EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Práce v systému LaTeX pro mírně pokročilé

Učební texty k semináři

Autoři:

Ing. Pavel Haluza (Mendelova univerzita v Brně)

Datum:

31.1.2012

Centrum pro rozvoj výzkumu pokročilých řídicích a senzorických technologií
CZ.1.07/2.3.00/09.0031

TENTO STUDIJNÍ MATERIÁL JE SPOLUFINANCOVÁN EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY

Obsah

1 Úvod 3

2 Možnosti systému LaTeX a jeho následníků 4

3 Grafika v dokumentech 8

4 Prezentace 10

5 Principy strukturního značkování 12

6 Realizace strukturního značkování v systémech typu TeX 14

7 Tvorba nových příkazů 17

8 Sázecí styly, balíčky a třídy 29

9 Zpracování výstupů z jiných zdrojů 31

10 Návrh dokumentního typu a odpovídajícího stylu 34

11 Existující styly a jejich rozbor 37

12 Literatura 38

Kapitola 1

Úvod

Práce v systému LaTeX představuje jednu z možných alternativ tvorby dokumentů pomocí počítače. Prostředky, které tento systém nabízí, jsou určeny pro širokou škálu různých dokumentních typů, přičemž velký přínos lze spatřovat u textů odborných, a to jak ve formě článků, tak i ucelených publikací.

Zabýváme-li se technologií tvorby dokumentů, zcela bezpochyby nás bude zajímat i efektivnost takového procesu. Tu můžeme vyjádřit například jako množství času nutné k vytvoření konkrétního dokumentu. Pokud bychom tentýž výstup řešili v různých programových systémech, můžeme pak na základě spotřeby času poměrně přesně určit, jak efektivní tato činnost je.

Na efektivitu však má podstatný vliv úroveň ovládnutí daného programu. Jednu a tutéž věc lze často provést několika způsoby, které se od sebe diametrálně liší pracností, kvalitou výstupu a možnostmi případných dalších úprav. Bohužel ne všechny příručky k různým programům sledují všechny zmíněné faktory, spíše nechávají na uživateli, kterou cestu si vybere.

Budeme se tedy v tomto textu snažit především o takové prvky, které efektivnost tvorby dokumentů podporují. Důležité je přitom vzít v úvahu, že k jejich zvládnutí je potřebná určitá časová investice, ale z praxe víme, že se taková investice bohatě vyplatí. Budeme se také zabývat kvalitou výstupu, která je podmíněna alespoň rámcovou znalostí typografických pravidel.

Učební text je orientován na dvě oblasti – první z nich je grafika a možnosti prezentace, druhá se věnuje tvorbě sázecích stylů a strukturnímu značkování dokumentů. Obsahem každé části jsou výchozí požadavky kladené na příslušné prvky v dokumentech vycházející z typografických pravidel, dále přehled vybraných principů a možností a ilustrační příklady, na kterých lze uvedené prvky vyzkoušet v praxi.

Pro čtení tohoto textu se předpokládá alespoň základní znalost práce se systémem LaTeX, neboť tento kurs navazuje na první část Základy sazby dokumentů v systému LaTeX. Pojmy, na které se text odkazuje, však lze nalézt v literatuře a v internetových zdrojích, výběr možností je uveden v závěrečném seznamu literatury.

Kapitola 2

Možnosti systému LaTeX a jeho následníků

Vývoj programového vybavení, které lze využít pro tvorbu dokumentů, jde stále kupředu. Pokud se podíváme na oblast systémů postavených na principu TeXu, lze zde vysledovat dva zcela zásadní počiny, které výrazným způsobem zvýšily možnosti využití a rozšířily okruh aplikací.

Prvním z nich je překladač TeXového zdrojového textu do PDF – pdfTeX. Výstupy ve formátu PDF dnes tvoří bezesporu majoritní způsob publikace dokumentů, slouží jako vstupy pro tiskové stroje, jako vhodný formát do heterogenního internetového prostředí i jako formát vhodný pro archivaci. Do překladače byly navíc přidány některé drobnější vymoženosti, jejichž výčet není v tomto místě podstatný. Dnešní situace je tedy taková, že překladač do formátu PDF tvoří jádro téměř všech systémů tohoto typu.

Druhým zásadním počinem je vytvoření „nadstavby“ XeTeX. Jak už napovídá název, původní záměr byla sazba zleva doprava i zprava doleva a práce s exotickými jazyky (arabština apod.), ale efekt je použitelný i pro zcela běžnou práci – jedná se o rozšíření možností práce se vstupem v kódování UTF-8 a o možnost práce s různými písmy. A jako je XeTeX rozšířením základního TeXu, je pak XeLaTeX rozšířením LaTeXu. Výrazně se zde zjednodušuje psaní speciálních znaků a vícejazyčných textů a diametrálně odlišně a jednodušeji lze použít jakýkoliv font, který je instalován v systému nebo je do systému zkopírován. Tyto možnosti významně přibližují uživatelům běžné rekvizity, které dříve byly dostupné pouze v interaktivních programech, ale zároveň nebyly narušeny původní možnosti, na které jsou uživatelé zvyklí, a také principy precizní sazby a jednoduché získávání vysoce kvalitních výstupů. Podíváme se nyní na zmíněné prvky po technické stránce.

Jak začít v systému XeLaTeX

Oproti dokumentu v systému LaTeX těch změn mnoho není. Dokonce lze říci, že dokument pro LaTeX lze bez jakýchkoliv úprav přeložit překladačem XeLaTeXu. To bychom ale nemohli využít zmiňovaných výhod. Proto rádi něco přidáme.

Minimální dokument, ve kterém již můžeme hlavních výhod využít, má tento tvar:

Minimální dokument

```
\documentclass{article}
\usepackage[czech]{babel}     standardní práce s češtinou
\usepackage{xltxtra}          zavede se i balíček fontspec
\begin{document}
Příliš žluťoučký kůň úpěl ďábelské ódy.
                              standardně je zavedeno písmo Latin Modern
\end{document}
```

Práce s písmy

V dokumentu je vidět balíček `xltxtra` – ten definuje několik podpůrných příkazů pro přizpůsobení zvyklostem v dokumentech formátu LaTeX. Ale stěžejním balíčkem je `fontspec`, který je

z balíčku `xltxtra` volán. Jeho základním příkazem je `\fontspec` a jeho varianty, umožňující použít jakékoliv písmo nakopírované v systému. Stejně jednoduše jako v jiných programových systémech stačí jen písmo připojit a je okamžitě k dispozici. Příklady:

Písmo

```
\fontspec[Mapping=tex-text]{Constantia}\Large
Základem firmy je -- jak víme -- \textbf{dobrý} marketing.
```

Po vysázení dostaneme:

Základem firmy je – jak víme – dobrý marketing.

Ve vysázeném textu se používají všechny „fontové“ vymoženosti – zde vidíme například slitek „fi“. Volitelný parametr `[Mapping=tex-text]` říká, že se mají také používat běžné „zvyklosti“ zdrojových textů, například dva spojovníky za sebou se mají proměnit v pomlčku. Písmo Constantia[1] je distribuováno s operačními systémy firmy Microsoft od roku 2003. Veškerá příprava na jeho připojení spočívá pouze ve zkopírování do adresáře, odkud operační systém bere všechna písma (typicky windows\fonts, v propracovanějších systémech /usr/share/fonts), a napsat jeho jméno do zmíněného příkazu.

Doba, kdy byly dokumenty psané v TEXu a jeho nadstavbách rozpoznatelné na dálku podle použitého písma Computer Modern a jeho modernějších příbuzných, je již pravděpodobně za námi. Písmo Computer Modern je vysoce kvalitní a pro matematickou sazbu například mimořádně vhodné, ale pro řadu dokumentů se nehodí[2]. Je typografickou chybou, použijeme-li pro určitý dokument písmo, které tam logicky ani esteticky nepatří. S možnostmi XELATEXu nabýváme schopností, které umožní tyto typografické zásady lépe realizovat.

Je možné použít fonty ve formátu Adobe Type 1, True type a hlavně také Open type. Fonty ve formátu Open Type byly dříve v programech typu TEX v podstatě nedostupné, přitom se jedná z typografického i technického hlediska většinou o velmi kvalitní rekvizity s mnoha dříve nedostupnými možnostmi.

Jednou z výhod kvalitních písem jsou například variantní číslice. Verzálkové číslice vhodné do tabulek (mají stejné šířky kvůli správnému svislému zarovnání) a minuskové číslice vhodné do běžného textu mohou být zařazeny volbou `Numbers=Uppercase`, resp. `Numbers=OldStyle`. Například v již zmíněném fontu Constantia:

Skákavé číslice

```
{\fontspec[Numbers=Uppercase]{Constantia}
verzálkové číslice do tabulek 1234567890}
{\fontspec[Numbers=OldStyle]{Constantia}
a~minuskové číslice do textu 1234567890.}
```

dává po vysázení:

verzálkové číslice do tabulek 1234567890
a minuskové číslice do textu 1234567890.

Z dalších voleb, které lze u zavedeného písma použít, lze zmínit ještě alespoň tyto tři:

- `Color` – nastavení barvy zaváděného písma. Zadává se jako název barvy a spolupracuje s balíkem `xcolor` místo staršího `color`, například

[1] Pro svůj Office 2007 si nechala firma Microsoft nakreslit skupinu 6 písem: Constantia, Cambria, Calibri, Corbel, Consolas, Candara. Jedná se o překvapivě kvalitní materiál, který lze nyní díky schopnostem XELATEXu využít i mimo produkty této firmy.

[2] Například pohádky jím vysázené připomínají spíše učebnici.

Barvy

```
\definecolor{salonclr}{cmyk}{0.6,0.8,0,0.3}
\fontspec[Color=salonclr]{Playbill}Saloon
```

se vysází jako **Saloon**

- `Scale` – nastavení základní velikosti písma. Zadává se jako koeficient, kterým se násobí původní velikost, ale k dispozici jsou ještě „chytré“ hodnoty `MatchLowercase` a `MatchUppercase`, které velikost upraví podle minusek nebo verzálek základního písma. Příklad:

Škálování

```
My \fontspec[Color=salonclr,
             Scale=MatchUppercase]{Playbill}Saloon
```

je po vysázení My **Saloon**

- `LetterSpace` – prostrkání. Prostrkání (nebo obecně úprava meziznakových mezer) je typograficky velmi problematická věc, kterou se nedoporučuje užívat, přesto je tento parametr zajímavý, protože umožňuje dříve velmi obtížně dostupnou rekvizitu. Pro koncipování různých akcidenčních prvků se může velmi hodit. Zadává se jako procento stupně písma, například:

Prostrkání

```
\fontspec[LetterSpace=20]{Cambria}PŘÍKLADY
```

dává po vysázení P Ř Í K L A D Y

Užitečné jsou rovněž příkazy příbuzné (mají stejnou syntax, ale jiný dosah). Jedná se o možnost nastavení základního písma celého dokumentu příkazem `\setmainfont`, nastavení bezserifové rodiny a strojopisné rodiny pomocí příkazů `\setsansfont`, resp. `\setmonofont`. Lze tak dokument ovládat jako celek, zároveň také zajistit typografickou správnost – identický font pro všechny prvky dokumentu.

Pro pohodlnější volbu písem lze často používané volitelné parametry soustředit do jednoho místa použitím příkazu `\defaultfontfeatures`.

V tomto dokumentu je například nastaveno:

Písma v dokumentu

```
\defaultfontfeatures{Mapping=tex-text}
\setmainfont{Lido STF}
\setmonofont[Scale=0.9]{Courier New}
\setsansfont{Calibri}
```

Symboly ve vstupním textu

Skutečnost, že XeLaTeX implicitně zpracovává vstupní text v kódování UTF-8, vede k zajímavým možnostem – jsou přímo zpracovávány speciální znaky, které se dříve musely vkládat pomocí příkazů. Podmínkou je, aby příslušný znak byl v daném fontu přítomen. Jedná se například o tyto možnosti:

- Uvozovky – lze je zapisovat přímo tak, jak budou vidět ve vysázeném textu, záleží jen na tom, zda editor umožňuje jednoduše uvozovky vkládat. Například text

Uvozovky

```
„slovo“ nebo také »slovo«
```

  vede k vysázení „slovo“ nebo také »slovo«.

- Pomlčky – jsou znakem, který lze efektivně zapisovat i v podobě zdvojeného (ztrojeného) spojovníku. Tato možnost však existuje jen tehdy, pokud při připojení fontu zapneme volbu `Mapping=tex-text`. Také lze použít vložení pomlčky přímo jejím kódem 2013h, resp. 2014h (to mnoho editorů umožňuje), například:

Pomlčky

```
Čtverčíková pomlčka je — a půlčtverčíková –.
```

  Po vysázení: Čtverčíková pomlčka je — a půlčtverčíková –.

- Znak euro – lze jej vložit přímo z klávesnice, ale dříve bylo pro jeho vysázení potřebné připojit balíček včetně instalace nových fontů, z nichž byl znak následně vybrán a vysázen. Tomu všemu se můžeme elegantně vyhnout:

Měny

```
Cena 20 € může být zaplacena i v britských £ nebo japonských ¥.
```

  Po vysázení: Cena 20 € může být zaplacena i v britských £ nebo japonských ¥.

- Další znaky – paragraf, znak násobení, ochranná známka, copyright. Měly společný nedostatek: byly sice dostupné jednoduchými příkazy, ale ty vybíraly příslušné znaky z fontu cmsy, což v mnoha případech nesouhlasilo s kresbou použitého okolního fontu. Protože se původně sázelo v podstatě jen písmem Computer Modern nebo jeho modernějšími alternativami, nevadilo to – kresba byla pro tento případ optimalizována. Ale dnes, kdy se používá mnoho jiných písem, je rozdíl v kresbě dost podstatný a na první pohled je i laikům zřetelný. Je tedy přirozené zapisovat tyto znaky přímo z klávesnice a vidět jejich správný obraz i ve vysázeném textu: `§ × ® ©` je vysázeno § × ® ©.

- Vybrané matematické symboly – zde lze zmínit zejména řecké mí pro předponu mikro-, znak pro stupeň a znaky pro běžné mocniny používané v jednotkách čtvereční (krychlové) metry apod. Podobně jako u skupiny předchozích symbolů je zde podstatná shodná kresba s okolním fontem místo výběr symbolu ze zvláštního fontu. Zápisem `30 µF, 5 °C, 2 m² a 3,6 m³` dostáváme 30 µF, 5 °C, 2 m² a 3,6 m³.

Z uvedených příkladu je patrné, že XeLaTeX usnadňuje běžnou práci a významně rozšiřuje možnosti použitelné prakticky ve všech dokumentech.

Kapitola 3

Grafika v dokumentech

Oblast počítačové grafiky je velmi rozsáhlá. Její základní prvky (rastrový a vektorový obrazový formát, grafické editory) patří k základům zpracování jakýchkoliv dokumentů. Představení grafických možností a balíčků `graphicx` a `color` bylo součástí úvodního kursu a je běžnou součástí všech příruček a manuálů týkajících se sazby v systémech typu LaTeX.

Konkrétně v systémech postavených na principu TeXu mají grafické prvky poněkud zvláštní postavení. Vzhledem k tomu, že základní princip není postaven na grafické editaci jako v jiných interaktivních systémech, je grafika až na výjimky vždy technicky řešena jako externí prvek a spolupráce se základními sazebními algoritmy TeXu je tedy nepřímá. Velkým posunem v grafických možnostech byl výstup překladu do formátu PostScript a později i PDF, ale tím paradoxně zároveň zmizela velmi důležitá nezávislost na výstupních zařízeních.

Ze všech grafických nástrojů, které jsou k dispozici, bychom se v tomto textu zmínili o dvou – prostředí `picture` a balík `pstricks`.

Prostředí `picture` má zcela výjimečnou vlastnost – jako jediné tvoří zmíněnou výjimku a není závislé na externím grafickém zpracování. Prvky obrazu jsou totiž tvořeny „písmeny“ speciálních fontů, které pak dohromady dávají šikmé čáry, kolečka, ovály, šipky apod. Z toho důvodu lze říct, že použití je velmi žádoucí a ve všech dokumentech, kde tomu tak je, bude přínosem a eliminuje problémy s jakoukoliv kompatibilitou. Zásadní nevýhodou ovšem je malý repertoár grafických prvků – nutnost pracovat se speciálními fonty omezuje jak možnosti různých tvarů, tak i jejich velikosti. Ve srovnání s běžnými možnostmi téměř jakéhokoliv grafického editoru jde tedy o zlomek, který ve většině případů nedostačuje požadavkům na grafické doplňky v dokumentech.

Podobným principem zápisu zdrojového textu se vyznačuje balík `pstricks`, který je standardní součástí distribucí již několik let. Jeho příkazy překonávají základní nevýhodu prostředí `picture`, ale ke kreslení využívají výstupního ovladače zobrazujícího vysázený text ve formátu PostScript. Z toho plyne omezení na tento formát a bohužel i nemožnost přímého použití v moderních překladačích (XeTeX) generujících výstup do PDF. Přesto však určitou manipulací a následným převodem lze výstup PostScript převést do PDF a získat požadovaný tvar.

Krátkou ukázkou prezentujme koncept `pstricks` – velká podobnost s prostředím `picture` umožňuje snadnou integraci do zdrojového textu:

```
\pspicture(10,5)
\psline(0,0)(1.8,3)                              linka
\psline[border=2pt]{*->}(0,3)(1.8,0)
\psframe*[linecolor=gray](2,0)(4,3)              rámeček
\psline[linecolor=white,linewidth=1.5pt]{<->}(2.2,0)(3.8,3)
\psellipse[linecolor=white,linewidth=1.5pt,              elipsa
        bordercolor=gray,border=2pt](3,1.5)(.7,1.4)
\endpspicture
```

Z ukázky je patrné, že každý objekt je ovlivnitelný řadou parametrů, které se dají i automatizovat (přednastavit), lze tedy globálně ovlivňovat tvar a velikost kreseb v celém dokumentu.

Balík obsahuje dlouhou řadu grafických objektů: úsečky s různými zakončeními šipkami a jinými prvky, kružnice, elipsy, křivky, rámečky, polygony atd. Zajímavou možností jsou tzv. uzly, tj. místa v grafickém prostoru, která mají svoje symbolické názvy a mohou být automaticky spojovány čarami nebo křivkami – tato technika umožňuje snadno kreslit diagramy, schémata, grafy apod. Objekty mohou mít řadu parametrů čar a výplní.

Základní manuál dodávaný do distribuce čítá přes 300 stran, pro další informace tedy čtenáře odkážeme na tento zdroj.

Na procvičení:

1. Nakreslete organizační schéma fakulty.
2. Změňte nastavení parametrů tak, aby čáry ve schématu z předchozí úlohy byly modré a měly tloušťku 0,75 bodu, výplně obdélníků byly desetiprocentní šedí.

Kapitola 4

Prezentace

Velmi častým úkolem je realizace prezentací. V praxi vidíme, že se masivně používá prezentační software, který je součástí kancelářských balíků, ale podobně jako programy pro zpracování textů mají i tyto součásti své nevýhody – jednou ze základních je nekvalitní práce s textem.

Prezentaci lze vytvořit jako obyčejný soubor – dokument ve formátu PDF, kde jednotlivé stránky (snímky) mají požadovaný obsah, v němž lze plně uplatnit všechny výhody sazby v systémech typu LaTeX. Zdálo by se tedy, že nemusíme ani žádný speciální programový systém používat. Pro samotnou prezentaci má však tento dokument minimální podporu. Proto vznikly různé balíčky umožňující podpořit jak tvorbu, tak i samotný proces prezentace takových dokumentů.

Dokumentní třídou, která má v tomto směru široké uplatnění a význačné vlastnosti, je bezesporu `beamer`. Při zachování možností sazby (zejména kvalitní fonty, speciální znaky, matematika) disponuje mnoha prvky pro podporu prezentace, orientace v dokumentu a zobrazování pomocných informací.

Základním prvkem prezentace je **snímek**, řešený prostředím `frame`. Posloupností snímků získáme prezentaci. Snímek může mít řadu nastavení a atributů. Velmi oblíbené jsou možnosti nastavení vzhledu celé prezentace, kterých je řada už předdefinována a lze jen využít velkého množství práce, která byla do konstrukce těchto vzhledů již vložena. Vzhled může být ještě modifikován barevným podáním, kde kombinace barev lze rovněž předdefinovat do barevných schémat – existuje již opět několik předdefinovaných.

Ukázka zdrojového textu představuje malou ochutnávku z rozsáhlých možností, kterým se na malém prostoru tohoto textu nemůžeme plně věnovat – čtenáře lze odkázat na 240stránkový manuál. Uvádíme kompletní text dokumentu pro překlad XeLaTeX.

Ochutnávka prezentace

```
\documentclass{beamer}
\usepackage[czech]{babel}
\usetheme{Frankfurt}                          nastavení vzhledu
\usecolortheme{orchid}                        nastavení barevného podání
\usepackage{xltxtra}
\begin{document}
\begin{frame}                                 snímek
\frametitle{Neexistuje největší prvočíslo}
\framesubtitle{Důkaz používá \textit{reductio ad absurdum}.}
\begin{theorem}
Neexistuje největší prvočíslo.
\end{theorem}
\begin{proof}
\begin{enumerate}                             lze používat běžné rekvizity
\item<1-| alert@1> Nechť $p$ je největší prvočíslo.   zvýraznění položky
\item<2-> Buď $q$ součinem prvních $p$ čísel.         postupné zobrazování
\item<3-> Pak $q+ 1$ není dělitelné libovolným z nich.
\item<1-> Z toho plyne, že $q+ 1$ je
          také prvočíslo a je větší než $p$.\qedhere
\end{enumerate}
\end{proof}
\end{frame}
\end{document}
```

V ukázce bylo použito příkazem `\usetheme` téma zvané `Frankfurt`. Témata, která jsou dále k dispozici, mají tyto názvy: Antibes, Boadilla, Frankfurt, Juanlespins, Montpellier, Singapore, Bergen, Copenhagen, Goettingen, Madrid, Paloalto, Berkeley, Darmstadt, Hannover, Malmoe, Pittsburgh, Berlin, Dresden, Ilmenau, Marburg, Rochester.

Podobně lze vystřídat i barevná schémata v příkazu `\usecolortheme` z následujícího repertoáru: albatross, seagull, fly, crane, wolverine, dove, beaver, beetle, lily, orchid, rose, whale, seahorse, dolphin.

Na procvičení:

1. Použijte podobný zdrojový text jako v ukázce a vyzkoušejte možnosti různých vzhledů. Začněte implicitním vzhledem bez použití příkazu `\usetheme`.

2. Zjistěte z manuálu, jak lze zobrazovat obsah prezentace a průběžnou pozici v obsahu.

Kapitola 5

Principy strukturního značkování

Systémy postavené na principu TEXu oplývají rozsáhlými možnostmi v mnoha příkazech, v internetových archivech lze nalézt nepřeberné množství balíčků řešících jednotlivé problémy počínaje například prací s grafikou a sazbou šachových diagramů konče. To vše však ještě zdaleka není všechno – kdybychom chtěli všechny ty možnosti zvládnout, nestačil by nám na to pracovní čas a možná ani celý život.

Uvedené možnosti je však potřeba vidět ještě i z poněkud jiného úhlu – proč je k dispozici tak nepřeberné množství různých stylů, balíčků a tříd? Nejde jen o řešení určitých dílčích problémů, ale o celkový přístup k tvorbě dokumentů, který je v jiných systémech prakticky nedostupný. Základním principem TEXu je možnost tvořit **vlastní (uživatelské) příkazy** a dokonce i **měnit význam znaků** ve vstupním souboru.

Je tedy přirozené, že správné užití takových systémů spočívá v ovládnutí schopnosti tvořit vlastní příkazy a přistupovat ke konstrukci dokumentu s těmito rekvizitami.

K tomu je však potřebné se podívat na tvorbu dokumentů z pohledu efektivní metody značkování.

Pojem **značka** představuje v dokumentu příkaz pro zpracovávající program. Značky nesou informace dvojího druhu – vizuální (definují vzhled příslušného dokumentního prvku) a strukturní (definují význam dokumentního prvku).

Příklad: `\textbf` je vizuální značka – říká jen, že úsek textu má být tučně, ale nevíme proč, `\section` je strukturní značka – říká, že úsek textu je nadpisem první úrovně, ale nevidíme, jak takový nadpis vypadá.

Je asi jasné, že máme-li dokument vůbec někde uvidět, je nutné mít vizuální informace. Ale pro zpracování a třeba i znovupoužití dokumentu jsou daleko významnější značky strukturní. Ke každé strukturní značce lze kdykoliv později přidat vizuální informaci, ale *opačně to nejde*. Důležité je, že strukturní informace, která hovoří o významu textového prvku, musí být do dokumentu dodána autorem. Těžko může někdo jiný například říct, která část textu je zdůrazněná, která část je nadpisem určité úrovně apod. Je však přirozené, že víme-li, že určitá část textu je nadpisem druhé úrovně, můžeme jí kdykoliv přisoudit vizuální podobu vhodnou pro aktuální účel a tuto podobu i podle potřeby změnit.

V tomto okamžiku je ovšem důležité ještě zdůraznit, že vizuální podoba strukturní značky může být oddělena od dokumentu, může být soustředěna do zvláštního souboru. Připojením tohoto souboru k dokumentu se připojí všechny vizuální definice a dokument dostane určitý tvar. Výměnou tohoto souboru můžeme z jednoho a téhož dokumentu dostat odlišnou vizuální podobu, aniž bychom do dokumentu zasáhli. To je velmi důležité pro znovupoužití dokumentů.

Je tato technologie dosažitelná v systémech postavených na principu TEXu? Odpověď je až překvapivě jednoznačná: Ano, a to v podobě těžko dosažitelné jinde. Máme tedy jedinečnou příležitost vytvářet dokumenty s vysokou úrovní obecnosti a s mnoha možnostmi definovat vizuální podobu.

Jak už bylo v příkladu zmíněno, jsou k dispozici značky vizuální i strukturní. Lze dokonce říct, že právě systém LATEX je vybaven mnoha strukturními značkami – pro nadpisy, pro poznámky pod

čarou, pro seznamy atd. Současně ale z praxe víme, že s těmito strukturními značkami nevystačíme, nebo potřebujeme jejich vizuální obměnu. Čtete-li například tento text, asi uvažujete, jakým způsobem byl označkován. Je to celkem jednoduché – kapitoly jsou značeny příkazem `\chapter`, sekce příkazem `\section` apod. Jak je ale možné, že kapitoly jsou sázeny modrým písmem a zarovnány vpravo s nápisem „Kapitola“ odděleným linkou? Jak je možné, že sekce jsou také v modré barvě a písmem stejným jako nadpisy kapitol?

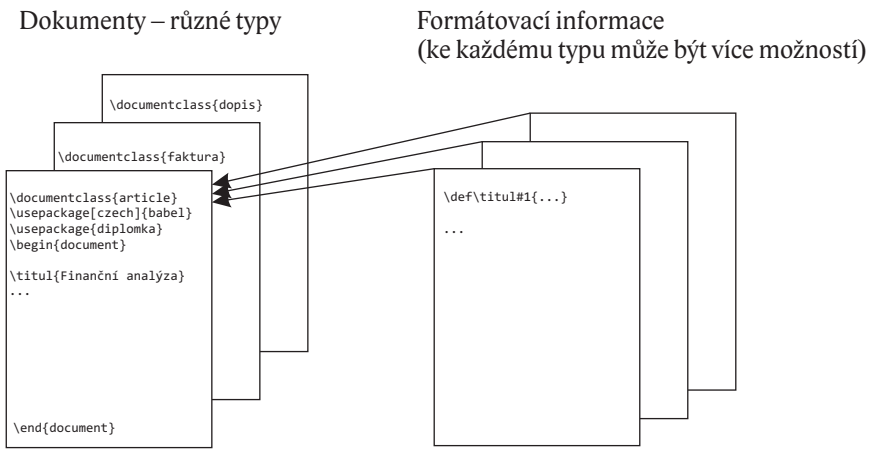
Bylo by *velmi nemoudré* přímo do dokumentu psát například místo každého začátku kapitoly ve tvaru `\chapter{Prezentace}` úsek v této podobě:

Takhle prosím ne...

```
\newpage
{\fontspec{Calibri}\color{blue}
\fontsize{12pt}{14pt}\selectfont Kapitola 3}
\par\medskip\hrule
\par \vskip 10mm
{\fontspec{Calibri}\color{blue}
\bfseries\fontsize{20pt}{24pt}\selectfont
Prezentace}
\par\vskip 5mm
```

Kdybychom potřebovali kdykoliv změnit tvar kapitolových nadpisů, museli bychom zhruba 14× editovat tentýž text. Ano, mohli bychom si řadu věcí zautomatizovat tím, že použijeme v editoru například funkci „Najdi a nahraď“ – ale kdybychom například chtěli nadpisy netučně, jak poznáme, že příslušné `\bfseries` patří k nadpisu a ne k nějakému úplně jinému textu?

Proto potřebujeme zcela jednoznačně takovému typu značkování zabránit. Do dokumentu je potřebné vkládat *výhradně* strukturní informace, jejichž vizuální podobu je velmi výhodné *soustředit mimo dokument*, abychom mohli pokud možno jediným zásahem ovlivňovat vždy všechny výskyty téhož prvku najednou. Kýžený stav ilustruje obrázek 5.1.



Obr. 5.1: Typy dokumentů a jejich formátování

Podobný systém formátování je znám i z jiných technologií – například kaskádové styly pro formátování dokumentů v jazyce HTML nebo šablony pro formátování dokumentů v programech typu LibreOffice Writer nebo Microsoft Word. Oproti těmto případům jsou však v systémech typu TEX k dispozici (jak už bylo zmíněno) daleko rozsáhlejší možnosti.

Kapitola 6

Realizace strukturního značkování v systémech typu TEX

Podíváme-li se na technologii systémů typu TEX, je pro kvalitní strukturní značkování k dispozici zejména možnost tvorby nových příkazů, tedy takových, které potřebujeme pro konkrétní typ dokumentu.

Na první pohled je v tomto přístupu rozpor a vnucuje se otázka: Proč máme tvořit nové příkazy, když už existuje taková masa příkazů, které už někdo udělal a které jsou vystaveny v rozsáhlých archivech na internetu?

Je samozřejmé, že využít těchto rozsáhlých hotových příkazů je velmi efektivní, ale jde spíše o poněkud jiný přístup k tvorbě dokumentů. *Primárním cílem* je vyjádřit **význam jednotlivých prvků** – ty jsou v každém dokumentu trochu jiné a vyžadují tedy své vlastní značkování.

Postup

Mohli bychom tedy uvedené myšlenky soustředit do určitého doporučení, jak na dokument pohlížet a jak při tom využít příslušnou technologii. Postupovat lze dvěma způsoby vedoucími ke stejnému výsledku:

1. Vložíme nebo získáme kompletní materiál dokumentu (texty, obrázky, tabulky, matematické výrazy).
2. Podle vloženého materiálu si ujasníme, z jakých **významových prvků** je dokument složen.
3. Do vloženého materiálu uvedeme ve vhodné podobě informace *o významech* jednotlivých prvků – strukturní značky.
4. Definujeme vizuální podobu všech potřebných prvků, tedy všech strukturních značek.

Druhý způsob se liší tím, co budeme mít dříve – můžeme totiž podle potřeby zaměnit pořadí uvedených kroků: Napřed „vyprojektujeme“ dokument, tedy připravíme jeho významové prvky a vyrobíme jejich vizuální podobu. Následně budeme vkládat (tvořit nebo získávat) materiál a opatřovat ho již hotovými definovanými značkami.

U každého dokumentu si můžeme vymyslet vlastní a zcela originální systém strukturních značek. Ukažme si na příkladu konkrétního dokumentu, jak budeme popsaný postup aplikovat.

Řešený příklad

Zadání: Potřebujeme vyrobit sbírku úloh z fyziky. Předpokládáme, že sbírka bude členěna do několika oddílů (Mechanika, Termika, Optika apod.) a oddíly do sekcí, v každém oddílu budeme

chtít příklady označovat čísly od jedničky, k zadáním příkladů budeme někdy potřebovat ilustrativní obrázek, někdy nápovědu, k příkladům bude potřeba někam do jiného místa (často do zadní části sbírky) vypsat správné výsledky. Sbírka by měla mít na začátku titulní stranu, vydavatelský záznam a obsah s názvy oddílů a sekcí.

Všimněte si, že zadání je do jisté míry přibližné, vůbec se například nezabývá vizuální podobou, ale je natolik výstižné, že popisuje potřebné prvky. Další požadavky na celkovou podobu dokumentu jsou také obvykle zadány, například požadovaný formát, z něhož lze odvodit řadu parametrů.

Návrh strukturních značek

Pracujeme v systému typu TEX, takže víme, jak mohou značky zhruba vypadat, podle toho také navrhneme tvary pro sbírku úloh. Někdy je možné si vybrat z více možností, jeden z použitelných tvarů může vypadat takto:

- Titulní stranu s názvem, autorem a rokem vydání označíme `\titul{název}{autor}{rok}`
- Vydavatelský záznam vyžaduje kromě již zadaných údajů ISBN, což zapíšeme příkazem `\vydzaznam{ISBN}`
- Místo, kde se má objevit obsah, označíme příkazem `\obsah`
- Oddíl sbírky označíme `\oddil{Název}` a sekci `\sekce{Název}`
- Každá úloha sbírky bude zadána příkazem `\uloha{zadání}{výsledek}`
- Pokud se v úloze vyskytne obrázek připravený k vložení v nějakém grafickém souboru, bude zapsán příkazem `\obrazek{soubor}{popisek}`
- Bude-li potřebné k úloze napsat nápovědu, zvolíme příkaz `\napoveda{text}`
- V zadní části budou vypsány výsledky příkazem `\vysledky`

Zdrojový text celé sbírky tedy může vypadat takto:

Sbírka úloh

```
\documentclass{sbirka}                  zatím tajemná dokumentní třída
\begin{document}
\titul{Sbírka úloh z fyziky}{Alfréd Noha}{Brno 2012}
\vydzaznam{ISBN 978-80-2312-143-8}
\obsah
\oddil{Mechanika}
\sekce{Rovnoměrný přímočarý pohyb}

\uloha{Z města $A$ vyjel v~7.00 hodin nákladní automobil...
       \obrazek{auto1}{Trasa nákladního automobilu}
      }{55,7\,km$\cdot$h$^{-1}$; setkají se v~8.55 h}
\uloha{...}{...}                        tady bude spousta dalších úloh
... atd.                                a také oddílů a sekcí
\vysledky                               zde se nějakým kouzlem objeví výsledky
\end{document}
```

Z takového zápisu sice vůbec nevyplývá, jak bude výsledný dokument vypadat, ale všechny důležité informace jsou v něm již zadány. Je samozřejmé, že dokument ani nepůjde v této chvíli zpracovat, protože překladač nic neví o nějaké třídě `sbirka` a o dalších příkazech, jejichž názvy jsme si vymysleli. Následující text pojednává právě o tom, jak tyto „neznámé“ realizovat a dosáhnout toho, že se nejen uvedený dokument přeloží, ale vysází se do podoby, jakou budeme potřebovat. A co víc, tuto podobu budeme moci změnit, pokud budeme chtít místo tištěné formy formu elektronickou nebo formu, ve které budou vybrány jen některé informace (například jen výsledky) atd.

V ilustrativním příkladu je možné si rovněž povšimnout, že základní výbava, tj. sazba speciálních znaků a matematických výrazů, je použitelná zcela univerzálně, máme tedy možnost precizně vysázet jednotlivé texty.

Za jednotlivými příkazy se bezesporu skrývá množství činností, které budeme po systému požadovat. Například automatické číslování, různé přenosy informací (do obsahu, do výsledků), skládání do stránek podle určitých pravidel apod. V tom je možné už vidět určitou efektivitu – část z těchto činností v jiných systémech svěřit počítači nemůžeme, protože k tomu nejsou potřebné nástroje.

Zároveň zde můžeme vidět zásadní rozdíl v přístupu k dokumentu v neinteraktivním prostředí, kde pracujeme se zdrojovým textem, a v interaktivním, kde pracujeme s náhledem dokumentu. Práce se zdrojovým textem je sice zejména pro začátečníky dost odrazující, ale pro efektivitu tvorby je velmi důležité si uvědomit, že fáze *vlastní tvorby* a fáze *vizuální úpravy* je možné striktně oddělit, a to i personálně. Práce s pouhým textem je daleko jednodušší, než *současná práce s textem a vizuální podobou* v náhledu, kde se uživatel velmi často uchyluje k neustálému manipulování s objekty na obrazovce pomocí myši a vlastně pracuje ručně – *za počítač*. Naší snahou tedy bude co nejvíc práce naopak počítači svěřit – on to udělá rychleji a přesněji.

Kapitola 7

Tvorba nových příkazů

Příkazy, které jsme si vymysleli, budeme chtít realizovat. Jak to lze udělat?

Ještě než se začneme zabývat konstrukcí, je potřebné uvést několik podstatných informací. Příkazy, které vidíme nebo používáme ve zdrojových textech a které se v literatuře též nazývají **řídicí sekvence** (control sequences), můžeme rozdělit na dvě zásadné odlišné kategorie:

1. Primitivní příkazy (primitiva) – příkazy implementované přímo v jádře TEXu. Je jich kolem 300.
2. Makra (makropříkazy) – nově deklarované pojmenované skupiny jiných řídicích sekvencí.

To, co budeme vytvářet, jsou tedy pouze makra. Všechny nadstavby jsou vlastně jen (někdy rozsáhlé) soustavy maker definovaných z primitivních příkazů a z jiných maker.

Pro jednoduchost budeme vše nazývat příkazy, jen tam, kde by hrozilo nějaké zmatení významů, upřesníme pojmenování.

Činnost překladače TEXu

Pro pochopení rozdílu mezi primitivem a makrem ještě musíme uvést, co vlastně všechno překladač TEXu zhruba dělá, než všechen materiál ze zdrojového textu zpracuje. Pracuje přibližně v těchto fázích:

- Vstupní fáze – čtení jednotlivých znaků a jejich rozdělení do posloupnosti symbolů, tzv. tokenů. Jeden token může být reprezentován samostatným znakem nebo například řídicí sekvencí. Každý token má mimo jiné i svou kategorii, která předurčuje jeho další zpracování.
- Fáze zpracování maker – je-li přečteným tokenem makro, provádí se tzv. expanze – makro se nahradí svým tělem. Pokud jsou v tomto těle další makra, opět se expandují, až zbudou jen jednotlivé znaky a primitivní příkazy.
- Fáze vlastní sazby – materiál s expandovanými makry a primitivními příkazy je interpretován a převeden do vysázené podoby (dvi, pdf).

Vidíme tedy, že se makra uplatňují v jiné fázi než primitivní příkazy a vlastní sazba je prováděna výhradně ze znaků a primitivních příkazů. Makra slouží tedy hlavně k tomu, abychom mohli zdrojový text zapisovat jednodušeji a přirozeněji – je to jakási vzdálená obdoba vyššího programovacího jazyka oproti strojovému kódu.

Princip zpracování zdrojového materiálu popsaný v těchto třech krocích není neobvyklý, můžeme něco podobného nalézt i v jiných programovacích jazycích. Hlavní kouzlo překladače TEXu však spočívá v tom, že *všechny fáze lze do značné míry ovlivňovat*. Takže například lze ovlivnit, do jaké kategorie patří čtené znaky, čímž lze zcela změnit chápání stejného textu, lze také ovlivnit, jak se bude provádět expanze maker, takže lze upravit pořadí, v jakém se makra na vstupu objeví apod. To s sebou nese hodně různorodých překvapivých možností a spoustu legrace. Malou ochutnávku předložíme v tomto textu.

Tvorba příkazů

A nyní zpět k příkazům, které potřebujeme například do naší sbírky úloh. Existuje několik možností, jak nový příkaz vytvořit:

- Příkaz `\def` – jedná se o jednu z nejfrekventovanějších cest, je to primitivní příkaz. Tento příkaz umožňuje vytvořit nové makro s až devíti parametry, zároveň však nekontroluje, zda už příkaz se stejným názvem existoval. Pokud to tak bylo, nový příkaz *bez varování nahrazuje* příkaz původní, třeba i primitivní. Platnost nově definovaného příkazu se omezuje na prostor od místa definice do konce nejbližší skupiny.
- Makro `\newcommand` – je to LaTeXová obdoba definičního příkazu s poněkud jinou syntaxí a zároveň také kontrolou dosavadní existence příkazu. Pokud už takový příkaz existoval, pokus o novou definici skončí chybou. Definice je platná ve skupině.
- Makro `\renewcommand` – doplněk k předchozímu příkazu, který slouží výhradně ke *změně definice* již existujícího příkazu.
- Makro `\newenvironment` – příkaz pro definici prostředí. Prostředí jsou specifickou rekvizitou systému LaTeX a jeho následníků. Příkaz opět kontroluje dřívější existenci a nedovolí změnu definice. Platí ve skupině.
- Makro `\renewenvironment` – doplněk předchozího příkazu sloužící výhradně k redefinici již existujícího prostředí.

LaTeXové varianty definičních příkazů mají svou výhodu v kontrole existence definovaného makra, ale jinak se nevyznačují žádnými odlišnými schopnostmi oproti příkazu `\def` a představují jeho pravé podmnožiny. Budeme se tedy v dalším textu zabývat pouze tímto primitivem a jeho možnostmi.

Tvary definic

Základním a nejjednodušším tvarem definice nového příkazu je:

```
\def\prikaz{tělo}
```

Název nového příkazu `\prikaz` je implicitně složen buď jako slovo z písmen anglické abecedy, nebo jako jeden neabecední znak. Tělo definice obsahuje posloupnost příkazů, která se v okamžiku použití makra *rozvine* do zdrojového textu (viz již zmíněná činnost překladače).

Příklady

Př. 7.1: `\def\autor{Alfréd Noha}` definuje příkaz `\autor`, který při svém uvedení vypíše text „Alfréd Noha“.

Př. 7.2: `\def\,{\kern 0.25em}` definuje příkaz `\,` – jeho uvedením vznikne pevná mezera velikosti čtvrtiny čtverčíku.

Př. 7.3: `\def\vakat{\newpage\thispagestyle{empty}~}` definuje příkaz pro vytvoření prázdné stránky v dokumentu (vakát)

Prefix `\global` umožňuje nařídit, aby definice byla platná i po skončení skupiny, ve které je uvedena. Například `\global\def\ISBN{ISBN 80-7234-234-8}` je globálně platnou definicí (od místa uvedení do konce dokumentu).

Místo zápisu `\global\def` lze zkráceně psát `\gdef` s identickým významem.

Neseparované parametry

Definovaný příkaz může mít parametry (celkem maximálně 9) označené `#1` až `#9`. Jejich použití ilustruje následující příklad:

```
\def\velke#1{{\Large\bfseries #1}}
```

V definici je symbol parametru zapsán bezprostředně za název nového příkazu a pak se ještě vyskytuje někde v těle definice. Symbolicky to znázorňuje skutečnost, že se v tom místě okopíruje skutečný parametr, vzniklý při použití příkazu. Parametr se v těle definice může vyskytovat libovolněkrát, třeba i nulakrát.

Příkaz můžeme použít takto: `\velke Nápis` nebo takto: `\velke{Nápis}` Mezi těmito dvěma možnostmi je samozřejmě rozdíl, ale zde pochopíme, co se chápe jako parametr: v případě tzv. neseparovaných parametrů se z následujícího textu vezme jeden token (v prvním případě znak „N“) nebo materiál ohraničený svorkami (ve druhém případě „Nápis“). Z tohoto příkladu je zároveň patrné, jak se chovají i jiné již známé příkazy, například v matematickém režimu index nebo exponent: `a_i` = `a_{i}`, ale `a_xy` není totéž co `a_{xy}`. Protože jedním tokenem může být i příkaz, můžeme například napsat `\frac\infty\infty` a bude to zlomek $\frac{\infty}{\infty}$, ale zlomek $\frac{ax}{by}$ už musíme zapsat se svorkami `\frac{ax}{by}`.

Skutečným parametrem může být materiál tvořící jeden odstavec. Je to poměrně chytrá obrana proti zapomenuté ukončovací svorce. Při překladu se ohlásí chyba, pokud je nalezen konec odstavce a není doposud nalezena svorka ukončující parametr. Z praxe lze jen dosvědčit, že tato obrana je velmi účinná, protože většina příkazů ve svém parametru moc materiálu nemá a konec odstavce přijde dřív, než by makro pohltilo zbytek celého dokumentu.

Ale existuje prefix `\long` umožňující definovat makro, jehož skutečným parametrem může být materiál přesahující jeden odstavec. Použijeme je pouze tehdy, potřebujeme-li ve skutečném parametru opravdu hodně materiálu.

Příklady

Př. 7.4: Uvažujme definici:

```
\def\titul#1#2#3{\newpage\thispagestyle{empty}
    \begin{center}
    \vspace*{0.2\textheight}
    {\Huge\bfseries #1}\par\vpace{15mm}
    {\Large\bfseries #2\par\vfill #3}
    \end{center}
}
```

Zkuste odhadnout, co bude příkaz dělat, ještě předtím, než jej vyzkoušíte.

Př. 7.5: Definujme příkaz, který udělá pěknou hlavičku tabulky o třech sloupcích:

```
\def\hlavicka#1#2#3{\begin{tabular}{|c|c|c|}\hline
    \bfseries #1 & \bfseries #2 & \bfseries #3 \\ \hline
}
```

Separované parametry

Velmi užitečnou možností při definici příkazu je použití znaků tvořících oddělovače (separátory) mezi parametry. Této vlastnosti se využívá v řadě aplikací a je to možnost, která není v definičních příkazech LaTeXu vůbec dostupná.

Jak se to dělá? V předchozím případě byly jednotlivé parametry zapsány těsně za sebou, přičemž se předpokládalo, že při použití se budou brát buď jednotlivé tokeny nebo skupiny ohraničené svorkami. Psát parametry ve svorkách je však někdy nepohodlné a někdy dokonce nemožné, protože vstup není tvořen textem editovaným z klávesnice, ale materiálem vzniklým třeba exportem z jiného programu.

Proto je možné *mezi* jednotlivé parametry vkládat znaky (přesněji tokeny), které jsou pak vyžadovány i při použití makra a vymezují materiál chápaný jako příslušný parametr. Vzniká tedy jakási maska parametrů, která se „přiloží“ ke zdrojovému textu a separačními znaky srovná s definicí.

Například definice `\def\bod(#1-#2){[#1~mm; #2~mm]}` má separátory: levá závorka, spojovník a pravá závorka. Při svém použití pak bude vyžadovat stejně rozmístěné separátory a materiál mezi nimi bude chápat jako skutečné parametry. Například zápis `\bod(25-50)` bude vysázen [25 mm; 50 mm].

Separátorem může být i token, a to i příkaz, který nebyl definován. Příklad:

```
\def\tabulka#1:#2\konectab{%                 separátory : \konectab
   \begin{table}[htb]\centering              plovoucí prostředí
     \caption{#1}                            využití standardního popisku
     \begin{tabular}{lcc}\hline
     Vzorek & Objem [l] & Rok \\             jednotná hlavička
     \hline
     #2                                      sem přijdou data
     \\ \hline
     \end{tabular}
   \end{table}
}
```

Příkaz můžeme použít ve zdrojovém textu takto:

Příklad použití

```
\tabulka Vzorky Velké Pavlovice:
   Veltlínské zelené & 2,1 & 2006 \\
   Rulandské šedé    & 1,4 & 2008 \\
   Muškát moravský   & 2,8 & 2010 \\
   Müller Thurgau    & 2,1 & 2009
\konectab
```

Na procvičení:

1. Vytvořte příkaz, který svůj parametr vypíše pětkrát, a to v různých stupních (normální velikost, dvakrát zvětšená a dvakrát zmenšená).

2. Vytvořte příkaz, který svůj parametr jen pohltí.

3. Vytvořte příkaz se dvěma parametry, který je vypíše v obráceném pořadí.
4. Vytvořte příkaz pro sazbu tabulky se dvěma řádky a dvěma sloupci, která bude zarovnána na střed sazby, všechna pole bude mít zarovnána vlevo a bude zcela ohraničena čarami.
5. Vytvořte příkaz pro vysazení obrázku ze zadaného souboru na plnou šíři sazby v plovoucím prostředí s popiskem pod obrázkem. Popisek nebude obsahovat číslo obrázku, jen text psaný kurzívou ve zmenšeném stupni, zarovnaný na střed.
6. Předpokládejte, že existuje soubor exportovaný z Excelu do formátu CSV se třemi údaji na řádku: jméno a příjmení s tituly, funkce, adresa. Navrhněte makro se separovanými parametry, které z těchto údajů vysází navštívenky. Předpokládejte, že do souboru CSV bude název makra doplněn na začátek každého řádku a na konec každého řádku bude doplněn smluvený znak.

Práce s rozměry

Pro zpracování určité situace musíme dost často manipulovat s rozměry jednotlivých objektů. Již víme, že k tomu slouží délkové registry, s těmi lze provádět různé operace a využívat je k nastavování potřebných velikostí.

Kromě předdefinovaných registrů navíc můžeme vytvářet registry vlastní. Přehled možností je uveden v následujícím seznamu:

- Vytvoření registru: `\newlength{\registr}` Po vytvoření má registr automaticky nastavenu nulovou délku.
- Násobení koeficientem: při použití registru lze zapsat například `0.7\linewidth` a dostaneme rozměr odpovídající součinu koeficientu a registru.
- Přičítání do registru: `\advance\registr by` *délka* přičte hodnotu délky do daného registru, délka může být vyjádřena číslem nebo jiným registrem a může mít záporné znaménko, což signalizuje odčítání.
- Násobení v registru: `\multiply\registr by` *hodnota*
- Dělení v registru: `\divide\registr by` *hodnota*

Ve všech příkazech můžeme pomocné slovo „by“ vynechat.

Příklady

Př. 7.6: Potřebujeme do vlastního registru `\delka` nastavit délku řádku sázeného s odstavcovou zarážkou. Zapíšeme postupně:

```
\newlength{\delka}
\delka=\linewidth                 vložení aktuální šíře sazby
\advance\delka by -\parindent     odečtení velikosti zarážky
```

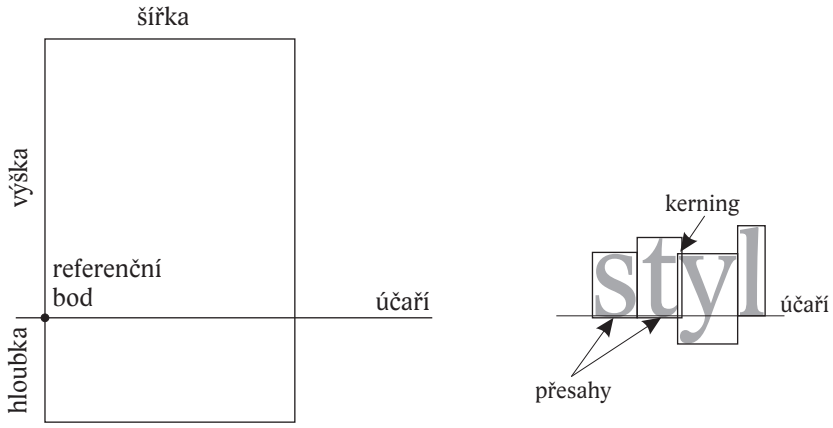
Př. 7.7: Předpokládejme, že máme dva obrázky p1 a p2 o shodné šířce 45 mm, potřebujeme je umístit na levý a pravý okraj a do mezery mezi ně napsat text „srovnejte“ tak veliký, aby mezi textem a obrázky zůstala mezera jen 2 mm.

Mezera mezi obrázky

```
\newlength{\velikost}              nový registr
\velikost=\textwidth
\advance\velikost by -96mm         odečteme obrázky a mezery
\noindent
\includegraphics{p1}\hfill         vysázíme obrázek
\resizebox{\velikost}{!}{srovnejte} upravíme text
\hfill\includegraphics{p2}         vysázíme druhý obrázek
```

Práce s boxy

Základním prvkem sazby v TEXu je tzv. **box** (někdy se překládá jako rámeček, ale pojem box bude pro použití lepší a kratší). Box si můžeme představit jako obdélník s určitou výškou a šířkou a definovanou pozicí účaří – viz obr. 7.1. Celý materiál odstavce se skládá ze tří prvků – boxů, mezer a tzv. **penalt** (trestů). Za normálních okolností se boxy k sobě kladou těsně bez mezer s účařím ve stejné výši. Mezerami nebo příkazy pro změnu této sazby se dá pozice následujícího boxu ovlivnit.



Obr. 7.1: Schéma boxu jako prvku sazby v systému TEX, boxy jednotlivých znaků a ukázka jejich sazby

Boxem může být jednotlivý znak (nejčastěji), ale také více materiálu – obrázek, tabulka nebo také stránka.

Máme-li určitý materiál „zabalen“ do boxu, můžeme s tímto boxem zacházet jako s písmenem a umisťovat ho do různých pozic.

Jak vytvořit box? Boxem může být:

- Jednořádkový materiál vzniklý příkazem `\makebox` nebo `\mbox}` nebo `\hbox`; příklad: `\mbox{SLOVO}`.
- Materiál vložený do úschovné oblasti typu box příkazem `\savebox` (viz dále).
- Materiál ohraničený linkou pomocí příkazu `\fbox`; např. `\fbox{rámeček}`.
- Čárový box vzniklý příkazem `\rule`; např. `\rule{12cm}{0.5pt}`.
- Materiál tabulky v prostředí `tabular`.
- Materiál obrázku v prostředí `picture`.

- Odstavcový materiál vzniklý příkazem `\parbox` nebo prostředím `minipage` (viz dále).
- Box s vertikálním materiálem vzniklý příkazem `\vbox`; např. `\vbox{Jedna\\dvě\\tři}`.

Vidíte, že paleta je široká, jen si vybrat ... A to není ještě všechno. Podíváme se alespoň na některé příklady, pro hlubší studium by v případě zájmu bylo vhodné sáhnout po literatuře uvedené v seznamu na konci textu. Ze zajímavých prvků vybereme:

- Vytvoření úschovného boxu pro materiál zařídíme příkazem `\newsavebox{\nazev}` Do takto vyrobeného boxu pak můžeme vložit obsah příkazem `\sbox{\nazev}{text}` a celý box pak můžeme libovolněkrát vložit do potřebného místa příkazem `\usebox{\nazev}`
- Odstavcové (paragrafové) boxy umožňují uschovat materiál odstavce nebo i více odstavců. Jde o příkazy `\parbox{šířka}{text}` a prostředí `\begin{minipage}{šířka}...\end{minipage}`. Tyto boxy mají implicitně účaří v polovině svého souhrnného výškového rozměru. Volitelnými parametry `[t]` resp. `[b]` však lze účaří posunout na úroveň prvního řádku vnitřního materiálu, resp. na jeho spodní okraj. Podobně lze posunout i vertikální referenční bod v prostředí `tabular`.

Co můžeme s boxy dělat? Zejména ovlivnit jejich umístění – základní změny lze dosáhnout příkazy `\raisebox` (vertikální umístění) a `\kern` (horizontální umístění). Dále můžeme změřit jejich rozměry, uložit si je do registru a dále s nimi pracovat – k tomu slouží příkazy `\wd` (šířka boxu), `\ht` (výška boxu) a `\dp` (hloubka boxu). Některé boxy (čárový, parbox, minipage, tabular) mohou změnit svůj referenční bod a pozici účaří (jak bylo již zmíněno), tím lze tyto boxy umístit vzhledem k okolí do jiné pozice.

Příklady

Př. 7.8: Předpokládejme, že potřebujeme v textu sázet něco jako exponenty nebo indexy, ale nechceme použít matematický režim, protože se jedná o textové komponenty. Například kdysi se používalo pro znázornění typu lokomotivy a jejího inventárního čísla složeného symbolu tvaru $1A^{VII}$. Vytvořme tedy příkaz pro sazbu takového označení. Příkaz bude mít dva parametry – symbol a exponent. Jako základní rekvizitu pro vertikální posun použijeme příkaz `\raisebox`:

Rakousko-Uhersko

```
\def\oznaceni#1#2{#1\raisebox{.5em}{\footnotesize #2}}
```

\raisebox má v prvním parametru zvýšení, ve druhém je box

Vzhledem k tomu, že jsme vertikální posun zadali v relativních jednotkách, bude označení mít stejný tvar bez ohledu na tom, v jaké situaci je použijeme (například veliké v nadpisu nebo malé v poznámce pod čarou).

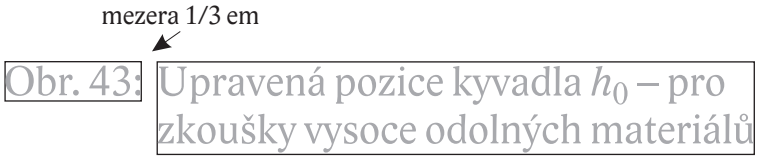
Př. 7.9: Vytváříme formulář, do něhož se budou některé položky doplňovat rukou – nachystáme do příslušného místa tečky. Potřebujeme ale, aby na takové vpisování byl v řádku dostatečný prostor, chceme tedy řádky v tomto místě oddálit. Vytvořme tedy příkaz `\tecky{rozměr}`, který vytvoří tečkovanou čáru o stanoveném rozměru a nad ním vytvoří potřebný prostor (například 5 mm) pro pohodlné vpisování.

Využijeme přitom skutečnosti, že čárový box může mít jeden rozměr nulový, takže čára nebude vysázena, ale druhý rozměr se plně započítává do sazby, jako kdyby tam čára byla. Takový čárový box se v literatuře nazývá strut. Čárový box můžeme využít i pro „odsunutí“ následujícího řádku, a to nepovinným parametrem příkazu posouvajícím box vertikálně:
`\rule[posunutí]{šířka}{výška}` . Definice může mít například tento tvar:

Vpisování do formulářů

```
\def\prostor{\rule{0mm}{5mm}}        čára o nulové šířce a výšce 5 mm
\def\tecky#1{%
   \hbox to #1{\prostor              vložení strutu
       \tiny\dotfill}                jemné tečky v celém boxu
}
```

Př. 7.10: Chceme vytvořit popisky k obrázkům podle schématu na obr. 7.2. Celková šíře obou boxů včetně oddělovací třetinové mezery musí být rovna šíři sazby v aktuálním místě.



Obr. 7.2: Schéma popisku k obrázkům

K řešení tohoto příkladu potřebujeme dvě rekvizity: vytvořit box, do kterého vložíme text „Obr. 43:“, a následně ho změřit, dále pak vypočítat zbytek volného prostoru pro text popisku, vytvořit pro něj odpovídající parbox a vysázet vše do jednoho celku.

Je samozřejmé, že popisek nebudeme sázet přímo, ale vytvoříme pro něj odpovídající příkaz s parametry číslo a text. Výsledek tedy může vypadat například takto:

Řešení sazby popisku

```
\newsavebox{\cislobox}               úschovný box pro číslo obrázku
\newlength{\sirkatxt}                registr pro vypočítanou šířku textu
    uvedené deklarace se uvedou jednou, nejsou zahrnuty do makra
\def\popisek#1#2{% 1=číslo, 2=text
    \sirkatxt=\linewidth
    \sbox{\cislobox}{Obr. #1:}       uložení čísla do boxu
    \advance\sirkatxt -\wd\cislobox  odečtení šířky čísla
    \advance\sirkatxt -0.333em       odečtení mezery = šířka textu
    \usebox{\cislobox}\hfill
    \parbox[t]{\sirkatxt}{#2}        vysázení obou boxů, mezera vyjde
}
```

Př. 7.11: Vytvořme uzavřenou tabulku na celou šíři sazby se dvěma sloupci, oba budou obsahovat souvislý text sázený na střed, přičemž šířky sloupců mají být v poměru 3:2.

Tabulka sázená prostředím `tabular` bohužel implicitně nemá možnosti požadované v takové úloze. Musíme si pomoci parboxy, jejichž přesnou šíři si napřed spočítáme. Vyjdeme přitom z celkové šíře sazby, od které odečteme mezisloupcovou mezeru, výsledek vydělíme pěti a pak pro levý sloupec násobíme třemi, pro pravý sloupec dvěma. Opět na celý objekt vytvoříme makro, jehož dvěma parametry budou texty v levém a pravém sloupci.

Pěkná tabulka

```
\newlength{\sloupec}
\long\def\peknatab#1#2{%                    bude pěkná tabulka
   \sloupec=\linewidth                      aktuální šíře sazby
   \advance\sloupec by -4\tabcolsep         odečtení mezisloupcových mezer
   \divide\sloupec by 5                     prvek šířky
   \noindent                                sazba bez případné zarážky
   \begin{tabular}{|c|c|}\hline
     \parbox[t]{3\sloupec}{#1}  &           levý sloupec, zarovnání nahoru
     \parbox[t]{2\sloupec}{#2} \\\hline     a pravý sloupec
   \end{tabular}
}
```

Připomeňme jen, že tabulka se správně vysází i tehdy, umístíme-li její použití například do položek výčtu (změní se výchozí hodnota `\linewidth`) nebo nastavíme-li v celém dokumentu „vzdušnější“ sazbu tabulek změnou rozměru `\tabcolsep`. V této flexibilitě a efektivitě asi nenajde uvedené řešení vážnějšího konkurenta u nějakých jiných interaktivních systémů ...

Na procvičení:

- Vytvořte příkaz pro sazbu kytarových akordů nad text písně.
- Vytvořte příkaz, který bude vhodný jako pole tabulky – bude sázet odstavcový materiál na prapor (se zarovnaným levým okrajem) na zadanou šíři s vertikálním zarovnáním k hornímu účaří. Zároveň bude zajištěno, že za dolním okrajem a před horním okrajem bude vždy vertikální mezera alespoň 7 bodů.
- Vytvořte příkaz pro sazbu tabulky se dvěma sloupci, první sloupec bude mít šířku čtvrtiny šíře sazby, druhý sloupec šíři dvou třetin šíře sazby, texty ve sloupcích budou sázeny na prapor a budou moci obsahovat materiál složený z více odstavců. Tabulka bude orámována čarami. Pro vnitřní materiál můžete využít příkazu definovaného v předchozí úloze.
- Vytvořte příkaz pro sazbu zastávkového jízdního řádu, který má v prvním sloupci pod sebou názvy zastávek a ve druhém sloupci ke každé zastávce seznam odjezdových časů. Využijte tohoto příkazu k sazbě celého jízdního řádu jedné trasy MHD. Mezi zastávkami a časy je svislá oddělovací čára.

Číslování

V různých částech dokumentu se objevují celočíselné hodnoty patřící k nějakým prvkům – stránkám, položkám výčtů, rovnicím, nadpisům, poznámkám apod. S těmito hodnotami lze nejen pracovat a ovlivňovat je kýženým způsobem, ale lze si vyrábět vlastní a používat je pro zcela neomezené aplikace.

Jde o registry zvané **čítače** (angl. counters). Jsou nejčastěji používány pro skutečné čítání po jedné (čísla stránek, poznámky, položky výčtů). Ale jako celočíselné proměnné je lze používat i jinak – pro uchování hodnot a výpočty, například pro pozice objektů v prostředí `picture`.

Předdefinované čítače mají svá jména obvykle spojena s objektem nebo prostředím, pro něž jsou používány – určitě uhodnete, na co jsou čítače například `page`, `footnote`, `section`. Vlastní

čítače mohou mít jména libovolná, ale nepodaří se nám vyrobit vlastní čítač s jménem, které už existuje. Co tedy můžeme všechno v této oblasti udělat?

- Vytvoření nového čítače – `\newcounter{muj}`. Po vytvoření má každý čítač automaticky hodnotu nula. Čítače můžeme deklarovat do závislých kaskád – čítač závislý na nějakém nadřízeném čítači vynuluje svou hodnotu, když se nadřízení posune o jedničku. Tuto závislost definujeme při vytvoření nepovinným parametrem: `\newcounter{muj}[section]` – hodnota čítače `muj` se vynuluje, když se čítač sekcí zvýší o jedna, tedy v každé sekci „začínáme od nuly“. Při definici nového závislého čítače musí nadřízený čítač již existovat, nemusí se ale jednat o čítač předdefinovaný.
- Vložení hodnoty do čítače – `\setcounter{muj}{20}`.
- Přičtení hodnoty do čítače – `\addtocounter{muj}{-5}`. Kladná hodnota zvyšuje, záporná snižuje hodnotu čítače. Změna hodnoty neovlivňuje případné podřízené čítače.
- Přičtení jedničky do čítače – `\stepcounter{muj}`, zároveň se vynulují podřízené čítače.
- Přičtení jedničky do referenčního čítače – `\refstepcounter{muj}`, operace je stejná jako v předchozím případě, ale takto obsluhovaný čítač má hodnotu, na niž se lze odvolávat příkazem `\ref`, případně příkazem `\pageref`.
- Získání vysázené hodnoty čítače – `\themuj`; s každým novým čítačem automaticky vzniká příkaz `\thenázev`, provádějící zobrazení hodnoty čítače do textové podoby vhodné pro sazbu. Implicitně se čítač vypisuje arabskými číslicemi, ale to jde změnit redefinicí příkazu `\the...`, například:
  `\def\themuj{\Roman{muj}}` nebo `\def\themuj{\thesection:\Alph{muj}}`
  (v prvním případě se čítač bude vypisovat římskými číslicem, ve druhém bude vypisována kombinace tvaru čítače sekcí, oddělovací dvojtečka a písmenná reprezentace čítače `muj`).
- Získání vnitřní celočíselné hodnoty čítače – `\value{muj}`. To je použitelné všude tam, kde se celočíselná hodnota očekává, například
  `\addtocounter{muj}{\value{section}}` nebo
  `\put(\value{muj},0){\line(1,0){\value{muj}}}`

Příklady

Př. 7.12: Jak dosáhneme toho, aby se v číslování stran přeskočilo 10 stránek?
Řešení: `\addtocounter{page}{10}`

Př. 7.13: Potřebujeme dosáhnout stavu, kdy se čísla rovnic vypisují ve tvaru číslo kapitoly, dvojtečka, písmenné označení rovnice. Lze toho dosáhnout? Řešení spočívá v redefinici výpisu čítače `equation`:
`\def\theequation{\thechapter:\alph{equation}}`

Př. 7.14: Potřebujeme vysázet tabulku, která je rozdělena do pěti částí. Každá část bude mít stejný nadpis jako část první, u druhé a další části bude za nadpisem v závorce slovo „pokračování“ a číslo tabulky bude u všech částí stejné. Jak to můžeme elegantně zařídit?
Řešení: Pro nadpis první tabulky použijeme například následující příkaz:

```
\def\zacatektab#1{\gdef\pokracnadpis{#1}     úschova nadpisu do makra
   \caption{#1}                               zobrazení nadpisu
   příkaz \caption ovšem zároveň posune číslování tabulek
}
```

Pro nadpisy pokračujících částí pak použijeme tento příkaz:

```
\def\pokractab{%
       tady už parametr není -- nebudeme přece název opisovat...
    \addtocounter{table}{-1}        vrátíme čítač o jedno zpět
    \caption{\pokracnadpis\ pokračování}
                                    a necháme vypsat název z makra
}
```

Na procvičení:

- Přepokládejte, že na každé stránce dokumentu je nejvýše jedna poznámka pod čarou. Zařiďte, aby se místo průběžných čísel odkaz na poznámku pod čarou vypisoval ve formě hvězdičky.
- Změňte označování čísel kapitol v dokumentu na písmena velké abecedy a čísel sekcí na velké římské číslice.
- Vytvořte vlastní čítač pro označování nových pojmů v učebnici. Využijte jej v příkazu pro výpis pojmu, který vypíše vypíše pojem v parametru tučně a za ním do závorky obyčejným řezem pořadové číslo pojmu.

Kategorické fígle

Zmínili jsme již, že při vstupu textu do překladače se každý token označí jistou kategorií. Těchto kategorií je 16 a ke znaku každé kategorie se pak překladač chová při zpracování jinak. Například kategorie nula obsahuje tzv. únikový znak – začátek příkazu (implicitně obrácené lomítko), kategorie jedna je otevření skupiny (levá složená závorka) atd. Všech 16 kategorií s podrobným popisem lze nalézt ve velmi důkladné Olšákově knize TEXbook naruby (2003). Znaky lze mezi kategoriemi přeřazovat, čímž lze zcela změnit chápání vstupního textu. Toho lze využít například v případě, že text získáváme z nějakého automatického zdroje a nechceme do něj vpisovat příkazy. Druhým markantním příkladem je tento text. Možná jste si při čtení položili otázku: jak je možné, že se v tom barevném rámečku vyskytují obrácená lomítka, svorky a další vymoženosti, aniž by se interpretovaly jako části příkazů? To je také hra změn kategorií – všechny tyto znaky jsou totiž v rámečku změněny na kategorii písmen, která se obyčejným způsobem zobrazí. Aplikací tohoto mechanismu je však mnohem víc a alespoň něco v tomto textu ještě ukážeme.

Základním příkazem, který vše zařizuje, je `\catcode`. Jeho zápis bude patrný z příkladů použití. Měněný znak se zapisuje sekvencí obrácený apostrof – obrácené lomítko – znak, pak následuje nepovinné rovnítko a číslo kategorie. Příkaz má platnost uvnitř skupiny, takže změny lze pohodlně ohraničit.

Příklady

Př. 7.15: Chceme v dokumentu, který neobsahuje žádnou matematiku, ale zato hodně dolarů, pohodlně psát tyto dolary bez povinného obráceného lomítka. Jak to uděláme?
`\catcode`\$=11` – teď má dolar kategorii obyčejného písmene a můžeme jej zapisovat bez problémů a bez obráceného lomítka kamkoliv do textu.

Př. 7.16: Chceme v dokumentu jednoduše zapisovat příkaz, který uvnitř tabulkového pole udělá strut nahoru 15 bodů a dolů 10 bodů. Bylo by samozřejmě možné do každého potřebného místa vypsat příkaz

```
\rule[-10pt]{0pt}{25pt}
```

To by ovšem zdrojový text vypadal velmi nepřehledně a navíc při jakékoliv změně bychom museli třeba dvacet výskytů opravovat. Takže uděláme příkaz, například

```
\def\mujstrut{\rule[-10pt]{0pt}{25pt}}
```

To už je lepší, ale pořád je to dost dlouhé. I kdybychom to zkrátili třeba na `\ms`, píšeme minimálně tři znaky a z toho ještě jedno obrácené lomítko. Co kdybychom ale nějaký „nepotřebný“ znak předefinovali na příkaz? Jde to vůbec? Sugestivní otázka vynucuje pochopitelně kladnou odpověď – každý znak můžeme přehodit do kategorie tzv. aktivních znaků (13) a tam s nimi můžeme provádět definice, jaké chceme. Nepotřebným znakem může být v rámci nějaké tabulky třeba zavináč, lomítko, vykřičník apod. Vybereme lomítko a provedeme:

Kterak lomítko k příkazu přišlo

```
\catcode`/=13                          lomítko je teď aktivním znakem
\def/{\rule[-10pt]{0pt}{25pt}}         přidělíme mu význam
\begin{tabular}{c}\hline
/ širší řádek \\ \hline                a vyzkoušíme
  užší řádek  \\ \hline
\end{tabular}
```

Na procvičení:

- Kategorie 1 a 2 znamenají začátek a konec skupiny. Zajistěte, aby se na začátek a konec skupiny daly použít znaky „menší než“ a „větší než“, pak svorky přeměňte na obyčejná písmena (kategorie 11).
- Nadefinujte příkaz tvořený znakem vykřičník, který vytvoří mezeru o šířce číslice nula. Mezeru o šířce nějakého vysázeného textu můžete vytvořit příkazem `\hphantom{text}`.
- Předpokládejte, že do zdrojového textu se vloudily entity jazyka HTML ve tvaru `&#8224;` a potřebujete zajistit, aby překlad na těchto sekvencích nehlásil chybu. Jak to uděláte? Jak zajistíte, aby se taková entita vůbec nezobrazila na výstupu? Předpokládejte, že kódová čísla mohou být různá.

Kapitola 8

Sázecí styly, balíčky a třídy

Přikazy potřebné v dokumentu lze definovat (a je to velmi výhodné – viz koncept strukturního značkování) v souboru mimo dokument. Jaké možnosti v systémech typu LaTeX máme k dispozici?

Samozřejmě je zase více možností:

- Vytvořit obyčejný textový soubor s definicemi, pojmenovat ho libovolně a příkazem `\input{soubor}` jej vložit do potřebného místa v dokumentu.
- Vytvořit soubor s definicemi – balíček a pojmenovat ho názvem s rozšířením `.sty`, tento soubor pak připojit k dokumentu příkazem `\usepackage` v preambuli dokumentu. Starší název balíčku je styl – odsud je i rozšíření `.sty`.
- Vytvořit třídu dokumentu – soubor s definicemi s názvem a rozšířením `.cls`, tento soubor připojit k dokumentu příkazem `\documentclass`.

Z hlediska tvorby příkazů jsou všechny možnosti prakticky rovnocenné, liší se relativními drobnostmi: třída dokumentu by měla být nejobecnější a měla by zahrnovat širokou skupinu podobných dokumentů. Balíček je obvykle koncipován jako rozšíření určité třídy, zatímco obyčejný vkládaný soubor by měl představovat jen specifické definice příkazů použitelných v určité instanci dokumentu.

Třída a balíček mají společnou vlastnost – jejich definice jsou k dispozici již v preambuli dokumentu. V této souvislosti se využívá jednoho pěkného triku s kategorií znaku @: V době, kdy jsou překladačem načítány třídy a balíčky, má znak @ třídu 11 – písmeno. Ve vlastním dokumentu ale dojde k záměně kategorie na třídu 12 – ostatní znaky, což znamená, že tento znak již nelze běžně použít v názvech příkazů. Vytvoříme-li tedy ve třídě nebo v balíčku nový příkaz s názvem například `\muj@nadpis`, bude všechno v pořádku, ale tento příkaz nebude možné použít v dokumentu (samozřejmě se může použít jako součást jiných příkazů). Proto se příkazy s názvy obsahujícími zavináč někdy nazývají **vnitřní** a tímto jednoduchým trikem se zabrání tomu, aby (neznalý) uživatel (nechtě) použil tento vnitřní příkaz pro svoje potřeby. Znalý uživatel ví, že si má změnit kategorii zavináče a může použít cokoliv kdekoliv ...

Podíváme se alespoň stručně na tvorbu třídy dokumentu. Třída (stejně jako balíček) může být do dokumentu zavedena s volitelnými parametry, které musí být nějak obslouženy. Obvykle je používáme k upřesnění některých nastavení nebo k modifikaci činnosti vybraných příkazů. Příkladem je volba `twoside` standardní třídy `article`. Jaké příkazy můžeme uvnitř třídy (a s odpovídajícími omezeními i uvnitř balíčku) používat?

- `\NeedsTeXFormat{formát}` – požadavek na zpracování určitým formátem (soubor `.fmt`). Pokud není tento nebo novější formát k dispozici, ohlásí se chyba a zpracování dále neprobíhá. Je to obrana proti překladu novějších verzí zdrojových textů ve starších instalacích, kde chybí řada vymožeností.

- `\LoadClass` – zavede jinou třídu, na základě které bude pravděpodobně vytvářená třída postavena (využije se maximum již odladěných příkazů).
- `\PassOptionsToClass{volby}{třída}` – zavede volby do vkládané třídy (nahrazuje volitelný parametr běžného příkazu `\documentclass`).
- `\ProvidesClass{jméno}` – deklaruje, že tento soubor obsahuje příkazy dané třídy. Informace se využívá při použití této třídy v jiné třídě.
- `\RequirePackage` – zavede balíček. Uvedené dva příkazy jsou silně doporučeny místo obdobného nízkoúrovňového příkazu `\input`. Umožňují totiž například sledovat všechny soubory, hlídají vícenásobné zavedení apod.
- `\DeclareOption{volba}{příkaz}` – definuje zpracování volby, pokud je použita.
- `\PassOptionsToPackage{volby}{balíček}` – vloží volby do volaného balíčku, nahrazuje volitelný parametr příkazu `\usepackage`.
- `\ProcessOptions` – nařizuje zpracování podle definic uvedených v předchozích příkazech `\DeclareOption`.

Jako příklad můžeme uvést příkazy v třídě, kterou budeme chtít využít pro naši sbírku úloh z fyziky. Jako možnou volbu implementujme řetězec `bezvysledku`, jehož uvedením se nebudou vypisovat výsledky úloh.

Příkazy uvedené na začátku souboru `sbirka.cls` by mohly mít například následující tvar:

sbirka.cls

```
\NeedsTeXFormat{LaTeX2e}                    použitelné i pro XeLaTeX
\ProvidesClass{sbirka}
\PassOptionsToClass{twoside}{article}
\LoadClass{article}                         použití bázové třídy s volbami
\RequirePackage{xltxtra}                    nutné pro možnosti XeLaTeXu
\RequirePackage[czech]{babel}               čeština
\DeclareOption{bezvysledku}{\def\vysledky{\relax}}
\ProcessOptions                             zpracování případné volby
```

Kapitola 9

Zpracování výstupů z jiných zdrojů

Nástroje, které jsme v předchozích kapitolách představili, můžeme použít na frekventovanou úlohu – zpracování dat pocházejících z různých zdrojů: jiných programů pro zpracování textů, databázových programů nebo tabulkových procesorů. Na malém prostoru, který je pro toto široké téma k dispozici, se zmíníme alespoň o jednom představiteli v každé z uvedených kategorií.

- Zdroj z jiného produktu pro zpracování textů má obvykle základní vlastnost – při konverzi se zcela ztrácejí strukturní informace a zůstávají pouze vizuální (nastavení řezu a stupně písma, vypočítané přesné rozměry tabulkových polí apod.). Chceme-li takový zdroj použít k sazbě kvalitního díla, obvykle to představuje většinu značek ručně přepsat na jiné. Nejde to automatizovat, protože musíme například rozhodnout, zda nastavené tučné písmo je vyznačení v textu, nadpis, hlavičkové pole tabulky apod. Mnohdy bez autorských informací ani kvalitně rozhodnout nemůžeme.

  Z technického hlediska je důležité navrhnout příkazy, kterými se původní značky budou nahrazovat, tedy navrhnout odpovídající styl nebo třídu.

  Jako specifickou situaci je možné chápat zdroj v podobě dokumentu v jazyce HTML. Teoreticky by bylo možné pomocí vhodného nastavení kategorií znaků a odpovídajícími definicemi takový soubor přímo přeložit.

- Zdroj z databázového programu – často může jít o textový soubor, jehož každý řádek obsahuje informace o jednom celku (osoba, skladová položka, provedená účetní operace apod.). Úkolem takového zpracování je vizualizace každého řádku v nejrůznějších podobách – dopisech, katalozích, sestavách atd.

  V každém začátečnickém kursu manipulace s běžnými textovými procesory je řešena úloha hromadné korespondence. Tato úloha tematicky spadá do této kategorie a její řešení již bylo zmíněno v úloze s tvorbou navštívenek. Připomeňme si je na příkladu poněkud jiného druhu:

  Nechť zdrojový soubor s názvem `data.txt` obsahuje na každém řádku název předmětu, jeho inventární číslo a zůstatkovou cenu. Předpokládejme, že původně byl soubor získán s oddělovači v podobě svislých čar mezi položkami, na začátku a na konci řádku, ale jednoduchou editací (třeba i s možností automatické náhrady) jsme soubor upravili na tento tvar:

Příklad vstupu

```
\predmet Skříň dvoudílná | 3589 | 487,50 |
\predmet Stůl pracovní   | 6657 | 521,00 |
```

  Úkolem je vypsat informace na samolepicí štítky, kterými se při inventarizaci jednotlivé předměty polepují.

Samolepky

```
\parindent 0pt          štítky budou vypisovány do řádku jako text
\unitlength 1mm
\def\predmet#1|#2|#3|{\begin{picture}(64,21.4)     přesné rozměry
   \put(32,18){\makebox(0,0)[t]{\bfseries #1}}  výpis názvu
   \put(32,9){\makebox(0,0)[t]{\ttfamily #2}}   výpis inv. čísla
   \put(32,1){\makebox(0,0)[b]{\footnotesize #3 Kč}}
   \end{picture}
}
\input{data.txt}                zobrazení samolepek
```

- Zdroj dat z tabulkových procesorů (nebudeme-li uvažovat úlohu podobnou té předchozí) lze využít pro sazbu tabulek. Princip bude obdobný, jen řešení rozdílné. Ilustrativním příkladem může být soubor `tab.csv` formátu CSV s oddělovacími středníky obsahující na každém řádku stejný počet polí (název odběratele, číslo zboží, počet kusů, cena za kus). Středníky jsou po exportu jen *mezi* údaji, takže editací dodáme název makra na začátek každého řádku a ještě jeden středník na konec každého řádku.

  Zpracovávající příkaz (například `\dotabulky`) může mít tuto definici:))

Tabulkový řádek

```
\newcounter{odber}                                  čítač pro poř. č.
\def\dsvetlo{\rule[-6pt]{0pt}{10pt}}                strut dolů
\def\hsvetlo{\rule{0pt}{15pt}}                      strut nahoru
\def\dotabulky#1;#2;#3;#4;{                         separované parametry
   \stepcounter{odber}\theodber. &                  pořadové číslo
   \parbox[t]{0.25\textwidth}{\raggedright          odhad šířky textu
             \hsvetlo#1\dsvetlo}                    dostatečný prostor
   & \hsvetlo#2 & \hsvetlo#3 & \hsvetlo#4 Kč \\
   \hline
}
```

S využitím uvedeného příkazu pak lze vysadit celou tabulku:

Tabulka

```
\def\thlav{\bfseries}                               font záhlaví
\begin{tabular}{|r|l|l|r|r|} \hline
    \thlav P. č. & \thlav Odběratel & \thlav Zboží
    & \thlav ks & \thlav Cena/ks \\ \hline
    \input{tab.csv}
\end{tabular}
```

Na procvičení:

- Napište potřebné příkazy pro vytvoření dopisu, do jehož adresy, oslovení a textu se budou vkládat údaje získané z databázového systému. Zvolte vhodně formát předávaných dat, námět a uspořádání dopisu.

Návrh dokumentního typu a odpovídajícího stylu

S informacemi, které nyní máme k dispozici, se směle můžeme pustit do tvorby třídy `sbirka`, pomocí níž vysadíme sbírku úloh z fyziky zmiňovanou jako příklad v kap. 6. Částečně jsme v některých příkladech už některé prvky použili, dáme je nyní do jednoho souboru a doplníme zbytek. Zůstaneme u jednoduchého tvaru, ilustrujícího jen základní princip.

Celý soubor definic je vhodné členit do určitých logických oddílů, aby se pak dobře hledaly definice, které mají nějaké společné poslání. V první části uvedeme příkazy patřící do definice samotné třídy, zpracování voleb a zavedení obecných balíčků:

Sbírka (začátek)

```
\NeedsTeXFormat{LaTeX2e}                    použitelné i pro XeLaTeX
\ProvidesClass{sbirka}
\PassOptionsToClass{twoside}{article}
\LoadClass{article}                         použití bázové třídy s volbami
\RequirePackage{xltxtra}                    nutné pro možnosti XeLaTeXu
\RequirePackage[czech]{babel}               užitečné balíčky: čeština
\RequirePackage{graphicx}                   grafika
\RequirePackage{xcolor}                     barvy
\DeclareOption{bezvysledku}{\def\vysledky{\relax}}
\ProcessOptions                             zpracování případné volby
```

Pokračování může obsahovat celkové parametry sazby:

Parametry dokumentu

```
\textwidth=125mm                    rozměry
\textheight=185mm
\oddsidemargin=5mm
\evensidemargin=15mm
\raggedbottom                       parametry sazby odstavců
\parindent 0em
\parskip .5\baselineskip
\tolerance=5000
\widowpenalty=10000
\clubpenalty=10000
\hyphenpenalty 5000
\brokenpenalty=10000
\def\topfraction{.9}                parametry plovoucích prostředí
\def\textfraction{.1}
\def\floatpagefraction{.85}
```

Je velmi vhodné soustředit definice písem do jednoho místa, protože máme pak přehled o jednotlivých písmových proporcích a můžeme je měnit v celém dokumentu.

Písma

```
\defaultfontfeatures{Mapping=tex-text}
\setmainfont{Cambria}
\setsansfont{Calibri}
\setmonofont[Scale=0.9]{Consolas}
\setmathsfontDigits{Cambria}                    písma pro matematiku
\setmathsfontLatin{Cambria}
\setmathsfontGreek{Cambria}
\setmathrm{Cambria}

\def\footnotefnt{\fontsize{8pt}{9pt}\selectfont}   písma pro různé prvky
\def\pagefnt{\fontsize{8pt}{10pt}\selectfont\bfseries}
\def\headfnt{\fontsize{8pt}{10pt}\selectfont}
\def\oddilfnt{\fontsize{25pt}{30pt}\selectfont\bfseries}
\def\oddilnazevfnt{\fontsize{20pt}{24pt}\selectfont\bfseries}
\def\sekcefnt{\fontsize{14pt}{18pt}\selectfont\bfseries}
\def\captfnt{\fontsize{8pt}{9pt}\selectfont\itshape}
```

Sazba stránek – často je potřeba upravit styl číslování stránek nebo údaje vypisované do živých záhlaví. Příklad ukazuje změnu definic příkazů ze standardní třídy `article` (názvy příkazů se zavináči ukazují na vnitřní makra). Jednu z možností můžeme uvést:

Živá záhlaví

```
\def\ps@sbirka{\def\@oddfoot{\relax}\def\@evenfoot{\relax}  paty prázdné
   \def\@oddhead{\parbox{\textwidth}{{\headfnt\rightmark}    pravé záhlaví
           \hfill{\pagefnt\thepage}\vskip3pt\hrule}}
   \def\@evenhead{\parbox{\textwidth}{{\pagefnt\thepage}     levé záhlaví
           \hfill{\headfnt\leftmark}\vskip3pt\hrule}}}
\pagestyle{sbirka}        nastavení nadefinovaného stylu stránek
```

Oddíly a sekce – zde je potřebné také definovat způsob číslování, návaznosti čítačů a jejich použití v příslušných nadpisech:

Členění textu

```
\newcounter{oddilnum}                                  potřebné čítače
\newcounter{sekcenum}[oddilnum]
\def\thesekcenum{\theoddilnum.\arabic{sekcenum}}

\def\oddil#1{\clearpage\refstepcounter{oddilnum}   oddíl textu
   \section*{\parbox{\textwidth}{\raggedright
             \Huge\theoddilnum\\[5mm]#1}}
   \vskip 15mm
   \addcontentsline{toc}{section}{\theoddilnum\quad #1}
   \markboth{\theoddilnum\ #1}{\theoddilnum\ #1}}

\def\sekce#1{\refstepcounter{sekcenum}                 sekce
   \subsection*{\Large\thesekcenum\quad #1}%
   \addcontentsline{toc}{subsection}{\thesekcenum\ #1}
   \markright{\thesekcenum\ #1}}
```

Sazba počátečních stran sbírky souvisí s příkazem pro titulní stranu, vydavatelský záznam a obsah. Zde se jedná o poměrně jednoduché definice:

Začátek sbírky

```
\def\titul#1#2#3{\thispagestyle{empty}
   \begin{center}
   \vspace*{0.2\textheight}
   {\Huge\bfseries #1}\par\vpace{15mm}
   {\Large\bfseries #2\par\vfill #3}
   \def\autortextu{#2}                    úschova informace
   \def\vroceni{#3}
   \end{center}
}
\def\vydzaznam#1{\newpage                 parametrem je ISBN
   \thispagestyle{empty}
   \vspace*{\fill}
   \noindent © \autortextu, \vroceni\\    použití uložených informací
   #1
}
\def\obsah{\newpage\section*{Obsah}       nadpis obsahu
    \markboth{Obsah}{Obsah}
    \@starttoc{toc}}                      vlastní výpis obsahu
```

Hlavním příkazem, který bude použit ve sbírce nejčastěji a bude tvořit její hlavní materiál, je výpis úlohy. Proto mu věnujeme další část. Zde je ještě jeden nedořešený problém – přenos výsledků úlohy do souhrnu na konci publikace. V tomto jednoduchém tvaru to vyřešíme prozatím tajemným příkazem `\dovysledku`, napovíme, že se tento problém řeší zápisem do pomocného souboru a odkážeme čtenáře na literaturu. K úloze ještě patří obrázek a nápověda. Opět použijeme jen velmi jednoduché tvary:

Úloha

```
\newcounter{ulohanum}[oddilnum]           číslování úloh
\def\theulohanum{\theoddilnum.\arabic{ulohanum}}
\def\uloha#1#2{\par\medskip\noindent
   \refstepcounter{ulohanum}
   {\bfseries Úloha \theulohanum:} #1
   \dovysledku{\theulohanum}{#2}
}
\def\obrazek#1#2{\begin{figure}\centering
   \includegraphics{#1}
   \caption{#2}\label{#1}                 název souboru je i návěštím
   \end{figure}
}
\def\napoveda#1{\par\smallskip\noindent
   {\bfseries Nápověda:} #1
}
```

Kapitola 11

Existující styly a jejich rozbor

Velkým zdrojem inspirace a možností řešení nejrůznějších situací jsou styly, balíčky a třídy, které jsou k dispozici v každé distribuci a v rozsáhlých internetových zdrojích. Z praktických zkušeností vyplývá, že velmi často stačí převzít nějakou definici z používané třídy `article.cls`, zkopírovat ji do vlastního stylu a mírně upravit.

V souhrnném příkladu předchozí kapitoly jsme měli možnost jednu z takových úprav vidět – redefinice živých záhlaví. Dalším frekventovaným případem je úprava způsobu sazby položek obsahu dokumentu – příkazy pro jednotlivé úrovně mají názvy `\l@section`, `\l@subsection` atd.

Jednoduchým balíčkem vhodným k prozkoumání může být například `dipp.sty` určený pro sazbu kvalifikačních prací na Provozně ekonomické fakultě Mendelovy univerzity v Brně. Je k dispozici na internetové adrese

`http://akela.mendelu.cz/~rybicka/prez/zpract/zavprace/dipp.sty`

Styl je v kódování ISO Latin 2, na stejném místě je ještě `dipp2.sty` v kódování PC Latin 2 a `dipp7.sty` v kódování Windows 1250.

V balíčku si můžeme všimnout těchto prvků:

- Stránkování je připraveno ve dvou variantách – plain a headings. Obě varianty předefinují standardní makropříkazy z třídy `article.cls`.
- Skupina příkazů pro sazbu úvodních stránek – titulní stránky, poděkování, prohlášení, abstrakty.
- Redefinice příkazů pro nadpisy `\section` atd.
- Redefinice seznamu použité literatury.
- Nastavení a změny parametrů sazby seznamů a výčtů. Oproti předdefinovanému americkému typografickému úzu je zde použit evropský přístup s menšími mezerami.
- Systém sazby obrázků a tabulek včetně popisků a dalších prvků – vkládání obrazových souborů, tabulkové nástroje na vnitřní pole a sazba figurálních mezer redefinicí vykřičníku.

Na procvičení:

- Najděte v souboru `article.cls` definici sazby titulní stránky. Změňte ji tak, aby hlavní nadpis byl sázen modře.
- Najděte v tomtéž souboru definici nadpisů sekcí, podsekcí atd. a zjistěte, jak se změní sazba, pokud změníte hodnotu ve čtvrtém parametru makra `\@startsection` ze záporné na kladnou.
- Prozkoumejte soubor `latex.ltx` a pokuste se najít definici `\@startsection`. Ověřte tam skutečnost, kterou jste zjistili v předchozí úloze.

Literatura

- Lamport, L. *LaTeX – A Document Preparation System: User's Guide and Reference Manual*. Reading: Addison-Wesley Publ. Comp., 1994.
- Goosens, M., Mittelbach, F., Samarin, A. *The LaTeX Companion*. Reading: Addison-Wesley, 1994.
- Olšák, P. *Typografický systém TeX*. Brno: Konvoj, 2000.
- Olšák, P. *TeXbook naruby*. Brno: Konvoj, 2006.
- Rybička, J. *LaTeX pro začátečníky*. Brno: Konvoj, 2003.
- Rybička, J., Čačková, P., Přichystal, J. *Průvodce tvorbou dokumentů*. Bučovice: Nakladatelství Stříž, 2011.
- Talandová, P. *Základy sazby dokumentů v systému LaTeX*. Učební texty k semináři. Brno: VUT v Brně, 2011.

Centrum pro rozvoj výzkumu pokročilých řídicích a senzorických technologií
CZ.1.07/2.3.00/09.0031

Ústav automatizace a měřicí techniky
VUT v Brně
Kolejní 2906/4
612 00 Brno
Česká Republika

http://www.crr.vutbr.cz
info@crr.vutbr.cz