CSTUG

soubor	aktuální verze
`dvitype.web`	3.6
`gftodvi.web`	3.0
`gftopk.web`	2.4
`gftype.web`	3.1
`mf.web`	2.71828182
`mft.web`	2.1
`pltotf.web`	3.6
`pooltype.web`	3.0
`tangle.web`	4.6
`tex.web`	3.141592653
`tftopl.web`	3.3
`vftovp.web`	1.4
`vptovf.web`	1.6
`weave.web`	4.5

ZPRAVODAJ

Československého sdružení uživatelů TeXu

1–4

2021

ISSN 1211-6661
Tištěná verze

ISSN 1213-8185
Online verze

Ročník 31

OBSAH

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

Petr Sojka

Milé čtenářky a čtenáři, TEXistky a TEXisté,

je mi potěšením, že se redakci Zpravodaje daří získávat odborné články vyjadřující poslání sdružení, tedy kvalitní typografii systémem TEX, že se stále nástroje a technologie posunují a můžete se o nich na stránkách tohoto čísla dozvědět, že i naše malá komunita k nim přispívá nemalým dílem, a že k nim také mohu přidat pár slov.

Články tohoto čísla ukazují, že sázecí systém TEX se má stále čile k světu, spěje k dokonalosti, jeho technologie jsou široce používány, stále adaptovány a zdokonalovány, a upevňuje si pozici pevného bodu ve světě vědeckého publikování.

Kolaborativní práce na dokumentech výrazně urychluje učicí křivku počítačové typografie sázecím systémem TEX. Systém Overleaf toto umožňuje: ve sdíleném úložišti v reálném čase sázet dokumenty, sdílet plnou instalaci TEXu včetně historických verzí distribuce TEXlive, nebo jen sdílet dokument s možností ho připomínkovat. Již šest miliónů uživatelů systém používá, a v prvním článku čísla o něm a jeho použití na Fakultě informatiky Masarykovy univerzity (FI MU) píše Vítek Novotný. Jako dlouholetý uživatel systému ho mohu pro kolaborativní editaci společných článků v LATEXu jen doporučit. Doufám, že jeho použití najdou i čtenáři tohoto čísla.

Číslo pokračuje článkem shrnujícím pohledem Petra Olšáka vše potřebné pro TEXového, ε-TEXového, XETEXového či LuaTEXového programátora. Potěší všechny, kteří chtějí mít na úrovni TEXu vše pod kontrolou, a porozumět do detailů všem primitivům, které jsou utajeny pod slupkou logického značkování LATEXu či ConTEXtu. Jako vedlejší produkt článku vznikl styl sazby článku do Zpravodaje v OPTEXu, tak snad se brzy dočkáme dalších článků v tomto formátu.

Autor TEXu po sedmi letech prošel 250, Karlem Berrym již předtřízených, chybových reportů k TEXu a METAFONTu! Grand Wizard ve svém článku připomíná svou neměnnou filosofii vývoje obou programů, komentuje je a informuje o vystavení šeků odměn za nalezení chyb. Jde o částky v součtu v řádu tisíců dolarů – tak velká touha po dokonalosti se tu zrcadlí, a tak složité je psaní rozsáhlého software! Máme novou verzi obou programů, o jedno desetinné místo bližší k π resp. e. Snad je 957. chyba již poslední, odhalení dalších lze případně čekat v roce 2029!

Se svými celoživotními zkušenostmi z praxe člena technické podpory Americké matematické společnosti se dělí Barbara Beeton v článku „Ladění LATEXových souborů“. Navržené postupy a nástroje pro diagnostiku chyb uvítá každý autor netriviálních dokumentů. Není potřeba znovunalézat kolo!

doi: 10.5300/2021-1-4/1

Jazyk Markdown si získal mezi značkovacími jazyky široké užití zejména mezi programátory pro svou kompaktnost. Balíček Markdown již pět let umožňuje milovníkům Markdownu a TEXovým mágům přímou sazbu v LATEXovém prostředí. Nové rozšíření balíčku umožňuje definovat témata, styly chování, a jejich nastavení (snippety) aplikovatelné na části textu. Jen houšť!

S Vítkovým článkem souvisí i další článek dalšího studenta FI MU, Dominika Reháka. Navrhovaná řešení umožní s typografickou kvalitou TEXu s využitím balíčku Markdown sázet dokumenty všech dokumentových formátů podporovaných konvertorem *Pandoc*. To může podpořit kvalitní typografii systémem TEX u široké palety autorů píšících v HTML, v jednoduchých značkovacích jazycích v jednoduchých textových editorech, či v méně zdatných sázecích systémech podporovaných programem *Pandoc*.

Miniknihy, povolené taháky, leporela, skládačky vám pomůže vytvořit přeložený článek Petera Wilsona. Toto jedenácté pokračování Wilsonova seriálu Zpravodajové čtyřčíslo uzavírá. Nechť vám je tvorba vlastních miniaturních knížek a četba celého čísla potěšením.

Letošní vlny kovidu stále nepřejí osobnímu setkání na každoroční valné hromadě. Díky pozměněným stanovám se tedy po necelém roce po Novém roce sejdeme online. Detaily budou včas rozeslány elektronicky. Valná hromada bude volební, budete si volit nový výbor. Hledáme další nové, neunavené, nadšené kandidáty do výboru, kteří by naši malou československou TEXovou loďku tlačili další tři roky. Podobně, podělte se s ostatními o své projekty a nápady nabídkou přednášky při valné hromadě nebo článku do Zpravodaje. Pište nám, prosím, na adresu listu výboru nebo mně osobně. Brzy na viděnou!

Summary: Introductory Word

Go forth and participate in CSTUG to make the bright future of TEX & Friends a reality! *You can!*

Masarykova univerzita, Fakulta informatiky, Botanická 68a, 602 00 Brno
sojka@fi.muni.cz

Overleaf: Kolaborativní webový editor LaTeXu

Vít Novotný

Předseda TUGu označil kolaborativní webový editor Overleaf za „jednu z nejdůležitějších změn ve světě TeXu za poslední roky“. V článku představuji Overleaf a jeho klíčové funkce z pohledu uživatele a uvádím změny plánované do budoucna.

Klíčová slova: textový editor, Overleaf, LaTeX

Úvod

Overleaf je svobodný [1] webový editor, který umožňuje snadnou spolupráci více autorů na LaTeXových dokumentech a který předseda TUGu Boris Veytsman označil za „jednu z nejdůležitějších změn ve světě TeXu za poslední roky“ [2]. Pro členy CSTUGu může být Overleaf odpovědí na otázku, kterou si Sdružení klade již od roku 1991. Tehdy Oldřich Ulrych v historicky prvním čísle Zpravodaje (tehdy TeX bulletinu) popsal, co by měl splňovat editor, který by umožnil „přípravu textů všem v příjemném prostředí“ [3]. Sdružení pro své členy během následujících tří let vybralo [4] a zakoupilo [5] editor CSED od Pavla Ševečka. Cesta to ale byla dlouhá a ukázala mimo jiné to, že neexistuje shoda ani na požadovaných funkcích editoru [6] ani na podpoře nesvobodného software, jako byl CSED [7].

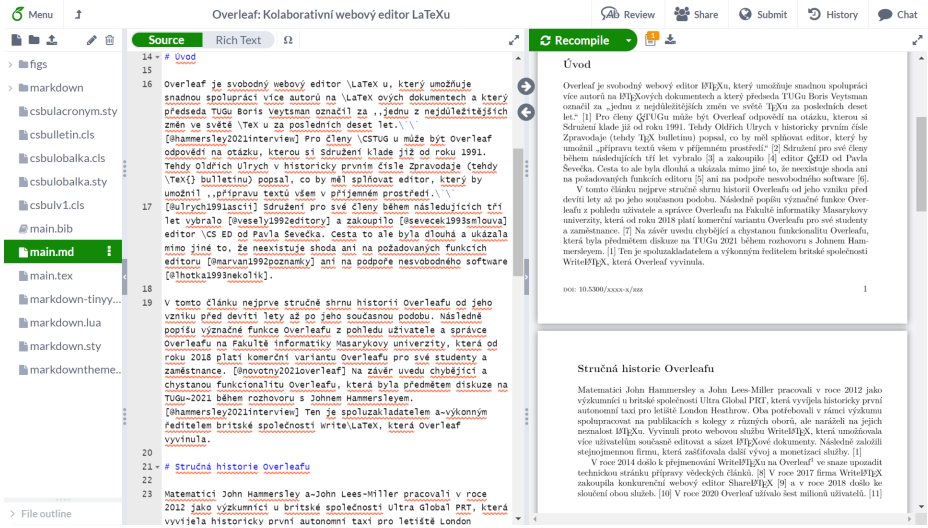
V tomto článku nejprve stručně shrnuji historii Overleafu od jeho vzniku před devíti lety až po jeho současnou podobu. Následně popisuji význačné funkce Overleafu z pohledu uživatele a správce Overleafu na Fakultě informatiky Masarykovy univerzity (FI MU), která od roku 2018 platí komerční variantu[1] Overleafu pro své studenty a zaměstnance [8]. Na závěr uvádím chybějící a chystanou funkcionalitu Overleafu, která byla předmětem diskuze na TUGu 2021 během rozhovoru s Johnem Hammersleyem [2], který je jedním ze zakladatelů Overleafu.

Stručná historie Overleafu

Matematici John Hammersley a John Lees-Miller pracovali jako výzkumníci u britské společnosti Ultra Global PRT, která vyvíjela historicky první autonomní taxi pro letiště London Heathrow. Oba potřebovali v rámci výzkumu spolupracovat

[1] Komerční variantu Overleafu vyzkoušíte na adrese `https://overleaf.com/`. Oproti self-hostingu svobodné varianty může ušetřit pořizovací, provozní a mzdové náklady. FI MU letos zaplatila 6 768 EUR za tisíc Pro licencí, viz `https://overleaf.com/user/subscription/plans`.

DOI: 10.5300/2021-1-4/3



Obrázek 1: Uživatelské rozhraní webového editoru Overleaf

na publikacích s kolegy z různých oborů, ale naráželi na jejich neznalost LaTeXu. V roce 2012 proto vyvinuli webovou službu WriteLaTeX, která umožňovala více uživatelům současně editovat a sázet LaTeXové dokumenty. Následně založili stejnojmennou firmu, která zašťiťovala další vývoj a monetizaci služby [2].

V roce 2014 došlo k přejmenování WriteLaTeXu na Overleaf[2] ve snaze upozadit technickou stránku přípravy vědeckých článků [9]. V roce 2017 firma WriteLaTeX zakoupila konkurenční webový editor ShareLaTeX [10] a v roce 2018 došlo ke sloučení obou služeb. V roce 2020 Overleaf užívalo šest milionů uživatelů [11].

Overleaf dnes

Od svého vzniku v roce 2012 získal Overleaf mnoho užitečných funkcí. V této sekci rozebírám pět z nich, které považuji z pohledu uživatele za obzvlášť významné. Na Obrázku 1 vidíte základní rozhraní Overleafu, na které se následně odkazuji.

Automatická sazba

Po každé změně LaTeXového dokumentu v levé části rozhraní dojde volitelně k překladu a aktualizaci náhledu v pravé části rozhraní. Interně používá Overleaf nástroj LaTeXMk a podobného efektu docílíte lokálně příkazem `latexmk -pvc`.

[2] *Overleaf* je anglický termín pro obsah, který pokračuje na opačné straně listu.

Obrázek 2: Vstupní hala Fakulty informatiky Masarykovy univerzity s plakátem Grand Wizarda (nalevo) a poutačem na webový editor Overleaf (napravo)

Tři způsoby práce

Ve webovém editoru je možné užívat dva způsoby práce: přímé úpravy zdrojového kódu (*Source*) a grafický režim (*Rich Text*). První režim se zaměřuje na pokročilé uživatele a umožňuje i emulovat klávesové zkratky a základní funkce editorů Vim a Emacs. Druhý režim se zaměřuje na uživatele zvyklé pracovat s textovými procesory, jako jsou Microsoft Word, Apache LibreOffice a Dokumenty Google.

Pro nejpokročilejší uživatele, kteří chtějí využívat svůj oblíbený desktopový editor (např. GSED), je určené gitové přemostění [12]. Díky němu mohou uživatelé psát dokumenty offline a úpravy posléze zveřejnit příkazem `git push`.

Reprodukovatelný překlad

Od roku 2020 Overleaf umožňuje pro každý dokument zvlášť uvést, pod jakou verzí distribuce TEX Live se má překládat [13]. To dává uživatelům jistotu, že jejich dokumenty si zachovají svou podobu i do budoucna.

Technicky je přepínání mezi verzemi řešené kontejnerovou virtualizací nástrojem Docker. Administrátoři svobodné varianty Overleafu se tedy nemusí obávat, že by museli udržovat několik souběžných instalací distribuce TEX Live. Přidání nové verze distribuce začíná a končí stažením příslušného dockerového obrazu.

Dokumentace LATEXu

Firma WriteLATEX zaměstnává experty na LATEX, kteří připravují uživatelskou dokumentaci na adrese `https://overleaf.com/learn`. Tato dokumentace je navzdory své přístupnosti poměrně obsáhlá a kromě Overleafu a LATEXu se věnuje

např. i balíčkům, které experti pokládají za užitečné, a pokročilejším tématům, jako jsou makroprogramování, model box-penalty-glue a stroj LuaTEX.

Galerie šablon, preprintové archivy a vědecké časopisy
Overleaf obsahuje galerii dokumentových šablon na adrese `https://overleaf.com/latex/templates`. Uživatelé mohou v galerii uveřejnit libovolný LaTeXový dokument přes tlačítko *Submit* v pravé horní části rozhraní, viz Obr. 1. FI MU používá galerii ke zveřejnění šablon svým studentům a zaměstnancům, viz Obr. 2.

Overleaf od roku 2014 rozvíjí spolupráci s preprintovými archivy a vědeckými časopisy. Přes tlačítko *Submit* je možné dokument zaslat do časopisů nakladatelství Springer, AMS a dalších. Editoři časopisu mohou obsah dokumentu komentovat a navrhovat změny přes záložku *Review* v pravé horní části rozhraní.

Overleaf zítra

Na TUGu 2021 proběhl rozhovor s Johnem Hammersleyem, během kterého se kromě stavebnic LEGO a vesmírného závodu řešila i budoucnost Overleafu [2].

Plnohodnotný offline režim
Jednou z významných výhod gitového přemostění je možnost práce bez internetového připojení. Tuto možnost by měl v budoucnosti podporovat i webový editor, první kroky k přidání podpory by měly proběhnout v roce 2022. To ocení především uživatelé mobilních zařízení. Na stole je i příprava mobilní aplikace.

Podpora dalších formátů TeXu
Overleaf se zaměřuje na přípravu vědeckých článků, kde je LaTeX bezesporu dominantním formátem. Absence podpory pro další TeXové formáty, jako jsou plain TeX, ConTeXt nebo OPTeX, však uměle omezuje konkurenceschopnost těchto formátů. Přidání podpory pro další formáty se však zatím neplánuje.

Podpora sdružení uživatelů TeXu
Šest milionů uživatelů Overleafu závisí na národních sdruženích uživatelů TeXu, která poskytují školení, uživatelskou podporu a klíčovou infrastrukturu, jako jsou webová zrcadla balíčkového repozitáře CTAN. Frank Mittelbach navrhl, že by Overleaf měl finančně podporovat sdružení uživatelů TeXu pro zachování udržitelnosti ekosystému TeXu, což je ve společném zájmu Overleafu a sdružení.

Otevřený formát pro reprodukovatelný překlad
Kontejnerová virtualizace pomocí Dockeru umožňuje Overleafu svázat jednotlivé dokumenty s konkrétními verzemi distribuce TEX Live. To umožňuje reprodukovatelný překlad dokumentů, ale vyžaduje neprakticky velké dockerové obrazy.

Britský TEXista Jonathan Fine představil projekt přenositelných TEXových dokumentů (PTD) [14], které by kromě zdrojového textu a přidružených dat obsahovaly i jednoznačné identifikátory všech dalších souborů potřebných pro překlad.[3] Spolu s webovým rozhraním pro resoluci identifikátorů by pak bylo možné rychle a reprodukovatelně nahlížet TEXové dokumenty nezávisle na Dockeru a Overleafu. Fine o PTD také přednášel na letošní konferenci PackagingCon [19].

Odkazy

1. OVERLEAF CONTRIBUTORS. *An open-source online real-time collaborative LaTeX editor* [online]. 2021-11-18 [cit. 2021-11-18]. Dostupné z: `https://github.com/overleaf/overleaf`.
2. HAMMERSLEY, John; DE SOUZA, Paulo Ney. *John Hammersley: Inverview* [online]. 2021-08-08 [cit. 2021-11-18]. Dostupné z: `https://youtu.be/WN3CsHI0r5Y?t=10h33m48s`.
3. ULRYCH, Oldřich. ASCII Editor pro TEX. *TEX bulletin*. 1991, roč. 1, č. 1, s. 10–12. ISSN 1211-6661. Dostupné z DOI: `10.5300/1991-1/10`.
4. VESELÝ, Jiří. Editory. *Zpravodaj CSTUGu*. 1992, roč. 2, č. 1, s. 36–40. ISSN 1211-6661. Dostupné z DOI: `10.5300/1992-1/36`.
5. ŠEVEČEK, Pavel. Smlouva o užívání díla CSED. *Zpravodaj CSTUGu*. 1993, roč. 3, č. 2, s. 88–91. ISSN 1211-6661.
6. MARVAN, M.; DEMEL, J. Poznámky ke koupi editoru. *Zpravodaj CSTUGu*. 1992, roč. 2, č. 2, s. 88–92. Dostupné z DOI: `10.5300/1992-2/88`.
7. ŠEVEČEK, Pavel. Několik poznámek nejen k výroční ceně CSTUGu. *Zpravodaj CSTUGu*. 1993, roč. 3, č. 2, s. 85–86. ISSN 1211-6661.
8. NOVOTNÝ, Vít. *Overleaf, webový editor LaTeXových dokumentů* [online]. Fakulta informatiky Masarykovy univerzity, 2021-11-18 [cit. 2021-11-18]. Dostupné z: `https://www.fi.muni.cz/tech/overleaf.html.cs`.
9. OVERLEAF TEAM. *WriteLaTeX is continued Overleaf* [online]. 2014-12-16 [cit. 2021-11-18]. Dostupné z: `https://overleaf.com/blog/190-writelatex-is-continued-overleaf`.

[3] Problém ruční reprodukovatelné sazby LaTeXových dokumentů řeší i Barbara Beeton ve svém článku [15], jehož překlad naleznete v tomto čísle. Automatické dohledání souborů potřebných pro překlad řeší Perlový program TEX Versioning System (TVS) [16, 17, 18], který vznikl v roce 1998 jako výstup bakalářské práce Davida Antoše.

10. *Exciting News – ShareLaTeX is joining Overleaf* [online]. 2017-07-20 [cit. 2021-11-18]. Dostupné z: https://overleaf.com/blog/518-exciting-news-sharelatex-is-joining-overleaf.
11. OVERLEAF TEAM. *Reaching six million users in an unprecedented six months* [online]. 2020-07-01 [cit. 2021-11-18]. Dostupné z: https://overleaf.com/blog/reaching-six-million-users-in-an-unprecedented-six-months.
12. OVERLEAF TEAM. *The Git bridge in Overleaf v2 is here!* [online]. 2019-01-03 [cit. 2021-11-18]. Dostupné z: https://overleaf.com/blog/the-git-bridge-in-overleaf-v2-is-here.
13. OVERLEAF TEAM. *New Feature: Select your TeX Live Compiler Version* [online]. 2020-04-30 [cit. 2021-11-18]. Dostupné z: https://overleaf.com/blog/new-feature-select-your-tex-live-compiler-version.
14. FINE, Jonathan. *PTD* [online] [cit. 2021-07-12]. Dostupné z: https://github.com/arxtex/ptd.
15. BEETON, Barbara. Debugging LaTeX files: Illegitimi non carborundum. *TUGboat* [online]. 2017, roč. 38, č. 2, s. 159–164 [cit. 2021-12-06]. Dostupné z: http://www.vim.tug.org/TUGboat/tb38-2/tb119beet.pdf.
16. ANTOŠ, David. *TVS: TeX Versioning System* [online]. 1998 [cit. 2021-12-07]. Dostupné z: https://www.ctan.org/pkg/tvs.
17. ANTOŠ, David. *TeX Versioning System* [online]. Fakulta informatiky Masarykovy univerity, 2000 [cit. 2001-06-16]. Dostupné z: http://www.fi.muni.cz/~xantos/TVS/.
18. ANTOŠ, David. TeX Versioning System aneb jak všechny zdrojové soubory uložiti. *Zpravodaj Československého sdružení uživatelů TeXu*. 2000, roč. 10, č. 1–3, s. 47–49. Dostupné z DOI: 10.5300/2000-1-3/47.
19. FINE, Jonathan. *Tools for packaging and using Portable TeX Documents* [online]. 2021-11-10 [cit. 2021-11-18]. Dostupné z: https://pretalx.com/packagingcon-2021/talk/XTAJ7Z/.

Summary: Overleaf, Collaborative Online LaTeX Editor

The president of TUG named the collaborative online editor Overleaf "one of the several most important changes in the TeX world for the last years". In this article, I introduce Overleaf and describe its key functions and planned features.

Keywords: text editor, Overleaf, LaTeX

Vít Novotný, witiko@mail.muni.cz

TEX in a Nutshell

Petr Olšák

Nowadays, many users discover TEX through high-level formats that hide the complexity of typesetting behind a facade of a friendly markup language. However, all except the simplest of typesetting tasks require that the user can understand what happens under the hood and knows how they can influence the algorithms of TEX when needed.

In this article, the author introduces the foundations of most high-level TEX formats, which will help the readers with their day-to-day work with TEX as well as their more difficult typesetting tasks. The readers are first introduced to the program TEX and its extensions. Then, they learn about the different processors of TEX and their modes. Finally, the readers learn about the registers and primitive commands of TEX as well as the macros of the plain TEX format. The word of the day is brevity as the exposition spans less than forty pages: An excellent reading material for an otherwise uneventful train ride!

The author has previously written three books about TEX, has developed the OpTEX format, maintains a dozen package on the CTAN archive, and has taught a university course about TEX for over twenty years.

Keywords: TEX, ε-TEX, pdfTEX, XeTEX, LuaTEX, microtypography, plain TEX

Pure TEX features are described here, no features provided by macro extensions. Only the last section gives a summary of plain TEX macros.

The main goal of this document is brevity. So features are described only roughly and sometimes inaccurately here. If you need to know more then you can read free available books, for example TEX by topic or TEXbook naruby. Try to type `texdoc texbytopic` in your system.

The OpTEX manual supposes that the user already knows the basic principles of TEX itself. If you are converting from LATEX to OpTEX for example[1] then you may welcome a summary document that presents these basic principles because LATEX manuals typically don't distinguish between TEX features and features specially implemented by LATEX macros.

I would like to express my special thanks to Barbara Beeton who read my text very carefully and suggested hundreds of language corrections and improvements and also discovered many of my real mistakes. Thanks to her, my text is better. But if there are any other mistakes then they are only mine and I'll be pleased if you send me a bug report in such case.

This article has been republished from CTAN with minor changes and the addition of the abstract with the permission of the author. Our changes do not alter the content of the article.

[1] Congratulations on your decision :-)

Table of contents

1 Terminology

The main principle of TEX is that its input files can be a mix of the material which could be printed and *control sequences* which give a setting for built-in algorithms of TEX or give a special message to TEX what to do with the inputted material.

Each control sequence (typically a word prefixed by a backslash) has its *meaning*. There are four types of meanings of control sequences:

- the control sequence can be a *register*; this means it represents a variable which is able to keep a value. There are *primitive registers*. Their values influence behavior of built-in algorithms (e.g., `\hsize`, `\parindent`, `\hyphenpenalty`). On the other hand *declared registers* are used by macros (e.g., `\medskipamount` used in plain TEX or `\ttindent` used by OpTEX).
- the control sequence can be a *primitive command*, which runs a built-in algorithm (e.g., `\def` declares a macro, `\halign` runs the algorithm for tables, `\hbox` creates a box in typesetting output).
- the control sequence can be a *character constant* (declared by `\chardef` or `\mathchardef` primitive command) or a font selector (declared by `\font` primitive command).
- the control sequence can be a *macro*. When it is read, it is replaced by its *replacement text* in the input queue. If there are more macros in the replacement text, all macros are replaced. This is called the *expansion process* which ends when only printable text, primitive commands (listed in section 13), registers (section 11), character constants, or font selectors remain.

Example. When TEX reads:

```
\def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX}
```

in a macro file, then the `\def` primitive command saves the information that `\TeX` is a control sequence with meaning "macro", the replacement text is declared here, and it is a mix of a material to be typeset: `T`, `E` and `X` and primitive commands `\kern`, `\lower`, `\hbox` with their parameters in given syntax. Each primitive command has a declared syntax; for example, `\kern` must be followed by a dimension specification in the format "decimal number followed by a unit". More about this primitive syntax is in sections 11, 12 and 13.

When a control sequence `\TeX` with meaning "macro" occurs in the input stream, then it is *expanded* to its replacement text, i.e. the sequence of typesetting material and primitive commands. The `\TeX` macro expands to `T\kern-.1667em\lower.5ex\hbox{E}\kern -.125emX` and the logo TEX is printed as a result of this processing.

None of the control sequences have their definitive meaning. A control sequence could change its meaning by re-defining it as a new macro (using `\def`), redeclaring it as an arbitrary object in TEX (using `\let`), etc. When you re-define a primitive control sequence then the access to its value or built-in algorithm is lost. This is a reason why OpTEX macros duplicate all primitive sequences (`\hbox` and `\_hbox`) with the same meaning and use only "private" control sequences (prefixed by `_`). So, a user can re-define `\hbox` without the loss of the primitive command `\_hbox`.

2 Formats, engines

TEX is able to start without any macros preloaded in the so-called *ini-TEX state* (the `-ini` option on the command line must be used). It already knows only primitive registers and primitive commands at this state.[2] When ini-TEX reads macro files then new control sequences are declared as macros, declared registers, character constants or font selectors. The primitive command `\dump` saves the binary image of the TEX memory (with newly declared control sequences) to the *format file* (`.fmt` extension).

The original intention of existing format files was to prepare a collection of macro declarations and register settings, to load default fonts, and to dump this information to a file for later use. Such a collection typically declares macros for the markup of documents and for typesetting design. This is the reason why we

[2] Roughly speaking, if you know all these primitive objects (about 300 in classical TEX, 700 in LuaTEX) and the syntax of all these primitive commands and all the built-in algorithms, then you know all about TEX. But starting to produce ordinary documents from this primitive level without macro support is nearly impossible.

call these files *format files*: they give a format of documents on the output side and declare markup rules for document source files.

When TEX is started without the `-ini` option, it tries to load a prepared format file into its memory and to continue with reading more macros or a real document (or both). The starting point is at the place where `\dump` was processed during the ini-TEX state. If the format file is not specified explicitly (by `-fmt` option on the command line) then TEX tries to read the format file with the same name which is used for running TEX. For example `tex document` runs TEX, it loads the format `tex.fmt` and reads the `document.tex`. Or `latex document` runs TEX, it loads the format `latex.fmt` and reads the `document.tex`.

The `tex.fmt` is the format file dumped when *plain TEX macros*[3] were read, and `latex.fmt` is the format file dumped when *LATEX macros* were read. This is typically done when a TEX distribution is installed without any user intervention. So, the user can run `tex document` or `latex document` without worry that these typical format files exist.

From this point of view, LATEX is nothing more than a format of TEX, i.e. a collection of macro declarations and register settings.

A typical TEX distribution has four common *TEXengines*, i.e. programs. They implement classical TEX algorithms with various extensions:

- TEX – only classical TEX algorithms by Donald Knuth,
- pdfTEX – an extension supporting PDF output directly and micro-typographical features,
- XeTEX – an extension supporting Unicode and PDF output,
- LuaTEX – an extension supporting Lua programming, Unicode, micro-typographical features and PDF output.

Each of them is able to run in ini-TEX state or with a format file. For example the command `luatex -ini macros.ini` starts LuaTEX at ini-TEX state, reads the `macros.ini` file and the final `\dump` command is supposed here to create a format `macros.fmt`. Then a user can use the command `luatex -fmt macros document` to load `macros.fmt` and process the `document.tex`. Or the command `luatex document` processes LuaTEX with `document.tex` and with `luatex.fmt` which is a little extension of plain TEX macros. Another example: `lualatex document` runs LuaTEX with `lualatex.fmt`. It is a format with LATEX macros for LuaTEX engine. Final example: `optex document` runs LuaTEX with `optex.fmt` which is a format with OpTEX macros.

[3] Plain TEX macros were made by Donald Knuth, the author of TEX. It is a set of basic macros and settings which is used (more or less) as a subset of all other macro packages.

3 Searching data

If TEX needs to read something from the file system (e.g. the primitive command `\input` ⟨*file name*⟩ or `\font` ⟨*font selector*⟩ `=` ⟨*file name*⟩ is used) then the rule "first wins" is applied. TEX looks at the current directory first or somewhere in the TEX installation second. The behavior in the second step depends on the used TEX distribution. For example TEXlive programs are linked with a *kpathsea* library and they do the following: Search for the given file in the current directory, then in the `~/texmf` tree (data are saved by the user here), then in the `texmf-local` tree (data are saved by the system administrator here; they are not removed when the TEX distribution is upgraded), then in `texmf-var` tree (data are saved automatically by programs from the TEX distribution here), and then in the `texmf-dist` tree (data from the TEXlive distribution). Each directory tree can be divided into sub-trees: first level `tex`, `fonts`, `doc`, etc.; the second level is divided by TEX engines or font types, etc.; more levels are typically organized to keep clarity. New files in the current directory or in the `~/texmf` tree are found without doing anything more, but new files in other places have to be registered by the `texhash` program (TEX distributions do this automatically during their installation).

4 Processing the input

The lines from input files are first transformed by the *tokenizer*. It reads input lines and generates a sequence of tokens. These are the main goals of the tokenizer:

- It converts each control sequence to a single token characterized by its name.
- Other input material is tokenized as "one token per character".
- A continuous sequence of multiple spaces is transformed into one space token.
- The end of the line is transformed into a space token, so that paragraph text can continue on the next input line and one space token is added between the last word on the previous line and the first word on the next line.
- The comment character `%` is ignored and all the text after it to the end of line is ignored too. No space is generated at the end of this line.
- Spaces from the begining of each line are ignored. Thus, you can use arbitrary indentation in your source file without changing the result.
- Each empty line (or line with only spaces) is transformed to the token `\par`. This token has primitive meaning: "finalize the current paragraph". This implies the general rule in TEX source files: paragraphs are terminated by empty lines.

The behavior of the tokenizer is not definitive. The tokenizer works with a table of category codes. Any change of category codes of characters (done by the primitive command `\catcode‘\` ⟨*character*⟩ `=` ⟨*code*⟩) influences tokenizer

processing. For example, the verbatim environment is declared using setting all characters to normal meaning.

By default, there are the following characters with special meaning. The tokenizer converts them or sets them as special tokens used in syntactic rules in TEX later. The corresponding category codes are mentioned here as an index of the character.

- `\`$_0$ – starts completion of a control sequence by the tokenizer.
- `{`$_1$ and `}`$_2$ – open and close group or have special syntactic meaning. The main syntactic rule is: each subsequence of tokens treated by macros or primitive commands must have these pairs of tokens balanced. There is no exception. The tokenizer treats them as special tokens with meaning "opening character$_1$" and "closing character$_2$".
- `%`$_{14}$ – comment character, removed by the tokenizer, along with everything that follows it on the line.
- `$`$_3$, `&`$_4$, `#`$_6$, `^`$_7$, `_`$_8$, `~`$_{13}$ – tokenizer treats them as a special tokens with meaning: "math-mode selector$_3$", "table separator$_4$", "parameter prefix for macros$_6$", "superscript prefix in math$_7$", "subscript prefix in math$_8$", "active character$_{13}$" (the active character `~` is defined as no-breakable space in all typical formats).
- Letters and other characters are tokenized as "letter character$_{11}$" or "other character$_{12}$".

If you need to print these special characters you can use `\%`, `\&`, `\$`, `\#` or `\_`. These five control sequences are declared as "print this character" in all typical TEX formats. Another possibility is to use a verbatim environment (it depends on the used format). Last alternative: you can use `\csstring\`⟨*character*⟩ in LuaTEX, because it has the primitive command `\csstring` which converts `\`⟨*character*⟩ to ⟨*character*⟩$_{12}$. The "active character$_{13}$" can be declared by `\catcode‘\`⟨*character*⟩`=13`. Such a ⟨*character*⟩ behaves like a control sequence. For example, you can define it by `\def`⟨*character*⟩`{...}` and use this ⟨*character*⟩ as a macro. If the term ⟨*control sequence*⟩ is used in syntactical rules in this document then it means a real control sequence or an active character. Each control sequence is built by the tokenizer starting from `\`$_0$. Its name is a continuous sequence of letters$_{11}$ finalized by the first non-letter. Note that OpTEX sets `_` as letter$_{11}$, thus control sequence names can include this character. LATEX sets the `@` as letter$_{11}$ when reading styles and macro files. You can look to such files and you will see many such characters inside private control sequence names declared by LATEX macros. If the first character after `\`$_0$ is a non-letter (i.e. ⟨*something*⟩$_{\neq 11}$), then the control sequence is finalized with only this character in its name. So called *one-character control sequence* is created. Other control sequences are *multiletter control sequences*. Spaces $\sqcup_{10}$ after multi-letter control sequences are

ignored, so the space can be used as a terminating character of the control sequence. Other characters used immediately after a control sequence are not ignored. So `\TeX !` and `\TeX!` gives the same result: the control sequence `\TeX` followed immediately by `!`$_{12}$. The tokenizer's output (a sequence of tokens) goes to the *expand processor* and its output goes to the *main processor* of TEX. The expand processor performs expansions of macros or a primitive command which is working at the expand processor level. See a summary of such commands in section 12. The main processor performs assignment of registers, declares macros by the `\def` primitive command, and runs all primitive commands at the main processor level. Moreover, it creates the typesetting output as described in the next section. The very important difference between TEX and other programs is that there are no strings, only sequences of tokens. We can return to the example `\def\TeX{...}` above in section 1. The token `\def` is a control sequence with meaning "declare a macro". It gets the following token `\TeX` and declares it as a macro with replacement text, which is the sequence of tokens:

T	\kern	-	.	1	6	6	7	e	m	\lower	.	5	e	x	\hbox	{	E	}	\kern
11		12	12	12	12	12	12	11	11		12	12	11	11		1	11	2	

-	.	1	2	5	e	m	X
12	12	12	12	12	11	11	11

If you are thinking like TEX then you must forget the term "string" because all texts in TEX are preprocessed by the tokenizer when input lines are read and only sequences of tokens are manipulated inside TEX. The tokenizer converts two `^`$_7$`^`$_7$ characters followed by an ASCII uppercase letter to the Ctrl-letter ASCII code. For example `^^M` is Ctrl-M (carriage return). It converts two `^`$_7$`^`$_7$ followed by two hexadecimal digits (`0123456789abcdef`) to a one-byte code, for example, `^^0d` is Ctrl-M too because it has code 13. Moreover, the tokenizer of XETEX or LuaTEX converts `^`$_7$`^`$_7$`^`$_7$`^`$_7$ followed by four hexadecimal digits or `^`$_7$`^`$_7$`^`$_7$`^`$_7$`^`$_7$`^`$_7$ followed by six hexadecimal digits to one character with a given Unicode.

5 Vertical and horizontal modes

When the main processor creates the typesetting output, it alternates between vertical and horizontal mode. It starts in *vertical mode*: all materials are put vertically below in this mode. For example `\hbox{a}\hbox{b}\hbox{c}` creates a above b above c in vertical mode. If something is incompatible with the vertical mode principle — a special command working only in horizontal mode or a character itself — then the main processor switches to *horizontal mode*: it opens an unlimited horizontal data row for typesetting material and puts material next to each other. For example `\hbox{a}\hbox{b}\hbox{c}` creates abc in horizontal mode. When an empty line is scanned, the tokenizer creates a `\par` token here and if the main processor is in horizontal mode, the `\par` command

finalizes the paragraph. More exactly it returns to vertical mode, it breaks the horizontal data row filled in previous horizontal mode to parts with the `\hsize` width. These parts are completed as *boxes* and they are put one below another in vertical mode. So, a paragraph of `\hsize` width is created. Repeatedly: if there is something incompatible with the current vertical mode (typically a character), then the horizontal mode is opened and all characters (and spaces between them) are put to the horizontal data row. When an empty line is scanned, then the `\par` command is started and the horizontal data row is broken into lines of `\hsize` width and the next paragraph is completed. In vertical mode, the material is accumulated in a vertical data column called the *main vertical list*. If the height of this material is greater than `\vsize` then its part with maximum `\vsize` height is completed as a *page box* and shipped to the *output routine*. A programmer or designer can declare a design of pages using macros in the output routine: header, footer, pagination, the position of the main page box, etc. The output routine completes the main page box with other material declared in the output routine and the result is shipped out as one page of the document. The main processor continues in vertical mode with the rest of the unused material in the main vertical list. Then it can switch to horizontal mode if a character occurs, etc... The plain TEX macro `\bye` (or primitive command `\end`[4]) starts the last `\par` command, finalizes the last paragraph (if any), completes the last page box, sends it to the output routine, finalizes the last page in it, and TEX is terminated. There are *internal vertical mode* and *internal horizontal mode*. They are activated when the main processor is typesetting material inside `\vbox{...}` or `\hbox{...}` primitive commands. More about boxes is in sections 7 and 13. Understanding of switching between modes is very important for TEX users. There are primitive commands which are context dependent on the current mode. For example, the `\par` primitive command (generated by an empty line) does nothing in vertical mode but it finalizes paragraph in horizontal mode and it causes an error in math mode. Or the `\kern` primitive command creates a vertical space in vertical mode or horizontal space in horizontal mode. The following primitive commands used in vertical mode start horizontal mode: the first character of a paragraph (most common situation) or `\indent`, `\noindent`, `\hskip` (and its alternatives), `\vrule` and the plain TEX macro `\leavevmode`[5]. When horizontal mode is opened, an indentation of `\parindent` width is included. The exception is only if horizontal mode is started by `\noindent`; then the paragraph has no indentation. The following primitive commands used in horizontal mode finalize the paragraph and return to vertical mode: `\par`, `\vskip` (and its alternatives), `\hrule`, `\end` and the plain TEX macro `\bye`.

[4] LATEX format re-defines this primitive control sequence `\end` to another meaning which follows the logic of LATEX's markup rules.

[5] The list is not exhaustive, but most important commands are mentioned.

6 Groups in TEX

Each assignment to registers, declaration macros or font selecting is local in groups. When the current group ends then the assignments made inside the group are forgotten and the values in effect before this group was opened are restored. The groups can be delimited by `{`$_1$ and `}`$_2$ pair or by `\begingroup` and `\endgroup` primitive commands or by `\bgroup` and `\egroup` control sequences declared by plain TEX. For example, plain TEX declares the macros `\rm` (selects roman font), `\bf` (selects bold font) and `\it` (selects italics) and it initializes by `\rm` font. A user can write:

```
The roman font is here {\it here is italics} and the
roman font continues.
```

Not only fonts but all registers are set locally inside a group. The macro designer can declare a special environment with font selection and with more special typographical parameters in groups. The following example tests understanding of vertical and horizontal modes.

```
{\hsize=5cm This is the first paragraph which should be
            formatted to 5\,cm width.}

But it is not true...
```

Why does the example above not create the paragraph with a 5 cm width? The empty line (`\par` command) is placed *after* the group is finished, so the `\hsize` parameter has its previous value at the time when the paragraph is completed, not the value 5 cm. The value of the `\hsize` register[6] is used when the paragraph is completed, not at the beginning of the paragraph. This is the reason why macro programmers explicitly put a `\par` command into macros before the local environment is finished by the end of the group. Our example should look like this:

```
{\hsize=5cm This is the first ... to 5\,cm width.\par}
```

7 Box, kern, penalty, glue

You can look at one character, say the `y`. It is represented by three dimensions: height (above baseline), depth (below baseline) and width. Suppose that there are more characters printed in horizontal mode and completed as a line of a paragraph. This line has its height equal to the maximum height of characters inside it, it has the depth equal to maximum depth of all characters inside it and

[6] and about twenty other registers which declare the paragraph design

it has its width. Such a sequence of characters encapsulated as one typesetting element with its height, depth and width is called a *box*. Boxes are placed next to each other (from left to right[7]) in horizontal mode or one below another in vertical mode. The boxes can include individual characters or spaces or boxes. The boxes can include more boxes. Paragraph lines are boxes. The page box includes paragraph lines (boxes). The finalized page with a header, page box, pagination, etc., is a box and it is shipped out to the PDF page. Understanding boxes is necessary for macro programmers and designers. You can create an individual box by the primitive command `\hbox{` ⟨*horizontal material*⟩ `}` or `\vbox{` ⟨*vertical material*⟩ `}`. The ⟨*horizontal material*⟩ is completed in internal horizontal mode and ⟨*vertical material*⟩ in internal vertical mode. Both cases open a group, create the material in a specified mode and close the group, where all settings are local. The ⟨*horizontal material*⟩ can include individual characters, boxes, horizontal *glues* or *kerns*. "Glue" is a special term for stretchable or shrinkable and possibly breakable spaces and "kern" is a term used for fixed nonbreakable spaces. The ⟨*vertical material*⟩ can include boxes, vertical glues or kerns. No individual characters. If you put an individual character in vertical mode (for example in a `\vbox`) then horizontal mode is opened. At the end of a `\vbox`[8] or when the `\par` command is invoked, the opened paragraph is finished (with current `\hsize` width) and the resulting lines are vertically placed inside the `\vbox`. The completed boxes are unbreakable and they are treated as a single object in the surrounding printed material. The line boxes of a paragraph have the fixed width `\hsize`, so there must be something stretchable or shrinkable in order to get the desired fixed width of lines. Typically the spaces between words have this feature.[9] These spaces have declared their *default size*, their *stretchability* and their *shrinkability* in the font metric data of the currently used font. You can place such glue explicitly by the primitive command `\hskip`:

```
\hskip ⟨default size⟩ plus⟨stretchability⟩ minus⟨shrinkability⟩
for example:
\hskip 10pt plus5pt minus2.5pt
```

This example places the glue with 10 pt default size, stretchable to 15 pt[10] and shrinkable to 7.5 pt as its minimal size. All glues in one line are stretched or shrunk equally but with weights given from their stretchability/shrinkability

[7] There is an exception for special languages.

[8] before the `\vbox` group is closed

[9] When the microtypographical feature `\pdfadjustspacing` is activated, then not only spaces are stretchable and shrinkable but individual characters are slightly deformed (by an invisible amount) too.

[10] It can be stretchable ad absurdum (more than 15 pt) but with very considerable *badness* calculated by TEX whenever glues are stretched or shrunk.

values. You can do experiments of this feature if you say `\hbox to` ⟨*size*⟩ `{...}`. Then the `\hbox` is created with a given width. Probably, the glues inside this `\hbox` must be stretched or shrunk. You can see in the log that the total *badness* is calculated, it represents the amount of a "force" used for all glue included in such an `\hbox`. An infinitely stretchable (to an arbitrary positive value) or shrinkable (to an arbitrary negative value) glue can exist. This glue is stretched/shrunk and other glues with finite amounts of stretching or shrinking keep their default size in such case. You can put infinitely stretchable/shrinkable glue using the reserved unit `fil` in an `\hskip` command, for example the command `\hskip 0pt plus 1fil` means zero default size but infinitely stretchable. There is a shortcut for such glue: `\hfil`. When you type `\hbox to\hsize{\hfil` ⟨*text*⟩ `\hfil}` then the ⟨*text*⟩ is centered. But if the ⟨*text*⟩ is wider than `\hsize` then TEX reports an `overfull \hbox`. If you want to center a wide ⟨*text*⟩ too, you can use `\hss` instead of `\hfil`. The `\hss` primitive command is equal to `\hskip 0pt plus1fil minus1fil`. The ⟨*text*⟩ printed by `\hbox to\hsize{\hss` ⟨*text*⟩ `\hss}` is now centered in its arbitrary size. A glue created with `fill` stretchability or shrinkability (double ell) is infinitely more stretchable or shrinkable than glues with only a `fil` unit. So, glues with `fill` are stretched or shrunk and glues with only `fil` in the same box keep their default size. For example, a macro declares centering a ⟨*text*⟩ by `\hbox to\hsize{\hss` ⟨*text*⟩ `\hss}` and a user can create the ⟨*text*⟩ in the form `\hfill` ⟨*real text*⟩. Then ⟨*real text*⟩ is printed flushed right because `\hfill` is a shortcut to `\hskip0pt plus1fill` and has greater priority than glues with only a `fil` unit. Common usage is `\hbox to0pt{` ⟨*text*⟩ `\hss}` or `\hbox to0pt{\hss` ⟨*text*⟩ `}`. The box with zero width is created and the text overlaps the adjacent text to the right (first example) or to the left (second example). Plain TEX declares macros for these cases: `\rlap{` ⟨*text*⟩ `}` or `\llap{` ⟨*text*⟩ `}`. The last line of each paragraph is finalized by a glue of type `\hfil` by default. When you write `\hfill` ⟨*object*⟩ in vertical mode (⟨*object*⟩ is something like a table, image or whatever else in the box) then ⟨*object*⟩ is flushed right, because the paragraph is started by the `\hfill` space but finalized only by `\hfil` space. If you type `\noindent\hfil` ⟨*object*⟩ then the ⟨*object*⟩ is centered. And putting only ⟨*object*⟩ places it to the left side because the common left side is the default placement rule in vertical mode. The same principles that apply to horizontal glues are also applicable to vertical modes where glues are created by `\vskip` commands instead of `\hskip` commands. You can write `\vbox to` ⟨*size*⟩ `{...}` and do experiments. When the paragraph breaking algorithm decides about the suitable breakpoints for creating lines with the desired width `\hsize`, then each glue is a potentially breakable point. Each glue can be preceded by a *penalty* value (created by the `\penalty` primitive) in the typical range −10000 to 10000. The paragraph breaking algorithm gets a penalty if it decides to break line at

the glue preceded by the given penalty value. If no penalty is declared for a given glue, then it is the same as a penalty equal to zero.[11]. The penalty value 10000 or more means "impossible to break". A negative penalty means a bonus for the paragraph breaking algorithm. The penalty −10000 or less means "you must break here". The paragraph breaking algorithm tries to find an optimum of breakpoint positions concerning to all penalties, to all badnesses of all created lines and to many more values not mentioned here in this brief document. The analogous optimal breakpoint is found in vertical material when TEX breaks it into pages. The concept "box, penalty, glue" with the optimum-fit breaking algorithms makes TEX unique among many other typesetting software.

8 Syntactic rules

A primitive command can get its parameters written after it. These parameters must suit syntactic rules given for each primitive command. Some parameters are optional. E.g. `\hskip` ⟨*dimen*⟩ `plus` ⟨*stretchability*⟩ `minus` ⟨*shrinkability*⟩ means that the parameter ⟨*dimen*⟩ must follow (it must suit syntactic rules for dimensions, see section 11) then the optional parameter prefixed by keyword `plus` can follow and then the optional parameter prefixed by `minus` can follow. We denote the optional parameters by underline in this document.

Keywords (typically prefixes to some parameters) may have optional spaces around them.

The explicit expressions of numbers (i.e. `75`, `"4B`, `‘K`; see section 11) should be terminated by one optional space which is not printed. This space can serve as a termination character which says that "whole number is presented here; no more digits are expected".

If the syntactic rule mentions the pair `{`, `}` then these characters are not definitive: other characters may be tokenized with this special meaning but it is not common. The text between this pair must be *balanced* with respect to this pair. For example the syntactic rule `\message{` ⟨*text*⟩ `}` supposes that ⟨*text*⟩ must not be `ab{cd`, but `ab{c{}}d` is allowed for instance.

By default, all parameters read by primitive commands are got from the input stream, tokenized and fully expanded by the expand processor. But sometimes, when TEX reads parameters for a primitive command, the expand processor is deactivated. We denote these parameters by red color. For example,

[11] More precisely: the paragraph breaking algorithm or page breaking algorithm can break horizontal list to lines (or vertical list to pages) at penalties (then it gets the given penalty) or at glues (then the penalty is zero). The second case is possible only if no penalty nor glue precedes. The item where the list is broken (penalty or glue), is discarded and all immediatelly followed glues, penalties and kerns are discarded too. They are called *discardable items*.

`\let` ⟨*control sequence*⟩ `=` ⟨*token*⟩ means that these parameters processed by the `\let` command are not expanded.

Whenever a syntactic rule mentions the `=` character (see the previous example with the `\let` command), then this is the equal sign tokenized as a normal character and it is optional. The syntactic rule allows to omit it. Optional spaces are allowed around this equal sign.

The concept of the optional parameters of primitive commands (terminated if something different from the keyword follows) may bring trouble if a macro programmer forgets to terminate an incomplete parameter text by the `\relax` command (`\relax` does nothing but it can terminate a list of optional parameters of the previous command). Suppose, for example, that `\mycoolspace` is defined by `\def\mycoolspace{\penalty42\hskip2mm}`. If a user writes `first\mycoolspace plus second` then TEX reports the error `missing number, treated as zero` in the position of `s` character and appends: `<to be read again> s`. A user who is unfamiliar with TEX primitive commands and their parameters is totally lost. The correct definition is: `\def\mycoolspace{\penalty42\hskip2mm\relax}`.

9 Principles of macros

Macros can be declared by the `\def` primitive command (or `\edef`, `\gdef`, `\xdef` commands; see below). The syntax is `\def` ⟨*control sequence*⟩ ⟨*parameters*⟩ `{` ⟨*replacement text*⟩ `}`. Here, the ⟨*parameters*⟩ are a sequence of formal parameters of the declared macro written in the form `#1`, `#2`, etc. They must be numbered from one and incremented by one. The maximum number of declared parameters is nine. These parameters can be used in the ⟨*replacement text*⟩. This specifies the place where the real parameter is positioned when the macro is expanded. For example:

```
\def\test #1{here is "#1".}
\test A        % expands to: here is "A".
\def\swap #1#2{#2#1}
\swap AB       % expands to: BA
\test {param}  % expands to: here is "param".
\swap A{param} % expands to: paramA
```

Note that there are two possibilities of how to write real macro parameters when a macro is in use. The parameter is one token by default but if there is `{` ⟨*something*⟩ `}` then the parameter is ⟨*something*⟩. The braces here are delimiters for the real parameter (no TEX group is opened/closed here). The example above shows a declaration of *unseparated parameters*. The parameters were declared by `#1` or `#1#2` with no text appended to such a declaration. But there is another

possibility. Each formal parameter can have a text appended in its declaration, so the general syntax of the declaration of formal parameters is `#1`⟨*text1*⟩`#2`⟨*text2*⟩ etc. If such ⟨*text*⟩ is appended then we say that the parameter is *separated* or *delimited* by text. The same delimiter must be used when the macro is in use. For example

```
\def\Test #1#2.#3 {first "#1", second "#2", third "#3"}
\Test ABC.DEF G % expands to: first "A", second "BC",
                % third "DEF". The letter G follows
                % after expansion.
```

In the example above the `#1` parameter is unseparated (one token is read as a real parameter if the syntax `{`⟨*parameter*⟩`}` is not used). The `#2` parameter is delimited by two dots and the `#3` parameter is delimited by space. There may be a ⟨*text0*⟩ immediately before `#1` in the parameter declaration. This means that the declared macro must be used with the same ⟨*text0*⟩ immediately appended. If not, TEX reports the error. The general rule for declaring a macro with three parameters is: `\def`⟨*control sequence*⟩⟨*text0*⟩`#1`⟨*text1*⟩`#2`⟨*text2*⟩`#3`⟨*text3*⟩`{`⟨*replacement text*⟩`}`. The rule "everything must be balanced" is applied to separated parameters too. It means that `\Test AB{C.DEF G}. H` from the example above reads `B{C.DEF G}` to the `#2` parameter and the `#3` parameter is empty because the space (the delimiter of `#3` parameter) immediately follows two dots. The separated parameter can bring a potential problem if the user forgets the delimiter or the delimiter is specified incorrectly. Then TEX reports an error. This error is reported when the first `\par` is scanned as part of the parameter (probably generated from an empty line). If you really want to scan as part of the parameter more paragraphs including `\par` between them, then you can use the `\long` prefix before `\def`. For example `\long\def\scan#1\stop{...}` reads the parameter of the `\scan` macro up to the `\stop` control sequence, and this parameter can include more paragraphs. If the delimiter is missing when a `\long` defined macro is processed, then TEX reports an error at the end of the file. When a real parameter of a macro is scanned then the expand processor is deactivated. When the ⟨*replacement text*⟩ is processed then the expand processor works normally. This means that if parameters are used in the ⟨*replacement text*⟩, then they are expanded here. If a macro declaration is used inside ⟨*replacement text*⟩ of another macro then the number of `#` must be doubled for inner declaration. Example:

```
\def\defmacro#1#2{%
   \def#1##1 ##2 {##1 says: #1 ##2.}%
}
\defmacro\hello{Hello}  % \def\hello#1 #2 {#1 says: Hello #2.}
\defmacro\goodbye{Good bye}
```

```
\hello   Jane Eric      % expands to: Jane says: Hello Eric.
\goodbye Eric John      % expands to: Eric says: Good bye John.
```

The exact implementation of the feature above: when TEX reads macro body (during \def, \edef, \gdef, \xdef) then each double #6 is converted to single #6 and each (unconverted yet) single #6 followed by a digit is converted to an internal mark of future parameter. This mark is replaced by real prameter when the defined macro is used. This rule of conversion of macro body has one exception: \edef{...\the\toks...} keeps the toks content unexpanded and without conversion of hashes. And there exists a reverse conversion from internal marks to #12⟨*number*⟩ and from #6 to #12#12 when TEX writes macro body by \meaning primitive. Note the % characters used in the \defmacro definition in the exmample above. They mask the end of lines. If you don't use them, then the space tokens are included here (generated by the tokenizer at the end of each line). The ⟨*replacement text*⟩ of \defmacro will be ⟨*space*⟩\def#1...{...}⟨*space*⟩ in such a case. Each usage of \defmacro generates two unwanted spaces. It is not a problem if \defmacro is used in the vertical mode because spaces are ignored in this mode. But if \defmacro is used in horizontal mode then these spaces are printed.[12] The macro declaration behaves as another assignment, so the information about such a declaration is lost if it is used in a group and the group is left. But you can use a \global prefix before \def or the primitive \gdef. Then the assignment is global regardless of groups. When \def or \gdef is processed then ⟨*replacement text*⟩ is read with the deactivated expand processor. We have alternatives \edef (expanded def) and \xdef (global expanded def) which read their ⟨*replacement text*⟩ expanded by the expand processor. The summary of \def syntax is:

\def ⟨*control sequence*⟩ ⟨*parameters*⟩ { ⟨*replacement text*⟩ }
\gdef ⟨*control sequence*⟩ ⟨*parameters*⟩ { ⟨*replacement text*⟩ }
\edef ⟨*control sequence*⟩ ⟨*parameters*⟩ { ⟨*replacement text*⟩ }
\xdef ⟨*control sequence*⟩ ⟨*parameters*⟩ { ⟨*replacement text*⟩ }

If you set \tracingmacros=2, you can see in the log file how the macros are expanded.

10 Math modes

The $3⟨*math text*⟩$3 specifies a math formula inside a line of the paragraph. It processes the ⟨*math text*⟩ in a group and in *internal math mode*. The $3$3⟨*math text*⟩$3$3 generates a separate line with math formula(s). It processes the ⟨*math text*⟩ in a group and in *display math mode*. The fonts in

[12] More precisely, they are transformed into horizontal glues used between words.

math mode are selected in a very specific manner which is independent of the current text font. Six different math objects are automatically detected in math mode: `\mathord` (normal material), `\mathop` (big operators), `\mathbin` (binary operators), `\mathrel` (relations), `\mathopen` (open brackets), `\mathclose` (close brackets), `\mathpunct` (punctuation). They can be processed in four styles `\displaystyle` (default in the display mode), `\textstyle` (default in the internal math mode), `\scriptstyle` (used for indexes or exponents, smaller text) and `\scriptscriptstyle` (used in indexes of indexes, even smaller text). The math typesetting algorithms were implemented in TEX by its author with great care. All typographical traditions of math typesetting were taken into account. There are three chapters about math typesetting in his TEXbook. Moreover, there is the detailed appendix G containing the exact specification of generating math formulae. This topic is unfortunately out of the scope of this short text. There is a good a piece of news: all formats (including LATEX) take the default TEX syntax for ⟨*math text*⟩. So, LATEX manuals or LATEX documents serve a good source if you want to get to know the rules of math typesetting by TEX. There is only one significant difference. Fractions are constructed at the primitive level by the `\over` primitive: `{` ⟨*numerator*⟩ `\over` ⟨*denominator*⟩ `}` but LATEX uses a macro `\frac` in the syntax `\frac{` ⟨*numerator*⟩ `}{` ⟨*denominator*⟩ `}`. Plain TEX users (including the author of TEX) prefer the syntax which follows the principle "how a human reads the formula". On the other hand, the `\frac` syntax is derived from machine languages. You can define the `\frac` macro by `\def\frac#1#2{{#1\over#2}}` if you want.

11 Registers

There are four types of registers used in TEX:

- *Counters*; their values are integer numbers. Counters are declared by `\newcount` ⟨*register*⟩ [13] or they are primitive registers (`\linepenalty` for example). TEX interprets primitive commands which represent an integer from an internal table as counter type register too (examples: `\catcode‘A`, `\lccode‘A`).
- *Dimen type*; their values are dimensions. They are declared by `\newdimen` ⟨*register*⟩ or they are primitive registers (`\hsize`, for example). TEX interprets primitive commands which represent a dimension value as dimen type register too (example: `\wd0`).

[13] The declarators `\newcount`, `\newdimen`, `\newskip` and `\newtoks` are plain TEX macros used in all known TEX formats. They provide ⟨*address*⟩ allocation and use the `\count` ⟨*address*⟩, `\dimen` ⟨*address*⟩, `\skip` ⟨*address*⟩ and `\toks` ⟨*address*⟩ TEX registers. The `\countdef`, `\dimendef`, `\skipdef` and `\toksdef` primitive commands are used internally.

- *Glue type*; their values are triples like in general `\hskip` parameters. They can be declared by `\newskip`⟨*register*⟩ or they are primitive registers (`\abovedisplayskip` for example).[14]
- *Token lists*; their values are sequences of tokens. They are declared by `\newtoks`⟨*register*⟩ or they are primitive registers (`\everypar` for example).

The following example shows how registers are declared, how a value is saved to the register, and how to print the value of the register.

```
\newcount  \mynumber
\newdimen  \mydimen
\newskip   \myskip
\newtoks   \mytoks
\mynumber = 42
\mydimen = -13cm
\myskip = 10mm plus 12mm minus1fil
\mytoks = {abCd ef}
To print these values use the primitive command "the":
\the\mynumber, \the\mydimen, \the\myskip, \the\mytoks.
\bye
```

This example prints: To print these values use the primitive command ”the”: 42, -369.88582pt, 28.45274pt plus 34.1433pt minus 1.0fil, abCd ef. Note that the human readable dimensions are converted to typographical points (pt). The general syntactic rule for storing values to registers is ⟨*register*⟩`=`⟨*value*⟩ where the equal sign is optional and it can be surrounded by optional spaces. Syntactic rules for each type of ⟨*value*⟩ depending on type of the register (i.e. ⟨*number*⟩, ⟨*dimen*⟩, ⟨*skip*⟩ and ⟨*toks*⟩) follows.

- The ⟨*number*⟩ could be
  1) a register of counter type;
  2) a character constant declared by `\chardef` or `\mathchardef` primitive command.
  3) an integer decimal number (with optional `+` or `-` prefixed)
  4) `"`⟨*hexa number*⟩ where ⟨*hexa number*⟩ can include all digits and letters `ABCDEF`;
  5) `'`⟨*octal number*⟩ where ⟨*octal num.*⟩ can include digits `01234567`;
  6) `` ` ``⟨*character*⟩ (the prefix is the reverse single quote `` ` ``). It returns the code of the ⟨*character*⟩. Examples: `` `A `` or one-character control sequence `` `\A ``). Both examples represent the number 65. The Unicode of the character is taken here if LuaTEX or XETEX is used;

[14] Very similar muglue type for math glues exists too but it is not described in this text.

7) `\numexpr` ⟨*num. expression*⟩ .[15] The ⟨*num. expression*⟩ uses operators `+`, `-`, `*` and `/` and brackets `(`, `)` in normal sense. The operands are ⟨*number*⟩s. It is terminated by something incompatible with the syntactic rule of ⟨*num. expression*⟩ or by `\relax`. The `\relax` (if it is used as a separator) is removed. If the result is non-integer, then it is rounded (not truncated).

The rules 3)–6) can be terminated by one optional space.

- The ⟨*dimen*⟩ could be
  1) a register of dimen type or counter type;
  2) a decimal number with an optional decimal point (and optional `+` or `-` prefixed) followed by ⟨*dimen unit*⟩. The ⟨*dimen unit*⟩ is `pt` (point)[16] or `mm` or `cm` or `in` or `bp` (big point) or `dd` (Didot point) or `pc` (pica) or `cc` (cicero) or `sp` (scaled point) or `em` (quad of current font) or `ex` (ex height of current font) or a register of dimen type;
  3) `\dimexpr` ⟨*dimen expression*⟩. The ⟨*dimen expression*⟩ uses operators `+`, `-`, `*` and `/` and brackets `(`, `)` in their normal sense. The operands of `+` and `-` are ⟨*dimen*⟩s, the operators of `*` or `/` are the pair ⟨*dimen*⟩ and ⟨*number*⟩ (in this order). The ⟨*dimen expression*⟩ is terminated by something incompatible with the syntactic rule of ⟨*dimen expression*⟩ or by `\relax`. The `\relax` (if it is used as a separator) is removed.

  The rule 2) can be terminated by one optional space.
- The ⟨*skip*⟩ could be:
  - a register of glue type or dimen type or counter type;
  - ⟨*dimen*⟩ `plus` ⟨*generalized dimen*⟩ `minus` ⟨*generalized dimen*⟩. Here, the ⟨*generalized dimen*⟩ is the same as ⟨*dimen*⟩, but normal ⟨*dimen unit*⟩ or pseudo-unit `fil` or `fill` or `filll` can be used.
- The ⟨*toks*⟩ could be
  - ⟨*expandafters*⟩ `{` ⟨*text*⟩ `}`. The ⟨*expandafters*⟩ is typically a sequence of `\expandafter` primitive commands (zero or more). The ⟨*text*⟩ is scanned without expansion but the exception can be given by ⟨*expandafters*⟩.

The main processor reads input tokens (from the output of activated or deactivated expand processor) in two contexts: *do something* or *read parameters*. By default it is in the context *do something*. When a primitive which allows parameters is read, the main processor reads the parameters in the context *read parameters*. Whenever the main processor reads a register in the context *do something* it assumes that an assignment of a value to the register is declared here. The following text (equal sign and ⟨*value*⟩) is read in the context *read*

[15] This is a feature of the ε-TeX extension. It is implemented in pdfTeX, XeTeX and LuaTeX.

[16] 1 pt = 1/72.27 in ≐ 0.35 mm; 1 pc = 12 pt; 1 bp = 1/72 in; 1 dd ≐ 1.07 pt; 1 cc = 12 dd; 1 sp = 2^{-16} pt = TeX accuracy.

parameters. If the following text isn't compliant to the appropriate syntactic rule, TEX reports an error. Examples of register manipulations:

```
\newcount\mynumber \newdimen\mydimen \newdimen\myskip
\hsize = .7\hsize  % see the rule for <dimen>, unit
                   % could be a register
\hoffset = \dimexpr 10mm - (\parindent + 1in) \relax
           % usage of \dimexpr
\myskip = 10pt plus15pt minus 3pt
\mydimen = \myskip  % the information
                    % "plus15pt minus 3pt" is lost
\mynumber = \mydimen  % \mynumber = 10*2^16 because
                      % \mydimen = 10*2^16 sp
```

Each dimension is saved internally as an integer multiple of the `sp` unit in TEX. When we need a conversion $\langle dimen\rangle \to \langle number\rangle$, then simply the internal unit `sp` is omitted. The summary of most commonly used primitive registers including their default value given by plain TEX follows.

- `\hsize=6.5in`, `\vsize=8.9in` are paragraph width and page height.
- `\hoffset=0pt`, `\voffset=0pt` give left margin and top margin of the page. They are calculated from the *page origin* which is defined by coordinates `\pdfvorigin=1in` and `\pdfhorigin=1in` measured from left upper corner of the page.
- `\parindent=20pt` is the indentation of the first line of each paragraph.
- `\parfillskip=0pt plus 1fil` is horizontal glue added to the last line of the paragraph.
- `\leftskip=0pt`, `\rightskip=0pt`. Glues added to each line in the paragraph from the left and the right side. If the stretchability is declared here, then the paragraph is ragged left/right.
- `\parskip=0pt plus 1pt` is the vertical space between paragraphs.
- `\baselineskip=12pt`, `\lineskiplimit=0pt`, `\lineskip=1pt`.
  The `\baselineskip` *rule* says: Two consecutive lines in the vertical list have the baseline distance given by `\baselineskip` by default. The appropriate real glue is inserted between the lines. But if this real glue (between boxes) is less than `\lineskiplimit` then `\lineskip` is inserted between the boxes instead.
- `\topskip=10pt` is the distance between the top of the page box and the baseline of the first line.
- `\linepenalty=10`, `\hyphenpenalty=50`, `\exhyphenpenalty=50`, `\binoppenalty=700`, `\relpenalty=500`, `\clubpenalty=150`, `\widowpenalty=150`, `\displaywidowpenalty=50`, `\brokenpenalty=100`,

`\predisplaypenalty=10000`, `\postdisplaypenalty=0`, `\interlinepenalty=0`, `\floatingpenalty=0`, `\outputpenalty=0`.
These penalties apply to various places in the vertical or horizontal list. Most important are `\clubpenalty` (inserted below the first line of a paragraph) and `\widowpenalty` (inserted before the last line of a paragraph). Typographical rules often demand us to set these registers to 10000 (no page break is allowed here).

- `\looseness=0` allows us to create of a "suboptimal" paragraph. The paragraph-building algorithm tries to build the paragraph with `\looseness` lines more than the optimal solution. If the `\tolerance` does not have a sufficiently large value then this setting is simply ignored. It is reset to zero after each paragraph is completed.
- `\spaceskip=0pt`, `\xspaceskip=0pt`. If non-negative they are used as glues between words. Default values are read from the font metric data of the current font.
- `\pretolerance=100`, `\tolerance=200`, `\emergencystretch=0pt` `\doublehyphendemerits=10000`, `\finalhyphendemerits=5000`, `\adjdemerits=10000`, `\hfuzz=0.1pt`, `\vfuzz=0.1pt` are parameters for the paragraph building algorithm (not described here in detail).
- `\hbadness=1000`, `\vbadness=1000`. TEX reports a warning about badness on the terminal and to the log file if it is greater than these values. The warning has the form `underfull \hbox` or `underfull \vbox`. The value `100` means that the `plus` limit for glues is reached.
- `\tracingonline=0`, `\tracingmacros=0`, `\tracingstats=0`, `\tracingparagraphs=0`, `\tracingpages=0`, `\tracingoutput=0`, `\tracinglostchars=1`, `\tracingcommands=0`, `\tracingrestores=0`, `\tracingscantokens=0`, `\tracingifs=0`, `\tracinggroups=0`, `\tracingassigns=0`.
  If these registers have positive values then TEX reports details about the processing of built-in algorithms to the log file. If `\tracingonline>0` then the same output is shown on the terminal.
- `\showboxbreadth=5`, `\showboxdepth=3`, `\errorcontextlines=5`. The amount of information shown when boxes are traced to the log file or an error is reported.
- `\language=0`. TEX is able to load more hyphenation patterns for more languages. This register points to the index of currently used hyphenation patterns. Zero means English.
- `\lefthyphenmin=2`, `\righthyphenmin=3`. Maximum letters left or right in hyphenated words.
- `\defaulthyphenchar=‘\-`. This character is used when words are hyphenated.
- `\globaldefs=0`. If it is positive then all settings are global.

- `\hangafter=1`, `\hangindent=0pt`. If `\hangindent` is positive, then after `\hangafter` lines all following lines are indented. Negative/positive values of `\hangindent` or `\hangafter` applies indentation from left or right and from the top or bottom of the paragraph. The `\hangindent` is set to 0 after each paragraph.
- `\mag=1000`. Magnification factor of all used dimensions. The value 1000 means 1:1.
- `\escapechar='\\` use this character in the `\string` primitive.
- `\newlinechar=-1`. If positive, this character is interpreted as the end of the line when printing to the log or by the `\write` primitive command.
- `\endlinechar='^^M`. This character is appended to the end of each input line. The tokenizer converts it (the Ctrl-M character) to the space token.
- `\time=now`, `\day=now`, `\month=now`, `\year=now`. The values about current time/date are set here when TEX starts to process the document. The `\time` counts minutes after midnight.
- `\prevdepth=*` includes the depth of the last box in vertical mode.
- `\prevgraph=*` includes the number of lines of the paragraph when `\par` finishes.
- `\overfullrule=5pt`. A rectangle to this width is appended after each overfull `\hbox`.
- `\mathsurround=0pt` is the space inserted around a formula in internal math mode.
- `\abovedisplayskip=12pt plus3pt minus9pt`,
  `\abovedisplayshortskip=0pt plus3pt`,
  `\belowdisplayskip=12pt plus3pt minus9pt`,
  `\belowdisplayshortskip=7pt plus3pt minus 4pt`.
  These spaces are inserted above and below a formula generated in math display mode.
- `\tabskip=0pt` is used by the `\halign` primitive command for creating tables.
- `\output={\plainoutput}`, `\everypar={}`, `\everymath={}`
  `\everydisplay={}`, `\everyhbox={}` `\everyvbox={}` `\everycr={}`,
  *`\everyeof={}`, `\everyjob={}`.
  These token lists are processed when an algorithm of TEX reaches a corresponding situations respectively: opens output routine, paragraph, internal math mode, display math mode, `\vbox`, `\hbox`, is at the end of a line in a table, at the end of an input file, or starts the job.

12 Expandable primitive commands

These commands are processed like macros, i.e. they expand to another sequence of tokens. Notes about notation are in this and the following sections. If the

documented command is from the εTEX extension (i.e. implemented in pdfTEX, XETEX and LuaTEX) then one * is prefixed. If it is from the pdfTEX extension (implemented in XETEX and LuaTEX too) then two ** are prefixed. If it is a LuaTEX only command then three *** are prefixed.

- `\string` ⟨*control sequence*⟩ expands to "the `\escapechar`" followed by the name of the control sequence. "The `\escapechar`" means a character with code equal to `\escapechar` or nothing if its value is out of range of character codes. All characters of the output are "other characters$_{12}$", only spaces (if any exist) are kept as space tokens $\sqcup_{10}$.
- ***`\csstring` ⟨*control sequence*⟩ works similarly to `\string` but without `\escapechar`.
- *`\detokenize` ⟨*expandafters*⟩ `{` ⟨*text*⟩ `}` re-tokenizes all tokens in the text. Control sequences used in ⟨*text*⟩ are re-tokenized like the `\string` primitive, spaces are tokens $\sqcup_{10}$, and all other tokens are set as "other characters$_{12}$".
- `\the` ⟨*register*⟩ expands to the value of the register. Examples appear in the previous section. The output is tokenized like of `\detokenize`. The exception is `\the` ⟨*tokens register*⟩: the output is the value of the ⟨*tokens register*⟩ without re-tokenizing and the expand processor does not expand this output in `\edef`, `\write`, `\message`, etc., arguments.
- `\scantokens` ⟨*expandafters*⟩ `{` ⟨*text*⟩ `}` re-tokenizes ⟨*text*⟩ using the actual tokenizer setting. The behavior is the same as when writing ⟨*text*⟩ to a virtual file and reading this file immediately.
- ***`\scantextokens` ⟨*expandafters*⟩ `{` ⟨*text*⟩ `}` resembles `\scantokens` but removes problems with end-of-virtual-file.
- `\meaning` ⟨*token*⟩ expands to the meaning of the ⟨*token*⟩. The text is tokenized like the `\detokenize` output.
- `\csname` ⟨*text*⟩ `\endcsname` creates a control sequence with name ⟨*text*⟩. If it is not already defined, then it gets the `\relax` meaning. For example `\csname TeX\endcsname` is the same as `\TeX`. The ⟨*text*⟩ must be expandable to characters only. Non-expandable control sequences (a primitive command at the main processor level, a register, a character constant, a font selector) are disallowed here. TEX reports the error `missing \endcsname` when this rule isn't compliant. Example: `\csname foo:\the\mynumber\endcsname` expands to control sequence `\foo:42` if the `\mynumber` is a register with the value 42. Another example: a macro programmer should implement a key/value dictionary using this primitive:

```
\def\keyval #1 #2 {\expandafter\def\csname
                   dict:#1\endcsname{#2}}
\def\value #1 {\csname dict:#1\endcsname}
\keyval Peter 21 % key=Peter, value=21, saved to
```

```
                    % the dictionary, it does
                    % \def\dict:Peter{21}
\value Peter % expands to \dict:Peter and then 21
```

- `\expandafter` ⟨*token1*⟩ ⟨*token 2*⟩ does the following transformation: ⟨*token 1*⟩ ⟨*expanded token 2*⟩. The token processor will expand ⟨*token 1*⟩ after such a transformation. The ⟨*expanded token 2*⟩ is only the first level of expansion. For example, a macro is transformed to its ⟨*replacement text*⟩ but without expansion of ⟨*replacement text*⟩ at this time. Or the `\csname...\endcsname` pair creates a control sequence but does not expand it at this time. If ⟨*token 2*⟩ is not expandable then `\expandafter` silently does nothing. The example above (the `\keyval` macro) shows the usage of `\expandafter`. We need not define `\csname` by `\def`; we want to define a `\dict:key`. The `\expandafter` helps here. The ⟨*token 2*⟩ can be another `\expandafter`. We can see `\expandafter` chains in many macro files. For example, `\expandafter\A\expandafter\B\expandafter\C\D` is processed as follows: `\A \B \C` ⟨*expanded*⟩ `\D`. The ⟨*expandafters*⟩ `{` ⟨*text*⟩ `}` syntax rule enables us to prepare ⟨*text*⟩ by `\expandafter`(s). For example `\detokenize{\macro}` expands to $\backslash_{12}$m$_{12}$a$_{12}$c$_{12}$r$_{12}$o$_{12}$. But if you need to detokenize the ⟨*repl. text*⟩ of the `\macro` then use `\detokenize\expandafter{\macro}`. Not only `\expandafter`s should be here. The expand processor does full expansion here until an opening brace {$_1$ is found.
- The general rule for all `\if*` commands is ⟨*if condition*⟩ ⟨*true text*⟩ `\else` ⟨*false text*⟩ `\fi`. The ⟨*if condition*⟩ evaluates and ⟨*true text*⟩ or ⟨*false text*⟩ is skipped or processed depending on the result of ⟨*if condition*⟩. When the expand processor is skipping the text due to an `\if*` command, it expands nothing in the skipped text. But it is noticing all control sequences with meaning `\if*`, `\else` and `\fi` during skipping in order to skip correctly all nested `\if*...\else...\fi` constructions. The following ⟨*if condition*⟩s are possible:
  - `\if` ⟨*token 1*⟩ ⟨*token 2*⟩ is true if
    a) both tokens are characters with the same Unicode (or ASCII code in classical TEX) or
    b) both tokens are control sequences (with arbitrary meaning but not "the character") or
    c) one token is a character, second is a control sequence equal to the character (by `\let`) or
    d) both tokens are control sequences, their meaning (set by `\let`) is the same character code.

    Example: you can say `\let\test=a` then `\if\test a` returns true.
  - `\ifx` ⟨*token 1*⟩ ⟨*token 2*⟩ is true if the meanings of ⟨*token 1*⟩ and ⟨*token 2*⟩ are the same.

- ◦ `\ifnum` ⟨*number 1*⟩ ⟨*relation*⟩ ⟨*number 2*⟩ . The ⟨*relation*⟩ could be `<` or `=` or `>`. It returns true if the comparison of the two numbers is true.
- ◦ `\ifodd` ⟨*number*⟩ returns true if the ⟨*number*⟩ is odd.
- ◦ `\ifdim` ⟨*dimen*⟩ ⟨*relation*⟩ ⟨*dimen*⟩ The ⟨*relation*⟩ could be `<` or `=` or `>`. It returns true if the comparison of the two dimensions is true.
- ◦ `\iftrue` returns constantly true, `\iffalse` returns constantly false.
- ◦ `\ifhmode`, `\ifvmode`, `\ifmmode` – true if the current mode is horizontal, vertical, math.
- ◦ `\ifinner` returns true if the current mode is internal vertical, internal horizontal or internal math mode.
- ◦ `\ifhbox` ⟨*box number*⟩ , `\ifvbox` ⟨*box number*⟩ , `\ifvoid` ⟨*box num.*⟩ returns true if the specified ⟨*box num.*⟩ represents `\hbox`, `\vbox`, void box respectively.
- ◦ `\ifcat` ⟨*token 1*⟩ ⟨*token 2*⟩ is true if the category codes of ⟨*token 1*⟩ and ⟨*token 2*⟩ are equal.
- ◦ `\ifeof` ⟨*file number*⟩ is true if the file attached to the ⟨*file number*⟩ by the `\openin` primitive does not exist, or the end of file was reached by the `\read` primitive.
- • *`\unless` ⟨*if condition*⟩ negates the result of ⟨*if condition*⟩ before skipping or processing the following text.
- • `\ifcase` ⟨*num.*⟩⟨*case 0*⟩ `\or` ⟨*case 1*⟩...`\or` ⟨*case n*⟩ `\else` ⟨*else text*⟩ `\fi`. This processes the branch given by ⟨*number*⟩ . It processes ⟨*else text*⟩ (or nothing if no ⟨*else text*⟩ is declared) when a branch with a given ⟨*number*⟩ does not exist.
- • *`\pdfstrcmp{` ⟨*string A*⟩ `}{` ⟨*string B*⟩ `}` is -1 if ⟨*string A*⟩ $<$ ⟨*string B*⟩ , 0 if they are equal or 1 otherwise. It is not implemented in LuaTEX.
- • `\noexpand` ⟨*token*⟩ . The expand processor does not expand the ⟨*token*⟩ if it is expanding the text in `\edef`, `\write`, `\message` or similar lists.
- • *`\unexpanded` ⟨*expandafters*⟩ `{` ⟨*text*⟩ `}` returns ⟨*text*⟩ and applies `\noexpand` to all tokens in the ⟨*text*⟩ .
- • **`\expanded{` ⟨*tokens*⟩ `}` expands ⟨*tokens*⟩ and reads these expanded ⟨*tokens*⟩ again.
- • *`\numexpr` ⟨*num. expression*⟩ , *`\dimexpr` ⟨*dimen expression*⟩ . Documented in the ⟨*dimen*⟩ and ⟨*number*⟩ syntax rules in section 11.
- • `\number` ⟨*number*⟩ , `\romannumeral` ⟨*number*⟩ prints ⟨*number*⟩ in decimal digits or as a roman numeral (with lowercase letters).
- • `\topmark` (last from previous page), `\firstmark` (first on current page), `\botmark` (last on current page). They expand to the corresponding `\mark` included in the current or previous page-box. Usable for implementing running headers in the output routine.

- `\fontname` ⟨*font selector*⟩ expands to the file name ***(or font name) of the font given by its ⟨*font selector*⟩. The `\fontname\font` expands to the file name of the current font.
- `\jobname` expands to the name of the main file of this document (without extension `.tex`).
- `\input` ⟨*file name*⟩ ⟨*space*⟩ (classical TEX), `\input"` ⟨*file name*⟩`"` or `\input{` ⟨*file name*⟩`}` opens the given ⟨*file name*⟩ and starts to read input from it. If the ⟨*file name*⟩ doesn't exist then TEX tries again to open ⟨*file name*⟩`.tex`. If that doesn't exist, TEX reports an error. The alternative syntax with `"..."` or `{...}` allows having spaces in the file names.
- `\endinput`. The current line is the last line of the file being input. The file is closed and reading continues from the place where `\input` of this file was started. `\endinput` done in the main file causes future reading from the terminal and a headache for the user.
- ***`\directlua {` ⟨*text*⟩`}` runs a Lua script given in ⟨*text*⟩.

13 Primitive commands at main processor level

Commands used for declaration of control sequences

- `\def`, `\edef`, `\gdef`, and `\xdef` were documented in section 9.
- `\long` is a prefix; it can be used before `\def`, `\edef`, `\gdef`, `\xdef`. The declared macro accepts the control sequence `\par` in its parameters.
- *`\protected` is a prefix; it can be used before `\def`, `\edef`, `\gdef`, `\xdef`. The declared macro is not expanded by the expand processor in `\write`, `\message`, `\edef`, etc., parameters.
- `\outer` is a prefix; it can be used before `\def`, `\edef`, `\gdef`, `\xdef`. The declared macro must be used only when the main processor is in the context *do something* or TEX reports an error.
- `\global` is a prefix; it can be used before any assignment (commands from this subsection and ⟨*register*⟩`=`⟨*value*⟩ settings). The assignment is global regardless of the current group.
- `\chardef` ⟨*cont. seq.*⟩`=`⟨*num.*⟩, `\mathchardef` ⟨*cont. seq.*⟩`=`⟨*num.*⟩ declares a constant ⟨*number*⟩. When the main processor is in the context *do something* and it gets a `\chardef`-ed control sequence, it prints the character with Unicode (ASCII code) ⟨*number*⟩ to the typesetting output. If it gets a `\mathchardef`-ed control sequence, it prints a math object (it works only in math mode, not documented here).
- `\countdef` ⟨*control seq.*⟩`=`⟨*number*⟩ declares ⟨*control sequence*⟩ as an equivalent to the `\count` ⟨*number*⟩ which is a register of counter type. The ⟨*number*⟩ here means an address in the array of registers of counter type.

The \count0 is reserved for the page number. Macro programmers rarely use direct addresses (1 to 9), more common is using the allocation macro \newcount ⟨control sequence⟩.

- \dimendef, \skipdef, \muskipdef, \toksdef when they are followed by ⟨control sequence⟩=⟨num.⟩ declare analogical equivalents to \dimen ⟨num.⟩, \skip ⟨number⟩, \muskip ⟨number⟩ and \toks ⟨number⟩. Usage of allocation macros \newdimen, \newskip, \newmuskip, \newtoks are preferred.
- \font ⟨font selector⟩=⟨file name⟩ ⟨space⟩ ⟨size specification⟩ declares the ⟨font sel.⟩ of a font implemented in the ⟨file name⟩ .tfm. The ⟨size spec.⟩ can be at ⟨dimen⟩ or scaled ⟨factor⟩. The ⟨factor⟩ equal to 1000 means 1:1. A new syntax (supported by Unicode engines) is

  \font ⟨font sel.⟩="⟨font name⟩:⟨font features⟩" ⟨size specification⟩
  \font ⟨font sel.⟩="[⟨font file⟩]:⟨font features⟩" ⟨size specification⟩

  The ⟨font file⟩ is a file name without an .otf or .ttf extension. The ⟨font features⟩ are font features prefixed by + or - and separated by a semicolon. The otfinfo -f ⟨file name⟩.otf command (on command line) can list them. LuaTEX supports alternative syntax: {...} instead of "...". For example \font\test={[texgyretermes-regular]:+onum;-liga} at12pt.
- \let ⟨control sequence⟩=⟨token⟩ sets to the ⟨control sequence⟩ the same meaning as ⟨token⟩ has. The ⟨token⟩ can be whatever, a character or a control sequence.
- \futurelet ⟨control sequence⟩ ⟨token 1⟩ ⟨token 2⟩ works in two steps. In the first step it does \let ⟨control sequence⟩=⟨token 2⟩ and in the second step ⟨token 1⟩ ⟨token 2⟩ is processed with activated token processor. Typically ⟨token 1⟩ is a macro that needs to know the next token.

Commands for box manipulation

- \hbox{⟨cmds⟩} or \hbox to ⟨dimen⟩{⟨cmds⟩} or \hbox spread ⟨dimen⟩{⟨cmds⟩} creates a box. The material inside this box is a ⟨horizontal list⟩ generated by ⟨cmds⟩ in horizontal mode in a group. The width of the box is the natural width of the ⟨horizontal list⟩ or ⟨dimen⟩ given by the to ⟨dimen⟩ parameter or it is spread by the ⟨dimen⟩ given by the spread ⟨dimen⟩ parameter. The height of the box is the maximum of heights of all elements in the ⟨horizontal list⟩. The depth of the box is the maximum of depths of all such elements. These elements are set on the common baseline (exceptions can be given by \lower or \raise commands).
- \vbox{⟨cmds⟩} or \vbox to ⟨dimen⟩{⟨cmds⟩} or \vbox spread ⟨dimen⟩{⟨cmds⟩} creates a box. The material inside this box is a ⟨vertical list⟩ generated by ⟨cmds⟩ in vertical mode in a group. The height of the box is the natural height of the ⟨vertical list⟩ (eventually modified by values from to

or `spread` parameters) without the depth of the last element. The depth of the last element is set as the depth of the box. The width of the box is the maximum of widths of elemens in the ⟨*vertical list*⟩. All elements are placed at the common left margin of the box (exceptions can be given by `\moveleft` or `\moveright` commands).

- `\vtop{`⟨*cmds*⟩`}` (with optional `to` or `spread` parameters) is the same as `\vbox`, but the baseline of the resulting box goes through the baseline of the first element in the ⟨*vertical list*⟩ (note that `\vbox` has its baseline equal to the baseline of the last element inside).
- `\vcenter{`⟨*cmds*⟩`}` (with optional `to` or `spread` parameters) is equal to `\vbox`, but its *math axis*[17] is exactly in the middle of the box. So its baseline is appropriately shifted. The `\vcenter` can be used only in math modes but given ⟨*cmds*⟩ are processed in vertical mode.
- `\lower`⟨*dimen*⟩⟨*box*⟩, `\raise`⟨*dimen*⟩⟨*box*⟩ move the ⟨*box*⟩ up or down by the ⟨*dimen*⟩ in horizontal mode. `\moveleft`⟨*dimen*⟩⟨*box*⟩, `\moveright`⟨*dimen*⟩⟨*box*⟩ move the ⟨*box*⟩ by the ⟨*dimen*⟩ in vertical mode.
- `\setbox`⟨*box number*⟩`=`⟨*box*⟩. TEX has a set of *box registers* addressed by ⟨*box number*⟩ and accessed via `\box`⟨*box number*⟩ or alternatives described below. The `\setbox` command saves the given ⟨*box*⟩ to the register addressed by ⟨*box number*⟩. Macro programmers use only 0 to 9 ⟨*box numbers*⟩ directly. Other addresses to box registers should be allocated by the `\newbox`⟨*control sequence*⟩ macro. The ⟨*control sequence*⟩ is equivalent to a ⟨*box number*⟩, not to the box register itself. The `\setbox` command does an assignment, so the `\global` prefix is needed if you want to use the saved box outside the current group.
- `\box`⟨*box number*⟩ returns the box from ⟨*box number*⟩ box register. Example: you can do `\setbox0=\hbox{abc}`. This `\hbox` isn't printed but saved to the register 0. At a different place you use `\box0`, which prints `\hbox{abc}`, or you can do `\setbox0= \hbox{cde\box0}` which saves the `\hbox{cde\hbox{abc}}` to the register 0.
- `\copy`⟨*box number*⟩ returns the box from ⟨*box number*⟩ box register and keeps the same box in this box register. Note that the `\box`⟨*box number*⟩ returns the box and empties the register ⟨*box number*⟩ immediately. If you don't want to empty the register, use `\copy`.
- `\wd`⟨*box number*⟩, `\ht`⟨*box number*⟩, `\dp`⟨*box number*⟩. You can measure or use the width, height and depth of a box saved in a register addressed by ⟨*box number*⟩. Examples `\mydimen=\ht0`, `\hbox to\wd0{...}`. You can reset the dimensions of a box saved in a register addressed by ⟨*box number*⟩. For

[17] The math axis is a horizontal line which goes through centers of + and − symbols. Its distance from the baseline is declared in the math font metrics.

example `\setbox0=\hbox{abc} \wd0=0pt \box0` gives the same result as `\hbox to0pt{abc}` but without the warning about overfull `\hbox`.

- `\unhbox` ⟨*box number*⟩, `\unvbox` ⟨*box num.*⟩, `\unhcopy` ⟨*box num.*⟩, `\unvcopy` ⟨*box num.*⟩ do the same work as `\box` or `\copy` but they don't return the whole box but only its contents, i.e. the horizontal or vertical material. Example: try to do `\setbox0=\hbox{abc}` and later `\setbox0=\hbox{cde\unhbox0}` saves the `\hbox{cdeabc}` to the box register 0. The `\unhbox` and `\unhcopy` commands return the `\hbox` contents and `\unvbox`, `\unvcopy` commands return the `\vbox` contents. If incompatible contents are saved, then TEX reports an error. You can test the type of saved contents by `\ifhbox` or `\ifvbox`.
- `\vsplit` ⟨*box number*⟩ `to` ⟨*dimen*⟩ breaks a column. The ⟨*vertical material*⟩ saved in the box ⟨*box number*⟩ is broken into a first part of ⟨*dimen*⟩ height and the rest remains in the box ⟨*box number*⟩. The broken part is saved as a `\vbox` which is the result of this operation. For example, you can say `\newbox\col \setbox\col=\vbox{...}` and later `\setbox0=\vsplit\col to5cm`. The `\box0` is a `\vbox` containing the first 5cm of saved material.
- `\lastbox` returns the last box in the current vertical or horizontal material and removes it.

Commands for rules (lines in the typesetting output) and patterns

- `\hrule` creates a horizontal line in the current vertical list. If it is used in horizontal mode, it finishes the paragraph by `\par` first. `\hrule` `width` ⟨*dimen*⟩ `height` ⟨*dimen*⟩ `depth` ⟨*dimen*⟩ creates (in general, with given parameters) a full rectangle (something like a box, but it isn't treated as the box) with given dimensions. Default values are: "width" = width of outer `\vbox`, "height" = 0.4 pt, "depth" = 0 pt.
- `\vrule` creates a vertical line in the current horizontal list. If it is used in vertical mode, it opens the horizontal mode first. `\vrule` `width` ⟨*dimen*⟩ `height` ⟨*dimen*⟩ `depth` ⟨*dimen*⟩ creates (in general, with given parameters) a full rectangle with given dimensions. Default values are: "width" = 0.4 pt, "height" = height of outer `\hbox`, "depth" = depth of outer `\hbox`.

  The optional parameters of `\hrule` and `\vrule` can be specified in arbitrary order and they can be specified more than once. In such a case, the rule "last wins" is applied.
- `\leaders` ⟨*rule*⟩ ⟨*glue*⟩ creates a glue (maybe shrinkable or stretchable) filled by a full rectangle. The ⟨*rule*⟩ is `\vrule` or `\hrule` (maybe with its optional parameters). If the ⟨*glue*⟩ is specified by an `\hskip` command (maybe with its optional parameters) or by its alternatives `\hss`, `\hfil`, `\hfill`, then the resulting glue is horizontal (can be used only in horizontal mode) and its dimensions are: width derived from ⟨*glue*⟩, height plus depth derived

from ⟨*rule*⟩. If the ⟨*glue*⟩ is specified by a `\vskip` command (maybe with its optional parameters) or by its alternatives `\vss`, `\vfil`, `\vfill`, then the resulting glue is vertical (can be used only in vertical mode) and its dimensions are: height derived from ⟨*glue*⟩, width derived from ⟨*rule*⟩, depth is zero.

- `\leaders` ⟨*box*⟩ ⟨*glue*⟩ creates a vertical or horizontal glue filled by a pattern of repeated ⟨*box*⟩. The positions of boxes are calculated from the boundaries of the outer box. It is used for the dots patterns in the table of contents. `\cleaders` ⟨*box*⟩ ⟨*glue*⟩ does the same, but the pattern of boxes is centered in the space derived by the ⟨*glue*⟩. Spaces between boxes are not inserted. `\xleaders` ⟨*box*⟩ ⟨*glue*⟩ does the same, but the spaces between boxes are inserted equally.

More commands for creating something in typesetting output

- `\par` closes horizontal mode and finalizes a paragraph.
- `\indent`, `\noindent`. They leave vertical mode and open a paragraph with/without paragraph indentation. If horizontal mode is current then `\indent` inserts an empty box of `\parindent` width; `\noindent` does nothing.
- `\hskip`, `\vskip`. They insert a horizontal/vertical glue. Documented in section 7.
- `\hfil`, `\hfill`, `\hss`, `\vfil`, `\vfill`, `\vss` are alternatives of `\hskip`, `\vskip`, see section 7.
- `\hfilneg`, `\vfilneg` are shortcuts for `\hskip 0pt plus-1fil` and `\vskip 0pt plus-1fil`.
- `\kern` ⟨*dimen*⟩ puts unbreakable horizontal/vertical space depending on the current mode.
- `\penalty` ⟨*number*⟩ puts the penalty ⟨*number*⟩ on the current horizontal/vertical list.
- `\char` ⟨*number*⟩ prints the character with code ⟨*number*⟩. The "character itself" does the same.
- `\accent` ⟨*number*⟩ ⟨*character*⟩ places an accent with code ⟨*number*⟩ above the ⟨*character*⟩.
- `\␣` is the control space. In horizontal mode, it inserts the space glue (like normal space but without modification by the `\spacefactor`). In vertical mode, it opens horizontal mode and puts the space. Note that normal space does nothing in vertical mode.
- `\discretionary{` ⟨*pre break*⟩ `}{` ⟨*post break*⟩ `}{` ⟨*no break*⟩ `}` works in horizontal mode. It prints ⟨*no break*⟩ in normal cases but if there is a line break then ⟨*pre break*⟩ is used before and ⟨*post break*⟩ after the breaking point. German Zucker/Zuk-ker (sugar) can be implemented by `Zu\discretionary{k-}{k}{ck}er`.

- `\-` is equal to `\discretionary{\char\hyphenchar` ⟨*font*⟩ `}{}{}`. The `\hyphenchar` ⟨*font*⟩ is used as a hyphenation character. It is set to `\defaulthyphenchar` value when the font is loaded, but it can be changed.
- `\/` does an italic correction. It puts a little space if the last character is slanted.
- `\unpenalty`, `\unskip` removes the last penalty/last glue from the current horizontal/vertical list.
- `\vadjust{` ⟨*cmds*⟩ `}`. This works in horizontal mode. The ⟨*cmds*⟩ must create a ⟨*vertical list*⟩ and `\vadjust` saves a pointer to this list into the current horizontal list. When `\par` creates lines of the paragraph and distributes them to a vertical list, each line with the pointer from `\vadjust` has the corresponding ⟨*vertical list*⟩ immediately appended after this line.
- `\insert` ⟨*number*⟩ `{` ⟨*cmds*⟩ `}`. The ⟨*cmds*⟩ create a ⟨*vertical list*⟩ and `\insert` saves a pointer to such a ⟨*vertical list*⟩ into the current list. The output routine can work with such ⟨*vertical list*⟩s. The footnotes or *floating objects* (tables, figures) are implemented by the `\insert` primitive.
- `\halign{` ⟨*decl.*⟩ `\cr` ⟨*row 1*⟩ `\cr` ⟨*row 2*⟩ `\cr...\cr` ⟨*row n*⟩ `\cr}` creates a table of boxes in vertical mode. The ⟨*declaration*⟩ declares one or more column patterns separated by `&`$_4$. The rows use the same character to separate the items of the table in each row. The `\halign` works in two passes. First it saves all items to boxes and the second pass performs `\hbox to` *w* for each saved item, where *w* is the maximum width of items in each actual column. Detailed documentation of `\halign` is out of scope of this manual. Only one example follows: the macro `\putabove` puts `#1` above `#2` centered. The width of the resulting box is equal to the maximum of widths of these two parameters. The ⟨*declaration*⟩ `\hfil##\hfil` means that the items will be centered: `\def\putabove#1#2{\vbox{\halign{ \hfil##\hfil\cr#1\cr#2\cr}}}`.
- `\valign` does the same as `\halign` but rows ↔ columns. It is not commonly used.
- `\cr`, `\crcr`, `\span`, `\omit`, `\noalign{` ⟨*cmds*⟩ `}` are primitives used by `\halign` and `\valign`.

Commands for register calculations

- `\advance` ⟨*register*⟩ `by` ⟨*value*⟩ does (formally) ⟨*register*⟩ `=` ⟨*register*⟩ `+` ⟨*value*⟩ . The ⟨*register*⟩ is counter type or dimen type. The ⟨*value*⟩ is ⟨*number*⟩ or ⟨*dimen*⟩ (depending on the type of ⟨*register*⟩).
- `\multiply` ⟨*register*⟩ `by` ⟨*number*⟩ does ⟨*register*⟩ `=` ⟨*register*⟩ `*` ⟨*number*⟩ .
- `\divide` ⟨*register*⟩ `by` ⟨*number*⟩ does ⟨*register*⟩ `=` ⟨*register*⟩ `/` ⟨*number*⟩ . If the ⟨*register*⟩ is number type then the result is truncated.
- See *`\numexpr` and *`\dimexpr`, expandable primitives documented in sections 11 and 12.

Internal codes

- \catcode ⟨*number*⟩ is category code of the character with ⟨*number*⟩ code. Used by tokenizer.
- \lccode ⟨*number*⟩ is the lowercase alternative to \char ⟨*number*⟩. If it is zero then a lowercase alternative doesn't exist (for example for punctuation). Used by the \lowercase primitive and when breaking points are calculated from hyphenation patterns.
- \uccode ⟨*number*⟩ is the uppercase alternative to \char ⟨*number*⟩. If it is zero, then the uppercase alternative doesn't exist. Used by the \uppercase primitive.
- \lowercase ⟨*expandafters*⟩ { ⟨*text*⟩ } and \uppercase ⟨*expandafters*⟩ { ⟨*text*⟩ } transform ⟨*text*⟩ to lowercase/uppercase using the current \lccode or \uccode values. Returns transformed ⟨*text*⟩ where catcodes of tokens and tokens of type ⟨*control sequence*⟩ are unchanged.
- \sfcode ⟨*number*⟩ is the spacefactor code of the \char ⟨*number*⟩. The \spacefactor register keeps (roughly speking) the \sfcode of the last printed character. The glue between words is modified (roughly speaking) by this \spacefactor. The value 1000 means factor 1:1 (no modification is done). It is used for enlarging spaces after periods and other punctuation in English texts.[18]

Commands for reading or writing text files

- Note that the main input stream is controlled by \input and \endinput expandable primitive commands documented in section 12.
- \openin ⟨*file number*⟩ = ⟨*file name*⟩ ⟨*space*⟩ (or \openin ⟨*file number*⟩ = { ⟨*file name*⟩ }) opens the file named ⟨*file name*⟩ for reading and creates a file descriptor connected to the ⟨*file number*⟩.[19] If the file doesn't exist nothing happens but a macro programmer can test this case by \ifeof ⟨*file number*⟩.
- \read ⟨*file number*⟩ to ⟨*control seq.*⟩ does \def ⟨*control seq.*⟩ { ⟨*repl. text*⟩ } where the ⟨*replacement text*⟩ is the tokenized next line from the file declared by \openin as ⟨*file number*⟩.
- \openout ⟨*file number*⟩ = ⟨*file name*⟩ ⟨*space*⟩ (or \openout ⟨*file number*⟩ = " ⟨*file name*⟩ ") opens the ⟨*file name*⟩ for writing and creates a file descriptor connected to ⟨*file number*⟩. If the file already exists, then its contents are removed.

[18] This does not comply with other typographical traditions, so the \frenchspacing macro which sets all \sfcodes to 1000 is used very often.

[19] Note that ⟨*file number*⟩ is an address to the file descriptor. Macro programmers don't use these addresses directly but by the \newread ⟨*control sequence*⟩ and \newwrite ⟨*control sequence*⟩ allocation macros.

- `\write` ⟨*file number*⟩ `{` ⟨*text*⟩ `}` writes a line of ⟨*text*⟩ to the file declared by `\openout` as ⟨*file number*⟩. But this isn't done immediately. TEX does not know the value of the current page when the `\write` command is processed because the paragraph building and page building algorithms are processed asynchronously. But a macro programmer typically needs to save current page to the file in order to read it again and to create a Table of contents or an Index. `\write` ⟨*file number*⟩ `{` ⟨*text*⟩ `}` saves ⟨*text*⟩ into memory and puts a pointer to this memory into the typesetting output. When the page is shipped out (by output routine), then all such pointers from this page are processed: the ⟨*text*⟩ is expanded at this time and its expansion is saved to the file. If (for example) the ⟨*text*⟩ includes `\the\pageno` then it is expanded to the correct page number of this page.
- `\closein` ⟨*file number*⟩, `\closeout` ⟨*file number*⟩ closes the open file. It is done automatically when TEX terminates its job.
- `\immediate` is a prefix. It can be used before `\openout`, `\write` and `\closeout` in order to do the desired action immediately (without waiting for the output routine).

Others primitive commands

- `\relax` does nothing. Used for terminating incomplete optional parameters, for example.
- `\begingroup` opens group, `\endgroup` closes group. The `{`$_1$ and `}`$_2$ do the same but moreover, they are syntactic constructors for primitive commands and math lists (in math mode). These two types of groups (declared by mentioned commands or by mentioned characters) cannot be mixed, i.e. `\begingroup...}` gives an error. Plain TEX declares `\bgroup` and `\egroup` control sequences as equivalents to `{`$_1$ and `}`$_2$. They can be used instead of `{`$_1$ and `}`$_2$ when we need to open/close a group, to create a math list, or when a box is constructed. For example, `\hbox\bgroup` ⟨*text*⟩ `\egroup` is syntactically correct.
- `\aftergroup` ⟨*token*⟩ saves the ⟨*token*⟩ and puts it back in the input queue immediately after the current group is closed. Then the expand processor expands it (if it is expandable). More `\aftergroup`s in one group create a queue of ⟨*token*⟩s used after the group is closed.
- `\afterassignment` ⟨*token*⟩ saves the ⟨*token*⟩ and puts it back immediately after a following assignment (⟨*register*⟩ `=` ⟨*value*⟩, `\def`, etc.) is done.
- `\lastskip`, `\lastpenalty` return the value of the last element in the current horizontal or vertical list if it is a glue/penalty. It returns zero if the element found is not the last.
- `\ignorespaces` ignores spaces in horizontal mode until the next primitive command occurs.

- `\mark{`⟨*text*⟩`}` saves ⟨*text*⟩ to memory and puts a pointer to it in the typesetting output. The ⟨*text*⟩ is used as expansion output of `\firstmark`, `\topmark` and `\botmark` expansion primitives in the output routine.
- `\parshape`⟨*number*⟩ ⟨*I*1⟩ ⟨*W*1⟩ ⟨*I*2⟩ ⟨*W*2⟩ . . . ⟨*In*⟩ ⟨*Wn*⟩ enables to set arbitrary shape of the paragraph. The ⟨*number*⟩ declares the amount of data: the ⟨*number*⟩ pairs of ⟨*dimen*⟩s follow. The i-th line of the paragraph is shifted by ⟨*Ii*⟩ to the right and its width is ⟨*Wi*⟩. The `\parshape` data are reset after each paragraph to zero values (normal paragraph).
- `\special{`⟨*text*⟩`}` puts the message ⟨*text*⟩ into the typesetting output. It behaves as a zero-dimension pointer to ⟨*text*⟩ and it can be read by printer drivers. It is recommended to not use this old technology when PDF output is created directly.
- `\shipout`⟨*box*⟩ outputs the ⟨*box*⟩ as one page. Used in the output routine.
- `\end` completes the last page and terminates the job.
- `\dump` dumps the memory image to a file named `\jobname.fmt` and terminates the job.
- `\patterns{`⟨*data*⟩`}` reads hyphenation patterns for the current `\language`.
- `\hyphenation{`⟨*data*⟩`}` reads hyphenation exceptions for current `\language`.
- `\message{`⟨*text*⟩`}` prints ⟨*text*⟩ on the terminal and to the log file.
- `\errmessage{`⟨*text*⟩`}` behaves like `\message{`⟨*text*⟩`}` but TEX treats it as an error.
- Job processing modes can be set by `\scrollmode` (don't pause at errors), `\nonstopmode` (don't pause at errors or missing files), `\batchmode` (`\nonstopmode` plus no output to the terminal). Default is `\errorstopmode` (stop at errors).
- `\inputlineno` includes the number of the current line from current file being input.
- `\show`⟨*control seq.*⟩, `\showbox`⟨*box num.*⟩, `\showlists`, `\showthe`⟨*register*⟩ are tracing commands. TEX prints desired result on the terminal and to the log file and pauses.

Commands specific for PDF output (available in pdfTEX, XƎTEX and LuaTEX)

- `\pdfliteral{`⟨*text*⟩`}` puts the ⟨*text*⟩ interpreted in a low level PDF language to the typesetting output. All PDF constructs defined in the PDF specification are allowed. The dimensions of the `\pdfliteral` object in the output are considered zero. So, if ⟨*text*⟩ moves the current typesetting point then the notion about its position from the TEX point of view differs from the real position. A good practice is to close ⟨*text*⟩ to `q...Q` PDF commands. The command `\pdfliteral` is typically used for generating graphics and for linear transformation.

- `\pdfcolorstack` ⟨*number*⟩ ⟨*op*⟩ `{` ⟨*text*⟩ `}` (where ⟨*op*⟩ is `push` or `pop` or `set`) behaves like `\pdfliteral{` ⟨*text*⟩ `}` and it is used for color switchers. For example when ⟨*text*⟩ is `1 0 0 rg` then the red color is selected. TEX sets the color stack at the top of each page to the color stack opened at the bottom of the previous page.
- `\pdfximage` `height` ⟨*dimen*⟩ `depth` ⟨*dimen*⟩ `width` ⟨*dimen*⟩ `page` ⟨*number*⟩ `{` ⟨*file name*⟩ `}` loads the image from ⟨*file name*⟩ to the PDF output and returns the number of such a data object in the `\pdflastximage` register. Allowed formats are PDF, JPG, PNG. The image is not drawn at this moment. A macro programmer can save `\mypic=\pdflastximage` and draw the image by `\pdfrefximage\mypic` (maybe repeatedly). Data of the image are loaded to the PDF output only once. The `\pdfximage` allows more parameters; see pdfTEX documentation.
- `\pdfsetmatrix {` ⟨*a*⟩ ⟨*b*⟩ ⟨*c*⟩ ⟨*d*⟩ `}` multiplies the current transformation matrix (for linear transformations) by `\matrix{` ⟨*a*⟩ `&` ⟨*c*⟩ `\cr` ⟨*b*⟩ `&` ⟨*d*⟩ `}`.
- `\pdfdest name{` ⟨*label*⟩ `}` ⟨*type*⟩ `\relax` declares a destination of a hyperlink. The ⟨*label*⟩ must match with the ⟨*label*⟩ used in `\pdfoutline` or `\pdfstartlink`. The ⟨*type*⟩ declares the behavior of the pdf viewer when the hyperlink is used. For example, `xyz` means without changes of the current zoom (if not specified). Other types should be `fit`, `fith`, `fitv`, `fitb`.
- `\pdfstartlink` `height` ⟨*dimen*⟩ `depth` ⟨*dimen*⟩ ⟨*attributes*⟩ `goto name{` ⟨*label*⟩ `}` declares the beginning of a hyperlink. A text (will be sensitive on mouse click) immediately follows and it is terminated by `\pdfendlink`. The height and depth of the sensitive area and the ⟨*label*⟩ used in `\pdfdest` are declared here. More parameters are allowed; see the pdfTEX documentation.
- `\pdfoutline` `goto name{` ⟨*label*⟩ `}` `count` ⟨*number*⟩ `{` ⟨*text*⟩ `}` creates one item with ⟨*text*⟩ in PDF outlines. ⟨*label*⟩ must be used somewhere by `\pdfdest name{` ⟨*label*⟩ `}`. The ⟨*number*⟩ is the number of direct descentants in the outlines tree.
- `\pdfinfo {` ⟨*key*⟩ `(` ⟨*text*⟩ `)}` saves to PDF the information which can be listed by the command `pdfinfo` ⟨*file*⟩`.pdf` on the command line for example. More ⟨*key*⟩ `(` ⟨*text*⟩ `)` should be here. The ⟨*key*⟩ can be `/Author`, `/Title`, `/Subject`, `/Keywords`, `/Creator`, `/Producer`, `/CreationDate`, `/ModDate`. The last two keywords need a special format of the ⟨*text*⟩ value. All ⟨*text*⟩ values (including ⟨*text*⟩ used in the `\pdfoutline`) must be ASCII encoded or they can use a very special PDFunicode encoding.
- `\pdfcatalog` enables us to set of a default behavior of the PDF viewer when it starts.
- `\pdfsavepos` saves an internal invisible point to the typesetting output. These points are processed when the page is shipped out: the numeric registers `\pdflastxpos` and `\pdflastypos` get values for the absolute position of this

invisible point (measured from the left upper corner of the page in `sp` units). The macro programmer can follow `\pdfsavepos` by the `\write` command and save these absolute positions to a text file which can be read in the next run of TEX in order to get these absolute positions by macros.

Microtypographical extensions (available in pdfTEX, LuaTEX and not all of them in XETEX)

- `\pdffontexpand` ⟨*font selector*⟩ ⟨*stretching*⟩ ⟨*shrinking*⟩ ⟨*step*⟩ declares a possibility to deform the characters from the font given by ⟨*font selector*⟩. This deformation is used when stretching or shrinking paragraph lines or doing `\hbox to{...}` in general. I.e. not only glues are stretchable and shrinkable. The numeric parameters are given in 1/1000 of the font size. ⟨*stretching*⟩ and ⟨*shrinking*⟩ are the maximum allowed values. The stretching or shrinking are not applied continuously but by the given ⟨*step*⟩. To activate this feature you must set the `\pdfadjustspacing` numeric register to a positive value.
- `\efcode` ⟨*font selector*⟩ ⟨*char. code*⟩ = ⟨*number*⟩ sets the degree of willigness of given character to be deformed when `\pdffontexpand` is used. Default value for all characters is 1000 and ⟨*number*⟩/1000 gives the proportion coefficient for stretching or shrinking of the character with respect to the "normal" deformation of characters with default value 1000.
- `\rpcode` ⟨*font sel.*⟩ ⟨*char. code*⟩ = ⟨*number*⟩, `\lpcode` ⟨*font sel.*⟩ ⟨*char. code*⟩ = ⟨*number*⟩ allows the declaration of hanging punctuation. Such punctuation is slightly moved to the right margin (if `\rpcode` is declared and the character is at the right margin) or to the left margin (for `\lpcode` by analogy). The ⟨*number*⟩ gives the amount of such movement in 1/1000 of the font size. To activate this feature you must set `\pdfprotrudechars` to a positive value (2 or more means a better algorithm).
- `\letterspacefont` ⟨*control seq.*⟩ ⟨*font selector*⟩ ⟨*number*⟩ declares a new font selector ⟨*control seq.*⟩ as a font given by the ⟨*font selector*⟩. Additional space declared by ⟨*number*⟩ is added between each two characters when the font is used. The ⟨*number*⟩ is 1/1000 of the font size. Unicode fonts support an analogous `letterspace=`⟨*number*⟩ font feature.
- The following commands have the same syntax as `\rpcode`: `\knbscode` (added space after the character), `\stbscode` (added stretchability of the glue after the character), `\shbscode` (added shrinkability after the character), `\knbccode` (added kern before the character), `\knaccode` (added kern after the character). To activate this feature you must to set `\pdfadjustinterwordglue` to a positive value. This feature is supported by pdfTEX only.

Commands used in math mode

- `\displaystyle`, `\textstyle`, `\scriptstyle`, `\scriptscriptstyle` switch to the specified style.

- `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose`, and `\mathpunct` followed by `{` ⟨*math list*⟩ `}` create a math object of the given type.
- `{` ⟨*numerator*⟩ `\over` ⟨*denominator*⟩ `}` creates a fraction. The primitive commands `\atop` (without fraction rule), `\above` ⟨*dimen*⟩ (fraction rule with given thickness) should be used in the same manner. The commands `\atopwithdelims`, `\overwithdelims`, `\abovewithdelims` allow us to specify brackets around the generalized fraction.
- `\left` ⟨*delimiter*⟩ ⟨*formula*⟩ `\right` ⟨*delimiter*⟩ creates a ⟨*formula*⟩ and gives ⟨*delimiter*⟩s around it with an appropriate size (compatible with the size of the formula). The ⟨*delimiter*⟩s are typically brackets.
- *`\middle` ⟨*delimiter*⟩ can be used inside the ⟨*formula*⟩ surronded by `\left`, `\right`. The given ⟨*delimiter*⟩ gets the same size as delimiters declared by appropriate `\left`, `\right`.
- Exponents and scripts are typically at the right side of the preceding math object. But if this object is a "big operator" (summation, integral) then exponents and scripts are printed above and below this operator. The commands `\limits`, `\nolimits`, `\displaylimits` used before exponents and scripts constructors (`^`$_7$ and `_`$_8$) declare an exception from this rule.
- `$$` ⟨*formula*⟩ `\eqno` ⟨*mark*⟩ `$$` puts the ⟨*mark*⟩ to the right margin as `\llap{$` ⟨*mark*⟩ `$}`. Analogously, `$$` ⟨*formula*⟩ `\leqno` ⟨*mark*⟩ `$$` puts it to the left margin.

14 Summary of plain TEX macros

Allocators

- `\newcount`, `\newdimen`, `\newskip`, `\newmuskip`, `\newtoks` followed by a ⟨*control seq.*⟩ allocate a new register of the given type and set it as the ⟨*control seq.*⟩. `\newbox`, `\newread`, `\newwrite` followed by a ⟨*control seq.*⟩ allocate a new address to given data (to a box register or to a file descriptor) and set it as the ⟨*control seq.*⟩. All these allocation macros are declared as `\outer` in plain TEX, unfortunately. This brings problems when you need to use them in skipped text or in macros (in ⟨*replacement text*⟩ for example). Use `\csname newdimen\endcsname \yoursequence` in such cases.
- `\newif` ⟨*control seq.*⟩ sets the ⟨*control seq.*⟩ as a boolean variable. It must begin with `if`; for example `\newif\ifsomething`. Then you can set values by `\somethingtrue` or `\somethingfalse` and you can use this variable by `\ifsoemthing` which behaves like other `\if*` primitive commands.

Vertical skips

- \bigskip does \vskip by one line, \medskip does \vskip by one half of a line and \smallskip does the vertical skip by one quarter of a line. The registers \bigskipamount, \medskipamount and \smallskipamount are allocated for this purpose.
- \nointerlineskip ignores the \baselineskip rule for the following box in the current vertical list. This box is appended immediately after the previous box. \offinterlineskip ignores the \baselineskip rule for all following boxes until the current group is closed.
- All vertical glues at the top of the page inserted by \vskip are ignored. Macro \vglue behaves like the \vskip primitive command but its glue is not ignored at the top of the page.
- Sometimes we must switch off the \baselineskip rule (for example by the \offinterlineskip macro). This is common in tables. But we need to keep the baseline distances equal. Then the \strut can be inserted on each line. It is an invisible box with zero width and with height+depth=\baselineskip.
- \normalbaselines sets the registers \baselineskip, \lineskip and \lineskiplimit for vertical placement to default values given by the format. The user can set other values for a while and then he/she can restore \normalbaselines.

Penalties

- \break puts penalty −10000, so a line/page break is forced here. \nobreak puts penalty 10000, so a line/page break is disabled here. It should be specified before a glue, which is "protected" by this penalty. \allowbreak puts penalty 0; it allows breaking similar to a normal space.
- \goodbreak puts penalty −500 in vertical mode, this is a "recommended" point for a page break.
- \filbreak breaks the page only if it is "almost full" or if a big object (that doesn't fit the current page) follows. The bottom of such a page is filled by a vertical glue, i.e. the default typographical rule about equal positions of all bottoms of common pages is broken here.
- \eject puts penalty −10000 in the vertical list, i.e. it breaks the page.

Miscellaneous macros

- \magstep⟨*number*⟩ expands to a magnification factor 1.2^x where x is the given ⟨*number*⟩. This follows old typographical traditions that all sizes (of fonts) are distinguished by factors 1, 1.2, 1.44, etc. For example, \magstep2 expands to 1440, because $1.2^2 = 1.44$ and 1000 is factor 1:1 in TEX. The \magstephalf macro expands to 1095 which corresponds to $1.2^{(1/2)}$.

- `\nonfrenchspacing` sets special space factor codes (bigger spaces after periods, commas, semicolons, etc.). This follows English typographical traditions. `\frenchspacing` sets all space factors as 1:1 (usable for non-English texts).
- `\endgraf` is equivalent to `\par`; `\bgroup` and `\egroup` are equivalents to `{`$_1$ and `}`$_2$.
- `\space` expands to space, `\empty` is an empty macro and `\null` is an empty `\hbox{}`.
- `\quad` is horizontal space 1 em (size of the font), `\qquad` is double `\quad`, `\enspace` is kern 0.5 em, `\thinspace` is kern 1/6 em, and `\negthinspace` makes kern −1/6 em.
- `\loop` ⟨*body 1*⟩ ⟨*if condition*⟩ ⟨*body 2*⟩ `\repeat` repeats ⟨*body 1*⟩ and ⟨*body 2*⟩ in a loop until the ⟨*if condition*⟩ returns false. Then ⟨*body 2*⟩ is not processed and the loop is finished.
- `\leavevmode` opens a paragraph like `\indent` but it does nothing if the horizontal mode is already in effect.
- `\line{` ⟨*text*⟩ `}` creates a box of line width (which is `\hsize`). `\leftline`, `\rightline`, `\centerline` do the same as `\line` but ⟨*text*⟩ is shifted left / right / is centered.
- `\rlap{` ⟨*text*⟩ `}` makes a box of zero size, the ⟨*text*⟩ is stuck out to the right. `\llap{` ⟨*text*⟩ `}` does the same and the ⟨*text*⟩ is pushed left.
- `\ialign` is equal to `\halign` but the values of the registers used by `\halign` are set to default.
- `\hang` starts the paragraph where all lines (except for the first) are indented by `\parindent`.
- `\texindent{` ⟨*mark*⟩ `}` starts a paragraph with `\llap{` ⟨*mark*⟩ `}`.
- `\item{` ⟨*mark*⟩ `}` starts paragraph with `\hang` and `\llap{` ⟨*mark*⟩ `}`. Usable for item lists. `\itemitem{` ⟨*mark*⟩ `}` can be used for the second level of items.
- `\narrower` sets wider margins for paragraphs (`\parindent` is appended to both sides); i.e. the paragraphs are narrower.
- `\raggedright` sets the paragraph shape with the ragged right margin. `\raggedbottom` sets the page-setting shape with the ragged bottoms.

Floating objects

- `\footnote{` ⟨*mark*⟩ `}{` ⟨*text*⟩ `}` creates a footnote with given ⟨*mark*⟩ and ⟨*text*⟩.
- `\topinsert` ⟨*object*⟩ `\endinsert` creates the ⟨*object*⟩ as a *floating object*. It is printed at the top of the current page or on the next page. `\midinsert` ⟨*object*⟩ `\endinsert` does the same as `\topinsert` but it tries if the ⟨*object*⟩ fits on the current page. If it is true then it is printed to its current position; no floating object is created.

Controlling of input, output

- `\obeyspaces` sets the space as normal, i.e. it deactivates special treatment of spaces by the tokenizer: more spaces will be more spaces and spaces at the beginning of the line are not ignored.
- `\obeylines` sets the end of each line as `\par`. Each line in the input is one paragraph in the output.
- `\bye` finalizes the last page (or last pages if more floating objects must be printed) and terminates TEX job. The `\end` primitive command does the same but without worrying about floating objects.

Macros used in math modes

- Spaces in math mode are `\,` (thin space), `\>` (medium space) `\;` (thick space, but still small), `\!` (negative thin space).
- `{` ⟨*above*⟩ `\choose` ⟨*below*⟩ `}` creates a combination number with brackets around it.
- `\sqrt{` ⟨*math list*⟩ `}` creates the square root symbol with the ⟨*math list*⟩ under it.
- `\root` ⟨*n*⟩ `\of{` ⟨*math list*⟩ `}` creates a general root symbol with the order of the root ⟨*n*⟩.
- `\cases{` ⟨*case 1*⟩ `&` ⟨*cond. 1*⟩ `\cr...\cr` ⟨*case n*⟩ `&` ⟨*cond. n*⟩ `}` creates a list of variants (preceded by a brace {) in math mode.
- `\matrix{` ⟨*a*⟩ `&` ⟨*b*⟩ `...&` ⟨*e*⟩ `\cr...\cr` ⟨*u*⟩ `&` ⟨*v*⟩ `...&` ⟨*z*⟩ `}` creates a matrix of given values in math mode (without brackets around it). `\pmatrix{` ⟨*data*⟩ `}` does the same but with ().
- `$$\displaylines{` ⟨*formula 1*⟩ `\cr...\cr` ⟨*formula n*⟩ `}$$` prints multiple (centered) formulae in display mode.
- `$$\eqalign{` ⟨*formula 1 left*⟩ `&` ⟨*formula 1 right*⟩ `\cr...\cr` ⟨*formula n left*⟩ `&` ⟨*formula n right*⟩ `}$$` prints multiple formulae aligned by `&` character in display mode.
- `\eqalignno` behaves like `\eqalign` but a second `&` followed by a ⟨*mark*⟩ can be in some lines. These lines place the ⟨*mark*⟩ in the right margin. `\leqalignno` does the same as `\eqalignno` but ⟨*mark*⟩ is put to the left margin.

Index

Petr Olšák, Czech Technical University in Prague
petr@olsak.net

TEX v kostce

Uživatelé dnes objevují TEX přes vysokoúrovňové formáty, které pečlivě skrývají složitost počítačové sazby za fasádou přívětivých značkovacích jazyků. Nicméně jakékoliv složitější sazečské úkoly vyžadují, aby uživatelé věděli, co se skrývá pod kapotou a jak mohou algoritmy TEXu ovlivnit, pokud je to zrovna potřeba.

Autor ve svém článku představuje základy, na kterých stojí většina dnešních TEXových formátů a které čtenářům pomohou s každodenní prací v TEXu i se záludnějšími sazečskými úkony. Čtenáři se nejprve seznámí s programem TEX a s jeho rozšířeními. Následně se dozví o procesorech TEXu a jejich režimech. Na závěr zjistí, jaké existují registry a primitivy TEXu a jaká makra nabízí formát plain TEX. Heslem dne je stručnost a autorův výklad zabírá pouze necelých 40 stran textu. Díky tomu se TEXovým mágem nebo mágyní můžete stát během jedné cesty vlakem!

Autor v minulosti napsal již tři knihy o TEXu, vyvinul formát OpTEX, udržuje množství balíčků na archivu CTAN a již více než dvacet let vyučuje vysokoškolský předmět o digitální sazbě a TEXu.

Klíčová slova: TEX, ε-TEX, pdfTEX, XeTeX, LuaTEX, mikrotypografie, plain TEX

Úpravy TEXu v roce 2021

Donald Knuth

Článek pojednává o změnách provedených v roce 2021 ve zdrojových souborech TEXu, METAFONTu a přidružených programů.

Klíčová slova: TEX, METAFONT, `tex.web`, `mf.web`

Tento článek přináší pokračování předcházejících příspěvků z let 2008 [2] a 2014 [3]. Opět jsem nesmírně vděčný všem, kteří mě upozornili na možné chyby v jádrech TEXu a METAFONTu, a také báječnému týmu expertů, vedenému Karlem Berrym, kteří pečlivě prošli všechna tato upozornění a vyfiltrovali z nich seznam těch, která opravdu vyžadují pozornost. Podle našeho dlouholetého plánu jsem tento seznam dostal 31. prosince 2020.

Karl napsal samostatný příspěvek [4] o své roli metafiltru. Ještě připomenu, že když jsem před sedmi lety dělal předchozí kolo úprav, musel jsem se vypořádat s „více než dvěma desítkami potenciálně problémovými místy“ [3]. Tentokráte ten počet byl přes 250!

V letech 2008 a 2014 došlo k drobným úpravám TEXu i METAFONTu a oběma programům tak přibyla v číslu verze jedna cifra. Dobrou zprávou je, že tyto úpravy jsou prakticky nepostřehnutelné. Nedá mi to a musím znovu citovat text z [2], protože odráží mou neměnnou filosofii (viz [5]):

> Rejstřík knihy *Digital Typography* uvádí jedenáct stránek, na kterých je zdůrazněno, že je nutné, aby zdrojový kód zůstal stabilní, a já naléhám na všechny vývojáře TEXu a METAFONTu, aby tyto stránky každých pár let četli. Každý objekt, který není úplně triviální, nemůže být optimální, v tom smyslu, že může být nějakým způsobem vylepšen (přičemž stále zůstane neoptimální). Proto vždy existuje důvod změnit něco, co není triviální. Nicméně jedna ze základních výhod TEXu je, že se nemění – s výjimkou oprav vážných chyb, které se ale velmi pravděpodobně dotknou pouze několika archivních dokumentů.

Uživatelé si stále mohou být jisti, že jsem v tomto kole úprav nepokazil nic, co předtím fungovalo. A kdo chce, může si TEX a METAFONT aktualizovat. Kdo nechce, nemusí.

Z anglického originálu [1] přeložil Jan Šustek.

 doi: 10.5300/2021-1-4/56

1. TEX verze 3.141592653

Pojďme se podívat na konkrétní detaily. Nová verze TEXu se od předchozí liší na pěti ne úplně triviálních místech. Většinou se úpravy týkají chybně naprogramovaných pokusů o zotavení se z chyby.

První dvě zvláštnosti objevil Xiaosa Zhang a oznámil to loni v létě na `tex.stackexchange` [6, 7]. Našel záludnou kombinaci kláves, při které loňský TEX umožnil uživateli dostat se do `\batchmode`, zatímco dále reagoval přes terminál! Dále zjistil, že možnost po chybě editovat zdrojový soubor (volba `E` v reakci na chybové hlášení) byla nabízena i tehdy, kdy by neměla, a to v okamžiku, kdy žádný vstupní soubor nebyl načítán.

Obě tyto chyby mohly způsobit, že TEX zhavaruje. Oboje tato vrátka jsou nyní již zavřená.

Jiné zvláštnosti si všiml v roce 2017 Udo Wermuth. Zjistil, že TEX může zdánlivě zamrznout při trasování, když je aktivní `\tracingparagraphs`. Důvod byl, že TEX našel a oznámil chybu a toto zapsal do log souboru. Pak TEX tiše čekal, než uživatel zareaguje, přičemž si neuvědomil, že chybová hlášení se na terminál nevypisují v době, kdy se trasují odstavce. Nově TEX nezůstane potichu. Uživatel uvidí chybové hlášení a výzvu k reakci.

Loni Udo narazil na zcela jiný druh chyby. Tato chyba neměla nic společného s uživatelovou interakcí a teoreticky na ni mohli narazit i jiní uživatelé při nějakém „skutečném“ běhu TEXu v průběhu posledních 35 let (pochybuji ale o tom). Předchozí verze TEXu nesprávně umožňovaly, aby ⟨*replacement text*⟩ v definici makra začínal okamžitě po `#\bgroup`, v rozporu s pravidlem jednoznačně popsaným v *TEXbook*u [8] na straně 275.

Odteď bude TEX důsledně postupovat podle tohoto pravidla. Pokud někdo dříve napsal

```
1 \def\makro#1#\bgroup ahoj#1}
```

nyní obdrží chybové hlášení. Nově je třeba pro stejné fungování napsat

```
2 \def\makro#1\bgroup{ahoj#1\bgroup}
```

A konečně 22. října 2020 Bruno Le Floch popsal chybu, které se možná v budoucnu bude říkat[1] „final bug in TEX“. Opět se týká maker. Předpokládejme, že jsme v definici makra použili devět parametrů `#1` až `#9`. V tuto chvíli už TEX další parametr ve zbytku části ⟨*parameter text*⟩ neočekává, protože devět parametrů je horní mez TEXu. Pokud nyní nesprávně použijeme `#`, TEX ohlásí chybu

```
3 ! You already have nine parameters.
```

a k ní příslušnou nápovědu

```
4 I'm going to ignore the # sign you just used.
```

[1] Překladatel se rozhodl uvedený pojem ponechat v originálním znění. (pozn. překl.)

Nápověda říká pravdu. Ale odteď bude nápověda definovat novou pravdu, a sice že TEX bude ignorovat také token, který následuje za nesprávným `#`. Teď už se nesprávný token nedostane dále do zpracování a nezpůsobí další problémy.

Všech pět uvedených chyb si vysloužilo šek na 327,68 dolarů (`0x$80.00`) v Bank of San Serriffe [9], protože uživatelé poukázali na závažné (i když velmi vzácně se vyskytující) nedostatky v implementaci TEXu. Kromě nich je v TEXu verze 3.141592653 zapracováno množství relativně malých oprav. Například předchozí verze TEXu mohly zamotat log soubor při nastaveném `\newlinechar=`p`.

Kvůli konzistenci se změnil také Plain TEX. Nyní je zaručeno, že `\muskip255` a `\toks255` jsou jen pomocné registry, které se nikdy nealokují přes `\newmuskip` a `\newtoks`. V nové verzi má `\fmtversion` hodnotu `3.1415926535`.

Ty méně triviální z uvedených změn jsou uvedeny v aktualizaci knihy *TEX: The Program* [10], která je nyní online ve formátu PDF na stránce [11], a v souboru `errata.tex`. Také jsou uvedeny v souborech `errorlog.tex`, `tex82.bug` a `plain.tex`. Ale úplná pravda je, jako vždy, pouze v aktualizovaném hlavním zdrojovém souboru `tex.web`. Všech pět těchto klíčových souborů i nadále bude k dispozici online v adresáři `systems/knuth/dist` v archivu CTAN [12].

Seznam všech chyb TEXu vznikl v roce 1978 a jeho prvních 14 let je dokumentováno v [13], kapitoly 10 a 11. Dalších pár let je pokryto v [5], kapitola 34, konče chybou 933 datovanou 10. března 1995 a objevenou Peterem Breitenlohnerem. No a aktuálně poslední chyba 957 možná nakonec bude ta úplně poslední. Kdo ví.

Když jsem prováděl aktuální úpravy, byla radost vidět, jak to s přístupem pomocí dokumentovaného programování všechno šlo jednoduše. Tento komplikovaný program byl napsán před 40 lety a i nyní je možné dostat se do jeho nejtemnějších míst bez problémů, stačí se pouze začíst do [10] a použít jeho rejstříky a minirejstříky. Nemůžu si pomoct, většinu úspěchu TEXu připisuji právě faktu, že byl napsán pomocí dokumentovaného programování.

2. METAFONT verze 2.71828182

A co kamarád TEXu? Už jsem si skoro myslel, že číslo verze METAFONTu zůstane 2.7182818, protože výstupy nově aktualizovaného programu se neliší od výstupů předchozí verze, až na nějaké triviální detaily. Například některá chybová hlášení se trochu změnila.

Nicméně výše uvedené dvě chyby TEXu objevené Xiaosou Zhangem se týkají také METAFONTu. Nyní věřím, že „final bug in METAFONT“ byl nalezen 3. července 2020 v [7], přestože se ve skutečnosti tato chyba týkala TEXu.

3. Přidružené programy

Také u více než desítky příbuzných programů jsem udělal drobné změny v jejich hlavních `web` souborech, zejména jsem opravil překlepy, přidal jsem oxfordské čárky a udělal jsem soubory mezi sebou více konzistentní. V programech TANGLE a WEAVE našli dvě nenápadné chyby Doug McKenna a David Fuchs, přičemž těchto chyb si nikdo nevšiml od začátku 80. let!

Pro přehlednost uvádím seznam všech `web` souborů, za které jsem zodpovědný:

soubor	aktuální verze	datum
`dvitype.web`	3.6	prosinec 1995
`gftodvi.web`	3.0	říjen 1989
`gftopk.web`	2.4	leden 2014
`gftype.web`	3.1	březen 1991
`mf.web`	2.71828182	leden 2021
`mft.web`	2.1	leden 2021
`pltotf.web`	3.6	leden 2014
`pooltype.web`	3.0	září 1989
`tangle.web`	4.6	leden 2021
`tex.web`	3.141592653	leden 2021
`tftopl.web`	3.3	leden 2014
`vftovp.web`	1.4	leden 2014
`vptovf.web`	1.6	leden 2014
`weave.web`	4.5	leden 2021

4. Typografické a další chyby

Dosud jsme se zabývali chybami, které se vyskytovaly v jednotlivých programech. Ale čtenáři samozřejmě hlásili také problémová místa v dokumentaci – ta se ve skutečnosti opravují nejhůře. *The TEXbook* [8] byl pod intenzivní kontrolou čtenářů po téměř čtyřicet let. A čtenáři z celého světa neustále posílali náměty, jak knihu vylepšit. Například posílali odpovědi na některá složitější cvičení.

Největší změny *TEXbook*u se týkaly detailů vkládání výplňků do matematických vzorců. Můj původní rozbor pomocí „vnitřních atomů“ se ukázal zcela chybný. Očividně si toho ale nikdo nevšiml až do prosince 2018, kdy Sophie Alpert poukázala na výrazné nesoulady v Dodatku G. Několik stránek textu se muselo přepsat a jsem samozřejmě rád, že uvedené detaily jsou nyní popsány správně.

Další významné změny se týkaly zpřesnění popisu syntaxe příkazů souvisejících s dělením slov. Hodně změn se udělalo také v rejstříku. Dohromady se změny týkaly 93 ze 483 stran *TEXbook*u, tj. zhruba 19 % stran.

Ještě více změn potkalo *The METAFONTbook* – dokonce 128 z 361 stran, tj. zhruba 35 % stran. Překlep se objevil i v rejstříku! Hlavní dva přispěvatelé do tohoto seznamu chyb, Hu Yajie a Udo Wermuth, zcela jistě patří mezi nejlepší korektory na světě. Konkrétně Yajie kromě mnoha vzájemně ortogonálních způsobů vylepšení knihy také pomáhal se zjednodušením formální syntaxe výrazů v METAFONTu.

5. Výročí *Computers & Typesetting*

Jeden z významných dnů mého života nastal 21. května 1986, když vydavatelství Addison-Wesley uspořádalo celodenní událost [14] v Bostonském počítačovém muzeu, aby oslavilo dokončení TEXu a METAFONTu. Bylo to poprvé, co jsem uviděl své knihy [8, 10, 15, 16, 17], které doslova byly ještě horké. A můj nejpříjemnější zážitek z toho dne byl, když spoluzakladatel tohoto vydavatelství Mel Cummings držel těchto pět knih v ruce, přičemž neskrýval svou spokojenost a hrdost. Mel totiž celý život strávil v tiskařském průmyslu a vytvářel technické knihy nejvyšší kvality – proto jsem byl potěšen, když jsem viděl jeho nadšení.

Když jsem si nyní znovu pročítal každou z 2668 stran těchto knih, neustále jsem cítil hrdost, že jsem mohl být součástí tohoto mimořádného společného počinu, zejména teď, když tyto knihy dosáhly nového vrcholu dokonalosti. Zdá se férové říct, že tyto knihy tvoří důležitý milník v historii typografie, protože samy popisují všechny detaily výpočtů použitých při jejich sazbě. „Kdyby se kopie těchto knih poslaly na Mars, byli by podle nich Marťani schopni sestavit posloupnost nul a jedniček, podle kterých se ty knihy vytiskly.“ [14]

Proto jsem nesmírně potěšen, že mohu oznámit, že vydavatelství Addison-Wesley právě vydalo jako „35th Jubilee Edition“ zcela nové výtisky svazků A, B, C a D, v nichž jsou zapracovány všechny úpravy provedené k únoru 2021. Konečně všechna „i“ mají správně tečku a všechna „t“ mají správně příčku! (Výtisk svazku E z roku 2017 zůstal nezměněn.)

6. Závěr

Celá rodina programů kolem TEXu se stále zdá být zdravá a krásná a neustále se přibližuje k úplné dokonalosti. Je prakticky nulová šance, že by nějaký dokument vytvořený předchozími verzemi TEXu nebo METAFONTu byl ovlivněn aktuálně provedenými změnami. A pořád existuje množství dobrovolníků, kteří k tomuto úspěchu přispívají.

Připravte se na Úpravy TEXu v roce 2029?!

Odkazy

1. KNUTH, Donald E. The TEX tuneup of 2021. *TUGboat*. 2021, roč. 42, č. 1, s. 7–10. Dostupné z DOI: 10.47397/tb/42-1/tb130knuth-tuneup21.
2. KNUTH, Donald E. The TEX tuneup of 2008. *TUGboat* [online]. 2008, roč. 29, č. 2, s. 233–238 [cit. 2021-12-06]. Dostupné z: https://tug.org/TUGboat/tb29-2/tb92knut.pdf.
3. KNUTH, Donald E. The TEX tuneup of 2014. *TUGboat* [online]. 2014, roč. 35, č. 1, s. 5–8 [cit. 2021-12-06]. Dostupné z: https://tug.org/TUGboat/tb35-1/tb109knut.pdf.
4. BERRY, Karl. TEX entomology in 2021. *TUGboat*. 2021, roč. 42, č. 1, s. 10. Dostupné z DOI: 10.47397/tb/42-1/tb130berry-filter.
5. KNUTH, Donald E. *Digital Typography*. Stanford, CA: Center for the Study of Language a Information (CSLI), 1999. CSLI Lecture Notes, č. 78. Druhé vydání (2012) obsahuje množství oprav.
6. 潇洒张. *Why there is a "Text line contains an invalid character" after "Undefined control sequence" and why "Q" required further input?* [Online]. Stack Exchange, 2020-07-27 [cit. 2021-12-06]. Dostupné z: https://tex.stackexchange.com/q/551313.
7. 潇洒张. *I need to understand some cmds of tex* [online]. Stack Exchange, 2020-07-03 [cit. 2021-12-06]. Dostupné z: https://tex.stackexchange.com/q/552113.
8. KNUTH, Donald E. *The TEXbook*. Reading, MA: Addison-Wesley, 1984. Computers & Typesetting. V současnosti jsou dostupné 35. výtisk (měkká vazba, 2017) a 23. výtisk (pevná vazba, 2021).
9. KNUTH, Donald E. *The Bank of San Serriffe* [online]. Stanford, CA: Stanford University, 2021-01-21 [cit. 2021-03-31]. Dostupné z: https://www-cs-faculty.stanford.edu/~knuth/boss.html.
10. KNUTH, Donald E. *TEX: The Program*. Reading, MA: Addison-Wesley, 1986. Computers & Typesetting. Od pátého výtisku (1994) xvi+600 stran. V současnosti je dostupný 11. výtisk (pevná vazba, 2021).
11. KNUTH, Donald E. *Computers & Typesetting* [online]. Stanford, CA: Stanford University [cit. 2021-12-06]. Dostupné z: https://www-cs-faculty.stanford.edu/~knuth/abcde.html.
12. CTAN TEAM. *CTAN: Comprehensive TEX Archive Network* [online]. DANTE [cit. 2021-12-06]. Dostupné z: https://ctan.org/.
13. KNUTH, Donald E. *Literate Programming*. Stanford, CA: Center for the Study of Language a Information (CSLI), 1992. CSLI Lecture Notes, č. 27.
14. BEETON, Barbara; GORDON, Peter; KNUTH, Donald E. Computers & Typesetting [coming out party]. *TUGboat* [online]. 1986, roč. 7, č. 2, s. 93–98 [cit. 2021-12-06]. Dostupné z: https://tug.org/TUGboat/tb07-2/

tb15knut.pdf. Své poznámky s dodatky jsem také zveřejnil jako kapitolu 28 v [5].
15. KNUTH, Donald E. *The METAFONT book*. Reading, MA: Addison-Wesley, 1986. Computers & Typesetting. V současnosti jsou dostupné 14. výtisk (měkká vazba, 2017) a 10. výtisk (pevná vazba, 2021).
16. KNUTH, Donald E. *METAFONT: The Program*. Reading, MA: Addison-Wesley, 1986. Computers & Typesetting. Od třetího výtisku (1991) xvi + 566 stran. V současnosti je dostupný 9. výtisk (pevná vazba, 2021).
17. KNUTH, Donald E. *Computer Modern Typefaces*. Reading, MA: Addison-Wesley, 1986. Computers & Typesetting. V současnosti je dostupný 8. výtisk (pevná vazba, 2017).

Summary: The TEX Tuneup of 2021

This paper describes the corrections made in 2021 in the source files of TEX, METAFONT and associated programs.

Keywords: TEX, METAFONT, `tex.web`, `mf.web`, tuneup

Donald Knuth, www-cs-faculty.stanford.edu/~knuth

Ladění LaTeXových souborů

Barbara Beeton

Každý uživatel LaTeXu se alespoň jednou během své práce setkal s problémem, kdy z nějakého důvodu LaTeX dokument nepřeloží. Jak se vypořádat s jednoduchými problémy v LaTeXu, je poměrně dobře známo, ale jsou i situace, na které běžné metody nestačí.

V tomto článku si představíme postupy řešení několika typů problémů, které se objevily během dlouholeté praxe člena technické podpory Americké matematické společnosti, který vyřizuje dotazy jak od autorů, tak i od samotných editorů. Podíváme se na obvyklé i méně časté problémy, se kterými se při své práci setkává. A se kterými se může setkat každý uživatel LaTeXu.

Klíčová slova: LaTeX, ladění, chyby, log soubor

1. Úvod

V roce 2016 Americká matematická společnost publikovala řádově 60 000 stran knih a časopisů. Většina z nich byla dodána v LaTeXových souborech, které vytvořili přímo příslušní autoři. Aby mohl být dokument přijat k tisku, musí mít vědecký přínos a musí být posouzen recenzenty. Autor může článek dodat ve formě elektronické nebo papírové, dokonce může být napsán i ručně. Není důležitý vzhled dokumentu, ale pouze jeho obsah. Knihy jsou smluvně zajišťovány zaměstnanci, z nichž všichni jsou profesionální matematici obeznámení s LaTeXem, ale v žádném případě to nejsou odborníci na TeX. S veškerými příchozími dokumenty se pak redakce musí nějak vypořádat.

Jestliže dodaný dokument není napsán v LaTeXu, přepíše jej do LaTeXu nějaká kompetentní osoba. Proto dále v článku předpokládejme, že dokument již je v LaTeXu.

Kvalita dodaných zdrojových souborů se napříč dokumenty velmi liší. To poskytuje velké množství příležitostí k testování a vylepšování ladicích dovedností členů redakce.

Z anglického originálu [1] přeložili Lucie Šustková a Jan Šustek.

doi: 10.5300/2021-1-4/63

Redakční práce probíhají na síti počítačů s Linuxem. Použité balíky maker spadají do tří skupin:

- balíčky patřící do TEXlive, které se aktualizují maximálně jednou ročně,
- lokální upravené verze balíčků a fontů (někdy také nové verze balíčků, které budou v následujícím vydání TEXlive),
- vlastní lokální makra AMS.

Všechny soubory jsou archivovány v systému Subversion. Archivy publikovaných knih a článků sahají desítky let do minulosti. U každé publikace jsou balíčkem snapshot zaznamenány aktuální verze LATEXu a všech použitých souborů – v případě potřeby je možné reprodukovat původní soubory a definice. Tento postup práce poskytuje stabilitu při zpracování dotisků, při dalších vydáních knih a při převodu existujících publikací do jiných formátů, jako jsou například elektronické knihy.

Výše uvedený postup však může být funkční a spolehlivý až v okamžiku, kdy jsou všechny soubory bez chyb a dokument je připravený k vytištění. Ale před tímto šťastným okamžikem se může přihodit spousta věcí. O nich pojednává tento článek, který navazuje na přednášku autorky na konferenci TUG@BachoTEX 2017.

V případě libovolné chyby v dokumentu platí hlavní obecná zásada: dostat se rychle a spolehlivě zpět do stavu, kdy ještě chyba nenastala.

2. Příprava dokumentů – plánujte dopředu

Existují určité zásady, které, pokud se důsledně dodržují, mohou z dlouhodobého hlediska dost usnadnit práci. První z nich je vybrat si vhodné nástroje a naučit se je používat.

Nejdůležitějším nástrojem je kvalitní textový editor nebo vývojové prostředí. Autorka tohoto článku používá emacs. Jsou však jiné možnosti. Některé jsou vhodné pro práci jediného uživatele na jednom počítači, jiné pro společnou práci více autorů online, a spousta dalších někde mezi tím. Seznam možných nástrojů lze najít v odpovědi na otázku položenou na TEX Stack Exchange [2] (dále jen „`tex.sx`“).[1]

Autorka tohoto článku dává přednost zpracování souborů z příkazového řádku. To umožňuje interaktivně opravit jednoduché chyby, jako jsou překlepy v názvech řídicích sekvencí, a tím se vyhnout hromadě navazujících chybových hlášení způsobených jedinou triviální chybou. Samozřejmě je v takovém případě důležité před dalším překladem stejným způsobem opravit zdrojový soubor.

Mezi funkce nejužitečnější pro ladění patří tyto:

- dobré možnosti vyhledávání,
- párování závorek a `\begin`–`\end`,

[1] LATEX Editors/IDEs, `http://tex.stackexchange.com/q/339`

- více oken viditelných současně,
- možnost přejít na konkrétní číslo řádku,
- určení počtu nalezených řetězců.

Dále je třeba dát si pozor na to, jak jsou adresáře a soubory umístěny a pojmenovány. Je vhodné se vyhnout mezerám v názvech souborů – ne všechny operační systémy dokážou takové soubory zpracovat. Podobně některé operační systémy rozlišují velká a malá písmena – aby se předešlo problémům, je doporučeno používat pro názvy souborů pouze malá písmena, případně číslice a spojovníky. Je dobré se vyhnout tečkám a znakům, které mají v TEXu zvláštní význam (například podtržítko).

Vyplatí se udržovat soubory v přijatelné velikosti. Pro velký dokument, například knihu nebo diplomovou práci, umístěte každou kapitolu do samostatného souboru a mějte jeden řídicí soubor, který bude jednotlivé soubory načítat makrem `\include`. S využitím LATEXového makra `\includeonly` vám to umožní pracovat v daný čas pouze na jedné kapitole. Pokud máte velké tabulky nebo obrázky, opět se vyplatí umístit je do samostatných souborů, protože pak můžeme celou tabulku nebo obrázek skrýt jediným procentem. (Další výhodou je jednodušší přesun na jiné místo v dokumentu v případě potřeby.)

Když editujete soubory, bývá dobrým zvykem ukončit načítané soubory příkazem `\endinput` umístěným na samostatném řádku. Tím se vyhnete problémům s načítáním případných dalších znaků, které se neohlášeně někdy přidají na konec souboru, když se tento soubor přenáší z jednoho systému na jiný. Do načítaných souborů však nikdy nevkládejte `\end{document}`.

Zjistěte si, kde se nachází log soubor. Některá vývojová prostředí jej před uživatelem ukrývají. Když se vám začne hroutit překlad a vy se nebudete moct podívat do log souboru a zjistit, v čem je problém, pak vás čekají hodně horké chvilky, než se vám podaří problém opravit.

A nikdy neaktualizujte svůj systém při práci na důležitém projektu. Nové verze balíčků mohou mít nové nekompatibilní vlastnosti a staré balíčky již nemusejí fungovat. Tato rada je samozřejmě k ničemu, když vám zrovna dosloužil hardware. Ale vy si přece děláte průběžné zálohy všech souborů, že ano?

3. Testujte ve vedlejších souborech

Jestliže chyba je složitější než jen překlep v názvu řídicí sekvence, snažte se předejít možným komplikacím – udělejte si svoje místo pro ladění. Ze všeho nejdříve zálohujte své soubory nebo celou pracovní adresářovou strukturu a zálohy umístěte na bezpečné místo. Vy víte, jaký je aktuální stav vašeho dokumentu, a do tohoto stavu byste se rádi zpátky dostali, pokud by ladění neprobíhalo úspěšně. Za žádných okolností nedělejte experimentální změny v souborech, od nichž nemáte kopii. Úplně nejlépe si vytvořte nový adresář a dělejte experimenty v něm.

Jestliže se dokument skládá z více než jednoho zdrojového souboru, začněte zkopírováním pouze řídicího souboru – toho, který načítá další soubory – do tohoto nového adresáře. K dalším souborům přistupujte pomocí odkazů. Na Linuxu je na to příkaz[2]

`ln -s` ⟨*soubor*⟩ ⟨*odkaz*⟩

Dokument překládejte LaTeXem interaktivně. Můžete tak snadno opravit jednoduché chyby ještě předtím, než zapříčiní další chyby a nejasná chybová hlášení. (Nezapomeňte pak opravit chybu jak v testovacím, tak v původním souboru.) A jestliže je chyba složitější a nelze opravit interaktivně (jako například neznámé nebo neukončené prostředí), pak lze úlohu ihned přerušit a před pokračováním problém vyřešit.

Pokud dokument překládáte v nonstop módu (což je obvyklá situace, když pracujete ve vývojovém prostředí), pak se seznam všech chyb (do maximálního počtu 100 chyb v jednom odstavci) zapíše do log souboru. I jediná chyba ale může způsobit množství dalších chybových situací a hlášení, které by nevznikly, kdyby nedošlo k prvotní chybě.

4. Některé nástroje pro interaktivní diagnostiku

V TeXu jsou k dispozici následující diagnostické příkazy, které požadovanou informaci vypíšou na terminál i do log souboru:

- Příkaz `\message{...}` vypíše danou zprávu na terminál i do log souboru. Může být použitý k informování uživatele, že se překlad dokumentu dostal až do konkrétního místa. Například

  ```
  \message{poslední sekce, strana \number\thepage^^J}
  ```

  vypíše

  ```
  poslední sekce, strana 66
  ```

- Příkaz `\show` vypíše aktuální význam tokenu. Zpracování dokumentu se pozastaví a čeká se na uživatelovu interakci. Například

  ```
  \show\LaTeX
  ```

  vypíše

  ```
  > \LaTeX=macro:
  ->\protect \LaTeX  .
  ```

[2] Jak v jiných operačních systémech udělat odkaz na soubor, to si čtenář může snadno dohledat na internetu.

Nebo

```
\show\protect
```

vypíše

```
> \protect=\relax.
```

Když uživatel odpoví `i`, může na aktuální místo zdrojového souboru vložit konkrétní text. Když pouze stiskne enter, bude překlad dokumentu pokračovat.

- Příkaz `\showthe` vypíše aktuální hodnotu registru. Stejně jako v předchozím případě se překlad pozastaví. Například

```
\showthe\hfuzz
```

vypíše

```
> 0.1pt.
```

Další diagnostické příkazy vypíšou informaci pouze do log souboru. (Tyto příkazy většinou vypíšou více informací, než potřebujeme, proto je třeba je používat s rozumem.) Z těchto příkazů autorka tohoto článku nejčastěji používá následující:

- Po nastavení registru `\tracingoutput` na kladnou hodnotu se vypíše obsah všech boxů na aktuální stránce.[3]
- Po nastavení registrů `\tracingmacros` a `\tracingcommands` na kladnou hodnotu se vypíšou detaily o prováděné expanzi maker a o prováděných příkazech hlavního procesoru.
- Registr `\errorcontextlines` určuje, kolik se maximálně vypíše řádků s detaily expanze, kterou TEX aktuálně v okamžiku chyby provádí. Je dobré nastavit hodnotu tohoto registru na více než 5, aby bylo možné chybu dobře diagnostikovat i v případě složité expanze.[4]
- Po nastavení registru `\tracingonline` na kladnou hodnotu budou všechny příkazy `\tracing...` vypisovat informace jak do log souboru, tak na terminál.

Detaily použití všech diagnostických příkazů lze najít v *TEXbooku* [3] nebo v knize *TEX by topic*[5] [4], případně v češtině v *TEXbooku naruby* [5].

[3] Množství vypsaného obsahu boxu a počet vypsaných úrovní boxů jsou určeny registry `\showboxbreadth` a `\showboxdepth`. Tyto registry je nutno nastavit na kladnou hodnotu. Implicitně jsou v LATEXu oba registry nastaveny na hodnotu −1, což znamená, že se žádný obsah boxů nevypíše. (pozn. překl.)

[4] Implicitně je registr `\errorcontextlines` v LATEXu nastaven na hodnotu −1, což znamená, že se žádné detaily expanze nevypíšou. (pozn. překl.)

[5] Kniha je dostupná také v TEXlive, v terminálu stačí napsat `texdoc texbytopic`.

5. Log soubor je váš kamarád

V log souboru (LA)TEXu jsou zaznamenány všechny prováděné aktivity – který soubor nebo font se aktuálně načítá, jaká čísla boxů a dalších registrů se přiřazují do aktuální řídicí sekvence nebo které důležité příkazy jsou právě předefinovávány. Z pohledu ladění je asi nejdůležitější, že jsou v log souboru do detailu zaznamenány chyby a je u nich uvedeno číslo řádku ve zdrojovém souboru.

Vždy si zkontrolujte, zda se v log souboru vyskytují řádky typu

```
! Undefined control sequence.
l.457 \fobx
          {%
```

V log souboru jsou zaznamenány také varování, avšak bez uvedení čísla řádku. Časté varování je

```
LaTeX Warning: There were undefined references.
```

Zjistit na základě varování nebo chybového hlášení, kde přesně je chyba, může být složité. Nicméně uvedená informace by vás měla nasměrovat na konkrétní řádek, kde zahájíte pátrání po chybě. Pokud vaše vývojové prostředí ukrývá log soubor před uživatelem, zjistěte si, kde tento soubor najít. A rozhodně log soubor nemažte bez toho, abyste se do něj podívali.

Ne vždy číslo řádku uvedené v chybovém hlášení odpovídá číslu řádku, kde se nachází chyba. Například v matematickém módu (ať už uzavřeném mezi dolary nebo do matematického prostředí) není možné ukončit odstavec. Proto na chybějící dolar TEX přijde až na konci odstavce, což může být o několik řádků později. (Toto omezení je také důvodem, proč není možné vložit prázdné řádky dovnitř matematického prostředí.) Příslušné chybové hlášení při chybějícím dolaru je

```
! Missing $ inserted.
```

Jelikož je tato chyba omezená na jeden odstavec, není problém tuto chybu najít a opravit.

Také chyba uvnitř obrázku, tabulky nebo víceřádkového vzorce se zpravidla projeví až na konci příslušného prostředí a ne okamžitě na řádku s chybou. Ale opět je oblast, kde se chyba vyskytuje, relativně malá a snadná na prohledání.

Další situací, kdy TEX nahlásí chybu daleko za jejím výskytem, je neukončená nebo nesprávně ukončená skupina, tj. část textu zahájená `{`, `\bgroup`, `\begingroup` nebo `\begin{`⟨*prostředí*⟩`}`. V případě nesprávně ukončeného prostředí LATEX nahlásí

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

Toho hlášení se vypíše okamžitě, kdy LaTeX narazí na nesprávné `\end`. Uvedené číslo řádku by mělo být správné. Na druhou stranu neukončenou skupinu TeX nahlásí až na konci dokumentu. Formát hlášení je odlišný a skládá se z několika řádků.

```
(\end occurred inside a group at level a)
### simple group (level b) entered at line m ({)
### semi simple group (level c) entered at line n (\begingroup)
### bottom level
(see the transcript file for additional information)
```

V tomto hlášení číslo a označuje, kolik skupin zůstalo neukončených. Tokeny v závorkách na koncích řádků určují, o jaký typ skupiny se jedná. Přitom token `{` může označovat také skupinu zahájenou tokenem `\bgroup`. Čísla řádků by opět měla správně informovat, kde byla příslušná skupina zahájena.

Další možná chybová hlášení jsou uvedena v dokumentaci různých balíčků. Většinou hlášení obsahují nějaké číslo řádku a toto číslo bývá dosti blízko místu skutečné chyby. Často je možné chybu najít bez většího hledání.

Jakmile už chybu objevíte a opravíte, pak je třeba opravit i původní soubor a pokračovat v jeho překladu. Na vedlejší testovací soubory můžete zapomenout, ty již svou funkci splnily.

Co ale dělat v případě, že se dokument skládá z více souborů? Ve kterém souboru máte hledat uvedené číslo řádku? To si popíšeme v následující sekci.

Důležitým ponaučením z této sekce je: Nikdy nemažte log soubor dříve, než z něj vyčtete všechny užitečné informace.

Někteří dokonce doporučují ukládat si log soubory průběžně pod různými názvy. Díky tomu pak je možné zjistit, co vše se změnilo mezi jednotlivými běhy TeXu.

6. Jeden z mnoha – ale který?

V této sekci budeme předpokládat, že chyba je v některém našem souboru, ne v balíčku.

Když v log souboru vidíte číslo řádku, nejdříve se podívejte na tento řádek v hlavním souboru. Ale když hlavní soubor má pouze 95 řádků a chyba je na řádku 2345, pak je tato rada k ničemu. V tom případě je třeba najít ten správný soubor, jehož se dané číslo řádku týká.

Udělejte si kopii log souboru a procházejte jej pozpátku od příslušného chybového hlášení. Jestliže už některé stránky byly vysázeny do pdf souboru, pak čísla těchto stránek (uvedená v hranatých závorkách, například `[69]`) mohou ukazovat na kapitolu a tu v ideálním případě máte ve vlastním souboru. Když toto nepomůže, promazávejte z kopie log souboru informace, které pro účely hledání nejsou potřebné.

Zprávy o přetečených boxech můžete ignorovat – tyto řádky smažte. Spárovaná dvojice závorek obvykle uzavírá název souboru a nějaký další materiál. Vyhledejte celou uzávorkovanou skupinu jako například

```
(C:/noname/debug/uvod.tex
Úvod
[1] [2]
)
```

a smažte ji. Co nakonec zůstane, je otevírací závorka následovaná názvem souboru – souboru, který byl otevřený, když byla oznámena chyba. Řádek s daným číslem se nachází v tomto souboru.

Ale co když číslo řádku bylo uvedeno až na konci dokumentu – situace s neukončenou skupinou? Tady přichází na řadu vhodně použitý `\end{document}`. Opět pracujte pouze s vedlejšími testovacími soubory a neměňte původní soubory, dokud nenajdete a neodstraníte chybu.

Začněte na konci řídicího souboru a vložte řádek `\end{document}` mezi dva řádky s `\include`. Je vhodné použít metodu bisekce – vložte `\end{document}` tak, aby zhruba polovina řádků s `\include` byla nad ním a polovina pod ním. (Více si o tom povíme v sekci 8.) Přeložte dokument LaTeXem. Jestliže stále zůstala neukončená skupina, je problémový soubor v první polovině. Jestliže nezůstala, je ve druhé polovině. Nyní přesuňte řádek `\end{document}` doprostřed té poloviny, ve které byl problém. Takto postupujte, dokud neodhalíte problémový soubor.

Situace je komplikovanější, pokud je neukončených více skupin (tj. $a > 1$), ale princip je stejný.

7. Pročištění souborů

V tuto chvíli již tušíte, ve kterém souboru by měla být chyba. A možná také tušíte, poblíž kterého řádku by ta chyba měla být. A nebo také máte pouze obecnou představu, kam byste se měli podívat. Aby se vám lépe pracovalo, zpracovávejte pouze jeden soubor. Všechny pro vaše účely nepotřebné věci promažte.

V řídicím souboru makrem `\includeonly` specifikujte ten podezřelý soubor, který jediný se má zpracovávat. Zakomentujte nepotřebné části kódu, které jsou mimo `\include`:

- balíčky nepotřebné pro testování,
- `\tableofcontents`
- `\printindex`
- věci související se seznamem literatury.

Pročistěte také podezřelý soubor. Nemusíte se bát, že soubor zničíte, je to pouze kopie, že? Následující věci můžete opatrně vymazat:

- řádky začínající procentem,
- řádky `\begin{comment}` a `\end{comment}` a vše mezi nimi,
- řádky mezi `\iffalse` a `\fi` včetně těchto řádků (tato konstrukce je ekvivalentní s komentářem).

Ujistěte se, že jsou všechny skupiny správně uzavřeny. To znamená zkontrolovat spárování všech `\begin` a `\end` a všech možných uzávorkování. Je užitečné tady použít funkci editoru, která zjistí počet výskytů daných řetězců. V bezchybném dokumentu by mělo platit

- počet `{` = počet `}` (pozor na řetězec `%}`, který se občas ve zdrojovém souboru vyskytuje),
- počet `\begin{` = počet `\end{`,
- počet `\begingroup` = počet `\endgroup`,
- počet `\bgroup` = počet `\egroup`,
- počet `\(` = počet `\)`,
- počet `\[` = počet `\]`,
- počet `$` je sudý,
- počet `$$` je sudý.

Nyní přeložte soubor LaTeXem a přečtěte si log soubor. Mnoho chyb je způsobeno nespárováním, proto pokud budete mít štěstí, podaří se vám takto chybu odhalit.

V dalších sekcích si ukážeme, co dělat v případě, že stále není jasné, kde je chyba.

8. Rozděl a panuj

Co chceme udělat, je izolovat jediný odstavec nebo nejmenší část souboru, které způsobují chybu. (Pracujte s kopií souboru a pro jistotu si vytvořte další kopii.)

Za vhodný odstavec někde v půlce souboru vložte prázdný řádek a pod něj příkaz `\endinput`. Ujistěte se, že jste takto nerozdělili `\begin` a `\end` nebo libovolnou skupinu. Přeložte dokument LaTeXem. Pokud se chyba nevyskytne, je problém ve druhé (nezpracované) polovině souboru. Smažte první (funkční) polovinu souboru a posunujte příkaz `\endinput`, dokud nenarazíte na problémový odstavec. Jestliže je řešení problému jednoduché, problém opravte a otestujte. Poté stejnou opravu proveďte a otestujte na celé testovací kopii souboru. A až si budete jisti, že jste problém opravili, proveďte a otestujte opravu na původním souboru.

Ale co když řešení problému není jednoduché? Co když část souboru, která zůstala, je pořád příliš obrovská na to, aby se rychle podařilo najít chybu? (Například se může jednat o dlouhý důkaz, kde jsou jednotlivé kroky vloženy jako

položky výčtového prostředí.) V tom případě vložte tuto část do nového souboru a upravujte tento nový soubor. (Mnohokrát se autorce tohoto článku stalo, že upravovala testovací kopii souboru, ale řídicí soubor stále načítal původní soubor. To dokáže naštvat. Dejte si pozor, ať načítáte ten soubor, který upravujete.)

Zmenšete velikost souboru zakomentováním položek, které vypadají neškodně. Zatím ale nic nemažte – co se zdá být neškodné, může ve skutečnosti být součástí problému. Tento proces opakujte tak dlouho, dokud vám nezůstane část souboru, jejímž vymazáním se eliminuje i (ta dosud nenalezená) chyba. Tato část souboru se nazývá „minimální (ne)funkční příklad“.

Prozkoumejte, co v souboru zůstalo, a využijte záchytné body z log souboru.

Až chybu najdete a je vám jasné, jak ji opravit, pak tu opravu proveďte a otestujte ve vaši kopii souboru a až pak opravu proveďte v původním souboru. Jestliže na žádnou další chybu nenarazíte, jste na dobré cestě.

Jestliže narazíte na nějaký další problém, jste opět na startu. Ale teď už víte, jak postupovat.

9. Když je potřeba další pomoc

Na internetu jsou dostupné různé užitečné stránky. Je možné, že nejste první, kdo narazil na podobný problém.

Je dobré se podívat do archivu `tex.sx`. Když nenajdete nic, co by se podobalo vašemu problému, vytvořte nový dotaz. (K vytvoření dotazu je nutné se zaregistrovat.) Aby se povedlo problém vyřešit, vložte celý minimální příklad. Vymažte z něj všechny komentáře a pokud je to vhodné (a možné), „anonymizujte“ příklad vložením smyšleného textu namísto reálného textu. Udělejte příklad co nejmenší, přitom aby pořád vykazoval problém. K příkladu přiložte příslušné řádky log souboru a také komentář, o co se v příkladu pokoušíte. Účastníci fóra `tex.sx` jsou učenliví a přátelští a mají rádi záhadné problémy. K tomu ale potřebují mít dostatek informací, aby mohli experimentovat. A když jim popíšete minimální příklad, který si mohou zkopírovat a vložit k sobě, pak jim experimenty půjdou rychleji.

10. Chyby před `\begin{document}`

Stále jsme neřešili situaci, kdy se chyba vyskytne už před `\begin{document}`. V tom případě pomůžou následující kroky.

- Udělejte si kopii log souboru a najděte problémový soubor, podobně jako v sekci 6.
- Pokud to není balíček, dále hledejte soubor, který se balíčkem načítá.
- Máte zkušenosti s vnitřními makry a registry LaTeXu?

Chybové hlášení na terminálu:

```
Overfull \hbox (23.1113pt too wide) in paragraph at lines 3288--3301
\OML/cmm/m/it/10.95 A$\T1/ptm/m/n/10.95 , as in
```

Příslušná část log souboru:

```
Overfull \hbox (23.1113pt too wide) in paragraph at lines 3288--3301
\OML/cmm/m/it/10.95 A$\T1/ptm/m/n/10.95 , as in Ÿ[], is a degree-
```

Příslušná část zdrojového kódu:

```
..., as in \S\ref{SS:changing}, is a degree-1 ...
```

Obrázek 1: Záhadné chybové hlášení

- Pokud zkušenosti nemáte, je ideální čas obrátit se na odborníka. Běžte na `tex.sx`. Jestliže podobný problém zatím nikdo neřešil, vytvořte nový dotaz. Buďte konkrétní, vložte svůj problémový zdrojový kód, který máte před `\begin{document}` a příslušnou část log souboru.
- Pokud zkušenosti máte, pokuste se problém vyřešit sami. Podívejte se na `tex.sx`. Jestliže podobný problém zatím nikdo neřešil, obraťte se na autora balíčku.

Tímto končí rozbor problémů, které se mohou vyskytnout ve vašich souborech. V další části autorka popisuje skutečný problém, na který narazila a dlouho se s ním trápila, než se ukázalo, že to ve skutečnosti nebyl problém LaTeXu, ale pouze se až v LaTeXu projevil.

11. Opravdová záhada

Jednou za čas nás ani důkladné krokování nepřivede k vyřešení problémů. Zdůrazním dvě související věci.

- Žiji a pracuji v USA a podle toho mám počítač nastavený, tj. například používám kódování ASCII.
- Spouštím LaTeX z terminálu a nepoužívám `\batchmode` ani `\nonstopmode`.

Při překladu jednoho dokumentu LaTeX neustále zamrzal a s ním zamrzal i terminál. Bylo nutné spustit nový terminál a z něj LaTeX ukončit. Poslední zobrazená informace byla o přetečeném boxu, vizte obrázek 1. Tato informace stačila k přesné lokalizaci problému. Nicméně zdrojový soubor se v daném místě zdál zcela v pořádku. Naštěstí se vytvořil log soubor, i když ne kompletní.

Po provedení výše uvedených kroků se povedlo redukovat dokument na jediný krátký odstavec. Po odstranění textu ze začátku tohoto odstavce se chyba přestala

vyskytovat. Zdálo se, že chyba souvisí s oním přetečeným boxem. Na tomto místě jsem se poradila s kolegy, kteří mají lepší znalosti operačního systému.

Po důkladném prozkoumámí log souboru je vidět záhadný znak Ÿ. (Znak je na pozici `"78` ve fontu `cmsy` a má unicodový kód U+0178.) Protože pracuji s anglickými dokumenty a málokdy se setkám s akcenty, nevídávám tak často znaky mimo ASCII a určitě ne v log souboru příslušném čistě anglickému textu. Ukázalo se, že problém byl způsoben systémovým prostředím, které nebylo nastaveno na vstup v kódování UTF8 a jako důsledek zamrzalo. Řešením se ukázalo vložit do souboru `.i18n` v domovském adresáři řádek

```
LANG="en_US.utf8"
```

Někdy to, co vypadá jako LaTeXová chyba, ve skutečnosti může být chyba úplně jinde.

12. Drobnosti (přidáno po konferenci)

Jsou problémy, které se vyskytují relativně často a dají se dobře identifikovat, avšak najít problémové místo ve zdrojovém souboru bývá obtížné.

- Varování o chybějícím znaku

  ```
  There is no ; in font nullfont!
  ```

  je téměř jistě důsledkem syntaktické chyby (chybějícího středníku) v prostředí `tikzpicture`. Další chybějící znaky interpunkce odkazující se na `nullfont` také mohou souviset s chybou uvnitř `tikzpicture`.
- Podobná varování odkazující se na znaky v jiném fontu se musejí prozkoumat důkladně. V log souboru není uvedeno číslo řádku, na němž k problému došlo. Nicméně je možné se podívat, jaké je číslo poslední vysázené strany, a podle toho projít následující vysázenou stranu, v čem se text na ní liší od zdrojového textu.

13. Poděkování

Autorka děkuje sdružení GUST za uspořádání konference TUG'17 v Bachotku společně s jeho každoroční konferencí. Dále děkuje účastníkům konference za jejich dotazy a užitečné připomínky.

Odkazy

1. BEETON, Barbara. Debugging LaTeX files: Illegitimi non carborundum. *TUGboat* [online]. 2017, roč. 38, č. 2, s. 159–164 [cit. 2021-12-06]. Dostupné z: `http://www.vim.tug.org/TUGboat/tb38-2/tb119beet.pdf`.

2. *Stack Exchange* [online]. Stack Exchange [cit. 2021-12-06]. Dostupné z: `https://tex.stackexchange.com/`.
3. KNUTH, Donald E. *The TEXbook*. Reading, MA: Addison-Wesley, 1986. Computers & Typesetting.
4. EIJKHOUT, Victor. *TEX by Topic: A TEXnician's Reference*. Wokingham, England: Addison-Wesley, 1992.
5. OLŠÁK, Petr. *TEXbook naruby*. Konvoj, 2001.

Summary: Debugging LaTeX Files

Every LaTeX user has, at least once in her career, been faced with a thorny problem when compilation shuts down for some obscure reason. How to deal with simple problems is reasonably well known, but there are situations when the time-honored methods fall short.

This article presents strategies and tactics for dealing with the many types of problems that have arisen during long experience as a member of the AMS technical support staff, handling questions from authors and the editorial staff. Both common and uncommon glitches are visited, with a bias toward avoiding problems in one's own work – something for everyone.

Keywords: LaTeX, debugging, errors, log file

Barbara Beeton, American Mathematical Society, Providence, RI, USA
bnb@ams.org

Markdown 2.10.0: LaTeXová témata a snippety

Vít Novotný

Na oslavu svých pátých narozenin dostal balíček Markdown několik zábavných superschopností [1]. Ačkoliv je většina z nich nezávislá na konkrétním formátu, LaTeX obdržel dvě exkluzivní novinky: LaTeXová témata a LaTeXové snippety. V článku obě novinky představuji a ukazuji, jak je mohou TeXoví mágové využít.

Klíčová slova: Markdown, LaTeX, LaTeXová témata, LaTeXové snippety

TeX je assembler světa digitální sazby. Nenabízí vysokoúrovňové abstrakce a umožňuje přípravu stručných, ale úzce zaměřených, nebo obecných, ale upovídaných značkovacích jazyků. Nad TeXem proto vznikají vysokoúrovňové programovací jazyky jako expl3 [2] a obecné a stručné značkovací jazyky jako markdown [3].

Cílem balíčku Markdown je přinést uživatelům TeXu oheň, aby mohli zažehnout každý prvek svých markdownových dokumentů. Markdown nemá ambice poskytovat zevrubná výchozí nastavení, která by vždy dávala uspokojivý výstup. Veškerá pozornost je soustředěna na to, aby mohli TeXoví mágové pohodlně určovat styl svých dokumentů. Ačkoliv tento přístup naplňuje UNIXovou filosofii dělání jedné věci pořádně, balíčku Markdown by prospělo, kdyby si nechal od TeXových mágů v lecčems poradit. Hleďte, už sedí v lavici s brkem a kalamářem!

Je paradox, že jedním z největších úspěchů projektu LaTeX je jeho prvotní nedostatek funkcí. Na rozdíl od katedrály ConTeXtu, kde je balíčků poskrovnu a většina vývoje probíhá centrálně, neduživý LaTeX dal vzniknout pozoruhodnému tržišti, které se snaží uspokojit nekonečno názorů, potřeb a chutí svých návštěvníků. Aby mohli TeXoví mágové u svých stánků s lektvary rozšiřovat balíček Markdown, přinesla verze 2.10.0 *LaTeXová témata* a *snippety* (česky *útržky kódu*).

V tomto článku představuji tři LaTeXová témata, která jsou součástí Markdownu. Následně ukazuji tvorbu vlastních témat a představuji LaTeXové snippety.

Vestavěná témata

LaTeXová témata jsou mágem zadefinované stavební bloky, které určují, co prvky jazyku markdown *dělají*. Témata lze sdílet, uplatňovat naráz jako jednotku abstrakce a vzájemně kombinovat pro dosažení vysokoúrovňových cílů.

Jelikož je LaTeX turingovsky úplný jazyk, můžeme pod to, co prvky jazyku markdown *dělají*, zahrnout nejen změny vzhledu, ale i libovolné výpočty a řízení toku programu. Tabulku jistě můžeme různými způsoby vysázet, ale stejně tak ji můžeme uložit jako matici, vypočítat její determinant, inverzi a vlastní čísla a podle výsledku automaticky vygenerovat obsah dokumentu nebo jeho části.

Balíček Markdown zahrnuje tři vestavěná témata, která zde uvádím od nejjednoduššího po nejsložitější: `witiko/tilde`, `witiko/dot` a `witiko/graphicx/http`.

 doi: 10.5300/2021-1-4/76

Vestavěné téma `witiko/tilde`

Téma `witiko/tilde` předefinovává vlnku (~) tak, aby sázela nedělitelnou mezeru:

```
\documentclass{article}
\usepackage[theme = witiko/tilde]{markdown}
\begin{document}
\begin{markdown}
Bartel~Leendert van~der~Waerden
\end{markdown}
\end{document}
```

Výše uvedený kód vysází text „Bartel Leendert van der Waerden“.

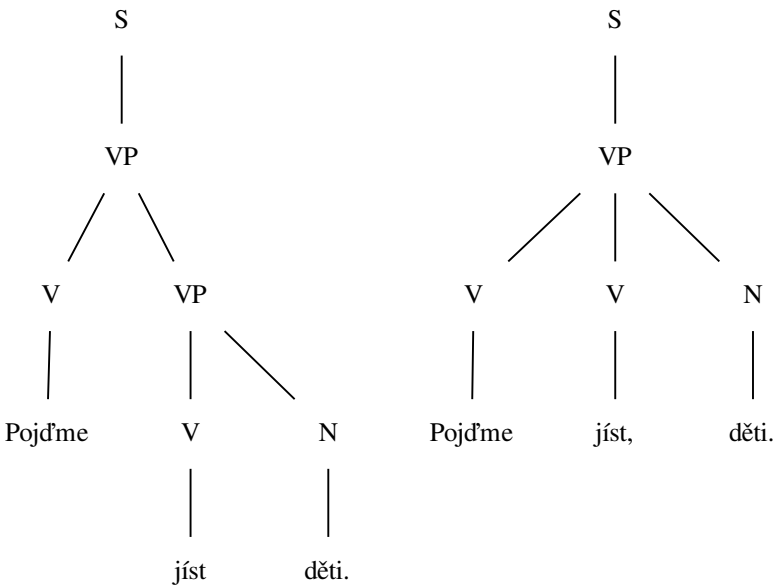
Vestavěné téma `witiko/dot`

Téma `witiko/dot` vykreslí bloky kódu se značkou `dot` programem GraphViz:

```
\documentclass{article}
\usepackage[texComments, theme = witiko/dot]{markdown}
\begin{document}
\begin{markdown}
``` dot Derivační stromy vět ▷ „Pojďme jíst děti." %
                             a ◁ „Pojďme jíst, děti."
digraph tree { graph [margin = 0]; node [shape = none,
               fontname = "times"]; edge [arrowhead = none]
  {rank=same; S1 [label = S]; S2 [label = S] }
  {rank=same; VP1 [label = VP]; VP2 [label = VP] }
  {rank=same; V1 [label = V]; VP3 [label = VP]
              V2 [label = V]; V3 [label = V]; N1 [label = N] }
  {rank=same; Pojďme; V4 [label = V]; N2 [label = N]
              Pojďme2 [label = Pojďme]; jíst1 [label = "jíst,"]
              děti1 [label = "děti."] }
  {rank=same; jíst; děti2 [label = "děti."] }
  S1 -> VP1; VP1 -> V1; VP1 -> VP3; V1 -> Pojďme; VP3 -> V4
  VP3 -> N2; V4 -> jíst; N2 -> děti2; S2 -> VP2; VP2 -> V2
  VP2 -> V3; VP2 -> N1; V2 -> Pojďme2; V3 -> jíst1; N1 -> děti1 }
\end{markdown}
\end{document}
```

Výše uvedený kód vysází Obrázek 1. Šipky v popisku (▷ a ◁) jsou Unicode znaky U+25E4 a U+25E5 předefinované pomocí LaTeXového balíčku newunicodechar.

Velikost obrázku a další atributy můžeme řídit příkazem `\setkeys{Gin}{...}` balíčku `graphicx`. Umístění obrázku můžeme změnit předefinováním příkazu `\fps@figure` z LaTeXového jádra.
```



Obrázek 1: Derivační stromy vět ▷ „Pojďme jíst děti.“ a ◁ „Pojďme jíst, děti.“

```
\documentclass{book}
\usepackage{markdown}
\markdownSetup{pipeTables,tableCaptions}
\begin{document}
\begin{markdown}
Introduction
============
## Section
### Subsection
Hello *Markdown*!

| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |

: Table
\end{markdown}
\end{document}
```

Chapter 1

Introduction

1.1 Section

1.1.1 Subsection

Hello *Markdown*!

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

Table 1.1: Table

Obrázek 2: Banner balíčku Markdown, který demonstruje jeho základní funkce

Vestavěné téma `witiko/graphicx/http`
Téma `witiko/graphicx/http` stahuje online obrázky programy GNU Wget a cURL a následně je zobrazí:

```
\documentclass{article}
\usepackage{markdown}
\markdownSetup{contentBlocks, theme = witiko/graphicx/http}
\begin{document}
\begin{markdown}
https://github.com/witiko/markdown/raw/main/banner.png
(Banner balíčku Markdown, který demonstruje jeho základní funkce)
\end{markdown}
\end{document}
```

Výše uvedený kód vysází Obrázek 2.

Tak jako dříve lze velikost a umístění obrázku řídit pomocí LaTeXových příkazů `\setkeys{Gin}{...}` a `\fps@figure`.

Tvorba vlastního tématu

Předtím, než začnete vytvářet své vlastní LaTeXové téma, měli byste se rozhodnout, jak se téma bude jmenovat. Jméno by mělo být ve tvaru *autor*`/`*cílový balíček*`/`*vlastní pojmenování*, kde *autor* udává osobu, která je za téma zodpovědná, *cílový balíček* značí softwarový balíček, ke kterému se téma vztahuje, a *vlastní pojmenování* vám umožňuje popsat konkrétní zaměření tématu pomocí dalších jmenných segmentů oddělených lomítky. *Cílový balíček* a *vlastní pojmenování* jsou volitelné, ale alespoň jeden z nich musí být přítomen.

Předpokládejme například, že mágyně Laura LaTeXová se rozhodne vytvořit jednoduché téma pro LaTeXovou třídu Beamer. Beamer sází prezentační slajdy a Lauřino téma nastaví markdownové nadpisy první a druhé úrovně tak, aby vysázely záhlaví jednotlivých prezentačních slajdů. Po chvilkovém zamyšlení se Laura rozhodne své téma pojmenovat `laura/beamer/nadpisy`.

Následně Laura *přechroupá* název tématu tak, že nahradí lomítka podtržítky a přidá předponu `markdowntheme` a koncovku `.sty`, čímž obdrží následující název souboru: `markdownthemelaura_beamer_nadpisy.sty`. Pod tímto názvem Laura vytvoří nový textový soubor s následujícím obsahem:

```
\ProvidesPackage{markdownthemelaura_beamer_nadpisy}[2021/06/04]
\markdownSetup{
  rendererPrototypes = {
    headingOne = {\frametitle{#1}},
    headingTwo = {\framesubtitle{#1}}
  }
}
```

Nakonec Laura použije své nové téma ve svých prezentačních slajdech spolu s tématem `witiko/dot`, které jí poslouží pro přípravu nutričního vyhodnocení:

```
\documentclass[aspectratio = 169]{beamer}
\usepackage[czech]{babel}
\usepackage[theme = witiko/dot,
            theme = laura/beamer/nadpisy]{markdown}
\setkeys{Gin}{width = \columnwidth, keepaspectratio}
\title{Vyhodnocení jídelníčku Zlého vlka}
\author{Laura \LaTeX ová}
\date{4. července 2021}
\begin{document}
\maketitle
\begin{frame}[fragile]
\begin{markdown}
# Jedl jsem -- snědl jsem
## Nutriční vyhodnocení
``` dot
digraph tree {
  Vlk -> Babička; Vlk -> Karkulka
  Vlk [label = "Zlý vlk"]; Karkulka [label = "Červená karkulka"] }
\end{markdown}
\end{frame}
\end{document}
```

Výše uvedený kód vysází prezentační slajdy na Obrázku 3. Laura následně téma hrdě vystaví u svého stánku na tržišti CTANu, kde se mu mohou obdivovat kolemjdoucí mágové a běžní uživatelé.[1]

Obrázek 3: Prezentační slajdy připravené tématem `laura/beamer/nadpisy`. Výživová hodnota babičky s Karkulkou (až 250 kcal [4]) nepřevyšuje ani bábovku, kterou Karkulka babičce nese. Chudák vlk musí být podvyživený.

[1] Tématy se můžete chlubit na adrese `https://github.com/witiko/markdown/discussions`.
```

LaTeXové snippety

Laura se rozhodla vytvořit téma `laura/seznamy/rimske`, které údajně způsobí, že číslované seznamy budou uvozené římskými číslicemi. To jsme tedy zvědavi!

```
\ProvidesPackage{markdownthemelaura_seznamy_rimske}[2021/06/04]
\markdownSetup{
  rendererPrototypes = {
    olItemWithNumber = {\item[\romannumeral#1\relax.]}
  }
}
```

Vše jde podle plánu, dokud se Laura nepokusí uplatnit své téma jen na jeden ze dvou seznamů a druhý seznam ponechat s arabskými číslicemi:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}
1. wahid
2. aithnayn
\end{markdown}   % Tohle nebude fungovat!
\begin{markdown*}{theme = laura/seznamy/rimske}
3. tres
4. quattuor
\end{markdown*}
\end{document}
```

Výše uvedený kód selže s chybovou hláškou „Can be used only in preamble“. Jádro vlka je v tom, že LaTeXová témata jsou plnohodnotné LaTeXové balíčky a jako takové sice mohou dělat libovolné úkony, ale nelze je uplatnit pouze na malé úseky kódu, protože úkony mohou být nevratné.

LaTeXové snippety umožňují oddělit příčinu od důsledku. Nejprve v záhlaví dokumentu načteme LaTeXové téma (to je příčina). V tématu definujeme snippety, což jsou pojmenovaná nastavení balíčku Markdown. Snippety následně můžeme uplatnit na malé úseky kódu (to je důsledek).

Laura nejprve zkrátí název svého tématu na `laura/seznamy` tak, že změní segment `rimske` na snippet:

```
\ProvidesPackage{markdownthemelaura_seznamy}[2021/06/04]
\markdownSetupSnippet{rimske}{
  rendererPrototypes = {
    olItemWithNumber = {\item[\romannumeral#1\relax.]}
  }
}
```

Zároveň ve svém dokumentu oddělí načtení tématu od uplatnění snippetu:

```
1 \documentclass{article}
2 \usepackage[theme = laura/seznamy]{markdown}
3 \begin{document}
4 \begin{markdown}
5 1. wahid
6 2. aithnayn
7 \end{markdown}
8 \begin{markdown*}{snippet = laura/seznamy/rimske}
9 3. tres
10 4. quattuor
11 \end{markdown*}
12 \end{document}
```

Výše uvedený kód vysází následující číslovaný seznam, jak jsme si všichni přáli:

1. wahid 2. aithnayn iii. tres iv. quattuor

Všimněte si, že snippet musíme volat celým jménem `laura/seznamy/rimske`. Toto opatření snižuje riziko kolize mezi snippety stejného názvu z různých témat. Snippety je možné definovat i mimo témata. V takovém případě bude snippet dostupný pod jménem `rimske`, což můžeme využít mimo jiné pro zavádění zkratek:

```
\markdownSetupSnippet{rimske}{snippet = laura/seznamy/rimske}
```

Následně můžeme na řádku 8 psát `\begin{markdown*}{snippet = rimske}`.

Odkazy

1. NOVOTNÝ, Vít. Markdown 2.10.0: LaTeX Themes & Snippets, Two Flavors of Comments, and LuaMetaTeX. *TUGboat*. 2021, roč. 42, č. 2, s. 186–193.
2. WRIGHT, Joseph. LaTeX3 programming: External perspectives. *TUGboat*. 2009, roč. 30, č. 1, s. 186–193.
3. NOVOTNÝ, Vít. Sazba textu označkovaného v jazyce Markdown uvnitř TeXových dokumentů. *Zpravodaj ČSTUGu*. 2016, roč. 26, č. 1–4, s. 78–93.
4. COLE, J. Assessing the calorific significance of episodes of human cannibalism in the Palaeolithic. *Sci Rep*. 2017.

Summary: Markdown 2.10.0: LaTeX Themes & Snippets

Celebrating its fifth birthday, the Markdown package has received a number of fresh fun features [1]. Although most are format-agnostic, LaTeX has received two exclusives: LaTeX themes and LaTeX setup snippets. In this article, I introduce both exclusive features and show how TeX mages can benefit from them.

Keywords: Markdown, LaTeX, LaTeX themes, LaTeX snippets

Vít Novotný, witiko@mail.muni.cz

Priama sadzba dokumentov rôznych formátov v TEXu pomocou nástroja Pandoc

Dominik Rehák

TEXový balík Markdown umožňuje priamo sádzať dokumenty v jazyku Markdown a štylizovať jednotlivé prvky jazyka Markdown. Neponúka však podporu pre iné formáty dokumentov. Naopak program Pandoc umožňuje konverziu mezi desiatkami formátov dokumentov vrátane TEXu a Markdownu, neumožňuje ale štylizovanie jednotlivých prvkov dokumentov.

Článok pojednáva o tom, ako je možné pridaním podpory medziformátu Pandocu do Markdownu umožniť priamu sazdbu rôznych textových formátov v TEXových dokumentoch. Zameriavam sa na množiny prvkov Markdownu a Pandocu a rozdiely medzi nimi, ktoré bude musieť nadchádzajúca implementácia prekonať. Nakoniec uvádzam plánované používateľské rozhranie pre TEX.

Kľúčové slová: Markdown, Pandoc, Lua, LATEX

Úvod

Makrobalík *Markdown* [1] umožňuje doplniť dokumenty sádzané v TEXu o úseky textu písané v odľahčenom značkovacom jazyku Markdown, ktoré počas sadzby transformuje na natívne TEXové makrá. Vzhľad dokumentov je možné ovplyvniť predefinovaním *medzimakier* sady `\markdownRenderer`..., ktoré Markdown interne využíva na vysádzanie jednotlivých prvkov textu. Makrobalík Markdown ale neponúka podporu pre iné značkovacie jazyky a formáty dokumentov.

Program *Pandoc* [2] umožňuje konverziu medzi desiatkami značkovacích jazykov a dokumentových formátov, ako sú dialekty Markdownu, formáty TEXu (LATEX a ConTEXt), HTML, docx, nebo roff. Výstup konverzie je možné ovplyvniť pomocou *filtrov*, čo sú používateľské programy, ktoré upravujú dokument počas konverzie. Pri konverzii do TEXu má ale autor len obmedzené možnosti, ako ovplyvniť vzhľad výsledných dokumentov.

Integráciou Pandocu do balíku Markdown by bolo možné využiť silné stránky oboch. Konkrétne by takáto integrácia umožnila vysádzať v TEXu dokumenty všetkých formátov, ktoré Pandoc podporuje ako vstupné, a upravovať vzhľad dokumentov jednoduchou úpravou medzimakier balíka Markdown. Medzimakrá balíka Markdown a prvky medziformátu programu Pandoc sa ale líšia a nie je možné ich priamočiaro na seba napojiť.

V tomto článku najprv zhŕňam aktuálne možnosti priamej sadzby rôznych formátov dokumentov v TEXu. Následne popisujem medzimakrá balíka Markdown a medziformát programu Pandoc. Zameriavam sa na rozdiely medzi nimi

DOI: 10.5300/2021-1-4/83

a navrhujem rozšírenie medzimakier Markdownu tak, aby lepšie korešpondovali s medziformátom Pandocu. Na záver ukazujem, ako bude používateľ TEXu môcť využiť Pandoc pre sadzbu rôznych formátov dokumentov.

Aktuálny stav vkladania dokumentov

Na vkladanie textových dokumentov iných formátov do TEXu momentálne existuje pár ďalších riešení, ktoré sa ale po bližšom preskúmaní ukážu ako nedostatočné.

Ak pre formát existuje možnosť exportu do PDF, je možné balíkom *pdfpages* [3] vložiť stránky vyexportovaného dokumentu priamo do dokumentu v TEXu. pdfpages ale neumožňuje žiadnu úpravu takto vložených stránok, čo nám znemožňuje akúkoľvek štylizáciu na strane TEXu. Taktiež tento prístup neumožňuje ani vloženie krátkych dokumentov do stredu súvislého textu, keďže pdfpages vie vkladať len celé stránky vloženého dokumentu pomedzi stránky TEXového výstupu.

Existujú tiež rôzne riešenia špecializujúce sa na konkrétne vstupné formáty. Príkladom je *GrindEQ*, balík programov, ktorý okrem iného poskytuje konverziu dokumentov programu Microsoft Word priamo do TEXu.[1] Nemienim spochybňovať kvalitu takejto konverzie. GrindEQ je ale proprietárny softvér, vyžaduje zakúpenie licencie a beží vo vnútri programu Microsoft Word, ktorý je navyše viazaný na systém Microsoft Windows. Akákoľvek integrácia programov balíku GrindEQ priamo do TEXu by teda bola extrémne náročná až nemožná.

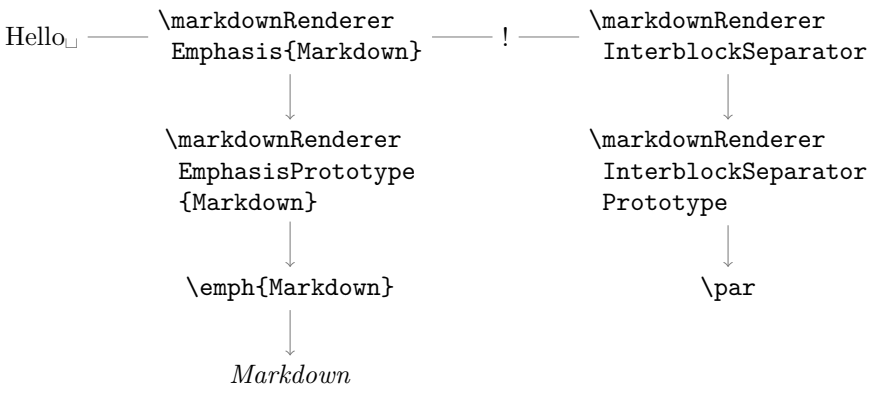
Popis architektúry Markdownu

Parser Markdownu v LATEXovskom balíku Markdown pochádza z knižnice *lunamark* [4], za ktorou zhodou okolností tiež stojí John MacFarlane, autor Pandocu. Knižnica je zameraná na rýchlu konverziu Markdownu do iných bežne využívaných značkovacích formátov ako HTML alebo Docbook, ale i dvoch formátov TEXu: LATEXu a ConTEXtu. Keďže je lunamark napísaný v jazyku Lua, ktorý je možné priamo spúšťať v sádzacom systéme LuaTEX, javil sa parser lunamarku vhodný na jeho začlenenie do Markdownu.

Rýchlosť konverzie bola jednou z priorít pri vývoji lunamarku, a preto konverzia neprebieha v dvoch fázach (do medziformátu a z medziformátu) ako v prípade Pandocu. Namiesto toho tu dochádza k priamej komunikácii medzi dvomi modulmi, vstupným (reader) a výstupným (writer). Výstupný modul implementuje funkcie, ktoré definujú reprezentáciu rôznych prvkov textu vo výstupnom formáte a vstupný modul počas toho, ako parsuje vstupný dokument, zároveň aj zostavuje výstup s pomocou funkcií výstupného modulu, ktoré priamo volá.

Tento princíp bol zachovaný aj v Markdowne – s tým rozdielom, že existujúce výstupné moduly pre podporované formáty boli nahradené jedným, ktorý pre

[1] `https://www.grindeq.com/index.php?p=word2latex`



Obr. 1: Reprezentácia medzimakier a výstupných makier balíku Markdown pre vstup „`Hello *Markdown*!`“

jednotlivé prvky textu generuje makrá `\markdownRenderer`.... Tieto makrá je možné predefinovať a tak upraviť štýl výstupu. Predvolenou hodnotou týchto makier sú makrá `\markdownRenderer`...`Prototype`, ktoré expandujú na natívnu reprezentáciu príslušných prvkov systému LaTeX (viď obr. 1 a 2).

Popis architektúry Pandocu

Konverzia v Pandocu prebieha v dvoch fázach: konverziou vstupného formátu na natívnu reprezentáciu dokumentu a následnou konverziou tejto reprezentácie na výstupný formát. Dokumentácia Pandocu túto natívnu reprezentáciu nazýva *abstract syntax tree*, skrátene AST a vskutku sa jedná o strom prvkov Pandocu, ktorý sa uprosted konverzie celý nachádza v operačnej pamäti (viď obr. 3).

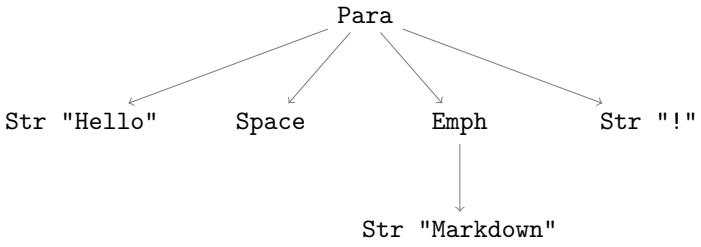
Pri zadaní výstupného formátu `native` vráti Pandoc na výstup celý tento strom vo svojej internej reprezentácii jazyka *Haskell*, v ktorom je Pandoc napísaný. Alternatívne je možné zadaním výstupného formátu `json` získať ekvivalentnú reprezentáciu tohoto stromu vo formáte JSON. Keďže existujú rôzne knižnice v jazyku Lua schopné parsovať formát JSON,[2] je pre naše rozšírenie Markdownu táto reprezentácia vhodnejšia.

Je nutné podoktnúť, že Pandoc delí svoje prvky na blokové (block elements) a neblokové (inline elements). To ovplyvňuje, ktoré prvky sa môžu nachádzať v potomkoch jednotlivých prvkov a často korešponduje s výslednou podobou prvku, napr. implikuje medzery medzi odstavcami, ktoré sú blokovými prvkami. Pri spracovaní AST sa ale týmto rozdelením netreba veľmi zaoberať.

[2]Knižnica, ktorú naše rozšírenie využíva, je dostupná na `https://github.com/rxi/json.lua`.

1. Tickbox Renderers
   (a) `TickedBox`
   (b) `HalfTickedBox`
   (c) `UntickedBox`
2. `InterblockSeparator`
3. `LineBreak`
4. `Ellipsis`
5. `Nbsp`
6. Special Character Renderers
   (a) `Ampersand`
   (b) `Backslash`
   (c) `Circumflex`
   (d) `DollarSign`
   (e) `Hash`
   (f) `LeftBrace`
   (g) `PercentSign`
   (h) `Pipe`
   (i) `RightBrace`
   (j) `Tilde`
   (k) `Underscore`
7. `CodeSpan`
8. `Link`
9. `Image`
10. `ContentBlock`
11. Bullet List
    (a) `UlBegin`
    (b) `UlBeginTight`
    (c) `UlItem`
    (d) `UlItemEnd`
    (e) `UlEnd`
    (f) `UlEndTight`
12. Ordered List
    (a) `OlBegin`
    (b) `OlBeginTight`
    (c) `OlItem`
    (d) `OlItemEnd`
    (e) `OlItemWithNumber`
    (f) `OlEnd`
    (g) `OlEndTight`
13. Definition List
    (a) `DlBegin`
    (b) `DlBeginTight`
    (c) `DlItem`
    (d) `DlItemEnd`
    (e) `DlDefinitionBegin`
    (f) `DlDefinitionEnd`
    (g) `DlEnd`
    (h) `DlEndTight`
14. Emphasis
    (a) `Emphasis`
    (b) `StrongEmphasis`
15. Block Quote
    (a) `BlockQuoteBegin`
    (b) `BlockQuoteEnd`
16. Code Block
    (a) `InputVerbatim`
    (b) `InputFencedCode`
17. YAML Metadata
    (a) `JekyllDataBegin`
    (b) `JekyllDataEnd`
    (c) `JekyllDataMappingBegin`
    (d) `JekyllDataMappingEnd`
    (e) `JekyllDataSequenceBegin`
    (f) `JekyllDataSequenceEnd`
    (g) `JekyllDataBoolean`
    (h) `JekyllDataNumber`
    (i) `JekyllDataString`
    (j) `JekyllDataEmpty`
18. Heading
    (a) `HeadingOne`
    (b) `HeadingTwo`
    (c) `HeadingThree`
    (d) `HeadingFour`
    (e) `HeadingFive`
    (f) `HeadingSix`
19. `HorizontalRule`
20. `Footnote`
21. `Cite`
22. `TextCite`
23. `Table`
24. `InlineHtmlComment`

Obr. 2: Úplný zoznam výstupných makier balíku Markdown k verzii 2.11.0



Obr. 3: Pandoc AST pre vstup Markdownu „`Hello *Markdown*!`“

1. `Plain`
2. `Para`
3. `LineBlock`
4. `CodeBlock`
5. `RawBlock`
6. `BlockQuote`
7. `OrderedList`
8. `BulletList`
9. `DefinitionList`
10. `Header`
11. `HorizontalRule`
12. `Table`
13. `Div`
14. `Null`
15. `Str`
16. `Emph`
17. `Underline`
18. `Strong`
19. `Strikeout`
20. `Superscript`
21. `Subscript`
22. `SmallCaps`
23. `Quoted`
24. `Cite`
25. `Code`
26. `Space`
27. `SoftBreak`
28. `LineBreak`
29. `Math`
30. `RawInline`
31. `Link`
32. `Image`
33. `Note`
34. `Span`

Obr. 4: Úplný zoznam prvkov medziformátu programu Pandoc

Podobnosti a rozdiely medzi architektúrami

Integrácia programu Pandoc do nástroja Markdown vyžaduje jednoznačné mapovanie z medziformátu Pandocu do medzimakier Markdownu. V tomto oddiele sa najprv zameriavam na prvky Pandocu, ktoré majú svoje ekvivalenty v makrách Markdownu. Následne popisujem prvky unikátne pre Pandoc a navrhujem rozšírenia makier Markdownu, ktoré zaistia bezproblémovú konverziu.

Spoločné prvky

Zopár prvkov Pandocu už dokonale korešponduje s existujúcimi makrami Markdownu a ich transformácia je triviálna. Jedná sa napríklad o zvýraznenia `Emph` a `Strong`, ktorým zodpovedajú makrá `Emphasis` a `StrongEmphasis`.[3] Tieto prvky majú jeden parameter, rovnako ako uvedené makrá, takže naše spracovanie by končilo volaním príslušných prototypov makier nad daným parametrom. Do tejto skupiny by sme mohli zaradiť aj prvky `HorizontalRule` a `LineBreak` s rovnomennými makrami, ako i prvky `Note`, `Code` a `CodeBlock` – tento síce s opačným poradím parametrov ako makro – s respektívnymi makrami `Footnote`, `CodeSpan` a `InputFencedCode`.

Iné prvky je nutné spracovať skôr, než ich dokážeme prepísať na existujúce makrá. Príkladom je prvok `Header`, ktorý si okrem svojho obsahu nesie i celé číslo definujúce, o koľkú úroveň hlavičky sa jedná. Tomu zodpovedajú makrá `HeadingOne` až `HeadingSix`. Podobným prípadom sú makrá zoznamov `OrderedList`, `BulletList` a `DefinitionList`, ktoré pozostávajú z polí vyjadrujúcich jednotlivé položky. Ich ekvivalentmi v Markdowne sú makrá vyznačujúce začiatky a konce celých štruktúr, ako aj ich položiek – napr. `UlBegin`, `UlEnd`, `UlItemBegin` a `UlItemEnd` u neusporiadaných zoznamov. (To isté platí i pre prvok `BlockQuote` a dvojicu makier `BlockQuoteBegin` + `BlockQuoteEnd`.)

Niekedy je toto spracovanie prvkov menej komplikované. Príkladom je makro `Cite`, ktoré môže v závislosti na počte poskytnutých citácií prijať rôzny počet parametrov, pričom prvým z nich je počet citácií. Získanie tohoto počtu z príslušného prvku `Cite` je v jazyku Lua triviálne, ostatné parametre potom získame určitým usporiadaním obsahov citácií.

Odkazy a obrázky Pandoc vyjadruje prvkami `Link` a `Image`. Prvým parametrom `Link` je zoznam prvkov tvoriaci text odkazu a druhým je dvojica (URL, titulok) špecifikujúca odkaz samotný.[4] Oproti tomu `Title` má parametre štyri – okrem textu odkazu a titulku prijíma dve URI, z nich jedno v nezmenenej podobe a druhé vhodné priamo na sadzbu, t. j. s ošetrenými špeciálnymi znakmi. Do dokumentu sa reálne dostane len ošetrená URI, využitie nezmenenej je na použí-

[3] Z dôvodu úspory miesta v tomto oddiele uvádzam skrátené názvy makier, t. j. napr. `Emphasis` a `StrongEmphasis` miesto `\markdownRendererEmphasis` a `\markdownRendererStrongEmphasis`.

[4] Titulok zodpovedá hodnote nepovinného HTML atribútu `title`.

teľovi. Podobnú formu má prvok `Image` a jeho rovnomenné makro, kde je naopak predvolene využitá len nezmenená URI, ktorá určuje cestu k obrázku.

Je nutné podotknúť, že u niektorých prvkov by využitie existujúcich makier mohlo byť mierne stratové. Napríklad u prvku `OrderedList` je možný aj výskyt dvoch vyčísliteľných typov, ktoré špecifikujú štýl zobrazenia čísel položiek[5] a oddeľovače týchto čísel[6]. To ale nedokážeme vyjadriť existujúcou skupinou makier `OlBegin`.... Tieto makrá tiež nemôžeme predefinovať, pretože by to mohlo viesť k spätnej nekompatibilite. (Toto je síce nad rámec môjho počiatočného rozšírenia, keďže ide o zriedkavo vyskytujúce sa prvky, ale do budúcnosti nevylučujem implementáciu úplnej reprezentácie cez prídavné makrá.)

Podobná strata by nastala pri spracovaní prvku `Table`, štruktúrne najzložitejšieho prvku Pandocu, príslušným makrom. Prvým parametrom `Table` je titulok tabuľky, za ktorým nasleduje pole dvojíc vyčísliteľných typov `Alignment` a `ColWidth`. Každá dvojica špecifikuje zarovnanie jedného zo stĺpcov, ako aj jeho relatívnu šírku. Zvyšné tri parametre tvoria záhlavie, zoznam tiel a pätu tabuľky. (Každé telo je navyše dvojica zložená z nepovinného záhlavia a hlavnej časti tela.) Všetky tieto časti tabuľky pozostávajú z niekoľkých zoznamov, ktoré vyjadrujú rady jednotlivých polí tabuľky.

Štruktúra makra `Table` je o niečo jednoduchšia a viac plochá. Prvými tromi parametrami sú titulok a dve prirodzené čísla R a S, ktoré určujú počet riadkov a stĺpcov tabuľky. Nasleduje reťazec S znakov `d`, `l`, `c` alebo `r`, ktorý určuje smer zarovnania v jednotlivých stĺpcoch. Na záver makro prijme R parametrov, každý z nich zložený z S parametrov, ktoré opäť vyjadrujú jednotlivé polia tabuľky. Vyjadriť šírky stĺpcov a určiť, ktoré riadky sú záhlavia, makro `Table` už nevie.

Prvky unikátne pre Pandoc

Tým sa dostávame k druhej polovici prvkov, ktorá nemá v makrách Markdownu svoje ekvivalenty. Patria sem napríklad jednoduché prvky `Underline`, `Strikeout`, `Superscript`, `Subscript` a `SmallCaps`, ktorých význam je zrejmý. Implementovať tieto makrá bude triviálne, ak vieme, ako rovnakého efektu docieliť v TEXu.

O niečo zaujímavejší je napríklad prvok `Quoted`, ktorý vyjadruje úsek textu ohraničený úvodzovkami. Okrem poľa potomkov, ktoré ohraničuje, obsahuje `Quoted` ešte hodnotu vyčísliteľného typu `QuoteType`, ktorý nadobúda hodnotu `SingleQuote` alebo `DoubleQuote` a tá vyjadruje, či sa jedná o jednoduché alebo dvojité úvodzovky. Po vzore makier `Heading`... by bolo ideálne toto na strane Markdownu vyjadriť dvomi samostatnými makrami, čím sa zachová konzistencia balíku. Podobne prvok `Math` obsahuje vyčísliteľný typ `MathType` s hodnotou `InlineMath` alebo `DisplayMath`, ktorý určuje, či sa majú priložené literály vysádzať v rámci riadku alebo ako samostatná rovnica.

[5] desiatkové čísla, veľké/malé rímske číslice alebo veľké/malé písmená

[6] bodka, zátvorka alebo dve zátvorky

Zvláštne prípady tvoria prvky `Str` a `Plain`, ktoré vyjadrujú „čistý“ úsek textu bez dodatočného značkovania[7] a tým pádom by bolo možné priamo vysádzať ich obsah. To isté môžeme povedať o prvku `Space`, ktorý reprezentuje jedinú medzeru alebo `Null`, ktorý je prázdny a nevyjadruje nič. Všetky tieto prvky by bolo triviálne nahradiť – v prípade `Null` dokonca zahodiť. Je ale možné, že by niektorí z používateľov Markdownu pre ne dokázali nájsť zmysluplné využitie. Preto pre ne tiež zavedieme príslušné makrá.

Na členenie textu poskytuje Pandoc prvky `Para`, `LineBlock` a `SoftBreak`. `Para` je zhruba ekvivalentný novému riadku s odsadením, `SoftBreak` novému riadku bez odsadenia. `LineBlock` potom vyjadruje skupinu neodsadených riadkov.

`RawBlock` a `RawInline` sú prvky označujúce úseky textu v iných formátoch. Generuje ich napríklad HTML značkovanie využité vo formáte Markdown. V dobe písania tohto článku stále nie je úplne jasné, ako s týmito prvkami nakladať. Ak by šlo o jeden z TEXových formátov, bolo by možné text jednoducho vložiť priamo do dokumentu, čo ale so sebou môže niesť isté bezpečnostné riziko. Ponúka sa tiež možnosť opakovanej konverzie týchto blokov na AST, čo by ale z pochopiteľných dôvodov bolo možné, len ak je Pandoc dostupný pri sadzbe, čo nemusí, ak len podsúvame Markdownu inak získaný AST.

Takisto nie je jasné, ako spracovať prvky `Div` a `Span`. Tie zodpovedajú HTML značkám `<div>` a `<span>` a ich príslušným nepovinným atribútom `label` a `class`, ktoré ale môžu obsahovať prakticky čokoľvek. Najrozumnejšou cestou asi bude predvolene vysádzať obsah týchto prvkov bez akýchkoľvek úprav a prípadné prvky so špeciálnymi atribútmi nechať na používateľa.

Makrá ako `TickedBox` alebo `Ellipsis`, ktoré som v tomto oddiele nespomenul, nemajú v Pandocu svoje ekvivalenty. To je, čo sa tohto rozšírenia týka, v poriadku.

Špecifiká plánovaného rozšírenia

Čo sa integrácie samotnej týka, najjednoduchšou cestou je pravdepodobne rozšíriť lunamark o vstupný modul schopný spracovania AST Pandocu a tento model následne integrovať do Markdownu. V dobe písania článku je práca na takomto module takmer hotová.[8]

Pandoc takisto umožňuje používateľom poskytnúť vlastný výstupný modul v jazyku Lua[9], takže alternatívnym riešením by bolo napísať výstupný modul, ktorý by priamo generoval makrá Markdownu. To by ale vyžadovalo prítomnosť nainštalovaného Pandocu na počítači, kde prebieha sadzba.

Spôsoby, ktorými bude rozšírenie možné využiť v dokumentoch, budú pravdepodobne kopírovať existujúce prostriedky na sadzbu jazyka Markdown. Na priamu

[7] Rozdiel `Str` a `Plain` spočíva v tom, že `Plain` je blokový a `Str` nie je. `Plain` sa vyskytuje okrem iného napr. v poliach tabuliek. Príklad výskytu `Str` je uvedený na obr. 1.

[8] `https://github.com/drehak/lunamark/blob/devel/lunamark/reader/pandoc_json.lua`

[9] `https://pandoc.org/MANUAL.html#custom-readers-and-writers`

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{pandoc}{format=man}
.TH "Pandoc User\[cq]s Guide" "" "July 18, 2021" "pandoc 2.14.1" ""
.hy
.SH NAME
pandoc - general markup converter
.SH SYNOPSIS
.PP
\f[C]pandoc\f[R] [\f[I]options\f[R]] [\f[I]input-file\f[R]]\&...
.SH DESCRIPTION
.PP
Pandoc is a Haskell library for converting from one markup format to
another, and a command-line tool that uses this library.
\end{pandoc}
\end{document}
```

Obr. 5: Ukážka využitia prostredia `pandoc`

NAME

pandoc - general markup converter

SYNOPSIS

`pandoc` [*options*] [*input-file*]...

DESCRIPTION

Pandoc is a Haskell library for converting from one markup format to another, and a command-line tool that uses this library.

Obr. 6: Ukážka výstupu prostredia `pandoc`

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\pandocInput[format=docx]{documentation-of-pandoc.docx}
\end{document}
```

Obr. 7: Ukážka využitia makra `pandocInput`

sadzbu bude podľa vzoru prostredia `markdown` dostupné prostredie `pandoc`, ktoré ale bude navyše vyžadovať informáciu o tom, o aký vstupný formát sa jedná (viď obr. 5 a 6). (Konečná forma sa môže mierne líšiť od ukážky.)

Niektoré vstupné formáty Pandocu, napr. *docx*, ale majú binárnu podobu, ktorá nie je vhodná na vkladanie do TEXovských dokumentov, a preto pre ich sadzbu bude nutné využiť makro `\pandocInput`, analogické k existujúcim makrám `\markdownInput` a `\input` (viď obr. 7).

Referencie

1. NOVOTNÝ, Vít. Sazba textu označkovaného v jazyce Markdown uvnitř TEXových dokumentů. *Zpravodaj Československého sdružení uživatelů TEXu*. 2016, č. 1–4, s. 78–93. Dostupné z DOI: 10.5300/2016-1-4/78.
2. MACFARLANE, John. *Pandoc: a universal document converter* [online]. 2006 [cit. 2021-11-22]. Dostupné z: `https://pandoc.org/`.
3. MATTHIAS, Andreas. *The pdfpages Package* [online]. 2021-03-06 [cit. 2021-11-22]. Dostupné z: `https://ctan.org/pkg/pdfpages`.
4. MACFARLANE, John; HAGEN, Hans; HOSNY, Khaled. *Lunamark* [online]. 2009 [cit. 2021-11-22]. Dostupné z: `http://jgm.github.io/lunamark/`.

Summary: Direct Typesetting of Various Document Formats in TEX Using the Pandoc Utility

The Markdown TEX package allows authors to typeset documents in the Markdown language and maintain control over how the documents will look. However, the package doesn't provide support for document formats other than Markdown. In contrast, the Pandoc tool enables the conversion between dozens of document formats including TEX and Markdown, but only provides rudimentary control over styling.

This article elaborates on the possibility of typesetting various text formats directly in TEX by adding support for Pandoc's intermediate document representation into the Markdown package. I focus mainly on the intermediate representations of Markdown and Pandoc as well as the differences between them, which my upcoming implementation will have to overcome. At the end of my article, I present the planned user interface for TEX.

Keywords: Markdown, Pandoc, Lua, LATEX

Dominik Rehák, drehak@firemail.cc

Mělo by to fungovat XI

Peter Wilson

Článek ukazuje několik způsobů, jak lze vytvořit miniaturní knihu z pouhého jediného listu papíru. Jsou popsána některá řešení tohoto úkolu v LaTeXu.

Klíčová slova: Miniaturní knihy, LaTeX, přehýbání, vyřazování stran

> If our understanding have a film of ignorance over it, or be blear with gazing on other false glisterings, what is that to truth?
>
> *Of reformation in England*
> John Milton

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

> When true simplicity is gained,
> To bow and bend, we will not be ashamed.
> To turn, turn, will be our delight
> 'Til by turning, turning, we come 'round right.
>
> *Simple Gifts*
> a Shaker hymn

V devadesátých letech vytvořil William Adams osmistránkovou brožuru nazvanou *One Typeface, Many Fonts* [2], kterou Vám doporučuji si sehnat, pokud ji ještě nemáte. Kromě svého obsahu a různých řezů písma je zajímavá také tím, že byla vytištěna pouze na jednu stranu jediného papíru, přičemž brožura vznikne vhodným nastřižením a přehnutím tohoto papíru. Sám jsem na tuto brožuru narazil před nedávnem, když jsem při stěhování procházel své věci.

Krátce předtím jsem se setkal se sbírkou miniaturních knih [3], z nichž některé jsou na obrázku 1.[1] Největší má rozměr 7,5×5,5 cm a jedná se o miniaturní knihu o miniaturních knihách. Dvě nejmenší mají rozměr 4×3 cm. Jedná se o inaugurační

Z anglického originálu *Glisterings* [1] přeložil Jan Šustek. Všechny díly *Glisterings* byly také sepsány do knihy, kterou je možné zakoupit prostřednictvím TUG na stránce `https://tug.org/store/glisterings/` (pozn. redakce)

[1] K roku 2000 držela rekord nejmenší kniha *Dvanáct zvířecích znamení – Čínský zvěrokruh*, který ve 100 výtiscích vyšla ve vydavatelství Toppan v Tokiu. Kniha má rozměr pouze 0,95 mm^2.

doi: 10.5300/2021-1-4/93

Obrázek 1: Některé miniaturní knihy. Jednotky na pravítku označují palce, body a pica.

Obrázek 2: Rozvržení stránek v leporelu.

řeč Johna Kennedyho v lednu 1961 a o projev Abrahama Lincolna v Gettysburgu v listopadu 1863. Velikost písma v těchto knihách je jen o málo menší než tady v poznámkách pod čarou.

Po těchto dvou příležitostech jsem se začal zajímat, jestli existují i jiné metody než ta Williamova, jak vytvořit miniaturní brožuru. Dlouhou dobu jsem bezvýsledně zkoušel stříhat a přehýbat počmárané papíry a pak jsem si vzpomněl na knihu Cherryl Moote [4], o níž se píše, že

> *Tato kniha ukazuje vzory ideální pro vytvoření miniaturních uměleckých knih na vaši tiskárně nebo kopírce. . . . Většina vzorů využívá standardní velikosti papírů.*

Je několik možností, jak vytisknout text na jednu stranu jediného papíru a pomocí přehýbání a stříhání z tohoto vytvořit přijatelnou knihu. Nejjednodušší je formát leporela, kde jsou jednotlivé stránky na papíru rozvrženy jako na obrázku 2. V tomto a dalších diagramech jednotlivá čísla označují pořadí stránky a orientaci textu. U číslic, kde by jejich orientace nebyla z diagramu zřetelná, jsou tyto číslice podtrhnuty. Tenké čáry označují, kde se papír přehne, tlusté čáry označují, kde se papír nastříhne. U leporela se papír přehýbá střídavě nahoru a dolů. Tento jednoduchý formát využívá mnoho japonských a čínských knih. Na obrázku 3 je část knihy *Životopisy dvanácti čínských učenců*, která je psána čínsky a anglicky.

Obrázek 3: Část čínské knihy ve formátu leporela. Celá kniha má rozměry 20×425 cm.

3	2
4	1

1	8	7	6
2	3	4	5

Obrázek 4: Vlevo rozvržení stránek ve francouzském přehybu. Vpravo rozvržení stránek ve dvouminutové knize.

Dalším jednoduchým formátem je francouzský přehyb, kde se papír přehne napůl a poté znovu napůl ve druhém směru. Vizte obrázek 4. Často se tento formát používá pro papírová přání.

William Adams označuje formát své brožury jako *stroke book*[2], zatímco Cherryl Moote jej nazývá *dvouminutová kniha*. Rovržení stránek je na obrázku 4.

Instrukce ke složení brožury jsou následující.

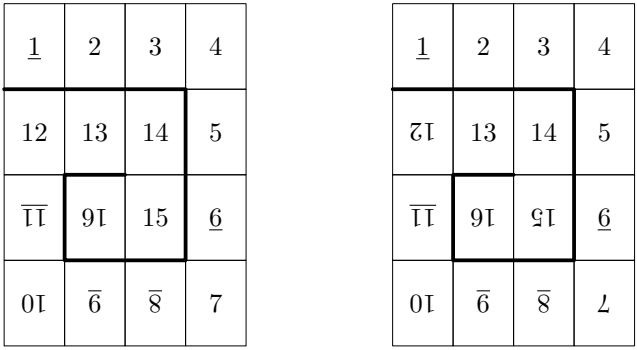
> Přehněte delší stranu papíru napůl (stránky 1|2 přehněte na stránky 6|5) textem ven (přehyb tvaru kopce). Každou půlku přehněte znovu napůl (6|5 na 7|4 a 1|2 na 8|3) textem dovnitř (přehyb tvaru údolí). Celý papír rozložte. Přehněte kratší stranu napůl (1|8|7|6 na 2|3|4|5) jako kopec. Přestřihněte polovinu dlouhého přehybu (tlustá čára). Zatlačte krajní strany směrem proti sobě (1|2 k 6|5 a 6|5 k 1|2), aby se střední strany dotýkaly (3 se dotýká 4 a 7 se dotýká 8). Nakonec přehněte stránky do požadovaného pořadí.

Nevěděl jsem, jakým způsobem William sesázel svých osm stránek takto na jeden list. Tipoval jsem, že nejdříve vysázel každou stránku na zvláštní list a pak použil nějaký software na stránkovou montáž, jako například `psnup`. Nakonec mi William napsal, že[3]

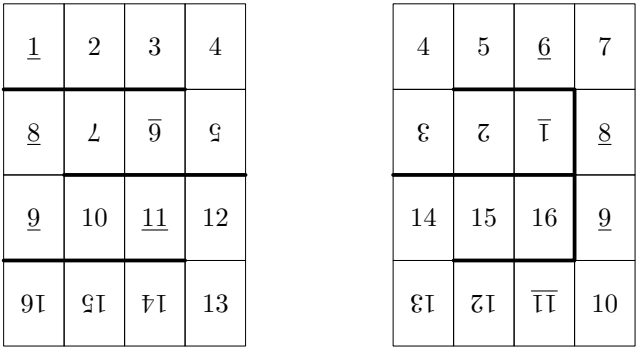
> *Vytvořil jsem jednotlivé stránky v kreslicím programu Altsys Virtuoso na mém NeXT Cube. Uložil jsem je jako eps soubory (užitečná funkce pdf*

[2] Překladateli se nepodařilo rozluštit, který z desítek významů slova *stroke* měl autor na mysli. (pozn. překl.)

[3] Osobní email 25.6.2008.



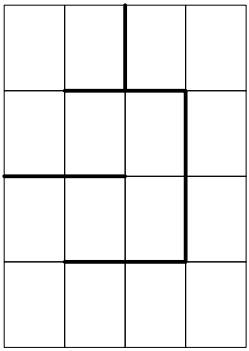
Obrázek 5: Vlevo spirálové rozvržení (počáteční přehyb tvaru kopce). Vpravo spirálové rozvržení (počáteční přehyb tvaru údolí).



Obrázek 6: Vlevo hadí rozvržení (počáteční přehyb tvaru kopce). Vpravo rozvržení tvaru T (počáteční přehyb tvaru kopce).

prohlížeče, který jsem používal). Pak jsem jednu stránku po druhé ručně posouval a otáčel na základě makety, kterou jsem si poskládal.

Pro stránkovou montáž existují některé čistě LaTeXové metody. Například balíček booklet [5] pro vytvoření brožury ze zmenšených stránek originálu, nebo balíček flowfram [6] Nicoly Talbot, který umožňuje definovat rámy na stránce a text pak automaticky přetéká z jednoho rámu do dalšího. Balíček pdfpages [7] Andrease Matthiase poskytuje nástroje pro stránkovou montáž v pdfLaTeXu. Kolekce programů PSUtils [8] Anguse Duggana obsahuje několik programů, například již zmiňovaný psnup, pro stránovou montáž postscriptových souborů. Pro pdf soubory existuje podobná kolekce programů Multivalent [9] Toma Phelpse. Já



Obrázek 7: Kamnové rozvržení.

jsem však chtěl vytvořit miniaturní knihy básniček a epigramů, vytištěné na jednu stranu jediného papíru, a to v LaTeXu a bez použití dalších balíčků. Pro tento účel se zdálo vhodné dělat stránkový zlom ručně a ne automaticky, jak to dělá balíček flowfram.[4]

Cherryl uvedla několik rozvržení stránek, které po vytištění na jediný papír běžného formátu[5] lze poskládat do miniaturní knihy. Tato rozvržení jsou na obrázcích 5–6. Abyste po vytištění složili knihu, nejdříve papír nastříhněte podél tlustých čar a poté papír přehýbejte v pořadí od první do šestnácté strany. Postupně střídejte přehyby tvaru kopce a údolí. Při čtení je pak třeba knihu různě otáčet.

Netuším, zdali tato jednotlivá rozvržení mají nějaké ustálené názvy, proto jsem je nazval vlastními slovy.

Samozřejmě si můžete navrhnout vlastní rozvržení. Pro sebe jsem si udělal rozvržení na obrázku 7. Netvrdím, že je to pěkné nebo užitečné ani nedávám návod, jak mají být stránky orientovány. Tvar řezů mi vzdáleně připomínal přikládání dřeva do kamen, proto jsem rozvržení nazval kamnové.

Možná jste zvědaví, jak byly vytvořeny všechny tyto diagramy. A i když nejste, odpovím. Vytvořil jsem je s využitím balíčku graphicx a prostředí `picture`. Takto například vypadá základ zdrojového kódu obrázku 5 vlevo – po čase se psaní toho kódu stává otravným.[6]

```
1 %% otočení argumentu vzhůru nohama
2 \newcommand*{\rupd}[1]{%
```

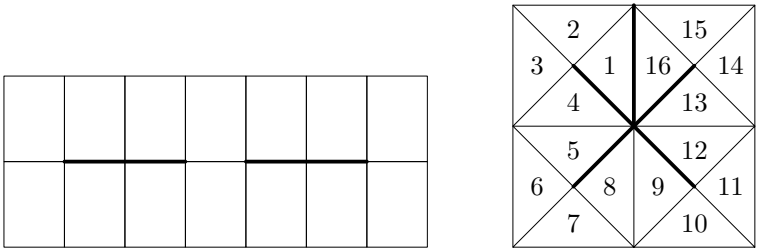
[4]Také se mi nepodařilo zjistit, jak lze automaticky vložit číslo strany dovnitř rámů.

[5]Konkrétně jde o formáty A4 a letter.

[6]Protože překladatel neměl k dispozici původní zdrojový kód článku a ani dostatek trpělivosti, rozhodl se tyto diagramy vysázet v METAPOSTu, kde s využitím vhodných maker byl zdrojový kód obrázků natolik jednoduchý, že k onomu „po čase“ nestihlo dojít. (pozn. překl.)

```
  \rotatebox[origin=c]{180}{#1}}
%%%% k ušetření psaní
\let\ul\underline
%%%% diagram
\begin{figure}
  \centering
  \setlength{\unitlength}{0.003\textwidth}
  \begin{picture}(100,120)
    %%%% tenké čáry
    \thinlines
    \put(0,0){\framebox(100,120){}}
    \put(25,0){\line(0,1){120}}
    \put(50,0){\line(0,1){120}}
    ...
    \put(0,90){\line(1,0){100}}
    %%%% tlusté čáry
    \linethickness{2pt}
    \put(0,90){\line(1,0){75}}
    \put(75,90){\line(0,-1){60}}
    \put(75,30){\line(-1,0){50}}
    \put(25,30){\line(0,1){30}}
    \put(25,60){\line(1,0){25}}
    %%%% čísla stran
    \linethickness{0pt}
    \put(0,90){\framebox(25,30){\ul{1}}}
    \put(25,90){\framebox(25,30){2}}
    ...
    \put(0,0){\framebox(25,30){\rupd{10}}}
    \put(25,0){\framebox(25,30){\rupd{\ul{9}}}}
    \put(50,0){\framebox(25,30){\rupd{\ul{8}}}}
    \put(75,0){\framebox(25,30){7}}
  \end{picture}
  \caption{Vlevo spirálové rozvržení ...}
  \label{fig:lay1M}
\end{figure}
```

Cherryl Moote popsala také některá další rozložení, kdy výsledná kniha je již složitější. Jedno z takových je na obrázku 8 vlevo. Jedná se pouze o spojení dvou dvouminutových rozvržení (viz obrázek 4 vpravo). Jestliže stránky přehneme jedním způsobem, dostaneme vazbu dos-à-dos, kdy dvě dvouminutové knihy mají společnou zadní desku. Když ale stránky přehneme jiným způsobem, budou mít obě knihy stránky vzájemně proložené.



Obrázek 8: Vlevo rozvržení pro vazbu dos-à-dos nebo pro proložené stránky. Vpravo trojúhelníkové leporelo.

Zcela jiný typ rozvržení je na obrázku 8 vpravo – trojúhelníkové leporelo. Čísla označují posloupnost jednotlivých stránek, jejich orientace je ponechána na autorovi.

Abyste věděli, které z těch mnoha možných rozvržení je vhodné pro váš projekt, je třeba si všechna rozvržení vyzkoušet a experimentovat s nimi.

Pro představu, jak vytvořit knihu z uvedených rozvržení, nabízím obrázky 9 a 10. Zkopírujte si tyto obrázky na celou stranu A4 a pak stříhejte, přehýbejte, zkoušejte a experimentujte. Obrázek 9 je určený pro spirálové rozvržení podle obrázku 5 vlevo – první strana je titulní a šestnáctá je tiráž. Obrázek 10 je určený pro spirálové rozvržení podle obrázku 5 vpravo – v tomto případě jsou první a šestnáctá strana prázdné a je možné na ně přilepit desky knihy, výsledek pak bude vypadat opravdu profesionálně.

Pokud se chcete pokusit o podobnou knihu, uvádím zdrojový kód rozvržení na obrázku 10.[7]

```
%%%% rozměry brožury
\newlength{\across}
\newlength{\down}
\setlength{\across}{0.2\textwidth}
\setlength{\down}{0.2\textheight}
%%%% nulová mezera mezi \fbox a jeho obsahem
\setlength{\fboxsep}{0pt}
\let\fbx\fbox
%%%% prostředí vplace je definováno ve třídě memoir
%%%% jeho obsah se vertikálně vycentruje
```

[7] V původním článku na sebe nenavazovaly okraje jednotlivých stránek, a to horizontálně ani vertikálně. Aby navazovaly horizontálně, bylo třeba přidat makro `\bezmezery` a na řádku 78 přesně nastavit šířku sazby. Aby navazovaly vertikálně, bylo třeba na řádku 80 přesně nastavit mezi řádkovou mezeru. Ona magická konstanta $-0.4\,\mathrm{pt}$ v obou případech znamená záporné posunutí o tloušťku čáry. (pozn. překl.)

Vitae Summa Brevis Ernest Dowson		They are not long, 3	The weeping and the laughter, 4
We pass the gate. 8	7 in us after	6 I think they have no portion	Love and desire and hate: 5
They are not long, 9	the days of wine and roses: 10	Out of a misty dream 11	Our path emerges for a while, 12
2008 The Herries Press		14 Within a dream.	then closes 13

Obrázek 9: Miniaturní kniha podle obrázku 5 vlevo.

	Vitae Summa Brevis Ernest Dowson	They are not long, 3	The weeping and the laughter, 4
8 We pass the gate.	7 in us after	6 I think they have no portion	5 Love and desire and hate:
They are not long, 9	the days of wine and roses: 10	Out of a misty dream 11	Our path emerges for a while, 12
	2008 The Herries Press	14 Within a dream.	13 then closes

Obrázek 10: Miniaturní kniha podle obrázku 5 vpravo.

```
\providecommand{\vplace}[1][1]{%
  \par\vspace{\stretch{#1}}}
\def\endvplace{\vspace*{\stretch{1}}\par}
%%%% vloží jednu minipage vycentrovanou dovnitř druhé
\newcommand{\portion}[1]{\fbx{%
  \begin{minipage}[c][\down][t]{\across}
    \centering
    \begin{minipage}[c][\down][t]{0.8\across}
      #1%
    \end{minipage}
  \end{minipage}}}
%%%% vertikálně vycentruje obsah \portion
\newcommand{\vcp}[3][1]{%
  \portion{\centering%
    \begin{vplace}[#1]#2\end{vplace}#3%
    \vspace*{\baselineskip}}%
  \bezmezery}
%%%% otočí obsah \vcp vzhůru nohama
\newcommand{\rvcp}[3][1]{%
  \rotatebox[origin=c]{180}{%
    \let\bezmezery\relax
    \vcp[#1]{#2}{#3}}%
  \bezmezery}
%%%% aby stránky na sebe horizontálně navazovaly
\newcommand{\bezmezery}{%
  \penalty0 \hskip-.4pt \ignorespaces}
%%%% čísla stran
\newcommand*{\pgn}[1]{{\tiny #1}}
%%%%%% sazba
\centerline{%
  \hsize\dimexpr.8\textwidth+2pt
  \vbox{
    \lineskip-.4pt
    \noindent
    %%%% řada 1
    \vcp{\mbox{}}{}
    \vcp{%
      \large \textit{Vitae Summa Brevis} \\[5mm]
      \large Ernest Dowson}{}
    \vcp{They are not long,}{\pgn{3}}
    \vcp{The weeping and the laughter,}{\pgn{4}}
    %%%% řada 2
```

```
    \rvcp{We pass the gate.}{\pgn{8}}
    \rvcp{in us after}{\pgn{7}}
    \rvcp{I think they have no portion}{\pgn{6}}
    \rvcp{Love and desire and hate:}{\pgn{5}}
    %%%% řada 3
    \vcp{They are not long,}{\pgn{9}}
    \vcp{the days of wine and roses:}{\pgn{10}}
    \vcp{Out of a misty dream}{\pgn{11}}
    \vcp{Our path emerges for a while,}{\pgn{12}}
    %%%% řada 4
    \vcp{\mbox{}}{}
    \rvcp{{\footnotesize The Herries Press\\[1cm]
      2008}}{}
    \rvcp{Within a dream.}{\pgn{14}}
    \rvcp{then closes}{\pgn{13}}
}}
```

Pro sazbu jsem nevyužil celou tiskovou oblast, ale pouze její část. Na řádcích 41–42 lze nastavit rozměr této části. Pokud používáte třídu memoir nebo balíček geometry, můžete snadno změnit rozměry tiskové oblasti. Při sazbě jsou jednotlivé stránky zarámované makrem `\fbox` (použitým uvnitř `\fbx`). Jestliže zarámování nechceme, předefinujeme například nad řádkem 77 makro `\fbx` následovně.

```
\renewcommand*{\fbx}[1]{#1}
```

Nechť vám je tvorba vlastních miniaturních knížek potěšením.

Poděkování

William Adams byl tak hodný, že prošel celý můj článek a já jsem pak mohl zapracovat mnoho z jeho návrhů. Jeden návrh jsem však nezapracoval a činím tak teď. William navrhuje vzít si jako další zdroj různých rozvržení nějakou literaturu o origami, například *The Folding Universe* [10]. I když není tato kniha přímo k tématu, stojí za to si ji prohlédnout.

Odkazy

1. WILSON, Peter. Glisterings. *TUGboat* [online]. 2010, roč. 31, č. 3, s. 177–183 [cit. 2021-12-06]. Dostupné z: `https://tug.org/TUGboat/tb31-3/tb99glister.pdf`.

2. ADAMS, William. *One Typeface, Many Fonts* [online]. 1997 [cit. 2010-11]. Dostupné z: `http://mysite.verizon.net/william_franklin_adams/portfolio/typography/onetype-sheet.pdf`.
3. BROMER, Anne C.; EDISON, Julian I. *Miniature Books: 4,000 Years of Tiny Treasures*. Abrams, 2007. ISBN 9780810992993.
4. MOOTE, Cherryl. *Copied, Bound & Numbered*. At Your Ease Publications, 2003. ISBN 0968881173.
5. WILSON, Peter. *Printing booklets with LaTeX* [online]. 2009 [cit. 2010-11]. Dostupné z: `https://ctan.org/pkg/booklet`.
6. TALBOT, Nicola L. C. *Creating flow frames for posters, brochures or magazines using flowfram.sty* [online]. 2010 [cit. 2010-11]. Dostupné z: `https://ctan.org/pkg/flowfram`.
7. MATTHIAS, Andreas. *The pdfpages package* [online]. 2010 [cit. 2010-11]. Dostupné z: `https://ctan.org/pkg/pdfpages`.
8. DUGGAN, Angus. *PSUtils* [online]. 2008 [cit. 2010-11]. Dostupné z: `http://www.tardis.ed.ac.uk/~ajcd/psutils`.
9. PHELPS, Tom. *Multivalent* [online]. 2008 [cit. 2010-11]. Dostupné z: `http://multivalent.sourceforge.net`.
10. ENGEL, Peter. *The Folding Universe*. Vintage, 1989. ISBN 0394757513.

Summary: It Might Work XI

This paper shows several ways how to create a miniature book printed on just a single sheet of paper. Some LaTeX solutions are given.

Keywords: Miniature books, LaTeX, folding

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

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:	Donald Ervin Knuth
Počet výtisků:	260
Uzávěrka:	7. 12. 2021
Odpovědný redaktor:	Jan Šustek
Redakční rada:	Pavel Haluza, Lukáš Novotný, Vít 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 Novotný
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`, `zpravodaj@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