Aktualizace srpen 2022

První setkání s TEXem

Petr Olšák

Autor programu TEX je profesor Donald Knuth.

TEX je ochranná známka American Mathematical Society.

Ostatní v manuálu použité názvy programových produktů, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Toto je aktualizované a zcela přepracované vydání původního textu „První setkání s TEXem“ z roku 1999.

Úvod

Tento manuál je koncipován jako „první seznámení s programem TEX[1] na jeden večer“. Měl by umožnit začátečníkovi porozumět základním principům TEXu. Manuál obsahuje ukázky jednoduchých dokumentů, které si může čtenář převzít do svého počítače a na nich TEX vyzkoušet. Je to dobrý první krok do pestrého světa tohoto programu plného zajímavých možností. Předvedené ukázky mimo jiné ilustrují základní principy psaní dokumentů v TEXu. Jsou zde úvodní ukázky pro plainTEX, OpTEX, LATEX a pro ConTEXt. Sekce 8 dává také srovnání těchto možností, takže si uživatel může hned rozhodnout, co je bližší jeho srdci a podle toho vybrat další literaturu pro doplňující studium.

Text manuálu uvádí moderní postupy práce s TEXem (rok 2022). Zastaralé postupy jsou pro úplnost shrnuty v závěrečné sekci 10.

Předpokládáme, že čtenář má určité důvody proč použít TEX, takže se zde nebudeme zdržovat výčtem jeho výhod, rozepisovat obšírně jeho historii a nebudeme polemizovat o užitečnosti či neužitečnosti dávkového či interaktivního systému na přípravu sazby.

Obsah

[1] Název TEX se čte „tech“, nikoli „teks“.

1 TEX a jeho okolí

TEX je formátor. Je to program, kterému předložíme vstupní text dokumentu v „holé“ textové podobě doplněný textovými značkami, které vymezují strukturu dokumentu nebo dávají TEXu pokyny o způsobu formátování dokumentu. Bývá obvyklé (ale není to nutné) pojmenovat tento soubor s použitím přípony `.tex`, například `dokument.tex`. Na výstupu pak po zpracování TEXem dostaneme formátovaný dokument v PDF souboru (`dokument.pdf`).

TEX tedy čte na svém vstupu textový soubor s dobře definovanou syntaxí jazyka značek a na výstupu je soubor s definitivním popisem sazby. Další programy a navazující software „okolo TEXu“ tvoří společně s TEXem *distribuci*. Dnes se nejčastěji používají volně přístupné distribuce TEXlive nebo MikTEX.

Začínající uživatel se samozřejmě hlavně ptá po způsobu, jak může v konkrétním operačním systému s konkrétní TEXovou distribucí s tímto programem pracovat, jak jej spustit, jakými tlačítky se ovládá textový editor, jaké nabídky jsou k dispozici, co nad kterým obrázkem udělá myš. Ptá se tedy po uživatelském rozhraní. Jednotlivé manuály o TEXu tradičně odkazují na tzv. „místní příručku“ (Local Guide), která by měla toto rozhraní popisovat. Tato příručka je závislá na použitém operačním systému, na čase jejího vzniku, na použité distribuci TEXu, na vybraném textovém editoru a někdy též na administrátorovi systému, který konfiguruje některé věci specificky pro větší pohodlí uživatelů. Manuály o TEXu uživatelské prostředí většinou nezmiňují (je totiž závislé na okolnostech) a popisují pouze na systému nezávislé vlastnosti TEXu jako formátoru. Ani tento text není v tomto ohledu výjimkou.

Při práci s TEXem je obvyklé mít otevřen v jednom okénku textový editor, ve kterém uživatel píše nebo modifikuje vstupní text, a ve vedlejším okénku prohlížeč výstupního souboru. Po modifikaci vstupního textu uživatel spustí TEX na pozadí například nějakou klávesovou zkratkou a ve vedlejším okénku vidí během pár sekund výslednou změnu v sazbě.

Textový editor, ve kterém připravujeme nebo modifikujeme vstupní texty dokumentů, nesmí ukládat na disk žádné skryté formátovací informace implementované jen pro tento editor (jako například změna fontu, měkké konce řádku apod.). To dělají tzv. textové procesory, které v případě práce s TEXem nepoužíváme.

Overleaf (`https://www.overleaf.com`) je příklad webového rozhraní umožňující spouštět TEX vzdáleně bez nutnosti jej instalovat a nabízí sdílenou přípravu dokumentů i více uživateli najednou.

2 TEXová rozšíření (engines)

Vývoj originálního TEXu byl autorem D. Knuthem zastaven v roce 1990. Následně začala vznikat rozšíření (TEX engines), která se chovají jako původní TEX, ale nabízejí další možnosti:[2]

PdfTEX	jako první nabízí přímý výstup do PDF souboru, neumí Unicode,
XETEX	podporuje navíc Unicode,
LuaTEX	podporuje Unicode a nabízí interní skriptovací jazyk Lua pro řízení sazby.

Všechna tato rozšíření jsou implementována jako samostatně spustitelné programy a jsou obsažena v běžných TEXových distribucích (TEXlive nebo MikTEX). V tomto manuálu se zaměříme výhradně na XETEX a LuaTEX[3], protože bez podpory Unicode jsou věci daleko složitější. O zastaralých postupech opírajících se o originální TEX nebo pdfTEX se zmíníme v poslední sekci. Je-li v operačním systému k dispozici příkazový řádek, pak XETEX spustíte pomocí příkazu `xetex` a LuaTEX pomocí příkazu `luatex`.

[2] Zde zmíněná rozšíření umějí navíc plno dalších věcí, které detailně nerozebíráme.

[3] Z hlediska perspektivy dalšího vývoje doporučujeme pracovat s LuaTEXem, protože nejenže toho umí daleko víc, ale navíc to vypadá, že vývoj XETEXu se zastavil.

3 Výchozí sady TEXových maker (formats)

Značky pro řízení sazby vkládané do dokumentu obvykle začínají znakem \ (například `\bf` pro přepnutí do tučného písma) a jsou to vesměs *makra*. Tím se myslí, že jakmile TEX na takovou značku narazí, rozvine ji podle dříve vložené definice na jednotlivé nízkoúrovňové příkazy a tyto příkazy vykoná. Těm nízkoúrovňovým příkazům se říká *TEX-primitivní příkazy*. Makrům se často taky nepřesně říká „příkazy“, protože nakonec z pohledu uživatele je to pokyn k nějaké činnosti. Definice všech maker a nastavení parametrů sazby jsou uloženy v lidsky čitelných textových souborech. Některé takové sady maker se opakovaně používají při zpracování každého dokumentu, a proto je TEX po instalaci distribuce přečte jen jednou a zkonvertuje je do binární podoby pro pozdější rychlejší čtení. Instalační program to dělá automaticky. Například sada maker pro LATEX je připravena v textovém souboru `latex.ltx`. Instalační program spouští jednotlivá TEXová rozšíření s parametrem `-ini`, což je pokyn k inicializaci maker, nastavení a výchozích fontů, tj. k jejich přečtení a uložení do binární podoby, souboru s příponou `.fmt`. Těmto souborům se říká z historických důvodů „formáty“, protože původně byly určeny pro deklaraci formátu dokumentu: byly tam použité fonty, nastavení parametrů sazby a makra obsahovala konkrétní řešení umístění sazby. To vše dohromady udává výsledný vzhled dokumentu, tedy formát dokumentu.

Dnes tomu stále říkáme formáty, ale vzhled dokumentu se dá i po přečtení výchozího `.fmt` souboru specifikovat dodatečným vymezením dalších potřebných fontů, dodatečným nastavením parametrů sazby a čtením dodatečných maker.

TEX bez maker a výchozího nastavení je vybaven jen TEX-primitivními příkazy a je v této podobě skoro nepoužitelný. Takže při běžné práci s TEXem se vždy nejprve čte nějaký `.fmt` soubor s výchozími makry a pak teprve dokument, ve kterém mohou být vymezeny další parametry sazby a makra. Spustíte-li programy `xetex` nebo `luatex` bez přepínače `-ini` a také bez přepínače `-fmt`, tyto programy automaticky přečtou formát PlainTEX, což je minimální sada maker a nastavení, ze které vycházejí víceméně všechny další dnes používané formáty. PlainTEX byl připraven autorem TEXu a je detailně dokumentován v knize TEXbook.

Asi nejčastěji používaný formát TEXu je LATEX. Například jeho `.fmt` soubor pro rozšíření XETEX má jméno `xelatex.fmt`, takže je možné XETEX s LATEXem spustit pomocí příkazu `xetex -fmt xelatex.fmt`. TEXové distribuce implementují pro tento příkaz zkratku `xelatex`. Seznam běžně používaných zkratek pro spuštění TEXu je uveden v tabulce níže.

LATEX vznikl v roce 1984. Jeho autorem je Leslie Lamport, který údržbu a vývoj předal dalším vývojářům na počátku 90. let. Dnes na LATEXu pracuje mnohočlenný tým dobrovlníků.

Hans Hagen zveřejnil v roce 1990 jinou sadu výchozích maker nazvanou ConTEXt, která od roku 2009 spolupracuje výhradně s LuaTEXem[4]. Spustíte ji příkazem `context`.

Petr Olšák vytvořil v roce 2020 další výchozí sadu maker pro LuaTEX nazvanou OpTEX. Spustíte ji příkazem `optex` což je zkratka za `luatex -fmt optex.fmt`.

Seznam běžně používaných příkazů pro spuštění TEXu:

`xetex`	PlainTEX v rozšíření XETEX
`luatex`	PlainTEX v rozšíření LuaTEX
`xelatex`	LATEX v rozšíření XETEX
`lualatex`	LATEX v rozšíření LuaTEX
`context`	ConTEXt
`optex`	OpTEX

K dispozici jsou i další příkazy (např. `latex`, `pdflatex`, `csplain`, `pdfcsplain`, `cslatex`, které jsou zmíněny v sekci 10. Jsou zastaralé a dnes už je nedoporučujeme používat.

Vývojáři Overleaf zřejmě se přátelí výhradně s LATEXem, takže vytvářejí podporu pro syntaktickou kontrolu LATEXových dokumentů a ve své dokumentaci žádný jiný TEXový formát

[4] V nejnovější verzi ConTEXtu je třeba použít LuaMetaTEX, což je nová vývojová větev LuaTEXu. Dnes (2022) zatím není přítomná v TEXových distribucích. Tam je k dispozici verze ConTEXu MkIV opírající se o původní LuaTEX.

nezmiňují, jakoby další formáty neexistovaly. Ale interně má Overleaf instalovaný kompletně celý TEXlive, takže se v něm dají spustit i formáty PlainTEX, ConTEXt nebo OpTEX. Dělá se to pomocí vložení souboru `latexmkrc` do projektu se specifickým obsahem. Viz například projekt https://www.overleaf.com/read/gghtcbvrpcdb který spouští OpTEX.

Shrnutí. Existují čtyři běžně používané formáty:

PlainTEX		základní makra od autora TEXu
LATEX	https://www.latex-project.org/	nejběžněji používaný formát
ConTEXt	https://wiki.contextgarden.net/	formát s jinou koncepcí než LATEX
OpTEX	https://petr.olsak.net/optex	nejblíže podobný PlainTEXu

O vlastnostech těchto formátů pojednávají další sekce, kde najdete také ukázky dokumentů určené pro konkrétní formáty.

4 Ukázka v plainTEXu

Zkuste si v nějakém textovém editoru vytvořit soubor `test.tex`, který obsahuje zkušební větu:

```
Hello world!
\bye
```

Pokud zpracujete tento soubor příkazem `luatex test`[5], dostanete výstupní soubor `test.pdf`. Navíc LuaTEX uloží informaci o zpracování do souboru `test.log`. Výsledný PDF soubor si můžete prohlédnout jakýmkoli PDF prohlížečem. Dostanete očekávaný výsledek:

Hello world!

Přitom dole na stránce je ještě vytištěno číslo strany: 1.

Vstupní dokument obsahuje přímo text, který chceme zformátovat. V tomto případě je tam ještě jedna řídicí sekvence `\bye`, která dává pokyn k ukončení činnosti TEXu. Kdyby tam nebyla, TEX by se bohužel nezastavil na konci souboru, ale chtěl by další části dokumentu číst z terminálu tak dlouho, dokud mu neřekneme `\bye` přímo na terminálu.

Zpracujete-li tentýž soubor LATEXem, tedy například příkazem `lualatex test`[6], obdržíte chybové hlášení:

```
! LaTeX Error: Missing \begin{document}.
```

Koncept značkování v LATEXu je totiž zásadně odlišný, je třeba vložit do dokumentu další informace. K tomu se vrátíme podrobněji v sekci 6.

PlainTEX bohužel implicitně pracuje s fonty obsahujícími jen ASCII znaky, takže text v jiném jazyce než v angličtině nebude fungovat. Proto záhy vznikly různé modifikace plainTEXu, například CSplain podporující českou a slovenskou sazbu, viz sekci 10. Tím se ale nebudeme podrobně zabývat, protože dnes máme k dispozici OpTEX, což je „moderní plainTEX“ pracující s Unicode fonty a podporující vesměs všechny jazyky.

[5] Takže se spustí rozšíření LuaTEX s formátem PlainTEX a přečte se soubor `test.tex`.

[6] Spustí se LuaTEX s formátem LATEX a následně se přečte soubor `test.tex`.

5 Ukázka dokumentu v OpTEXu

Pomocí příkazu `optex test` bez problémů zpracujeme dokument „Hello world!“ z předchozí sekce, protože OpTEX respektuje plainTEXovou koncepci značkování dokumentu. OpTEX ovšem nabízí plno dalších možností v původním plainTEXu nedostupných (nebo obtížně dostupných). Na první pohled nejpodstatnější možností je příkaz `\fontfam`, kterým zavedeme Unicode fonty vybrané fontové rodiny. Bez toho bychom nemohli zpracovávat český text. Významy úvodních příkazů jsou vysvětleny za komentářovým znakem %.

```
\fontfam[Termes]  % Rodina fontů Termes, svobodná alternativa k Times.
\typosize[12/14]  % Základní font 12 bodů, řádkování 14 bodů.
\cslang           % Použijí se pravidla pro české dělení slov.
\csquotes         % Značky \"..." vytvoří české uvozovky.
\verbchar`        % Pomocí `...` budeme zapisovat verbatim kódy.

\tit Můj první dokument

Zkouším napsat první text v~\TeX/u. Tento odstavec musí být
tak dlouhý, aby bylo vidět, že se rozlomil aspoň na dva řádky.

Jednotlivé odstavce oddělujeme od sebe prázdným řádkem. Prázdnými řádky
vůbec nešetříme, protože zvyšují přehlednost zdrojového textu.
Vyzkoušíme si nyní několik věcí.

\begitems
* Budeme používat české \"uvozovky", které se liší od anglických.
  Uvědomíme si, že použití "těchto znaků" je úplně špatně!
* Rozlišujme mezi spojovníkem (je-li), pomlčkou ve větě~---
  a dlouhou pomlčkou---ta se používá v~anglických dokumentech.
* Zkusíme přepnout do {\bf tučného písma}, nebo do
  {\it kurzívy}. Také vyzkoušíme {\tt strojopis}.
* Vypravíme se na malou exkurzi do matematiky: $a^2 + b^2 = c^2$.
  Zjistíme, že číslo -1 je zde napsáno špatně (prokletý spojovník),
  zatímco správně má být $-1$.
* Protože \% uvozuje komentář a \$ přepíná do matematické sazby,
  musíme před ně napsat zpětné lomítko, chceme-li je dostat do dokumentu.
\enditems

\nonum\sec Závěr

To by pro začátek stačilo. Příkazem `\bye` ukončíme své pokusy.
\bye
```

Na výstupu po zpracování příkazem `optex dokument` pak dostaneme:

Můj první dokument

Zkouším napsat první text v TEXu. Tento odstavec musí být tak dlouhý, aby bylo vidět, že se rozlomil aspoň na dva řádky.

Jednotlivé odstavce oddělujeme od sebe prázdným řádkem. Prázdnými řádky vůbec nešetříme, protože zvyšují přehlednost zdrojového textu. Vyzkoušíme si nyní několik věcí.

- Budeme používat české „uvozovky“, které se liší od anglických. Uvědomíme si, že použití ”těchto znaků” je úplně špatně!
- Rozlišujme mezi spojovníkem (je-li), pomlčkou ve větě – a dlouhou pomlčkou—ta se používá v anglických dokumentech.
- Zkusíme přepnout do **tučného písma**, nebo do *kurzívy*. Také vyzkoušíme `strojopis`.
- Vypravíme se na malou exkurzi do matematiky: $a^2 + b^2 = c^2$. Zjistíme, že číslo -1 je zde napsáno špatně (prokletý spojovník), zatímco správně má být -1.
- Protože % uvozuje komentář a $ přepíná do matematické sazby, musíme před ně napsat zpětné lomítko, chceme-li je dostat do dokumentu.

Závěr

To by pro začátek stačilo. Příkazem `\bye` ukončíme své pokusy.

Na tomto příkladě si předvedeme základní vlastnosti TEXu platné i pro další formáty. Pokud je něco platné výhradně jen v OpTEXu, bude to zde výslovně uvedeno.

5.1 Oddělení obsahu od formy

Ve vstupním textu dokumentu bychom měli důsledně dbát na oddělení obsahu od formy. Forma je v ukázce vymezena pomocí příkazů `\fomtfam` a `\typosize`. Zatímco od řádku `\tit` je už jen se smluvenými značkami zapsán obsah dokumentu. Pokud se nám například nelíbí, jak makro `\tit` umístí nadpis, jakou dá mezeru pod nadpis, jak velkým písmem nadpis vytvoří, není správné příkazy, které toto zařídí, psát jednotlivě do dokumentu místo makra `\tit`. Je spíše vhodné v části vymezující formu dokumentu předefinovat výchozí makro `\tit` tak, aby to vyhovovalo našim typografickým požadavkům.

Ideálně na dokumentu mohou pracovat dva lidé: autor, který píše obsah doplněný smluvenými značkami a nemusí rozumět, jak jsou značky definovány. Dále na dokumentu pracuje typograf, který nastavuje parametry sazby a definuje smluvené značky jako makra tak, aby to bylo v souladu s typografickým návrhem. Ten musí pochopitelně umět TEX.

Pro autora textu jsou důležité následující vlastnosti TEXu.

5.2 Odstavce, řídicí sekvence, konce řádků, mezery

Text se dělí na odstavce. Ve vstupním souboru oddělujeme odstavce od sebe prázdným řádkem. Tvrdé ukončení řádku (v ukázce třeba za slovem „být“) neukončí odstavec. Nicméně je možné napsat celý odstavec do jediného řádku ve vstupním souboru, ale další odstavec musí být vždy oddělen od předchozího prázdným řádkem.

Řídicí sekvence jsou uvozeny znakem \ (zpětné lomítko) a jejich název sestává výhradně z písmen, nikoli z číslic nebo jiných znaků. Jakmile se při čtení názvu řídicí sekvence objeví znak, který není písmeno, TEX řídicí sekvenci (bez toho znaku) považuje za přečtenou a v běžném kontextu ji také vykoná. První následující nepísmenný znak pak pokračuje jako parametr řídicí

sekvence nebo je už součástí dalšího textu k vytištění. Je-li prvním nepísmenným znakem za řídicí sekvencí mezera, ignoruje se.

Konec řádku funguje jako mezera, takže v textu máme „být“ na konci jednoho řádku a „tak dlouhý“ na začátku dalšího. Vytiskne se „být tak dlouhý“ nikoli „býttak dlouhý“.

Více mezer vedle sebe se chová jako jediná mezera. Libovolné množství mezer na začátku řádku je ignorováno, takže mezerami můžete odsadit řádky vstupního textu pro větší přehlednost. V ukázce jsme odsadili text mezi `\begitems` a `\enditems`.

Řídicí sekvence `\fontfam`, `\typosize`, `\cslang`, `\csquotes`, `\verbchar`, `\tit`, `\begitems`, `\enditems` `\nonum`, `\sec` použité v ukázce jsou makra, která jsou k dispozici výhradně v OpTEXu. Jejich význam (a význam mnoha dalších) najdete v uživatelské dokumentaci k OpTEXu. Řídicí sekvence `\TeX`, `\bf`, `\it`, `\tt`, `\bye` jsou makra definovaná i v PlainTEXu a OpTEX je zná taky.

5.3 Skupiny vymezené `{` a `}`

Závorky `{...}` mají v TEXu speciální význam navíc závislý na kontextu. V běžném textu závorka `{` otevře skupinu ve které veškerá nastavení jsou uložena lokálně, takže když závorka `}` ukončí skupinu, vrací se TEX k původnímu nastavení. Jejich použití vidíte v textu `{\bf tučného písma}`, kde uvnitř skupiny je nastaveno přepínačem `\bf` tučné písmo a toto nastavení mizí na konci skupiny. Skupiny lze do sebe vnořovat. V jiném kontextu se tyto závorky používají jako vymezovače parametrů maker, což ještě uvidíme zejména v LATEXu.

5.4 Nedělitelná mezera

Znak `~` má ve všech formátech speciální význam: je to nedělitelná mezera. V ukázce vidíme jeho použití za předložkou v. V češtině totiž neslabičné předložky nesmějí zůstat na konci řádku. V OpTEXu je možno na začátku dokumentu zavést speciální balíček, který vymění normální mezeru za nezlomitelnou za všemi neslabičnými předložkami automaticky, takže nemusíte vlnky za předložky explicitně psát. Balíček zavedete na začátku dokumentu pomocí `\load[vlna]` a funguje to za předpokladu, že je aktivována česká nebo slovenská sazba pomocí `\cslang` nebo `\sklang`. Ovšem existuje ještě plno dalších míst, kde by měla být nezlomitelná mezera, což je třeba ošetřit manuálně vložením znaku `~`. Tady vidíme typickou vlastnost TEXu: veškeré speciální pokyny pro sazbu jsou *viditelné*, tj. znak nezlomitelné mezery `~` ve vstupním textu vidíme. Kdybychom používali znak nezlomitelné mezery definovaný svým kódem `0xA0`, neviděli bychom ve zdrojovém textu žádnou odlišnost od normální mezery, což je nepraktické.

5.5 Logo `\TeX`, mezera za řídicí sekvencí se ignoruje

Řídicí sekvence `\TeX` vytiskne logo TEX a je v ukázce ukončena znakem `/`, který se nevytiskne. To je vlastnost výhradně OpTEXu. V jiných formátech je řídicí sekvence `\TeX` definována také, ale bez možnosti ji ukončovat znakem `/`. V takovém případě je třeba psát `v~\TeX u`, protože mezera za řídicí sekvencí ji ukončí a dále se ignoruje (nevytiskne). Budete mít ale problém, pokud za řídicí sekvencí `\TeX` mezeru skutečně chcete vytisknout například ve větě „TEX může použít každý“. V OpTEXu stačí psát `\TeX/ může použít každý` zatímco v ostatních formátech je třeba psát `\TeX{} může použít každý`. V tomto případě byla za sekvencí `\TeX` otevřena a hned zavřena skupina, což v sazbě nic neudělá, ale následující mezera už se normálně vytiskne.

5.6 Jednoznakové řídicí sekvence

Existují také řídicí sekvence uvozené znakem `\` za nímž následuje jediný znak, kterým není písmeno. V ukázce vidíme takové sekvence `\"`, `\%` a `\$`. Případná mezera za takovými sekvencemi se tentokrát neignoruje.

5.7 Seznam znaků se speciálním významem

TEX má typicky nastaveny tyto znaky jako speciální:

`\`	uvozuje řídicí sekvence
`{, }`	začátek a konec skupiny nebo dalších syntaktických konstrukcí
`$`	začátek a konec matematické sazby
`%`	zahájení komentáře do konce řádku
`~`	nezlomitelná mezera
`^, _`	konstruktory pro exponenty a indexy v matematické sazbě
`&`	vymezení položek v tabulce
`#`	označení parametru v textu maker

To pochopitelně znamená, že tyto znaky nemůžete použít přímo v textu s tím, že očekáváte, že se jen vytisknou. Chcete-li je vytisknout, můžete použít jednoznakové řídicí sekvence `\{`, `\}`, `\&`. `\%`, `\_`, `\&`, a `\#`. Všimněte si, že tímto způsobem se nepodaří vytisknout znaky `\`, `~`, `^`. Pokud chcete takový znak přímo tisknout, je možno použít TEXový příkaz `\char` za kterým následuje číslo udávající kód znaku. To ty kódy si nemusíte pamatovat, protože je lze zapsat ve tvaru `` `\znak ``, tedy obrácený apostrof následovaný zpětným lomítkem `\` a následovaný příslušným znakem. Takže třeba `` `\\ `` je kód znaku zpětné lomítko a `` \char`\\ `` tento znak vytiskne. Nebo `` \char`\~ `` vytiskne ~ a `` \char`\^ `` vytiskne ^. Místo toho je ale obvyklejší vymezit deklarací `\verbchar` další speciální znak pro doslovně psané kódy (*verbatim kódy*). Mezi dvojicí takových znaků se pak tiskne všechno doslova, jak je napsáno ve vstupním souboru.[7] V ukázce máme na předposledním řádku takto vytištěno slovo `\bye`.

Poznamenejme, že v OpTEXu je výjimečně znak `_` nastaven jako normální znak, takže jej vytisknete přímo bez komplikací. Přitom v matematické sazbě (tj. mezi znaky `$...$`) tento znak funguje jako konstruktor indexu stejně, jako v ostatních formátech.

5.8 Uvozovky

V ukázce jsou české uvozovky vytvořené pomocí jednoznakové řídicí sekvence `\"` následované uvozeným textem a ukončené znakem `"`. To je možné, je-li předem deklarováno `\csquotes`. Kdybychom místo toho deklarovali třeba `\enquotes`, měli bychom v celém dokumentu místo českých anglické uvozovky. To je vlastnost OpTEXu. Je ovšem možné vždy psát uvozovky do vstupního souboru přímo, tedy `„takto“`. Ovšem mnohým lidem se takové znaky špatně píší, nemají je třeba přímo na klávesnici snadno dostupné nebo se v textovém souboru zobrazují ne zcela věrohodně a nemáte jistotu, zda jsou správě použity. Pak je možné si české uvozovky definovat třeba značkou `\uv` na začátku dokumentu takto:

```
\def\uv#1{„#1"}
```

a můžete pak tuto značku používat takto: `\uv{text}` což vytiskne „text“. Definuje se zde makro `\uv` s jedním parametrem `#1`, který je při použití zapsaný za sekvencí `\uv` a je obklopen závorkami `{...}`, jež v tomto kontextu nemají význam otevření a zavření skupiny, ale vymezují začátek a konec parametru. V definici je její obsah také uzavřen mezi `{...}` a je tam řečeno, že nejprve se vytiskne znak „ a za ním příslušný parametr a nakonec znak “. Takovou definici si TEXový uživatel napíše, pokud nechce opakovaně hledat znaky „ a “ na své klávesnici. Najde si je jen jednou, když píše tu definici.

[7] To je vlastnost OpTEXu, v jiných formátech se verbatim texty zapisují mírně jinak.

5.9 Neobvyklé znaky mají své řídicí sekvence

Psaní řídicích sekvencí místo neobvyklých znaků je v TEXu velmi běžné. Například pro stovky až tisíce znaků používaných v matematické sazbě jsou ve formátech TEXu připraveny řídicí sekvence[8], takže místo abyste psali `$α + β ≤ γ$` a dlouze hledali každý znak někde v nabídkách znaků, na klávesnici rychle napíšete `$\alpha + \beta \le \gamma$` a máte stejný výsledek $\alpha + \beta \le \gamma$. Matematici znají řídicí sekvence pro znaky, které používají ve svých textech, zpaměti a je pro ně tedy rozhodně jednodušší a hlavně rychlejší napsat na klávesnici `\beta` nebo `\le` než lovit v hlubinách rozsáhlých nabídek myší ten pravý. Seznam řídicích sekvencí pro matematickou sazbu najdete v různých příručkách, například v Typesetting Math with OpTEX v sekci 1.10. Také můžete vyzkoušet nástroj Detexify, ve kterém znak načrtnete a aplikace odpoví, jakou TEXovou řídicí sekvenci máte použít.

5.10 Ligatury

TEX automaticky tvoří ligatury (slitky): ve vstupním souboru napíšete `f` následované `i` a TEX to promění ve slitek fi, například ve slově fialka. To se stane tehdy, pokud použitý font tento slitek obsahuje a je pro něj zapnutá podpora nahrazování slitků (což implicitně je). Běžné typografické slitky jsou fi a fl, případně ffi a ffl. Kromě toho TEX používá své vlastní slitky navržené autorem TEXu pro přehlednější psaní některých na klávesnici nedosažitelných znaků. Dvojice spojovníků se promění v krátkou pomlčku a trojice těchto znaků v dlouhou pomlčku. V ukázce vidíme použití těchto slitků ve druhé odrážce. Dále existují slitky pro anglické uvozovky a španělské obrácené otazníky a vykřičníky.

5.11 Jiný výstupní vzhled prvního dokumentu

Na závěr této sekce ukážeme výhodu oddělení obsahu od formy. Zkuste například místo `\fontfam[Termes]` napsat na začátku dokumentu `\fontfam[Heros]`, což je rodina fontů se svobodnou licencí nahrazující komerční Helvetiku. Dostanete tento výsledek:

Můj první dokument

Zkouším napsat první text v TEXu. Tento odstavec musí být tak dlouhý, aby bylo vidět, že se rozlomil aspoň na dva řádky.

Jednotlivé odstavce oddělujeme od sebe prázdným řádkem. Prázdnými řádky vůbec nešetříme, protože zvyšují přehlednost zdrojového textu. Vyzkoušíme si nyní několik věcí.

- Budeme používat české „uvozovky“, které se liší od anglických. Uvědomíme si, že použití "těchto znaků" je úplně špatně!
- Rozlišujme mezi spojovníkem (je-li), pomlčkou ve větě – a dlouhou pomlčkou—ta se používá v anglických dokumentech.
- Zkusíme přepnout do **tučného písma**, nebo do *kurzívy*. Také vyzkoušíme `strojopis`.
- Vypravíme se na malou exkurzi do matematiky: $a^2 + b^2 = c^2$. Zjistíme, že číslo -1 je zde napsáno špatně (prokletý spojovník), zatímco správně má být −1.
- Protože % uvozuje komentář a $ přepíná do matematické sazby, musíme před ně napsat zpětné lomítko, chceme-li je dostat do dokumentu.

Závěr

To by pro začátek stačilo. Příkazem `\bye` ukončíme své pokusy.

[8] Ve všech formátech se používají stejné názvy převzaté většinou z Plain TEXu.

Je vidět, že kompletně celý dokument je nyní v bezpatkovém písmu Heros: titulky, kurzíva i matematická sazba. Do části vstupního textu, kde je obsah dokumentu, jsme nemuseli zasahovat. Podobně můžeme nastavit velikost odstavcové zarážky, mezery mezi odstavci, okraje, umístění a typografii nadpisů atd. Vše děláme odděleně na jiném místě, než je obsah dokumentu. Deklaraci vzhledu dokumentu můžeme také mít v jiném souboru, který je z hlavního souboru s obsahem dokumentu nejprve přečten příkazem `\input{soubor}`.

6 Ukázka stejného dokumentu v LaTeXu

Příkazem `lualatex dokument`[9] zkuste zpracovat tento soubor:

```
\documentclass[a4paper,12pt]{article}  % V LaTeXu povinné
\usepackage{polyglossia}               % umožňuje nastavení jazyka
\setdefaultlanguage{czech}             % české vzory dělení, atd.
\usepackage{fontspec}                  % umožňuje nastavení fontu
\setmainfont[Ligatures=TeX,
    Extension=.otf,
    UprightFont= *-regular,
    BoldFont=*-bold,
    ItalicFont=*-italic,
    BoldItalicFont=*-bolditalic
]{texgyretermes}                       % fontová rodina Termes
\usepackage{unicode-math}
\setmathfont{[texgyretermes-math]}     % rodina Termes pro matematiku

\usepackage{hyperref}                  % klikací věci
\usepackage{graphicx}                  % vkládání obrázků
\usepackage{amsmath}                   % AMS matematika
\usepackage{amsfonts}                  % definuje \mathbb

\newcommand\uv[1]{„#1"}                % uvozovky (nemám je na klávesnici)

\begin{document}

\title{Můj první dokument}
\date{}
\maketitle

Zkouším napsat první text v~\TeX u. Tento odstavec musí být
tak dlouhý, aby bylo vidět, že se rozlomil aspoň na dva řádky.

Jednotlivé odstavce oddělujeme od sebe prázdným řádkem. Prázdnými řádky
vůbec nešetříme, protože zvyšují přehlednost zdrojového textu.
Vyzkoušíme si nyní několik věcí.

\begin{itemize}
\item Budeme používat české \uv{uvozovky}, které se liší od ``anglických''.
      Uvědomíme si, že použití "těchto znaků" je úplně špatně!
\item Rozlišujme mezi spojovníkem (je-li), pomlčkou ve větě~---
      a dlouhou pomlčkou---ta se používá v~anglických dokumentech.
\item Zkusíme přepnout do \textbf{tučného písma}, nebo do
      \textit{kurzívy}. Také vyzkoušíme \ttexttt{strojopis}.
\item Vypravíme se na malou exkurzi do matematiky: $a^2 + b^2 = c^2$.
      Zjistíme, že číslo -1 je zde napsáno špatně (prokletý spojovník),
      zatímco správně má být $-1$.
```

[9] Všimněte si rozdílu: příkaz `luatex` spustí LuaTeX s formátem PlainTeX, zatímco `lualatex` spustí LuaTeX s formátem LaTeX.

```
\item Protože \% uvozuje komentář a \$ přepíná do matematické sazby, musíme
      před ně napsat zpětné lomítko, chceme-li je dostat do dokumentu.
\end{itemize}

\section*{Závěr}

To by pro začátek stačilo. Příkazem \verb`\end{document}` ukončíme
své pokusy.
\end{document}
```

Po zpracování dostanete zhruba stejný výsledek, jako byl předveden v předchozí sekci.

Následující poznámky se věnují specifickým záležitostem LaTeXu, protože základní vlastnosti TeXu již byly zmíněny v předchozí sekci a ty zde platí zcela stejně.

6.1 Povinný úvodní `\documentclass`

LaTeXový dokument musí začínat příkazem `\documentclass` za kterým je nepovinný parametr uvedený v hranatých závorkách (v ukázce je uveden formát papíru a velikost základního písma) a dále povinný parametr ve svorkách udávající výchozí styl dokumentu. Typicky se používá `article` nebo `report` pro psaní článků, `book` pro psaní knih. Ale takových základních stylů nabízí LaTeX více, každý z nich má svůj soubor maker s příponou `.cls`, který se při vykonání příkazu `\documentclass` přečte. V našem příkladě se tedy čte soubor `article.cls`.

6.2 LaTeXová prostředí

LaTeX pracuje s prostředími, která lze do sebe vnořovat. Tato prostředí jsou vždy vymezena dvojicí `\begin{nazev}...\end{nazev}` a každý dokument musí mít povinně aspoň hlavní prostředí `document`, ve kterém je schován celý dokument. V ukázce máme ještě vnořené prostředí `itemize`.

Před zahájením prostředí `document` je tzv. LaTeXová preambule. V ní se typicky pomocí čtení dodatečných balíčků maker příkazem `\usepackage` specifikují další vlastnosti LaTeXu a parametry sazby.

6.3 Vymezení jazyka dokumentu a fontů

V ukázce je nejprve zaveden balíček maker `polyglossia`, který definuje mimo jiné příkaz `\setdefaultlanguage`. Následně je tímto pčíkazem řečeno, že chceme česká dělení slov.

Dále je zaveden balíček maker `fontspec` umožňující čtení Unicode fontů a pomocí jeho příkazu `\setmainfont` se seznamem parametrů je zavedena rodina Termes. Potom je zaveden balíček `unicode-math` a rodina Termes je nastavena i pro matematickou sazbu. To celé dohromady například v OpTeXu zastoupí příkaz `\fontfam[Termes]`.

6.4 LaTeXové balíčky

LaTeX samotný bez zavedení dodatečných balíčků maker toho neumí mnoho. Byl totiž vyvinut v 80. letech minulého století a bylo rozhodnuto, že základ LaTeXu, neboli tzv. LaTeXové jádro bez dodatečně načtených balíčků, zůstane na úrovni nabídky funkcí, které uměl LaTeX v době svého vzniku. Tehdy například neexistovaly dokumenty s hypertextovými odkazy, takže pokud chceme mít PDF s hypertextovými odkazy, potřebujeme zavést další balíček `hyperref`. Tehdy se nedalo jednotným způsobem a snadno do dokumentu vkládat externí obrázky, chceme-li to, potřebujeme zavést další balíček `graphicx`. Tehdy se neumělo pracovat s barvami v dokumentu, potřebujeme-li to, musíme zavést balíček maker `xcolor`. Poznamenejme ještě, že to „`x`“ v názvu balíčku (někdy na konci a někdy na začátku) naznačuje, že v podpoře barev nebo vkládání obrázků se dříve používal jiný, dnes už zastaralý balíček, zatímco dnes existuje jeho nová vývojová větev, která se pomocí „`x`“ odlišuje od názvu původního balíčku.

LaTeXové jádro podporuje jen základní možnosti pro matematickou sazbu. Americká matematická společnost vymezila (původně v tzv. AMSTeXu) další potřebné konstrukty pro tvorbu matematických formulí. Ty jsou v LaTeXu k dispozici po zavedení balíčků `amsmath` a `amsfonts`.

Každé zavedení balíčku pomocí `\usepackage{nazev}` znamená, že se přečte soubor `nazev.sty` a případně z něj ještě mnohé další soubory maker. O tom se můžete přesvědčit v `.log` souboru.

Poznamenejme pro srovnání, že v OpTeXu žádný ze zde uvedených dodatečných balíčků zavádět nepotřebujeme, protože OpTeXové jádro umí vše potřebné pro tvorbu dokumentů běžně dnes užívaných.

Typická LaTeXová preambule obsahuje zavedení i desítek různých balíčků. Uživatelé si preambuli kopírují z dokumentu do dokumentu i mezi sebou a neustále ji obohacují o zavádění dalších a dalších balíčků. Takže se velmi často stává, že je v preambuli zavedeno plno balíčků, které nakonec v dokumentu nepotřebují. To nejenže zpomaluje zpracování, ale je to taky náchylnější k různým záhadným chybám například z důvodu nekompatibility balíčků, zavedení balíčků v nesprávném pořadí atd. Velmi silně tedy doporučuji vždy očistit preambuli na minimum a v žádném případě nezavádět balíček, jehož vlastnosti v dokumentu nepotřebujeme. V ukázce jsme se dopustili tohoto prohřešku, protože v našem dokumentu ve skutečnosti nepotřebujeme hypertextové odkazy, nevkládáme do dokumentu externí obrázek a i předvedená matematická formule je natolik jednoduchá, že konstrukty z AMS na ní nevyužijeme. Přesto jsem zavedení takových balíčků do ukázky zařadil, protože to jsou nejběžněji používané balíčky, aby byl LaTeX aspoň trochu schopen zpracovávat dokumenty s obvyklými současnými požadavky na vlastnosti dokumentů.

6.5 Definování vlastních maker

V preambuli si také uživatelé mohou definovat svá další makra. V ukázce je definice makra `\uv` na české uvozovky. Stejné makro jsme definovali už v sekci 5.8, jenže jinak. Tehdy to bylo pomocí TeX-primitivního příkazu `\def`, zatímco v nynější ukázce máme definci provedenou LaTeXovým makrem `\newcommand`. Všimněte si též odlišné syntaxe příkazu `\def` (parametr ve formě `#1` se explicitně uvede před tělem definice) a `\newcommand` (počet parametrů se uvede do hranaté závorky). Důležitá odlišnost také je, že makro `\newcommand` nejprve zkontroluje, zda definovaná řídicí sekvence už nemá náhodou nějaký význam v TeXu či v použitých makrech přidělen. Pokud ano, odmítne ji předefinovat a LaTeX končí chybou. Na druhé straně `\def` nic nekontroluje a sekvenci definuje nebo předefinuje.

Vysvětlíme si důvod popsané vlastnosti `\newcommand`. Interních příkazů TeXu a interních maker je obrovské množství (tisíce až desetitisíce) a nedá se předpokládat, že by je uživatel všechny znal a pamatoval si je. Příkazem `\def` by pak mohl nějakou interní věc nevědomky a snadno předefinovat a došlo by k vnitřnímu poškození maker a zpracování dokumentu by se zřejmě zhroutilo. Proto dělá `\newcommand` tu kontrolu.

V OpTeXu žádnou takovou kontrolu nepotřebujeme, protože veškeré interní názvy používají jiný jmenný prostor důsledně oddělený od uživatelského, takže uživatel si může definovat co chce a předefinuje si tak jen svá vlastní makra nebo řídicí sekvence, které sám používá. Ale to je jeho věc, třeba to tak chce.

LaTeX se snaží oddělit od uživatelského jmenného prostoru pomocné řídicí sekvence tím, že v jejich názvu použije znak `@`. Podíváte-li se do některých LaTeXových souborů maker, obvykle se na vás tyto znaky vyrojí. Ovšem toto oddělení uživatelského jmenného prostoru není vůbec uděláno důsledně.

6.6 Značky použité v úvodní ukázce

Titul v LaTeXu uděláme tak, že nejprve vymezíme potřebné údaje pomocí příkazů `\title`, `\date`, `\author` a poté tyto údaje naráz vytiskneme příkazem `\maketitle`.

Sekce se v LaTeXu zahajuje příkazem `\section` a má-li být bez čísla, je za ním hvězdička. Pak následuje parametr příkazu. V LaTeXu jsou parametry uživatelských příkazů téměř výhradně

vymezeny závorkami `{...}`, které v tomto případě nemají význam skupiny (TEXovou skupinu si případně otevře a zavře makro samo).

Všimněte si také, že místo přepínačů fontů `\bf`, `\it` a `\tt` (používaných v plain TEXu) jsou v LATEXu preferovány příkazy s parametrem: `\textbf`, `\textit` a `\texttt`. Ne, že by LATEX neznal i původní přepínače z PlainTEXu, ale v LATEXových manuálech není doporučeno je užívat. Když je nakonec LATEXový uživatel přece jen použije, pak to udělá občas bohužel špatně jako `\bf{text}` a pak se diví, že tučné písmo pokračuje i za slovem „text“. Je to tím, že `\bf` není makro s jedním parametrem ale přepínač jehož platnost končí na konci skupiny, ve které byl použit. To je sice více přirozené v TEXu samotném, ale méně přirozené pro LATEXové uživatele, kteří se učí používat výhradně makra s parametry.

6.7 LATEXový příkaz \\

Tento příkaz je v LATEXu poměrně nešťastně definován jako konec řádku uvnitř odstavce, přičemž odstavec pokračuje na dalším řádku. To bohužel svádí neznalce k používání tohoto příkazu, kdykoli chtějí mít v dokumentu nedokončený řádek. Ale každý normální dokument má nedokončené řádky výhradně jen na koncích odstavce. Tam ukončíte odstavec přirozeně prázdným řádkem. Příkaz \\ tam v žádném případě nepoužívejte. Používání příkazu \\ v odstavci (ať uvnitř nebo na jeho konci) vyjadřuje jen neznalost autora, že v typografii se dokument zásadně člení na odstavce s pevně definovanou a hlavně jednotnou formou. Příkaz \\ se ale hodí v kontextu tabulek, kde skutečně má význam ukončení řádku tabulky.

6.8 Kde hledat další informace

Počáteční vědomosti o LATEXu můžete získat z dokumentu The Not So Short Introduction to LATEX2e. Český překlad sice také existuje, ale je dnes už poměrně zastaralý, zatímco anglický originál je průběžně revidován a měněn podle nejnovějších poznatků.

Obecně informaci k LATEXovému balíčku nebo i jinému nástroji z TEXlive získáte zadáním příkazu `texdoc nazev`. Například po `texdoc hyperref` dostanete dokumentaci k balíčku `hyperref`.

7 Ukázka pro ConTEXt

Pro úplnost uvedeme ještě stejnou ukázku připravenou pro zpracování ConTEXtem.

```
\setuppapersize[A4]                  % formát papíru
\setuppagenumbering[location=footer] % stránková číslice bude dole
\mainlanguage[cz]                    % dělení slov podle českých pravidel
\setupbodyfont[termes,12pt]          % písmo Termes velikost 12pt

\setupindenting[yes, medium]

\setuphead[title][style=bfd, align=center]
\setuphead[section][style=bfb, number=no, after={\blank[small]}]

\setupitemize[1][packed][symbol=1,before={\blank[medium]},after={\blank[medium]}]
\setupitemize[2][packed][symbol=a,stopper=)]

\let\uv=\quotation

\starttext

\title{Můj první dokument}

Zkouším napsat první text v~\TeX u. Tento odstavec musí být
tak dlouhý, aby bylo vidět, že se rozlomil aspoň na dva řádky.
```

```
Jednotlivé odstavce oddělujeme od sebe prázdným řádkem. Prázdnými řádky
vůbec nešetříme, protože zvyšují přehlednost zdrojového textu.
Vyzkoušíme si nyní několik věcí.

\startitemize
\item Budeme používat české \uv{uvozovky}, které se liší od anglických.
      Uvědomíme si, že použití "těchto znaků" je úplně špatně!
\item Rozlišujme mezi spojovníkem (je-li), pomlčkou ve větě~---
      a dlouhou pomlčkou---ta se používá v~anglických dokumentech.
\item Zkusíme přepnout do {\bf tučného písma}, nebo do
      {\it kurzívy}. Také vyzkoušíme {\tt strojopis}.
\item Vypravíme se na malou exkurzi do matematiky: $a^2 + b^2 = c^2$.
      Zjistíme, že číslo -1 je zde napsáno špatně (prokletý spojovník),
      zatímco správně má být $-1$.
\item Protože \% uvozuje komentář a \$ přepíná do matematické sazby,
      musíme před ně napsat zpětné lomítko, chceme-li je dostat do dokumentu.
\stopitemize

\section{Závěr}

To by pro začátek stačilo. Příkazem \type{\stoptext} ukončíme své pokusy.

\stoptext
```

Po spuštění `context dokument` dostanete stejný výsledek jaký je ukázán v sekci 5.

ConTEXt má stejně jako LATEX hlavní prostředí a vnořená prostředí. Hlavní prostředí se vymezí dvojicí `\starttext` a `\stoptext`. Obecně se prostředí vymezují dvojicí `\startnazev` ... `\stopnazev`.

Typické pro ConTEXt je, že v preambuli nastavujete vzhled dokumentu sadou příkazů `\setupněco`, kde seznamem parametrů v hranaté závorce dáváte najevo, co si přejete. Například pokud bychom nenastavili `\setuppagenumbering` jak je uvedeno v ukázce, měli bychom stránkovou číslici nahoře. Pokud bychom nenastavili `\setupindenting`, měli bychom odstavce bez odstavcové zarážky.

Ke ConTEXtu existuje rozsáhlé množství dokumentace, najdete je na https://wiki.contextgarden.net.

8 Srovnání uvedených formátů

Nejstarší je PlainTEX, kterým se všechny ostatní formáty více či méně inspirovaly.

Druhý nejstarší je LATEX. Byl původně navržen skutečně jako „formát“, tj. Leslie Lamprort si zahrál na typografa a předpokládal, že ostatní uživatelé LATEXu se typografií nebudou zabývat a zaměří se jen na obsah dokumentů. Tj. budou to autoři. Nabídl čtyři připravené typografické styly `article`, `book`, `report` a `slides` a všechny byly založeny na tehdy jediné dostupné sadě fontů: Computer Modern. Uživatelé byli typicky autoři odborných publikací. Změna typografického nastavení se moc nepředpokládala.

Protože ale TEX dokáže dodatečně načíst i jiné fonty, změnit za chodu typografické nastavení a již definovaná makra, začaly postupně vznikat balíčky, které různým způsobem umožnily nastavit i jiný vzhled dokumentů než původní Lamportův. Ale není to uděláno s takovou jednotnou koncepcí, jako například příkazy `\setupněco` v ConTEXtu.

LATEX nejen umožňuje vkládat balíčky dalších maker pomocí `\usepackage`, ale dnes je vkládání takových balíčků v podstatě nutností, protože LATEXové jádro toho příliš mnoho neumí. Balíčků od různých autorů je ve veřejné síti (zejména na síti CTAN, kde najdete archiv všech TEXových věcí) obrovské množství. Jsou jich tisíce. Jakýkoli méně běžný požadavek na sazbu v LATEXu je z velké pravděpodobnosti vyřešen nějakým balíčkem. Takže běžná komunikace uživatelů LATEXu na internetu (dále jen Síť) vypadá následovně. Dotaz: „Chtěl bych udělat tohle, neumím to, jak se to dělá?“. Odpověď: „Použij balíček xyz“.

Číst zdroje těch balíčků a použít se z nich při tvorbě vlastních maker není snadné, protože věci jsou často zbytečně komplikovány snahou o univerzálnost nebo prostě jen tím, že autora nenapadlo, jak to udělat přehledněji a přímočařeji. Ne všichni autoři balíčků dokonale umějí TEX.

LATEX i ConTEXt se snaží odstínit uživatele od vnitřní logiky TEXu, což nemusí být dobrý nápad, protože nakonec se spustí TEX (jen s příslušnými makry) a TEX zejména bude reportovat případné chyby. Jednoduchým příkladem je typická chyba `undefined control sequence`, které nemusejí LATEXoví uživatelé rozumět, protože oni definují pomocí `\newcommand` „příkazy" a nikoli „řídicí sekvence", o kterých se v LATEXové dokumentaci nic nepíše. Mohou být zmateni i kvůli dalším odlišnostem v terminologii. Znalost TEX-primitivní úrovně se v LATEXové dokumentaci moc nepěstuje.

Snahu LATEXu o implementaci nové vrstvy nad TEXem si ukážeme na problematice nastavení velikosti odstavcové zarážky. TEX samotný disponuje primitivním registrem `\parindent` a jeho hodnota určuje velikost odstavcové zarážky. Chceme-li mít zarážku 15 bodů, napíšeme v TEXu přirozeně `\parindent=15pt` někam na začátek dokumentu. V LATEXovém manuálu se ale dočtete, že pro toto nastavení byste měli použít `\setlenght{\parindent}{15pt}`. Makro `\setlenght` s těmito dvěma parametry neudělá na pozadí nic jiného než přiřazení `\parindent=15pt`, ale máme zde pro uživatele jiný způsob vyjadřování. Primitivní konstrukce se v LATEXu moc nepoužívají, i když tam samozřejmě taky fungují.

ConTEXt dělá odstínění od TEX-primitivních konstrukcí ještě důsledněji. V něm použijete příkaz `\setupindenting[yes, 15pt]`, což taky nakonec udělá `\parindent=15pt`, ale uživatel už ani netuší, že TEX používá nějaký primitivní registr `\parindent` na nastavení této věci.

LATEXoví vývojáři pokročili v poslední době v odstínění původních TEXových konstrukcí ještě dál. Vyvinuli novou vrstvu jazyka nad makrojazykem TEXu, nazvali ji `Expl3`. Syntaxe psaní maker v tomto jazyku je zcela odlišná od TEX-primitivní filosofie a mnohým, kteří umějí dobře psát svá makra na TEX-primitivní úrovni, může tato nová syntaxe připadat až absurdní a překomplikovaná. Přitom nakonec nejde o nic jiného, než že se tato nezvyklá syntaxe převede pomocí maker až na TEX-primitivní úroveň a na té se vykoná. LATEXoví vývojáři nyní předpokládají, že tuto novou vrstvu budou používat autoři nových makrobalíčků pro LATEX. Trasování chyb, případně hledání, jak věci fungují, se ovšem stává skoro nemožným. Nicméně se tou syntaxí snaží autoři Expl3 přiblížit způsobu uvažování současných programátorů[10], možná to tedy bude někomu vyhovovat a pro programování maker shledá, že to je užitečný nástroj.

OpTEX naopak razí filosofii nedělat věci jinak než primitivím TEXovým způsobem, nevymýšlet žádné lexikální vrstvy nad tím. Takže odstavcovou zarážku nastavujete přímočaře pomocí `\parindent=15pt`. OpTEX předpokládá, že uživatel zvládá do jisté míry samotný TEX a bude tedy používat přímo TEXové konstrukce a TEX-primitivní příkazy, chce-li si vyřešit nějaký typografický úkol. K získání přehledu o tom, jak funguje TEX, slouží dokument TEX in a Nutshell, kde je fungování TEXu shrnuto na 26 stránkách.

Shrňme si to. Předpokládejme, že chcete vyřešit cosi, o čem se v uživatelském manuálu použitého formátu explicitně nepíše. V PlainTEXu či v OpTEXu to uděláte za pomocí TEX-primitivních příkazů a znalostí TEXu. Uživatelé LATEXu vrhnou dotaz do Sítě a dozví se, že mají použít balíček maker xyz. Nadále nebudu rozumět tomu, jak to je interně uděláno. Uživatelé ConTEXtu v rozsáhlé dokumentaci asi nakonec najdou ten správný `\setupněco` s těmi správnými parametry.

Srovnejme ještě používané formáty z pohledu jejich dokumentace. PlainTEX je kompletně dokumentován v knize TEXbook, kde je též úplná dokumentace k TEXu samotném. Ovšem není to zrovna snadné čtení pro začátečníka, kterého text zavede asi hend na začátku do slepých uliček, protože se tam třeba podrobně řeší, že písmeno ě zapíšeme pomocí `\v e`, že to je makro postavené na primitivním příkazu `\accent`, který sestavuje znak z jednotlivých segmentů s různými kódy jednoho nebo více fontů. To v rozšířeních TEXu podporujících Unicode opravdu dnes nepotřebujeme používat, tam máme na vstupu `ě`, což odpovídá nějakému kódu

[10] Včetně používání termínů „funkce" nebo „proměnná", třebaže to vše jsou ve skutečnosti TEXová makra.

a ve fontu pod stejným kódem máme taky ě. TEX je také úplně dokumentován pro české čtenáře v knize TEXbook naruby. Ale nenajdete tam nic, co umějí dnešní TEXová rozšíření, protože kniha je z roku 2001. Na druhé straně stručné shrnutí TEX in a Nutshell zahrnuje i nejdůležitější vlastnosti současných rozšíření TEXu.

OpTEX je uživatelsky dokumentován na necelých třiceti stránkách. Zbylý text dokumentace, která má dohromady 210 stran, je technický rozbor všech použitých maker OpTEXu s vysvětlením, jak to funguje. Umíte-li aspoň rámcově primitivní TEX, porozumíte tomu a máte možnost se v této technické části dokumentace inspirovat a začít si třeba dělat vlastní sofistikovaná makra.

Rozhodnete-li se pro LATEX, můžete začít číst The Not So Short Introduction to LATEX2e. Píšou tam v podtitulku, že to dáte za 139 minut, což by znamenalo přečíst jednu stránku za méně než minutu, protože dohromady to má 155 stran. Dále ke každému použitému balíčku bývá dokumentace, což dohromady dává desetitisíce stran různých dokumentací od různých autorů a různého stáří, tj. různě koncipovaných. Začátečník se v tom obvykle utopí a pak hledá pomoc v šablonách (hotových LATEXových dokumentech), které jsou samy o sobě napsány v rozličné kvalitě. Velmi dobrá je ovšem dokumentace k LATEXu na Overleaf. Na Síti najdete k LATEXu plno rad, návodů a příkladů nejrůznějšího stáří a kvality. Ne vždy je to k dobru věci a začátečník se v tom množství informaci spíše ztratí. LATEX tady totiž je už skoro 40 let.

ConTEX má veškerou dokumentaci na jediném místě, ovšem je jí opravdu velké množství.

9 Diagnostika chyb

TEX při své činnosti vytváří soubor s příponou `log`, kam zapisuje protokol o zpracování. Míra jeho podrobnosti závisí na dalším nastavení. V každém případě tam najdeme záznamy o chybách a různá varování.

Představme si, že jsme udělali překlep v názvu nějaké řídicí sekvence a chceme tedy po TEXu, aby zpracoval řídicí sekvenci, která není v TEXu definována. Třeba místo správného `\LaTeX` napíšeme omylem `\latex` (s malými písmeny).

Spustíme-li TEX z příkazového řádku (třeba `luatex dokument` nebo `optex dokument`), pak se vybraná část protokolu zpracování zobrazuje i na terminál a TEX se zastaví při první chybě. Na terminálu to vypadá takto:

```
This is LuaTeX, Version 1.15.0 (TeX Live 2022)
 restricted system commands enabled.
(./pok.tex This is OpTeX (Olsak's Plain TeX), version <1.08 Aug 2022>

! Undefined control sequence.
l.13 Taky v \latex
               u je možno občas něco vytvořit.
?
```

Na prompt otazníku lze reagovat klávesou `Enter` a zpracování bude pokračovat. Nebo napište `x Enter` a zpracování ukončíte. Další možnosti zjistíte zadáním `? Enter`, ale tyto možnosti už nejsou příliš obvyklé.

Začátečníci opakovaně kladou dotazy na Síti k chybě `undefined control sequence`, přitom bohužel často rozbourají formátování toho chybového hlášení nebo uvedou jen ten jeden řádek s oznámením chyby, ale nic z toho, co je pod tím. Přitom v chybovém hlášení samotném se nedozvíte, jaká to je ta nedefinovaná řídicí sekvence, to zjistíte na konci dalšího řádku, který přepisuje buď řádek ze vstupního souboru nebo data z nějakého makra, které se zrovna v dané chvíli rozvinulo (expandovalo) a TEX jeho obsah postupně vykonává. Důležité je, že ten řádek nebo obsah makra je rozdělen vedví a na konci první jeho části je ona nedefinovaná řídicí sekvence. To začátečníci bohužel často nevědí a reportují pak chybové hlášení neúplné.

Je třeba také zdůraznit, že pokračování ve zpracování po objevení první chyby obvykle nemá smysl. Třebaže se TEX snaží podle typu chyby nějak z toho vybruslit (a v hlášení o chybě většinou také naznačuje jak), skoro jistě se nevyhneme zavlečeným chybám. Bohužel, některá grafická uživatelská rozhraní tuto skutečnost moc nerespektují. Spouštějí na pozadí TEX s přepínačem, který způsobí, že po reportování chyby se nezastaví, ale pokračuje automaticky dále. Pak toto uživatelské rozhraní vypíše extrakcí z `log` souboru celou řadu chybových hlášení, přičemž je třeba vědět, že jen to první je podstatné a ty ostatní mohou být způsobeny neúspěšným pochopením celého kontextu, co autor vlastně měl na mysli. Je pak často zbytečné o následných chybách přemýšlet.

Uživatelská rozhraní často zatajují celkový výpis chyby a zobrazí jen název chyby. To už víme, že to je špatně, protože třeba z názvu chyby `undefined control sequence` se opravdu nedozvíme, o jakou řídicí sekvenci se jedná. Vždy tedy pátrejte po tom, kde ve vámi užívaném pracovním prostředí máte možnost zjistit úplný výpis `log` souboru. Bez něj se neobejdete.

TEX při reportu chyb používá svou terminologii, což rovněž může některé zejména LATEXové uživatele zmást. Pokud například v LATEXu děláte tabulku, dozvíte se v manuálu, že jednotlivé řádky se ukončují pomocí `\\`. Když na nějaký `\\` zapomenete, TEX reportuje chybu `missing \cr inserted`. O příkazu `\cr` ale žádný LATEXový manuál nic nepíše. To je totiž TEX-primitivní příkaz na ukončení řádků v tabulce, který je v LATEXu interně spuštěn při použití příkazu `\\` v tabulce. LATEXovým uživatelům se tedy chybová hlášení mohou zdát příliš mysteriózní. Tady se ukazuje, že se hodně vyplatí znát TEX samotný, protože chyby nakonec reportuje TEX a používá k tomu svou terminologii.

Kromě chyb najdete na terminálu i v `log` souboru někdy nějaká varovaní. Nejčastěji je to `Underfull \hbox` s uvedením chybovosti (badness). Toto hlášení říká, že bylo potřeba na mezery mezi slovy vynaložit větší „násilí“ k jejich roztažení, než by odpovídalo běžným estetickým požadavkům. Intenzita toho násilí je vyjádřena hodnotou badness. Stává se to obvykle při požadavku na odstavec do bloku, kdy je třeba ten blok vyrovnat správnou úpravou mezer mezi slovy. Hodnoty badness menší než 10 000 se dají akceptovat, ovšem někdo s vyššími estetickými nároky sleduje i takové hodnoty a umí se s tím vypořádat. Jak se to dělá je ale mimo rozsah tohoto základního textu. Hodnoty badness rovny 10 000 (ony nikdy nejsou větší, to je mezní strop pro badness) už často znamenají závažnější problém. Kromě toho se mohou vyskytovat přetečené boxy `Overfull \hbox`, což je také závažný problém, protože to znamená, že část textu je vystrčena ze sazby do okraje o určitou velikost, která je v hlášení reportována.

Makra (použité formáty) mohou přidávat další varování i chybová hlášení. Typická jsou varovaní o nekompletních odkazech, kdy existuje odkaz na nějaký cíl, ale chybí v dokumentu ten cíl. Je-li cíl v dokumentu později než odkaz, pak makra použitého formátu ukládají informace o takových situacích do pracovních souborů a při dalším spouštění TEXu je přečtou. Takže při prvním spuštění TEXu můžete mít plno varování o neexistujících cílech a je tam doplněno, že máte zkusit spustit TEX ještě jednou. Když to provedete, TEX vytěží z pracovního souboru z předchozího běhu potřebné informace a už si třeba tolik nestěžuje.

Úroveň informací o zpracování v `log` souboru se dá zvýšit pomocí nastavení různých `\tracing` registrů TEXu na kladnou hodnotu. Například po `\tracingmacros=3` vidíme v `log` souboru informace o všech expanzích všech maker až do TEX-primitivní úrovně. A při použití `\loggingall` máme v `log` souboru kompletní zprávu o činnosti všech vnitřních algoritmů TEXu. Bohužel, takový `log` soubor bývá příliš rozsáhlý. Je tedy dobré dát `\loggingall` jen před místo v dokumentu, kde chceme zjistit podrobnější informace o sazbě. A schováme-li `\loggingall` do skupiny, po jejím ukončení podrobné výpisy ustanou.

Nevíte-li si s nějakým problémem rady, stojí za to si věc zálohovat a pak se snažit dokument zmenšit tak, aby stále byla patrná ta chyba. Takže smažete kusy textu, které chybu nezpůsobují, umažete volání některých makrobalíků, které pro demonstraci chyby nejsou důležité. Při tomto zmenšování dokumentu se často stane, že po odmazání další věci začne sazba fungovat a tím se Vám podaří zjistit příčinu chyby. Pokud přesto zůstává příčina chyby utajena, a máte přitom po těchto pokusech připraven skutečně tzv. *minimální dokument*, pak se s tímto příkladem můžete obrátit na diskusní fóra. Důležité je, že ostatní uživatelé Sítě mají možnost si Váš minimální

dokument přetáhnout myší do svého počítače a začít s tím experimentovat. Bez minimálního dokumentu na Síť s žádostí o radu raději vůbec nechoďte. TEX je natolik citlivý na úplně všechny vstupy, že jen vágní popis chyby naprosto nestačí. Pokud si držíte palec, že možná někdo bude vědět, protože se setkal s podobnou chybou, tak Vám skoro jistě dříve ten palec upadne, než se dočkáte kloudné odpovědi. Takže pamatujte: minimální příklad demonstrující problém je bezpodmínečně nutný. Také je potřeba v dotazu připojit, jakým formátem (příkazem) dokument zpracováváte. Tj. zda to je `luatex` nebo `lualatex` nebo `optex` nebo cokoli dalšího.

10 Zastaralé technologie a postupy

TEX byl za dobu své 40leté existence používán nejrůznějšími způsoby, které odpovídaly technologickým možnostem příslušné doby. Podpora těchto postupů je typicky v TEXových distribucích stále přítomna. Navíc na Síti najdete už naveky přítomné obrovské množství rad a návodů opírajících se o tyto zastaralé metody. Začátečník se v moři takových informací může utopit a netuší, čím by se už nemusel zabývat a čeho se při přípravě nových dokumentů vyvarovat. Proto jsem přidal na závěr tuto sekci. Uvádí zastaralé metody, kterým bychom se dnes měli vyhnout. Cokoli je zde zmíněno, je podmíněno slovy: *dnes takto ne*. Dnes preferujeme TEXová rozšíření spolupracující s moderními Unicode fonty a s přímým výstupem do PDF formátu. Jakékoli odbočení z těchto možností znamená utopit se ve starých komplikovaných postupech. Ty v této sekci jen částečně naznačím bez hlubšího technického popisu, aby bylo zřejmé, čeho se vyvarovat.

Pokud některou ze zde uvedených metod přesto používáte, pak určitě pro to máte pádné důvody, které se neopírají jen o to, že jste na Síti našli nějaký zastaralý návod tuto metodu zmiňující. Nebo jen proto, že vám to poradil nějaký uživatel, který viděl TEX z jedoucího vlaku naposledy před dvaceti lety.

10.1 Výstupní fomát DVI

TEX je výrazně starší než dnes používaný formát PDF. Je také starší než v devadesátých letech hojně používaný jazyk pro popis tiskového materiálu PostScript. Pro TEX byl původně navržen výstupní formát DVI[11]. Na tento binární formát navazovala v TEXové distribuci sada programů schopných tento formát číst a zobrazit dokument na nejrůznějších zařízeních: promítnout na obrazovce nebo vytisknout na všelijakých tiskárnách. Každá tiskárna měla svůj jazyk, kterým komunikovala s počítačem a měla v TEXové distribuci svůj program zvaný DVI-ovladač, který četl DVI, tj. na zařízení nezávislý výstup z TEXu, a komunikoval s konkrétní tiskárnou pomocí na zařízení závislých pokynů pro tuto tiskárnu. Tím došlo k tisku dokumentu.

Jakmile se v devadesátých letech objevil jazyk PostScript, byl vytvořen konvertor z DVI do PostScriptu jménem `dvips`. PostScriptový soubor už bylo možné přímo bez dalších úprav poslat na tiskárnu nebo jiné zařízení vybavené PostSriptovým RIPem. Pokud levnější tiskárna nebyla tímto RIPem vybavena, dal se použít volně šířený Ghostscript, který umí interpretovat PostScript a je vybaven na své výstupní straně mimo jiné ovladači běžných levných tiskáren. Tím se redukoval počet potřebných DVI ovladačů na dva: ovladač na prohlížení dokumentu na obrazovce a dále `dvips` pro tisk.

Poté se objevil formát PDF, který z profesionálních DTP studií postupně vytlačil PostScript. K tomu byl vyvinut konvertor z DVI do PDF `dvipdf`[12]. Navíc Ghostscript dokáže při interpretaci PostScriptu také vytvořit PDF. Byl k tomu účelu připraven příkaz `ps2pdf`. K vytvoření PDF se tedy začaly používat cesty TEX → DVI → `dvipdf` → PDF nebo dokonce TEX → DVI → `dvips` → PostScript → `ps2pdf` → PDF. Dnes takto ne.

[11] DVI je zkratka DeVice Independent a znamená na zařízení nezavislý.
[12] Později vznikly další generace tohoto programu `dvipdfm`, `dvipdfmx`, `xdvipdfmx`.

10.2 Bitmapové fonty

V době zrodu TEXu neexistovaly vůbec žádné fonty vhodné pro profesionální typografii. Autor TEXu vytvořil zřejmě první implementaci fontové rodiny pro počítač ve vektorovém popisu (byla to rodina Computer Modern) a vyvinul k tomu také rastrovací program Metafont. Nejprve bylo potřeba vytvořit rastry (bitmapy) potřebných fontů tímto programem pro konkrétní zařízení[13] a následně tyto bitmapové fonty četl DVI ovladač a posílal je tiskárně pro potřebu tisku.

Později byla do pdfTEXu a do DVI ovladačů přidána schopnost vyvolat si program Metafont podle potřeby automaticky, když se ukázalo, že v počítači není rastr fontu pro potřebné rozlišení zatím k dispozici. Tato vlastnost zůstala v programech pdfTEX a XƎTEX zabudována dodnes a prakticky to znamená, že pokud uděláte překlep v názvu fontu nebo potřebný font opravdu nemáte na počítači, probudí se k životu TEXoví veleještěři zvaní `mktexmf` a Metafont z doby počítačového diluvia a začnou se snažit vyrábět bitmapové fonty.

Dnes se pracuje výhradně s fonty ve vektorovém popisu. V této podobě se vkládají do výstupního PDF a teprve prohlížeč nebo tiskárna provádí jejich rastrování do potřebných bitmap podle požadovaného rozlišení a technických možností použitého zařízení. Uživatel si tuto skutečnost ani neuvědomí. Je-li v PDF zaveden přímo bitmapový font, výsledek dopadne zcela neuspokojivě, takže konverzi fontů do bitmapové podoby už v počítači (jak bylo dříve obvyklé) nechceme a potřebujeme se ji za všech okolností velkým obloukem vyhnout.

V PostScriptové éře už bylo možné posílat PostScriptové tiskárně vektorové popisy fontů. K tomu účelu byl program `dvips` vybaven možností konfigurovat takzvané mapování fontů a převzít pak přímo vektorový popis fontů, tehdy ve formátu `pfb`, a zařadit jej do výstupního PostScriptového souboru určeného pro tisk. Taková konfigurace je další komplikace navíc, protože to vyžaduje udržování mapovacích pravidel konzistentních se skutečně dostupnými fonty. Dnes takto ne.

10.3 8bitové fonty

Tímto pojmem máme na mysli fonty, které mohly najednou obsahovat v konkrétním uspořádání (tzv. kódování) maximálně 256 znaků. Mluvíme o osmi bitech proto, že pořadová čísla použitých znaků (jejich kódy) bylo možné uložit do 8bitového registru ($2^8 = 256$). TEX prapůvodně dokázal mít ve fontu jen 128 znaků, pak od roku 1989 přešel na 256 znaků. To vedlo k různým nápadům, jaké znaky do fontu nad původní ASCII tabulku[14] zahrnout a jak je uspořádat. Vznikala různá vzájemně nekompatibilní 8bitová kódování. Například v roce 1992 bylo na TEXové konferenci v Corku dohodnuto tzv. T1 kódování zahrnující znaky pokud možno všech evropských jazyků píšících latinkou s případnými akcenty. Tam patří i čeština. Kuriozitou tohoto kódování bylo, že bylo nekompatibilní se všemi ostatními standardy a navíc toto evropské kódování neobsahovalo znak Euro, protože pochopitelně takový znak v roce 1992 nebyl ještě znám. Přesto bylo toto kódování T1 v TEXu hojně používáno.

Pouze 256 znaků v jediném fontu se jevilo jako stále větší omezení a vznikaly přidružené (companion) fonty obsahující další znaky, třeba zmíněné Euro. Také různá vzájemně nekompatibilní kódování přinášela jen potíže.

PostScriptové fonty `pfb` umožňovaly zahrnout vektorové popisy libovolného množství znaků (není zde tedy omezení na 256) a dalo se k těm znakům přistupovat podle jejich jména. Nicméně systémy používající tyto fonty (včetně TEXu) toho neuměly využít a dokázaly si v jednom okamžiku vždy z fontu vybrat maximálně 256prvkovou podmnožinu a s ní pracovat. Takže jeden font mohl být do dokumentu zaveden v různých instancích, každá z nich byla specifikována tím fontem samotným a uspořádaným seznamem 256 znaků vybraných

[13] Uvědomíme si, že každé zařízení mohlo pracovat s jiným vnitřním technickým rozlišením, takže potřebovalo jiné rastry téhož fontu.

[14] Zcela původní TEXové fonty dodržovaly ASCII kódování navíc jen částečně, občas se autor fontu rozhodl na nějakou pozici umístit zcela jiný znak. Například místo < byl znak ¡, protože v textové sazbě se nepředpokládalo použití znaku < a v matematické sazbě se správný znak vytiskl ze speciálního matematického fontu.

z tohoto fontu (takzvané `enc` soubory). Aby toho nebylo málo, TEX disponoval ještě metodou virtuálních fontů, které umožňovaly překódování fontů pomocí speciálních binárních skriptů nad rámec použitých `enc` souborů a také to umožňovalo sestavovat kompozitní znaky z jednotlivých částí. Nainstalovat pro TEX správně font pak byla práce pro školeného specialistu, který musel vědět, jak vygenerovat a kam a proč dát `vf` soubory s virtuálními skripty, `enc` soubory s kódováními, `pfb` soubory s vektorovými popisy znaků, `map` soubory s konfigurací pro `dvips` nebo pdfTEX a `tfm` soubory s metrickými údaji fontu pro TEX. Nakonec ještě bylo třeba vytvořit a instalovat `fd` soubory se speciálními makry pro podporu 8bitových fontů v LATEXu.

Další komplikace během PostScriptového období vyplývaly z toho, že se dvě firmy Microsoft a Adobe nedokázaly dohodnout na společném formátu fontů takže se pracovalo (lidově řečeno) s fonty pro MAC nebo se zcela jinými fonty pro Windows, které nebyly kompatibilní s dříve jmenovanými. K dohodě mezi těmito firmami došlo až koncem devadesátých let a společný formát fontů byl nazván OpenType (přípona `otf`) a začal konečně podporovat Unicode.

Dnes stačí umístit OpenType font buď do aktuálního adresáře nebo do určeného místa v TEXové distribuci nebo jej instalovat do operačního systému. Pak jej máte přímo přístupný v rozšířeních XETEX a LuaTEX pomocí TEX-primitivního příkazu `\font` a na komplikace z PostScriptového období můžete s radostí zapomenout. Kvalitní font typicky obsahuje veškeré potřebné znaky, ke kterým přistupujete přirozeně přímo díky tomu, že XETEX i LuaTEX umějí pracovat v Unicode.

10.4 Další TEXové formáty

V sekci 3 jsou uvedeny nejběžnější formáty TEXu, neboli výchozí sady maker. Během dlouhého vývoje TEXu jich postupně vzniklo další nepřeberné množství. V konfiguracích TEXových distribucí jich najdete desítky. Nejčastěji to jsou formáty specializující se na některé jazyky, protože výchozí formát PlainTEX (podporující jen ASCII znaky a sestavování akcentovaných znaků pomocí TEXových příkazů) byl pro další jazyky nepoužitelný.

Pro češtinu a slovenštinu vznikl v roce 1992 CSplain, který místo původních Computer Modern fontů zaváděl do své paměti tzv. CSfonty. Ty byly 8bitové v kódování podle standardu ISO-8859-2 v horní části tabulky a podle Computer Modern fontů ve spodní části tabulky. Toto specifické kódování je v TEXu označováno IL2, jeho rozšířená verze pak má označení XL2. CSplain měl svůj výstup původně směrovaný do DVI (příkaz `csplain`) a později též do PDF (příkaz `pdfcsplain` opírající se o pdfTEX). Zhruba od roku 2000 má CSplain aktivován na vstupní straně konvertor EncTEX, který převádí Unicode vstup do IL2 kódování. Pokud příslušný Unicode znak není v IL2 podporován, konvertor mohl vytvořit TEXovou sekvenci, která se dala definovat podle potřeby, aby se příslušný znak nějakým vnitřním postupem (lokální změnou fontu, sesazením ze segmentů atd.) vytisknul. Vidíme, že to byla pouze „z nouze ctnost“ přinášející plno dalších potíží. Byla způsobena faktem, že CSplain pracuje s rozšířením pdfTEX a ten podporuje jen 8bitové fonty.

V kódování IL2 byly pro TEX a speciálně pro CSplain připraveny i základní PostScriptové fonty, takže tento formát podporoval kromě CSfontů také celou řadu dalších rodin fontů. Ovšem hlavní TEXový vývoj v této PostScriptové době směřoval na vytváření 8bitové podpory fontů podle T1 kódování. Pozůstatky po tomto snažení najdeme v rozsáhlé míře stále obsaženy v TEXových distribucích. Obojí kódování IL2 a T1 bychom dnes už měli odsunout na smetiště dějin a nezabývat se tím.

Vývoj LATEXu má také bohatou historii. Do roku 1994 se používal LATEX který končil na verzi 2.09 a dokumenty z té doby se poznají tak, že místo úvodního `\documentclass` mají `\documentsyle`. Následně se začal používat LATEX s označením 2ε a ten se používá dosud. Byly připraveny příkazy `latex` a `pdflatex`, které spouštěly originální TEX, později pdfTEX. Pro tato rozšíření pak ukázkový dokument vypadá takto:

```
\documentclass[a4paper,12pt]{article}  % V LaTeXu povinné
\usepackage[utf8]{inputenc}            % nastavení vstupního kódování
\usepackage[T1]{fontenc}               % nastavení 8bitového kódování fontů
\usepackage{times}                     % rodina Times, ve skutečnosti Termes
\usepackage[czech]{babel}              % nastavení českého jazyka

\usepackage{hyperref}                  % klikací věci
\usepackage{graphicx}                  % vkládání obrázků
\usepackage{amsmath}                   % AMS matematika
\usepackage{amsfonts}                  % definuje \mathbb

\begin{document}

\title{Můj první dokument}
\date{}
\maketitle
...
\end{document}
```

Vidíme tedy, že místo místo balíčku `fontspec` pro zavádění Unicode fontů bylo potřeba specifikovat 8bitové kódování fontů T1 balíčkem `fontenc`. Také bylo potřeba uvést v parametru balíčku `inputenc`, jaké je použito kódování vstupních souborů. Kromě nejběžnějšího kódování Unicode v textových souborech[15] existovala nabídka ještě celé řady dalších 8bitových kódování podle nejrůznějších standardů. Dvojice parametrů balíčků `inputenc` a `fontenc` pak vymezila kódovací proces, jak má TEX změnit vstupní kódování souborů na kódování použitých fontů. Tento vnitřní algoritmus používal ještě mezikrok zvaný LICR (LATEX internal character representation), který poněkud nešťastně používal krátké TEXové sekvence jako třeba `\v` pro háček. Ty se pak nedaly v dokumentu použít pro jiné účely.

Výběr rodiny fontů Times byl proveden speciálním balíčkem `times`, který uživatele odstínil od faktu, že názvy souborů s fonty byly poněkud obskurní: `ptmr8t`, `ptmri8t`, `ptmri8t`[16] a že byly většinou mapovány na nějakou svobodnou alternativu komerčního Times. Typicky na NimbusRomanNo9L (dodávaný svobodně s Ghostscriptem). Konečně místo balíčku `polyglossia` se dříve používal balíček `babel`, který se dnes při použití Unicode fontů už nedoporučuje.

Vzhledem k tomu, že toto řešení používá 8bitové fonty, víme, že se mu máme dnes vyhnout a použít místo příkazů `latex` nebo `pdflatex` rozšíření `xelatex` nebo `lualatex` podporující Unicode. A že nemáme používat balíčky `inputenc` a `babel` a hlavně ne balíček `fontenc` specializující se na 8bitová kódování.

V devadesátých letech existovala ještě alternativa LATEXu zvaná CSLATEX. Ta umožňovala nastavit kódování IL2 i pro vzory dělení slov, což hlavní vývojová větev LATEXu nepodporovala. Místo balíčku `babel` s parametrem `[czech]` se v této verzi formátu používal přímo balíček `czech`, tedy `\usepackage{czech}`. I toto řešení dnes ale patří jen do vzpomínek pamětníků a je už kolem deseti let označeno v TEXových distribucích jako zastaralé, třebaže stále je možné je spustit příkazy `cslatex` nebo `pdfcslatex`. Nedělejte to.

[15] V textových souborech je Unicode realizován pomocí standardu `utf8`, proto jsme tuto zkratku použili v ukázce.
[16] Důvod: staré operační systémy nedovolovaly názvy souborů delší než 8 znaků.