Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner
LaTeXová kuchařka/2

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 6 (1996), No. 4, 269–289

Persistent URL: http://dml.cz/dmlcz/149774

Terms of use:

© Československé sdružení uživatelů TeXu, 1996

Institute of Mathematics of the Czech Academy of Sciences provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.

This document has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ: The Czech Digital Mathematics Library* http://dml.cz

LaTeXová kuchařka/2

Zdeněk Wagner

Otvíráme druhý díl LaTeXové kuchařky. Minule jsme se zaměřili na základy, o nichž se již dále zmiňovat nebudeme. Předpokládáme, že čtenář tohoto dílu již ví, jak se má zachovat, když se mu v dokumentu objeví chyba `Invalid use of \spacefactor`.

Nyní se již začneme zabývat metodami, s jejichž pomocí budeme zasahovat do vzhledu dokumentu. Povíme si o problematice, která v plainu není řešena. Zatímco uživatel plainu musí nadefinovat vše sám, LaTeXistovi stačí pouhá úprava několika málo maker.

Abychom zdůraznili, že se jedná o pokračování, navážeme číslování kapitol. Trochu předběhneme výklad a ukážeme, jak to bylo provedeno. Minulý díl končil kapitolou 7. Proto je na začátku tohoto dokumentu:

```
\setcounter{section}{7}
```

Příkaz `\section` zvětší tento čítač o jednotku a příslušnou hodnotu vytiskne. Tím jsme dosáhli, že následující kapitola má číslo 8.

8. Změna písma

Tento text je zařazen jako reakce na problémy s CSTeXem. V počátku byly do tohoto balíku zařazeny virtuální skripty počeštěných PostScriptových fontů včetně `.fd` souborů pro LaTeX. Omylem však CSTeX neobsahoval balík PSNFSS. Uživatelé pak došli k závěru, že PostScriptové fonty nelze v LaTeXu použít. To byla ovšem pravda pouze částečná. Nestačilo napsat `\usepackage{times}`, neboť příslušný balík nebyl instalován. Řešením byl povel:

```
\fontfamily{ptm}\selectfont
```

Nyní si vysvětlíme, jak a proč výše uvedený příkaz funguje.

8.1. Charakteristiky písma

Typografie rozeznává kromě velikosti tři další charakteristiky písma. LaTeX přidává jeden atribut navíc, a o tom si povíme nejdříve. Snad

každý uživatel TEXu se ve svých začátcích pokusí napsat v textu znaky < a >. Výsledkem je ovšem ¡ a ¿. V matematických vzorcích mu správně fungovalo < i >. Uživatel je tedy zmaten. Na vině je právě ten čtvrtý atribut, to je kódování fontu.

Představte si, že by klávesy na vašem počítači byly jednoznačně očíslovány, například pořadovými čísly. Takovému očíslování můžeme říkat kódování. Vskutku má každá klávesa interně přiřazeno nějaké číslo – kód. Při stisku klávesy je tento kód vyslán do počítače a příslušné vstupní obvody tak poznají, kterou klávesu uživatel zmáčkl. Pokud se má nějaký znak zobrazit na obrazovce či vytisknout na tiskárně, musí se opět vyslat pořadové číslo tohoto znaku v příslušném fontu. Přitom počítač kóduje znaky jinak než klávesnice a kód znaků ve fontu může také být zcela odlišný. Navíc je kódování různých fontů obecně různé. Zpracování dokumentu v (LA)TEXu[1] tedy zahrnuje několik konverzí. Nejprve operační systém převede kód klávesy na kód, kterému rozumí. To se odehrává zejména při psaní zdrojového textu editorem. TEX pak při čtení souboru konvertuje znaky do svého vnitřního kódování. Před tiskem opět provádí konverzi podle kódování použitého fontu. Pokud mají všechny fonty stejné kódování, můžeme TEX naučit pracovat přímo v kódování fontu. Druhá konverze tím odpadne. Pokud používáme pouze sedmibitový ASCII kód nebo máme na úrovni operačního systému stejný kód, jaký mají naše fonty (to je případ UNIXu a kódování IL2), nepotřebujeme konverzi žádnou.

Většina uživatelů si zřejmě vystačí s jediným kódováním textu. Připusťme však, že musíme do textu propašovat větu: „В системе LATEX возможно писать по русски.“ Je zřejmé, že азбука má některé znaky odlišné, takže i jejich umístění ve fontu (tj. kódování) bude jiné. Uvedený příklad lze vytisknout i beze změny kódování, ale totéž neplatí o fontech matematických. Zkuste si například vysázet nějaký text (nejlépe anglický) fontem `cmex10`. Ve výsledku asi původní text nepoznáte. Proto se matematické výrazy zapisují speciálními makry a konverzi do kódu fontu provádí TEX během jejich expanze.

Původně byl TEX vytvořen pro angličtinu a používal tedy pouze sedmibitový kód. Písmena s diakritikou v něm nebyla obsažena. Proto se „ř“ muselo skládat ze samostatného háčku a písmene „r“. Toto sedmibitové

[1] Tímto novotvarem autor vyjadřuje, že příslušné tvrzení platí jak pro plain TEX tak pro LATEX.

kódování se nyní označuje OT1. Později byl vytvořen osmibitový kód obsahující všechny potřebné akcentované znaky. Autoři LaTeXu očekávají, že jeho kódování, označované T1, se stane standardem. Rozložení znaků v kódech OT1 a T1 je dosti odlišné a pro matematickou sazbu je kód OT1 potřebný vždy. To brání zavedení nových fontů na staříčkých počítačích s malým diskem. Pro českou a slovenskou sazbu byl tedy vytvořen jiný font, který jednak šetří prostor na disku, a navíc má diakritická znaménka vytvořena podle národních typografických zvyklostí. Označuje se IL2 a je konzervativním rozšířením původního kódu OT1.

Představme si nyní, co se stane, když chceme vytisknout „ř“. Uvažujme nejsložitější případ, kdy chceme v jednom dokumentu použít několik různých fontů s odlišným kódováním a zpracováváme dokumenty psané s různým kódováním češtiny. Znak vyjadřující „ř“ v daném vstupním kódování musí být aktivní a definovaný tak, aby se expandoval na `\v{r}`. Tuto sekvenci již v zásadě můžeme použít k tisku „ř“, neboť příkaz `\v` je definován v každém textovém kódu. Například v OT1 bude konečným výsledkem expanze `\accent20r`. Primitiv `\accent` ovšem potlačí dělení slova. Pokud tedy příslušný font má znak „ř“, provede se odpovídající konverze. Dokument lze díky těmto vlastnostem zpracovat i na instalaci, která má pouze fonty s kódováním OT1. Výsledek sice bude vzhledem k potlačení dělení slov významně horší, nicméně pro hrubé čtení postačí. Pokud by tento složitý rys, a to jsme postup konverze popsali velmi stručně a nepřesně, nebyl implementován, nedal by se dokument na odlišné instalaci vytisknout vůbec (nebo až po mnoha zásazích do textu).

Problematika kódování je téměř vyčerpána, ale stále jsme nevysvětlili problém nastíněný v prvním odstavci této podkapitoly. Všimněte si rozdílu mezi < a $<$. První znak je v písmu simulujícím psací stroj, zatímco druhý znak je matematický. Znaky ¡ a ¿ pocházejí z obvyklého patkového písma Computer Modern, které ovšem znaky $<$ a $>$ neobsahuje. OT1 totiž není kódováním v pravém slova smyslu, neboť umístění znaků závisí i na volbě rodiny a řezu písma. Písmo psacího stroje má < na místě, kde základní proporcionální písmo obsahuje ¡. Při změně řezu na kurzívu se ze znaku \$ stane £. Tyto problémy dědí i kódování IL2.

O dalších atributech písma se zmíníme jen zkratkovitě. Nechceme zde suplovat učebnice typografie. Spíše uvedeme názvy charakteristik ve spojení s odpovídajícími LaTeXovými příkazy.

O rodině písma (`\fontfamily`) jsme se již zmínili. Dalšími charakteristikami jsou řez (`\fontshape`) a duktus (`\fontseries`).

8.2. Příkazy pro výběr písma

Řekli jsme si, že každé písmo je charakterizováno kódováním, rodinou, duktem, řezem a stupněm (velikostí). Všechny tyto charakteristiky budeme chtít ve svých dokumentech měnit nezávisle.

Výběr písma je implementován v plain TEXu i ve všech dalších formátech. Změna charakteristik však není zcela nezávislá.

Starý LATEX 2.09 poskytoval příkazy pro několik zvolených fontů. Přitom příkaz pro změnu velikosti písma současně nastavil základní rodinu, řez i duktus. Představte si, že jste měli větu:

Chceme ve větě napsat jedno slovo velké.

Nyní se rozhodneme převést celou větu do kurzívy. Výsledek nás překvapí. Uprostřed věty kurzíva nebude:

Chceme ve větě napsat jedno slovo *velké.*

Navíc nebyla zařazena ani kurzívní korekce. Při takové změně tedy bylo nutno projít celý text a na mnoha místech ručně dopisovat změny fontů. Pokud se nám přece jen kurzíva nelíbila a chtěli jsme obnovit původní stav, museli jsme opět změnit text na mnoha místech. Proto již pro LATEX 2.09 vytvořil Frank Mittelbach „New Font Selection Scheme“ (NFSS) umožňující ortogonální změny jednotlivých atributů. LATEX 2.09 ještě nerozeznával kódování fontů. To je implementováno až v tzv. NFSS2, které obsahuje LATEX 2_ε. Nyní tedy {`\it Chceme ve větě napsat {\Large jedno slovo} velké.`} vysází:

Chceme ve větě napsat *jedno slovo* *velké.*

Uvedený příklad se může zdát příliš vyumělkovaný. Je pravda, že typografická kvalita tohoto příkladu je dosti bídná. Představte si však, že máte delší text, kde jsou nadpisy vysázeny větším písmem, případně písmem tučným. Po provedení korektury jsme požádáni, abychom text přesadili například do švabachu. Dáme tedy za `\begin{document}` příkaz pro změnu fontu. Použijeme-li původní LATEX 2.09, budou větší a tučné nadpisy v Computer Modern Roman vypadat zvlášť ohavně. Musíme tedy ještě předefinovat formátování nadpisů – a v budoucnu to budeme muset opakovat pro každý jiný font. LATEX 2_ε nám tuto práci ušetří.

Za zmínku stojí i nová možnost. Díky ortogonálnímu výběru lze kombinací `\bfseries` a `\itshape` psát ***tučnou kurzívou***, což dřívější LaTeX nepřipouštěl bez zavedení nového fontu pomocí maker, jež nebyla ve zdrojovém kódu dokumentována. Aby se předešlo nejednoznačnostem při zpracování starých souborů, mají jednoduché příkazy `\bf`, `\it` a podobné týž význam jako v původním LaTeXu 2.09.

V minulé části jsme již uvedli, že máme makra deklarativní a příkazová. Příkazové makro, např. `\textit`, se hodí pro vysázení kratší části textu v kurzívě. Příkazová makra se automaticky starají o vkládání kurzívních korekcí. Deklarativní makro, např. `\itshape`, mění příslušný atribut uvnitř skupiny, v níž je použito. Hodí se zejména v uživatelských definicích dalších maker. Kurzívní korekce je nutno zadat explicitně.

Kurzívní korekce se vkládá při přechodu mezi skloněným a stojatým písmem. Nevypadalo by ovšem dobře, kdyby se korekce vložila před čárku, tečku, případně jiný podobný znak. Při automatickém vkládání se tedy kurzívní korekce vynechá v případě, že za ní následuje znak uvedený v `\nocorrlist`. Standardní definice tohoto makra je:

```
\newcommand{\nocorrlist}{,.}
```

Uživatel může toto makro předefinovat. Z hlediska rychlosti je vhodné, aby jednotlivé znaky byly uvedeny v pořadí četnosti výskytu v dokumentu (nejčastější na prvním místě). Kromě této globální změny lze potlačit vložení kurzívní korekce jednotlivě uvedením makra `\nocorr` na odpovídající straně textu v příkazovém makru pro změnu písma.

Atributy písma jsou voleny makry uvedenými v tabulce 1. Skutečný význam jednotlivých příkazů lze změnit na uživatelské úrovni. Každé makro totiž volá jistý „hook“, jehož definice může být změněna. Seznam těchto vnitřních maker včetně jejich hodnot je uveden v tabulce 2.

Makra z tabulky 1 tedy provádějí složitější činnost. Například `\bfseries` je expandováno na `\fontseries{\bfdefault}\selectfont`. Po změně makra `\bfdefault` zvolí tudíž `\bfseries` jiný duktus.

Některá písma mohou obsahovat rodiny, dukty či řezy, pro něž nejsou definována makra. Musíme tedy použít příkazy nižší úrovně. Pro změnu rodiny, duktu a řezu máme tedy postupně makra `\fontfamily`, `\fontseries` a `\fontshape`. Kódování se změní výhradně makrem `\fontencoding`. Tato makra pouze sdělí systému NFSS, že se nějaký atribut bude měnit, ale neprovedou vlastní výběr písma. Změna se provede až příkazem `\selectfont`. Zkuste si ve svém LaTeXu následující příkazy:

Tabulka 1: Makra pro změnu písma

Příkaz	*Deklarace*	*Význam*
`\textrm{...}`	`{\rmfamily...}`	Patkové písmo
`\textsf{...}`	`{\sffamily...}`	Bezpatkové písmo
`\texttt{...}`	`{\ttfamily...}`	`Písmo psacího stroje`
`\textmd{...}`	`{\mdseries...}`	Střední duktus
`\textbf{...}`	`{\bfseries...}`	**Tučné písmo**
`\textup{...}`	`{\upshape...}`	Stojaté písmo
`\textit{...}`	`{\itshape...}`	*Kurzíva*
`\textsl{...}`	`{\slshape...}`	*Skloněné písmo*
`\textsc{...}`	`{\scshape...}`	Kapitálky
`\emph{...}`	`{\em...}`	*Zvýrazněné* písmo
`\textnormal{...}`	`{\normalfont...}`	Základní písmo

Tabulka 2: Definice atributů písma (hooks)

Hook	*Hodnota*	*Použito při:*
`\encodingdefault`	`OT1`	Kódování základního písma
`\familydefault`	`\rmdefault`	Rodina základního písma
`\seriesdefault`	`m`	Duktus základního písma
`\shapedefault`	`n`	Řez základního písma
`\rmdefault`	`cmr`	`\rmfamily` nebo `\textrm`
`\sfdefault`	`cmss`	`\sffamily` nebo `\textsf`
`\ttdefault`	`cmtt`	`\ttfamily` nebo `\texttt`
`\bfdefault`	`bx`	`\bfseries` nebo `\textbf`
`\mddefault`	`m`	`\mdseries` nebo `\textmd`
`\itdefault`	`it`	`\itshape` nebo `\textit`
`\sldefault`	`sl`	`\slshape` nebo `\textsl`
`\scdefault`	`sc`	`\scshape` nebo `\textsc`
`\updefault`	`n`	`\upshape` nebo `\textup`

```
{\bfseries text}
{\fontseries{b}\selectfont text}
{\fontseries{sbc}\sffamily text}
```

Všimněte si, že poslední řádek výše uvedeného příkladu neobsahuje `\selectfont`. Makro `\sffamily` se totiž expanduje na `\fontfamily {\sfdefault}\selectfont`. Poslední řádek je tedy přibližně ekvivalentní zápisu:

```
{\fontseries{sbc}\fontfamily{sf}\selectfont text}
```

Ukázali jsme si, že je naprosto dostačující, když uvedeme `\selectfont` pouze jednou až po definování změn všech požadovaných atributů. Nyní předpokládejme, že chceme jedno slovo napsat v tučné kurzívě rodiny Times, přičemž příslušný font má kódování T1. Měli bychom tedy v dokumentu napsat

```
\fontencoding{T1}\fontfamily{ptm}\fontseries{bx}
\fontshape{it}\selectfont
```

To bychom se ale upsali. Naštěstí nám NFSS nabízí zkrácený zápis:

```
\usefont{T1}{ptm}{bx}{it}
```

Všimněte si, že zde již nepotřebujeme `\selectfont`. Význam parametrů je z ukázky zřejmý.

Nyní již tedy víme, proč funguje `\fontfamily{ptm}\selectfont`. Tento způsob ale není zcela správný. Čísla stránek a běžné záhlaví se tiskne základním písmem dokumentu, takže `\normalfont` přepne zpět na Computer Modern. pokud chceme vysázet celý dokument písmem Times Roman, měli bychom tedy na jeho začátek vložit:

```
\renewcommand\rmdefault{ptm}
```

Tak se chová standardní balík TIMES. V něm jsou navíc změněny rodiny písma bezpatkového a písma psacího stroje.

Nezmínili jsme se ještě o makru `\fontsize`. To slouží ke změně velikosti písma. Makro má dva parametry. Prvním parametrem se definuje stupeň písma, druhý parametr určuje rozteč účaří. V obou parametrech můžeme uvést jak prosté číslo, tak TEXovský rozměr. Pro tisk plakátů nebo folií určených k promítání s oblibou používám:

```
\fontsize{10mm}{13mm}\selectfont
```

Makra `\normalsize`, `\small` a podobná jsou však poněkud složitější. O tom si ale povíme později.

9. Zavedení nového písma

V předchozím textu jsme si vysvětlili, jak máme LaTeXu sdělit, které písmo chceme pro sazbu použít. Mohli jsme ale specifikovat pouze takové písmo, které již LaTeX zná. Nyní si povíme, jak naučíme LaTeX další písma.

9.1. Primitiv `\font`

LaTeX umožňuje použití mnoha TeXových primitivů. Jedním z nich je právě `\font`. Primitiv je velmi rychlý, protože nepodléhá žádným expanzím. Chceme-li tedy z fontu použít jeden nebo několik málo znaků, např. když máme v nějakém fontu vytvořený obrázek, je primitiv `\font` vhodným kandidátem. Pro zavádění písma pro text se však nehodí.

Představme si, že písmo pro azbuku zavedeme pomocí primitivu `\font` takto:

```
\font\azb=wncyr10
```

Pak pomocí `{\azb pisheme bukvy}` skutečně пишеме буквы, protože v daném fontu „sh“ tvoří ligaturu „ш“. Takový font je ale z hlediska LaTeXu statický, takže *v kurzívě* буквы писать нельзя. Stejně tak, маленькие буквы nedokážeme vytisknout.

Ekvivalentním způsobem je i zavedení písma LaTeXovým makrem `\newfont`. Syntaxe je jiná, ale výsledek je stejný.

Podobnou zradu způsobí zavedení fontu s jiným kódováním. Zde jsme přepnuli kódování na T1, takže sázíme fontem `dcr10`. Protože jsme změnili kódování a nemáme česká písmena definována jako aktivní, musíme vše zapisovat TeXovými sekvencemi. Text ale dopadne dobře, neboť se o to LaTeX postará. V následujícím odstavci zavedeme `dcr10` primitivem `\font`. Podívejte se, co z toho vyjde!

Podobnou zradu zpùsobí zavedení fontu s jiným kódováním. Zde jsme pøepnuli kódování na T1, tak¿e sázíme fontem `dcr10`. Proto¿e jsme zmìnili kódování a nemáme èeská písmena definována jako aktivní, musíme vže zapisovat TeXovými sekvencemi. Text ale dopadne dobøe, nebož se o to LaTeX postará. V následujícím odstavci zavedeme `dcr10` primitivem `\font`. Podívejte se, co z toho vyjde!

Kódy některých znaků se shodují, proto je text alespoň částečně srozumitelný. Nicméně, zpravodaj vysázený takovým způsobem by se vám jistě nelíbil. Potřebujeme tedy jiný mechanismus pro zavádění fontů.

9.2. Makro \DeclareFixedFont

Představme si, že nyní zavedeme font dcr10 pomocí:

```
\DeclareFixedFont{\DC}{T1}{\familydefault}{\seriesdefault}
{\shapedefault}{10}
```

Potom jako přepínač písma použijeme \DC a znovu vytiskneme předposlední odstavec předchozí podkapitoly.

Podobnou zradu způsobí zavedení fontu s jiným kódováním. Zde jsme pøepnuli kódování na T1, tak¿e sázíme fontem dcr10. Proto¿e jsme zmìnili kódování a nemáme èeská písmena definována jako aktivní, musíme vše zapisovat TEXovými sekvencemi. Text ale dopadne dobøe, neboť se o to LATEX postará. V následujícím odstavci zavedeme dcr10 primitivem \font. Podívejte se, co z toho vyjde!

Ani tentokrát se to nepovedlo. LATEX sice má informaci o kódování, ale zapomněl jej změnit. Musíme tedy za \DC zapsat ještě \selectfont, čímž se vše opraví. Důkazem je následující odstavec.

Podobnou zradu způsobí zavedení fontu s jiným kódováním. Zde jsme přepnuli kódování na T1, takže sázíme fontem dcr10. Protože jsme změnili kódování a nemáme česká písmena definována jako aktivní, musíme vše zapisovat TEXovými sekvencemi. Text ale dopadne dobře, neboť se o to LATEX postará. V následujícím odstavci zavedeme dcr10 primitivem \font. Podívejte se, co z toho vyjde!

Kódování se nám sice změnilo, ale stejně font zůstává statický. Pro plnohodnotnou definici textového fontu potřebujeme jiný přístup.

9.3. Jak LATEX hledá fonty

Následující text nebude zcela přesný. Pro zajištění efektivity jsou totiž některé fonty interní. Jsou tedy zavedeny již během generování formátu. Málo používané fonty jsou externí. Znamená to, že až při zpracování dokumentu hledá LATEX jejich definici. Pro jednoduchost popíšeme pouze způsob, jak LATEX zavádí externí fonty. S interními fonty se provádí totéž, ale již při vytváření formátu.

LATEX musí rozumět kódování fontu. $\mathcal{CS}$-fonty mají kódování IL2, ale to standardní LATEX nezná. Má-li s takovým fontem pracovat, musí se toto kódování nejprve naučit. LATEX tedy načte soubor IL2enc.def, kde musí být kódování definováno. Obsah tohoto souboru zde popisovat nebudeme. Zájemce odkážeme na standardní dokumentaci, která je povinnou součástí každé distribuce LATEXu.

Předpokládejme nyní, že chceme použít Times-Roman v kódování KOI-8-ČS. Rodinu tohoto písma označujeme zkratkou `ptm`. Někde uvnitř se tedy vyskytnou příkazy `\fontencoding{KOI}` a `\fontfamily{ptm}`. LaTeX se proto porozhlédne po definici kódování KOI-8-ČS. Pokud toto kódování nezná, přečte jej ze souboru `KOIenc.def`. Známe-li definici kódování, můžeme již načítat deklaraci rodiny písma. Ta se v uvedeném případě bude vyskytovat v souboru `KOIptm.fd`. Z tohoto příkladu je snad zřejmé, jak se tvoří názvy fd-souborů pro daná kódování a rodiny fontů. Uživatelé UNIXu si musí uvědomit, že malá/velká písmena v názvech těchto souborů jsou významná. V jiných operačních systémech lze klidně psát i `koiptm.fd`, čímž se ovšem ztrácí přehlednost.

Soubor s deklarací fontu začíná definicí rodiny písma příkazem `\DeclareFontFamily`. Toto makro má tři parametry. Prvním parametrem je kódování, druhým parametrem je název rodiny a na závěr se uvádí příkaz, který se má provést automaticky při zvolení rodiny. Třetí parametr bývá obvykle prázdný. Times-Roman v kódování KOI-8-ČS bychom tedy uvedli definicí:

```
\DeclareFontFamily{KOI}{ptm}{}
```

Poslední parametr lze využít pro změny, které se budou týkat celé rodiny. Například v písmu psacího stroje chceme potlačit dělení. Provedeme to způsobem, který definuje TeX. Při rozdělení slova se za první část vysadí spojovník. TeX ale potřebuje vědět, jak vlastně takový spojovník vypadá. Pro každý font je proto definován parametr `\hyphenchar`, v němž je uložen kód spojovníku. Pokud nastavíme tento kód na -1, znamená to pro TeX, že spojovník neexistuje a dělení se tudíž potlačí. Tuto změnu můžeme provádět automaticky tím, že ji zapíšeme do třetího parametru deklarace rodiny. Při změně rodiny pak LaTeX tuto operaci provede automaticky za nás. Pro písmo psacího stroje tak budeme mít následující definici (v originálním kódování OT1):

```
\DeclareFontFamily{OT1}{cmtt}{\hyphenchar\font=-1}
```

Poslední parametr deklarace rodiny písma lze použít i k jiným trikům. Máme např. písmo, jehož kódování je téměř shodné s IL2, ale některé (nepísmenné) znaky se liší. Mohli bychom definovat nové kódování, ale pak bychom museli znovu vytvořit formát a specifikovat vzory dělení pro další kódování, čímž bychom zbytečně plýtvali pamětí. Protože kódy všech písmen se shodují, můžeme dělit slova podle pravidel určených kódováním IL2. Při zvolení rodiny pouze sdělíme LaTeXu, aby předefinoval

několik symbolů. Pro písmo rodiny `bask` pak taková deklarace vypadá:

```
\DeclareFontFamily{IL2}{bask}{\chardef\textellipsis=8 %
  \def\textsection{^^a7}%
  \def\textdagger{^^83}\def\textdaggerdbl{^^84}}
```

V předchozím textu jsme si popisovali makra pro změnu jednotlivých atributů písma. Uvedli jsme jako příklad `\fontseries{sbc}`, ale nevysvětlili jsme, odkud se to „sbc“ vzalo. Obvykle najdeme v dokumentaci písma, v jakých řezech a duktech je k dispozici. Pokud takovou informaci nemáme, musíme se podívat do fd-souboru a vyčíst ji z příkazů `\DeclareFontShape`. Toto makro má šest parametrů. První čtyři jsou kódování, rodina, duktus a řez. V pátém parametru definujeme velikosti a externí jména fontů. Šestý parametr je obvykle prázdný a specifikují se v něm příkazy, které se mají automaticky provést při zvolení tohoto fontu. Je to obdoba posledního parametru makra `\DeclareFontFamily`.

Nyní se více zastavíme u pátého parametru. Ten totiž poskytuje mnoho různých možností. Předpokládejme, že máme nový font, a chceme LaTeXu sdělit, jak jej má používat. První čtyři parametry zapíšeme snadno, a právě ten pátý nám dá nejvíce práce. Musíme ke každé velikosti definovat jméno fontu, který má TeX zavést. Začneme fiktivním fontem z rodiny `muj` v kódování OT1, který máme k dispozici pouze ve velikosti 10 pt. Pro střední duktus a základní řez pak můžeme psát:

```
\DeclareFontShape{OT1}{muj}{m}{n}{ <10> mujmr10 }{}
```

Máme-li font i ve velikosti 12 pt, použijeme definici:

```
\DeclareFontShape{OT1}{muj}{m}{n}{ <10> <12> mujmr10 }{}
```

Pak při zvolení velikosti 12 pt zavede TeX `mujmr10` ve zvětšení 1,2, což obvykle vyjadřujeme jako `\scaled 1200`.

PostScriptové obrysové fonty jsou libovolně zvětšovatelné a RIP je vygeneruje v libovolné velikosti. Pro všechny velikosti pak máme týž externí soubor a v deklaraci nahradíme konkrétní velikost nekonečným rozpětím. V popisu počeštěného písma Times máme např. pro tučnou kurzívu deklaraci:

```
\DeclareFontShape{IL2}{ptm}{bx}{it}{<-> ptmbi8z}{}
```

Vezměme si nyní příkaz:

```
\DeclareFontShape{IL2}{fnt}{m}{sl}{<-12> fntsmall
                                   <12-> fntlarge}{}
```

Zde jsme definovali skloněné (slanted) písmo fiktivní rodiny `fnt` středního duktu. Pro stupeň menší než 12 pt se zavede externí font `fntsmall`, pro 12 pt a větší se použije `fntlarge`.

V tomto rozsáhlejším úvodu jsme si popsali základní tvar makra `\DeclareFontShape`. Nyní si povíme více o možnostech specifikace velikosti a externího jména. Velikost (stupeň) písma lze zadat buď jednoduchými čísly jako `<10>` a `<14.4>` nebo rozmezím dvou čísel, např. `<5-10>`. Tato specifikace pak platí pro velikost větší nebo rovnou 5 pt a menší než 10 pt. Jednu mez můžeme vynechat. Potom `<-12>` specifikuje všechny velikosti menší než 12 pt, `<14.4->` definuje externí font pro velikost větší nebo rovnou 14.4 pt. Vynecháme-li obě meze, definujeme pak jedním externím fontem všechny velikosti. Jak jsme již ukázali, používá se takový zápis zejména ve spojení s PostScriptovými obrysovými fonty. Font je obvykle dostupný v několika velikostech. Zápis pak můžeme zkrátit vypuštěním opakující se informace. Například písmo Pandora existuje pouze jako jeden METAFONTový soubor, z něhož získáváme další velikosti zvětšováním či zmenšováním. Deklarace pro obvyklé stupně známé z METAFONTu pak vypadá:

```
\DeclareFontShape{OT1}{panr}{m}{n}{
  <5> <6> <7> <8> <9> <10> <10.95> <12>
  <14.4> <17.28> <20.74> <24.88> pan10 }{}
```

Nyní požádáme LaTeX, aby zavedl font pandora ve velikosti 12 pt. Z výše uvedené deklarace NFSS pochopí, že potřebujeme `pan10` ve zvětšení 1,2. Jak ale LaTeX přišel na to, že základní externí font má stupeň 10 pt? Tato informace je uvedena v metrickém souboru `pan10.tfm`. Pokud jej programem TFTOPL převedeme do textové formy, najdeme v `pan10.pl` informaci:

```
(DESIGNSIZE R 10.0)
```

V METAFONTu zadáváme velikost písma pomocí parametrů `font_size` a `designsize`, ale jejich popis patří do jiného článku. Tyto parametry totiž musí znát tvůrce fontu. Chceme-li hotové písmo použít v LaTeXu, nemusíme o jejich existenci vůbec vědět. Je zcela postačující, když si uvědomíme, že velikost písma je zapsána v metrickém souboru a LaTeXové (nebo TeXové – z hlediska uživatele to není podstatné) algoritmy si příslušnou informaci najdou.

Případ, který jsme právě diskutovali, je vlastně nejjednodušší „size function“, tzv. prázdná funkce. Má nejkratší možné jméno, protože se

používá nejčastěji. Existuje řada různých funkcí. Pokud se v zápisu vyskytuje hvězdička, pak vše nalevo od hvězdičky je jméno funkce, napravo jsou argumenty. Některé funkce mají též nepovinný argument, který se pak uvádí v hranatých závorkách. Pokud specifikace velikosti neobsahuje hvězdičku, pak se jedná o prázdnou funkci a celý text je argument.

Základní podobu prázdné funkce jsme si již popsali. Prázdná funkce však může mít jeden nepovinný parametr. Představme si, že máme písmo `mujps`, které chceme kombinovat s fontem Times. Naše písmo má však při stejném nominálním stupni vyšší kresbu a pro estetické sladění bychom jej chtěli redukovat na 93 %. toho dosáhneme automaticky následující definicí velikosti:

```
<-> [.93] mujps
```

Funkce „s“ je funkčně identická s prázdnou funkcí. Jediný rozdíl spočívá v tom, že se žádné diagnostické zprávy nepíší na obrazovku. Do log-souboru se však zapisují vždy. Název funkce je odvozen z prvního písmene anglického slova silence – ticho. Použití funkce „s“ může vypadat například takto:

```
\DeclareFontShape{OT1}{muj}{b}{sl}{ <-> s * [.93] mujps }{}
```

Písma generovaná METAFONTem jsou obvykle dostupná v řadě diskrétních velikostí. Abychom si ušetřili zápis, použijeme s výhodou funkci „gen“ nebo její tichou variantu „sgen“. Zápis:

```
<8> <9> <10> gen * cmtt
```

je pak zkratkou za:

```
<8> cmtt8 <9> cmtt9 <10> cmtt10
```

I tato funkce rozeznává nepovinný argument, který má opět význam zvětšení stejně jako u prázdné funkce.

Některé rodiny písem jsou bohaté na různé řezy a dukty, v jiných rodinách jsou možnosti silně omezeny. Když si v dokumentu zvolíme neexistující kombinaci, NFSS vybere nějaký náhradní font. To ale nemusí být volba, která se nám líbí. Představte si, že sázíte švabachem a požadujete skloněné písmo. Skloněný (slanted) švabach ale nemáte, a NFSS tedy použije skloněný Computer Modern. To je naprosto nežádoucí. Zcela logická je v takovém případě náhrada švabachovou kurzívou, kterou třeba máte k dispozici. Takové substituce lze definovat funkcí „sub“ nebo její tichou variantou „ssub“. Argument této funkce se uvádí ve formě rodina/duktus/řez, kde všechny části označujeme příslušnými zkratkami.

Například v definičním souboru rodiny Times s kódováním IL2 je skloněné písmo nahrazeno kurzívou následujícím příkazem:

```
\DeclareFontShape{IL2}{ptm}{m}{sl}{<-> sub * m/it}{}
```

Písmo Zapf-Chancery-Medium-Italic máme v tiskárně pouze v jediné kombinaci. Téměř celý soubor pak obsahuje substituce:

```
\DeclareFontFamily{IL2}{pzc}{}
\DeclareFontShape{IL2}{pzc}{m}{it}{<-> pzcmi8z}{}
\DeclareFontShape{IL2}{pzc}{m}{sl}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{m}{n}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{m}{sc}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{n}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{sl}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{it}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{sc}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{n}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{sl}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{it}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{sc}{<-> sub * m/it}{}
```

Ve výše uvedených příkladech jsme nahradili celý řez písma řezem jiným. Máme ale další možnost. Předpokládejme, že nějaké písmo je dostupné pouze v některých velikostech. Pokud uživatel zvolí stupeň, který dané písmo nemá, použijeme rozumnou náhradu. Například můžeme mít definici:

```
\DeclareFontShape{OT1}{cmss}{m}{sl}{ <-8> sub * cmss/m/n
  <8> <9> gen * cmssi <10> <10.95> cmssi10
  <12> <14.4> cmssi12
  <17.28> <20.74> <24.88> cmssi17 }{}
```

Fonty velikosti menší než 8 pt budou tedy nahrazeny základním řezem cmss/m/n. Takový řez ovšem neexistuje a definice obsahuje další substituce. Vidíme tedy, že substituce se mohou řetězit, ale pochopitelně nesmějí obsahovat cyklus. Výhoda takového zápisu spočívá v tom, že měníme pouze jednu skupinu, když v nové distribuci získáme nový font.

Další funkce zde již nebudeme popisovat, protože se bez nich obvykle obejdeme. Zájemce odkážeme v závěru článku na příslušnou literaturu.

9.4. NFSS pro starý LaTeX 2.09

Na tomto místě nechceme dělat reklamu starému LaTeXu, ale zastavíme se u něj z jiného důvodu. Přadstavme si, že nám někdo věnuje (nebo prodá) nové písmo a soubor maker pro zavedení tohoto písma právě do NFSS v LaTeXu 2.09. My jsme ovšem pokrokoví a používáme LaTeX 2_ε. Pokud pro nás LaTeX 2.09 není jen obstarožní záhadou, máme většinu práce již hotovu. LaTeX 2.09 neměl fd-soubory, ale nové fonty byly zaváděny pomocí stylů. Makra pro zavedení nového stylu tedy najdete v souboru s příponou `.sty`. Podíváte-li se do takového souboru, bude se vám zdát povědomý. Rodina je tam definována makrem `\@newfontfamily` a jednotlivé řezy jsou specifikovány v `\@newfontshape`. Jejich syntaxe je stejná jako v NFSS2, kterou používá LaTeX 2_ε. Chybí pouze kódování, protože LaTeX 2.09 kódování nerozeznával. Stačí tedy jednoduchá záměna maker a doplnění kódování fontu. Taková operace je v libovolném editoru otázkou několika minut. Výsledný soubor musí mít samozřejmě jméno podle vzoru `KŌDrodina.fd`. Uživatelé OS/2 a UNIXu mohou mít tendenci k extrémně dlouhým jménům souborů. Například si pojmenují rodinu Zapf-Chancery-Medium-Italic plným jménem (bez spojovníků) a pro kód KOI-8-ČS pak dostanou soubor `KOIZapfChanceryMediumItalic.fd`. Vězte však, že takový soubor nebude přenositelný do operačních systémů, kde jsou jména souborů omezena na $8+3$. Pokud jej vůbec nějak dostanete do DOSu, bude se výsledný soubor jmenovat `KOIZAPFC.FD`. V některých implementacích pak taková změna může činit potíže. V nových verzích emTeXu je tento problém (naštěstí) ošetřen.

10. Praktický postup zavádění nového písma

Základní teorii jsme tedy zvládli a můžeme se pustit do práce. Jak ale máme postupovat? Nejlepší je, když nám příslušný soubor napíše nějaký automat. Standardem je balík FONTINST, pro počeštěné fonty lze použít CSPSFONT, jenž je součástí $\mathcal{CS}$TeXu. Může se ovšem stát, že žádný z těchto balíků nemáme, nebo se nám zdá složitý a nechceme se jej učit, případně nemáme všechny informace a potřebujeme narychlo vytvořit alespoň přibližný popis, který půjde v našem dokumentu použít. Čeká nás tedy trocha práce a následující odstavce popíší doporučený postup.

Nejprve musíme identifikovat kódování fontu. Často lze kódování odhadnout z povahy fontu, nebo jej dokonce víme od autora písma. Pokud takovou informaci nemáme, musíme si vytisknout tabulku a srovnat ji s definicemi známých kódů. Dingbaty a jiné exotické fonty obvykle nevyhovují žádnému kódování. Zde musíme použít „neznámé“ (undefined) kódování, které se značí symbolem „U“.

V dalším kroku zvolíme jméno rodiny. Jsou to obvykle první společná písmena jmen souborů. Pokud budeme takové písmo používat výhradně na svém domácím počítači, stačí, když zvolíme pro novou rodinu název, který ještě není použit. Přesto je vhodné držet se zavedených konvencí – viz doporučenou literaturu.

Každé písmo má své metrické soubory s příponou `.tfm`, které musí být součástí distribuce daného fontu. V případě písma vytvořeného METAFONTem se může stát, že dostaneme pouze zdrojové soubory, a metriky si musíme vygenerovat METAFONTem sami. Takový případ je však již řídký. Některá písma mohou obsahovat navíc virtuální skripty. Poznáme to podle toho, že k některým souborům s příponou `.tfm` existuje soubor s příponou `.vf`. Při zkoumání se pak zaměříme právě na tyto soubory. Na metriky, ke kterým nemáme stejnojmenný virtuální skript, zapomeneme. To v žádném případě neznamená, že je můžeme zahodit. Tyto soubory se musí vyskytovat v adresáři, kde je najde TEX a ovladače výstupních zařízení. My se však o ně nebudeme zajímat.

Dešifrováním jmen souborů vyzkoumáme, v jakých řezech a duktech je písmo dostupné. Obvykle ve jménech již najdeme odpovídající písmena `b`, `bx`, `sc` (nebo jen `c`), `ti` (místo očekávaného `it` pro *text italics* – na rozdíl od `mi` pro *math italics*) a konečně `sl` nebo `o` pro skloněné (anglicky *slanted* nebo *oblique*) písmo. Někdy jsou však názvy silně kryptické a nemáme jinou možnost než vytisknout si vzorek písma a z jeho vzhledu usoudit, o jaký řez a duktus se vlastně jedná.

Nakonec potřebujeme zjistit, v jakých velikostech je písmo dostupné. Vezměme si například písmo OCR-B. Podíváme se na metrické soubory a najdeme `ocrb5.tfm`, `ocrb6.tfm`, `ocrb7.tfm`, `ocrb8.tfm`, `ocrb9.tfm` a `ocrb10.tfm`. Číslo obvykle určuje stupeň písma. Protože OCR asi nepoužijeme pro normální text, zvolíme kódování U a rodinu pojmenujeme ocrb. Pak lze psát:

```
\DeclareFontFamily{U}{ocrb}{}
\DeclareFontShape{U}{ocrb}{m}{n}{
  <5> <6> <7> <8> <9> <10> gen * ocrb }{}
```

Pokud by distribuce OCR-B obsahovala pouze metriky a hotové pk-soubory pro výstupní zařízení, byli bychom tím hotovi. My však máme METAFONTový zdroj, takže lze generovat i větší písmo. Pro obvyklé velikosti pak napíšeme:

```
\DeclareFontFamily{U}{ocrb}{}
\DeclareFontShape{U}{ocrb}{m}{n}{
  <5> <6> <7> <8> <9> <10> gen * ocrb
  <10.95> <12> <14.4> <17.28> <20.74> <24.88> ocrb10 }{}
```

Měli bychom ještě nadefinovat substituce pro jiné kombinace duktu a řezu, ale to již necháme čtenářům za domácí úkol. Vše pak zapíšeme do souboru `Uocrb.fd`, který umístíme do adresáře, který prohledává LaTeX. Můžeme však uvedená makra zapsat i do preambule dokumentu. K tomu přistoupíme zejména v případě, že jsme napsali narychlo neúplnou deklaraci. Máme-li `\DeclareFontFamily{U}{ocrb}` v dokumentu, nenačte již LaTeX `Uocrb.fd`, i kdyby tento soubor existoval.

Výše uvedené příkazy jsme zkopírovali těsně nad tento odstavec. Nyní lze zapsat:

```
{\fontsize{14}{18}\usefont{U}{ocrb}{m}{n}%
                                    VZOR OCR 1234567890}
```

Výsledkem je pak:

VZOR OCR 1234567890

Všimněte si, že jsme požadovali stupeň 14 pt, který není k dispozici. LaTeX pak vybere nejbližší existující velikost, v našem případě 14,4 pt. Na to je nutno dávat pozor, neboť někdy to může vést k velmi nepříjemným následkům.

U PostScriptových fontů je situace poněkud jednodušší. RIP jej totiž dokáže vyrastrovat v libovolné velikosti, takže obvykle specifikujeme neomezené rozmezí velikostí `<->`. Článek Pavla Herouta (najdete jej ve Zpravodaji č. 1/96 na straně 43) obsahoval ukázku ve fontu Souvenir. Autor vytvořil svůj článek v LaTeXu 2.09 a poskytl metriky i virtuální skripty příslušného fontu pro kódování XL2. To je vlastně rozšířením IL2, přičemž rozšiřující znaky nebyly nutné. Redakce používá LaTeX 2_ε, a aby mohla být vytištěna ukázka na straně 63, byl během zlomku minuty ručně vytvořen soubor `IL2pso.fd` obsahující:

```
\DeclareFontFamily{IL2}{pso}{}
\DeclareFontShape{IL2}{pso}{m}{n}{<->psol8z}{}
```

```
\DeclareFontShape{IL2}{pso}{m}{it}{<->psoli8z}{}
\DeclareFontShape{IL2}{pso}{m}{sl}{<->psoli8z}{}
\DeclareFontShape{IL2}{pso}{bx}{n}{<->psod8z}{}
\DeclareFontShape{IL2}{pso}{bx}{it}{<->psodi8z}{}
\DeclareFontShape{IL2}{pso}{bx}{sl}{<->psodi8z}{}
```

Pak již fungovaly obvyklé příkazy `\bfseries`, `\itshape` a jiné. Nemáme zde ovšem KAPITÁLKY, protože jejich virtuální skripty nebyly k dispozici. V příkladech použití tohoto fontu jsme je nepotřebovali, proto nebyly v definičním souboru uvedeny jejich substituce. Takový definiční soubor má pak omezenou použitelnost, ale představuje snadné řešení pro případ, že potřebujeme nový font zavést rychle alespoň v některých řezech.

Nyní umíme vytvořit velký nápis fontem Times-Roman stupně 30 pt. Odpovídající font pro obrazovkový prohlížeč ovšem nemáme. Pokud jsme si ale třeba pro Mattesův prohlížeč napsali správně substituční soubor a máme dobře nainstalovaný MFJOB, vhodný náhradní font se automaticky vygeneruje. Na PostScriptové tiskárně pak budeme mít skutečně Times-Roman. Když jej ale zkombinujeme s písmem Computer Modern stejné velikosti, budou nám písmenka poskakovat. Maximální dostupná velikost cm-fontů je totiž 24,88 pt! Lze to napravit zásahem do fd--souboru, ale v matematice to chodit nebude – tam je zavedení nové velikosti podstatně složitější. Mohli bychom i pro matematiku použít Times-Roman, ale ani to není jednoduché. Matematický font musí totiž obsahovat řadu parametrů `\fontdimen`, které program AFM2TFM nevytvoří. Můžeme je sice vytvořit ručně, ale je to pracné a zjištění správných hodnot vyžaduje jistý grafický cit. Potřebujeme-li velké nápisy obsahující matematiku, musíme vše vytisknout v nějaké velikosti podporované LaTeXem a zvštěit celý DVI-soubor parametrem tiskového ovladače nebo PostScriptovou transformací.

11. Změna základní velikosti písma

Všechny standardní třídy dokumentů nastavují stupeň písma na 10 pt. Použitím parametru `11pt` nebo `12pt` lze písmo zvětšit. Co ale můžeme udělat v případě, že požadujeme jinou velikost než 11 pt či 12 pt?

Velikost písma již umíme změnit. Mohli bychom si tedy předefinovat `\normalsize`. Rozhodneme se pro základní stupeň 8 pt s dvoubodovým řádkovým prokladem. Naše definice pak bude:

```
\renewcommand\normalsize{\fontsize{8}{10}\selectfont}
```

Zpočátku bude vše fungovat. Ve složitějším dokumentu však objevíme příliš veliké mezery, například okolo matematických vzorců nebo v enumeracích. LaTeX se totiž snaží „myslet“ za uživatele. Řada rozměrů by měla být v souladu s velikostí písma. Makra pro změnu velikosti fontu na to pamatují. V souboru `size10.clo` je uvedeno:

```
\renewcommand\normalsize{%
   \@setfontsize\normalsize\@xpt\@xiipt
   \abovedisplayskip 10\p@ \@plus2\p@ \@minus5\p@
   \abovedisplayshortskip \z@ \@plus3\p@
   \belowdisplayshortskip 6\p@ \@plus3\p@ \@minus3\p@
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}
\normalsize
\newcommand\small{%
   \@setfontsize\small\@ixpt{11}%
   \abovedisplayskip 8.5\p@ \@plus3\p@ \@minus4\p@
   \abovedisplayshortskip \z@ \@plus2\p@
   \belowdisplayshortskip 4\p@ \@plus2\p@ \@minus2\p@
   \def\@listi{\leftmargin\leftmargini
               \topsep 4\p@ \@plus2\p@ \@minus2\p@
               \parsep 2\p@ \@plus\p@ \@minus\p@
               \itemsep \parsep}%
   \belowdisplayskip \abovedisplayskip
}
```

TeXové primitivy `\abovedisplayskip`, `\abovedisplayshortskip`, jakož i `\belowdisplayskip` a `\belowdisplayshortskip` definují vertikální mezery okolo matematických rovnic. Makra `\topsep` apod. určují rozměry použité ve výčtech. Těm se budeme věnovat v jiném dílu kuchařky, takže je zde ponecháme bez komentáře.

Všimněte si hojného „rojení zavináčů“. Pro větší efektivitu je řada opakovaně se vyskytujících výrazů uložena do maker a registrů. Šetří se tím paměť počítače i čas. Například `\@minus` představuje jediný token, zatímco pro `minus` potřebujeme pět tokenů. Snadno si domyslíme, že `\@xpt = 10pt`, `\@xiipt = 12pt` a `\p@ = 1pt`.

Definici `\normalsize` ale na první pohled nevidíme. Makra pro

změnu stupně písma totiž definujeme prostřednictvím `\@setfontsize`. Tento příkaz má tři parametry: jméno makra, které chceme definovat, stupeň písma a vzdálenost účaří. Přibližně tedy vypadá funkce takto:

```
\def\@setfontsize#1#2#3{%
    \def#1{\fontsize{#2}{#3}\selectfont}}
```

Skutečnost je složitější, ale to nás v tomto okamžiku nebude zajímat. Všimneme si ale toho, že rozměry můžeme zadávat v libovolných jednotkách. Představme si, že chceme vytisknout plakát. Hodilo by se nám základní písmo velikosti 10 mm s dvoumilimetrovým řádkovým prokladem. Vytvoříme si pro tento účel soubor `size10mm.sty`, kde rozměry převedeme z pointů na milimetry. Po triviální úpravě budeme hotovi a soubor bude začínat příkazy:

```
\ProvidesPackage{size10mm}
\renewcommand\normalsize{%
   \@setfontsize\normalsize{10mm}{12mm}
   \abovedisplayskip 10mm \@plus2mm \@minus5mm
   \abovedisplayshortskip \z@ \@plus3mm
   \belowdisplayshortskip 6mm \@plus3mm \@minus3mm
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}
\normalsize
\renewcommand\small{%
   \@setfontsize\small{9mm}{11mm}%
   \abovedisplayskip 8.5mm \@plus3mm \@minus4mm
   \abovedisplayshortskip \z@ \@plus2mm
   \belowdisplayshortskip 4mm \@plus2mm \@minus2mm
   \def\@listi{\leftmargin\leftmargini
               \topsep 4mm \@plus2mm \@minus2mm
               \parsep 2mm \@plus1mm \@minus1mm
               \itemsep \parsep}%
   \belowdisplayskip \abovedisplayskip
}
```

Celý soubor má 217 řádků, proto jej zde neuvádíme. Jistě si zbytek domyslíte sami.

Zdálo by se, že jsme v úvodu této kapitolky zakázali použití makra `\fontsize`. Není to ale pravda. V neobvyklých případech potřebujeme

neobvyklé velikosti písma. Týká se to třeba titulních stránek nebo rozměrných tabulek. Zde bychom si s jednoduchými makry `\normalsize`, `\large`, `\small` atd. nemuseli vystačit. Také můžeme potřebovat jiný řádkový proklad. Tehdy použijeme `\fontsize`, ale nikdy nebudeme takto měnit velikost písma v běžném dokumentu.

Řádkový proklad je častým kamenem úrazu. Budeme se mu věnovat v dalším dílu kuchařky. Tím se vypořádáme s fonty a začneme s popisem dalších triků.

V minulé části jsme slíbili, že ukážeme, jak lze změnit vzhled obsahu. Bohužel se to do tohoto čísla již nevešlo. Zbývá tím dluh, který napravíme v prvním čísle příštího roku.

12. Doporučená literatura

Pro další studium lze doporučit již zmíněný článek Pavla Herouta, kde najdeme skutečné základy. Doporučujeme však, abyste při definicích nepoužívali přímo adresáře. Raději využijte konfigurační soubor, kterým definujete pro TEX i pro ovladače, kde se soubory daného typu vyskytují. Pak bude vaše práce přenositelná i do jiných instalací. Uvítají to vaši kolegové a za odměnu třeba i oni budou vytvářet své produkty s ohledem na přenositelnost a ušetří tím i vaši práci.

Při pokusu o zavedení nového písma se můžete setkat se spoustou problémů ještě dříve, neš se vám podaří vytisknout první písmeno. O těchto problémech a možnostech jejich diagnostiky a odstranění si můžete přečíst v tomto čísle Zpravodaje ve článku Arnošta Štědrého. Článek začíná na straně 249.

Již tradičně doporučíme knihu: Michel Goossens, Frank Mittelbach, Alexander Samarin – *The LaTeX Companion*. Addison Wesley, Reading 1994, ISBN 0-201-54199-8, zejména kapitolu 7.

Užitečné informace jsou i v souboru `usrguide.tex`, který je standardní součástí distribuce LaTeXu.

Zdeněk Wagner
`wagner@mbox.cesnet.cz`