volně psát TEXové příkazy a další znaky mezi dolarovými znaménky a mezi znaky s lomítkem [10, sekce 2.3.1.38].

2.2.4 Další syntaktická rozšíření

Několik implementací markdownu včetně balíčku Markdown také poskytuje podporu pro další syntaktická rozšíření, která umožňují autorům psát následující

prvky v jejich dokumentech: úkoly, *důrazná zalomení řádků* (anglicky *emphatic line breaks*), horní a dolní indexy a přeškrtnutí [7, sekce 1; 5, sekce 1]. Pro každý z těchto prvků mohou TEXperti definovat formátování na základě požadavků vydavatele.

2.3 Nové formáty TEXu

Aby byl balíček Markdown užitečnější, byla nedávno přidána podpora pro nové formáty, nejen plain TEX, LATEX a ConTEXt.

Nové formáty podporované balíčkem Markdown jsou TEX4ht, který autoři mohou použít k vytváření webových stránek místo PDF dokumentů pro tisk, a také OpTEX, což je minimalistický formát založený na plain TEXu. OpTEX TEXpertům poskytuje mnoho funkcí větších formátů, jako je vestavěný výběr písma, podpora bibliografie, zvýraznění syntaxe v kódu, hypertextové odkazy a další.

Pokud chcete nové formáty vyzkoušet, existují vzorové soubory, které si můžete stáhnout a vysázet [11].

2.4 Vazba na konverzní program Pandoc

Už několikrát jsem zmínil nástroj Pandoc. Pandoc je nástroj pro konverzi dokumentů, který umožňuje převod mezi desítkami formátů dokumentů včetně různých formátů TEX. Pandoc ale produkuje surový kód pro TEX, což komplikuje změnu vzhledu jednotlivých prvků textu bez ručních úprav výstupu Pandocu.

Ve své bakalářské práci můj student Dominik Rehák vyvinul propojení mezi balíčkem Markdown a nástrojem Pandoc [12]. Toto propojení umožňuje autorům psát jejich dokumenty v textovém procesoru, jako je Microsoft Word, a poté vysázet své dokumenty prostřednictvím balíčku Markdown, jako by byly napsány v markdownu, což umožňuje využití existujících šablon pro balíček Markdown.

2.5 Dceřiné softwarové balíčky Markdownu

Balíček Markdown implementuje mnoho užitečných funkcí a některé jsou dostatečně obecné na to, aby mohly být od balíčku odděleny a poskytnuty uživatelům jako balíčky samostatně.

Jak jsem již zmínil, balíček Markdown umožňuje autorům psát metadata dokumentu pomocí jazyka YAML. Dříve byla podpora YAMLu poskytována externí knihovnou tinyyaml v jazyce Lua, která byla distribuována jako součást balíčku Markdown. Díky přispěvateli jménem Zeping Lee je nyní tinyyaml dostupná jako nezávislá knihovna na CTANu [13].

Za druhé, ačkoli balíček Markdown používá pod kapotou jazyk Lua, funguje bezproblémově s TEXovými stroji, jako jsou pdfTEX a XƎTEX. Schopnost provádět kód v jazyce Lua s jakýmkoli TEXovým strojem je nyní poskytována jako součást balíčku lt3luabridge psaného v jazyce expl3, který je také dostupný na CTANu [14].

2.6 Komunikace a správa projektu

Před osmi lety, když jsem začal s vývojem balíčku Markdown, rozhodoval jsem o veškerém jeho dalším směřování. Dnes se na vývoji podílí řada autorů, TEXpertů a vývojářů.

Dříve veškerá komunikace probíhala na GitHubu, který je sice vhodný pro vývojáře, ale je již méně vstřícný k autorům. Proto jsme nedávno přidali místnosti na Discordu a Matrix.org [15; 16], což jsou diskusní sociální sítě, kde mohou autoři, TEXperti a vývojáři diskutovat o svých problémech a o budoucnosti balíčku Markdown.

3 Co se chystá?

Nyní se krátce podíváme na některé funkce balíčku Markdown, které plánujeme v blízké budoucnosti.

3.1 Stabilní verze 3.0.0

Koncem června jsme vydali první veřejnou alfa verzi Markdownu 3.0, kterou si můžete stáhnout na GitHubu. Stabilní vydání Markdownu 3.0 plánujeme na září[1] a vydání na CTANu plánujeme před vydáním TEX Live 2024, aby všechny verze TEX Live 2023, bez ohledu na to, kdy jste je nainstalovali, stále obsahovaly Markdown 2. To považujeme za odpovědné a rozumné rozhodnutí, protože změny mezi Markdownem 2 a 3 jsou značné. O tom si řekneme více později.

3.1.1 Otevřené problémy

Prvním důvodem, proč je aktuální vydání Markdownu 3.0 označeno jako alfa verze, je několik otevřených problémů.

Konkrétně: implementace CommonMarku způsobila úniky paměti [18]. Tyto úniky byly narychlo opraveny v alfa verzi, ale před stabilním vydáním bude potřeba řádná oprava.

Implementace CommonMarku také šestinásobně zvýšila počet jednotkových testů [19], což způsobilo problémy s výkonností naší služby pro průběžnou integraci. V alfa verzi jsme dočasně snížili počet testovaných TEXových strojů, což mohlo způsobit, že jsme přehlédli některé regrese v dalších TEXových strojích. Proto musíme jednotkové testy urychlit, obnovit plné testování a zjistit, zda byly v mezičase zavedeny nějaké regrese, a až poté vydat stabilní verzi 3.0.

Dalším problémem je implementace atributů v balíčku Markdown, která je mírně nekompatibilní s jinými implementacemi, jako je Pandoc [20], což opravíme v příštích dvou měsících ještě před stabilním vydáním.

[1] Verze 3.0.0 balíčku Markdown vyšla již 25. srpna 2023 [17] a zahrnuje všechny opravy a změny, které popisuji v sekcích 3.1.1 a 3.1.2.

3.1.2 Nová syntaktická rozšíření

Kromě otevřených problémů je dalším důvodem, proč je aktuální vydání Markdownu 3.0 označené jako alfa verze, také množství plánovaných funkcí, které ještě nebyly implementovány.

První z těchto funkcí je syntaktické rozšíření, které autorům umožní psát atributy nejen vedle obrázků, ale také vedle tabulek [21]. To považujeme za velmi užitečné pro autory, kteří potřebují na své tabulky odkazovat z textu.

Další funkcí jsou tzv. *bohaté TEXové odstavce* [22]. Jak si možná vzpomenete, jazyk markdown byl původně vyvinut jako preprocesor pro psaní webových stránek v HTML a v HTML blokové prvky, jako jsou seznamy, vždy ukončují odstavec. Typograf však může seznam použít jako způsob, jak vysázet výčet uprostřed věty, a takový seznam by neměl odstavec ukončit. Ve většině TEXových formátů, jako je LATEX, můžete takový seznam napsat vynecháním prázdného řádku za seznamem. Naše syntaktické rozšíření umožní autorům udělat totéž v markdownu. Vynecháním prázdného řádku mohou autoři naznačit, že konec blokového prvku, jako je seznam, by neměl ukončit aktuální odstavec, podobně jako je to možné v TEXu.

3.1.3 Průvodce migrací z Markdownu 2

Pokud se chystáte vyzkoušet alfa verzi Markdownu 3, což doporučujeme, a pokud jste dlouhodobými uživateli Markdownu 2, měli byste vědět, že některé zastaralé funkce byly v Markdownu 3 odstraněny.

Konkrétně jsme odstranili volbu `hardLineBreaks`, která dříve způsobila, že všechna zalomení řádků v markdownovém zdrojovém textu vytvořila zalomení řádku ve výstupu, což je užitečné například při sazbě poezie. V Markdownu 3 však všechna zalomení řádků v markdown textu vytvářejí TEXové makro měkkého zalomení, které ve výchozím nastavení nevytváří žádný viditelný výstup, ale může být předefinováno TEXpertem tak, aby na výstupu vytvořilo například zalomení řádku nebo něco docela jiného.

Dále jsme přejmenovali několik prvků tak, aby názvy lépe označovaly sémantiku prvků a nezávisely na tom, jak jsou prvky běžně formátovány. Mezi zasažené prvky patří například poznámky pod čarou, které byly přejmenovány na poznámky, a vodorovné čáry, jež byly přejmenovány na tematické zlomy.

Pokud máte existující šablonu pro Markdown 2, která používá některé ze zastaralých funkcí, můžete buďto použít TEX Live 2023 s nezměněnou šablonou, nebo můžete distribuovat Markdown 2 společně s nezměněnou šablonou a používat TEX Live 2024 a jeho novější verze, nebo si můžete najmout TEXperta, který vám s ochotou aktualizuje vaši šablonu z Markdownu 2 na Markdown 3.

3.2 Budoucí rozvoj

Po stabilním vydání Markdownu 3 máme v plánu ještě několik dalších funkcí.

3.2.1 Implicitní identifikátory sekcí

Autoři se mohou těšit na nová syntaktická rozšíření, která jim umožní automaticky generovat identifikátory sekcí pro křížové odkazy [23; 24].[2]

3.2.2 Řízení horizontálního členění textu

V současnosti mohou autoři vždy psát více odstavců textu a používat blokové prvky, jako jsou seznamy a obrázky. V některých situacích to však nemusí být žádoucí. TEXpert může například vytvořit šablonu, která použije markdown dokument poskytnutý autorem k vygenerování hlavičky a patičky dokumentu; zde je žádoucí, aby autor mohl psát pouze řádkové prvky.

Proto plánujeme možnost ovládat povolenou úroveň obsahu v markdownu:

```
\markdownInput[contentLevel = inline]{obsah-hlavičky-dokumentu.md}
```

Dále plánujeme přidat TEXový příkaz `\markinline`, který umožní autorům, kteří stále upřednostňují TEX, používat řádkové prvky jazyka markdown uprostřed TEXového odstavce, aby se seznámili se syntaxí markdownu a možná dokonce zlepšili čitelnost svých dokumentů [26]:

```
TEXový odstavec s \markinline|[hypertextovým odkazem](http://...)|.
```

3.2.3 Vestavěná podpora formátu OpTEX

Jak jsem již zmínil, balíček Markdown již podporuje formát OpTEX. Tato podpora je ale v současnosti obsažena pouze v ukázkovém dokumentu spolu s textem ukázkového dokumentu. V budoucnu bychom chtěli podporu OpTEXu přesunout do samostatného modulu, který by autoři a TEXperti mohli jednoduše importovat do svých dokumentů [27].

3.2.4 Vestavěná podpora generického TEXového výstupu v Pandocu

Také jsem zmínil, že můj student Dominik Rehák vyvinul propojení mezi balíčkem Markdown a nástrojem Pandoc, což umožňuje sázet dokumenty v různých formátech prostřednictvím balíčku Markdown. Tato implementace je však pouze ověřením koncepce v jazyce Lua. V budoucnu bychom chtěli přidat plnohodnotnou implementaci v jazyce Haskell přímo do kódu nástroje Pandoc, což jsem již předběžně probral s hlavním vývojářem Pandocu, Johnem MacFarlanem a dalšími.

3.2.5 Víceformátová témata

Ačkoli je nejlepší používat šablonu vyvinutou TEXpertem, balíček Markdown poskytuje výchozí formátovací definice pro většinu prvků, které autoři mohou použít jako výchozí bod. Tyto výchozí definice jsou však často nedostačující a také zaplevelují kód balíčku Markdown.

V současnosti mohou TEXperti používat koncept témat [28] při vývoji šablon pro balíček Markdown v LATEXu a v budoucnu plánujeme přidat podporu témat všem podporovaným formátům tak, aby jedno téma mohlo mít více formátovacích

[2] Implicitní identifikátory sekcí jsou součástí baličku Markdown od verze 3.2.0 z 21. října 2023 [25].

definic pro různé TEXové formáty [29]. S pomocí této nové technologie plánujeme vytvořit jedno výchozí téma, které by zahrnovalo všechny výchozí formátovací definice balíčku Markdown, a toto téma by pak bylo spravováno odděleně od balíčku Markdown. To by umožnilo serióznější implementaci než to, co máme nyní: nedokonalé zástupné definice vynucené nepřítomností šablony.

3.2.6 Mailing list a decentralizovaná správa

Jak jsem již zmínil, používáme nyní diskusní sociální sítě, na kterých hovoříme o budoucnosti balíčku Markdown. Mnoho diskusí týkajících se TEXu však probíhá na mailing listech. Proto bychom v budoucnu chtěli vytvořit mailing list pro balíček Markdown.

A pokud balíček Markdown a jeho uživatelská základna dále poroste, budeme muset prozkoumat některé možnosti pro decentralizovanější správu, aby rozhodování neprobíhalo jen na diskusních sociálních sítích a v mé hlavě, ale abychom měli systém hlasování, který by umožňoval rozhodovat o nových funkcích balíčku Markdown transparentnějším způsobem.

4 Závěr

Co říci závěrem? V této přednášce jsme si představili nové a plánované funkce balíčku Markdown pro TEX z pohledu tří zainteresovaných stran: autorů, TEXpertů a vývojářů. Slovy Petra Sojky a Johna Lennona bych chtěl říci: Prosím, vše, co říkáme, je „dejte markdownu šanci!“

Doufám, že jste přednášku shledali zajímavou, a rád zodpovím veškeré vaše otázky buď nyní, během přestávky na kávu, nebo v průběhu konference. Děkuji!

5 Dotazy publika

Moderátor: Děkuji. Plánujete umožnit konverzi ze souboru .md na soubor .tex?

Jaký druh souboru? Soubor .tex?

Moderátor: Napíšete soubor .md pro markdown a můžete jej konvertovat na tištěnou verzi pomocí TEXu. Můžete jej konvertovat na soubor .tex?

Ano, balíček Markdown poskytuje rozhraní pro příkazový řádek, které můžete použít k převodu souboru .md na mezireprezentaci pomocí TEXových maker a s tou pak můžete dále nakládat, jak uznáte za vhodné.

Moderátor: Dobře. Jsou zde nějaké otázky?

Michael Schlüter: Upřímně řečeno, jsem trochu skeptický. Opravdu potřebujete další značkovací jazyk? LATEX je již dobře definovaný jazyk, ale někteří lidé říkají: „Ó, to je pro mě příliš složité, potřebuji něco jednoduššího.“ Ale čím více funkcí přidáte do svého značkovacího jazyka, tím menší je tato výhoda. A jsou dvě mřížky

opravdu čitelnější než například příkaz pro sekci? A spousta dalších problémů. Čím více speciálních znaků definujete, tím více nepříjemných překvapení připravíte uživatelům, protože najednou má speciální znak speciální význam místo toho, aby byl převzat doslovně. Takže vidím spoustu problémů s dalším značkovacím jazykem, více problémů než výhod, což je spíše poznámka než otázka.

Dobře, nejsem si jistý, co přesně byla otázka. Odpovím na některé body, které jste zmínil.

V LaTeXu existuje koncept jazyka pro značkování dokumentů. LaTeX 2$_\varepsilon$ je jedním možným značkovacím jazykem, ale projekt LaTeX předpokládá, že bude existovat více značkovacích jazyků, které autoři mohou používat a které budou nezávislé na LaTeXovém backendu. Takže to všechno bude LaTeX, ale autoři mohou používat různé značkovací jazyky. Myslím si tedy, že myšlenka přidání dalších značkovacích jazyků je součástí étosu projektu LaTeX.

Také jste zmínil speciální znaky, které mohou způsobit problémy. Když sázíme text v jazyce markdown, přepneme kategorie znaků, takže by to neměl být problém. A kategorie přepneme zpět, kdykoliv se objeví surový blok, kde používáte svá vlastní TeXová makra, čili by to uživatelům nemělo působit žádné potíže.[3]

Moderátor: Ještě nějaká otázka?

Henri Menke: Ne ani tak otázka jako spíš komentář/nabídka. Máte kanál na Matrix.org, který rád zahrnu pod TUGový kanál na Matrix.org, stačí mě kontaktovat na konci přednáškového bloku.

Dobře, děkuji!

Moderátor: Ještě nějaká otázka? Ano.

Posluchač: Když přidáte podporu pro tabulky a YAML, jste velmi blízko dialektu GitHub-Flavored Markdown, který mnoho lidí zná z GitHub Pages a dalších míst. Budete mít výchozí nastavení a předpokládáte podporu pro další varianty markdownu kromě CommonMarku? GitHub Pages je velmi běžný.

Takže Vaše otázka je, zda bychom byli otevřeni podpoře jiných variant markdownu kromě CommonMarku.

Posluchač: Ano, myslím, že vaše podpora tabulek je velmi blízká rozšíření pro tabulky na GitHubu a GitHub také podporuje preambuli v jazyce YAML, takže jste, myslím, velmi blízko GitHubu.

Dobře, nejsem si jistý, že jsem zachytil celou otázku, ale rád bych odpověděl na bod týkající se CommonMarku: Ne v současné době. Naším plánem je podporovat CommonMark a pokud potřebujete dodávat dokumenty v jiném vstupním formátu včetně různých dialektů markdownu, musíte je nejprve převést do CommonMarku.[3]

[3] Obě otázky jsem detailněji zodpověděl v článku, který vyšel v časopise *TUGboat* [30].

Moderátor: Dobře.

Barbara Beeton: Připravil jste svou prezentaci Markdownem, nebo něčím jiným?

[smích]

Ne, přiznám se, že prezentaci jsem vytvořil v Google Slides.

[bučení a smích]

Moderátor: Dobře, po tomto přiznání...

[smích]

Moderátor: Můžeme znovu poděkovat našemu řečníkovi.

[potlesk]

Odkazy

1. STARÝ NOVOTNÝ, Vít. *Markdown 3: What's new, what's next?* [online]. TUG, 2023-07-15 [cit. 2023-09-15]. Dostupné z: `https://tug.org/tug2023/files/sa-03-novotny-markdown3/novotny-markdown3-slides.pdf`.
2. STARÝ NOVOTNÝ, Vít. *TUG 2023 – Markdown 3: Co je nového a co se chystá?* [online]. YouTube, 2023-07-15 [cit. 2023-09-15]. Dostupné z: `https://youtu.be/U8XjTOhJkg0`.
3. THOMPSON, Michael. *pandoc-discuss*. Re: Error in "cabal install pandoc" [online]. Google Groups [cit. 2022-10-04]. Dostupné z: `https://groups.google.com/g/pandoc-discuss/c/tKB4E7y6H2E/m/OiieKAuWsl4J`.
4. NOVOTNÝ, Vít. Sazba textu označkovaného v jazyce Markdown uvnitř TEXových dokumentů. *Zpravodaj CSTUGu*. 2016, **26**(1–4), 78–93.
5. NOVOTNÝ, Vít. Markdown 2.17.1: What's new, what's next? *TUGboat*. 2022, **43**(3), 276–278.
6. GENČUR, Andrej. *An implementation of the CommonMark standard into the Markdown package for TEX* [online]. 2023. [cit. 2023-09-15]. Dostupné z: `https://is.muni.cz/th/r7z7l/`. Bakalářská práce. Masarykova univerzita. Vedoucí práce Vít STARÝ NOVOTNÝ.
7. NOVOTNÝ, Vít; REHÁK, Dominik; HOFTICH, Michal; VRABCOVÁ, Tereza. Markdown 2.15.0: What's new? *TUGboat*. 2022, **43**(1), 10–15.
8. NOVOTNÝ, Vít. Attributes in Markdown. *TUGboat*. 2023, **44**(1), 99–101.
9. NOVOTNÝ, Vít. *Side-by-side images* [online]. GitHub, 2023-05-15 [cit. 2023-09-15]. Dostupné z: `https://github.com/Witiko/markdown/discussions/302#discussioncomment-5902455`.

10. NOVOTNÝ, Vít. *Markdown package user manual* [online]. CTAN, 2023-04-27 [cit. 2023-09-15]. Dostupné z: https://mirrors.ctan.org/macros/generic/markdown/markdown.html. Verze 2.23.0-0-g0b22f91.
11. STARÝ NOVOTNÝ, Vít. *markdown/examples at main* [online]. GitHub, 2023-09-15 [cit. 2023-09-15]. Dostupné z: https://github.com/Witiko/markdown/tree/main/examples.
12. REHÁK, Dominik. *Generic TEX writer for the Pandoc document converter* [online]. 2023. [cit. 2023-09-15]. Dostupné z: https://is.muni.cz/th/umhg5/. Bakalářská práce. Masarykova univerzita. Vedoucí práce Vít STARÝ NOVOTNÝ.
13. LEE, Zeping. *lua-tinyyaml: A tiny YAML (subset) parser for pure Lua* [online]. CTAN, 2023-04-05 [cit. 2023-09-15]. Dostupné z: https://ctan.org/pkg/lua-tinyyaml. Verze 0.4.3.
14. NOVOTNÝ, Vít. *lt3luabridge: Execute Lua code in any TEX engine that exposes the shell* [online]. CTAN, 2022-10-24 [cit. 2023-09-15]. Dostupné z: https://ctan.org/pkg/lt3luabridge. Verze 2.0.2.
15. NOVOTNÝ, Vít. *The Markdown package for TEX* [online]. Discord, 2022-08-23 [cit. 2023-09-15]. Dostupné z: https://discord.gg/8xJsPghzSH.
16. NOVOTNÝ, Vít. *The Markdown package for TEX* [online]. Matrix.org, 2022-08-23 [cit. 2023-09-15]. Dostupné z: https://matrix.to/#/#witiko-markdown:matrix.org.
17. NOVOTNÝ, Vít Starý; PRENTICE, Lloyd; PEISCHL, Marei; WILLIS, Didier; VRABCOVÁ, Tereza. *Markdown: Release 3.0.0* [online]. GitHub, 2023-08-25 [cit. 2023-09-27]. Dostupné z: https://github.com/Witiko/markdown/releases/tag/3.0.0.
18. STARÝ NOVOTNÝ, Vít. *Fix memory leak in CommonMark implementation* [online]. GitHub, 2023-06-26 [cit. 2023-09-15]. Dostupné z: https://github.com/Witiko/markdown/issues/308.
19. NOVOTNÝ, Vít. *Implement batching and summarization to unit tests* [online]. GitHub, 2023-01-09 [cit. 2023-09-15]. Dostupné z: https://github.com/Witiko/markdown/issues/245.
20. NOVOTNÝ, Vít. *Make our implementation of attributes compatible with jgm/pandoc* [online]. GitHub, 2023-05-15 [cit. 2023-09-15]. Dostupné z: https://github.com/Witiko/markdown/issues/304.
21. STARÝ NOVOTNÝ, Vít. *Add support for attributes on tables* [online]. GitHub, 2023-06-26 [cit. 2023-06-26]. Dostupné z: https://github.com/Witiko/markdown/issues/310.
22. NOVOTNÝ, Vít. *Add support for TEX-like rich paragraphs* [online]. GitHub, 2018-02-28 [cit. 2023-06-26]. Dostupné z: https://github.com/Witiko/markdown/issues/30.

23. NOVOTNÝ, Vít. *Add support for Pandoc `auto_identifiers` syntax extension* [online]. GitHub, 2022-12-29 [cit. 2023-06-26]. Dostupné z: `https://github.com/Witiko/markdown/issues/237`.
24. NOVOTNÝ, Vít. *Add support for Pandoc `gfm_auto_identifiers` syntax extension* [online]. GitHub, 2022-12-29 [cit. 2023-06-26]. Dostupné z: `https://github.com/Witiko/markdown/issues/238`.
25. NOVOTNÝ, Vít Starý; GENČUR, Andrej. *Markdown: Release 3.2.0* [online]. GitHub, 2023-10-21 [cit. 2023-10-21]. Dostupné z: `https://github.com/Witiko/markdown/releases/tag/3.2.0`.
26. NOVOTNÝ, Vít. *Add `contentLevel` Lua option and a `\markinline` plain TEX command* [online]. GitHub, 2023-04-28 [cit. 2023-06-26]. Dostupné z: `https://github.com/Witiko/markdown/issues/300`.
27. NOVOTNÝ, Vít. *Support OpTEX* [online]. GitHub, 2022-11-12 [cit. 2023-06-26]. Dostupné z: `https://github.com/Witiko/markdown/issues/215`.
28. NOVOTNÝ, Vít. Markdown 2.10.0: LaTeXová témata a snippety. *Zpravodaj CSTUGu*. 2021, **31**(1–4), 76–82.
29. NOVOTNÝ, Vít. *Add support for universal (cross-format) themes* [online]. GitHub, 2023-03-08 [cit. 2023-06-26]. Dostupné z: `https://github.com/Witiko/markdown/issues/276`.
30. STARÝ NOVOTNÝ, Vít. Markdown 3 at TUG 2023: Reflections from the Q&A session. *TUGboat*. 2023, **44**(3). Preprint dostupný z: `https://www.overleaf.com/read/mjghwhrbgmfj`.

Summary: Markdown 3: What's New, What's Next?

The Markdown package for TeX has provided an extensible and format-agnostic markup language for the past seven years. In this article, I present the third major release of the Markdown package and the changes it brings compared to version 2.10.0. In the article, I target the three major stakeholders of the Markdown package. Writers will learn about the new elements, which they can type in their Markdown documents, TeXperts will learn how they can style Markdown documents in different TeX formats, and developers will learn about the governance and the development of the Markdown package and how they can extend Markdown with new elements. The article is a Czech translation of my talk at TUG 2023 [1; 2].

Keywords: Markdown, CommonMark, Lua, TeX, LaTeX, OpTeX, Pandoc

Vít Starý Novotný, witiko@mail.muni.cz

A Roadmap for Universal Syllabic Segmentation

Ondřej Sojka, Petr Sojka, Jakub Máca

Space- and time-effective segmentation (word hyphenation) of natural languages remains at the core of every document rendering system, be it TEX, web browser, or mobile operating system. In most languages, segmentation mimicking syllabic pronunciation is a pragmatic preference today.

As language switching is often not marked in rendered texts, the typesetting engine needs *universal syllabic segmentation*. In this article, we show the feasibility of this idea by offering a prototype solution to two main problems:

A) Using Patgen to generate patterns for several languages at once; and

B) lack of Unicode support in tools like Patgen or TEX (patterns in UTF-16 encoding) is missing.

For A), we have applied it to generating universal syllabic patterns from wordlists of nine syllabic, as opposed to etymology-based, languages (namely, Czech, Slovak, Georgian, Greek, Polish, Russian, Turkish, Turkmen, and Ukrainian). For B), we have created a version of Patgen that uses the Judy array data structure and compared its effectiveness with the trie implementation.

With the data from these nine languages, we show that:

A) developing universal, up-to-date, high-coverage, and highly generalized universal syllabic segmentation patterns is possible, with a high impact on virtually all typesetting engines, including web page renderers; and

B) bringing wide character support into the hyphenation part of the TEX suite of programs is possible by using Judy arrays.

Keywords: syllabification, hyphenation, universal syllabic patterns preparation

1. Motivation

Justified alignment achieved with a quality hyphenation algorithm is both optically pleasing and saves time to read, in addition to saving trees. Only *quality hyphenation* allows interword spaces to be as uniform as possible, close to Gutenberg's ideal of spaces of fixed width. A high coverage, space- and time-effective hyphenation (segmentation) algorithm of all natural languages is badly needed[1]

This is an updated and enriched version of the paper [1] published in the TUG 2023 proceedings issue of the TUGboat journal.

[1] `https://bugzilla.mozilla.org/show_bug.cgi?id=672320`

DOI: 10.5300/2023-3-4/125

as it remains at the core of every document rendering system, be it TEX, web browsers supporting HTML with CSS 3, or an operating system providing text rendering for mobile applications.

In most languages, segmentation mimicking syllabic pronunciation is pragmatically preferred today. As language switching is often not marked in texts, and cannot be safely guessed from the words themselves, language-agnostic orthographic syllabification, is needed. We call this task *universal syllabic segmentation*, or in short, the syllabification problem.

The syllabification problem has been tackled by several finite state [2] or, more recently, machine learning techniques [3, 4, 5, 6]. Bartlett et al. [3] uses structured support vector machines (SVM) to solve syllabification as a tagging problem. Krantz et al. [7] leverage modern neural network techniques with long short-term memory (LSTM) cells, a convolutional component, and a conditional random field (CRF) output layer, and demonstrated cross-linguistic generalizability, syllabifying English, Dutch, Italian, French, Manipuri, and Basque datasets together.

From an orthographic viewpoint (hyphenation), universal language solutions today should reflect the Unicode standard [8]. Internal support for full (multibyte) Unicode, a must in today's operating systems and applications, *is missing* in the TEX family of programs, e.g. in Patgen and TEX itself. The internal processing is thus limited by the internal one-byte representation of language characters and is hardwired into the optimized code of these programs. Therefore, processing CJK (Chinese, Japanese, Korean) languages or sets of languages whose character representations need multibyte handling is close to impossible. Special "hacks" are needed for character and font encodings both on the input side (package `inputenc`) and output side (packages `fontenc` or `fontspec`) are not backed by internal wide character support.

Since both TEX and Patgen have hardwired 8-bit character representations, to develop practically useable universal syllabic hyphenation, one needs to overcome these constraints.

In this paper we a) constructively show the feasibility of preparation of universal syllabic patterns, b) demonstrate a version of Patgen with wide character support, and c) discuss further steps to do in the TEX program suite to make language hyphenation Unicode-compliant.

The paper is structured as follows. In Section 2 we define the terminology and describe the language data we have used in our experiments. Section 3 reminds the reader about the principles of the hyphenation algorithm in TEX and of Patgen-based pattern generation and pattern representation possibilities. Section 4 evaluates the experiments with universal pattern generation. In Section 5 we elaborate on possible routes towards wide character support in the typesetting engines and Patgen. As usual, we sum up and conclude in the final Section 6.

"The concept of the syllable is cross-linguistic, though formal definitions are rarely agreed upon, even within a language. In response, data-driven syllabification methods have been developed to learn from syllabified examples. ... Syllabification can be considered a sequence labeling task where each label delineates the existence or absence of a syllable boundary." [7]

2. Syllabification

Human beings convey meaning by pronouncing words as sequences of phonemes. Phonology studies the structure of phonemes we are able to pronounce as syllables [9]. Etymologically, a syllable is an Anglo-Norman variation of Old French sillabe, from Latin syllaba, from Greek (syllabē), "that which is held together; a syllable, several sounds or letters taken together" to make a single sound. [10]

When we delineate boundaries in the orthographic representation of words, we speak about *hyphenation* of words as sequences of *characters*.

2.1. Hyphenation as syllabification

There are subtle differences between syllabification and hyphenation, though. Let us take the Czech word *sestra*. The Czech language authorities [11] allow hyphenations as *se-s-t-ra*, while agreeing that there are only two syllables based on **C**onsonant and **V**owel sequencing: either *se-stra* (CV-CCCV), or *ses-tra* (CVC-CCV), or *sest-ra* (CVCC-CV). As with hyphenation, defining segments for syllabification is full of exceptions. The Czech sentence *Strč prst skrz krk* or word *scvrnkls* (CCCCCCCC) contain consonants-only syllables.

There are also rare cases where word segmentation should differ in different contexts. It may be necessary within one language (different hyphenation *re-cord* and *rec-ord* depending on its part of speech), or between different languages. When developing universal syllabic patterns, these theoretically possible segmentations should not be allowed in the input hyphenated wordlist used for training. But this should not matter, as e.g. Liang's `hyphen.tex` patterns do not cover more than 10% of positions [12] and few complain about this coverage.

2.2. Data preparation

To show the feasibility of universal pattern generation, we have collected wordlists for a dozen languages, as shown in Table 1. The chosen languages a) have a wide diversity in alphabets and syllables and b) have existing hyphenation patterns as an approximation for syllable segments. The wordlists were collected from public sources or provided for our research as stratified dictionaries from TenTen corpora [15] by Lexical Computing. We used wordlists sorted by frequency and cut at below 5% of word occurrences, to eliminate typos appearing in documents. Each tenth word was taken into a wordlist—a stratified sampling technique inspired by

Table 1: Language resources and patterns used in pattern development experiments. All data was converted to UTF-8 and contains lowercase alphabetic characters only. Alphabet size (# chars) counts characters appearing in the language wordlist collected. Languages were chosen for diversity of size of patterns and syllables.

Language	#words	#chars	#patterns	#syllables	source, alphabet
Czech+Slovak (cz+sk)	606,499	47	8,231	2,288,413	[13], Latin
Georgian (ka)	50,644	33	2,110	224,799	[14], Georgian
Greek (el-monoton)	10,432	48	1,208	37,736	[14], Greek
Panjabi (pa)	892	52	60	2,579	[14], Gurmukhi
Polish (pl)	20,490	34	4,053	65,510	[14], Latin
Russian (ru)	19,698	33	4,808	75,532	[14], Cyrillic
Tamil (ta)	46,526	48	71	209,380	[14], Tamil
Telugu (te)	28,849	66	72	125,508	[14], Telugu
Thai (th)	757	64	4,342	1,185	[14], Thai
Turkish (tr)	24,634	32	597	103,989	[14], Latin
Turkmen (tk)	9,262	30	2,371	33,080	[14], Latin
Ukrainian (ua)	17,007	33	1,990	65,099	[14], Cyrillic

Knuth [16] that was already used successfully in pattern generation [17]. Wordlists were hyphenated by legacy patterns, mostly taken from [14].

Alphabet analysis and statistics are shown in Table 2. The total number of letters appearing in all languages exceeds 245, the maximum number of letters that current Patgen can support. This is why wide-character representation (Unicode UTF-16) support in Patgen (and then in the hyphenator library in a typesetting engine) would be needed to extend our generation to more languages.

3. Pattern development

The idea of squeezing the hyphenated wordlist into the set of patterns was originated in the dissertation of Frank Liang [12], supervised by Donald Knuth. For the automated generation of patterns from a wordlist, Liang wrote the Patgen program. Patgen was one of the very first programs that harnessed the power of data with supervised machine learning. Programmed originally to support English and ASCII, it was later extended to be usable for 8-bit characters and for wordlists that contain at most 245 characters [18]. It is capable of efficient lossy or lossless *compression* of hyphenated dictionaries, with several orders of

magnitude compression ratio. Generated patterns have minimal length, e.g., the shortest context possible, which results in their *generalization* properties.

In general, *exact lossless* pattern *minimization is non-polynomial* by reduction to the minimum set cover problem [19]. For Czech, *exact lossless* pattern generation *is feasible* [20], while reaching *100% coverage* and simultaneously *no errors*. Strict pattern minimality (size) is not an issue nowadays.

This idea and its realization is a programming pearl. Motivated by space and time constraints, instead of the classical solution of dictionary problem in the logarithmic time of dictionary size, the word hyphenation is computed from patterns in constant time, where the constant is given by *word* length.

Space needed for patterns in the *packed trie* data structure is typically in tens of kB, which is several orders of magnitude smaller than the wordlist size. With fine-tuned parameters of pattern generation in the so-called *levels*, one can prepare patterns with zero errors and almost full coverage of hyphenation points from the input dictionary.

For practical use, patterns are collected in the repository maintained by the TEX community [14]. It is no surprise that most if not all leading typesetting engines deploy this "competing pattern engineering technology" [21].

Table 2: Language alphabet overlaps. Cells contain a number of lowercase letters that overlap between languages. In total, 13 languages contain in total 412 different lowercase letters, more than Patgen is capable of digesting.

Language	cz+sk	ka	el	pa	pl	ru	ta	te	th	tr	tk	ua
Czech+Slovak (cz+sk)	47	0	0	0	26	0	0	0	0	25	28	0
Georgian (ka)	0	33	0	0	0	0	0	0	0	0	0	0
Greek (el-monoton)	0	0	48	0	0	0	0	0	0	0	0	0
Panjabi (pa)	0	0	0	52	0	0	0	0	0	0	0	0
Polish (pl)	26	0	0	0	34	0	0	0	0	23	22	0
Russian (ru)	0	0	0	0	0	33	0	0	0	0	0	29
Tamil (ta)	0	0	0	0	0	0	48	0	0	0	0	0
Telugu (te)	0	0	0	0	0	0	0	66	0	0	0	0
Thai (th)	0	0	0	0	0	0	0	0	64	0	0	0
Turkish (tr)	25	0	0	0	23	0	0	0	0	32	25	0
Turkmen (tk)	28	0	0	0	22	0	0	0	0	25	30	0
Ukrainian (ua)	0	0	0	0	0	29	0	0	0	0	0	33

```
    h y p h e n a t i o n
p1            1n a
p1                1t i o n
p2             n2a t
p2                  2i o
p2          h e2n
p3  h y3p h
p4          h e n a4
p5          h e n5a t
    h0y3p0h0e2n5a4t2i0o0n
```

hy-phen-ation → 2 6
... → ...
... → ...
key → data

Solution to the dictionary problem:
For key part (the word) to store
the data part (its division)

Figure 1: Eight patterns "compete" how to hyphenate *hyphenation*. Winners are patterns `hy3ph` and `hen5at` generated at the highest covering level (odd numbers) generation. The level hierarchy allows for storing exceptions, exceptions to exceptions, exceptions to exceptions to exceptions, ..., with character contexts as parameters. [12]

3.1. Patterns

The patterns "compete" with each other whether to split the word at a position, given varying characters in both side contexts; see Figure 1.

We have shown how effective and powerful the technique is, and that its power depends on the *parameters* of pattern generation [20]. The key is the proper setting of Patgen parameters for pattern generation. The idea of universal segmentation with Patgen has been proposed already in [22]. There, we demonstrated the techniques for the development of two languages together, Czech and Slovak, and developed a joint wordlist and patterns [13].

We wanted to extend the technique to other Slavic and syllabic languages. The bottleneck for adding new languages was Patgen and TEX's constraint of one-byte character support only for storing patterns in tries. We thought of using a modern data structure that would allow wide character trie representation. That was the task for a bachelor's thesis: use a Judy array [23].

3.2. Judy arrays

The Judy array, also known as simply Judy, is a data structure that implements a sparse dynamic array, allowing for versatile applications such as dynamically sized arrays and associative arrays. Judy is internally implemented as a tree structure, where every internal node has 256 ancestor nodes. The most interesting thing about this structure is that it tries to be as memory-efficient as possible by effectively using the available cache, avoiding unnecessary access to the main memory. As a result, Judy is both fast and memory-efficient.

The feasibility of utilizing the Judy structure for storing hyphenation patterns is demonstrated in the thesis [23]. In Chapter 4, it is shown that Judy has the potential to be faster and more memory-efficient compared to the original trie when working with patterns. Further, Chapter 5 explores the potential integration of Judy into Patgen and the consequent impact on Patgen's generation process. The results from this chapter indicate that rewriting Patgen with Judy is possible but would require an almost complete overhaul of Patgen's code and algorithms. This redevelopment would yield a Patgen version capable of handling input of any kind, enabling the generation of patterns composed of arbitrary alphabets. However, it is important to note that the generation process would be approximately four times slower than the current implementation. This is due to the hiding of access to the inner nodes of stored tries in Judy. As this access is not needed in TEX for the hyphenation of individual words, using some variant of Judy in a TEX successor would make hyphenation faster.

Table 3: Different word hyphenation overlaps. Cells contain a number of the same words that are segmented differently between languages. Differences are caused typically by suboptimal coverage patterns used to hyphenate the wordlist (*vi-bram* vs. *vib-ram*, *up-gra-de* vs. *upg-ra-de*). We remove the differently hyphenated words when joining wordlists for the final syllabic generation.

Language	cz+sk	ka	el	pa	pl	ru	ta	te	th	tr	tk	ua
Czech+Slovak (cz+sk)	9	0	0	0	388	0	0	0	0	640	69	0
Georgian (ka)	0	0	0	0	0	0	0	0	0	0	0	0
Greek (el-monoton)	0	0	0	0	0	0	0	0	0	0	0	0
Panjabi (pa)	0	0	0	0	0	0	0	0	0	0	0	0
Polish (pl)	388	0	0	0	0	0	0	0	0	187	9	0
Russian (ru)	0	0	0	0	0	0	0	0	0	0	0	125
Tamil (ta)	0	0	0	0	0	0	0	0	0	0	0	0
Telugu (te)	0	0	0	0	0	0	0	0	0	0	0	0
Thai (th)	0	0	0	0	0	0	0	0	0	0	0	0
Turkish (tr)	640	0	0	0	187	0	0	0	0	0	80	0
Turkmen (tk)	69	0	0	0	9	0	0	0	0	80	0	0
Ukrainian (ua)	0	0	0	0	0	125	0	0	0	0	0	0

3.3. Universal pattern generation

To pursue the idea of universal syllabic pattern generation, we have checked whether the legacy patterns hyphenate the same valid word in different languages differently. The result with a short discussion is in Table 3. The expectation

Table 4: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk, ua with *custom* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 33.23 seconds, 11,238 patterns, 77 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	2,407	2,066,410	280,020	70,588	1 3	1 3 12
2	2,375	2,025,245	8,866	111,753	2 4	1 1 5
3	4,626	2,118,063	19,213	18,935	3 6	1 2 4
4	2,993	2,117,739	5,920	19,259	3 7	1 4 2

Table 5: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk, ua with *correct optimized* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 35.43 seconds, 29,742 patterns, 219 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	7,188	2,049,375	164,224	87,623	1 3	1 5 1
2	4,108	2,042,249	14,094	94,749	1 3	1 5 1
3	15,010	2,134,692	20,544	2,306	2 6	1 3 1
4	6,920	2,133,458	815	3,540	2 7	1 3 1

Table 6: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk, ua with *size optimized* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 29.75 seconds, 14,321 patterns, 101 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	1,201	2,092,928	598,321	44,070	1 3	1 2 20
2	2,695	1,736,372	5,274	400,626	2 4	2 1 8
3	4,835	2,102,803	20,094	34,195	3 5	1 4 7
4	6,508	2,099,607	210	37,391	4 7	3 2 1

that syllable-forming principles are universal, as phonology theory suggests, is confirmed. The errors we have found were due to the difference between hyphenation and syllabification caused by inconsistent markup rather than a principled difference in word morphology, e.g. a compound word segmented in one language, and given as a single word in the other.[2]

[2]Compound words can evolve in perception into single words even within one language. Examples are the evolution of *e-mail* into *email* or *roz-um* into syllabic *ro-zum* in Czech.

Table 7: Comparison of the efficiency of different approaches to pattern generation of Czechoslovak and of universal patterns. Note that the size of universal patterns grows sublinearly with the number of languages. The generalization ability of universal patterns is only slightly worse than that of Czechoslovak ones. The experience from the development of Czechoslovak patterns shows that performance could be improved by consistent markup of wordlist data.

Wordlist	Parameters	Good	Bad	Missed	Size	Patterns
Czechoslovak	custom	99.87%	0.03%	0.13%	32 kB	5,907
Czechoslovak	correctopt	99.99%	0.00%	0.01%	45 kB	8,231
Czechoslovak	sizeopt	99.67%	0.00%	0.33%	40 kB	7,417
Universal	custom	99.10%	0.28%	0.90%	77 kB	11,238
Universal	correctopt	99.83%	0.04%	0.17%	219 kB	29,742
Universal	sizeopt	98.25%	0.01%	1.75%	101 kB	14,321

Table 8: Results of 10-fold cross-validation (learning on 90%, and testing on remaining 10%). Generalization properties (performance on words not seen during training) are compared with Czechoslovak patterns. By adding 7 languages, the generalization abilities of universal patterns are only slightly worse.

Wordlist	Parameters	Good	Bad	Missed
Czechoslovak	custom	99.85%	0.22%	0.15%
Czechoslovak	correctopt	99.95%	0.15%	0.05%
Czechoslovak	sizeopt	99.58%	0.18%	0.42%
Universal	custom	99.04%	1.07%	0.96%
Universal	correctopt	99.37%	1.30%	0.63%
Universal	sizeopt	98.42%	0.95%	1.56%

We removed all colliding words when joining wordlists into the wordlist universal pattern generation. As mentioned earlier, we collected words for nine languages (cz, sk, ka, el, pl, ru, tr, tk, ua).

We generated universal patterns with the same three sets of Patgen parameters (custom, correct optimized, and size optimized) as when generating Czechoslovak patterns. The results are shown in Tables 4 (custom), 5 (correct optimized) and 6 (size optimized). The results are comparable with generation for two languages and confirm the feasibility of universal pattern development.

We did not pursue 100% coverage at all costs because the source data is noisy, and we do not want the patterns to learn all the typos and inconsistencies. Also, the size of the new languages was rather small, compared to Czechoslovak.

4. Evaluation

We evaluated the quality of developed patterns by two metrics. *Coverage* of hyphenation points in the training wordlist tells how the patterns correctly predicted hyphenation points used in training. *Generalization* means how the patterns behave on unseen data, on words not available in the data used during Patgen training. The methodology is the same as we used in the development of Czechoslovak patterns [13].

In Table 7, we compare the efficiency of different approaches to hyphenating 2 languages and 9 languages from one pattern set. We see that the performance of universal patterns is comparable in size and quality to double- or single-language ones—there is only a negligible difference. Table 8 shows that generalization qualities, given the small input size wordlists, are very good, and comparable to the fine-tuned Czechoslovak results. Investing in the purification and consistency of input wordlists (as we did for Czech and Slovak) would result in near-perfect syllabic patterns with almost 100% coverage and no errors.

5. Future work

A natural further step is to merge further languages where the syllabic principle is used for hyphenation. For that, one would need a version of Patgen we provisionally call UniPatgen. This version would support Unicode not only in I/O but also internally as a wide character (UTF-16) character encoded in the pattern representation in either a packed trie or Judy array. This would allow merging more languages *without* increasing the computational complexity of hyphenation, and only a sublinear increase of pattern size. We believe that coverage may differ from 100% only by words that should be hyphenated differently in different languages—our estimate is in small, single-digit percents, while, as mentioned above, the widely-used `hyphen.tex` patterns do not cover 10+%!

Another possible extension in pattern development is the support of a specific hyphenation penalty for compound word borders. This extension, discussed already 30 years ago [17], would generate patterns first for compound words, and only after fixing them continue with pattern generation for all other hyphenation points. The TEX engine would then set the hyphenation penalties depending on level ranges in patterns found for the hyphenated word. This extension is orthogonal with support for universal patterns but might require increasing the maximal number of levels allowed in patterns to two digits.

There are several open questions for the TEX development community:

1. Should the universal syllabic patterns ever be developed?

2. If so, should the needed *internal* wide character representations be added to the TEX suite of programs? That is, to TEX-based engines not yet supporting it[3] and Patgen or UniPatgen.
3. If not, should it be handled by external segmenters on TEX's input, based on Patgen's proposed successor, UniPatgen?
4. If UniPatgen was developed, should it be added to the distribution, together with Unicode patterns included and supported in repositories like [14]?
5. Should UniPatgen, and LuaTEX, add a dependency on a Judy library, or should a more conservative solution be sought and implemented? With a conservative solution, which data structure to use for storing patterns? Should the memory be allocated dynamically, to overcome the abundant explosion of format size that stores the patterns, as output by iniTEX?
6. Should UniPatgen (and TEX engines) additionally and orthogonally support patterns and a different hyphenation penalty for compound word borders, currently available in e.g. the German wordlist [24]?

We would appreciate qualified opinions on these decisions being sent to authors.

"All we are saying, give patterns a chance."
Our paraphrase of John Lennon's protest song refrain

6. Conclusion

Preparation of language-agnostic, i.e. universal, syllabic segmentation patterns could be done! We have demonstrated this possibility by generating patterns based on the wordlists of nine languages with current Patgen. They have superb generalization qualities, high coverage of hyphenation points (more than most legacy patterns), and virtually no errors. Their use could have a high impact on virtually all typesetting engines including web page renderers.

Supporting wide characters in Patgen is a critical requirement for adding more languages. We have shown that bringing wide character support into the hyphenation part of the TEX suite of programs is possible by using the Judy array. It will allow generating and deploying patterns for the whole Unicode character set. We have discussed a possible roadmap to make this a reality in typesetting engines including TEX successors.

Acknowledgments
We are indebted to Don Knuth for questioning the common properties of Czech and Slovak hyphenation during our presentation of [20] at TUG 2019, which has led us in this research direction. We also thank everyone on whose shoulders we build our work, e.g. for wordlists by Lexical Computing, and to all who commented on our work at TUG 2021 [13] and TUG 2023. We thank CSTUG for TUG travel support.

[3] https://cs.overleaf.com/learn/latex/TeX_primitives_listed_by_TeX_engine

References

1. SOJKA, Ondřej; SOJKA, Petr; MÁCA, Jakub. A roadmap for universal syllabic segmentation. *TUGboat*. 2023, vol. 44, no. 2. ISSN 0896-3207. Dostupné také z: https://doi.org/10.47397/tb/44-2/tb137sojka-syllabic.
2. HARALAMBOUS, Yannis. New hyphenation techniques in Ω_2. *TUGboat*. 2006, vol. 27, no. 1, s. 98–103. Dostupné také z: https://tug.org/TUGboat/tb27-1/tb86haralambous-hyph.pdf.
3. BARTLETT, Susan; KONDRAK, Grzegorz; CHERRY, Colin. Automatic Syllabification with Structured SVMs for Letter-to-Phoneme Conversion. In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Assoc. for Computational Linguistics, 2008, s. 568–576. Dostupné také z: https://aclweb.org/anthology/P08-1065.
4. MARCHAND, Yannick; ADSETT, Connie R.; DAMPER, Robert I. Automatic Syllabification in English: A Comparison of Different Algorithms. *Language and Speech*. 2009, vol. 52, no. 1, s. 1–27. Dostupné z DOI: 10.1177/0023830908099881.
5. SHAO, Yan; HARDMEIER, Christian; NIVRE, Joakim. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics*. 2018, vol. 6, s. 421–435. Dostupné z DOI: 10.1162/tacl_a_00033.
6. TROGKANIS, Nikolaos; ELKAN, Charles. Conditional Random Fields for Word Hyphenation. In: *Proceedings of the 48th Annual Meeting of the ACL*. Uppsala, Sweden: ACL, 2010, s. 366–374. Dostupné také z: https://aclweb.org/anthology/P10-1038.
7. KRANTZ, Jacob; DULIN, Maxwell; PALMA, Paul De. Language-Agnostic Syllabification with Neural Sequence Labeling. *CoRR*. 2019, vol. abs/1909.13362. Dostupné také z: https://arxiv.org/abs/1909.13362.
8. THE UNICODE CONSORTIUM. *The Unicode Standard: Worldwide Character Encoding. Version 15.1.* Mountain View, CA, USA: Unicode, Inc., 2023. ISBN 978-1-936213-32-0. Dostupné také z: https://unicode.org/versions/Unicode15.1.0.
9. MADDIESON, Ian. Syllable Structure. In: DRYER, Matthew S.; HASPELMATH, Martin (eds.). *The World Atlas of Language Structures Online*. Leipzig: Max Planck Institute for Evolutionary Anthropology, 2013. Dostupné také z: https://wals.info/chapter/12.
10. *Online Etymology Dictionary. "syllable"* [online]. [N.d.]. [cit. 2023-07-23]. Dostupné z: https://www.etymonline.com/word/syllable.
11. *Internetová jazyková příručka (Internet Language Reference Book)* [online]. 2023. [cit. 2023-07-06]. Dostupné z: https://prirucka.ujc.cas.cz/?id=135.

12. LIANG, Franklin M. *Word Hy-phen-a-tion by Com-put-er*. 1983. Dostupné také z: https://tug.org/docs/liang/liang-thesis.pdf. Dis. pr. Stanford University.
13. SOJKA, Petr; SOJKA, Ondřej. New Czechoslovak Hyphenation Patterns, Word Lists, and Workflow. *TUGboat*. 2021, vol. 42, no. 2. ISSN 0896-3207. Dostupné také z: https://doi.org/10.47397/tb/42-2/tb131sojka-czech.
14. ROSENDAHL, Arthur; MIKLAVEC, Mojca. *TEX hyphenation patterns*. TUG, 2023. Dostupné také z: http://hyphenation.org/tex. Accessed 2023-07-05.
15. JAKUBÍČEK, Miloš; KILGARRIFF, Adam; KOVÁŘ, Vojtěch; RYCHLÝ, Pavel; SUCHOMEL, Vít. The TenTen Corpus Family. In: *Proc. of the 7th International Corpus Linguistics Conference (CL)*. Lancaster, 2013, s. 125–127.
16. KNUTH, Donald E. *3:16 Bible Texts Illuminated*. A-R Editions, Inc., 1991. ISBN 0-89579-252-4.
17. SOJKA, Petr; ŠEVEČEK, Pavel. Hyphenation in TEX—Quo Vadis? *TUGboat*. 1995, vol. 16, no. 3, 280–289. Dostupné také z: https://tug.org/TUGboat/tb16-3/tb48soj1.pdf.
18. LIANG, Franklin M.; BREITENLOHNER, Peter. *PATtern GENeration Program for the TEX82 Hyphenator* [Electronic documentation of PATGEN program version 2.4 on CTAN. https://ctan.org/pkg/patgen]. 1999.
19. SOJKA, Petr. *Competing Patterns in Language Engineering and Computer Typesetting*. 2005. Dostupné také z: https://researchgate.net/publication/265246931_Competing_Patterns_in_Language_Engineering_and_Computer_Typesetting/. Dis. pr. Faculty of Informatics.
20. SOJKA, Petr; SOJKA, Ondřej. The Unreasonable Effectiveness of Pattern Generation. *TUGboat*. 2019, vol. 40, no. 2, s. 187–193. Dostupné také z: https://tug.org/TUGboat/tb40-2/tb125sojka-patgen.pdf.
21. SOJKA, Petr. Competing Patterns for Language Engineering. In: SOJKA, Petr; KOPEČEK, Ivan; PALA, Karel (eds.). *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*. Brno, Czech Republic: Springer-Verlag, 2000, s. 157–162. LNAI 1902. Dostupné z DOI: 10.1007/3-540-45323-7_27.
22. SOJKA, Petr; SOJKA, Ondřej. Towards Universal Hyphenation Patterns. In: HORÁK, Aleš; RYCHLÝ, Pavel; RAMBOUSEK, Adam (eds.). *Proceedings of Recent Advances in Slavonic Natural Language Processing—RASLAN 2019*. Karlova Studánka, Czech Republic: Tribun EU, 2019, 63–68. Dostupné také z: https://nlp.fi.muni.cz/raslan/2019/paper13-sojka.pdf. https://is.muni.cz/publication/1585259/?lang=en.

23. MÁCA, Jakub. *Judy*. Brno, Czech Republic, 2023. Dostupné také z: `https://is.muni.cz/th/kru3j`. Bachelor Thesis supervised by Petr Sojka and defended at Masaryk University, Faculty of Informatics.
24. LEMBERG, Werner. *A database of German words with hyphenation information*. 2023. Dostupné také z: `https://repo.or.cz/wortliste.git`.

Plán pro univerzální slabičnou segmentaci

Prostorově a časově efektivní segmentace (dělení slov) přirozených jazyků zůstává jádrem každého sázecího systému, ať už jde o TEX, webový prohlížeč nebo mobilní operační systém. Ve většině jazyků je dnes pragmaticky preferováno slabičné dělení reflektující výslovnost při čtení.

Vzhledem k tomu, že přepínání jazyků často není v textech označeno, renderovací stroj (webový prohlížeč či TEX) potřebuje *univerzální* slabikovou segmentaci. V předloženém článku ukazujeme proveditelnost této myšlenky tím, že nabízíme prototypové řešení dvou hlavních problémů:

A) použití Patgenu ke generování vzorů pro několik jazyků najednou; a
B) neexistence podpory Unicode v nástrojích jako Patgen nebo TEX (vzory v kódování UTF-16).

Pro A) jsme ke generování univerzálních slabičných vzorů použili seznamy slov devíti slabičných jazyků (čeština, slovenština, gruzínština, řečtina, polština, ruština, turečtina, turkmenština a ukrajinština). Pro B) jsme vytvořili verzi Patgen, která používá datovou strukturu Judy array, a porovnali její efektivitu s implementací trie.

S údaji z těchto devíti jazyků ukazujeme, že:

A) vyvinutí univerzálních, obecných slabičných vzorů s vysokým pokrytím je možné, a to s velkým dopadem na prakticky všechny sázecí stroje včetně webových prohlížečů; a
B) podpora Unicode znaků ve vzorech dělení slov v programech TEX a Patgen je možná pomocí Judy array.

Klíčová slova: slabikování, dělení slov, příprava univerzálních slabičných vzorů

Ondřej Sojka, Fakulta informatiky Masarykovy univerzity, Brno, ČR 454904 (at) mail dot muni dot cz, ORCID 0000-0003-2048-9977

Petr Sojka, Fakulta informatiky Masarykovy univerzity, Brno, ČR sojka (at) fi dot muni dot cz, ORCID 0000-0002-5768-4007

Jakub Máca, Fakulta informatiky Masarykovy univerzity, Brno, ČR 514024 (at) mail dot muni dot cz, ORCID 0009-0008-1583-3183

Co by každý (LA)TEXový nováček měl znát

Barbara Beeton

O LATEXu se říká, že se v něm dají vytvořit kvalitně vypadající dokumenty, ale za cenu toho, že je složitější se jej naučit. To je sice pravda, nicméně tyto obavy je možné zmírnit, když pochopíme několik základních principů a naučíme se vyhýbat některým technikám, které se mohou zdát jasné, avšak nevedou k cíli. Cílem tohoto článku je ukázat začínajícím uživatelům dobré LATEXové návyky dříve, než se začnou projevovat ty špatné.

Klíčová slova: LATEX, nováček, chyby, log soubor

Úvod

V tomto článku budu popisovat situace, se kterými jsem se v průběhu svého pracovního života často setkávala, a to zejména na dvou místech:

- při práci v týmu TEXnické podpory významného matematického nakladatelství, kde bylo nutné odpovídat na dotazy autorů a psát uživatelskou dokumentaci,
- na online TEXovém fóru na StackExchange [2], kde uživatelé pokládají hromady dotazů od jednoduchých až po velmi pokročilé. Členové komunity sepsali na fóru seznam často odkazovaných odpovědí [3].

Několikrát zde budu opakovat obecnou radu: Přečtěte si dokumentaci.

Důležité pojmy

Existuje několik pojmů, které se k začínajícím uživatelům nedostanou, nebo se k nim dostanou, ale jsou pro ně nesrozumitelné. Pojďme se na ně podívat.

Šablona

Mnoho nových (LA)TEXových[1] uživatelů si myslí, že třída dokumentu (soubor s příponou .cls) je šablonou pro konkrétní typ dokumentu. I když jde tato myšlenka správným směrem, není to úplně pravda. Šablonou bývá zdrojový soubor

Z anglického originálu [1] přeložil Jan Šustek.

[1] Použijeme-li v článku výraz „(LA)TEX“, týká se příslušný text jak programu TEX, tak jeho nadstavby LATEX. Pokud závorky nepoužijeme, týká se text pouze LATEXu. (pozn. překl.)

DOI: 10.5300/2023-3-4/139

s příponou .tex, který obsahuje kostru dokumentu napsaného v daném stylu. Soubor začíná příkazem `\documentclass` a ukazuje použití základních příkazů v dokumentu. Do něj je pak možné přidat vlastní definice a text dokumentu. V ideálním případě lze šablonu zpracovat (LA)TEXem bez chyb, nicméně výsledek zřejmě nebude nijak užitečný.

Příkazový řádek

Většina nových uživatelů v současnosti spouští (LA)TEX z editoru nebo z jiného grafického rozhraní, přičemž jej spouští neinteraktivně a čekají, dokud program neskončí (s chybou, nebo bez), nebo se nezacyklí. Spouštění z příkazového řádku naopak umožňuje s programem komunikovat a v určitých situacích opravit chybu za běhu, nebo, pokud to není možné, ukončit program dříve, než začne vypisovat množství neužitečných chybových zpráv.

Jedním druhem chyb opravitelných za běhu je překlep v názvu příkazu.

```
! Undefined control sequence.
l.137 \scetion
                {Nadpis}
?
```

Uživatel může překlep opravit napsáním

```
i\section
```

a stiskem klávesy Enter. Samozřejmě je potom nutné chybu opravit také ve zdrojovém souboru.

Na druhou stranu, překlep v názvu prostředí takto opravit nelze. V takovém případě uživatel napíše `x` a stiskne Enter, čímž se program ukončí. Po opravě zdrojového souboru program spustí znovu.

Jestliže po neopravitelné chybě uživatel program neukončí, budou pravděpodobně následovat další chyby způsobené tím, jak se (LA)TEX snažil ze situace zotavit. Příslušné chybové zprávy už ale budou docela matoucí. Tomu je lepší se vyhnout a program ukončit.

Soubor .log

Vždy, když se spustí (LA)TEX, vytvoří se textový soubor s příponou .log. Soubor se vytvoří ve stejném adresáři jako hlavní zdrojový soubor – najděte si jej. Soubor obsahuje všechny chybové zprávy a varování, ale obsahuje také seznam všech načítaných souborů, u načítaných tříd a balíčků navíc i s uvedením jejich verze. Dále obsahuje čísla zpracovaných stran a informace o použitých zdrojích. V článku se zaměříme pouze na některé z těchto věcí. V dřívějším článku [4] (a jeho českém překladu [5]) je popsáno ladění zdrojového souboru také ve složitějších situacích.

Základní struktura

V této sekci probereme některé základní prvky (LA)TEXu.

Příkazy

Jednotlivé instrukce se (LA)TEXu předávají pomocí příkazů (tzv. řídicích sekvencí), které zpravidla začínají znakem \ (zpětné lomítko). Existují dva typy řídicích sekvencí:

- zpětné lomítko následované jedním znakem jiným než písmeno (tzv. řídicí znak),
- zpětné lomítko následované jedním nebo více písmeny (tzv. řídicí slovo), která mohou být malá i velká (přičemž na jejich velikosti záleží), avšak nemůže se jednat o číslice ani žádné speciální znaky.[2]

Řídicí sekvence mohou mít jeden nebo více argumentů (`\title{...}`) nebo mohou být bez nich (`\alpha`). Řídicí slovo bývá ukončeno mezerou nebo libovolným jiným nepísmenným znakem, přičemž takovou ukončovací mezeru TEX odstraní. Naopak mezeru za řídicím znakem TEX neodstraní a projeví se na výstupu. V (LA)TEXu jsou některé řídicí znaky již nadefinovány tak, že vysází příslušný znak. Například `\#`, `\%`, `\$` a `\&` vysází #, %, $ a &.

Uživatel může nadefinovat vlastní příkazy nebo může předefinovat stávající. LATEX nabízí příkaz `\newcommand` k nadefinování nového příkazu. Přitom se kontroluje, jestli příkaz se stejným jménem již nebyl nadefinován dříve, a pokud byl, pak definici neprovede a vypíše chybovou zprávu. (Primitivní TEXový příkaz `\def` takovou kontrolu neprovádí.) Pro předefinování existujícího příkazu slouží LATEXový příkaz `\renewcommand`.

Při předefinování si musíte být naprosto jistí, že víte, co děláte! Například předefinování příkazu `\par` je hodně riskantní, protože LATEX používá příkaz `\par` uvnitř svých příkazů skrytě před uživatelem a jeho jiná definice snadno způsobí mnoho chyb.

Řídicí slova obsahující jediné písmeno také není moc vhodné předefinovávat, protože mnoho z nich slouží k sazbě akcentů nebo k sazbě různých neanglických písmen. Tato písmena se mohou vyskytnout například v cizojazyčných jménech. Představme si, že v článku o komplexních číslech nadefinujeme příkaz pro imaginární jednotku a zároveň budeme citovat autora Cema Yıldırıma.

```
\renewcommand{\i}{\ensuremath{\sqrt{-1}}}
Cem Y\i ld\i r\i m
```

Pak se nemůžeme divit, když na výstupu dostaneme

Cem Y$\sqrt{-1}$ld$\sqrt{-1}$r$\sqrt{-1}$m

[2] Diskuse o kategoriích znaků a přesných pravidlech tvorby řídicích tokenů přesahuje rámec tohoto článku. Pro detaily může čtenář nahlédnout do [6]. (pozn. překl.)

Řídicí znaky obsahující cifru (`\0`, `\1` atd.) nejsou v LaTeXu nadefinovány a jsou uživatelům k dispozici pro vlastní příkazy.

Prostředí

Pro LaTeX je charakteristické používání prostředí, což je blok kódu mezi

`\begin{`⟨*prostředí*⟩`}` a `\end{`⟨*prostředí*⟩`}`

Název prostředí na začátku i konci se musí shodovat. V opačném případě se vypíše chybová zpráva

`! LaTeX Error: \begin{`⟨*prostředí1*⟩`} on input line` ⟨*číslo*⟩
`ended by \end{`⟨*prostředí2*⟩`}.`

Většinu prostředí je možné vnořit do jiných prostředí. Pak je ale nutné jednotlivá prostředí ukončovat ve správném pořadí.

K definování příkazů LaTeX obsahuje ještě příkaz `\newenvironment` a novější příkazy `\NewDocumentCommand`, `\NewDocumentEnvironment` a ke všem analogické příkazy pro předefinování. Pro jejich popis si přečtěte dokumentaci.[3]

Módy

Zjednodušeně řečeno, aktuální mód určuje, kde na výstupu se právě nacházíme. My se však na módy podíváme na úrovni vstupního souboru.

Módy jsou tři: vertikální, horizontální a matematický.

(La)TeX je ve vertikálním módu na začátku dokumentu, po příkazu `\par` nebo po vynechaném řádku ve vstupním souboru.

Do horizontálního módu se dostaneme, když začneme psát text. Pro přechod z vertikálního do horizontálního módu je také možné použít příkazy `\indent`, `\noindent` nebo `\leavevmode`. V horizontálním módu se několik po sobě jdoucích mezer považuje za jedinou mezeru. Tady je důležité, že se jedná o mezery jdoucí po sobě. Na začátku řádku jsou mezery ignorovány. Konec řádku se také považuje za mezeru, i když ve vstupním souboru není vidět. Některé editory při zalamování textu automaticky vkládají znak konce řádku, jiné ne, a také na různých operačních systémech je znak konce řádku definován různě. Tyto odchylky TeX ihned na vstupu zohlední a uživatel pak nepozná rozdíl. Více si o tom povíme později.

Matematický mód je určen pro sazbu vzorců. Vzorec můžeme vložit přímo do textu odstavce nebo vycentrovat na zvláštní řádek. Vzorce v textu vysázíme pomocí `$...$` nebo `\(...\)`. Jednořádkové nečíslované vycentrované vzorce vysázíme pomocí `$$...$$` nebo `\[...\]`. Pro víceřádkové vzorce jsou v LaTeXu

[3] V dokumentaci [7] se píše, že příkazy `\NewDocumentCommand`, `\NewDocumentEnvironment` a jim podobné byly do jádra LaTeXu přidány až v roce 2020, proto jejich popis v dřívější dokumentaci asi nenajdete. (pozn. překl.)

balíčky amsmath nebo mathtools. (Přečtěte si dokumentaci. Balíček amsmath se načítá uvnitř balíčku mathtools, proto není nutné načítat oba balíčky.) Vycentrované vzorce obvykle patří k předchozímu odstavci, proto nad vzorcem nevynechávejte prázdný řádek (to by mohlo vést i k nesprávnému zalomení stránky nad vzorcem).

Uvnitř matematického módu není dovoleno vložit prázdný řádek. To je rozhodnutí Knutha s cílem jednoduššího nalezení chyb ve zdrojovém souboru. Matematická sazba totiž nikdy nepokračuje do dalšího odstavce.

Skupiny

Dalším ze základních prvků je skupina. Ta umožňuje lokálně omezit platnost různých nastavení a definic.

Skupinu tvoří například matematický mód. Určité symboly a operace jsou možné pouze uvnitř matematického módu a jinde nejsou povoleny. V textu odstavce matematický mód obvykle začíná a končí znakem `$` a tyto znaky musí být spárovány. Začátek centrovaného vzorce přeruší tok textu a na jeho konci se TEX vrátí zpět do horizontálního módu, případně, pokud následuje prázdný řádek nebo příkaz `\par`, do vertikálního módu. Více si o tom povíme později.

Další možnost vytvoření skupiny je uzavřít text do složených závorek `{...}`. Uvnitř skupiny můžeme dočasně předefinovat příkaz a ten se po ukončení skupiny vrátí na svou původní definici. Namísto složených závorek můžeme se stejným výsledkem[4] uzavřít text mezi příkazy `\begingroup` a `\endgroup`.

Skupina se také automaticky vytvoří, když uzavřeme materiál do boxu. Materiál se uzavře do boxu také v prostředí `minipage`, příkazech `\mbox`, `\parbox` nebo například v příkazech balíčku tcolorbox.

Některá prostředí (avšak ne všechna) také vytvářejí skupiny. Například prostředí `theorem` – v něm se text přepíná do kurzívy a po jeho ukončení se přepne zpět.

Mezery v textu

Cílem vysoce kvalitní sazby je, aby všechny mezery v textu byly stejně široké. To je ale možné pouze v případě sazby na praporek (tzn. odstavec je zarovnaný pouze na jedné straně), kdy mezery mají svou přirozenou šířku. Obvykle se však preferuje zarovnání odstavce do bloku a TEX je naprogramovaný tak, aby i při zarovnání do bloku byly šířky mezer podobné.

V anglických dokumentech[5] má být mezera za koncem věty širší než ostatní mezislovní mezery. To TEX dělá. V dokumentech v jiných jazycích je možné toto

[4] Existují jasně popsané situace, ve kterých se výsledek mírně liší, jejich popis však přesahuje rámec tohoto článku. Pokročilejší čtenář může nahlédnout do [6]. (pozn. překl.)

[5] Tato podsekce se týká zejména anglických dokumentů, případně částí dokumentů, v nichž je jako jazyk nastavena angličtina. (pozn. překl.)

chování vypnout příkazem `\frenchspacing`.[6] Nicméně akademické dokumenty jsou často psány anglicky a obsahují dost zkratek, a proto není pro TEX vždy zřejmé, kde končí věta. Aby za zkratkou následovala normálně široká mezera, píšeme za zkratku `\␣` (zpětné lomítko a mezera). Porovnejte

`abc␣vs.␣xyz` (abc vs. xyz) oproti `abc␣vs.\␣xyz` (abc vs. xyz).

Jestliže se za zkratkou nemá zlomit řádek, použijte znak `~` (vlnka).

```
See Fig.~37 on p.~159.
```

Podobná, ale opačná situace může nastat, když tečka následuje za velkým písmenem. TEX předpokládá, že se jedná o počáteční písmeno jména, a za tečku pak vkládá normální mezislovní mezeru. Avšak někdy je velké písmeno součástí zkratky a ta se může vyskytnout na konci věty. V LATEXu je definován příkaz `\@`, který se v takových případech vkládá mezi ono velké písmeno a tečku, aby se mezera za tečkou chovala jako mezera za koncem věty.

```
I like CSTUG\@.
```

Z předchozích odstavců dostáváme jednoduché pravidlo: S výjimkou konce věty (a v menší míře také za dalšími interpunkčními znaménky nebo uvnitř matematických výrazů) mají být všechny mezery na jednom řádku stejně široké. Pokud nejsou, něco není v pořádku.

Zavlečené mezery

Může se stát, že na výstupu dostaneme několik mezer za sebou. V převážné většině je to důsledkem toho, že se snažíme definovat nové příkazy tak, aby definice byly dobře čitelné. Mějme například definici

```
\newcommand{\abc}{
  \emph{abc def}
}
```

Potom při vstupu

```
nějaký text \abc\ další text
```

dostaneme na výstupu

nějaký text *abc def* další text

Mezery navíc kolem šikmého textu nesprávně vložil náš příkaz `\abc`. Jak jsme si řekli dříve, konce řádku TEX považuje za mezery. Proto musíme použít znak `%`, ten uvozuje komentář a TEX ignoruje vše, co je za ním až do konce řádku, včetně znaku konce řádku. S opravenou definicí

[6] Pokud v (LA)TEXu máme nastaveno, že aktuální jazyk je čeština nebo slovenština, pak je `\frenchspacing` automaticky nastaveno vnitřními makry. (pozn. překl.)

```
\newcommand{\abc}{%
  \emph{abc def}%
}
```

dostaneme správný výsledek

nějaký text *abc def* další text

Další příčinou zavlečených mezer může být přítomnost několika po sobě jdoucích objektů, které nejsou součástí hlavního textu, například poznámek pod čarou nebo položek v rejstříku.

```
Důležitý pojem\index{pojem}
  \index{jiný název}
  \index{varianta názvu}
je v~rejstříku uveden na několika místech.
```

V tomto případě se každý znak konce řádku převede na mezeru, a protože tyto mezery nejsou na vstupu souvislé, na výstupu do jediné mezery nesplynou.

Důležitý pojem je v rejstříku uveden na několika místech.

Opět je řešením použít znak `%`. Přitom ale musíme dát pozor, aby nám zůstala přesně jedna mezera.

```
Důležitý pojem\index{pojem}%
  \index{jiný název}%
  \index{varianta názvu}
je v~rejstříku uveden na několika místech.
```

Ne vždy je procento řešením

Připomínám, že mezeru, která ukončuje řídicí slovo, TEX odstraní a použití znaku `%` není nutné. Na některých místech může použití znaku `%` dokonce způsobit problémy.

Při načítání číselných hodnot hlavní procesor TEXu pokračuje ve čtení hodnoty, dokud nenarazí na jiný znak než číslici. Proto když na konci řádku máme `\xyz=123`, neměl by následovat znak `%`. Kdyby totiž následoval a další řádek by začínal číslicí, byla by tato číslice stále součástí načítané hodnoty.

Podobně když TEX načítá hodnotu pružné délky, například `\parskip=2pc`, zjišťuje, zda následují klíčová slova `plus` nebo `minus`. Je lepší načítání ukončit prázdnou skupinou `{}`. Kdybychom takto načítání neukončili a zároveň by `plus` nebo `minus` bylo součástí navazujícího textu, ohlásil by TEX chybu a příslušná zpráva by pro uživatele byla dost matoucí.

Opravdu nečekané mezery

Některé situace nejsou na první pohled předvídatelné. Tento případ se objevil v jednom online dotazu. Dokument používal balíček colorbox[7] a pomocí něj autor podbarvil jeden znak uprostřed slova. Je j da! Kolem znaku se vytvořily mezery. K podbarvení byly použity následující příkazy.

```
\usepackage{colorbox}
\newcommand{\seda}[1]{\colorbox{black!20}{#1}}
Je\seda{j}da!
```

Dokumentace k balíčku nabízí jako řešení vynulovat velikost okraje `\colorbox`u. Přidala jsem k tomuto řešení ještě `\strut`, aby barva byla zřetelná také nad a pod písmenem. Jejda!

```
\renewcommand{\seda}[1]{{\fboxsep=0pt
  \colorbox{black!20}{#1\strut}}}
Je\seda{j}da!
```

Přestože se nejedná úplně o problém pro začátečníky, je dobré vědět, že i taková situace může nastat a že je třeba se nebát v takových situacích obrátit na odborníka.

Konce odstavců a vertikální mód

Na konci odstavce TEX přechází z horizontálního do vertikálního módu. Prázdný řádek nebo příkaz `\par` tento přechod zajistí. Je důležité vědět, v jakém módu se aktuálně TEX nachází, protože jsou činnosti, které je lepší provádět ve vertikálním módu. Nejdůležitější z těchto činností je vkládání plovoucích objektů (obrázků, tabulek, algoritmů).

Dále je třeba mít na paměti, že některá nastavení se provádějí až na konci odstavce. Jedním z nich je vertikální vzdálenost účaří, která závisí na velikosti fontu. Příliš mnoho začátečníků se snaží ukončovat odstavce dvěma zpětnými lomítky. Tím se ale odstavec neukončí a vzdálenost účaří se nenastaví. To může vyústit ve výsledek, který

[7] Stejná situace nastává s balíčkem tcolorbox.

je vidět v tomto odstavci.

```
\huge
Dále je třeba mít na paměti, že
...
který je vidět v~tomto odstavci.\\
```

Problém uvedený výše nezpůsobí žádnou chybu ani varování. Řešením je v takovém případě správně ukončit odstavec. Některá prostředí (avšak ne všechna) jsou definována tak, že se na jejich konci odstavec automaticky ukončí.

Vertikální vzdálenost mezi odstavci je určena hodnotou `\parskip`. Tato hodnota bývá definována ve třídě dokumentu, nicméně je možné ji dle potřeby změnit. Někdy je třeba vložit vertikální mezeru ručně. K tomu slouží příkaz `\vspace` nebo `\vskip` použitý ve vertikálním módu (tedy za prázdným řádkem nebo příkazem `\par`).

Dvě zpětná lomítka

Co opravdu neukončuje odstavec, je řídicí znak `\\` (dvě zpětná lomítka). Tento příkaz ukončuje řádek v tabulkách, víceřádkových vzorcích, v básních a v některých dalších situacích. Ale neukončuje odstavec a může vyvolat mnoho chybových zpráv nebo varování.

Jestliže se příkaz `\\` vyskytne ve vertikálním módu, vypíše se chybová zpráva

```
! LaTeX Error: There's no line here to end.
```

Jestliže se před příkazem `\\` vyskytuje vysázená mezera, vytvoří se na výstupu další prázdný řádek, který tam být nemá.

Pokud je řádek končící příkazem `\\` příliš krátký, vypíše se varování

```
Underfull \hbox (badness 10000) in paragraph
          at lines ⟨číslo⟩--⟨číslo⟩
```

Takové varování může být správné, ale je třeba ho zkontrolovat.

Jestliže po `\\` následuje text v hranatých závorkách, `[který se má vysázet]`, dostaneme podivnou chybovou zprávu

```
! Missing number, treated as zero.
```

Příkaz `\\` má totiž nepovinný parametr (ten se dává do hranatých závorek), který udává, jak velká vertikální mezera se má vložit pod příslušným řádkem. Abychom vysázeli příslušný text v hranatých závorkách, musíme před levou závorku vložit příkaz `\relax`.

K ručnímu ukončení řádku odstavce je vhodnější použít příkaz `\newline`.

Změny písma

Změna písma je jednoduchým způsobem, jak v dokumentu zvýraznit důležité pojmy nebo odlišit text se speciálním významem. Mnoho takových změn je přímo zakomponováno do tříd a balíčků. Například nadpisy se sázejí **tučně**, teorémy *kurzívou* a některé časopisy nastavují popisky obrázků bezpatkově, aby je odlišily od okolního textu.

LaTeX nabízí dvě možnosti, jak změnit písmo. První typ příkazů načte svůj argument a pouze ten vysází jiným písmem. Mezi takové příkazy patří `\textbf{...}` pro **tučné písmo**, `\textit{...}` pro *kurzívu* nebo `\textsf{...}` pro bezpatkové písmo. Druhý typ příkazů jsou přepínače, které změní aktuální písmo a tato změna platí až do další změny nebo do konce skupiny. Mezi takové příkazy patří `{\bfseries...}`, `{\itshape...}` nebo `{\sffamily...}`. K těmto příkazům doporučuji přečíst si příslušnou dokumentaci.

Několik příkazů pro změnu písma se chová různě v závislosti na kontextu. Příkaz `\emph{...}` přepne do *kurzívy*, pokud je okolní text antikvou, *a přepne do* antikvy*, pokud je okolní text kurzívou.* V matematickém módu příkaz[8] `\text{...}` vysází text ve stejném písmu, jaké bylo aktuální před začátkem matematického módu. Například uvnitř teorému příkaz `\text{...}` vysází text kurzívou. Když chceme, aby text byl vždy antikvou, použijeme příkaz `\textup`.

Základní TeX definoval pro většinu změn písma dvoupísmenné příkazy. Tyto příkazy jsou druhého uvedeného typu, tj. přepínače. V LaTeXu bychom se těmto dvoupísmenným příkazům měli vyhnout, protože výše uvedené LaTeXové příkazy obsahují některá vylepšení, například provádějí hladší přechod mezi kurzívou a antikvou.

Matematická sazba

Matematická sazba vždy tvoří skupinu. Jestliže matematická sazba začne, musíme ji jasně ukončit. V horizontálním módu matematická sazba začíná a končí znakem `$`. Ke stejnému účelu jsou v LaTeXu definovány také příkazy `\(...\)` a prostředí `math`. Balíčky amsmath a mathtools nabízejí množství prostředí pro matematickou sazbu. Stojí za to se je naučit z příslušné dokumentace.

V matematickém módu (LA)TeX všechny mezery na vstupu ignoruje. Ve vstupním souboru je můžeme použít dle libosti, aby byl soubor dobře čitelný. Pro zvýšení čitelnosti ale v matematickém módu nemůžeme použít prázdný řádek. To způsobí chybu a TeX vypíše zprávu

```
! Missing $ inserted.
```

[8] Příkaz `\text` je definován v balíčku amstext, který se načítá uvnitř balíčku amsmath. (pozn. překl.)

Stejná zpráva se vypíše, pokud matematický mód neukončíme před koncem odstavce nebo pokud v textovém módu použijeme symbol, který je možné použít pouze v matematickém módu.

Jestliže se prázdný řádek vyskytne v některém matematickém prostředí balíčku amsmath, způsobí to chybu a první chybová zpráva bude

```
! Paragraph ended before \⟨prostředí⟩ was complete.
<to be read again>
```

Za touto chybovou zprávou budou následovat další, ale ty už budou docela matoucí. Všechny tyto následující chybové zprávy zmizí, pokud se opraví první chyba, tedy v našem případě odstraní-li se prázdný řádek.

Jestliže skutečně potřebujeme prázdný řádek, abychom zvýšili čitelnost zdrojového souboru, můžeme použít řádek obsahující pouze znak `%`.

Stejně jako u všech prostředí, i tady se musí název prostředí v příkazu `\end` shodovat s názvem prostředí v příkazu `\begin`, a to ve správném pořadí.

Prostředí pro sazbu víceřádkových vzorců by se neměla používat pro jednořádkové vzorce. Pro nečíslované jednořádkové vzorce můžeme použít `$$...$$` nebo LaTeXové příkazy `\[...\]` nebo prostředí `displaymath`.

I když LaTeX nabízí prostředí `eqnarray` pro víceřádkové zarovnané vzorce, nepoužívejte toto prostředí. Pokud je rovnice dlouhá a má své číslo, pak se rovnice a číslo vysázejí přes sebe. Namísto tohoto prostředí používejte raději příslušná prostředí z balíčku amsmath.

Tabulky, obrázky a další plovoucí objekty

Dovolený počet plovoucích objektů na stránce, jejich pozice a mezery mezi nimi jsou určeny ve třídě dokumentu. Když něco nefunguje podle očekávání, pak každý, na koho se obrátíte pro radu, bude chtít vědět, jakou třídu dokumentu používáte.

Ve vstupním souboru musí být plovoucí objekt v místě, kdy je ještě na výstupní stránce pro příslušný objekt dostatek místa. Speciálně při dvousloupcové sazbě to znamená, že prostředí `figure*` a `table*` musejí být ve vstupním souboru dříve než cokoliv jiného na aktuální stránce. Mechanismy LaTeXového jádra neumožňují, aby při dvousloupcové sazbě byly objekty přes celou šířku stránky vysázeny jinde než na horním okraji stránky. Existují balíčky, které tyto mechanismy vylepšují, nicmíně o nich zde psát nebudeme.

Základní třída article nastavuje následující hodnoty:

- maximální počet plovoucích objektů na stránce obsahující text: 3
- maximální počet plovoucích objektů na horním okraji stránky: 2
  maximální pokrytí stránky těmito objekty: 70 %
- maximální počet plovoucích objektů na dolním okraji stránky: 2
  maximální pokrytí stránky těmito objekty: 30 %

- minimální pokrytí stránky textem: 20 %
- minimální pokrytí stránky objekty vysázenými na samostatnou stránku: 50 % výšky sazby[9]

Jestliže je vkládaný objekt malý, má být vložen na přesně dané místo a vejde se tam, pak jej nevkládejte jako plovoucí objekt. Je lepší jej vložit přímo příkazem `\includegraphics` nebo příkazem pro vysázení tabulky, případně můžeme příkaz zapouzdřit do `\begin{center}...\end{center}`.[10]

Balíček wrapfig umožňuje obtékání textu kolem vložených obrázků. Pro detaily si přečtěte dokumentaci k balíčku.

Je zvykem psát popisky nad tabulky, ale pod obrázky. Pokud vložený objekt není plovoucí, nemůžeme použít obvyklý příkaz `\caption`. Namísto něj použijte balíček caption a příkaz `\captionof`.

Třída dokumentu a preambule

Když začínáte psát nový dokument, nejdříve si zvolte třídu dokumentu. Jestliže píšete článek do konkrétního časopisu, přečtěte si pokyny redakce a zjistěte si, jakou třídu a případně další soubory máte použít. Často používané třídy pro různé časopisy jsou k dispozici na CTAN [8].

Jestliže píšete nějaký projekt nebo diplomovou práci, zjistěte si příslušné požadavky; pokud má vaše instituce k dispozici konkrétní šablonu, použijte ji. Pokuste se zjistit, zda se šablona průběžně vyvíjí a jestli je k ní dostupná technická podpora. Přečtěte si dokumentaci.

Třída dokumentu musí definovat základní strukturu a důležité příkazy používané v dokumentu. Jestliže se připravovaný dokument výrazně liší od dokumentů, pro které je určena příslušná třída, je právě teď nejlepší čas obrátit se na někoho zkušenějšího.

Ne všechny součásti jsou ve třídě dokumentu přímo definovány. Například výběr stylu bibliografie může být ponechán na autorovi. Pro tyto situace byly v LaTeXu vytvořeny balíčky.

Příprava dokumentu

Většina balíčků se načítá v preambuli dokumentu, což je ta část zdrojového souboru, která se nachází mezi `\documentclass` a `\begin{document}`. Výjimkou jsou balíčky načítané příkazem `\RequirePackage`, který se obvykle používá před `\documentclass` – může se stát, že na tomto místě bude nutné načíst některá speciální nastavení.

[9] Do výšky sazby se nezapočítávají záhlaví a zápatí.

[10] V plovoucím prostředí pro vycentrování používejte příkaz `\centering`.

Někteří autoři si vytvoří preambuli, která je vhodná pro jeden konkrétní dokument, a poté tuto preambuli používají pro další dokumenty, přičemž přidávají další balíčky a tak dále. A potom přijde nějaký nováček, který si vezme tuto preambuli jako základní šablonu a bez jejího pochopení ji používá. Nedělejte to!

Začněte použitím vhodné třídy dokumentu a další balíčky, volby a definice přidávejte, až když je budete potřebovat. Načítání balíčků si uspořádejte do logických bloků (například všechny fontové balíčky dejte k sobě) a pečlivě si hlídejte, abyste žádný balíček nenačetli více než jednou. Jestliže některý balíček načítáte s volbami, pak jsou při jeho dalším načtení nové volby ignorovány. Některé balíčky interně načítají další balíčky. Například balíček mathtools načítá amsmath a balíček amssymb načítá amsfonts. Důležité je také pořadí balíčků. Balíček hyperref musí být načítaný téměř poslední – těch několik balíčků, které se mohou načítat až po hyperref, je dobře zdokumentováno.

Přečtěte si dokumentaci.

Zpracování dokumentu

Jakmile máme vytvořen zdrojový soubor, je na čase jej zpracovat (LA)TEXem a vygenerovat soubor výstupní. Je několik programů, ze kterých si podle potřeby můžeme vybrat, například pdfLATEX, XELATEXnebo LuaLATEX. Tyto programy můžeme spustit z příkazového řádku nebo prostřednictvím textového editoru. Někdy musíme zdrojový soubor zpracovat (LA)TEXem vícekrát. Kolikrát, to závisí na tom, jak je nutné přesouvat různé informace a jak se tyto informace mění.

Informace o křížových odkazech se zapisují do souboru .aux, informace o sekcích se zapisují do souboru .toc. Dokument může obsahovat i další seznamy. Seznam literatury se někdy zpracovává dalším programem (v takovém případě je potřebné zkontrolovat, jestli externí program nenahlásil chybu) a přeformátovaná data o jednotlivých záznamech jsou uložena do dalšího souboru. Potom je nutné LATEX spustit ještě minimálně dvakrát – jednou pro načtení souboru .aux a dalších vedlejších souborů (pro správné nastavení křížových odkazů) a podruhé pro správné nastavení čísel stran (jelikož například vložení obsahu nebo seznamu literatury v předchozím spuštění LATEXu mohlo text dokumentu posunout).

Předchozí odstavce předpokládají, že ve zdrojovém souboru nejsou chyby. Případné chyby jsou zaznamenány v souboru .log. Zjistěte si, kde se tento soubor nachází, a zvykněte si jej kontrolovat. Například varování o znacích chybějících v nějakém fontu se za běhu (LA)TEXu nezobrazí a jsou pouze v souboru .log.

`Missing character: There is no` ⟨*znak*⟩ `in font` ⟨*název*⟩`!`

Někdy soubor .log obsahuje skupinu chyb majících čísla řádku blízko sebe. V takovém případě první číslo řádku je to, na kterém (LA)TEX narazil na problém, zatímco na dalších uvedených řádcích ve skutečnosti chyba vůbec nemusí být.

Opravte první chybu a spusťte (LA)TEX znovu. Často se stane, že původní další chyby už se znovu neobjeví.

Přeji hodně štěstí. Časem se to naučíte.

A ještě... Nezapomeňte si přečíst dokumentaci.

Poděkování

Autorka děkuje samcarter, Mikaelovi Sundquistovi a (jako vždy) Karlu Berrymu za nápady a za nalezení a odstranění jejích překlepů. Dokážu překlepy najít u cizích článků, ale ne u svých.

Odkazy

1. BEETON, Barbara. What every (LA)TEX newbie should know. *TUGboat*. 2023, **44**(2), 164–169.
2. *TEX StackExchange* [online]. [cit. 2023-11-13]. Dostupné z: `https://tex.stackexchange.com`.
3. SCHARRER, Martin. *Often referenced questions* [online]. 2023-05-09. [cit. 2023-11-13]. Dostupné z: `https://tex.meta.stackexchange.com/q/2419`.
4. BEETON, Barbara. Debugging LATEX files – Illegitimi non carborundum. *TUGboat*. 2017, **38**(2), 159–164.
5. BEETON, Barbara. Ladění LATEXových souborů. *Zpravodaj CSTUGu*. 2021, **31**(1), 63–75.
6. OLŠÁK, Petr. *TEXbook naruby*. 2. vyd. Konvoj, 2001. Dostupné také z: `http://petr.olsak.net/ftp/olsak/tbn/tbn.pdf`.
7. MITTELBACH, Frank; FISCHER, Ulrike. *The LATEX Companion*. 3. vyd. Addison-Wesley Professional, 2023.
8. *CTAN: Search* [online]. [cit. 2023-11-13]. Dostupné z: `https://ctan.org/search`.

Summary: What Every (LA)TEX Newbie Should Know

LATEX has a reputation for producing excellent results, but at the cost of a steep learning curve. That's true, but by understanding a few basic principles, and learning how to avoid some techniques that may seem obvious but often lead one into the weeds, it's possible to avoid some of that pain. Our goal here is to encourage good habits before bad habits have had a chance to develop.

Keywords: LATEX, newbie, errors, log file

Barbara Beeton, TUGboat, Providence, RI, USA, `bnb@tug.org`

Sazba textu české lidové písně „Když jsem já sloužil“ pomocí modulu l3seq jazyka expl3

Vít Starý Novotný

Jazyk plain TEX vznikl pro sazbu knih a turingovsky úplným programovacím jazykem se stal až na konci svého vývoje. Zatímco příprava textu dokumentů a úpravy vzhledu jsou v plain TEXu přímočaré, programování naráží na chybějící základní datové struktury a na odloženou expanzi maker, která neodpovídá běžnému vyhodnocování v moderních imperativních jazycích.

Ve stroji LuaTEX je možné programovat také v imperativním programovacím jazyce Lua. Jazyk Lua sice zmíněnými neduhy plain TEXu netrpí, ale komunikace mezi TEXem a Luou není přímočará a při předávání dat dochází ke ztrátě důležitých informací, jako jsou kategorie TEXových znaků.

Programovací jazyk expl3 nabízí zlatou střední cestu a umožňuje uživatelům programovat v TEXu způsobem, na který jsou zvyklí z moderních imperativních programovacích jazyků.

V tomto článku představuji modul l3seq jazyka expl3, který poskytuje datovou strukturu seznamu. Možnosti modulu demonstruji na sazbě textu české lidové písně *Když jsem já sloužil*. Implementaci v jazyce expl3 porovnávám s implementací v plain TEXu.

Klíčová slova: LATEX3, expl3, l3seq, xparse, plain TEX, OpTEX, texty písní

TEX je strojový kód světa digitální sazby, který programátorům poskytuje minimum vysokoúrovňových abstrakcí. V předchozím článku [1, sekce 2] jsem představil vysokoúrovňový programovací jazyk expl3 [2, 3], díky němuž mohou uživatelé programovat v TEXu způsobem, na který jsou zvyklí z moderních imperativních programovacích jazyků. V dalším článku jsem představil modul l3keys jazyka expl3 [4, sekce 3.2], jenž umožňuje přípravu datových sběrnic. V tomto článku představuji modul l3seq jazyka expl3, který poskytuje datovou strukturu seznamu. Možnosti modulu demonstruji na sazbě textu české lidové písně *Když jsem já sloužil* [5]. Implementaci v jazyce expl3 porovnávám s implementací v plain TEXu.

Nejprve v sekci 1 popisuji algoritmus pro generování textu písně. V sekci 2 na straně 155 algoritmus implementuji pomocí modulu l3seq jazyka expl3. Následně v sekci 3 na straně 157 vytvářím uživatelské rozhraní pomocí LATEXového balíčku xparse. Dále v sekci 4 na straně 158 ukazuji vysázený text písně. Nakonec v sekci 5 na straně 159 algoritmus implementuji v plain TEXu a porovnávám obě implementace. V sekci 6 na straně 163 shrnuji poznatky z článku a jejich přínos pro čtenáře.

DOI: 10.5300/2023-3-4/153

1 Algoritmus pro generování textu písně

V této sekci představuji českou lidovou píseň *Když jsem já sloužil* [5]. Nejprve píseň zasazuji do historického kontextu a popisuji strukturu textu písně. Následně popisuji algoritmus pro generování textu písně.

Píseň *Když jsem já sloužil* je autobiografie mladého muže, který sloužil devět let jako zemědělský dělník. Vypravěč zmiňuje robotu a omezování sňatků vrchností, což děj zasazuje do dob Rakouska-Uherska před zrušením nevolnictví roku 1781.

Text písně se dělí na sloky, které popisují jednotlivé roky vypravěčovy služby. První věta každé sloky udává rok služby a obdrženou odměnu, např.: „Když jsem já sloužil to *3. léto*, vysloužil jsem si *husičku* za to.“ Následuje dlouhé souvětí s popisem odměn za dosavadní roky služby počínaje současným rokem a konče prvním rokem, např.: „A ta husa chodí bosa a ta kačka bláto tlačká a to kuře krákoře, běhá po dvoře.“ Sloku zakončuje vypravěč domněnkou o aktivitách své partnerky, např.: „Má panenka pláče doma v komoře.“

Pokud si odměny ve čtvrtém pádu (např. „husičku“) a popisy odměn (např. „ta husa chodí bosa“) uložíme do dvou seznamů `čtvrté_pády` a `popisy` délky $N = 9$, můžeme text písně vygenerovat Algoritmem 1. V algoritmu je použita funkce **zip**, která postupně navrací prvky obou seznamů jako dvojice.

Algoritmus 1: Text písně *Když jsem já sloužil*

```
   Data: Seznamy čtvrté_pády a popisy délky N
 1 i ← 1; popisy_pozpátku ← []
 2 for čtvrtý_pád, popis in zip(čtvrté_pády, popisy) do
 3     if i < N then
 4         print "když jsem já sloužil to " + i + ". léto"
 5     else
 6         print "když jsem já sloužil poslední léto"
 7     print "vysloužil jsem si " + čtvrtý_pád + " za to"
 8     j ← i; popisy_pozpátku ← [popis] + popisy_pozpátku
 9     for popis_pozpátku in popisy_pozpátku do
10         if j < N then print "a " + popis_pozpátku
11         else print popis_pozpátku
12         j ← j − 1
13     if i < N then
14         print "má panenka pláče doma v komoře"
15     else
16         print "má panenka stele postel v komoře"
17     i ← i + 1
```

2 Implementace algoritmu pomocí modulu l3seq

V této sekci rozpracujeme soubor `kdyz-jsem-ja-slouzil.sty`, který bude implementovat Algoritmus 1 pomocí modulu l3seq jazyka expl3. Hotový soubor `kdyz-jsem-ja-slouzil.sty` můžeme stáhnout online [6].

```
\ProvidesExplPackage
  {kdyz-jsem-ja-slouzil}{2023-08-28}{1.0.0}%
  {Balíček pro sazbu textu lidové písně ,,Když jsem já sloužil``}

% Globální proměnné
\seq_new:N \g_kdyzjsemjaslouzil_ctvrte_pady_seq
\seq_new:N \g_kdyzjsemjaslouzil_popisy_seq

% Lokální proměnné
\int_new:N \l_kdyzjsemjaslouzil_aktualni_sloka_int
\int_new:N \l_kdyzjsemjaslouzil_posledni_sloka_int
\bool_new:N \l_kdyzjsemjaslouzil_posledni_sloka_bool
\seq_new:N \l_kdyzjsemjaslouzil_popisy_pozpatku_seq

% Funkce pro sazbu textu celé písně
\cs_new:Nn
  \kdyzjsemjaslouzil_vysazej_pisnicku:
  {
    % Nastav počáteční hodnoty proměnných i a N.
    \int_set:Nn \l_kdyzjsemjaslouzil_aktualni_sloka_int { 1 }
    \int_set:Nn \l_kdyzjsemjaslouzil_posledni_sloka_int {
      \seq_count:N \g_kdyzjsemjaslouzil_popisy_seq }
    % Současně procházej hodnoty polí čtvrté_pády a popisy a volej
    % nad nimi pomocnou funkci pro sazbu textu jedné sloky.
    \seq_map_pairwise_function:NNN
      \g_kdyzjsemjaslouzil_ctvrte_pady_seq
      \g_kdyzjsemjaslouzil_popisy_seq
      \kdyzjsemjaslouzil_vysazej_sloku:nn
    % Vymaž pomocné pole s popisy v opačném pořadí.
    \seq_clear:N \l_kdyzjsemjaslouzil_popisy_pozpatku_seq
  }

% Funkce pro sazbu textu jedné sloky
\cs_new:Nn
  \kdyzjsemjaslouzil_vysazej_sloku:nn
  { % Nastav pomocnou proměnnou na pravdivostní hodnotu i = N,
    % která udává, jestli sázíme poslední sloku.
```

```
\bool_set:Nn
  \l_kdyzjsemjaslouzil_posledni_sloka_bool
  {
    \int_compare_p:nNn
      { \l_kdyzjsemjaslouzil_aktualni_sloka_int }
      =
      { \l_kdyzjsemjaslouzil_posledni_sloka_int }
  }
% Vysázej rok služby a obdrženou odměnu.
Když~jsem~já~sloužil~
\bool_if:NTF
  \l_kdyzjsemjaslouzil_posledni_sloka_bool
  { poslední~léto }
  {
    to~
    \int_use:N
      \l_kdyzjsemjaslouzil_aktualni_sloka_int
    .~léto
  }
, \\ vysloužil~jsem~si~#1~za~to. \\

% Vysázej popisy odměn za dosavadní roky služby.
\seq_put_left:Nn
  \l_kdyzjsemjaslouzil_popisy_pozpatku_seq
  { #2 }
\seq_map_indexed_inline:Nn
  \l_kdyzjsemjaslouzil_popisy_pozpatku_seq
  {
    % Při popisu odměny pro současný rok začni verzálkou.
    \int_compare:nTF
      { ##1 = 1 }
      {
        % V poslední sloce vynech počáteční spojku „a".
        \bool_if:NTF
          \l_kdyzjsemjaslouzil_posledni_sloka_bool
          {
            \str_uppercase:f { \tl_head:n { ##2 } }
            \tl_tail:n { ##2 }
          }
          { A~##2 }
      }
      { a~##2 }
```

```
      % Při popisu odměny pro první rok přidej čárku.
      \int_compare:nNnT
        { ##1 }
        =
        { \l_kdyzjsemjaslouzil_aktualni_sloka_int }
        { , } \\
    }
  % Vysázej vypravěčovu domněnku o aktivitách partnerky.
  má~panenka~
  \bool_if:NTF
    \l_kdyzjsemjaslouzil_posledni_sloka_bool
    { stele~postel~ }
    { pláče~doma~ }
  v~komoře. \par
  \int_incr:N
    \l_kdyzjsemjaslouzil_aktualni_sloka_int
}
```

3 Uživatelské rozhraní pomocí LaTeXového balíčku xparse

Programovací jazyk expl3 má pro uživatele TeXu poměrně neobvyklou syntax. V této sekci ukončíme soubor `kdyz-jsem-ja-slouzil.sty` a pomocí LaTeXového balíčku vytvoříme uživatelské rozhraní, které bude pro uživatele přirozenější.

```
\RequirePackage { xparse }
\NewDocumentEnvironment
  { kdyz-jsem-ja-slouzil }
  { }
  {
    \NewDocumentCommand
      \sloka
      { m m }
      {
        \seq_put_right:Nn
          \g_kdyzjsemjaslouzil_ctvrte_pady_seq
          { ##1 }
        \seq_put_right:Nn
          \g_kdyzjsemjaslouzil_popisy_seq
          { ##2 }
      }
  }
  { \kdyzjsemjaslouzil_vysazej_pisnicku: }
```

4 Sazba textu písně

V této sekci vytvoříme a vysázíme ukázkový soubor `priklad-latex.tex`. Soubor nejprve načte soubor `kdyz-jsem-ja-slouzil.sty`, který jsme připravili v sekcích 2 a 3, a následně vysází text písně *Když jsem já sloužil*.

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage[T1]{fontenc}
\usepackage{lmodern,multicol,parskip}
\usepackage{kdyz-jsem-ja-slouzil}
\begin{document}
\begin{multicols}{2}
\begin{kdyz-jsem-ja-slouzil}
\sloka {kuřátko}   {to kuře krákoře, běhá po dvoře}
\sloka {kachničku} {ta kačka bláto tlačká}
\sloka {husičku}   {ta husa chodí bosa}
\sloka {vepříka}   {ten vepř jako pepř}
\sloka {telátko}   {to tele hubou mele}
\sloka {kravičku}  {ta kráva mléko dává}
\sloka {volečka}   {ten vůl jako kůl}
\sloka {botičky}   {ty boty do roboty}
\sloka {děvčátko}  {to děvčátko jak poupátko}
\end{kdyz-jsem-ja-slouzil}
\end{multicols}
\end{document}
```

Soubor zpracujeme příkazem `lualatex priklad-latex` a obdržíme tento výstup:

Když jsem já sloužil to 1. léto,
vysloužil jsem si kuřátko za to.
A to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 2. léto,
vysloužil jsem si kachničku za to.
A ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 3. léto,
vysloužil jsem si husičku za to.
A ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 4. léto,
vysloužil jsem si vepříka za to.
A ten vepř jako pepř
a ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 5. léto,
vysloužil jsem si telátko za to.
A to tele hubou mele
a ten vepř jako pepř

a ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 6. léto,
vysloužil jsem si kravičku za to.
A ta kráva mléko dává
a to tele hubou mele
a ten vepř jako pepř
a ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 7. léto,
vysloužil jsem si volečka za to.
A ten vůl jako kůl
a ta kráva mléko dává
a to tele hubou mele
a ten vepř jako pepř
a ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil to 8. léto,
vysloužil jsem si botičky za to.
A ty boty do roboty
a ten vůl jako kůl
a ta kráva mléko dává
a to tele hubou mele
a ten vepř jako pepř
a ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka pláče doma v komoře.

Když jsem já sloužil poslední léto,
vysloužil jsem si děvčátko za to.
To děvčátko jak poupátko
a ty boty do roboty
a ten vůl jako kůl
a ta kráva mléko dává
a to tele hubou mele
a ten vepř jako pepř
a ta husa chodí bosa
a ta kačka bláto tlačká
a to kuře krákoře, běhá po dvoře,
má panenka stele postel v komoře.

5 Implementace algoritmu pomocí plain TEXu

V této sekci vytvoříme soubor `kdyz-jsem-ja-slouzil.opm`, který bude implementovat Algoritmus 1 v jazyce plain TEX. Pro usnadnění použijeme formát OpTEX, který přidává nad rámec plain TEXu podporu pro jmenné prostory [7, sekce 2.2.3], jež využijeme při definici lokálních proměnných, a podporu pro vícesloupcovou sazbu [7, sekce 1.5.2], kterou využijeme v ukázkovém souboru.

```
\_codedecl
\sequence {Balíček pro sazbu textu lidové písně ,,Když jsem já
  sloužil`` <1.0.0>}
\_namespace {kdyzjsemjaslouzil}

% Lokální proměnné
\newcount \.pocetslok
\newcount \.cislosloky
\newcount \.cisloradku
```

```
% Pomocné funkce pro práci se seznamy, které jsou zde
% implementovány jako hašová tabulka.
\def \.sdef #1{\expandafter \def \csname #1\endcsname}
\def \.suse #1{\csname #1\endcsname}
% Pomocná funkce pro změnu prvního písmene na verzálku.
\def \.prvnivelke #1{\uppercase \expandafter {#1}}

% Funkce pro uložení jedné sloky
\def \sloka #1#2{%
  \advance \.pocetslok 1
  \.sdef {čtvrtý pád:\the \.pocetslok}{#1}%
  \.sdef {popis:\the \.pocetslok}{#2}%
}

% Funkce pro sazbu textu celé písně
\def \vysazej_pisnicku {%
  \loop
  \ifnum \.cislosloky < \.pocetslok
    % Vysázej rok služby a obdrženou odměnu.
    \advance \.cislosloky 1
    Když jsem já sloužil
    \ifnum \.cislosloky = \.pocetslok
      poslední
    \else
      to \the \.cislosloky.~%
    \fi
    léto,\nl
    vysloužil jsem si
    \.suse {čtvrtý pád:\the \.cislosloky}
    za to.\nl
    % Vysázej popisy odměn za dosavadní roky služby.
    {%
      \.cisloradku = \.cislosloky
      \loop
      % Při popisu odměny pro současný rok začni verzálkou.
      \ifnum \.cisloradku = \.cislosloky
        % V poslední sloce vynech počáteční spojku „A".
        \ifnum \.cislosloky < \.pocetslok A \else
          \expandafter \expandafter \expandafter
          \expandafter \expandafter
          \.prvnivelke
        \fi
```

```
      \else a \fi
      \.suse {popis:\the \.cisloradku}%
      % Při popisu odměny pro první rok přidej čárku.
      \ifnum \.cisloradku = 1 , \fi \nl
      \advance \.cisloradku -1
      \ifnum \.cisloradku > 0 \repeat
    }%
    % Vysázej vypravěčovu domněnku o aktivitách partnerky.
    má panenka
    \ifnum \.cislosloky = \.pocetslok
      stele postel
    \else
      pláče doma
    \fi
    v~komoře.
    \par \medskip
  \repeat
}%
\_endnamespace
\_endcode
```

Následně vytvoříme ukázkový soubor `priklad-optex.tex`. Soubor nejprve načte soubor `kdyz-jsem-ja-slouzil.opm`, který jsme připravili v této sekci, a následně vysází text písně *Když jsem já sloužil*.

```
\input kdyz-jsem-ja-slouzil.opm
\chyph
\fontfam [lm]
\parindent 0pt
\begmulti 2
\sloka {kuřátko}   {to kuře krákoře, běhá po dvoře}
\sloka {kachničku} {ta kačka bláto tlačká}
\sloka {husičku}   {ta husa chodí bosa}
\sloka {vepříka}   {ten vepř jako pepř}
\sloka {telátko}   {to tele hubou mele}
\sloka {kravičku}  {ta kráva mléko dává}
\sloka {volečka}   {ten vůl jako kůl}
\sloka {botičky}   {ty boty do roboty}
\sloka {děvčátko}  {to děvčátko jak poupátko}
\vysazej_pisnicku
\endmulti
\bye
```

Soubor zpracujeme příkazem `optex priklad-optex` a obdržíme výstup ze sekce 4.

Nyní porovnáme implementaci v jazyce expl3 ze sekcí 2 a 3 s implementací v jazyce plain TEX z této sekce. Implementace v jazyce expl3 má takřka dvojnásobnou délku. Délkový rozdíl je způsoben především rozvláčností jazyka expl3, ne složitostí kódu, a dopad na čitelnost kódu je tedy sporný.

Implementace seznamů v plain TEXu není přímočará a vyžaduje opatrnou práci s expanzí. Implementace z této sekce se proto seznamům vyhýbá a nahrazuje je hašovou tabulkou. Díky tomu je implementace v plain TEXu stručná a čitelná, ale není obecná: programátor se musí přizpůsobit omezením plain TEXu a připravit kód na míru řešenému problému, zatímco jazyk expl3 disponuje bohatou knihovnou datových struktur, které je možné použít pro řešení různých problémů.

Recenzent článku navrhl ještě jednodušší řešení v idiomatickém plain TEXu, které implementaci algoritmu nevyčleňuje do samostatného balíčku a které jednotlivé sloky okamžitě sází, aniž by si je předtím ukládalo do datové struktury. Navržené řešení s drobnými úpravami uvádím se svolením recenzenta:

```
\fontfam [lm]
\parskip = 6pt plus 2pt
\parindent = 0pt
\def \sloka #1#2#3{\par
  \ifx^#3^\else
    \edef \popisy {%
      #3\ifx \popisy \empty \else \nl a \fi \popisy
    }%
  \fi
  Když jsem já sloužil #1 léto,\nl
  vysloužil jsem si #2 za to.\nl
  \A \popisy, běhá po dvoře,\nl
  má panenka \cosidela v komoře.\par
}
\def \popisy {}
\def \cosidela {pláče doma }   \def \A {A }
\sloka {to první}  {kuřátko}   {to kuře krákoře}
\sloka {to druhé}  {kachničku} {ta kačka bláto tlačká}
\sloka {to třetí}  {husičku}   {ta husa chodí bosa}
\sloka {to čtvrté} {vepříka}   {ten vepř jako pepř}
\sloka {to páté}   {telátko}   {to tele hubou mele}
\sloka {to šesté}  {kravičku}  {ta kráva mléko dává}
\sloka {to sedmé}  {volečka}   {ten vůl jako kůl}
\sloka {to osmé}   {botičky}   {ty boty do roboty}
\def \cosidela {stele postel } \def \A {}
\sloka {poslední}  {děvčátko}  {To děvčátko jak poupátko}
\bye
```

6 Závěr

Díky programovacímu jazyku expl3 mohou uživatelé TEXu programovat způsobem, na který jsou zvyklí z moderních imperativních programovacích jazyků.

V článku jsem představil modul l3seq jazyka expl3, který poskytuje datovou strukturu seznamu. Možnosti modulu jsem demonstroval na sazbě textu české lidové písně *Když jsem já sloužil*. Dále jsem ukázal, že text písně je možné stručně a čitelně vysázet také v jazyce plain TEX, pokud se programátor přizpůsobí omezenému počtu základních datových struktur v plain TEXu.

Odkazy

1. NOVOTNÝ, Vít. Vysokoúrovňové jazyky pro TEX. *Zpravodaj CSTUGu*. 2022, **32**(1–4), 35–48. Dostupné z DOI: 10.5300/2022-1-4/35.
2. THE LATEX PROJECT. *The expl3 package and LATEX3 programming* [online]. CTAN, 2023-08-29 [cit. 2023-10-05]. Dostupné z: https://tug.ctan.org/macros/latex/contrib/l3kernel/expl3.pdf.
3. THE LATEX PROJECT. *The LATEX3 interfaces* [online]. CTAN, 2023-08-29 [cit. 2023-10-05]. Dostupné z: https://tug.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf.
4. STARÝ N., Vít. Nápadovník jmen pro tvůrčí psaní v LuaTEXu. *Zpravodaj CSTUGu*. 2023, **33**(1–2), 3–38. Dostupné z DOI: 10.5300/2023-1-2/3.
5. HROUDOVÁ, Eva. *Když jsem já sloužil* [online]. ProMaminky.cz, 2015-06-20 [cit. 2023-08-13]. Dostupné z: https://www.promaminky.cz/pisnicky/lidove-36/kdyz-jsem-ja-slouzil-287.
6. STARÝ N., Vít. *Sazba textu české lidové písně „Když jsem já sloužil“ pomocí modulu l3seq jazyka expl3: Release The latest version* [online]. GitHub, 2023-09-10 [cit. 2023-09-10]. Dostupné z: https://github.com/Witiko/typesetting-czech-folksong-with-l3seq/releases/tag/latest.
7. OLŠÁK, Petr. *OpTEX: Format Based on Plain TEX and OPmac* [online]. CTAN, 2023-05-25 [cit. 2023-09-18]. Dostupné z: https://mirrors.ctan.org/macros/optex/doc/optex-doc.pdf. Verze 1.12.

Summary: Typesetting Lyrics of Czech Folksong “Když jsem já sloužil” using the l3seq Module of Expl3 Language

The language of plain TEX was developed for typesetting books and only became a Turing-complete programming language at the end of its development. Whereas writing and designing documents is straightforward in plain TEX, programming is difficult due to a lack of basic data structures and the delayed macro expansion, which is different from modern imperative programming languages.

In the LuaTEX engine, authors can also program in the imperative programming language of Lua. Although Lua does not share the limitations of plain TEX, passing data between TEX and Lua is not straightforward and important information such as token category codes are lost in transit.

The expl3 programming language combines the best of both worlds and allows authors to program in TEX in a way that is similar to modern imperative programming languages.

In this article, I introduce the l3seq module of the expl3 language that provides the list data structure. Using l3seq, I typeset the lyrics of the Czech folksong *Když jsem já sloužil.* I also compare the l3seq implementation with plain TEX.

Keywords: LATEX3, expl3, l3seq, xparse, plain TEX, OpTEX, song lyrics

Vít Starý Novotný, witiko@mail.muni.cz

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Jan Šustek
Počet výtisků:	250
Uzávěrka:	30. 11. 2023
Odpovědný redaktor:	Jan Šustek
Redakční rada:	Pavel Haluza, Lukáš Novotný, Vít Starý Novotný, Michal Růžička a Jan Šustek (šéfredaktor)
Vědecká rada:	Ján Buša (předseda), Jiří Demel, Jaromír Kuben (zástupce předsedy), Jiří Rybička a Petr Sojka
Technická redakce:	Vít Starý Novotný
Jazyková korektura:	Zbyněk Michálek
Evidenční číslo MK:	E 7629
Adresa:	CSTUG, Nejedlého 373/1, 638 00 Brno
Email:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz`

www server sdružení:
`https://www.cstug.cz`

CONTENTS

Vít Starý Novotný: Editorial 61

Vít Starý Novotný: CSTUG at the TUG 2023 Conference 63

Jan Šustek: On Generating Documented Source Code by Blocks in TEX . 66

Jan Šustek: How to Enable Page Breaks in Embedded Images 102

Vít Starý Novotný: Markdown 3: What's New, What's Next? 111

Ondřej Sojka, Petr Sojka, Jakub Máca: A Roadmap for Universal Syllabic Segmentation 125

Barbara Beeton: What Every (LA)TEX Newbie Should Know 139

Vít Starý Novotný: Typesetting Lyrics of Czech Folksong "Když jsem já sloužil" using the l3seq Module of Expl3 Language 153