

TEX pro pragmatiky

TEX – plainTEX – \mathcal{C}_S plain – OPmac

Petr Olšák

Předmluva

T_EX (vyslovujeme tech) je volně dostupný systém na vytváření elektronické sazby vysoké kvality [6, 22, 23]. Je vybaven sofistikovaným makrojazykem. T_EX byl vytvořen Donaldem Knuthem v 70. letech minulého století. Přesto se používá i v dnešní době a v mnoha vlastnostech dosud nemá konkurenci.

ČSplain je jednoduché rozšíření T_EXu obsahující makra a vzory dělení slov, které umožní psát v T_EXu mimo jiné česky nebo slovensky. Toto rozšíření je rovněž volně dostupné [10] a je odvozeno z minimálního makrobalíku k T_EXu zvaného plainT_EX. Existují daleko rozsáhlejší makrobalíky k T_EXu, například L^AT_EX nebo ConT_EXt. Jimi se tato kniha nezabývá.

OPmac [11] je jednoduché volně dostupné rozšíření ČSplainu vytvořené rovněž pomocí maker T_EXu. Nabízí podobné vlastnosti jako L^AT_EX, ale je snadnější pro uživatele a je jednodušší na úrovni implementace. Je součástí instalačního balíčku ČSplainu.

Tato příručka chce vyplnit mezeru mezi krátkým dokumentem *První setkání s T_EXem* (pro nováčky) [12] a *T_EXbookem naruby* (pro pokročilé uživatele) [13]. Do uvedené mezery byl zatím vklíněn český překlad „Jemný úvod do T_EXu“ [3], ovšem ten považuji z části za zastaralý a z části za zbytečně jemný a místy rozvláčný.

Už v názvu knihy přiznávám, že jsem se inspiroval názvem *L^AT_EX pro pragmatiky* [19], což je volně dostupná velmi zdařilá příručka pro L^AT_EX, která mi posloužila jako vzor k dosažení podobného cíle: umožnit pragmatikovi v co nejkratší době vládnout T_EXem, ČSplainem a souborem maker OPmac jako poučený uživatel. Už po přečtení kapitol 1 až 7 bude čtenář schopen vytvářet i poměrně sofistikované dokumenty s automaticky generovaným obsahem, rejstříkem, hyperlinkovými odkazy, obrázky atd. Další kapitoly využijí zejména ti čtenáři, kteří chtějí sami programovat jednoduchá makra. Kniha by měla umožnit porozumění souvislostí tak, aby bylo možné případně navázat studiem *T_EXbooku naruby* a začít účelně využívat mocný, ale nepříliš obvyklý makrojazyk T_EXu.

Text si neklade za cíl popsat naprosto přesně veškeré vlastnosti T_EXu. K tomu slouží *T_EXbook naruby*, na který v textu často odkazuji zkratkou TBN. Vybral jsem to, co předpokládám, že pragmatik potřebuje hlavně použít a čemu potřebuje rozumět. Přesto je nakonec v textu zmíněno zhruba 90 % všech příkazů T_EXu a maker plainT_EXu.

Předpokládám, že čtenář má nainstalovanou nějakou T_EXovou distribuci [22, 23] včetně ČSplainu ve verzi aspoň Dec. 2012. Má tedy v systému připraveny příkazy `csplain` resp. `pdfcsplain`, kterými bude zpracovávat dokumenty s výstupem do DVI resp. PDF. Není-li toto splněno, je možné zjistit více informací o instalaci ČSplainu na webové stránce <http://petr.olsak.net/csplain.html> a zde v dodatku A.

Toto je pracovní, i když zveřejněná, verze textu pro začínající T_EXisty. V tuto chvíli je text zcela dokončen a zpracovávají se korektury...

© Petr Olšák, 2013, 2014

Verze textu: 19. 9. 2014

URL: <http://petr.olsak.net/tpp.html>

Obsah

1 Úvod do T_EXu	6
1.1 Jednoduchý dokument	6
1.2 Členění dokumentu – obsah a forma	7
1.3 Názvy souborů, vyhledávání souborů	9
1.4 Uživatelské prostředí	9
1.5 Řídící sekvence, příkazy, makra, registry	10
2 Zpracování vstupu	11
2.1 Pravidla o mezerách a odstavcích	11
2.2 Pravidla o řídících sekvencích	12
2.3 Seznam speciálních znaků	12
2.4 Změny kategorií	13
2.5 Tokenizace	13
2.6 Potlačení speciálních znaků (verbatim)	13
3 Jednotlivé znaky ve výstupu	14
3.1 Přímý tisk znaků	14
3.2 Znaky sestavené z akcentů	14
3.3 Znaky implementované jako řídící sekvence	15
3.4 Ligatury, pomlčky	15
3.5 Mezery	16
3.6 Příkaz <code>\char</code>	16
3.7 Mírná odlišnost od ASCII u některých fontů	17
4 Hladká sazba	18
4.1 Základní parametry pro formátování odstavce	18
4.2 Automatické dělení slov	19
4.3 Další parametry pro řádkový zlom	20
4.4 Vertikální, odstavcový a další módy	20
4.5 Umístění sazby na stránce	21
5 Fonty	22
5.1 Skupiny v T _E Xu	22
5.2 Příkaz <code>\font</code>	23
5.3 Fontové soubory, výběr rodiny fontů	24
5.4 Zvětšování a zmenšování fontů v C _S plainu	25
5.5 Italská korekce	27
6 Matematická sazba	28
6.1 Příklad a základní vlastnosti	28
6.2 Soubory maker <code>ams-math.tex</code> a <code>tx-math.tex</code>	29
6.3 Matematické abecedy	30
6.4 Zlomky	31
6.5 Matematické symboly a znaky	31
6.6 Závorky	37
6.7 Odmocniny, akcenty	38
6.8 Speciality	39
6.9 Krájení vzoreček do více řádků	42
6.10 Přidání další matematické abecedy	43
7 Použití OPmac	45
7.1 Velikosti fontů a řádkování	45
7.2 Okraje	46
7.3 Členění dokumentu	47

7.4	Další číslované objekty a odkazy na ně	47
7.5	Odrážky	49
7.6	Tvorba automaticky generovaného obsahu	50
7.7	Barvy, vodoznaky	51
7.8	Klikací odkazy	51
7.9	Verbatim texty	52
7.10	Tabulky	54
7.11	Vkládání obrázků	56
7.12	Poznámky pod čarou a na okraji	57
7.13	Bibliografické údaje	58
7.14	Sestavení rejstříku	61
7.15	Poslední strana	63
8	Programování maker	64
8.1	Makrozáklady	64
8.2	Tanec s parametry	66
8.3	Numerické výpočty	68
8.4	Větvení	69
8.5	Cykly	71
8.6	Další příkazy, bez nichž se opravdový makroprogramátor neobejde	72
9	Boxy, linky, mezery	75
9.1	Pružné a pevné mezery	76
9.2	Centrování	76
9.3	Boxy s vyčnívající sazbou	77
9.4	Linky	79
9.5	Další manipulace s boxy	80
10	Co se nevešlo jinde	83
10.1	Vertikální mezery a stránkový zlom	83
10.2	Plovoucí objekty	85
10.3	Dekorace stránek, výstupní rutina	86
10.4	Čtení a zápis textových souborů	88
10.5	Ladění dokumentu, hledání chyb	91
11	Možnosti pdfTeXu	93
11.1	Základní parametry	93
11.2	Dodatečné informace k PDF	93
11.3	Nastavení výchozích vlastností PDF prohlížeče	94
11.4	Hyperlinky	96
11.5	Klikací obsahy po straně prohlížeče	98
11.6	Lineární transformace sazby	98
11.7	Vkládání externí grafiky	99
11.8	Stránková montáž pdfTeXem	101
11.9	Využití elementárních PDF příkazů pro grafiku	102
11.10	Mikrotypografická rozšíření	108
A	Generování formátů	111
A.1	Módy INITEX a VIRTEX	111
A.2	Generování formátu C _S plain	112
B	Všeliká rozšíření TeXu	114
B.1	eTeX	114
B.2	X _Y TeX	117
B.3	LuaTeX	120

C	Numerické a metrické údaje	122
C.1	Numerický údaj	122
C.2	Metrický údaj	122
D	Dvouzobáková konvence	124
E	Vstupní kódování, $\text{enc}\text{T}_{\text{E}}\text{X}$, UTF-8	125
E.1	$\text{Enc}\text{T}_{\text{E}}\text{X}$	125
E.2	$\text{Enc}\text{T}_{\text{E}}\text{X}$ v \LaTeX	126
E.3	Vstupní kódování v $\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ a $\text{Lua}\text{T}_{\text{E}}\text{X}$	127
F	Fonty v $\text{T}_{\text{E}}\text{X}$ové distribuci	128
F.1	Základní přehled $\text{T}_{\text{E}}\text{X}$ ových fontů	128
F.2	Instalace nového OTF fontu	130
F.3	Kódování textových fontů	133
G	Více jazyků v \LaTeX	136
H	Literatura a odkazy	138
I	Rejstřík řídicích sekvencí	139

Kapitola 1

Úvod do T_EXu

1.1 Jednoduchý dokument

Pomocí textového editoru můžete vytvořit soubor třeba s názvem `pokus.tex` a s tímto obsahem:

```
{\bf Pokusný dokument}
```

Vstupní soubor zpracovaný T_EXem nazýváme {\it zdrojový text dokumentu}. Ten pořizuje autor dokumentu v běžném textovém editoru, který nepřidává do souboru žádné nepatřičné neviditelné znaky.

Odstavce ve zdrojovém textu jsou odděleny prázdným řádkem. Jejich rozdělení do řádků nemá vliv. T_EX si každý odstavec nakonec zformátuje podle vlastního uvážení a na základě nastavení interních parametrů.

Pro logo T_EXu je použita sekvence {\tt\char'\TeX}. Dále jsou připraveny sekvence na vyznačování: {\tt\char'\it} pro kurzívu, {\tt\char'\bf} pro tučný řez a {\tt\char'\tt} pro strojopis. Vyznačovaný text musí být obklopen složenými závorkami a uvedená sekvence musí být vložena dovnitř těchto závorek před vyznačený text.

% Toto je komentář, který se netiskne.

Do zdrojového textu je možno vložit komentář od procenta do konce řádku.

Komentář je T_EXem ignorován. % Toto je taky komentář.

Je-li potřeba vytisknout %, je nutné před ně vložit zpětné lomítko.

Krátkou pomlčku zapišeme pomocí -- a dlouhou pomocí ---.

Matematický vzorec píšeme mezi dolary: $\$a^2 + b^2 = c^2\$$.

Na konec zdrojového textu dokumentu je potřeba vložit sekvenci {\tt\char'\bye}.

```
\bye
```

Vše, co je napsáno za sekvencí \bye je T_EXem ignorováno.

Nyní můžete tento dokument zpracovat T_EXem s připraveným formátem C_Splain pomocí příkazu uvedeném na příkazovém řádku:

```
csplain pokus
```

Tento příkaz vytvoří soubor `pokus.dvi`¹⁾. Dnes se doporučuje místo DVI raději přímo vyrobit PDF soubor, tedy v tomto příkladě `pokus.pdf`. K tomu slouží příkaz:

```
pdfcsplain pokus
```

¹⁾ Přípona `.dvi` odpovídá binárnímu formátu DVI (DeVice Independent). Je to implicitní výstupní formát T_EXu, který je možné následně zpracovat programy `xdvi`, `evince`, `kdví` (prohlížení v X Window Systemu), `YAP`, `windvi` (prohlížení v MS Windows), `dvips` (konverze do PostScriptu), `dvipdf`, `dvipdfm` (konverze do PDF).

Na terminálu se objeví hlášení o použité verzi \TeX u a \LaTeX u, dále o tom, že byla vytvořena jedna stránka [1] a kam byl uložen výstup. Výsledek si můžete prohlédnout prohlížečem PDF nebo DVI a vypadá takto:

Pokusný dokument

Vstupní soubor zpracovaný \TeX em nazýváme *zdrojový text dokumentu*. Ten pořizuje autor dokumentu v běžném textovém editoru, který nepřidává do souboru žádné nepatřičné neviditelné znaky.

Odstavce ve zdrojovém textu jsou odděleny prázdným řádkem. Jejich rozdělení do řádků nemá vliv. \TeX si každý odstavec nakonec zformátuje podle vlastního uvážení a na základě nastavení interních parametrů.

Pro logo \TeX u je použita sekvence `\TeX`. Dále jsou připraveny sekvence na vyznačování: `\it` pro kurzívu, `\bf` pro tučný řez a `\tt` pro strojopis. Vyznačovaný text musí být obklopen složenými závorkami a uvedená sekvence musí být vložena dovnitř těchto závorek před vyznačený text.

Do zdrojového textu je možno vložit komentář od procenta do konce řádku. Komentář je \TeX em ignorován. Je-li potřeba vytisknout %, je nutné před ně vložit zpětné lomítko.

Krátkou pomlčku zapíšeme pomocí `-` a dlouhou pomocí `—`. Matematický vzorec píšeme mezi dolary: $a^2 + b^2 = c^2$.

Na konec zdrojového textu dokumentu je potřeba vložit sekvenci `\bye`.

● **Cvičení 1** ● Zpracování tohoto dokumentu \LaTeX em si vyzkoušejte na svém počítači. Pokud jste dostali výsledek s poničenými českými znaky, je to tím, že máte zdrojový text v jiném kódování, než v jakém jej předpokládá \LaTeX . Od verze \LaTeX u Dec. 2012 se předpokládá vstupní kódování v UTF-8.

K vytištění zpětného lomítka (`\`) je ve zdrojovém textu použita konstrukce `\char‘\`. Znak `‘` za slovem `\char` je tzv. *zpětný apostrof* (na klávesnici vlevo nahoře). Přímý apostrof (na klávesnici vpravo) naopak vypadá takto `'` a používá se pro jiné účely.

● **Poznámka** ● Pokud se ve zdrojovém textu dokumentu vyskytne nějaký překlep nebo chyba v řídicí sekvenci nebo v jiné konstrukci důležité pro zpracování, \TeX se typicky zastaví a na terminálu oznámí chybu. Je možné jej pomocí Enter přinutit k pokračování zpracování. Pomocí klávesy **X** ukončíte předčasně zpracování. O dalších možnostech pojednává sekce 10.5.

1.2 Členění dokumentu – obsah a forma

Předchozí ukázka neukazuje základní výhodu \TeX u: možnost oddělit obsah od formy. Obsah může psát autor textu, který bude pouze poučen o tom, že odděluje odstavce prázdnými řádky a naučí se používat malé množství značek, kterými vyznačí strukturu svého dokumentu. Tyto značky pro něj naprogramuje programátor *maker*. Ten se stará o dvě věci: nejen navrhuje a programuje značky pro autora, ale taky tímto programováním vytváří výstupní vzhled dokumentu.

Především je zcela nesprávné psát nadpis pomocí `{\bf Text nadpisu}`. Programátor *maker* by měl definovat například značku `\titul` a poučit autora textu, aby používal pro nadpis tuto značku. Programátor dále rozhodne, jak velkým fontem bude nadpis vytištěn, jaké budou kolem nadpisu mezery atd. Řeší tedy také typografii dokumentu. Pro autora

textu je poněkud kryptické používat `{\tt\char'\bf}` pro vyznačení řídicí sekvence. Programátor maker mu tedy připraví značku `\seq`.

Představme si, že programátor maker dodal autorovi soubor `makra.tex` a vysvětlil mu, že za značku `\titul` napíše text titulu ukončený prázdným řádkem a za značku `\seq` píše název řídicí sekvence ve svorkách. Autor pak vytvořil následující text:

```
\input makra % načtení maker dodaných programátorem/typografem
```

```
\titul Pokusný dokument
```

Vstupní soubor zpracovaný `\TeX`em nazýváme *{it zdrojový text dokumentu}*. Ten pořizuje autor dokumentu v běžném textovém editoru, který nepřidává do souboru žádné nepatřičné neviditelné znaky.

Pro logo `\TeX`u je použita sekvence `\seq{\TeX}`. Dále jsou připraveny sekvence na vyznačování: `\seq{it}` pro kurzívu, `\seq{bf}` pro tučný řez a `\seq{tt}` pro strojopis.

Na konec zdrojového textu je potřeba vložit sekvenci `\seq{bye}`.
`\bye`

Obsah souboru `makra.tex` uvedený níže může připadat čtenáři poněkud záhadný. Pokud ale bude pozorně číst i další kapitoly této knihy, bude mu posléze vše jasné. Nyní jsme teprve na začátku, takže ponecháme tento kód nevysvětlen.

```
% Zde jsou definice, které připravil typograf:
\chypb          % csplain, aktivace českých vzorů dělení slov
\input ncncnt    % Rodina fontů New-Century
\def\sizespec{at11pt} \resizeall \tenrm % Velikost písma 11pt
\baselineskip=13pt          % Velikost řádkování 13pt
\letfont\bigbf=\tenbf at14.4pt      % font pro nadpis
\def\titul#1\par{\bigskip          % definice nadpisu
  \centerline{\bigbf#1\unskip}\nobreak\medskip}
\def\seq#1{{\tt\char92 #1}}          % definice tisku sekvence
```

● **Cvičení 2** ● Vyzkoušejte vytvořit dva soubory `makra.tex` a `dokument.tex` s výše uvedeným obsahem a zpracovat je příkazem `pdfcsplain dokument`. Měli byste dostat něco takového:

Pokusný dokument

Vstupní soubor zpracovaný `\TeX`em nazýváme *zdrojový text dokumentu*. Ten pořizuje autor dokumentu v běžném textovém editoru, který nepřidává do souboru žádné nepatřičné neviditelné znaky.

Pro logo `\TeX`u je použita sekvence `\TeX`. Dále jsou připraveny sekvence na vyznačování: `\it` pro kurzívu, `\bf` pro tučný řez a `\tt` pro strojopis.

Na konec zdrojového textu je potřeba vložit sekvenci `\bye`.

Je možné spojit dokument do jediného souboru, tj. místo příkazu `\input makra` vložit přímo obsah maker do tohoto místa. Je ale obvyklejší to oddělit, tj. nerozptylovat autora poněkud kryptickými makry.

V různých souborech mohou být pro autora připraveny různé sady maker řešící například odlišně vzhled dokumentu. Takové sadě maker se říká styl. Autor pak může zvolit vhodný styl pro svůj dokument pomocí výběru odpovídajícího souboru maker.

Ačkoli je často autor i programátor maker jedna a tatáž osoba, je velmi rozumné neustále mít na paměti oddělení obsahu dokumentu od formy. Třebaže je \TeX tolerantní a umožní masit kódy maker kdekoli, třeba uprostřed rozepsaného dokumentu, měli bychom dodržovat jistou kulturu při tvorbě zdrojových kódů. Mezi značky pro autora patří značka pro vymezení nadpisu, sekce, podsekce, značky pro zdůraznění částí textu (kurzívou nebo jinak) a plno značek pro tvorbu matematických vzorců. Autor může také používat další značky vymezující speciální strukturu dokumentu (odrážky, tabulky, vložení obrázku atd.). Vše ostatní spadá do domény programátora maker.

● **Cvičení 3** ● Po zpracování dokumentu z předchozí ukázky vznikl soubor `dokument.pdf` a taky protokol o zpracování `dokument.log`. Zkuste se do tohoto protokolu podívat a zjistit z něj, jaké soubory byly při zpracování dokumentu \TeX em načteny.

1.3 Názvy souborů, vyhledávání souborů

V předchozím textu jsme se seznámili s příkazem `\input`, za kterým následuje název souboru, který je \TeX em přečten. Tento název je *parametrem příkazu* a je ukončený mezerou. To prakticky znamená, že mezera není vhodný znak, který by se mohl vyskytovat v názvu souboru, pokud tento soubor chceme použít v \TeX u. Některá rozšíření \TeX u sice nabízejí jisté možnosti, jak vnutit příkazu `\input` i soubor obsahující v názvu mezery, ale je mnohem lepší mezery do názvů souboru nedávat. Stejně tak se nedoporučuje dávat do názvů souborů znaky s háčky a čárkami, chcete-li se vyhnout potížím.

Pozorný čtenář si jistě všiml, že v předchozí ukázce v parametru příkazu `\input` i na příkazové řádce je vynechána přípona souboru. \TeX v takovém případě nejprve přednostně hledá soubor se jménem, jak je v parametru napsáno. Pokud ho nenajde, zkusí připojit příponu `.tex` a hledá znovu. Přípona `.tex` je tedy při zpracování \TeX em významnější než jakákoli jiná a uživatel ji nemusí při specifikování souboru psát. Má-li jméno souboru jakoukoli jinou příponu, je třeba je napsat kompletně celé.

\TeX hledá soubory přednostně v aktuálním adresáři. Pokud tam soubor není, hledá jej ve struktuře adresářů \TeX ové distribuce. V jakém pořadí a podle jakých pravidel tuto adresářovou strukturu prohledává, závisí na použité \TeX ové distribuci a na její konfiguraci. Popis chování konkrétní \TeX ové distribuce najdete v dokumentaci k distribuci, nikoli v této příručce.

1.4 Uživatelské prostředí

\TeX není vázán na žádné konkrétní uživatelské prostředí. V tomto případě platí: jak si kdo ustele, tak si lehne. Jinými slovy, záleží na uživateli, jaké prostředí si vytvoří či použije. Jaký zvolí textový editor, zda bude z editoru klávesovou zkratkou volat překlad dokumentu \TeX em s formátem `CSplain`, v jakém programu si bude prohlížet výstupní podobu dokumentu atd. Konkrétní rady na toto téma v této příručce nenajdete. Autor tohoto textu popsal své uživatelské prostředí v článku [15], ale není to samozřejmě jediné možné řešení.

Na rozdíl od všelijakých příruček k jiným programům v této knize nehledejte rady typu Alt-Shift-levý klik myši udělá to či ono, menu programu vypadá tak či onak. Naším cílem je umět zkrotit \TeX a porozumět jeho chování. \TeX je pouhý interpret-formátor, který na vstupu čte zdrojový text a na výstupu tvoří DVI nebo PDF. Myši se s ním nedomluvíte.

1.5 Řídicí sekvence, příkazy, makra, registry

Činnost \TeX u v jednotlivých místech dokumentu je určena *řídicími sekvencemi* uvozenými zpětným lomítkem. Tyto řídicí sekvence mají v \TeX u přiděleny rozličné *významy*. Můžeme je rozdělit do pěti základních druhů:

- *Příkazy*. \TeX je vybaven před prvním čtením souborů před vygenerováním formátu¹⁾ asi třemi sty zabudovanými příkazy. Viz TBN, str. 332–458. Přímé použití těchto příkazů autorem textu je málo obvyklé, častěji se používají složené povely, neboli makra. V naší ukázce najdeme například příkazy `\font` (zavede nový font), `\def` (definuje makro). V literatuře se často příkazům říká *primitivní příkazy* nebo *primitivy*, aby se zdůraznilo, že jsou nedílnou součástí \TeX u samotného. To znamená, že nejsou důsledkem dodatečného „učení \TeX u“, kdy \TeX během generování formátu nebo během načítání dalších deklaračních částí dokumentu rozšiřuje repertoár řídicích sekvencí `\langleněco\rangle` o další. V této příručce budeme primitivním příkazům říkat krátce příkazy.
- *Makra* jsou povely složené z typicky více interních povelů a deklarované obvykle příkazem `\def`. Například řídicí sekvence `\TeX` je makro definované v souboru `plain.tex` takto: `\def\TeX{\T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX}` Když autor napíše do svého dokumentu `\TeX`, program \TeX si tento povel rozloží na následující: písmeno T, příkazy `\kern-.1667em \lower.5ex \hbox{E} \kern-.125em`, za kterými následuje ještě písmeno X. Kurzívou jsou zde vyznačeny parametry uvedených příkazů. V tuto chvíli je příliš brzo na podrobný rozbor významů těchto příkazů. Berte to jen jako ilustraci makra.
- *Registry* jsou něco jako „proměnné“ v \TeX u. Nesou obvykle numerický nebo metrický údaj. Rozlišujeme interní registry \TeX u, jejichž hodnoty nějak ovlivní sazbu, a deklarované registry, které mohou být použity v makrech. Například interní registr `\parindent` obsahuje velikost odstavcové zarážky nebo `\baselineskip` obsahuje velikost řádkování.
- *Znakové konstanty* jsou z interního pohledu \TeX u numerické konstanty deklarované příkazem `\chardef` nebo `\mathchardef`. V naší ukázce je použita znaková konstanta `\%`. Objeví-li se znaková konstanta v kontextu „vytiskni ji“, \TeX vysází znak s kódem rovným této konstantě. Třeba `\%` je deklarována pomocí `\chardef\%=37`, takže to je konstanta 37. V ukázce je řídicí sekvence `\%` použita v kontextu „vytiskni ji“, takže \TeX vysadí znak s kódem 37, což je procento.
- *Přepínače fontu* jsou deklarovány příkazem `\font`. Podrobněji viz sekci 5.2.

V dodatku I je abecední seznam řídicích sekvencí uvedených v této knize. U každé sekvence je zkratka určující její význam a odkaz na stránky, kde je sekvence vysvětlena nebo se o ní aspoň významným způsobem mluví. Na takovém místě je sekvence podbarvena žlutě.

Autor textu nebo uživatel s jednodušším myšlením vnímá všechny řídicí sekvence jako příkazy, protože mají za úkol typicky něco vykonat. My budeme ale důsledně rozlišovat mezi příkazy, makry, registry, znakovými konstantami a přepínači fontu.

V této knize se postupně seznámíme jednak s významnými příkazy a registry \TeX u, ale také s makry, které se \TeX naučil během generování formátů `plain` a `CSplain`. V kapitole 7 se také seznámíme s makry z balíku `OPmac`.

¹⁾ Formát \TeX u zahrnuje načtenou sadu maker, fontů a vzorů dělení slov. Podrobněji viz dodatek A.

Kapitola 2

Zpracování vstupu

V této kapitole popíšeme, jak \TeX zpracovává řádky vstupního souboru.

2.1 Pravidla o mezerách a odstavcích

Více mezer těsně vedle sebe se považuje za jedinou mezeru. Mezery na začátku řádku jsou zcela ignorovány. Konec řádku při přechodu na další řádek se promění v mezeru.¹⁾ Zcela prázdný řádek ukončí odstavec. Další zcela prázdné řádky (těsně následující po ukončení odstavce) už neprovedou nic. Příklad:

```
Text odstavce
na    dvou řádcích.
```

```
\bye
```

V ukázce je odstavec ukončený prázdným řádkem. Tento odstavec se uvnitř \TeX u promění v text:

```
Text odstavce na dvou řádcích.
```

Povšimneme si, že mezi slovy `odstavce` a `na` se objevila mezera, která původně byla přechodem na nový řádek. Dále snad nikomu neuniklo, že více mezer mezi slovy `na` a `dvou` se promění v mezeru jedinou.

Každý odstavec může být napsán ve vstupním souboru na libovolně mnoha řádcích, které \TeX podle popsaných pravidel nejprve převede na jeden interní řádek a zalomí si jej pak do výstupních řádků podle svého algoritmu.

● **Poznámka** ● Jednotlivé odstavce můžete do zdrojového textu napsat každý do jednoho řádku. Dříve textové editory skoro výhradně dlouhé řádky nezobrazovaly celé, ale jejich část utekla „za roh“. To pochopitelně snižuje přehlednost. Často se tedy používají textové editory, které dlouhé řádky automaticky lámou. Buď jsou to editory, které po rozlomení nechávají na koncích řádků tvrdé mezery (tj. vzniká skutečně více řádků) nebo to jsou editory, které dlouhý řádek lámou dynamicky jen pro zobrazení v okně editoru, ale ve skutečnosti ponechávají dlouhý řádek. \TeX u je to jedno. Podstatné ale je, že mezi odstavci je prázdný řádek (tj. dvakrát Enter, ne jen jednou jako u ostatních dnešních aplikací).

Používáte-li druhý typ textového editoru (s dynamicky zlomenými řádky jen pro zobrazení v editoru), pak musíte vědět, že komentář uvozený znakem `%` je ukončen až skutečným koncem řádku.

Autor této knížky preferuje editory prvního druhu a nechává řádky v odstavci zlomené natvrdo. Pak je takový text kompaktní a čitelný v libovolném textovém editoru. A člověk se v takovém textu lépe orientuje. Platí důležité pravidlo, které významně odlišuje \TeX od většiny jiných značkovacích jazyků: *zdrojový text dokumentu není určen jen pro stroj, ale také pro člověka, který v něm píše, upravuje, hledá atd.* Je tedy žádoucí i zdrojový text psát co nejpřehledněji.

● **Cvičení 4** ● Prověřte vlastnosti svého textového editoru. Láme řádky automaticky a vkládá na jejich konce pevná odřádkování nebo zobrazuje dlouhé řádky jen dynamicky zalomené? Prozkoumejte další možnosti svého textového editoru v návaznosti na jeho použití v \TeX u.

¹⁾ Je-li konec řádku schován za komentářovým znakem (procentem), pak mezera nevzniká.

2.2 Pravidla o řídicích sekvencích

Řídicí sekvence začíná znakem `\` a pokud následují písmena, je za řídicí sekvenci považována maximální souvislá sekvence těchto písmen. Teprve první další znak, který není písmenem, už není do této sekvence zahrnut. Je-li tímto ukončovacím znakem mezera, je ignorována. Jiné znaky za řídicí sekvenci (například číslice) ignorovány nejsou. Druhé pravidlo sestavení řídicí sekvence: pokud těsně za znakem `\` nenásleduje písmeno, ale jiný znak, `TeX` sestaví řídicí sekvenci tvaru `\langle znak \rangle`, kde `\langle znak \rangle` je jediný znak v řídicí sekvenci. Vše, co následuje, je normálně zpracováno. Následuje-li v tomto případě mezera, není ignorována.

Za písmena jsou implicitně v `TeXu` považovány znaky `A–Z`, `a–z`. V `CSplainu` jsou za písmena považovány také znaky české a slovenské abecedy `Á–ž`. Na velikosti písmen záleží. Ostatní znaky písmena nejsou a mohou vytvořit (při výchozím nastavení) jen řídicí sekvenci tvaru `\langle znak \rangle` dle druhého pravidla. Příklady řídicích sekvencí:

```
\totojedlouharidicisekvence \totojeřidicisekvencevcsplainu
A něco užitečnějšího: Můžeme psát v TeX u, přitom logo TeX je vytvořeno
řídicí sekvencí. Několik % lidí možná bude překvapeno.
Konečně si rozmyslete co udělá \it1 a \it2.
\bye
```

• **Cvičení 5** • Zpracujte tuto ukázkou `CSplainem`. Shledáte, že nejprve si `TeX` zanádá, že nemá definovány první dvě řídicí sekvence. Na to reagujte klávesou `Enter`. Ve výstupu vidíte, že je spojeno logo `TeX` s následujícím znakem `u`, protože mezera za řídicí sekvencí je ignorována. Méně žádoucí je spojení loga `TeX` se slovem `je`. Ani více mezer nepomohlo, ty se nejprve promění v jednu a ta je ignorována. Na druhé straně za sekvencí `%` se mezera zachová, protože to je sekvence `\langle znak \rangle` sestavená podle druhého pravidla.

Na předposledním řádku ukázky se zdá, že tam jsou dvě různé řídicí sekvence, ale zdání klame. Je tam dvakrát stejná sekvence `\it`, jednou za ní těsně následuje jednička a podruhé dvojka. Protože `\it` přepíná do kurzívy, uvedený řádek dá tento výstup: Konečně si rozmyslete co udělá *1* a *2*. Mezi zápisem `\it1` a `\it 1` není žádný rozdíl.

Vynutit si mezeru za řídicí sekvencí lze různými způsoby. Třeba lze použít příkaz `\L` (zpětné lomítko následované mezerou), který vytvoří mezeru: `TeX\` . Další možností je použít závorky pro parametry a skupiny `{}`, které v sazbě neudělají nic, ale řídicí sekvenci od dalšího textu oddělí. Vypadá to pak jak makro s prázdným parametrem:

Můžeme psát v `TeX{u}`, přitom logo `TeX{}` je vytvořeno...

2.3 Seznam speciálních znaků

Kromě mezery a zpětného lomítka interpretuje `TeX` ještě některé další znaky speciálním způsobem. V následující tabulce je jejich přehled.

znak	význam	kategorie
mezera	speciální chování, viz sekci 2.1	10
<code>\</code>	zahajuje řídicí sekvenci, viz sekci 2.2	0
<code>{ }</code>	obklopují lokální skupinu nebo parametry	1, 2
<code>%</code>	zahajuje komentář do konce řádku	14
<code>\$</code>	zahajuje/končí sazbu v matematickém módu	3
<code>&</code>	separátor v tabulkách	4
<code>#</code>	označení parametrů maker	6
<code>^ _</code>	konstruktor mocniny a indexu	7, 8
<code>~</code>	aktivní znak, nedělitelná mezera	13

Chcete-li vytisknout znaky %, \$, & a #, je třeba přidat dopředu zpětné lomítko, tedy psát `\%`, `\$`, `\&` a `\#`. Pokud chcete vytisknout libovolný speciální znak (nejen %, \$, & a #), lze použít konstrukci `{\tt\char'\langleznak\rangle}`. Například `{\tt\char'\backslash}` vytiskne zpětné lomítko.

2.4 Změny kategorií

Je dobré vědět, že uvedené speciální znaky nejsou speciální „od přírody“, ale proto, že mají přidělenou speciální kategorii uvedenou v posledním sloupci tabulky. Každému znaku lze změnit kategorii příkazem `\catcode⟨kód⟩=⟨hodnota⟩`. Takže každý znak se může chovat jako zpětné lomítko, dostane-li přidělenou kategorii 0, každý znak se může chovat jako {, dostane-li přidělenou kategorii 1 atd. Znaky, které se přímo tisknou, mají nastavenou kategorii 11 (písmeno) a 12 (ostatní znak).

• Cvičení 6 • Vyzkoušejte:¹⁾

```
\catcode'\=12 Tady už znak \ nevytváří žádné řídicí sekvence
ovšem teď nemůžeme jednoduše napsat \catcode'\=0 abychom to vrátili
zpět, protože zápisem \catcode jsme nevytvořili řídicí sekvenci.
Nejsme schopni ani ukončit dokument pomocí \end nebo \bye.
```

Uvedený příklad ukazuje, že změny kategorií jsou dosti nebezpečné a měli by k nim přikročit až pokročilí \TeX isté například po přečtení odstavce 1.3 z TBN.

Pohlédneme-li do souborů maker k \LaTeX u, vidíme, že se to tam hemží řídicími sekvencemi, které obsahují zavináče @. Je to proto, že \LaTeX nastavuje zavináči při čtení souborů maker kategorii 11 (písmeno), takže se může vyskytovat v názvech řídicích sekvencí. Na konci čtení souboru s makry se vrací kategorie zavináče na hodnotu 12, takže autor textu nemůže jednoduše interní řídicí sekvenci definovanou v souborech maker použít nebo měnit.

2.5 Tokenizace

Při čtení řádků vstupního souboru probíhá takzvaná tokenizace. Každý znak se stává *tokenem*, což je žetonek nesoucí kód tohoto znaku a jeho kategorii v době přečtení ze vstupního souboru. \TeX si ukládá do své paměti (ve formě maker, obsahu parametrů atd.) výhradně řetězce tokenů, nikoli znaky samotné. Kategorie jednou přečteného znaku (uloženého například v makru) se nedá později jednoduše změnit, tj. zůstává ke znaku přirostlá, třebaže je později změněn údaj `\catcode`. Tato změna se týká jen nově čtených znaků.

Token si lze představit jako dvojici [$\langle znak \rangle$, $\langle kategorie \rangle$] s jedinou výjimkou: řídicí sekvence. Ta se po přečtení stává v \TeX u jediným tokenem, tentokrát nesoucí informaci o názvu této řídicí sekvence. Takový token nemá kategorii.

2.6 Potlačení speciálních znaků (verbatim)

Chcete-li v \TeX u vytisknout nějaký souvislý text bez interpretace speciálních znaků tak, jak je zapsán ve zdrojovém textu, včetně rozdělení do řádků, je potřeba použít makro, které mění ve vymezené oblasti kategorie všech znaků na hodnotu 11 nebo 12. Při použití `OPmac` (viz sekci 7.9) jsou k tomuto účelu připravena makra `\begtt`, `\endtt` a `\activettchar`.

¹⁾ Přiřazení `\catcode'\=12` je shodné s přiřazením `\catcode92=12`, kde číslo 92 je ASCII kód znaku \. O různých zápisech konstant v \TeX u je možné se dočíst v dodatku C.

Kapitola 3

Jednotlivé znaky ve výstupu

3.1 Přímý tisk znaků

Jednoduše: napíšete A a vytiskne se A. Takto tiskneme v plainT_EXu písmena anglické abecedy, číslice, interpunkci (tečka, čárka, dvojtečka, středník, vykřičník, otazník) a další znaky * / [] () + =. V C_Splainu lze takto tisknout i znaky české a slovenské abecedy Á á Ä ä Č č Ď ě Ě ě Í í Ĺ ĺ Ľ ľ Ľ ľ Ň ň Ó ó Ö ö Ô ô Ř ř Š š Ť ť Ú ú Ů ů Ű ű Ý ý Ž ž. C_Splain od verze Dec. 2012 jednoznačně předpokládá, že vstupní soubory kódují tyto znaky v UTF-8. Existují některé další znaky přímo podporované C_Splainem a je možné základní podporovanou sadu znaků rozšířit. Tyto věci jsou popsány v dodatku E.

3.2 Znaky sestavené z akcentů

Můžete použít následující makra na přidání akcentu k písmenu:

<code>\‘a</code> à	<code>\=a</code> ā	<code>\~a</code> ã	<code>\d a</code> ą
<code>\’a</code> á	<code>\^a</code> â	<code>\"a</code> ä	<code>\t a</code> â
<code>\v a</code> ǎ	<code>\.a</code> ȁ	<code>\b a</code> ȁ	<code>\r a</code> ȁ (jen po <code>\csaccents</code>)
<code>\u a</code> ǻ	<code>\H a</code> ǻ	<code>\c a</code> ȳ	

C_Splain nabízí makro `\csaccents`, které lze uvést jednou na začátku dokumentu. Toto makro přeprogramuje výše uvedená makra na sestavení akcentu tak, že v případě znaků z české a slovenské abecedy nevytvářejí kompozitní znaky, ale celistvé znaky. Pak se třeba `\v` d promění správně v ě, zatímco bez použití `\csaccents` se `\v` d promění v nesprávné ě.

Je samozřejmě daleko přirozenější psát české texty přímo (jak je uvedeno v sekci 3.1) než pou`\v` z`\’`ivat p`\v` repisy pomoc`\’i` akcent`\r` u. Druhou možnost využijete jen ve výjimečných krátkých textech. Například posílám-li svůj článek do zahraničí, píšu své jméno jako Ol`\v` s`\’`ak. Pak mám jistotu, že nikdo moje jméno neponičí různými konverzemi kódování, protože text obsahuje jen ASCII znaky a tuto notaci dokáže zpracovat T_EX neměnným způsobem už zhruba třicet let.

● **Poznámka** ● Pozorný čtenář si jistě všiml, že z důvodů popsaných v sekci 2.2 píšeme za řídicími sekvencemi pro sestavení akcentu tvaru `\<písmeno>` mezeru, která oddělí sekvenci od následujícího znaku. U sekvencí `\<znak>` (kde `<znak>` není písmenem) naopak mezeru nepíšeme. To nevypadá příliš jednotně a mezery navíc ruší, neboť vytvářejí klamné zdání, že je to mezera mezi slovy. Na pomoc mohou přijít relativně neškodné závorky vymezující skupiny nebo hranice parametrů. Třeba slovo Olšák lze zapsat jako Ol`\v{s}``\’{a}`k.

● **Cvičení 7** ● Najděte v souboru `plain.tex` definice maker na sestavení akcentu. Zjistíte, že se opírají o příkaz `\accent<kód>`, kde `<kód>` je kód znaku, který se použije jako akcent nad následujícím písmenem.

● **Poznámka** ● Načtení souborů `utf8lat1.tex` a `utf8lata.tex` do dokumentu umožňuje v C_Splainu přímé použití vstupních znaků z částí Unicode tabulky Latin1 Supplement a Latin Extended-A bez nutnosti psát výše uvedené řídicí sekvence. Podrobněji viz dodatek E.

3.3 Znaký implementované jako řídicí sekvence

Některé znaky, které nemusejí být na všech klávesnicích dostupné, je možné zapsat jako řídicí sekvenci. Následující výpis ukazuje seznam těchto znaků.

```
plainTeX: \ss ß \l l \L L \ae æ \oe œ \AE Æ \OE OE
          \o ø \O Ø \i i \j J \aa å \AA Å
          \S § \P ¶ \copyright © \dots ... \dag † \ddag ‡
navíc csplain: \clqq „ \crqq “ \elqq “ \erqq ” \elq ‘ \erq ’
          \flqq « \frqq » \promile ‰
```

Uvedené řídicí sekvence jsou implementovány jako znakové konstanty nebo makra v závislosti na tom, zda znak je dostupný ve fontu nebo ne.

Sekvence `\clqq` a `\crqq` se používají jako levá a pravá česká uvozovka. Pro „uvozený text“ je v `CSplainu` připraveno makro `\uv`, které se používá takto: `\uv{uvozený text}`.

Místo psaní řídicích sekvencí z předchozího výpisu je možné v `CSplainu` použít přímo znaky, které chcete vytisknout, jako UTF-8 kódy ve vstupním souboru. Takže například stačí jednoduše napsat „uvozený text“, pokud Vám to umožní ovladač klávesnice. Nebo v právnických textech bohatě využijete přímo znak §, který nemusíte přepisovat jako `\S`.

O tom, jak použít další znaky, pojednává sekce 6.5 (pro matematické vzorečky) a dodatky E a F.3 (obecně).

• **Poznámka** • Znak `"`, který je obvykle dostupný na anglické klávesnici, nijak nesouvisí s českými uvozovkami. Je to znak pro jednotku „palec“. PlainTeX jej nicméně transformuje na levou anglickou uvozovku, která vypadá hodně podobně jako česká pravá. V angličtině uvozený text vypadá takto: “quoted text”.

3.4 Ligatury, pomlčky

Ligatura (česky slitek) je shluk dvou nebo více znaků, které dohromady vytvářejí jeden znak ve fontu. Za tradiční ligatury se považuje dvojice `fi`, která přechází ve `fl`, trojice `ffi`, která vytvoří `ffl`, dále `fl` se slije na `fl`, `ff` přejde na `ff` a konečně `ffl` vytvoří `ffl`. Starodávná písma nebo písma exotických jazyků mohou obsahovat další ligatury.

TeX si přečte tabulku ligatur z použitého fontu a na základě ní automaticky vytváří ligatury. Uživatel se o to nemusí starat, takže napíše třeba `fialka` a vytiskne se `fialka`, pokud je ligatura v metrice použitého fontu deklarována.

Kromě výše uvedených tradičních ligatur souvisejících s písmenem `f` zavedl Knuth další speciální TeXové ligatury do fontů Computer Modern. Většina dalších fontů, které jsou implementovány pro TeX, tuto koncepci TeXových ligatur přebírá. Následuje seznam TeXových ligatur:

```
-- → -      --- → —      ‘ ‘ → “      ’ ’ → ”      ? ‘ → ¿      ! ‘ → ¡
```

První dvě ligatury tvoří krátkou a dlouhou pomlčku, kterou typicky nemáme jednoduše přístupnou na klávesnici. Je pohodlné pomlčky psát jako `--` nebo `---` a zdrojový text je srozumitelný. Další dvě ligatury se používají pro anglický “quoted text”. Poslední dvě ligatury se asi moc nepoužívají, Španělě zřejmě budou své znaky psát do textu přímo.

• **Poznámka** • V typografii se používá daleko víc „vodorovných čárek“, než máme běžně přístupných na klávesnici. Běžný znak z klávesnice „-“ se promění ve *spojovník* „-“, což je nejkratší a poněkud tučnější vodorovná čárka používaná na rozdělování slov na koncích řádků (to dělá TeX automaticky). Dále se spojovník vyskytuje ve spojení „je-li“, nebo „slovník česko-německý“. *Krátká pomlčka* „--“ se užívá ve významu „až“, například *strany 14--16*, *dálnice Praha--Brno*. V češtině a slovenštině se tato pomlčka

používá i jinde, například k oddělení myšlenek ve větě. Pak musí být oddělena z obou stran mezerami a neměla by stát na začátku dalšího řádku. Dlouhá pomlčka „---“ se vyskytuje v anglickém textu, v češtině jen výjimečně. Konečně znak *minus* „\$-“ se používá v matematické sazbě. Srovnajte nesprávné -1 a správné −1. Pravidla sazby takových věcí jsou upravena v normě ČSN 01 6910 [1].

3.5 Mezery

Jak bylo řečeno v sekci 2.1, více mezer se promění v jednu, a pokud tato není součástí syntaktického pravidla (např. pro ukončení parametru nebo řídicí sekvence), pak se v odstavci vytiskne. Je to tzv. zlomitelná a pružná mezislovní mezer. Zlomitelná proto, že \TeX může při sazbě odstavce zlomit v této mezeře řádek. Pružná proto, že přizpůsobí v mírné toleranci svůj rozměr tak, aby byla sazba odstavce realizována do bloku (což je implicitní nastavení formátování odstavce).

Zlomitelnou pružnou mezislovní mezeru lze vytvořit také příkazem $\backslash\hspace{}$ (zpětné lomítko následované mezerou) nebo makrem \backslashspace . Příkaz $\backslash\hspace{}$ mohou výjimečně použít autoři textů jako „vynucenou mezeru“, zatímco makro \backslashspace je užitečné spíše pro tvůrce maker.

Kromě toho je v plain \TeX u připraven aktivní znak vlnka „~“, který vytvoří nezlomitelnou pružnou mezislovní mezeru. Používá se například za neslabičnými předložkami, abychom \TeX u zabránili lámat řádky tak, že na konci řádku zůstane třet neslabičná předložka. Píšeme třeba $\text{\textit{v}}\text{\textit{le}}\text{\textit{se}}$, $\text{\textit{k}}\text{\textit{v}}\text{\textit{e}}\text{\textit{s}}\text{\textit{n}}\text{\textit{i}}\text{\textit{c}}\text{\textit{i}}$.¹⁾ Protože je otrava pořád na to myslet, existují programy, které doplní tuto vlnku za neslabičné předložky do zdrojového textu dokumentu dodatečně²⁾. Nezlomitelná mezer musí být taky na dalších místech, která už automat sám nepozná a autor (nebo korektor) si to musí pohlídat. Typický výskyt nezlomitelné mezery je mezi číselným vyjádřením a navazujícím slovem, zkratkou či symbolem, například³⁾:

Odstavec~3, 100~km/h, §~17 zákona 111/1998~Sb., 7.~kavalérie.

Makro \backslashquad vytvoří zlomitelnou nepružnou mezeru velikosti čtverčíku (tj. velikosti písma). Makro \backslashqqquad vloží mezeru dvojnásobku \backslashquad a makro \backslashenskip vytvoří mezeru polovičního \backslashquad . Makro \backslashthinspace vloží tenkou nezlomitelnou nepružnou mezeru.

Příkaz $\backslashhskip\langle velikost \rangle$ vytvoří zlomitelnou nepružnou mezeru dané velikosti, například $\backslashhskip 1.5mm$ vytvoří mezeru velikosti 1,5 mm. Zápis $\backslashnobreak\hskip\langle velikost \rangle$ vytvoří nezlomitelnou nepružnou mezeru dané velikosti. Konečně příkaz \backslashhskip může mít ve svém parametru pokračování ve tvaru $\backslashhskip\langle velikost \rangle plus\langle velikost \rangle minus\langle velikost \rangle$, což umožní implementovat pružnou mezeru se specifikovanou tolerancí roztažení a stažení. O této možnosti pojednává sekce 9.1 a dále TBN na straně 77.

3.6 Příkaz \backslashchar

Příkaz $\backslashchar\langle kód \rangle$ vytiskne znak s daným kódem z aktuálně nastaveného fontu. Parametr $\langle kód \rangle$ je numerický údaj⁴⁾. Například $\backslashchar65$ vytiskne A, protože ASCII kód znaku A je 65.

¹⁾ Poznamenávám, že použití nezlomitelné *pružné* mezery za neslabičnými předložkami je správné, zatímco většina textových procesorů používá v tomto případě nezlomitelnou *nepružnou* mezeru, což je z typografického pohledu nevhodné. Mezera za předložkami má pružit stejně jako mezer mezi slovy.

²⁾ Například <http://petr.olsak.net/ftp/olsak/vlna/> nebo soubor maker *vlna.tex* pro \encTeX .

³⁾ Typografové doporučují mezi číslem a jednotkou psát tenkou nezlomitelnou nepružnou mezeru, což není v ukázce předvedeno. V takovém případě potřebujete psát $100\backslashthinspace km/h$ nebo při použití OPmac stačí psát $100\backslash,km/h$. Zápis $100~km/h$ je ale v souladu s normou ČSN 01 6910 [1], která pružnost a velikost mezer nerozlišuje.

⁴⁾ Jaké jsou v \TeX u možnosti pro zápis numerického údaje, pojednává dodatek C.

• **Cvičení 8** • Vraťte se k příkladu tisku speciálních znaků pomocí `\char` v sekci 2.3 a za pomoci dodatku C vysvětlete, proč to funguje.

3.7 Mírná odlišnost od ASCII u některých fontů

Knuth se (bohužel nepříliš koncepčně) odklonil v několika málo místech ve svých fontech Computer Modern od kódování ASCII. Úplný výčet těchto odlišností ukazuje následující tabulka.

kód znaku	znak dle ASCII	Computer Modern
60	<	ı
62	>	ı̇
92	\	“
123	{	—
124		—
125	}	”
36	\$	\$, v kurzívě £

Knuthovy fonty Computer Modern jsou implicitní v plain \TeX u a jejich konzervativní rozšíření $\mathcal{C}\mathcal{S}$ fonty zachovávající uvedené odlišnosti jsou implicitní v $\mathcal{C}\mathcal{S}$ plainu. Pokud zavědete jiné novější fonty, je pravděpodobné, že budou v dolní části tabulky zcela respektovat viditelné ASCII znaky a popsany problém odpadá.

Při použití Computer Modern fontů nebo $\mathcal{C}\mathcal{S}$ fontů je nutné výše uvedené ASCII znaky psát jako součást matematického vzorečku nebo strojopisem, do kterého se přepíná pomocí makra `\tt`. Strojopis je totiž v této rodině fontů jediný font, který přesně respektuje kódování ASCII. Znak `\`, `{`, `}`, `$` jsou navíc v \TeX u speciální, takže při jejich tisku nestačí přejít do `\tt`, ale je třeba ještě použít trik s `\char` (viz sekci 2.3).

• **Poznámka** • V době vzniku \TeX u bylo možno do jednoho fontu umístit jen 128 znaků, takže se šetřilo. To je asi důvod, proč Knuth vystrnadil ze svých fontů některé ASCII sloty. Znak `<` a `>` se používají v matematické sazbě jako „je menší“ a „je větší“. V matematické sazbě (mezi dolary) ty znaky fungují správně, protože v tom případě \TeX loví znaky z matematických fontů podle speciálního algoritmu. Třeba `$2>1$` vytvoří $2 > 1$. Stejně tak znak `|` funguje v matematické sazbě správně. Znak `{`, `}` a `\` byly dříve rovněž určeny výhradně pro matematickou sazbu (množinové závorky a množinový rozdíl). Je možné je mezi dolary vytisknout pomocí `\{`, `\}`, `\setminus` resp. `\backslash`. Proč se konečně Knuth rozhodl do kurzívy místo dolaru vložit libru se lze jen dohadovat: že by tím projevil svůj smysl pro humor? Dobrá zpráva je, že všechny ostatní novější fonty mají na pozici ASCII pro dolar skutečně jen \$.

Kapitola 4

Hladká sazba

Hladká sazba, tj. pouze text, se člení na odstavce. Odstavce jsou od sebe odděleny ve zdrojovém souboru prázdným řádkem. V místě každého prázdného řádku \TeX automaticky vloží příkaz `\par`, který má za úkol ukončit formátování právě dočteného odstavce.

Text každého odstavce si \TeX interně uloží do jednoho řádku a v okamžiku vykonávání příkazu `\par` se pokusí tento jeden interní řádek rozlomit do řádků o délce stanovené registrem `\hsize`. Tyto řádky vloží pod sebe do výsledné sazby.

● **Cvičení 9** ● Zkuste vytvořit dostatečně dlouhý odstavec ukončený prázdným řádkem. Pak napište `\hsize=.5\hsize` a pod tímto nastavením vytvořte text dalšího odstavce. Celý dokument ukončete makrem `\bye`. Druhý odstavec bude mít poloviční šířku než první.

4.1 Základní parametry pro formátování odstavce

Následující přehled shrnuje základní registry, které ovlivní vzhled výsledného odstavce.

<code>\hsize</code>	... Šířka odstavce.
<code>\parindent</code>	... Odstavcová zarážka, tj. velikost mezery před prvním slovem v odstavci.
<code>\baselineskip</code>	... Vzdálenost mezi účarími sousedních řádků.
<code>\parskip</code>	... Dodatečná vertikální mezera mezi odstavci.
<code>\leftskip</code>	... Mezera vkládaná vlevo do každého řádku.
<code>\rightskip</code>	... Mezera vkládaná vpravo do každého řádku.
<code>\hangindent,</code> <code>\hangafter</code>	... Registry umožňující vytvořit obdélníkový výsek v odstavci.
<code>\parshape</code>	... Příkaz umožňující vytvořit libovolný tvar odstavce.

Odstavcová zarážka `\parindent` je v plain \TeX u implicitně nastavena na 20 pt¹⁾. Ovlivní všechny odstavce v dokumentu. Pokud v prvním odstavci po nadpisu kapitoly odrážka chybí, je to způsobeno makrem na tvorbu nadpisu. Chcete-li výjimečně zahájit odstavec bez odstavcové zarážky, použijte na jeho začátku příkaz `\noindent`. Naopak příkaz `\indent` zahájí odstavec s odstavcovou zarážkou. Nicméně odstavcová zarážka se do začátku odstavce vkládá automaticky (jakmile není zahájen příkazem `\noindent`), takže `\indent` není nutné explicitně používat.

Jednotlivé řádky odstavce \TeX vloží pod sebe. Jejich vzdálenost (tj. vzdálenost jednoho účarí řádku od druhého) je určena registrem `\baselineskip`, který je implicitně nastaven v plain \TeX u na 12 pt. Velikost písma je implicitně nastavena na 10 pt, takže se do řádků vejde. Pokud by byl text v řádcích tak vysoký, že se do řádků nevejde, budou se řádky „opírat“ jeden o druhý a rozhodí to řádkování. Toto chování je možné změnit pomocí registrů `\lineskip` a `\lineskiplimit` (viz sekci 10.1).

Registr `\parskip` umožňuje nastavit vertikální mezera mezi odstavci. Implicitně se vkládá mezera nulové velikosti s nepatrnou možností se roztáhnout (viz sekci 10.1).

Registry `\leftskip` a `\rightskip`, které vkládají stejné mezery do každého řádku odstavce, se využívají při nastavení formátování odstavce na praporek. Stačí příslušnou mezera nastavit na 0 pt plus libovolné roztažení:

¹⁾ Jednotky používané v \TeX u (pt, mm, cm, in, pc, bp, dd, cc, sp, em, ex) jsou shrnuty v dodatku C.

```
\leftskip = 0pt plus1fill % levý praporek, text zarovnán vpravo
\rightskip = 0pt plus1fill % pravý praporek, text zarovnán vlevo
```

Nastavíte-li takto současně `\leftskip` i `\rightskip`, jsou řádky v odstavci centrovány. Často se doporučuje při sazbě na pravý praporek nastavit roztažitelnost s omezenou tolerancí, abychom přinutili \TeX rozdělovat slova, tedy: `\rightskip = 0pt plus2em`. Viz například makro `\raggedright` z `plain \TeX` u.

O registrech `\hangindent`, `\hangafter` a příkazu `\parshape`, pomocí kterých lze nastavit tvar odstavce odlišný od obdélníkového bloku, pojednává TBN v odst. 6.5.

4.2 Automatické dělení slov

\TeX se při lámání odstavce do řádků pokusí nejprve vyjít jen s mezerami mezi slovy. Pokud by tak vznikly řádky příliš roztažené nebo stažené, pokusí se navíc dělit slova. Je ovšem potřeba správně nastavit vzory pro dělení slov v souladu s jazykem, který je použit. Chcete-li naopak zcela zakázat dělení slov, napište `\hyphenpenalty=10000`.

`Plain \TeX` nastavuje výhradně jen anglické vzory dělení slov. `CSplain` také implicitně nastavuje anglické vzory dělení slov, ale následujícími makry je možné přepnout do vzorů jiných jazyků¹):

```
\chyph % vzory dělení slov pro češtinu
\shyph % vzory dělení slov pro slovenštinu
\ehyph % návrat k implicitním vzorům dělení angličtiny
```

Pokud tedy píšete český dokument, doporučuji hned jako první řádek dokumentu napsat:

```
\chyph % use csplain
```

Jestliže se někdo pokusí tento dokument zpracovat jiným formátem než `CSplain`, obdrží chybové hlášení „undefined control sequence“ a navíc se v tom hlášení přepíše uvedený komentář „use csplain“, takže uživatel bude vědět, co má s dokumentem dělat.

Uvedená makra `\chyph`, `\shyph` a `\ehyph` je možné použít i opakovaně před některými částmi dokumentu. \TeX je vybaven interním mechanismem, který dovolí přepínat různé jazyky i uvnitř jediného odstavce.

Makra `\chyph` a `\shyph` navíc nastavují mezerování za tečkami za větou tak, aby tyto mezery byly stejné jako mezislovní mezery. Tomu se říká `\frenchspacing`. Implicitně při `\ehyph` inklinují mezery za tečkou k ochotnějšímu roztahování než mezislovní mezery. Tomuto mezerování se říká `\nonfrenchspacing` a odpovídá to americké typografické tradici, ve které se větší mezerou znázorňuje konec věty.

Nejste-li spokojeni s rozdělením slova, je možné zařadit slovo do slovníku výjimek pomocí příkazu `\hyphenation` v záhlaví dokumentu nebo vyznačit výjimku přímo v textu dokumentu příkazem `\-` vloženým dovnitř slova. Příklad:

```
\hyphenation{roz-dě-lit vý-jim-ka}
... text obsahuje kom\pli\ko\va\né slovo, které se špatně rozdělilo.
```

Příkaz `\-` uvnitř slova vyznačuje potenciální místo dělení slova. Takže při nerozdělení slova se v tom místě nevytiskne nic viditelného, zatímco při rozděleném slově se tam objeví samozřejmě spojovník.

Deklarace `\hyphenation` obsahuje slova rozdělená spojovníkem a oddělená od sebe mezerami. Tato deklarace se vztahuje k jazyku, který je v době čtení deklarace inicializován (pomocí přepínačů `\chyph`, `\shyph`, `\ehyph`).

¹) Dodatek G popisuje možnosti `CSplainu` použít desítky dalších jazyků.

4.3 Další parametry pro řádkový zlom

Mezery mezi slovy sice inklinují k roztažení, ale ne příliš ochotně. Takže se někdy stane, že \TeX mezery neroztáhne, ale pokusí se je stlačit. To se třeba také nepovede a pak \TeX vypíše hlášení `\overfull \hbox` a v dokumentu ponechá řádek vyčnívající ze sazby označený vpravo černým obdélníčkem. To je stav, který jistě není možné takto nechat. Není-li možné přeformulovat odstavec (protože nejste autoři textu), můžete přistoupit k většímu roztažení mezer. Míru ochoty roztažení těchto mezer nad obvyklou implicitní hodnotu je možné nastavit v registru `\emergencystretch`, například `\emergencystretch=2em`¹⁾. Typicky zlobí jen některý odstavec. Pak je možné toto nadstandardní povolení k roztažení poskytnout jen tomuto odstavci. To uděláte tak, že na konec odstavce (před prázdný řádek) napíšete `{\emergencystretch=2em\par}`.

Šířku černého obdélníčku, který se připojuje vpravo od přetečených boxů, je možné nastavit pomocí `\overfullrule`. Implicitně je `\overfullrule=5pt`. \LaTeX například nastavuje `\overfullrule=0pt`, aby nebyl uživatel těmi černými obdélníčky iritován.

Na terminálu a v souboru `.log` se objevují zprávy nejen o přetečených, ale také o podtečených boxech, tj. o takových řádcích, ve kterých jsou mezery roztaženy nad esteticky přípustnou mez. V registru `\hbadness` je možné nastavit úroveň chybovosti roztažení, pod kterou se varovné hlásky nevypisují. Plain \TeX nastavuje `\hbadness=1000`, takže \TeX na terminálu upozorňuje pouze na podtečené boxy s chybovostí větší než 1000²⁾. Nastavení tohoto registru ovlivní jen „ukecanost“ v `.log` souboru a na terminálu, vlastní sazbu to neovlivní.

Podrobnější popis vlastností algoritmu zlomu odstavce na řádky, vyhodnocení chybovosti a popis dalších registrů na řízení těchto věcí jsou popsány v TBN v odstavci 6.4.

4.4 Vertikální, odstavcový a další módy

Na začátku zpracování dokumentu je \TeX v *hlavním vertikálním módu*, ve kterém plní interní sloupec sazby. Ten je na začátku zpracování pochopitelně prázdný. Ve vertikálním módu \TeX ignoruje všechny mezery a všechny příkazy `\par`. Jakmile se v tomto módu na vstupu objeví znak, který se má vytisknout, považuje ho \TeX za zahajovací znak odstavce a přechází do *odstavcového módu*. Před zahajovací znak vloží horizontální mezeru velikosti `\parindent`. Do odstavcového módu může \TeX přejít taky explicitně pomocí `\indent` nebo `\noindent` a dále je možné jej přinutit přejít do tohoto módu makrem `\leavevmode` nebo příkazem `\hskip` (vkládá horizontální mezeru).

V odstavcovém módu \TeX ukládá jednotlivé znaky sazby do interního řádku a dělá to tak dlouho, dokud nenarazí na `\par` (neboli na prázdný řádek). Tehdy \TeX rozlomí interní řádek do řádků odstavce o šířce `\hsize`. Pak se vrací do nadřazeného vertikálního módu a předá mu také vyhotovené řádky odstavce. Vertikální mód zařadí tyto řádky do svého interního sloupce sazby.

Odstavcový mód je možné ukončit také bez explicitního `\par`: příkazem `\vskip` (vkládá vertikální mezeru), příkazem `\end` (konec zpracování dokumentu), dále makrem `\bye` nebo na konci `\vboxu` (o `\vboxech` si povíme v kapitole 9). Ve všech těchto případech \TeX nejprve ukončí rozpracovaný odstavec, jakoby tam byl příkaz `\par`.

Přepínání módů souvisí se základní dovedností každého programu na vytváření sazby: realizace řádkového a stránkového zlomu. Úkolem hlavního vertikálního módu je postupně

¹⁾ Tj. celkové roztažení mezer na jednom řádku je povoleno zvětšit o 2 em.

²⁾ Chybovost 1000 odpovídá zhruba dvojnásobnému roztažení mezer přes povolenou mez. Hodnota chybovosti 100 odpovídá přesně roztažení mezer na povolenou mez a hodnota 10 000 odpovídá libovolně velkému roztažení.

plnění interního sloupce sazby. \TeX z tohoto sloupce po dostatečném naplnění postupně odlamuje díly o výšce `\vsize` a takto odlomenou část sazby předává `\output` rutině k doplnění čísla strany a k případnému dalšímu kompletování strany¹⁾. Z vertikálního módu \TeX přechodně a dle pravidel popsaných výše přechází do odstavcového módu, jehož výstupem (rozlomenými řádky odstavce o šířce `\hsize`) pak plní interní sloupec sazby.

● **Cvičení 10** ● Zdůvodněte, proč je jedno, zda mezi odstavci je jeden nebo více prázdných řádků. Odpověď si nejprve rozmyslete a pak ji najdete v poznámce pod čarou.²⁾

\TeX používá ještě *vnitřní vertikální* nebo *vnitřní horizontální mód*. V těchto módech je sloupec sazby nebo řádek plněn do \TeX ového boxu, viz kapitolu 9. Hlavní nebo vnitřní vertikální mód označujeme stručně jako *vertikální mód*, odstavcový nebo vnitřní horizontální mód nazýváme *horizontální mód*.

Dále \TeX přechází mezi dolary `$. . . $` do *vnitřního matematického módu*, ve kterém \TeX sestaví matematický vzoreček a ten vrátí do nadřazeného horizontálního módu (vzoreček uvnitř odstavce). Konečně mezi zdvojenými dolary `$$. . . $$` přechází \TeX do *display matematického módu*, ve kterém také sestaví vzoreček, a ten umístí pod předchozí odstavec na samostatný řádek. Podrobnější informace o matematických módech jsou v kapitole 6. Následuje přehled \TeX ovských módů.

hlavní vertikální mód (probíhá stránkový zlom)	} vertikální mód
vnitřní vertikální mód (uvnitř <code>\vbox</code> , <code>\vtop</code> , viz kapitolu 9)	
odstavcový mód (probíhá sestavení odstavce)	
vnitřní horizontální mód (uvnitř <code>\hbox</code> , viz kapitolu 9)	} horizontální mód
vnitřní matematický mód (mezi <code>\$. . . \$</code>)	
display mód (mezi <code>\$\$. . . \$\$</code>)	} matematický mód, viz kapitolu 6

4.5 Umístění sazby na stránce

Šířka sazby je dána šířkou odstavců, tedy registrem `\hsize`. Výška sazby na stránce je dána hodnotou registru `\vsize`.

Umístění sazby na stránce³⁾ vzhledem ke krajům papíru je řízeno registry `\hoffset` a `\voffset`. Ty určují posun levého horního rohu sazby vzhledem k „počátku sazby“, který je na papíře stanoven 1 in od levého kraje a 1 in od horního kraje papíru. K tomuto počátku \TeX připočte `\hoffset` (směrem doprava) a `\voffset` (směrem dolů) a tam umístí levý horní roh sazby. Je-li `\hoffset` záporný, posune sazbu od počátku sazby doleva a je-li `\voffset` záporný, posune sazbu nahoru. Implicitně je v `plainTeXu` `\hoffset=0pt` i `\voffset=0pt`, takže levý a horní okraj má na papíře velikost 1 in. Dále jsou v `plainTeXu` implicitně zvoleny registry `\hsize` a `\vsize` tak, že i pravý a dolní okraj má velikost 1 in na papíře formátu US letter. `CSplain` na rozdíl od toho nastavuje výchozí hodnoty `\hsize` a `\vsize` tak, že je pravý a dolní okraj veliký 1 in na stránce formátu A4.

Povšimněme si, že \TeX nepracuje s žádným registrem, který by obsahoval skutečnou velikost papíru, na který bude dokument vytisknuto. Sazbu umístí jen vzhledem k levému hornímu rohu skutečného papíru a o rozměr papíru se nestará.⁴⁾

V sekci 7.2 je popsáno makro `\margins`, které umožní pohodlně nastavit okraje sazby pro různé formáty papíru.

¹⁾ O `\output` rutině je více řečeno v sekci 10.3 a v TBN v sekci 6.8.

²⁾ První prázdný řádek (první `\par`) ukončí zpracovávání odstavce a vrátí se do vertikálního módu. Další následující prázdné řádky (další `\par`) jsou ve vertikálním módu ignorovány.

³⁾ Do sazby na stránce se v tomto případě nezapočítává záhlaví a zápatí stránky (např. číslo strany). Tyto údaje jsou umístěny do okraje stránky.

⁴⁾ Pdf \TeX nastavuje i velikost média, tedy papíru. Viz sekci 11.1.

Kapitola 5

Fonty

PlainT_EX má načtenou rodinu fontů Computer Modern a C_Splain má načtenou víceméně stejnou rodinu fontů zvanou C_Sfonty. Ta rozšiřuje tabulku znaků Computer Modern fontů o znaky české a slovenské abecedy. Touto rodinou je vytištěn tento text, takže čtenář má představu, jak fonty v této rodině vypadají. K dispozici jsou následující makra, která přepínají mezi jednotlivými variantami použité rodiny fontů:

<code>\rm</code>	Regular	(základní řez, Roman, antikva)
<code>\bf</code>	Bold	(tučný)
<code>\it</code>	<i>Italic</i>	(kurzíva)
<code>\bi</code>	<i>BoldItalic</i>	(tučná kurzíva) % jen v C _S plainu od Dec. 2012
<code>\tt</code>	Typewriter	(strojopis)

Nastavení aktuálního textového fontu fontovým přepínačem nebo makrem trvá až do nastavení dalšího fontu nebo do ukončení *skupiny*. Začneme tedy nejprve tématem, které s fonty souvisí jen okrajově:

5.1 Skupiny v T_EXu

Uvnitř skupiny jsou vesměs všechna nastavení T_EXu lokální. To znamená, že po ukončení skupiny se tato nastavení vracejí do původního stavu, jaký byl před zahájením skupiny. Skupina se v T_EXu zahajuje symbolem { a ukončuje se symbolem }. Skupiny mohou být do sebe libovolněkrát vnořeny.

Mezi nastavení, která jsou lokální ve skupině, patří:

- Nastavení aktuálního fontu fontovým přepínačem nebo makrem.
- Nastavení hodnot vesměs všech registrů.
- Definice maker.
- Veškeré další deklarace řídicích sekvencí.

Funkci skupin si ukážeme na makrech pro přepínání fontů:

Na začátku je nastaven font Regular, {ve skupině pokračuje Regular `\bf` a nyní probíhá tisk fontem Bold, {`\it` nyní *Italic*,} tady se vracíme k fontu Bold, {`\tt` strojopis / {`\it` kurzíva}} a zde je návrat do stavu před vstupem do první skupiny, tedy k fontu Regular.

Tento text vytvoří následující výstup:

Na začátku je nastaven font Regular, ve skupině pokračuje Regular **a nyní probíhá tisk fontem Bold**, *nyní Italic*, tady se vracíme k fontu Bold, strojopis / kurzíva a zde je návrat do stavu před vstupem do první skupiny, tedy k fontu Regular.

K čemu použít skupiny? Do skupin je typicky vkládána speciální sazba, která vyžaduje odlišný font, odlišnou velikost fontu nebo odlišné nastavení parametrů sazby (citace, poznámky pod čarou atd.). Někdy editor sborníku schová do skupiny taky každý článek ze sborníku zvlášť, protože autor článku může ve svém textu vytvořit své speciální nastavení a svá speciální makra, což by mohlo ovlivnit sazbu i ostatních článků.

Začátečníka možná překvapí, že lokální nastavení parametrů odstavce (řádkování, šířka odstavce) je sice možné, ale skupinu je nutné ukončit až po příkazu `\par`, který odstavec vytvoří, tedy až po prázdném řádku. Takže:

```
{\hsize=5cm Tady má být odstavec, který je široký jen 5~centimetrů.}

Ono to nefunguje. % \par je totiž až v místě, kde má \hsize původní rozměr.
Ale:

{\hsize=5cm Tady má být odstavec, který je široký jen 5~centimetrů.

}
Toto funguje. Překvapivě funguje i toto:

Tady má být odstavec, který je široký 5~centimetrů.
{\hsize=5cm\par}

A tady už je odstavec normální šířky.
\bye
```

Vysvětlíme si, proč funguje případ `{\hsize=5cm\par}`. V době sestavování odstavce \TeX pouze plní svůj interní řádek. Teprve v okamžiku příkazu `\par` se podívá na aktuální hodnotu registru `\hsize` a podle ní zalomí odstavec. Příkaz `\par` přitom nastane uvnitř skupiny, ve které je `\hsize=5cm`. Po ukončení této skupiny se `\hsize` vrací k původní hodnotě. Prázdný řádek, který případně následuje, sice vytvoří další `\par`, ale to neprovede nic, protože nyní je \TeX ve vertikálním módu.

Závorky `{ }` mají v \TeX u kromě vymezení skupin i další funkce: vymezují hranice parametrů maker a vymezují těla definic maker. Význam závorek vyplývá z kontextu použití. Bez ohledu na význam těchto závorek, \TeX důsledně hlídá jejich *balancování*, tj. nikdy nesmí v jednom syntaktickém celku být více otevíracích než zavíracích závorek nebo obráceně.

Plain \TeX deklaruje řídicí sekvence `\bgroup` a `\egroup`, které fungují stejně jako závorky `{ }`, takže otevírají a zavírají skupinu. Dále \TeX disponuje příkazy `\begingroup` a `\endgroup`, které dělají taky totéž. Tyto řídicí sekvence se používají výhradně na zahájení a ukončení skupiny. Nemají (na rozdíl od závorek samotných) žádný jiný význam a nepodléhají kontrole na balancování závorek.

5.2 Příkaz `\font`

Známe-li název fontu (přesněji název souboru s příponou `.tfm`, tj. \TeX font metric) a je-li tento soubor instalován v \TeX ové distribuci, pak takový font můžeme zavést do dokumentu příkazem `\font` takto:

```
\font \přepínač = <soubor>
nebo
\font \přepínač = <soubor> at<velikost>
nebo
\font \přepínač = <soubor> scaled<faktor>
```

Například v systému je soubor `pzcmi8z.tfm`, o kterém (dejme tomu) víme, že implementuje font *Zapf-Chancery* a tento font potřebujeme dostat do sazby. Pak stačí psát:

```
\font \zapfchan = pzcmi8z      % soubor píšeme bez přípony .tfm
...
Tady je normální text a {\zapfchan tady je text ve fontu Zapf-Chancery}.
```

Příkaz `\font` zavede do paměti \TeX u specifikovaný font a deklaruje `\přepínač`, který je možné později v dokumentu použít jako *přepínač fontu*. Pokud není specifikována velikost

fontu, je font zaveden ve své „přirozené velikosti“, což bývá skoro vždy 10 pt, není-li řečeno jinak. Rovnítko (jak je v T_EXu zvykem) je v deklaraci `\font` nepovinné a nepovinné jsou i mezery kolem něj.

● **Poznámka** ● Příkaz `\font` deklaruje *fontový přepínač*, zatímco `\rm`, `\bf`, `\it`, `\bi`, `\tt` jsou makra, která obsahují po řadě fontové přepínače `\tenrm`, `\tenbf`, `\tenit`, `\tenbi` a `\tentt`. Kromě toho tato makra dělají jistou práci v matematické sazbě, což bude vysvětleno v kapitole 6. Je dobré rozlišovat mezi fontovými přepínači a těmito makry, i když z uživatelského pohledu se chovají stejně.

● **Cvičení 11** ● Podívejte se do souboru `plain.tex`, jak jsou deklarovány přepínače `\tenrm`, `\tenbf`, `\tenit` a `\tentt` a jak jsou definována makra `\rm`, `\bf`, `\it`, `\tt`.

Chcete-li zavést font v jiné než přirozené velikosti, stačí použít v deklaraci `\font` klíčové slovo `at` následované metrickým údajem. Například `\font\zpch=pzcmi8z at13.5pt` připraví font *Zapf-Chancery* ve velikosti 13,5 bodů. Konečně deklarace s klíčovým slovem `scaled` umožní zavést font ve stanoveném násobku vůči jeho přirozené velikosti. Údaj za slovem `scaled` je numerický, přitom číslo 1000 znamená faktor jedna ku jedné. Takže:

```
\font \zapfhalf pzcmi8z scaled500 % font poloviční velikosti
\font \zapfdouble pzcmi8z scaled2000 % dvojnásobně velký font
\font \zapfscaled pzcmi8z scaled1400 % zvětšený 1,4 krát
```

Nadpisy různých důležitostí mají v dokumentu obvykle font zvětšený 1,2 krát a dále 1,2 krát velikost už zvětšeného fontu, tj. 1,44 krát velikost základního fontu atd. PlainT_EX nabízí pro tyto konstanty makro `\magstep`, které se někdy hodí použít za slovem `scaled` v příkazu `\font` takto:

```
\font\fa=csbx10 scaled\magstep1 % jako scaled1200, tj. 1,2 krát větší
\font\fb=csbx10 scaled\magstep2 % jako scaled1440, tj. 1,2x1,2 krát větší
\font\fc=csbx10 scaled\magstep3 % jako scaled1728, tj. 1,2^3 krát větší
\font\fd=csbx10 scaled\magstep4 % jako scaled2074, tj. 1,2^4 krát větší
\font\fh=csbx10 scaled\magstephalf % jako scaled1095, sqrt(1,2) krát větší
```

S fonty v T_EXu, zejména s jejich původními fonty rodiny Computer Modern a s fonty od nich odvozenými, je poněkud komplikovanější pořízení. Důvody jsou zejména dva:

- Stejně písmo může mít pro různé velikosti různé fontové soubory (má tedy speciální kresbu znaků pro různé velikosti).
- Soubory jsou pojmenovány nečitelnými zkratkami, ve kterých se obtížně orientuje.

Tato problematika je blíže rozvedena v dodatku F.

5.3 Fontové soubory, výběr rodiny fontů

Chcete-li použít jinou rodinu fontů než implicitní Computer Modern (resp. C_Sfonty), je možné příkazem `\input` načíst některý z následujících *fontových souborů*. Jsou to soubory z balíčku C_Splainu, které obsahují sadu příkazů `\font` k načtení specifikované rodiny fontů ve čtyřech základních variantách (Regular, Bold, Italic a BoldItalic).

```
ctimes.tex % Times Roman
cavantga.tex % Avantgarde Book
cbookman.tex % Bookman
chelvet.tex % Helvetica
cncent.tex % New Century
cpalatin.tex % Palatino
```


Po zavedení kteréhokoli z těchto souborů budou přepínače `\tenrm`, `\tenbf`, `\tenit` a `\tenbi` přepínat do variant příslušné rodiny fontů. Navíc přepínač `\tentt` přepne do fontu Courier. Protože tyto přepínače jsou součástí maker `\rm`, `\bf`, `\it`, `\bi` a `\tt`, lze jednoduše říci, že tato makra začnou vybírat varianty zvolené rodiny fontů. Implicitně je také inicializována varianta `\rm` dané rodiny. Takže třeba:

```
\chyph % use csplain, Czech
\input cbookman
Tady píšu ve fontu Bookman, \bf nyní píšu ve fontu Bookman Bold
\it a toto je vytištěno fontem Bookman Italic.
\bye
```

Balíček `CSplain` obsahuje také fontový soubor `lfonts.tex`, který zavede rodinu fontů Latin Modern. Ta vizuálně vypadá jako Computer Modern, ale fonty jsou navíc připraveny v rozličných formátech a kódováních, viz dodatky B.2 a F.3.

Další fontové soubory z balíčku `CSplain` mají typicky předponu `cs-`. Původním fontovým souborům s předponou `c` (které existují v `CSplain` už více než deset let) jsem ponechal jejich tradiční název, ale nové fontové soubory zakládám s předponou `cs-`. Takže třeba:

```
\input cs-character % Fonty rodiny Charter
\input cs-polta     % Antykwa Półtawskiego
\input cs-antt      % Antykwa Toruńska
```

Příkaz na příkazovém řádku `tex cs-all` vypíše všechny dostupné fontové soubory. Mají-li uživatelé plain \TeX u zájem, vytvoří si jistě další takové soubory metodou analogie.

Mezi fontovými soubory najdete takové, které zavádějí \TeX -gyre fonty, což jsou fonty implementující řezu původně implicitní sady 35 PostScriptových fontů, která má být přítomna v každém PostScriptovém RIPu. Fonty jsou ovšem nazvány jinak, protože jsou zcela nově implementovány a připraveny v rozličných formátech a kódováních.

```
cs-termes.tex      % TeX Gyre Termes, jako Times Roman
cs-adventor.tex    % TeX Gyre Adventor, jako Avantgarde Book
cs-bonum.tex       % TeX Gyre Bonum, jako Bookman
cs-heros.tex       % TeX Gyre Heros, jako Helvetica
cs-pagella.tex     % TeX Gyre Pagella, jako Palatino
cs-schola.tex      % TeX Gyre Schola, jako New Century
cs-cursor.tex      % TeX Gyre Cursor, jako Courier
```

Některé rodiny fontů nastavují více variant než základní čtyři (například Helvetica). Stojí za to si po zavedení zvoleného fontového souboru přečíst, co se píše na terminálu a v `.log` souboru.

Návod na instalaci dalších rodin fontů je uveden v dodatku F.2.

5.4 Zvětšování a zmenšování fontů v `CSplain`

Fonty načtené během generování formátu `CSplain` i fonty zavedené fontovými soubory mají implicitní velikost 10pt. To jistě není dostačující. Pro nadpisy se používají fonty větší, pro poznámky pod čarou menší. Velikost fontů pro běžný text může být taky jiná než implicitních 10 bodů. `CSplain` proto nabízí makra `\resizefont`, `\resizeall`, `\regfont` a `\letfont`, která řeší uvedené požadavky.

Makro `\resizefont` přepínač změny velikost fontu `\přepínač` podle obsahu makra `\sizspec`. Po použití `\resizefont` `\přepínač` bude `\přepínač` přepínat do stejného fontu jako předtím, ale ve velikosti `\sizspec`. Tato změna velikosti je lokální ve skupině. Makro `\sizspec` může obsahovat buď `at<velikost>` nebo `scaled<faktor>`. Příklad:

```
\def\sizespec{at12pt}\resizefont\tenrm
Nyní \tenrm přepíná do CS Roman at12pt.
\bye
```

Makro `\resizeall` aplikuje `\resizefont` na přepínače `\tenrm`, `\tenbf`, `\tenit`, `\tenbi` a `\tentt` a dále na všechny přepínače registrované pomocí `\regfont`. Příklad:

```
\font\zapfchan=pzcmi8z \regfont\zapfchan
\def\sizespec{at13.5pt}\resizeall \tenrm
Nyní je veškerá sazba ve velikosti 13,5~pt včetně {\it kurzívy},
{\bf tučné varianty} i fontu {\zapfchan Zapf-Chancery}.
Názvy tenrm, tenbf, tenit atd. je nyní potřeba považovat jen jako
\uv{historický odkaz plain\TeX{}u}. Nemají nic společného s deseti body.
```

```
\def\sizespec{at8pt}\resizeall \tenrm
Nyní je veškerá sazba ve velikosti 8~pt.
\bye
```

Je možné nejprve načíst rodinu fontů fontovým souborem a pak dodatečně stanovit velikost sazby:

```
\input cpalatin \def\sizespec{at11pt}\resizeall \tenrm
Sazba bude probíhat v rodině Palatino ve velikosti 11~bodů.
\bye
```

Makro `\letfont` novýpřepínač = \známýpřepínač *<specifikace velikosti>* deklaruje novýpřepínač jako přepínač stejného fontu, do jakého přepíná `\známýpřepínač`, ale velikost tohoto fontu bude odpovídat *<specifikaci velikosti>*. Například:

```
\letfont\titlefont=\tenbf at15pt % nebo:
\letfont\titlefont=\tenbf scaled\magstep4
```

Tím je připraven přepínač fontu `\titlefont`, který bude přepínat do tučné varianty zvolené rodiny fontů ve stanovené velikosti.

Následující příklad připraví font pro nadpisy a makro `\small`, které nastaví fonty všech variant do menší velikosti pro poznámky pod čarou. Nadpisy budou ve fontu Helvetica, běžný text i poznámky pod čarou v Times Roman.

```
\input chelvet
\letfont\titlefont = \tenbf at14.4pt
% \titlefont přepíná do fontu pro nadpisy Helvetica Bold at14,4pt.
\input ctimes
\def\sizespec{at11pt}\resizeall \tenrm
% Běžný text bude sázen v rodině fontu Times Roman at11pt.
\def\small{\def\sizespec{at9pt}\resizeall \tenrm}
% Makro \small přepne (lokálně) celou rodinu Times Roman do 9~pt.
```

Tento způsob zvětšování (zmenšování) fontů se týká jen textových fontů. Jak zvětšit (zmenšit) matematické fonty je popsáno v sekci 6.2 a v sekci 7.1.

Některé fonty (typicky například z rodiny Computer Modern) mají různé kresby pro různé designované velikosti fontu. Viz ukázkou na straně 128 dole. Výše uvedený postup zvětšování a zmenšování fontů provádí implicitně jen geometrickou transformaci. Pokud ale zavedete příkazem `\input` soubor `ams-math.tex`, je připravena možnost fonty Computer Modern zvětšovat a zmenšovat s ohledem na jejich designované velikosti. Není-li definováno makro (nebo registr) `\dgsiz`, pak se stále provádí zvětšování a zmenšování lineárně. Je-li `\dgsiz` definováno, pak se při požadavku na zvětšení (zmenšení) vybere vhodný font s designovanou velikostí nejbližší danému `\dgsiz`. Příklad:

```

\input ams-math.tex
\letfont\velky = \tenrm at18pt % provede se geometrické zvětšení fontu
                                % navrženého pro 10pt, tedy csr10 at18pt
\def\dgsize{18pt} \letfont\velky = \tenrm at18pt
                                % v tomto druhém případě se použije font csr17 vhodný
                                % pro 17pt zvětšený na 18pt (nejbližší designovaná velikost)
\def\dgsize{18pt} \letfont\maly = \tenrm at5pt
                                % toto je odstrašující příklad: zmenší se font vhodný
                                % pro velké velikosti, tedy csr17 at5pt. Správně má být:
\def\dgsize{5pt} \letfont\maly = \tenrm at5pt % použije se font csr5 at5pt

```

Uživatelé OPmac se o takové technické detaily nemusejí starat. Makra `\typosize` a `\typoscale` vyberou vhodnou designovanou velikost za ně. Viz sekci 7.1. Naopak, uživatel, který chce mít věci pod kontrolou a chce si nastavit zvětšování podle designované velikosti i pro jiné rodiny fontů než jen Computer Modern, najde inspiraci v souboru maker `ams-math.tex`.

5.5 Italická korekce

Znaky z jednotlivých fontů jsou kladeny těsně za sebou na úrovni účaří. Pokud se přejde ze skloněného fontu na neskloněný, například `{\it děd}ku` (*děd*ku), může dojít k překrytí znaků. Proto \TeX nabízí příkaz `\V`, který vloží mezeru, která je rovna velikosti šikmého přesahu vrchní části předchozího znaku. Takže bychom měli správně psát `{\it děd\}/}ku` (*děd*ku). Pokud na konci sazby šikmým fontem následuje tečka nebo čárka, je vhodnější před ní italickou korekci nedávat.

OPmac definuje makro `\em`, které funguje jako `\it`, ale navíc se automaticky postará o vložení italické korekce `\/`, takže se o to uživatel starat nemusí. Makro `\em` navíc uvnitř kurzívy (`\it`) způsobí naopak přechod do antikvy (`\rm`), uvnitř tučné antikvy (`\bf`) přejde do tučné kurzívy (`\bi`) a konečně uvnitř tučné kurzívy (`\bi`) přejde do tučné antikvy (`\bf`). Obecně se snaží zdůraznit text v kontextu okolního textu. Je to zkratka za „emphasize“. Příklad použití: `{\em zdůrazněný text}`.

Kapitola 6

Matematická sazba

Matematická sazba je drahá a náročná. Proto ji Knuth umístil mezi dolary.

6.1 Příklad a základní vlastnosti

Jestliže napíšeme:

```
Matematický vzoreček může být uvnitř odstavce
$(a+b)^2 = a^2 + 2ab + b^2$ nebo na samostatném řádku:
$$
\sum_{k=0}^{\infty} e^{(\alpha+i\beta_k)} =
e^{\alpha} \sum_{k=0}^{\infty} e^{i\beta_k} =
e^{\alpha} \sum_{k=0}^{\infty} (\cos \beta_k + i \sin \beta_k).
$$
Další text pod vzorečkem pokračuje.
```

na výstupu dostaneme:

Matematický vzoreček může být uvnitř odstavce $(a+b)^2 = a^2 + 2ab + b^2$ nebo na samostatném řádku:

$$\sum_{k=0}^{\infty} e^{(\alpha+i\beta_k)} = e^{\alpha} \sum_{k=0}^{\infty} e^{i\beta_k} = e^{\alpha} \sum_{k=0}^{\infty} (\cos \beta_k + i \sin \beta_k).$$

Další text pod vzorečkem pokračuje.

TeX tvoří vzorečky ve vnitřním matematickém módu ($\$...\$$) nebo v display módu ($$$...$$$). Pravidla pro tvorbu vzorečků v obou módech jsou stejná, liší se jen výsledek svým umístěním (v odstavci nebo na samostatném řádku) a některé znaky jsou v display módu poněkud větší, jak za chvíli uvidíme.

V matematických módech TeX sestavuje sazbu podle výrazně odlišných pravidel od běžného textu. Hlavní odlišnosti jsou tyto:

- Fungují tam konstruktory indexu ($_$) a mocniny ($^$).
- Fungují tam řídicí sekvence pro speciální symboly ($\backslash alpha$, $\backslash sum$, $\backslash infty$).
- Sazba písmen ve vzorečku probíhá implicitně matematickou kurzívou, což je písmo vhodné pro označení proměnných a dalších matematických objektů, například množin. Číslice a symboly (např. $+$, $-$, $=$) jsou naopak tištěny antikvou (vzpřímeným fontem) bez nutnosti mezi fonty přepínat.
- Fontové přepínače pro textový font nemají v matematice žádný vliv. Nicméně makra $\backslash rm$, $\backslash bf$, $\backslash it$ a $\backslash bi$ přepínají textové fonty i matematické abecedy, viz sekci 6.3.
- Mezery ve vstupním textu ve vzorečku se ignorují. TeX mezery při tisku vzorečku vymyslí sám. Takže $\$a+b\$$ dává stejný výsledek jako $\$a + b\$$.
- V matematice fungují natahovací závorky, tvorba zlomků a další speciální sazba, se kterou se za chvíli seznámíme.
- Je zakázáno ukončit v matematickém módu odstavec (prázdným řádkem nebo $\backslash par$).

Z ukázky vidíme, jak se používají konstruktory indexu a mocniny. Je-li indexem nebo mocninou jediný znak, stačí ho napsat přímo za konstruktor (`\beta_k`, `a^2`). Je-li v indexu nebo mocnině celý výraz složený z více znaků, je třeba ho zapouzdřit do svorek, aby \TeX poznal, kde výraz začíná a končí: `e^{\alpha+i\beta_k}`. V tomto příkladě kulaté závorky nestačí, ty \TeX vnímá jako obyčejný znak. Potřebuje mít výraz ve svorkách. Takže `$e^{\alpha+i\beta_k}$` vytvoří $e^{(\alpha + i\beta_k)}$, což asi autor neměl na mysli.

Na příkladu sumy také vidíme, že konstruktor indexu a mocniny se užívá i v případě, kdy to explicitně není index nebo mocnina. Místo indexu se v případě znaku `\sum` vytiskne spodní bižuterie sumy (tedy vzorec, který se zmenší a umístí centrován pod sumu). Místo exponentu bude horní bižuterie.

Na ustálené matematické zkratky (`cos`, `sin`) jsou v \TeX u připravena speciální makra `\cos`, `\sin`. Kdybychom napsali jen `cos\alpha`, byl by to prohrěšek proti dobré matematické sazbě. Dostali bychom totiž $\cos\alpha$, což je psáno kurzívou a není to tedy ustálená zkratka, ale součin proměnných c krát o krát s krát α .

Sazba samostatného vzorečku pomocí `$$...$$` v display módu probíhá tak, že se přeruší aktuálně sestavovaný odstavec a nad a pod vzoreček se vloží vhodné vertikální mezery. Pokud ale na začátku `$$...$$` je \TeX ve vertikálním módu, nejprve zahájí prázdný odstavec a nad vzorečkem proto vytvoří nevhodně velkou vertikální mezeru. Z toho plyne doporučení: *pokud možno nikdy nenechávejte prázdný řádek před `$$...$$`*. Naopak prázdným řádkem po `$$...$$` dáváte najevo, že následující odstavec začne jako obvykle s odsazením. Není-li za `$$...$$` prázdný řádek, pak další text pokračuje bez odsazení a je to míněno jako pokračování přerušného odstavce.

6.2 Soubory maker `ams-math.tex` a `tx-math.tex`

Plain \TeX implicitně nastavuje fonty pro matematiku na základní velikost 10 pt, indexy mají 7 pt, indexy indexů mají 5 pt a jsou použity fonty rodiny Computer Modern. Toto nastavení není snadné změnit. \LaTeX proto nabízí soubory maker `ams-math.tex` a `tx-math.tex`. Oba udělají následující činnost:

- Přeprogramují koncepci zavedení matematických fontů plain \TeX u tak, aby se daly snadno měnit jejich velikosti pomocí `\setmathsizes[⟨t⟩/⟨i⟩/⟨ii⟩]` následované příkazem `\normalmath`. Například: `\setmathsizes[12/8.4/6]\normalmath`.
- Přidávají do sady použitelných matematických znaků další desítky znaků podle rodiny fontů $\mathcal{MST\TeX}$ u

Soubory maker `ams-math.tex` a `tx-math.tex` se liší v tom, že první z nich pracuje s fonty Computer Modern (vhodné v případě, že i textový font je z rodiny fontů Computer Modern, Latin Modern nebo něco podobného). Druhý soubor zavádí pro matematické vzorečky TX fonty, které obsahují matematické znaky vizuálně navazující na font Times Roman. Tyto znaky se dají (na rozdíl od Computer Modern) kombinovat s většinou textových fontů dynamické nebo přechodové antikvy, jinak řečeno s většinou nabízených textových fontů jiných než Computer Modern.

Následující příklad ukazuje nevýhodu implicitního zavedení matematických fontů podle plain \TeX u a dále řešení pomocí `\setmathsizes` z `ams-math.tex`.

```
\def\seq#1{{\tt\char'\#1}}
\def\sizespec{at12pt} \resizeall \tenrm % Velikost textového písma 12 pt
\baselineskip=14pt
```

Rodina textových fontů je ve velikosti 12\,pt, ale matematika zvětšená není: `$a^2+b^2=c^2$`. Abychom ji mohli také zvětšit, musíme použít

`\seq{setmathsizes}` následované `\seq{normalmath}`. Tato makra jsou definována v souboru `{\tt ams-math.tex}` nebo `{\tt tx-math.tex}`.

```
\input ams-math.tex
\setmathsizes[12/8.4/6] % [základní/indexová/index-indexová] velikost v pt
\normalmath
```

Nyní i vzorečky mají správnou velikost: $a^2+b^2=c^2$.

`\bye`

Tento příklad dá následující výsledek:

Rodina textových fontů je ve velikosti 12 pt, ale matematika zvětšená není: $a^2 + b^2 = c^2$. Abychom ji mohli také zvětšit, musíme použít `\setmathsizes` následované `\normalmath`. Tato makra jsou definována v souboru `ams-math.tex` nebo `tx-math.tex`.

Nyní i vzorečky mají správnou velikost: $a^2 + b^2 = c^2$.

Není obvykle nutné soubor maker `ams-math.tex` nebo `tx-math.tex` zavádět explicitně pomocí `\input`, protože `tx-math.tex` je načten automaticky při použití některého z fontových souborů `ctimes.tex`, ..., `cpalatin.tex`, `cs-charter.tex`, ..., `cs-schola.tex`. Dále soubor `ams-math.tex` je zaveden při použití OPmac nebo při použití fontového souboru `lmfonts.tex`. Že je některý ze souborů matematických maker zaveden, poznáme na terminálu a v `.log` souboru podle hlášení:

```
ams-math.tex
FONT: AMS math fonts - \mathchardef's prepared, 12 math families preloaded.
tx-math.tex
FONT: TX math fonts - \mathchardef's prepared, 14 math families preloaded.
```

• **Poznámka** • Při použití příkazů `\typosize` a `\typoscale` z OPmac (viz sekci 7.1) je zvětšení textových i matematických fontů společné a není nutné používat `\setmathsizes`.

6.3 Matematické abecedy

Písmena anglické abecedy A–Z, a–z se vysází ve vzorečku matematickou kurzívou a číslice antikvou. To je implicitní nastavení *matematické abecedy*, kterou můžeme měnit pomocí následujících přepínačů matematické abecedy:

- implicitní: matematická kurzíva $ABC \dots XYZ, abc \dots xyz$, antikva: 0123456789,
- `\it`: textová kurzíva $ABC \dots XYZ, abc \dots xyz$, 0123456789,
- `\rm`: textová antikva $ABC \dots XYZ, abc \dots xyz$, 0123456789,
- `\cal`: jednoduché kaligrafické znaky $ABC \dots \mathcal{XYZ}$ (jen pro velká písmena),
- `\bf`: tučný řez $ABC \dots XYZ, abc \dots xyz$, 0123456789.
- `\oldstyle`: skákavé číslice 0123456789 (jen pro číslice).

Je-li zaveden soubor maker `ams-math.tex` nebo `tx-math.tex`, pak je matematická abeceda `\bf` předdefinována a jsou k dispozici další matematické abecedy:

- `\bf`: tučný řez bez serifů $ABC \dots XYZ, abc \dots xyz$, 0123456789,
- `\bi`: tučný skloněný řez bez serifů $ABC \dots XYZ, abc \dots xyz$, 0123456789,
- `\script`: kroucenější kaligrafické znaky $\mathcal{ABC} \dots \mathcal{XYZ}$ (jen pro velká písmena),

- `\frac`: fraktura $\mathfrak{ABC} \dots \mathfrak{XYZ}$, $\mathrm{abc} \dots \mathrm{xyz}$, 0123456789,
- `\bbchar`: zdvojené znaky $\mathbb{ABC} \dots \mathbb{XYZ}$ (jen pro velká písmena).

O tom, jak přidat další matematickou abecedu, pojednává sekce 6.10.

Uvedené přepínače v matematické sazbě ovlivní právě jen sazbu písmen anglické abecedy a číslic. Ostatní znaky ve vzorečku nejsou přepínačem ovlivněny. Přepínače fungují podobně jako přepínače fontů, tj. mají vliv až do použití dalšího přepínače nebo do konce skupiny, ve které byly použity. Přitom každý vzoreček má samostatnou skupinu (jinak řečeno symboly $\$ \dots \$$ nebo $\$ \$ \dots \$ \$$ vymezují skupinu) a dále svorky uvnitř vzorečku vymezují skupiny včetně těch, které jsou použity za konstruktory indexu a mocniny.

• **Poznámka** • V rodině fontů Computer Modern se matematická kurzíva mírně liší od textové. Jiné rodiny fontů obvykle nemají zvlášť nakreslenou matematickou kurzívu, takže po zavedení souboru maker `tx-math.tex` je v matematické sazbě implicitní textová kurzíva, která se přebírá z použité rodiny pro textový font.

6.4 Zlomky

Zlomek vytvoříme pomocí konstrukce $\{\langle \textit{\texttt{čítatel}} \rangle \over \langle \textit{\texttt{jmenovatel}} \rangle\}$, tj. svorky vymezují hranice zlomku a `\over` se píše v místě zlomkové čáry. Například:

$\$ \$ \{4x^2 + 2 \over x^3 - x^2 + x - 1\} = \{x+1 \over x^2+1\} + \{3 \over x-1\}. \$ \$$

$$\frac{4x^2 + 2}{x^3 - x^2 + x - 1} = \frac{x + 1}{x^2 + 1} + \frac{3}{x - 1}.$$

Hranice zlomku není nutné vymezovat, pokud se shodují s hranicí vzorečku, tedy $\$1 \over 2\$$ vytvoří jednu polovinu.

Zlomek tedy píšeme tak, jak jej v angličtině přečteme. \LaTeX naproti tomu používá na zlomky makro $\text{\texttt{\frac}}{\langle \textit{\texttt{čítatel}} \rangle}{\langle \textit{\texttt{jmenovatel}} \rangle}$. Pokud je uživatel na makro `\frac` zvyklý nebo ho považuje za účelné používat i v \plainTeX u, musí si je nejprve definovat: $\text{\def\frac#1#2{\{#1 \over #2\}}}$.

Podobně jako `\over` funguje příkaz `\atop`, jen s tím rozdílem, že nevytiskne zlomkovou čáru. Pro kombinatorická čísla se ovšem více hodí příkaz `\atopwithdelims` $\langle \textit{\texttt{závorky}} \rangle$, který pracuje jako `\atop`, ale navíc kolem výsledného objektu dá specifikované závorky vhodné velikosti. Tento příkaz je použit třeba v makru `\choose` (viz `plain.tex`). Příklad:

$\$ \$ \{n \choose k\} = \{n! \over k! (n-k)!\}. \$ \$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

6.5 Matematické symboly a znaky

Kromě matematických abeced a číslic se v matematických vzorcích mohou vyskytovat stovky dalších symbolů. Nejprve uvedeme ty, které jsou přímo dosažitelné na klávesnici a pak budeme pokračovat znaky, které lze vytisknout pomocí řídicí sekvence.

základní znaky: \cdot (tečka), $/$ (lomítko), $|$ (svislá čára), $!$ (vykřičník), $?$ (otazník),

binární operátory: $+$ (plus), $-$ (mínus), $*$ (konvoluce),

relace: $=$ (rovná se), $<$ (je menší), $>$ (je větší), $:$ (poměr),

závorky: $()$ (kulaté), $[\]$ (hrnaté), svorky $\{ \}$ je nutno napsat pomocí $\text{\texttt{\{ \}}}$,

interpunkce: $,$ (čárka), $;$ (středník),

speciální: $'$ (derivative), např. $\$f'(x)\$$ vytvoří $f'(x)$.

Podle typu symbolu T_EX vkládá mezi symboly v matematickém vzorečku mezery. Základní znaky jsou výše uvedené znaky z prvního řádku, dále písmena, číslice a plno dalších znaků uvedených v následujících tabulkách. Základní znaky klade T_EX vedle sebe bez mezer. Kolem binárních operátorů udělá menší mezeru ($a - b$), ale výjimečně, kdy tento operátor přechází v unární operátor, mezeru mezi operátor a základní znak nekládá ($-b$). Kolem relací dělá mírně větší mezery než kolem binárních operací. K vnější straně natahovacích závorek a za interpunkci připojuje menší mezeru.

Další znaky jsou dosažitelné pomocí řídicích sekvencí. **Písmena řecké abecedy** patří mezi základní znaky:

α	<code>\alpha</code>	ν	<code>\nu</code>	ω	<code>\omega</code>
β	<code>\beta</code>	ξ	<code>\xi</code>	Γ	<code>\Gamma</code>
γ	<code>\gamma</code>	π	<code>\pi</code>	Δ	<code>\Delta</code>
δ	<code>\delta</code>	ϖ	<code>\varpi</code>	Θ	<code>\Theta</code>
ϵ	<code>\epsilon</code>	ρ	<code>\rho</code>	Λ	<code>\Lambda</code>
ε	<code>\varepsilon</code>	ϱ	<code>\varrho</code>	Ξ	<code>\Xi</code>
ζ	<code>\zeta</code>	σ	<code>\sigma</code>	Π	<code>\Pi</code>
η	<code>\eta</code>	ς	<code>\varsigma</code>	Σ	<code>\Sigma</code>
θ	<code>\theta</code>	τ	<code>\tau</code>	Υ	<code>\Upsilon</code>
ϑ	<code>\vartheta</code>	υ	<code>\upsilon</code>	Φ	<code>\Phi</code>
ι	<code>\iota</code>	ϕ	<code>\phi</code>	Ψ	<code>\Psi</code>
κ	<code>\kappa</code>	φ	<code>\varphi</code>	Ω	<code>\Omega</code>
λ	<code>\lambda</code>	χ	<code>\chi</code>		
μ	<code>\mu</code>	ψ	<code>\psi</code>		

Malé řecké písmeno omikron je stejné, jako malé písmeno o, takže nemá svou řídicí sekvenci. Stejně tak plno velkých řeckých písmen je shodných s písmeny latinské abecedy a nemají tedy speciální řídicí sekvenci.

Je možné si všimnout, že malá řecká písmena jsou psána kurzívou a velká antikvou. To je v matematice obvyklý standard.

Některá malá řecká písmena mají svou variantní podobu, např. `\varrho`. V mnoha případech vypadá tato variantní podoba v sazbě lépe než standardní. Stojí tedy za to na začátek dokumentu napsat:

```
\let\epsilon=\varepsilon \let\theta=\vartheta
\let\rho=\varrho \let\phi=\varphi
```

Po takové deklaraci může autor psát `\epsilon` a vytiskne se ε atd.

Kromě řecké abecedy jsou k dispozici **další základní znaky**:

\aleph	<code>\aleph</code>	\wp	<code>\wp</code>	\top	<code>\Top**</code>
\beth	<code>\beth*</code>	\Re	<code>\Re</code>	\bot	<code>\Bot**</code>
\gimel	<code>\gimel*</code>	\Im	<code>\Im</code>	\square	<code>\square*</code>
\daleth	<code>\daleth*</code>	∂	<code>\partial</code>	\blacksquare	<code>\blacksquare*</code>
\digamma	<code>\digamma*</code>	∞	<code>\infty</code>	\lozenge	<code>\lozenge*</code>
\eth	<code>\eth*</code>	\prime	<code>\prime</code>	\blacklozenge	<code>\blacklozenge*</code>
\varkappa	<code>\varkappa*</code>	\backprime	<code>\backprime*</code>	\parallel	<code>\parallel</code>
\hbar	<code>\hbar</code>	\emptyset	<code>\emptyset</code>	\angle	<code>\angle</code>
\hslash	<code>\hslash*</code>	∇	<code>\nabla</code>	\measuredangle	<code>\measuredangle*</code>
\imath	<code>\imath</code>	\surd	<code>\surd</code>	\sphericalangle	<code>\sphericalangle*</code>
\jmath	<code>\jmath</code>	\top	<code>\top</code>	\triangle	<code>\triangle</code>
ℓ	<code>\ell</code>	\bot	<code>\bot</code>	\blacktriangle	<code>\blacktriangle*</code>

∇	<code>\triangledown*</code>	\textcircled{S}	<code>\circledS*</code>	\diagdown	<code>\diagdown*</code>
\blacktriangledown	<code>\blacktriangledown*</code>	\clubsuit	<code>\clubsuit</code>	\varnothing	<code>\varnothing*</code>
\backslash	<code>\backslash</code>	\diamond	<code>\diamondsuit</code>	\nexists	<code>\nexists*</code>
\forall	<code>\forall</code>	\Diamond	<code>\Diamonddot**</code>	\Finv	<code>\Finv*</code>
\exists	<code>\exists</code>	\heartsuit	<code>\heartsuit</code>	\Game	<code>\Game*</code>
\star	<code>\bigstar*</code>	\spadesuit	<code>\spadesuit</code>	\mho	<code>\mho*</code>
\neg	<code>\neg</code>	\varclubsuit	<code>\varclubsuit**</code>	\Bbbk	<code>\Bbbk*</code>
\complement	<code>\complement*</code>	$\var\diamondsuit$	<code>\vardiamondsuit**</code>	λ	<code>\lambda slash**</code>
\flat	<code>\flat</code>	\varheartsuit	<code>\varheartsuit**</code>	λ	<code>\lambda sbar**</code>
\natural	<code>\natural</code>	\varspadesuit	<code>\varspadesuit**</code>		
\sharp	<code>\sharp</code>	\diagup	<code>\diagup*</code>		

Hvězdička za řídicí sekvencí v této tabulce i v následujících tabulkách označuje symboly, které nejsou dosažitelné v běžném plainTeXu, ale jsou k dispozici po zavedení souboru `ams-math.tex` nebo `tx-math.tex`. Dvěma hvězdičkami jsou označeny symboly přístupné jen po zavedení `tx-math.tex`, tj. nejsou ani v `ams-math.tex`.

Výše uvedené znaky jsou základní, takže jsou kladeny k ostatním základním znakům těsně bez mezer. Chcete-li například ze znaku `\diamondsuit` udělat binární operaci (což se zapisuje jako $a \diamond b$, nikoli $a \diamond b$), je potřeba TeXu říci, že znak používáte v kontextu binární operace, aby mohl přidat správné mezery. K tomu stačí v místě operátoru psát `\mathbin{\diamondsuit}` nebo si pro to můžete definovat nové makro, například `\def\sop{\mathbin{\diamondsuit}}` a psát $\$a\sop b\$$. Pro změnu typu znaků nebo deklarování celého textu jako znak jistého typu slouží příkazy:

<code>\mathord</code>	<code>{\text}</code>	% <code>\text</code> bude typu základní znak
<code>\mathop</code>	<code>{\text}</code>	% <code>\text</code> poslouží jako velký operátor (například suma)
<code>\mathbin</code>	<code>{\text}</code>	% <code>\text</code> bude binární operátor (jako například +, -)
<code>\mathrel</code>	<code>{\text}</code>	% <code>\text</code> bude znakovým pro relaci (jako například =)
<code>\mathopen</code>	<code>{\text}</code>	% <code>\text</code> bude jako otevírací závorka
<code>\mathclose</code>	<code>{\text}</code>	% <code>\text</code> bude jako zavírací závorka
<code>\mathpunct</code>	<code>{\text}</code>	% <code>\text</code> bude jako interpunkce

Po zavedení souboru `tx-math.tex` jsou k dispozici ještě **vzpřímená řecká písmena**:

α	<code>\upalpha**</code>	ι	<code>\upiota**</code>	σ	<code>\upsigma**</code>
β	<code>\upbeta**</code>	κ	<code>\upkappa**</code>	ς	<code>\upvarsigma**</code>
γ	<code>\upgamma**</code>	λ	<code>\uplambda**</code>	τ	<code>\uptau**</code>
δ	<code>\updelta**</code>	μ	<code>\upmu**</code>	υ	<code>\upupsilon**</code>
ϵ	<code>\upepsilon**</code>	ν	<code>\upnu**</code>	ϕ	<code>\upphi**</code>
ε	<code>\upvarepsilon**</code>	ξ	<code>\upxi**</code>	φ	<code>\upvarphi**</code>
ζ	<code>\upzeta**</code>	π	<code>\uppi**</code>	χ	<code>\upchi**</code>
η	<code>\upeta**</code>	ϖ	<code>\upvarpi**</code>	ψ	<code>\uppsi**</code>
θ	<code>\uptheta**</code>	ρ	<code>\uprho**</code>	ω	<code>\upomega**</code>
ϑ	<code>\upvartheta**</code>	ϱ	<code>\upvarrho**</code>		

Následuje tabulka **binárních operátorů**:

\pm	<code>\pm</code>	\star	<code>\star</code>	\div	<code>\div</code>
<i>METAPOST</i>	<code>\mp</code>	\diamond	<code>\diamond</code>	\cap	<code>\cap</code>
\backslash	<code>\setminus</code>	\circ	<code>\circ</code>	\cup	<code>\cup</code>
\cdot	<code>\cdot</code>	\bullet	<code>\bullet</code>	\oplus	<code>\oplus**</code>
\times	<code>\times</code>	\bigcirc	<code>\medcirc**</code>	\uplus	<code>\uplus</code>
$*$	<code>\ast</code>	\bullet	<code>\medbullet**</code>	\sqcap	<code>\sqcap</code>

\sqcup	<code>\sqcup</code>	\odot	<code>\circledcirc</code>	\circledast	<code>\circledast</code>	\curlywedge	<code>\curlywedge</code>
\sqcupplus	<code>\sqcupplus</code>	\otimes	<code>\circledast</code>	\circledast	<code>\circledast</code>	\curlyvee	<code>\curlyvee</code>
\sqcap	<code>\sqcap</code>	\ominus	<code>\circledast</code>	\circledast	<code>\circledast</code>	\leftthreetimes	<code>\leftthreetimes</code>
\triangleleft	<code>\triangleleft</code>	\triangleleft	<code>\triangleleft</code>	\circledast	<code>\triangleleft</code>	\rightthreetimes	<code>\rightthreetimes</code>
\triangleright	<code>\triangleright</code>	\triangleright	<code>\triangleright</code>	\circledast	<code>\triangleright</code>	\dotplus	<code>\dotplus</code>
\bigtriangleup	<code>\bigtriangleup</code>	\uparrow	<code>\bigtriangleup</code>	\circledast	<code>\bigtriangleup</code>	\intercal	<code>\intercal</code>
\bigtriangledown	<code>\bigtriangledown</code>	\downarrow	<code>\bigtriangledown</code>	\circledast	<code>\bigtriangledown</code>	\divideontimes	<code>\divideontimes</code>
\rhd	<code>\rhd</code>	\dagger	<code>\rhd</code>	\dagger	<code>\dagger</code>	\lessdot	<code>\lessdot</code>
\lhd	<code>\lhd</code>	\ddagger	<code>\lhd</code>	\ddagger	<code>\ddagger</code>	\gtrdot	<code>\gtrdot</code>
\wr	<code>\wr</code>	\amalg	<code>\wr</code>	\amalg	<code>\amalg</code>	\ltimes	<code>\ltimes</code>
\bigcirc	<code>\bigcirc</code>	\boxdot	<code>\bigcirc</code>	\boxdot	<code>\boxdot</code>	\rtimes	<code>\rtimes</code>
\unrhd	<code>\unrhd</code>	\boxplus	<code>\unrhd</code>	\boxplus	<code>\boxplus</code>	\smallsetminus	<code>\smallsetminus</code>
\unlhd	<code>\unlhd</code>	\boxtimes	<code>\unlhd</code>	\boxtimes	<code>\boxtimes</code>	\invamp	<code>\invamp</code>
\vee	<code>\vee</code>	\boxminus	<code>\vee</code>	\boxminus	<code>\boxminus</code>	\boxast	<code>\boxast</code>
\wedge	<code>\wedge</code>	\centerdot	<code>\wedge</code>	\centerdot	<code>\centerdot</code>	\boxslash	<code>\boxslash</code>
\oplus	<code>\oplus</code>	\veebar	<code>\oplus</code>	\veebar	<code>\veebar</code>	\boxbar	<code>\boxbar</code>
\ominus	<code>\ominus</code>	\barwedge	<code>\ominus</code>	\barwedge	<code>\barwedge</code>	\boxslash	<code>\boxslash</code>
\otimes	<code>\otimes</code>	\doublebarwedge	<code>\otimes</code>	\doublebarwedge	<code>\doublebarwedge</code>	\Wr	<code>\Wr</code>
\oslash	<code>\oslash</code>	\Cup	<code>\oslash</code>	\Cup	<code>\Cup</code>		
\odot	<code>\odot</code>	\Cap	<code>\odot</code>	\Cap	<code>\Cap</code>		

Tabulka **relací** je velmi rozsáhlá především kvůli $\mathcal{A}\mathcal{M}\mathcal{T}\mathcal{E}\mathcal{X}$ u, tedy znakům definovaných v souboru `ams-math.tex`. Další mraky relací přidávají TX fonty v souboru `tx-math.tex`:

\leq	<code>\leq</code>	\models	<code>\models</code>	\leqslantless	<code>\leqslantless</code>
\geq	<code>\geq</code>	$($	<code>\smile</code>	\leqslantgtr	<code>\leqslantgtr</code>
\equiv	<code>\equiv</code>	$)$	<code>\frown</code>	\curlyeqprec	<code>\curlyeqprec</code>
\prec	<code>\prec</code>	$($	<code>\smallsmile</code>	\curlyeqsucc	<code>\curlyeqsucc</code>
\succ	<code>\succ</code>	$)$	<code>\smallfrown</code>	\preccurlyeq	<code>\preccurlyeq</code>
\sim	<code>\sim</code>	$ $	<code>\mid</code>	\leqq	<code>\leqq</code>
\preceq	<code>\preceq</code>	\parallel	<code>\parallel</code>	\leqslant	<code>\leqslant</code>
\succeq	<code>\succeq</code>	\doteq	<code>\doteq</code>	\lessgtr	<code>\lessgtr</code>
\simeq	<code>\simeq</code>	\perp	<code>\perp</code>	\risingdotseq	<code>\risingdotseq</code>
\ll	<code>\ll</code>	\circlearrowright	<code>\circlearrowright</code>	\fallingdotseq	<code>\fallingdotseq</code>
\gg	<code>\gg</code>	\circlearrowleft	<code>\circlearrowleft</code>	\succcurlyeq	<code>\succcurlyeq</code>
\asymp	<code>\asymp</code>	\Vdash	<code>\Vdash</code>	\geqq	<code>\geqq</code>
\subset	<code>\subset</code>	\Vvdash	<code>\Vvdash</code>	\geqslant	<code>\geqslant</code>
\supset	<code>\supset</code>	\vdash	<code>\vdash</code>	\gtrless	<code>\gtrless</code>
\approx	<code>\approx</code>	\circeq	<code>\circeq</code>	\sqsubset	<code>\sqsubset</code>
\subseteq	<code>\subseteq</code>	\succsim	<code>\succsim</code>	\sqsupset	<code>\sqsupset</code>
\supseteq	<code>\supseteq</code>	\gtrsim	<code>\gtrsim</code>	\vartriangleright	<code>\vartriangleright</code>
\cong	<code>\cong</code>	\gtrapprox	<code>\gtrapprox</code>	\vartriangleleft	<code>\vartriangleleft</code>
\sqsubset	<code>\sqsubset</code>	\multimap	<code>\multimap</code>	\trianglerighteq	<code>\trianglerighteq</code>
\sqsupset	<code>\sqsupset</code>	\therefore	<code>\therefore</code>	\trianglelefteq	<code>\trianglelefteq</code>
\bowtie	<code>\bowtie</code>	\because	<code>\because</code>	\between	<code>\between</code>
\in	<code>\in</code>	\doteqdot	<code>\doteqdot</code>	\blacktriangleright	<code>\blacktriangleright</code>
\ni	<code>\ni</code>	\triangleq	<code>\triangleq</code>	\blacktriangleleft	<code>\blacktriangleleft</code>
\propto	<code>\propto</code>	$\prec\sim$	<code>\prec\sim</code>	\vartriangle	<code>\vartriangle</code>
\vdash	<code>\vdash</code>	$\less\sim$	<code>\less\sim</code>	\equiv	<code>\equiv</code>
\dashv	<code>\dashv</code>	\lessapprox	<code>\lessapprox</code>	\lesseqgtr	<code>\lesseqgtr</code>

	<code>\gtreqless*</code>		<code>\eqsim*</code>		<code>\Eqqcolon**</code>
	<code>\lesseqqgtr*</code>		<code>\shortmid*</code>		<code>\Coloneq**</code>
	<code>\gtreqqless*</code>		<code>\shortparallel*</code>		<code>\Eqqcolon**</code>
	<code>\varpropto*</code>		<code>\thicksim*</code>		<code>\strictli**</code>
	<code>\Subset*</code>		<code>\thickapprox*</code>		<code>\strictfi**</code>
	<code>\Supset*</code>		<code>\approxeq*</code>		<code>\strictiff**</code>
	<code>\subseteqq*</code>		<code>\preapprox*</code>		<code>\circledless**</code>
	<code>\supseteqq*</code>		<code>\succapprox*</code>		<code>\circledgtr**</code>
	<code>\bumpeq*</code>		<code>\curvearrowleft*</code>		<code>\lJoin**</code>
	<code>\Bumpeq*</code>		<code>\curvearrowright*</code>		<code>\rJoin**</code>
	<code>\lll*</code>		<code>\backepsilon*</code>		<code>\Join**</code>
	<code>\ggg*</code>		<code>\mappedfromchar**</code>		<code>\openJoin**</code>
	<code>\pitchfork*</code>		<code>\Mapstochar**</code>		<code>\lrtimes**</code>
	<code>\backsim*</code>		<code>\Mappedfromchar**</code>		<code>\opentimes**</code>
	<code>\backsimeq*</code>		<code>\mmapstochar**</code>		<code>\Perp**</code>
	<code>\lvertneqq*</code>		<code>\mmappedfromchar**</code>		<code>\leadstoext**</code>
	<code>\gvertneqq*</code>		<code>\Mmapstochar**</code>		<code>\leadsto**</code>
	<code>\lneqq*</code>		<code>\Mmappedfromchar**</code>		<code>\boxright**</code>
	<code>\gneqq*</code>		<code>\varparallel**</code>		<code>\boxleft**</code>
	<code>\lneq*</code>		<code>\varparallelinv**</code>		<code>\boxdotright**</code>
	<code>\gneq*</code>		<code>\colonapprox**</code>		<code>\boxdotleft**</code>
	<code>\precnsim*</code>		<code>\colonsim**</code>		<code>\Diamondright**</code>
	<code>\succnsim*</code>		<code>\Colonapprox**</code>		<code>\Diamondleft**</code>
	<code>\lnsim*</code>		<code>\Colonsim**</code>		<code>\Diamonddotright**</code>
	<code>\gnsim*</code>		<code>\doteq**</code>		<code>\Diamonddotleft**</code>
	<code>\precneqq*</code>		<code>\multimapinv**</code>		<code>\boxRight**</code>
	<code>\succneqq*</code>		<code>\multimapboth**</code>		<code>\boxLeft**</code>
	<code>\precnapprox*</code>		<code>\multimapdot**</code>		<code>\boxdotRight**</code>
	<code>\succnapprox*</code>		<code>\multimapdotinv**</code>		<code>\boxdotLeft**</code>
	<code>\lnapprox*</code>		<code>\multimapdotboth**</code>		<code>\DiamondRight**</code>
	<code>\gnapprox*</code>		<code>\multimapdotbothA**</code>		<code>\DiamondLeft**</code>
	<code>\diagup*</code>		<code>\multimapdotbothB**</code>		<code>\DiamonddotRight**</code>
	<code>\diagdown*</code>		<code>\VDash**</code>		<code>\DiamonddotLeft**</code>
	<code>\varsubsetneq*</code>		<code>\VvDash**</code>		<code>\circlearight**</code>
	<code>\varsupsetneq*</code>		<code>\cong**</code>		<code>\circleleft**</code>
	<code>\subsetneqq*</code>		<code>\preceq**</code>		<code>\circleddotright**</code>
	<code>\supsetneqq*</code>		<code>\succeq**</code>		<code>\circleddotleft**</code>
	<code>\varsubsetneqq*</code>		<code>\coloneqq**</code>		<code>\multimapbothvert**</code>
	<code>\varsupsetneqq*</code>		<code>\eqqcolon**</code>		<code>\multimapdotbothvert**</code>
	<code>\subsetneq*</code>		<code>\coloneq**</code>		<code>\multimapdotbothBvert**</code>
	<code>\supsetneq*</code>		<code>\eqcolon**</code>		<code>\multimapdotbothAvert**</code>
			<code>\Coloneq**</code>		

Pokud před relací napíšeme `\not`, získáme přeškrtnutou relaci. PlainTeX dále disponuje další rozsáhlou sadou **relací** ve tvaru šipek:

	<code>\leftarrow</code>		<code>\leftrightarrow</code>		<code>\leftharpoonup</code>
	<code>\Leftarrow</code>		<code>\Leftrightarrow</code>		<code>\leftharpoondown</code>
	<code>\rightarrow</code>		<code>\mapsto</code>		<code>\leftrightharpoons*</code>
	<code>\Rightarrow</code>		<code>\hookleftarrow</code>		<code>\rightleftharpoons</code>

\rightarrow	<code>\rightharpoonup</code>	\swarrow	<code>\swarrow</code>	\P	<code>\Rsh*</code>
\rightarrow	<code>\rightharpoonowdown</code>	\nwarrow	<code>\nwarrow</code>	\rightsquigarrow	<code>\rightsquigarrow*</code>
\leftarrow	<code>\longleftarrow</code>	\twoheadrightarrow	<code>\twoheadrightarrow*</code>	\leftrightsquigarrow	<code>\leftrightsquigarrow*</code>
\Leftarrow	<code>\Longleftarrow</code>	\twoheadleftarrow	<code>\twoheadleftarrow*</code>	\looparrowleft	<code>\looparrowleft*</code>
\rightarrow	<code>\longrightarrow</code>	\leftleftarrows	<code>\leftleftarrows*</code>	\looparrowright	<code>\looparrowright*</code>
\Rightarrow	<code>\Longrightarrow</code>	\rightrightarrows	<code>\rightrightarrows*</code>	\Rrightarrow	<code>\Rrightarrow*</code>
\Leftrightarrow	<code>\Longleftrightarrow</code>	\uparrows	<code>\uparrows*</code>	\Lleftarrow	<code>\Lleftarrow*</code>
\mapsto	<code>\longmapsto</code>	\downdownarrows	<code>\downdownarrows*</code>	\Nearrow	<code>\Nearrow**</code>
\hookrightarrow	<code>\hookrightarrow</code>	\upharpoonright	<code>\upharpoonright*</code>	\Searrow	<code>\Searrow**</code>
\uparrow	<code>\uparrow</code>	\downharpoonright	<code>\downharpoonright*</code>	\Narrow	<code>\Narrow**</code>
\Uparrow	<code>\Uparrow</code>	\upharpoonleft	<code>\upharpoonleft*</code>	\Svarrow	<code>\Svarrow**</code>
\downarrow	<code>\downarrow</code>	\downharpoonleft	<code>\downharpoonleft*</code>	\dashleftarrow	<code>\dashleftarrow**</code>
\Downarrow	<code>\Downarrow</code>	\rightarrowtail	<code>\rightarrowtail*</code>	\dashrightarrow	<code>\dashrightarrow**</code>
\Updownarrow	<code>\Updownarrow</code>	\leftarrowtail	<code>\leftarrowtail*</code>	\dashleftrightharrow	<code>\dashleftrightharrow**</code>
\Downarrow	<code>\Downarrow</code>	\leftrightarrows	<code>\leftrightarrows*</code>	\leftrightsquigarrow	<code>\leftrightsquigarrow**</code>
\nearrow	<code>\nearrow</code>	\rightleftarrows	<code>\rightleftarrows*</code>		
\searrow	<code>\searrow</code>	\Lsh	<code>\Lsh*</code>		

Specialitou T_EXu jsou tzv. **velké operátory**, což není jenom suma, ale taky následující znaky:

\sum	<code>\sum</code>	\bigcap	<code>\bigcap</code>	\odot	<code>\bigodot</code>
\prod	<code>\prod</code>	\bigcup	<code>\bigcup</code>	\otimes	<code>\bigotimes</code>
\coprod	<code>\coprod</code>	\bigsqcup	<code>\bigsqcup</code>	\oplus	<code>\bigoplus</code>
\int	<code>\int</code>	\bigvee	<code>\bigvee</code>	\uplus	<code>\biguplus</code>
\oint	<code>\oint</code>	\bigwedge	<code>\bigwedge</code>		

Velké operátory jsou skutečně velké v display módu a jsou poněkud menší ve vnitřním matematickém módu. Bižuterie těchto operátorů se umístí nad a pod operátor v display módu a mírně vpravo od operátoru v odstavci. Příklad:

V odstavci je $\sum_{n=1}^{\infty} \frac{1}{n^2}$ konvergentní a na samostatném řádku je $\sum_{n=1}^{\infty} \frac{1}{n^2}$ taky konvergentní.

V odstavci je $\sum_{n=1}^{\infty} \frac{1}{n^2}$ konvergentní a na samostatném řádku je

$$\sum_{n=1}^{\infty} \frac{1}{n^2}$$

taky konvergentní.

T_EX také umožňuje nastavit pevnou polohu bižuterie u velkých operátorů. Následuje-li těsně za operátorem a před symboly $_$ a $^$ příkaz `\limits`, bižuterie bude nahoře a dole, je-li tam příkaz `\nolimits`, bižuterie bude vpravo. V obou případech bez ohledu na aktuální matematický mód. Takže například $\sum\limits_1^{\infty}$ vytvoří sumu s bižutérií nahoře a dole i v textu odstavce: \sum_1^{∞} .

TX fonty (po zavedení `tx-math.tex`) přidávají další **velké operátory**:

\bigplus	<code>\bignplus**</code>	\bigsqcap	<code>\bigsqcap**</code>
\bigsqcupplus	<code>\bigsqcupplus**</code>	\oiint	<code>\oiintop**</code>
\bigsqcapplus	<code>\bigsqcapplus**</code>	\oint	<code>\ointctrlockwiseop**</code>

\oint	<code>\ointclockwiseop**</code>	\oint	<code>\ointctrlockwiseop**</code>
\oint	<code>\sqintop**</code>	\oint	<code>\varoiintclockwiseop**</code>
\times	<code>\varprod**</code>	\oint	<code>\ointclockwiseop**</code>
\iint	<code>\iintop**</code>	\oint	<code>\varoiintctrlockwiseop**</code>
\iiint	<code>\iiintop**</code>	\oint	<code>\oiintctrlockwiseop**</code>
\iiint	<code>\iiiintop**</code>	\oint	<code>\varoiiiiintclockwiseop**</code>
$\int \cdots \int$	<code>\idotsintop**</code>	\oint	<code>\oiintclockwiseop**</code>
\oint	<code>\oiintop**</code>	\oint	<code>\varoiiiiintctrlockwiseop**</code>
\oint	<code>\varointctrlockwiseop**</code>	\oint	<code>\sqintop**</code>
\oint	<code>\varointclockwiseop**</code>	\oint	<code>\sqiiintop**</code>
\oint	<code>\fintop**</code>		

Ustálené matematické zkratky definované v plainTeXu:

\arccos	<code>\arccos</code>	\deg	<code>\deg</code>	\lg	<code>\lg</code>	\Pr	<code>\Pr</code>
\arcsin	<code>\arcsin</code>	\det	<code>\det</code>	\lim	<code>\lim</code>	\sin	<code>\sin</code>
\arg	<code>\arg</code>	\dim	<code>\dim</code>	\liminf	<code>\liminf</code>	\sinh	<code>\sinh</code>
\cos	<code>\cos</code>	\exp	<code>\exp</code>	\limsup	<code>\limsup</code>	\sup	<code>\sup</code>
\cosh	<code>\cosh</code>	\gcd	<code>\gcd</code>	\ln	<code>\ln</code>	\tan	<code>\tan</code>
\cot	<code>\cot</code>	\hom	<code>\hom</code>	\log	<code>\log</code>	\tanh	<code>\tanh</code>
\coth	<code>\coth</code>	\inf	<code>\inf</code>	\max	<code>\max</code>		
\csc	<code>\csc</code>	\ker	<code>\ker</code>	\min	<code>\min</code>		

Bižuterie pod zkratkami `\lim` a `jim` podobnými se umístí podle stejného pravidla, jako u velkých operátorů. Vyzkoušejte: `\lim_{x\to 0_+} \{1\over x\} = \infty`. Jak jsou zkratky definovány najdete v souboru `plain.tex`. Můžete si snadno definovat další.

6.6 Závorky

Přímo zapsaná závorka bez další informace má ve vzorci svou běžnou velikost. Množinové závorky `\{...\}` se v TeXu zapisují pomocí `\{...\}`.

Na zvětšování závorek slouží příkazy `\left<závorka>` a `\right<závorka>`. Těmi je možné obklopit matematický výraz. Příkazy `\left` a `\right` musejí vzájemně párovat na stejné úrovni skupiny ve vzorečku. Závorky za nimi zapsané mohou být libovolné z těchto: `() [] \{ \} < > |`. Specifikované závorky za `\left` a `\right` budou odpovídajícím způsobem zvětšeny podle velikosti obklopeného výrazu. Příklad:

$$\text{\texttt{\$}\texttt{\$}\texttt{\backslash left}(1 + \{x\over n\} \texttt{\backslash right})^n. \texttt{\$}\texttt{\$}}$$

$$\left(1 + \frac{x}{n}\right)^n.$$

Správné párování příkazů `\left` a `\right` je povinné, ale jejich argumenty mohou být jakékoli závorky. Přitom tečka jako argument znamená neviditelnou závorku. Příklad:

$$\text{\texttt{\$}\texttt{\$}\texttt{\backslash left. \texttt{\backslash left}(1+\{x\over n\}\texttt{\backslash right})^n \texttt{\backslash right}|_{n\to\infty} = e^x. \texttt{\$}\texttt{\$}}}$$

$$\left(1 + \frac{x}{n}\right)^n \Big|_{n \rightarrow \infty} = e^x.$$

Pozor na znaky `<` a `>`, které fungují jako lomené závorky jen při použití `\left` nebo `\right`. Samostatně vytvoří znaky „menší než“ a „větší než“. Chcete-li samostatnou lomenou závorku, pište `\angle` a `\rangle`. Příklad:

`$ 0 \leq x < a $` je totéž jako `$ x\in \langle 0, a \rangle $`.

$0 \leq x < a$ je totéž jako $x \in (0, a)$.

Kromě závorek `() [] \{ \} < > |` jsou v `plainTeX` připraveny další natahovací závorky (použitelné za příkazy `\left` a `\right`). Zde je jejich přehled:

<code>\backslash</code>	<code>\updownarrow</code>	<code>\lfloor</code>
<code>\langle</code>	<code>\Downarrow</code>	<code>\rfloor</code>
<code>\rangle</code>	<code>\Uparrow</code>	<code>\lbrack**</code>
<code>\ </code>	<code>\updownarrow</code>	<code>\rbrack**</code>
<code>\downarrow</code>	<code>\lceil</code>	<code>\llbracket**</code>
<code>\uparrow</code>	<code>\rceil</code>	<code>\rrbracket**</code>

Všechny tyto „závorky“ fungují i v základní velikosti, bez použití `\left`, `\right`.

Matice jsou obvykle taky obklopeny velkými závkami. Pro matice je v `plainTeX` připraveno makro `\matrix{<data>}`. Přitom `<data>` jsou dělena do řádků pomocí příkazu `\cr` a jednotlivé položky jsou odděleny pomocí znaku `&`. Příklad:

`$$ \left[\matrix{a&b&c\cr d&e&f\cr g&h&i} \right]. $$`

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}.$$

Protože kulaté závorky kolem matic jsou nejobvyklejší, je pro takovou matici připravena zkratka `\pmatrix{<data>}`. V tomto případě není nutné psát kolem `\pmatrix` další závorky. Tedy:

`$$ {\bf J}_\lambda = \pmatrix{1&\lambda&0\cr 0&1&\lambda\cr 0&0&1}. $$`

$$J_\lambda = \begin{pmatrix} 1 & \lambda & 0 \\ 0 & 1 & \lambda \\ 0 & 0 & 1 \end{pmatrix}.$$

Někdy se sejde více vnořených závorek. `PlainTeX` umožňuje pro větší přehlednost odlišit je velikostí. Pro levé závorky slouží řada maker `\bigl`, `\Bigl`, `\biggl`, `\Biggl`, a pro pravé `\bigr`, `\Bigr`, `\biggr`, `\Biggr`. Makra se používají jako prefix před závkami. Příklad:

`$$ \bigr(f(x)-g(x)\bigr) \Bigl((x-y) + \bigr(f(x)-f(y)\bigr) \bigl(g(x)-g(y)\bigr)\Bigr). $$`

$$(f(x) - g(x)) \left((x - y) + (f(x) - f(y)) (g(x) - g(y)) \right).$$

6.7 Odmocniny, akcenty

Odmocniny, pruh nebo svorku nad výrazem nebo pod výrazem vytvoříme například takto:

<code>\sqrt{a^2 + b^2}</code>	$\sqrt{a^2 + b^2}$	<code>\root n\of{2x-3}</code>	$\sqrt[n]{2x-3}$
<code>\overline{a+bi}</code>	$\overline{a+bi}$	<code>\underline{x+y}</code>	$\underline{x+y}$
<code>\overrightarrow{u+v}</code>	$\overrightarrow{u+v}$		
<code>\overleftarrow{u+v}</code>	$\overleftarrow{u+v}$		
<code>\overbrace{c+c+\cdots+c}^{k\times}</code>	$\overbrace{c+c+\cdots+c}^{k \times}$		
<code>\underbrace{1+1+\cdots+1}_n</code>	$\underbrace{1+1+\cdots+1}_n$		

Příklad vložené odmocniny:

`$$ \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}. $$`

$$\sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + x}}}}$$

Nad jednotlivé znaky matematické sazby je možné vložit **matematický akcent**. Akcenty uvedené v tabulce v sekci 3.2 fungují jen v textu mimo matematiku a naopak následující akcenty je možno použít jen v matematické sazbě a slouží pro různé vyznačení proměnných.

\acute{a}	<code>\acute{a}</code>	\breve{a}	<code>\breve{a}</code>	\dot{a}	<code>\dot{a}</code>	\grave{a}	<code>\grave{a}</code>	\vec{a}	<code>\vec{a}</code>
\bar{a}	<code>\bar{a}</code>	\check{a}	<code>\check{a}</code>	\ddot{a}	<code>\ddot{a}</code>	\hat{a}	<code>\hat{a}</code>	\tilde{a}	<code>\tilde{a}</code>

PlainTeX ještě definuje dva roztahovací akcenty, které mají snahu (i když ne příliš značnou) se rozprostřít nad celý vzorec:

$\widetilde{a+b}$	<code>\widetilde{a+b}</code>	$\widehat{a+b}$	<code>\widehat{a+b}</code>
-------------------	------------------------------	-----------------	----------------------------

6.8 Speciality

• **Mezery** • Pokud není mezerování vzorce uspokojivé, je možno přidat pomocí `\,` malou nepružnou mezeru, dále `\;` je mírně větší a pružná mezera a `\!` je záporná mezera velikosti jako `\,`. Tyto mezery fungují jen v matematickém módu.¹⁾ V horizontálním i matematickém módu funguje mezislovní mezera explicitně zadaná příkazem `\quad` a dále velké mezery `\quad` (mezera ve velikosti písma) a `\qquad` (dvojnásobná `\quad`). Příklad:

`$$ \alpha\,,(x+y)\,,\quad \int_a^b f(x)\,dx\,,\quad \Gamma_i. $$`

$$\alpha(x+y), \quad \int_a^b f(x) dx, \quad \Gamma_i.$$

V matematické sazbě také funguje mezera daná příkazem `\hspace{velikost}`.

• **Text pomocí `\hbox` v matematice** • Pokud potřebujete ve vzorečku napsat nějaké vlídne slovo, je možné do vzorečku vložit `\hbox{<text>}`. Tento `<text>` je T_EXem zpracován ve vnitřním horizontálním módu, tj. implicitně antikvou, fungují tam textové přepínače fontů a mezery mezi slovy. Příklad:

`$$\sum_{n=0}^{\infty} (-1)^n a_n \hbox{ konverguje, je-li } a_n \searrow 0. $$`

$$\sum_{n=0}^{\infty} (-1)^n a_n \text{ konverguje, je-li } a_n \searrow 0.$$

Povšimněte si, že před slovem „konverguje“ je v boxu mezera, která se vytiskne. Naopak mezera před sekvencí `\hbox` je v matematice a je ignorována. Také stojí za povšimnutí, že uvnitř `\hbox` se znovu přechází do vnořeného matematického módu. Tentýž výsledek se dá získat naopak ukončením `\hbox`:

`$$\sum_{n=0}^{\infty} (-1)^n a_n \hbox{ konverguje, je-li } a_n \searrow 0. $$`

Rozdíl mezi těmito dvěma přístupy bychom poznali v okamžiku, kdy za vlídným slovem je matematický text se zlomkem nebo velkým operátorem. V prvním případě by ten zlomek nebo operátor byl malý, ve druhém případě velký.

¹⁾ Po zavedení O_Pmac funguje `\,` taky v horizontálním módu, používá se třeba mezi čísly a jednotkami.

• **Poznámka** • Pouhé přepnutí do `\rm` v matematickém módu nevytvoří žádné mezery mezi slovy. Museli byste je tam vnutit pomocí `_`.

• **Rozbočka pomocí `\cases`** • Je-li například funkce dána různými vzorečky, potřebujeme rozbočku, pro kterou je připraveno makro `\cases`. Příklad:

```
$$ f(x) = \cases{1/x & pro $x\neq 0$, \cr 7 & pro $x=0$.} $$
```

$$f(x) = \begin{cases} 1/x & \text{pro } x \neq 0, \\ 7 & \text{pro } x = 0. \end{cases}$$

Makro `\cases{<data>}` má svá data členěna do řádků pomocí `\cr`, přičemž na každém řádku jsou dvě položky oddělené znakem `&`. První položka se zpracuje v matematickém módu, zatímco druhá položka ve vnitřním horizontálním módu (textovém). Takže pokud chcete ve druhé položce napsat něco matematického, musíte tam vstoupit do matematického módu pomocí dolarů.

Pokud jsou v `\cases` řádky příliš natěsnány na sebe, pak lze za příkaz `\cr` přidat třeba `\noalign{\smallskip}`. Toto je obecná vlastnost příkazu `\cr`, že za něj lze vložit vertikální materiál pomocí `\noalign{<materiál>}`.

• **Trojtečky** • Pomocí `\ldots` umístíme tečky na účař a pomocí `\cdots` umístíme tečky na osu společně se znaky $+$, $-$. Je třeba tečky umísťovat s ohledem na okolní symboly, tj. například mezi znaky $+$ patří na osu a kolem čárek na účař. Čárka nebo znaménko operace se píše před i za trojtečkou. Příklad:

```
$$ M=\{a_1, a_2, \ldots, a_n\}, \quad \text{quad } s_M = a_1+a_2+\cdots+a_n. $$
```

$$M = \{a_1, a_2, \dots, a_n\}, \quad s_M = a_1 + a_2 + \dots + a_n.$$

PlainT_EX definuje ještě vertikální trojtečku `\vdots` a šikmou `\ddots`. Příklad:

```
$$ {\bf A} = \pmatrix{a_{11} & a_{12} & \ldots & a_{1n} \cr
a_{21} & a_{22} & \ldots & a_{2n} \cr
\vdots & \vdots & \ddots & \vdots \cr
a_{m1} & a_{m2} & \ldots & a_{mn}}. $$
```

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}.$$

Mimo matematický mód funguje trojtečka `\dots`. Za třemi tečkami nikdy nepíšeme tečku čtvrtou z konce věty...

• **Zmenšený text nad relací** • Makro `\buildrel<text nad relací>\over<relace>` dá nad znak relace text. Příklad:

```
$$ \alpha\cdot(x_1,x_2,\ldots,x_n) \buildrel \rm def \over =
(\alpha x_1, \alpha x_2,\ldots, \alpha x_n). $$
```

$$\alpha \cdot (x_1, x_2, \dots, x_n) \stackrel{\text{def}}{=} (\alpha x_1, \alpha x_2, \dots, \alpha x_n).$$

• **Poznámka** • Povšimněte si, že symbol násobení byl vytvořen pomocí `\cdot`, což umístí správně tečku na matematickou osu. V literatuře se bohužel často setkáváme s nesprávným umístěním tečky pro násobení na účař.

• **Fantóm** • Pro vyrovnání nebo podepření matematické sazby se používá tzv. *fantóm*. Je to utajený nevytištěný vzorec, který nicméně v sazbě „překáží“, jakoby vytištěn byl. K dispozici jsou makra `\phantom{<vzorec>}`, `\vphantom{<vzorec>}` nebo `\hphantom{<vzorec>}`.

Druhé uvedené makro vytvoří překážku stejné výšky, jako $\langle vzorec \rangle$, ale nulové šířky. Třetí makro vytvoří překážku šířky $\langle vzorce \rangle$ s nulovou výškou. Je-li $\langle vzorec \rangle$ zapsán jako jediný token, není třeba psát svorky kolem něj. Příklad ukážeme na maticích, ve kterých položky implicitně pod sebou centrují, což ve výjimečných případech nevypadá dobře:

$$\begin{pmatrix} -1 & 1 \\ 0 & -3 \end{pmatrix}.$$

Je tedy možné použít `\phantom`:

```


$$\begin{matrix} \text{\$} \text{\texttt{\textbackslash pmatrix}} \{x+1 \ \& \ y \ \text{\texttt{\textbackslash cr}} \ x \ \& \ y-1\} \ \text{\texttt{\textbackslash cdot}} \\ \text{\texttt{\textbackslash pmatrix}} \{-1 \ \& \ \text{\texttt{\textbackslash phantom}}+1 \ \text{\texttt{\textbackslash cr}} \ \text{\texttt{\textbackslash phantom}}+0 \ \& \ -3\} = \\ \text{\texttt{\textbackslash pmatrix}} \{-x-1 \ \& \ x-3y+1 \ \text{\texttt{\textbackslash cr}} \ -x \ \& \ x-3y+3\}. \text{\texttt{\textbackslash \$}} \end{matrix}$$


$$\begin{pmatrix} x+1 & y \\ x & y-1 \end{pmatrix} \cdot \begin{pmatrix} -1 & 1 \\ 0 & -3 \end{pmatrix} = \begin{pmatrix} -x-1 & x-3y+1 \\ -x & x-3y+3 \end{pmatrix}.$$


```

V tomto příkladě fantóm zaujal místo jako plus, které je stejně velké jako mínus, takže se čísla pod sebou zarovnala. Jiný příklad použití, kde znak „t“ podpírá sazbu, aby byly pruhy ve stejné výšce:

```


$$\text{\$} \text{\texttt{\textbackslash overline}} \ t + \text{\texttt{\textbackslash overline}} \ u = \text{\texttt{\textbackslash overline}} \{t+u\} \text{\texttt{\textbackslash \$}} \text{ zatímco}$$


$$\text{\$} \text{\texttt{\textbackslash overline}} \ t + \text{\texttt{\textbackslash overline}} \{\text{\texttt{\textbackslash u\textbackslash vphantom}} \ t\} = \text{\texttt{\textbackslash overline}} \{t+u\} \text{\texttt{\textbackslash \$}}.$$


```

Ukázka dává tento výsledek: $\overline{t} + \overline{u} = \overline{t+u}$ zatímco $\overline{t} + \overline{u} = \overline{t+u}$.

• **Poznámka** • Fantóm funguje i mimo matematiku a používá se například pro dorovnání výjimek v tabulkách.

• **Desetinná čárka místo tečky** • Česká norma vyžaduje pro zápis čísel použití desetinné čárky, například 3,1415. Američani místo toho píší tečku. S tečkou nemají v \TeX u problém, protože je nastavena jako základní znak, takže nevznikají kolem ní mezery. Ovšem čárka je nastavena jako interpunkce, takže za ní vzniká nežádoucí mezera: 3, 1415. Pro psaní desetinných čísel v češtině je tedy potřeba potlačit mezerování za čárkou, což lze pomocí obklopení znaku svorkami: $\{3, \}1415$.

Následující alternativní řešení problému čárky ukazuje, jak je matematická sazba z hlediska programování maker flexibilní. Napíšete-li `\mathcode'\.=\'`, někde do úvodu dokumentu, promění se všechny tečky v matematické sazbě v čárky, které nemají kolem sebe mezery. Toto vyžaduje pro uživatele náhradní řešení, pokud tečku v matematické sazbě skutečně chce. Může třeba použít `\.`, pokud definujeme `\mathchardef'\.=\'`.

Vlastnosti příkazů `\mathcode` a `\mathchardef` přesahují rámec tohoto textu. Je možné je dohledat v TBN.

• **Synonyma** • Některé řídicí sekvence mají v $\text{plain}\text{\TeX}$ u ještě svou alternativu.

\neq	<code>\neq</code>	jako <code>\not=</code>	\vee	<code>\lor</code>	jako <code>\vee</code>
\neg	<code>\ne</code>	jako <code>\not=</code>	\neg	<code>\lnot</code>	jako <code>\neg</code>
\geq	<code>\ge</code>	jako <code>\geq</code>	\ni	<code>\owns</code>	jako <code>\ni</code>
\leq	<code>\le</code>	jako <code>\leq</code>	$\{$	<code>\lbrace</code>	jako <code>\{</code>
\rightarrow	<code>\to</code>	jako <code>\rightarrow</code>	$\}$	<code>\rbrace</code>	jako <code>\}</code>
\wedge	<code>\land</code>	jako <code>\wedge</code>	\parallel	<code>\Vert</code>	jako <code>\parallel</code>

• **Další možnosti** • Pro matematickou sazbu existuje celá řada dalších příkazů a maker. V následujícím přehledu jsou uvedeny jen heslovitě. Podrobnější význam těchto sekvencí si může čtenář dohledat v TBN.

`\displaystyle`, `\textstyle`, `\scriptstyle`, `\scriptscriptstyle` – nastavení velikosti fontů jako v display módu, ve vnitřním matematickém módu, v indexech (mocnínách) a v indexech (mocninách) vyšší úrovně.

`\mathchoice`, `\mathpalette` – řešení variantní sazby podle toho, zda sazba probíhá v display módu, ve vnitřním matematickém módu, v indexech (mocnínách) a v indexech (mocninách) vyšší úrovně.

`\vcenter` – jako `\vbox`, ale výsledek je centrován na matematickou osu.

• **Poznámka** • Z krátkého přehledu v této sekci je zřejmé, že \TeX sice v mnoha případech vytvoří matematickou sazbu správně, ale někdy je mu třeba trošku pomoci. K tomu je potřeba dobře znát pravidla matematické sazby. Ta jsou z hlediska \TeX u pěkně shrnuta v [8].

6.9 Krájení vzorečků do více řádků

Výsledek sazby vnitřního matematického módu v odstavci se stává součástí interního řádku, který \TeX posléze rozlomí do řádků odstavce. Má-li \TeX při této činnosti potřebu zlomit řádek uvnitř vzorečku, udělá to pouze za binární operací nebo za relací. Dostane za to trest nastavený v registrech `\binoppenalty` a `\relpenalty`. Nastavíte-li tyto registry na hodnotu 10000, \TeX nebude ve vzorečkách lámat. Chcete-li potlačit zlom jen na určitém místě v sazbě, je možné v daném místě psát například: `$a + b + \nobreak c$`.

Sazba vytvořená mezi zdvojenými dolary (display mód) vždy zaujme jeden řádek implicitně centrováný. Pokud tam napíšete moc dlouhý vzorec, \TeX to nezajímá a pouze ohlásí na terminál a do `.log` souboru `Overfull \hbox`. Autor se v tomto případě musí sám rozhodnout, jak chce dlouhý vzorec rozlámat. Má následující možnosti:

Pro víceřádkové vzorce bez zarovnání lze použít `\displaylines`.

```


$$\begin{aligned} (3x^3 + 4x^2 + 5x + 6) \cdot (x^6 + x^2 + 5) &= \\ = 3x^9 + 4x^8 + 5x^7 + 6x^6 + 3x^5 + 4x^4 + 20x^3 + 26x^2 + 25x + 30. \end{aligned}$$


```

$$\begin{aligned} (3x^3 + 4x^2 + 5x + 6) \cdot (x^6 + x^2 + 5) &= \\ = 3x^9 + 4x^8 + 5x^7 + 6x^6 + 3x^5 + 4x^4 + 20x^3 + 26x^2 + 25x + 30. \end{aligned}$$

Jednotlivé řádky v `\displaylines` jsou odděleny znakem `\cr`. Každý jednotlivý řádek je centrován. Znak relace, ve kterém jste se rozhodli zlomit řádek, by se měl podle českých typografických pravidel zopakovat na dalším řádku znovu.

Pro sazbu více vzorců pod sebou se zarovnanými relacemi slouží `\eqalign`.

```


$$\begin{aligned} x + 2y + 3z &= 600 \\ 12x + y - 3z &= 7 \\ 4x - y + 5z &= -5 \end{aligned}$$


```

Při použití `\eqalign{<data>}` jsou `<data>` také rozdělena do řádků pomocí `\cr`. Kromě toho ale v každém řádku se musí vyskytovat jeden znak `&`, který určuje místo, kde budou řádky přesně pod sebou. Tento znak se obvykle píše před relaci.

Pokud chtějí autoři svou soustavu rovnic „vytunit“, mohou použít fantóma:

```


$$\begin{aligned} x + 2y + 3z &= 600 \\ 12x + \phantom{y} - 3z &= \phantom{y}7 \\ 4x - \phantom{y} + 5z &= -5 \end{aligned}$$


```

Makro `\eqalign` lze použít i na řízené zalomení vzorce:

```


$$\begin{aligned}
p(x)q(x) &= (3x^3 + 4x^2 + 5x + 6) \cdot (x^6 + x^2 + 5) = \\
&= 3x^9 + 4x^8 + 5x^7 + 6x^6 + 3x^5 + 4x^4 + 20x^3 + 26x^2 + 25x + 30.
\end{aligned}$$


```

Na číslování rovnic slouží příkaz `\eqno`*⟨značka⟩*, který je potřeba zapsat před ukončovací `$$`. Například:

```


$$f(x+y) = f(x) + f(y). \quad \text{\eqno (1)}$$


```

Značka se umístí vpravo ke straně a je zpracována v matematickém módu. Místo `\eqno` lze použít `\leqno`*⟨značka⟩*, pak se značka umístí k levému okraji. Makro `OPmac` nabízí automatické číslování rovnic, viz sekci 7.4.

Při použití `$$ \eqalign{⟨soustava⟩} \eqno⟨značka⟩ $$` dostane přidělenou značku soustava jako celek. Pokud chcete dát značku každé rovnici zvlášť, je potřeba použít makro `\eqalignno` například takto:

```


$$\begin{aligned}
x + 2y + 3z &= 600 && \text{\rm(A)} \\
12x + y - 3z &= 7 && \text{\rm(B)} \\
4x - y + 5z &= -5 && \text{\rm(C)}
\end{aligned}$$


```

Každý řádek nyní obsahuje dva znaky `&`. První je pro označení místa, které bude zarovnáno pod sebou a druhý odděluje text pro značku.

6.10 Přidání další matematické abecedy

V sekci 6.3 byly shrnuty matematické abecedy, které jsou k dispozici po zavedení souboru `ams-math.tex` nebo `tx-math.tex`. Nyní si ukážeme, jak je možné přidat další matematickou abecedu dle vlastního výběru. V ukázce je použit font `pzcmi8z`, o kterém víme, že obsahuje písmo *Zapf Chancery*.

Po zavedení `ams-math.tex` nebo `tx-math.tex` stačí psát:

```

\def\zapf {\fam 15 }
\addto\normalmath {\loadmathfamily 15 pzcmi8z } \normalmath
\addto\boldmath {\loadmathfamily 15 pzcmi8z }

```

V ukázce je použito makro `\addto` z `OPmac`, které přidá k existujícímu makru další část textu. Pokud není použit `OPmac`, je třeba si toto jednořádkové makro z `opmac.tex` opsat. Po aplikaci uvedených tří řádků je připraveno makro `\zapf`, které v matematickém módu přepíná do matematické abecedy, například takto: $\mathbf{a} + \mathbf{b} = \mathbf{c}$.

Makro `\zapf` z příkladu se stane přepínačem matematické abecedy, tedy ovlivní znaky A–Z, a–z a čísla 0–9. Podléhá správnému zmenšování v indexech a podindexech a změně

velikosti celého vzorečku pomocí `\setmathsizes`. V ukázce je vidět, že byl font `pzcmi8z` použit i pro tučnou verzi matematických vzorečků deklarovanou pomocí `\boldmath`, která se používá v nadpisech. Zcela stejný font pro tučnou verzi byl použit jenom proto, že písmo Zapf Chancery bohužel nemá tučnou variantu.

Matematické abecedy úzce souvisejí s tzv. *matematickými rodinami fontů*, do kterých se přepíná příkazem `\fam<číslo>`. Každá rodina může implementovat nejvýše jednu matematickou abecedu. Klasický \TeX disponuje jen šestnácti matematickými rodinami, které jsou číslovány od 0 do 15. Soubory `ams-math.tex` a `tx-math.tex` alokují rodiny s čísly 0 až 13, takže pro uživatele zůstávají jen rodiny 14 a 15. V ukázce byla použita rodina 15. Kromě ní už tedy může uživatel zavést jen jednu další matematickou abecedu pod číslem 14.

Je sice možné již zavedené rodiny přepsat jinými fonty, ale to můžete dělat jen tehdy, když velmi dobře víte, co činíte. Existující rodiny totiž neobsahují jen matematickou abecedu, ale též další znaky, o které tím nenávratně přijdete. Navíc mohou obsahovat cenné metrické informace, podle kterých se řídí formátování vzorečků. Když tedy dobře víte, co činíte (máte pečlivě nastudovanu sekci 5.3 z TBN), pak je možná lepší si metodou analogie vytvořit další soubor podobný jako `ams-math.tex` nebo `tx-math.tex`. Zmíněné soubory jsou komentovány a jsou tam popsány i některé další vlastnosti.

Kapitola 7

Použití OPmac

Autoři textů, kteří nechtějí programovat vlastní složitá makra a pro běžné úlohy si vystačí s připravenými makry, mohou využít soubor maker OPmac (Olšákova plainTeXová makra) [11]. Tento soubor nabízí autorům srovnatelnou funkcionalitu jako L^AT_EX, a přitom je určen pro plainTeX. Na začátku dokumentu stačí uvést:

```
\input opmac
```

a tím se autorovi otevírají možnosti popsané v této kapitole. Obvyklé zahájení dokumentu může vypadat například takto:

```
\input opmac      % zavedení makra OPmac
\chyph            % použijte csplain, zapnutí češtiny
\input lmfons     % použití Latin Modern fontů
\typesize[12/14]  % nastavení základní velikosti sazby
```

I tvůrci maker se ovšem mohou v souboru `opmac.tex` inspirovat. Podrobnou dokumentaci na technické úrovni lze dohledat na webové stránce k OPmac v souboru `opmac-d.pdf`. Hlavním krédem OPmac je slogan „v jednoduchosti je síla“. Tomuto sloganu začnou rozumět ti programátoři maker, kteří se podívají do vnitřností L^AT_EXu a do vnitřností OPmac a provedou srovnání.

OPmac nabízí autorům textů způsob značkování dokumentů podobně, jako to dělá L^AT_EX. Ovšem na rozdíl od L^AT_EXu je značkování mírně jiné a umožní vytvářet přehlednější a méně ukecané zdrojové texty. OPmac neřeší typografii dokumentu. Předpokládá se, že po zavedení OPmac se použijí další makra zaměřená na vzhled dokumentu.

V této kapitole je uživatelská dokumentace k OPmac určená především pro autory textů. Je víceméně převzatá z webových stránek¹⁾ ze souboru `opmac-u.pdf`. Na webové stránce²⁾ je dále souhrn návodů a tipů, které pokrývají plno dalších rozšíření, která se v L^AT_EXu obvykle řeší specializovanými balíčky.

7.1 Velikosti fontů a řádkování

Všechna makra popsaná v této sekci nastavují změny ve fontech a dalších parametrech jen lokálně, takže jsou-li ve skupině, za ní se nastavení vrací k původním hodnotám.

Makro `\typesize[⟨velikost fontu⟩/⟨řádkování⟩]` nastaví velikost textových i matematických fontů a řádkování³⁾. Je-li některý z parametrů prázdný, makro nastaví jen údaje plynoucí z neprázdného parametru. Parametry neobsahují jednotku, jednotka pt se doplní v makru. Příklady

```
\typesize[10/12]      % to je implicitní nastavení
\typesize[11.5/12.5]  % font velikosti 11,5 pt, řádkování 12,5 pt
\typesize[8/]         % font velikosti 8 pt, řádkování nezměněno
```

Makro `\typoscale[⟨faktor font⟩/⟨faktor řádkování⟩]` zvětší nebo zmenší velikost textových i matematických fontů resp. řádkování $\langle faktor \rangle$ krát aktuální velikost. Faktor je

¹⁾ <http://petr.olsak.net/opmac.html>.

²⁾ <http://petr.olsak.net/opmac-tricks.html>.

³⁾ Řádkování v odstavci odpovídá hodnotě nastavené na konci odstavce. Chcete-li tedy `\typesize` použít uvnitř skupiny a změnit tím řádkování odstavce, je nutné ukončit skupinu až po ukončení odstavce (až po prázdném řádku nebo `\par`).

celé číslo, přitom 1000 znamená faktor jedna ku jedné (jako za slovem `scaled` v příkazu `\font`). Je-li parametr prázdný, je to stejné, jako by byl roven 1000.

```
\typoscale[800/800] % fonty i řádkování se zmenší na 80 %
\typoscale[\magstep2/] % \magstep2 je 1440, tj. fonty se zvětší 1,44krát
```

Toto makro zvětší nebo zmenší velikost textových i matematických fontů vzhledem k aktuální velikosti písma. Takže třeba `\typoscale[500/]... \typoscale[500/]` zmenší font na polovinu a dále na čtvrtinu. Někdy je ale žádoucí (např. při přechodu na poznámky pod čarou) zmenšit fonty vzhledem ke stále stejné velikosti písma. V takovém případě stačí psát `\typobase\typoscale[⟨font⟩/⟨řádkování⟩]`. Pak se font zvětší/zmenší vzhledem k *základnímu písmu*, což je písmo nastavené po prvním použití `\typosize` nebo `\typoscale`.

Makra `\thefontsize[⟨velikost fontu⟩]` nebo `\thefontscale[⟨faktor⟩]` změní velikost jen aktuálního textového fontu, nemění žádné jiné fonty a nemění řádkování.

Všechna zde uvedená makra na změnu velikosti fontů jsou vybavena inteligencí: hledají metriku, která má svou designovanou velikost nejbližše požadované velikosti. Takže při požadavku na velikost 13pt se použije metrika `csr12 at13pt`, zatímco při velikosti 7.5pt se použije metrika `csr8 at7.5pt`. Data pro tuto inteligenci jsou přečtena ze souboru `ams-math.tex`, kde je najdete v místě užití maker `\regtfm`.

7.2 Okraje

O možnostech nastavení okrajů přímo v `TeXu` a o výchozím nastavení v `plainTeXu` bylo řečeno v sekci 4.5. `OPmac` umožňuje toto nastavení změnit makrem:

```
\margins/⟨pg⟩ ⟨formát⟩ (⟨levý⟩,⟨pravý⟩,⟨horní⟩,⟨dolní⟩)⟨jednotka⟩
například:
\margins/1 b5 (2,2,2,2)cm % nastaví všechny okraje na 2 cm pro papír B5.
⟨pg⟩... 1 = shodné okraje pro všechny stránky,
⟨pg⟩... 2 = okraje pro liché stránky, sudé mají prohozeny ⟨levý⟩/⟨pravý⟩,
⟨formát⟩... a3, a4, a5, a3l, a4l, a5l, b5, letter nebo uživatelem definovaný,
⟨levý⟩,⟨pravý⟩,⟨horní⟩,⟨dolní⟩... velikosti okrajů,
⟨jednotka⟩... mm, cm, in, pt, pc, bp, dd, cc.
```

Každý z parametrů `⟨levý⟩`, `⟨pravý⟩`, `⟨horní⟩`, `⟨dolní⟩` může být prázdný. Jsou-li prázdné oba `⟨levý⟩` i `⟨pravý⟩`, je zachováno nastavení `\hsize` a levý i pravý okraj je stejný. Je-li jen jeden z parametrů `⟨levý⟩`, `⟨pravý⟩` prázdný, zůstává zachováno `\hsize` a neurčený okraj se dopočítá. Jsou-li `⟨levý⟩` i `⟨pravý⟩` neprázdné, jsou oba okraje určeny a je podle nich upraveno `\hsize`. Analogické pravidlo platí pro `⟨horní⟩`, `⟨dolní⟩` v souvislosti s výškou sazby `\vsize`. Například

```
\margins/2 a4 (,18,,)mm % vnější okraj na dvojstraně 2*A4 je 18mm
                        % \hsize, \vsize beze změny.
```

Uživatel může před použitím `\margins` definovat vlastní `⟨formát⟩` papíru pomocí deklarace `\sdef{pgs:⟨formát⟩}{(⟨šířka⟩,⟨výška⟩)⟨jednotka⟩}`. `OPmac` například implicitně definuje:

```
\sdef{pgs:a4}{(210,297)mm} \sdef{pgs:letter}{(8.5,11)in}
\sdef{pgs:b5}{(176,250)mm} \sdef{pgs:a4l}{(297,210)mm}
```

Celou sazbu na úkor okrajů je možno zvětšit/zmenšit makrem `\magscale[⟨faktor⟩]`. Například `\magscale[500]` zmenší sazbu na polovinu. Při této změně zůstává na místě „Knuthův bod“, tj. bod o souřadnicích (1 in, 1 in) od levého a horního okraje. Sazba

samotná je zalomena zcela stejně. Jednotky použité v dokumentu jsou od této chvíle relativní. Například po `\magscale[2000]` je použita jednotka v dokumentu `1mm` ve skutečnosti `2mm`. Makro `\magscale` ponechává nezměněny jen rozměry stránek dané formátem stránek (A4, A3 atd.). Možnost použití makra: `\magscale[1414] \margins/1 a4 (,,)mm` umístí sazbu, která je určena pro tisk na a5, doprostřed stránky a4 a odpovídajícím způsobem ji zvětší, aby se to korektorům lépe četlo.

7.3 Členění dokumentu

Dokument se může skládat z kapitol, kapitola ze sekcí a sekce z podsekcí. Titul vyznačte pomocí `\tit<titul><prázdný řádek>`, kapitolu zahajte `\chap<titul><prázdný-řádek>`, dále novou sekci zahajte `\sec<titul><prázdný řádek>` a podsekcí `\secc<titul><prázdný řádek>`. Takže třeba:

```
\chap Brouci

\sec Chrousti

\secc 0 nesmrtelnosti chroustů

Bla bla bla bla ...
Bla bla bla a ještě bla.
```

Kapitoly se automaticky číslovají jedním číslem, sekce dvěma čísly (číslo kapitoly.sekce) a podsekcí třemi čísly. Pokud dokument neobsahuje kapitoly, číslo kapitoly chybí, tj. sekce má jedno číslo a podsekcí dvě.

Implicitní vzhled nadpisů kapitol, sekcí a podsekcí je dán v makrech `\printchap`, `\printsec` a `\printsecc`. Můžete se na obsah těchto maker podívat do technické dokumentace nebo do `opmac.tex`. Můžete se těmito makry inspirovat a třeba je předefinovat podle vlastního typografického návrhu.

První odstavec za titulem kapitoly, sekce a podsekcí není odsazen. Pokud jej chcete mít odsazen jako ostatní odstavce, napište `\let\firstnoindent=\relax`.

Jestliže je název kapitoly, sekce nebo podsekcí příliš dlouhý, rozlomí se do řádků. V takovém případě je někdy lepší rozdělit název do řádků manuálně. K tomu slouží makro `\nl`, které odřádkuje v místě použití (`newline`). Toto makro se navíc v obsahu chová jako mezera.

Kapitola, sekce, podsekcí se nečísluje, předchází-li `\nonum`. Kapitola, sekce, podsekcí se neobjeví v obsahu, předchází-li `\notoc`.

7.4 Další číslované objekty a odkazy na ně

Kromě kapitol, sekcí a podsekcí se automaticky číslovají ještě rovnice a popisky pod obrázky a pod tabulkami.

Pokud je na konci `display` módu uvedeno `\eqmark`, tato rovnice bude číslovaná. Formát číslování je implicitně jediné číslo uzavřené v kulaté závorce resetované při každém zahájení nové sekce. Příklad: `$$ a^2 + b^2 = c^2. \eqmark $$` vytiskne

$$a^2 + b^2 = c^2. \tag{1}$$

Je-li potřeba očíslovat jednotlivé rovnice sestavené pomocí `\eqalignno`, pak použijte `\eqmark` v posledním (třetím) sloupci například takto:

```

$$
\eqalignno{a^2+b^2 &= c^2 & \eqmark \cr
           c &= \sqrt{a^2+b^2} & \eqmark \cr}
$$

```

Ukázka dává tento výsledek:

$$a^2 + b^2 = c^2 \quad (2)$$

$$c = \sqrt{a^2 + b^2} \quad (3)$$

Dalšími číslovanými objekty jsou popisky. Popisek pod obrázky je potřeba uvést slovem `\caption/f` a popisek pod nebo nad tabulkami slovem `\caption/t`. Pak následuje text popisku ukončený prázdným řádkem. Příklad:¹⁾

```

\hfil\table{rl}{věk & hodnota \cr\noalign{\smallskip}
               0--1 & neměřitelná \cr
               1--6 & projevující se \cr
               6--12 & výrazná \cr
               12--20 & extrémní \cr
               20--60 & mírnější \cr
               60--$\infty$ & umírněná} % vytvoření tabulky
\par\nobreak\medskip
\caption/t Závislost závislosti na počítačích na věku

```

Tato ukázka vytvoří:

věk	hodnota
0–1	neměřitelná
1–6	projevující se
6–12	výrazná
12–20	extrémní
20–60	mírnější
60–∞	umírněná

Tabulka 7.4.1 Závislost závislosti na počítačích na věku

Vidíme, že makro `\caption/t` doplnilo slovo „Tabulka“ následované číslem. Toto číslo přebírá číslo sekce a doplňuje ještě číslo tabulky. Podobně se chová `\caption/f`, jen místo slova „Tabulka“ se v textu zjeví slovo „Obrázek“. Obrázky a tabulky jsou číslovány nezávisle. Popisek je centrován. Je-li popisek delší na více řádcích, je centrován poslední řádek.

Způsob číslování lze změnit jinou definicí makra `\thednum` (pro rovnici), `\thetnum` (pro tabulky) a `\thefnum` (pro obrázky). Makro `OPmac` je definuje implicitně takto:

```

\def\thednum{(\the\dnum)}
\def\thetnum{\thesecnum.\the\tnum}
\def\thefnum{\thesecnum.\the\fnum}

```

Makro `OPmac` vloží slovo „Tabulka“ v závislosti na nastaveném jazyce. Jazyk nastavují přepínače pro vzory dělení `\chyph`, `\shyph`, `\ehyph`. Při `\shyph` dostaneme „Tabulka“ a při `\ehyph` máme „Table“. Podobně se chovají slova „Obrázek/Obrázok/Figure“ a „Kapitola/Kapitola/Chapter“. Jiná automaticky generovaná slova `OPmac` nepoužívá.

¹⁾ Makro `\table` je vysvětleno v sekci 7.10. Příkaz `\hfil` posune tabulku doprostřed. Konečně konstrukce `\par\nobreak\medskip` vytvoří vertikální nezlomitelnou mezeru velikosti poloviny řádku mezi tabulkou a popiskem. Viz sekci 10.1.

Předefinovat tato slova lze pomocí `\sdef`, jak ukazuje následující příklad, který zamění celá slova za zkratky.

```
\sdef{mt:t:cs}{Tab.} \sdef{mt:t:sk}{Tab.} \sdef{mt:t:en}{Tab.}
\sdef{mt:f:cs}{Obr.} \sdef{mt:f:sk}{Obr.} \sdef{mt:f:en}{Fig.}
```

Na automaticky číslované objekty je nutno se občas v textu odkazovat. Protože dopředu nevíme, pod jakým číslem se rovnice, sekce, tabulka atd. vytiskne, je třeba použít interní lejblíky k označení odkazovaných objektů. K tomu slouží makro `\label` [*lejblík*], které musí předcházet makru, jež generuje číslo. Není nutné, aby `\label` předcházel těsně danému makru. Tedy například:

```
\label[chroust] \sec 0 nesmrtnosti chroustů

\label[zavislaci]
\hfil\table{rl}{...} % vytvoření tabulky
\caption/t Závislost závislosti na počítačích na věku.

\label[pythagoras]
$$ a^2 + b^2 = c^2 \eqmark $$
```

Nyní můžeme hovořit o~seksi~`\ref[chroust]` na straně~`\pgref[chroust]` nebo taky o~rovnici~`\ref[pythagoras]` na straně~`\pgref[pythagoras]`. Dále bude potřeba upozornit na tabulku~`\ref[zavislaci]` na straně~`\pgref[zavislaci]`, která shrnuje jistý druh závislosti.

Text z ukázky vytvoří zhruba toto: „Nyní můžeme hovořit o sekci 2.1 na straně 13 nebo taky o rovnici (1) na straně 15. Dále bude potřeba upozornit na tabulku 5.3.1 na straně 42, která shrnuje jistý druh závislosti.“

Jestliže se v textu vyskytují dopředné reference (tj. odkazujeme na objekt, který ještě není vytištěn) nebo text odkazuje na stránky (`\pgref`), je nutné \TeX ovat dokument aspoň dvakrát.

7.5 Odrážky

Jednotlivé myšlenky je občas potřeba vypíchnout odrážkami. Prostředí s odrážkami se vymezuje sekvencemi `\beginitems` a `\enditems`. Uvnitř tohoto prostředí je hvězdička aktivním znakem, který zahajuje odrážky. Prostředí s odrážkami je možné vnořit do sebe. Pomocí `\style<znak>` hned za slovem `\beginitems` je možné vymezit některé z předdefinovaných vzhledů odrážek:

```
\style o % malý puntík
\style 0 % velký puntík $\bullet$ (implicitní volba)
\style - % spojovník
\style n % odrážky číslované 1., 2., 3., ...
\style N % odrážky číslované 1), 2), 3), ...
\style i % odrážky číslované (i), (ii), (iii), (iv), ...
\style I % odrážky číslované I, II, III, IV, ...
\style a % odrážky s písmeny a), b), c), ...
\style A % odrážky s písmeny A), B), C), ...
\style x % malý čtvereček
\style X % velký čtvereček
```

Příklad:

```

\beginitems \style n
* Tady je první myšlenka.
* A tady druhá, která je rozdělena na
  \beginitems \style a
  * podmyšlenku
  * a hned následuje další podmyšlenka,
  * poslední podmyšlenka.
  \enditems
* Tady je třetí myšlenka.
\enditems

```

vytvoří následující výstup:

1. Tady je první myšlenka.
2. A tady druhá, která je rozdělena na
 - a) podmyšlenku
 - b) a hned následuje další podmyšlenka,
 - c) poslední podmyšlenka.
3. Tady je třetí myšlenka.

Chcete-li uvnitř prostředí s odrážkami vytisknout hvězdičku, pište `\char‘*`.

Pomocí `\sdef{item:⟨písmeno⟩}{⟨text⟩}` si můžete dodefinovat vzhled odrážek podle svých představ. Implicitní odrážku lze předefinovat pomocí `\def\normalitem{⟨text⟩}`.

Jednotlivá prostředí s odrážkami se odsazují podle velikosti registru `\iindent`, který je nastaven na hodnotu `\parindent` v době čtení souboru `opmac.tex`. Pokud později změníte `\parindent`, doporučuji na stejnou hodnotu nastavit `\iindent`. Vertikální mezera nad a pod prostředím s odrážkami je řízena makrem `\iiskip`.

7.6 Tvorba automaticky generovaného obsahu

Makro `\maketoc` vytiskne v místě svého použití obsah dokumentu bez nadpisu, jen jednotlivé řádky obsahu. Odsazení jednotlivých řádků je nastaveno na násobky registru `\iindent`. Často je potřeba dokument \TeX ovat vícekrát, než se obsah zjeví a než se čísla stran srovnají správně, protože po prvním zjevení obsahu se mohou stránky posunout jinak.

Titulek k obsahu by neměl být číslovaný a neměl by se zjevit v obsahu, takže jej napíšeme třeba pomocí

```
\nonum\notoc\sec Obsah
```

Titulky kapitol, sekcí a podsekcí zapisuje OPmac pro účely sestavení obsahu do externího souboru `.ref`. Může se stát, že uživatel v těchto textech použije nějaké komplikované makro, které se pak v souboru „rozsype“ do takového stavu, že nejde vzápětí přelit. V takovém případě je potřeba makro zabezpečit proti expanzi při zápisu do souboru pomocí deklarace `\addprotect\makro`. Takto deklarované makro je pak zabezpečené proti expanzi do `.ref` souboru. Například OPmac deklaruje:

```
\addprotect~ \addprotect\TeX \addprotect\thefontsize \addprotect\em
```

a mnoho dalších. Není možné ale předvídat všechno, co může uživatel nacpat do titulku sekce nebo kapitoly. Dostanete-li se tedy do potíží s rozsypaným makrem v `.ref` souboru, pak je třeba makro zabezpečit pomocí `\addprotect`, a také je potřeba před dalším \TeX ováním vymazat `.ref` soubor.

7.7 Barvy, vodoznaky

Makra uvedená v této sekci nastavují barvy jen při přímém vytváření PDF. Takže při výstupu do DVI tato makra neudělají nic. Barvu textu můžete nastavit pomocí přepínačů `\Blue`, `\Red`, `\Brown`, `\Green`, `\Yellow`, `\Cyan`, `\Magenta`, `\White`, `\Grey`, `\LightGrey` a `\Black`. Je potřeba si uvědomit, že tyto přepínače pracují globálně nezávisle na skupině uvnitř boxu i mimo, prostě kdekoli. Barvu jinou než černou je potřeba nakonec vypnout přepínačem `\Black`. Pokud barvu jinou než černou zapnete na některé stránce a text v této barvě přeteče do stránek dalších, při opakovaném \TeX ování se objeví správná barva i na následujících stránkách. Tuto skutečnost si totiž \TeX musí ujasnit prostřednictvím pomocného souboru s odkazy.

Společně s barvou textu tyto přepínače zapínají i barvu větších obdélníkových ploch vytvořených pomocí `\hrule` a `\vrule`. Ve specifikaci formátu PDF je nicméně ještě jeden přepínač barev nezávislý na barvě textu a větších ploch. Jsou to barvy linek, které mají tloušťku menší než 1 bp. Tento „druh barvy“ se přepíná stejnými přepínači `\Blue`, ..., `\Black`, ovšem před tímto přepínačem musí bezprostředně předcházet slovo `\linecolor`. Takže `\linecolor\Red` zapne barvu linek na červenou, ale barva textu zůstane původní. Nebo `\Black` vrátí do normálu barvu textu, zatímco `\linecolor\Black` způsobí, že také linky budou dále černé.

Kromě uvedených barevných přepínačů si můžete „namíchat“ v režimu CMYK i barvy vlastní. Stačí se inspirovat, jak jsou uvedené přepínače definovány:

```
\def\Red{\setcmykcolor{0.0 1.0 1.0 0}}
\def\Brown{\setcmykcolor{0 0.67 0.67 0.5}} ...
```

Uvnitř skupiny, která se nerozpadne do více stránek, je možno před přepnutím barvy použít `\localcolor`. \TeX si v místě tohoto makra zapamatuje aktuální barvu a po ukončení skupiny se k ní vrátí. Následuje příklad¹⁾, který vytvoří **podbarvený text**:

```
\def\podbarvi#1#2#3{\setbox0=\hbox{#3}\leavevmode
  {\localcolor\rlap{#1\strut\vrule width\wd0\#2\box0}}
\podbarvi\Yellow\Brown{Tady je hnědý text na žlutém pozadí.}
```

Pokud nemáte jistotu, že se skupina uzavře na stejné stránce, a chcete ve skupině něco obarvit, je potřeba místo `\localcolor` použít `\longlocalcolor`. Nechcete-li rozlišovat mezi tolika prefixy, můžete použít návod na stránce OPmac triků²⁾ nazvaný „přepínače barev jako přepínače fontů“.

Vodoznakem je míněn šedý text opakující se na každé stránce, který je vytištěn pod obvyklým textem. Například OPmac nabízí makro `\draft`, které způsobí, že každá stránka obsahuje šikmo napsaný veliký šedý nápis DRAFT. Můžete se inspirovat v technické dokumentaci, jak je to uděláno a pozměnit to k obrazu svému.

7.8 Klikací odkazy

Pokud napíšete na začátek dokumentu `\hyperlinks{<color in>}{<color out>}`, pak se v dokumentu při výstupu do PDF stanou klikacími:

- čísla generovaná pomocí `\ref` a `\pgref`,
- čísla kapitol, sekcí, podsekcí a stránek v obsahu,
- čísla nebo značky generované pomocí `\cite` (odkazy na literaturu),
- texty tištěné pomocí makra `\url` nebo `\link`.

¹⁾ K pochopení tohoto příkladu je nejspíš potřebné nejprve přečíst kapitoly 8 a 9.

²⁾ <http://petr.olsak.net/opmac-tricks.html#coluser>

Poslední z uvedených odkazů je externí a bude mít barvu $\langle color\ out\rangle$, zatímco ostatní čísla jsou interními odkazy a budou mít barvu $\langle color\ in\rangle$. Příklady:

```
\hyperlinks{\Blue}{\Green}  % vnitřní odkazy modré, URL zelené
\hyperlinks{}{}             % aktivace odkazů bez nastavení barev
```

Je možné zobrazit rámečky ohraničující aktivní plochu pro klikání. Tyto rámečky jsou viditelné jen v PDF prohlížeči, při tisku na tiskárně se nezobrazují. Stačí těmto rámečkům „namíchat“ barvu (tentokrát RGB) a definovat některé ze sekvencí `\pgborder`, `\tocborder`, `\citeborder`, `\refborder` a `\urlborder`. První část jména řídící sekvence určuje, jakých odkazů se to týká. Naříklad:

```
\def\tocborder{1 0 0} % odkazy v obsahu vlevo budou mít červený rámeček
\def\pgborder{0 1 0}  % odkazy na stránky budou mít zelený rámeček
\def\citeborder{0 0 1} % odkazy na publikace budou mít modrý rámeček
```

Implicitně tato makra nejsou definována, což znamená, že rámečky nevzniknou.

Manuálně je možno vytvořit cíl odkazu makrem `\dest` [$\langle typ\rangle$]: $\langle lejblík\rangle$ a klikací text makrem `\link` [$\langle typ\rangle$]: $\langle lejblík\rangle$ [$\langle color\rangle$]{ $\langle text\rangle$ }. Parametr $\langle typ\rangle$ je typ odkazu (toc, pg, cite, ref nebo další).

Makro `\url` vytiskne odkaz do internetu. Například `\url{http://petr.olsak.net}` vytvoří `http://petr.olsak.net`. Text je psán strojopisem a může se lámat do řádků za lomítky. Je-li nastaveno `\hyperlinks`, stává se tento text aktivním vnějším odkazem. Vyskytují-li se v argumentu `\url` znaky %, \, #, \$, { a }, je třeba použít %, \%, \#, \\$, \{ a \}. Ostatní speciální znaky ~, _, ^, & lze napsat do parametru `\url` přímo. Dále je možno do parametru `\url` napsat | k označení místa, kde je dovoleno zlomit řádek.

Libovolný text odkazovaný na web lze vložit pomocí `\ulink` [$\langle URL\rangle$]{ $\langle text\rangle$ }. Například `\ulink{http://petr.olsak.net/opmac.html}{stránky OPmac}` vytiskne text „stránky OPmac“, který je při zapnutém `\hyperlinks` aktivním odkazem.

Dokument může mít svůj přehledový obsah umístěný v levé záložce PDF prohlížeče tak, že klikáním na něj se přechází v dokumentu na požadované místo. Ve specifikaci PDF se tomu říká „outlines“. Makro, které uvedenou věc zařídí, se jmenuje `\outlines`{ $\langle úroveň\rangle$ }. Záložky budou implicitně rozevřeny do $\langle úroveň\rangle$ včetně. Takže třeba při $\langle úroveň\rangle=0$ jsou vidět jen úrovně kapitol. Texty v záložkách používají systémový font, který nemusí správně zobrazit česká a slovenská písmena. Proto OPmac konvertuje texty do záložek tak, že tam jsou pro jistotu bez hacku a carek. Chcete-li vypnout tuto konverzi, napište `\def\toasciidata{}`. Chcete-li akcenty zachovat, načtěte makro soubor `pdfuni.tex`.

Samotný řádek do záložek vložíte makrem `\insertoutline`{ $\langle text\rangle$ }. Text v tomto případě nepodléhá konverzi. V sazbě se neobjeví nic, jen se toto místo stane cílem, kam odkaz ze záložky směřuje. Obsah se do záložek vloží celý během činnosti makra `\outlines`, takže další řádky vložené pomocí `\insertoutline` tomuto obsahu předcházejí nebo následují podle toho, zda předcházejí nebo následují místu, kde je použito `\outlines`.

7.9 Verbatim texty

Vytisknout část textu verbatim „tak jak je“ bez interpretace speciálních znaků lze v prostředí vymezeném makry `\begtt` a `\endtt`. Příklad:

```
\begtt
Tady je vše
      napsáno bez  interpretace speciálních znaků, jakými
      jsou mezera, %, $, \, ~, ^, _, {, }, #, &.
\endtt
```

Ve výstupu se objeví:

Tady je vše
napsáno bez interpretace speciálních znaků, jakými
jsou mezera, %, \$, \, ~, ^, _, {, }, #, &.

Není-li za `\endtt` prázdný řádek, nemá následující odstavec výchozí odsazení.

Je-li před zahájením `\begtt` nastaven registr `\ttline` na nezápornou hodnotu, bude makro číslovat řádky. První řádek má číslo `\ttline+1` a po práci makra se registr `\ttline` posune na číslo posledního vytištěného řádku. Takže v dalším prostředí `\begtt ... \endtt` číslování pokračuje tam, kde přestalo. Implicitně je `\ttline=-1`, takže číslování neprobíhá.

Levé odsazení každého řádku v `\begtt ... \endtt` je nastaveno na `\ttindent`. Tento registr má výchozí hodnotu rovnou `\parindent` (v době čtení souboru `opmac.tex`). Vertikální mezera nad a pod verbatim výpisem je vložena makrem `\ttskip`.

Makro `\begtt` zahájí skupinu a v ní nastaví všem speciálním znakům plainTeXu kategorii 12. Pak spustí makro `\tthook`, které je implicitně prázdné. V něm je možno nastavit další kategorie znaků podle potřeby. Definici aktivních znaků je potřeba udělat pomocí `\adef⟨znak⟩{⟨text⟩}`. Normální `\def` nefunguje, důvod je vysvětlen v TBN na straně 26. Příklad:

```
\def\tthook{\adef!{?}}
\begtt
Nyní se každý vykřičník promění v otazník. Že nevěříte? Vyzkoušejte!
\endtt
```

Jednou definovaný `\tthook` funguje ve všech verbatim výpisech, dokud jej nepředefinujete jinak. Tipy:

```
\def\tthook{\typosize[9/11]} % jiná velikost verbatim výpisů
\def\tthook{\ttline=0} % všechny výpisy číslovány od jedničky
\def\tthook{\adef{ }{\char'\ }} % místo mezer budou vaničky
```

Verbatim lze tisknout i v řádku uvnitř odstavce. Pomocí `\activettchar⟨znak⟩` si uživatel zvolí znak, který bude aktivní a bude zahajovat i končit verbatim výpisy uvnitř odstavce. Verbatim výpis se v odstavci nikdy nerozlomí (je v boxu). Autor makra `OPmac` obvykle nastavuje `\activettchar`, takže pak může psát třeba toto:

Je-li před zahájením `"\begtt"` nastaven registr `"\ttline"` na nezápornou...

Znak nastavený pomocí `\activettchar` má lokální platnost a ruší se také pozdějším nastavením `\activettchar` na jinou hodnotu. Při zahájení každého řádkového verbatim výpisu se spustí makro `\intthook`, které je implicitně prázdné. **Upozornění:** deklaraci `\activettchar⟨znak⟩` vložte až po přečtení všech makrosouborů. Důvodem je, že `\activettchar` nastavuje `⟨znak⟩` jako aktivní, což může při čtení souborů maker vadit.

Verbatim výpisy je možné tisknout z externího souboru. Například

```
\verbinput (12-42) program.c
```

vytiskne ve stejné úpravě, jako při použití `\begtt, ... \endtt`, řádky 12 až 42 ze souboru `program.c`. Parametry v kulaté závorce mohou vypadat také takto:

```
\verbinput (-60) program.c % výpis od začátku souboru do řádku 60
\verbinput (61-) program.c % výpis od řádku 61 do konce souboru
\verbinput (-) program.c % výpis celého souboru
\verbinput (70+10) program.c % výpis od řádku 70, tiskne 10 řádků
```

V dalších ukázkách OPmac čte od řádku, který následuje za naposledy přečteným řádkem souboru z předchozího volání `\verbinput`. Je-li soubor čten poprvé, začíná číst prvním řádkem. Tento prvně čtený řádek je označen v komentářích jako `n`.

```
\verbinput (+10) program.c % výpis deseti řádků od řádku n
\verbinput (+) program.c % výpis od řádku n do konce souboru
\vebrinput (-5+7) program.c % vynechá 5 řádků, od n+5 tiskne dalších 7
\verbinput (-3+) program.c % vynechá 3 řádky, tiskne do konce souboru
```

Narazí-li čtení na konec souboru dřív, než je vytištěno vše, co si žádá uživatel, prepis souboru je ukončen a žádná chyba se nezjeví.

Výpisy makrem `\verbinput` jsou ovlivněny registrem `\ttindent` a makrem `\tthook` stejně, jako prostředí `\begtt...\endtt`. Při `\ttline<-1` se netisknou čísla řádků. Je-li `\ttline=-1`, čísluje se podle řádků souboru. Při nezáporném `\ttline` se řádky číslovají od `\ttline+1`.

7.10 Tabulky

L^AT_EXoví uživatelé jsou zvyklí při vymezení pravidel zarovnávání v tabulce používat deklarace typu `{cclr}`. Každé písmeno vymezí jeden sloupec v tabulce, přitom písmeno `c` znamená centrováný sloupec, `l` je sloupec zarovnaný doleva a `r` sloupec zarovnaný doprava. Podobnou možnost deklarace jednoduchých tabulek nabízí OPmac v makru `\table{<deklarace>}{<data>}`. Příklad:

```
\table{||lc|r||}{\crl
  měsíc      & zboží      & cena\hfil \crl\ \tskip.2em
  leden      & nořas      & 14 kKč   \cr
  únor       & skejt      & 2 kKč    \cr
  červenec   & jachtička & 3,4 MKč  \crl}
```

Uvedený příklad povede na následující výsledek:

měsíc	zboží	cena
leden	nořas	14 kKč
únor	skejt	2 kKč
červenec	jachtička	3,4 MKč

Ve skutečnosti výsledek nebude uprostřed řádku, ale tam, kam `\table` napíšete. Kromě písmen `c`, `l`, `r` se v `<deklaraci>` mohou objevit znaky „svislítko“, které vymezují svislou čáru mezi sloupci.

V datové části musí být tolik sloupců, kolik jich bylo deklarováno. Jsou odděleny znakem `&` nebo symbolem pro konec řádku `\cr`. Z toho vyplývá, že na každém řádku musí být v datové části o jeden znak `&` méně, než je počet sloupců. Nedodržíte-li toto pravidlo, T_EX se pomstí chybovou hláškou

```
! Extra alignment tab has been changed to \cr
```

nebo vytvoří nedomrlou tabulku. Místo symbolu pro konec řádku `\cr` je možno použít `\crl` (přidá jednoduchou vodorovnou čáru) nebo `\crl1` (přidá dvojitou vodorovnou čáru), `\crl1` (přidá vodorovnou čáru přerušenou svislými dvojitými linkami, tj. interrupted) nebo `\crl11` (přidá dvojitou vodorovnou čáru přerušenou svislými dvojitými linkami). Těsně za `\cr`, `\crl` atd. může následovat `\tskip<velikost>`, což vytvoří vertikální mezeru dané velikosti, přitom se nepřeruší svislé čáry v tabulce.

Za povšimnutí stojí, že v ukázce u slova „cena“ je připojeno `\hfil`, což vloží pružnou mezeru vpravo od položky. Protože sloupec `r` obsahuje implicitní stejnou pružnou mezeru vlevo, je slovo „cena“ centrováno, zatímco ostatní údaje ve sloupci jsou zarovnány napravo. O příkazu `\hfil` je více řečeno v sekci 9.1.

Makro `\table` pracuje s předdefinovanými hodnotami, které můžete změnit, pokud chcete dosáhnout jiný vzhled tabulky:

```
\def\tabiteml{\enspace} % co vkládá vlevo každé datové položky
\def\tabitemr{\enspace} % co vkládá vpravo každé datové položky
\def\tabstrut{\strut}    % podpěra vymezující výšku řádků
\def\vvkern{1pt}        % velikost mezery mezi dvojitou svislou linkou
\def\hhkern{1pt}        % velikost mezery mezi dvojitou vodorovnou linkou
```

Vyzkoušejte si tabulku po `\def\tabiteml{\enspace}\def\tabitemr{\enspace}`. Sloupce budete mít na sebe nalepeny bez mezer. Příklad definice `\tabstrut`:

```
\def\tabstrut{\vrule height11pt depth3pt width0pt}
```

Tento příklad vymezuje v tabulce vzdálenost mezi účarím 14 pt, z toho 11 pt je rezervováno pro přetahy nad účarím a 3 pt pro přetahy pod účarím. Vyskytne-li se větší písmeno, zvětší to v daném místě řádkování.

OPmac definuje `\strut` v návaznosti na zvoleném řádkování (podle parametru makra `\typosize`) zhruba takto:

```
\def\strut{\vrule height.709\baselineskip depth.291\baselineskip width0pt}
```

Tip: vyzkoušejte si `\def\tabiteml{\enspace}\def\tabitemr{\enspace}`. Ty dolary způsobí, že každá datová položka bude zpracována v matematickém módu. Makro `\table` se nyní podobá L^AT_EXovému prostředí `array`.

Makro `\frame{<text>}` vytvoří rámeček kolem `<textu>` s vnitřními okraji o velikostech `\vvkern` a `\hhkern`. Například `\frame{ahoj}` vytvoří `[ahoj]`. Povšimněte si, že účarí rámovaného textu zůstalo nezměněno. Pokud chcete mít tabulku s dvojitými čarami, je výhodné ji vytvořit po stranách a nahoře a dole s jednoduchými čarami a celou ji zabalit do `\frame`:

```
\frame{\table{\c|l|l|r|c|}{\cr
\multispan4\vrule\hss\bf Nadpis\hss\vrule\tabstrut\cr
\noalign{\kern\hhkern}\crli
první & druhý & třetí & čtvrtý \crlli
sedmý & osmý & devátý & desátý \crli}}
```

Nadpis			
první	druhý	třetí	čtvrtý
sedmý	osmý	devátý	desátý

V ukázce je použito makro plainT_EXu `\multispan{počet}`, které vytvoří položku v tabulce napříč stanoveného počtu sloupců. Při použití tohoto makra je nutné vynechat `<počet>-1` oddělovačů `&`. Konečně příkaz `\noalign{<sazba>}` vloží sazbu mezi řádky tabulky a musí být uveden těsně za `\cr`, `\crli` atd.

Kromě předdefinovaných znaků `c`, `l`, `r`, `i` se může v `<deklaraci>` objevit libovolný další symbol, stačí připravit `\def\tabdeclare{<symbol>{\vlevo##\vpravo}}`. V technické dokumentaci je příklad deklarace položky `P`, která se při delším textu láme do více řádků.

Tloušťka všech čar je v T_EXu implicitně 0,4 pt. OPmac umožňuje tuto implicitní tloušťku nastavit jinak pomocí `\rulewidth=<šířka>`, například `\rulewidth=1.5pt`.

Další příklad použití makra `\table` najdete v sekci 7.4. Významnou inspiraci i k poměrně složitým tabulkám lze také nalézt na stránkách OPmac triků¹⁾. Pokud potřebujete vytvořit ještě komplikovanější tabulky, nezbude než prostudovat TBN, kapitolu čtvrtou.

¹⁾ <http://petr.olsak.net/opmac-tricks.html>

7.11 Vkládání obrázků

Makro `\inspic<jméno>.<přípona><mezera>` vloží obrázek. Obrázek bude mít šířku danou registrem `\picw`¹⁾, pokud je tento registr nastaven na nenulovou hodnotu. Implicitní hodnota registru je 0pt, což znamená, že bude obrázek vložen ve své přirozené velikosti. Analogicky lze nastavit výšku obrázku registrem `\picheight`²⁾. Přípony souboru s obrázkem mohou být `png`, `jpg`, `jbig2`, `pdf`.

Obrázek je vyhledán v adresáři `\picdir`. Toto makro je implicitně prázdné, tj. obrázek je vyhledán v aktuálním adresáři.

O umístění obrázku v sazbě se musíte postarat vlastními prostředky. Například:

```
\picw=.3\hsize \centerline{\inspic hodiny.jpg }  
\nobreak\medskip  
\caption/f Hodiny na brněnském náměstí Svobody
```

Uvedený příklad vytvoří:



Obrázek 7.11.1 Hodiny na brněnském náměstí Svobody

Makro `\inspic` pracuje jen při výstupu do PDF. Pokud máte nastaven výstup do DVI, můžete použít makro `epsf.tex`. Vzhledem k omezeným možnostem (obrázek jen ve formátu EPS) není tento způsob práce s obrázky v makru OPmac podporován.

Je-li to vzhledem k charakteru obrázku vhodné (například grafy, obrázek sestavený z čar), doporučuje se přednostně použít vektorový formát obrázku, tedy například programem Inkscape³⁾ vytvořit PDF. Naopak pro fotografie se hodí bitmapový formát. I v tomto případě je někdy dobré přemýšlet a nepřehánět to s množstvím pixelů v obrázku. I obrázek pořízený v nejvyšším rozlišení moderní zrcadlovky nakonec snadno zmenšíte pomocí `\picw=1cm`, ale zkuste se pak podívat na velikost výsledného PDF souboru. Někdy je tedy vhodné obrázky před použitím přerastovat, například v program Gimp⁴⁾. Na fotografie stačí obvykle rozlišení 150 dpi počítané dle cílového rozměru obrázku.

Pokud obrázek nebo tabulka dělá potíže při stránkovém zlomu, je třeba je obklopit makry `\midinsert` a `\endinsert` (včetně popisku). To způsobí, že obrázek nebo tabulka odpluje na začátek následující strany. Podrobněji viz sekci 10.2.

Chcete-li obrázek nebo tabulku otočit, lze použít lineární transformaci sazby, což popisuje sekce 11.6.

¹⁾ Existuje také alternativní název tohoto registru `\picwidth`. Ovšem psát `\picw` je kratší.

²⁾ Zkratka pro tento registr není zavedena. Je vhodné nastavit jen jeden z registů `\picwidth` nebo `\picheight`, aby nedošlo k deformaci obrázku.

³⁾ <http://inkscape.org/>

⁴⁾ <http://www.gimp.org/>

Makro `\inspic` není rozumné použít při opakovaném vkládání stejného obrázku v dokumentu (opakující se grafika na každé straně nebo obrázek jako odrážka ve výčtu položek). V takovém případě je vhodnější natáhnout obrázek do PDF dokumentu jen jednou pdfT_EXovým příkazem `\pdfximage` a dále opakovat jeho zobrazení na různých místech dokumentu pomocí `\pdfrefximage`. Viz též sekci 11.7.

7.12 Poznámky pod čarou a na okraji

Poznámku pod čarou vytvoříte pomocí `\fnote{<text>}`. V místě tohoto zápisu v textu se objeví automaticky generovaná značka a pod čarou dole na stránce je tato značka zopakována a vedle ní je `<text>`.

Značka je implicitně definovaná jako číslo v exponentu následované závorkou. Číslování poznámek je na každé stránce započato jedničkou¹). Čísla jsou vygenerována správně až po opakovaném T_EXování. Při prvním zpracování jsou místo čísel otazníky.

Implicitní značkování je možné změnit předdefinováním makra `\thefnote`. Například:²)

```
\def\thefnote{\ifcase\locfnum\or
* \or** \or*** \or${\dag}$ \or${\ddag}$ \or${\dag}\dag$ \fi}
```

Po použití tohoto makra bude první poznámka mít hvězdičku, druhá dvě hvězdičky atd. Uvedená definice předpokládá, že na jedné stránce nebudete mít více než 6 poznámek.

Makro `\fnote` je možné zapsat jen v běžném textu odstavce, nikoli v boxu (například v tabulce). Chcete-li odkazovat třeba z tabulky, je nutné v tabulce vytvořit jen značky a mimo tabulku (ovšem tak, aby to neuteklo na jinou stránku) zapíšete texty poznámek. K vytvoření značky použijte `\fnotemark<číslo>` a text (bez značky) vytvoří `\fnotetext{<text>}`. Příklad:

```
{\typoscale[/1200]\table{||lc|rl||}{\crl
měsíc      & zboží                & cena\hfil \crl\l \tskip.2em
leden      & noťas\fnotemark1            & 14 Kč      \cr
únor       & skejt\fnotemark2             & 2 Kč       \cr
červenec   & jachtička\fnotemark3         & 3,4 MKč    \crl}}
\par \fnotetext{notebook}\fnotetext{skateboard}\fnotetext{jachta}
```

Čísla za slovy `\fnotemark` je třeba psát od jedné v každé tabulce/boxu. Nemusejí souviset se skutečným číslem poznámky. Například, je-li na stejné stránce nad tabulkou z ukázky normální `\fnote`, bude mít vytištěno číslo 1, odkazy v tabulce budou mít čísla 2, 3, 4 a případná další poznámka pod tabulkou na stejné stránce obdrží číslo 5.

Poznámku na okraji stránky vytvoříte pomocí řídicí sekvence `\mnote{<text>}`. Poznámka je vlevo (na pravou zarážku) na sudé stránce a je vpravo (na levou zarážku) na liché stránce. Tuto vlastnost mají poznámky až po opakovaném T_EXování. Při prvním T_EXování jsou všechny poznámky vpravo. Chcete-li mít poznámky i při opakovaném T_EXování jen vpravo nebo jen vlevo, pište do úvodu dokumentu `\fixmnotes\right` nebo `\fixmnotes\left`.

Řídicí sekvenci `\mnote{<text>}` můžete napsat do odstavce nebo před odstavce. S odstavcem samotným to nic neudělá. Řádek odstavce, kde je `\mnote` vložena jako neviditelná značka, je na stejné úrovni, jako první řádek textu poznámky.

¹) Toto chování se dá změnit vložením `\runningfnotes` na začátek dokumentu. V takovém případě se poznámky číslovají od jedné v celém dokumentu. Další možnosti číslování jsou uvedeny v technické dokumentaci.

²) Příkaz `\ifcase` větví zpracování podle hodnoty makra `\locfnum`. Toto makro obsahuje číslo poznámky na stránce.

Text poznámky je od sazby odsazen o `\mnoteindent` a maximální šířka poznámky je `\mnotesize`. Text poznámky se rozlomí do více řádků, aby nepřesáhl `\mnotesize`.

Není ošetřen případ, kdy je `\mnote` víceřádková a je umístěna na úroveň například posledního řádku strany. Pak text poznámky přechuhuje poněkud dolů ze strany. Nebo se poznámky mohou překrývat. Je tedy nutné `\mnote` použít jen na velmi krátké poznámky a případně si tento jev pohlídat a ošetřit při definitivní sazbě manuálně. Pro manuální ošetření vertikální polohy poznámek slouží `\mnoteskip`, což je registr, který udává, o kolik se má následující poznámka (a jen ta) posunout nahoru. Při záporné hodnotě se posune dolů. Například `\mnoteskip=2\baselineskip \mnote{<text>}` posune poznámku o dva řádky výše.

7.13 Bibliografické údaje

Odkazy. Pomocí `\cite[<lejblík>]` nebo `\cite[<lejblík1>,<lejblík2>,<lejblík3>]` atd. vytvoříme v textu odkazy na položky v seznamu literatury. V seznamu literatury je třeba uvést záznamy, které mají odkazované lejblíky. Tyto záznamy dostanou v seznamu automaticky vygenerovaná čísla a sekvence `\cite` se pak promění na číselné odkazy, například [27] nebo [18, 42, 24] atd. Řady čísel v jednom `\cite` se seřadí podle velikosti, když je v úvodní deklaraci dokumentu napsáno `\sortcitations` a tato čísla [1, 2, 3, 5, 6] se promění v intervaly [1–3, 5–6] při `\shortcitations`.

Při `\nonumcitations` se odkazy nepřevádějí na čísla. K tomu je potřeba použít navazující Bib_TE_Xový styl (např. `alpha`, `apalike`) nebo rozšířenou formu makra `\bib`, viz níže. Odkazy vypadají při stylu `alpha` třeba takto [Nov08] a při stylu `apalike` takto [Novák, 2008].

Příkaz `\rcite[<lejblíky>]` funguje jako `\cite[<lejblíky>]`, ale kolem odkazů nejsou přidány závorky. Možnost využití: `[\rcite[novak08],~s.~13]` vytvoří například odkaz [17, s. 13]. Závorky kolem musíte napsat sami.

Příkaz `\ecite[<lejblík>]{<text>}` vytiskne pouhý `<text>`, který se chová jako odkaz na literaturu. Příklad

Z[~]výsledků Nováka `[\ecite[novak08]{2008},~s.~13]` plyne...

vytiskne: Z výsledků Nováka [2008, s. 13] plyne... Přitom `novak08` je registrován do seznamu citovaných položek a třeba při `\hyperlinks` bude číslo 2008 prolinkováno s odpovídající položkou v seznamu literatury.

Příklady redefinice `\cite` pro alternativní formátování odkazů:

```
\def\cite[#1]{(\rcite[#1])}      % \cite[lejblík] vytvoří (27)
\def\cite[#1]{${\rcite[#1]}$}    % \cite[lejblík] vytvoří ^{27}
```

Seznam literatury je možné vložit do dokumentu čtyřmi různými způsoby:

- Manuálně: pomocí jednotlivých položek `\bib[<lejblík>]` přímo v dokumentu.
- S využitím Bib_TE_X¹⁾ makrem `\usebibtex{<bib-báze>}{<bst-styl>}`.
- Využitím jednou vygenerované databáze makrem `\usebbl/<typ> <bbl-báze>`.
- Přímým čtením `.bib` databáze makrem `\usebib/<typ> (<style>) <bib-báze>` bez využití Bib_TE_X.

Jednotlivé způsoby jsou níže probrány podrobněji.

Manuálně vložený seznam literatury v dokumentu vypadá například takto:

¹⁾ Bib_TE_X je program, který čte databáze s bibliografickými údaji a na základě použitých `\cite` z těchto databází vybírá jen potřebné údaje, třídí je a podle `.bst` stylu je připravuje pro načtení do dokumentu.

`\bib[tnb]` P. Olšák. `{\it\TeX{}}book naruby.` 468~s. Brno: Konvoj, 2001.

`\bib[tst]` P. Olšák. `{\it Typografický systém \TeX{}}`
300~s. Brno: Konvoj, 2000.

Výše uvedená ukázka dá následující výstup:

[1] P. Olšák. *TEXbook naruby*. 468 s. Brno: Konvoj, 2001.

[2] P. Olšák. *Typografický systém TEX*. 300 s. Brno: Konvoj, 2000.

Je možný i rozšířený způsob zápisu `\bib` [*lejblik*] = {*značka*} *text záznamu*. Údaj *značka* se použije do odkazů při zapnutém `\nonumcitations`. Například:

`\bib [tnb] = {Olšák, 2001}`

`OLŠÁK, P. {\it\TeX{}}book naruby.` 468~s. Brno: Konvoj, 2001.

Využití BibTEXu. Předpokládá se, že uživatel disponuje souborem *bib-báze*.bib, ve kterém jsou nashromážděny bibliografické údaje ve formátu, v jakém je čte program BibTEX. V TEXové distribuci jistě najdete nějaký .bib soubor, tak se do něj podívejte. Lejblikem je první údaj u každého bibliografického záznamu. Soubor *bib-báze*.bib by měl obsahovat bibliografické údaje, které jsou nadmnožinou toho, co potřebuje uživatel vypsát ve svém dokumentu. Na místo, kde budete chtít vypsát seznam literatury, vložte `\usebibtex{bib-báze}{bst-styl}`. Parametr *bib-báze* je jméno souboru bez přípony .bib, ve kterém jsou připraveny bibliografické záznamy. Parametr *bst-styl* je jméno stylového souboru bez přípony .bst, který použije BibTEX pro konverzi ze zdroje *bib-báze*.bib do výstupu *dokument*.bbl. Tento výstup pak bude makrem `\usebibtex` přečten a vložen do dokumentu.

Typicky používané *bst-styly* jsou *plain*, *alpha*, *apalike*, *ieeetr*, *unsrt*. Styl *alpha* způsobí, že se místo čísel začnou zjevovat v dokumentu zkratky, a to jednak v seznamu literatury a jednak v místě výskytu `\cite`. V tom případě pochopitelně `\shortcitations` nefunguje a pokud byste se o něj pokusili, tak makro havaruje. Na internetu existují desítky, možná stovky dalších .bst stylů.

Při prvním zpracování dokumentu TEXem makro `\usebibtex` připraví vstupní pokyny pro BibTEX do souboru *dokument*.aux a zjistí, že soubor *dokument*.bbl zatím neexistuje. To dá najevo na terminálu:

WARNING: .bbl file doesn't exist. Use the `“\bibtex {dokument}”` command.

Přejděte tedy na příkazový řádek a napište `\bibtex {dokument}`. Tím se spustí program BibTEX, který přečte ze souboru *dokument*.aux vstupní pokyny (kterou otevřít .bib databázi, který .bst styl a jaké lejbliky jsou požadovány) a na základě toho vygeneruje soubor *dokument*.bbl, který obsahuje výběr jen těch záznamů, které byly uživatelem citovány pomocí `\cite`. Soubor *dokument*.bbl je navíc zkonvertovaný z .bib formátu do formátu čitelného TEXem. Tato konverze je řízena stylem .bst.

Když znovu TEXujete dokument, makro `\usebibtex` v tomto případě shledá, že soubor *dokument*.bbl existuje, načte jej a vytvoří seznam literatury. Seznam obsahuje jen citované položky. Druhé spuštění TEXu obvykle nestačí, protože `\cite` jsou typicky dopřednými referencemi, takže zatím nemají ponětí o přiřazení čísel k *lejblikům* v seznamu literatury. To se dozvědí až v místě použití `\usebibtex`, což je typicky na konci dokumentu. Takže teprve třetí TEXování dá vše do pořádku.

Seznam literatury obsahuje po použití BibTEXu jen citovaná dílka. Pokud chcete do seznamu zařadit další položky, které nejsou v textu explicitně odkazovány makrem `\cite`, použijte `\nocite{lejblik}`. Toto makro dá BibTEXu pokyn, aby do seznamu zahrnul i položku s *lejblikem*, ale v místě použití tohoto makra se nevytiskne nic. Konečně

pomocí `\nocite[*]` dáváme Bib_{TEX}u vzkaz, že chceme mít v seznamu literatury celou `.bib` databázi.

Zdroj bibliografických záznamů může být ve více `.bib` souborech. V takovém případě stačí jejich názvy oddělit čárkou: `\usebibtex{<bib-báze1>,<bib-báze2>}{<bst-styl>}`.

Někdy se stane, že autoři `.bib` databází nebo `.bst` stylů neopustili při tvorbě těchto souborů L_AT_EXový způsob myšlení a občas jim uklouzne nějaká L_AT_EXová konstrukce z prstů až do počítače. Odtud se dostane do čteného `.bbl` souboru a náš plain_{TEX} si s tím nebude vědět rady. K tomu slouží seznam `\bibtexhook`, kde můžete uvést definice těchto L_AT_EXových konstrukcí. Tyto definice budou mít lokální platnost jen při čtení `.bbl` souboru. Například:¹⁾

```
\def\bibtexhook{\def\emph##1{\em##1}}\def\frac##1##2{{##1\over##2}}
```

Využití jednou vygenerované databáze. Tvorba seznamů literatury Bib_{TEX}em má jistou nevýhodu. Pokud později do dokumentu vložíte další `\cite[<lejblik>]`, musíte veškerou anabázi s Bib_{TEX}em provést znovu. A protože v současné době probíhá inflace odborných publikací způsobená tím, že se podle kvanta publikací a citací daňový poplatník rozhodl odměňovat vědce, je každé zjednodušení práce s bibliografickými záznamy přínosné. Makro OPmac navrhuje řešení, při kterém stačí použít Bib_{TEX} pro mnoho nových článků jen jednou.

1. Vytvořte si zvláštní dokument `<mojebáze>.tex`, do kterého napíšete:

```
\input opmac \genbbl{<bib-báze>}{<bst-styl>} \end
```

2. Po _{TEX}ování dokumentu `<mojebáze>.tex` spusťte `bibtex <mojebáze>`. Tím se vytvoří soubor `<mojebáze>.bbl`.
3. Zpracujte _{TEX}em soubor `<mojebáze>.tex` ještě jednou. Vytvoří se seznam veškeré literatury, který byl v souboru `<bib-báze>.bib`, přitom každá položka je označena svým `<lejblikem>`. Vytiskněte si tento výstup a dejte si jej na nástěnku.
4. Uložte soubor `<mojebáze>.bbl` někam, kde jej umí přečíst _{TEX} bez ohledu na to, v kterém pracujete adresáři.
5. Přejděte k editaci svého dokumentu, pište `\cite` nebo `\nocite` podle potřeby a v místě seznamu literatury dejte sekvenci `\usebbl/<typ> <mojebáze>`. Údaj `<typ>` má tyto možnosti:

```
\usebbl/a <mojebáze> % vypsát kompletně celou <mojebáze> (a=all),
\usebbl/b <mojebáze> % jen \(\no)cite údaje řadit dle <mojebáze> (b=base),
\usebbl/c <mojebáze> % jen \(\no)cite řadit podle pořadí citace (c=cite).
```

Kroky 2 až 4 je nutno opakovat pouze tehdy, když budete chtít přidat do `<mojebáze>.bbl` další údaj, tj. po upgradu souboru `<bib-báze>.bib`. Prudí-li různí odběratelé vaší vědecké činnosti požadavky na různé `<bst-styly>`, stačí si vygenerovat podle různých stylů různé soubory typu `mybbl-plain.bbl`, `mybbl-ieeeetr.bbl`.

Přímé čtení .bib databáze je možné po `\input opmac-bib.tex`. Tato přídatná makra navíc používají externí balíček na čtení `.bib` souborů `librarian.tex` od Paula Isamberta. Užití je podobné jako při `\usebbl`:

```
\usebib/c (<style>) <bib-báze> % řadit podle pořadí citace (c=cite),
\usebib/s (<style>) <bib-báze> % řadit podle klíče ve stylu (s=style).
```

Zde `<bib-báze>` je jeden nebo více `.bib` souborů oddělených čárkou bez mezery a bez přípony. Z nich se vyberou jen záznamy označené v dokumentu pomocí `\cite` nebo `\nocite`.

¹⁾ V ukázce jsou zdvojeny znaky #. Důvod je popsán v sekci 8.2.

Parametr `<style>` udává část jména souboru `opmac-bib-<style>.tex`, ve kterém je specifikace formátování položek. Součástí balíčku jsou styly `simple` a `iso690`.

Formátování seznamu literatury je řízeno makrem `\printbib`, které je vloženo na začátek každé položky v seznamu. Implicitně makro tiskne čísla položek do hranatých závorek a při použití `\nonumcitations` předsadí první řádek položky a nepřidává nic. Makro může využít `\the\bibnum` pro tisk čísla nebo `\the\bibmark` pro tisk značky (při `\nonumcitations`). Příklady:

```
% Číslování položek bez hranatých závorek:
\def\printbib{\hangindent=\parindent \indent \llap{\the\bibnum. }}
```

```
% Tisk zkratk při použití bibTeXového stylu alpha a \nonumcitations:
\def\printbib{\hangindent=\parindent \noindent [\the\bibmark]\quad}
```

Další příklady (třeba jak T_EX změří šířku největšího čísla a podle toho vypočítá odsazení celého seznamu) jsou uvedeny na stránce OPmac triků¹).

7.14 Sestavení rejstříku

Makro pro zanášení slov do rejstříku je navrženo s ohledem na optimalizaci počtu úhozů na klávesnici. Autor už napsal své dílo, má daný termín odevzdání a nyní ho čeká úmorná práce vyhledávání slov v textu, která by měla přijít do rejstříku, a jejich vyznačování. Je třeba mu tuto práci co nejvíce usnadnit.

Pro zanesení slova do rejstříku slouží makro `\ii`. Je to zkratka za „insert to index“. Jeho parametr je `<slovo>` bez mezery ukončené mezerou (obecnější tvar parametru uvedeme později). Toto slovo se přepíše do rejstříku, ve kterém jsou všechna takto deklarovaná slova seřazena podle abecedy a jsou k nim připojena čísla stránek, na kterých bylo použito odpovídající makro `\ii<slovo>`. Příklad:

Tady mluvím o `\ii apelativum` apelativu, které provokovalo moji zvědavost.

Makro `\ii<slovo>` viditelně neudělá v sazbě nic. Přilepí se na následující slovo (v našem příkladě slovo „apelativu“) jako skrytá značka. Číslo strany, kde se ta značka objeví, bude v rejstříku vedle slova „apelativum“.

Je-li `\ii` zapsáno ve vertikálním módu, zahájí se v daném místě odstavec, aby se mohla neviditelná značka z `\ii` nalepit na následující slovo. Pokud si to z nějakých důvodů nepřejete, použijte interní variantu makra `\iiindex{<slovo>}`, která nezahajuje odstavec.

Pokud se v rejstříku má objevit stejné slovo jako v textu, není nutno je psát dvakrát. Stačí použít makro `\iid` (zkratka za „\ii double“):

Hlavní zásady jsou `\iid nestrannost`, `\iid pravdomluvnost` a `\iid odvaha`.

To povede ke stejnému výsledku jako

```
Hlavní zásady jsou \ii nestrannost nestrannost,
\ii pravdomluvnost pravdomluvnost a \ii odvaha odvaha.
```

Povšimněte si, že čárky a tečky jsou odstraněny od dublovaného slova, protože mezera je ukončovací znak parametru `\iid`. Do textu se mezera vrátí právě tehdy, když nenásleduje tečka nebo čárka. V našem příkladě před spojkou „a“ mezera ve výsledku je, ale před tečkou nebo čárkou mezera není.

Vlastnosti makra `\iid` jsou tímto popsány zcela. Vraťme se k makru `\ii`, které poskytuje další možnosti.

¹) <http://petr.olsak.net/opmac-tricks.html>

Parametr `\ii` je vždy ukončen mezerou. Může obsahovat čárky (bez mezer), které naznačují, že se do rejstříku dává více slov:

```
{\bf Definice.}
\ii lineární~prostor,vektorový~prostor
{\em Lineárním prostorem} (nebo též vektorovým prostorem) rozumíme ...
```

To dává totéž jako při `\ii lineární~prostor \ii vektorový~prostor` a tato ukázka demonstruje ještě jednu věc: je-li potřeba do parametru `\ii` dostat mezeru, pište vlnku nebo napište heslo uzavřené ve svorkách.

Pokud se v rejstříku objeví hesla skládající se z více slov, obvykle chceme, aby u hesla, které opakuje první slovo, se toto slovo v rejstříku nevypisovalo opakovaně, ale aby bylo nahrazeno pomlčkou. Například:

```
lineární podprostor 12, 16, 18, 29
— prostor 12, 16–32, 51
— závislost 18–20, 34
```

Při takovém požadavku pište místo vlnky mezi slovy lomítko. Příklad:

```
\ii lineární/prostor,vektorový/prostor
```

Makro pak začne kontrolovat, zda se po abecedním seřazení neobjeví stejná slova pod sebou. Pokud ano, napíše jen první slovo a místo ostatních dá pomlčku.

Někdy je vhodné kromě hesla `lineární/prostor` zařadit i heslo `prostor/lineární`. Aby se to nemuselo psát dvakrát, je k dispozici zkratka `@` napsaná za čárku na konci parametru:

```
\ii lineární/prostor,vektorový/prostor,@
% je totéž jako \ii lineární/prostor,vektorový/prostor
% \ii prostor/lineární,prostor/vektorový
```

Počet lomítek v hesle pro rejstřík není omezen. Můžete tedy vytvořit víceúrovňový rejstřík. Nicméně je třeba vědět, že zkratka `@` nevytváří všechny permutace, ale jen přesune první údaj před lomítkem za všechny ostatní. Takže `\ii a/b/c,@` je totéž jako `\ii a/b/c \ii b/c/a`.

Samotný rejstřík vznikne v místě sekvence `\makeindex`. Rejstřík obsahuje data z předchozího zpracování dokumentu `TeX`em, takže je potřeba `TeX`ovat aspoň dvakrát. Makro `\makeindex` abecedně seřadí data v rejstříku podle českých a slovenských pravidel řazení a upraví odkazy na stránky (aby se stránky neopakovaly a inklinovaly k zápisu ve tvaru 26–28). Makro `\makeindex` se nestará o prostředí, do kterého sazbu vyvrhne, ani o nadpis. To musíme udělat sami. `OPmac` nabízí pro sazbu do více sloupců makra `\begmulti<počet-sloupců> ... \endmulti`. Příklad:

```
\sec Rejstřík\par
\begmulti 3 \makeindex \endmulti
```

Do rejstříku musejí být zařazena jen „čistá“ slova, která neobsahují makra expandující na primitivní příkazy `TeX`u. Pokud chcete vytisknout v rejstříku něco komplikovanějšího, můžete sestavit slovník výjimek pomocí maker `\iis<heslo><mezeras><{<tisk>}>` (název makra můžete číst jako „`\ii` speciální“). Funkce je vysvětlena na příkladu:

```
\iis chikvadrat {$\chi$-kvadrát}
\iis relax {{\tt \char'\relax}}
\iis Goedelova/věta/o~neúplnosti {G\"odelova/věta/o~neúplnosti}
\iis věta/o~neúplnosti/Goedelova {věta/o~neúplnosti/G\"odelova}
```

Lze pak psát `\ii relax, \ii chikvadrat` nebo `\ii Goedelova/věta/o~neúplnosti,@`. `OPmac` abecedně řadí podle těchto hesel, ale když dojde na potřebu heslo vytisknout do

rejstříku, vytiskne místo těchto hesel materiál, který je uveden na pravé straně slovníku. Tímto způsobem lze řešit nejen tisk hesel, která je potřeba ošetřit speciálními makry (v příkladu slovo *relax*), ale také výjimky abecedního řazení. Slovník výjimek je možný zapsat kamkoli před `\makeindex`, typicky se píše na začátek dokumentu.

Výjimku z řazení dvojhlásky *ch* (například ve slově *mochнатý*, tj. *mnohonohý*) je možné zařídit pomocí tečky, která má stejně jako ostatní interpunkční znaky, nulovou řadící platnost (OPmac hesla řadí, jako by tam interpunkce nebyla). Takže třeba takto:

```
... \ii moc.hnatý ...  
\iis moc.hnatý {mochнатý}
```

Je-li při zpracování `\makeindex` zapnutý anglický jazyk (implicitní nastavení nebo po přepínači `\ehyph`), pak se *ch* neinterpretuje jako dvojhláska. Ostatní pravidla řazení zůstávají nezměněna.

Pro různé speciální znaky můžete využít znak `@`, který se řadí před celou abecedou. Speciální znak pak nahradíte až ve slovníku výjimek. Takže třeba `\ii Ernst~@~Young` pro řazení a `\iis Ernst~@~Young {Ernst \& Young}` pro tisk.

7.15 Poslední strana

Číslo poslední strany dokumentu (to nemusí být počet stran) je uloženo při opakovaném zpracování \TeX em v registru `\lastpage`. K tomu musí být otevřen soubor `.ref` s daty pro křížové odkazy, rejstřík a obsah. Pokud pracujete s těmito daty, je soubor `.ref` automaticky otevřen. Pokud ne, můžete si vynutit jeho otevření makrem `\openref`. Číslování stránek ve tvaru $\langle \textit{číslo} \rangle / \langle \textit{počet stran} \rangle$ zajistíte například takto:

```
\footline={\hss \rm \the\fontsize[10]\the\pageno/\the\lastpage \hss}
```


Kapitola 8

Programování maker

Autorům textů asi postačí znalosti z předchozích kapitol. Je to srovnatelné s informacemi, které mohou získat z běžných L^AT_EXových příruček. Pro zájemce o T_EX, kteří se chtějí dozvědět něco více o tom, jak T_EX funguje, a chtějí umět aspoň částečně číst cizí makra a tvořit vlastní, jsou určeny další kapitoly včetně této.

Následující text tedy ocení zejména ti pragmatici, jejichž pragmatismus dospěl tak daleko, že vědí, že je účelnější o T_EXu něco vědět než jen bezmyšlenkovitě přebírat hotová řešení.

8.1 Makrozáklady

Makro definujeme například příkazem `\def\cosi{text}`. V tomto příkladě je řídicí sekvence `\cosi` přiřazen význam makra s obsahem `text`. Kdykoli později je použita tato řídicí sekvence `\cosi`, T_EX ji promění (říkáme, že ji expanduje) na definovaný obsah `text`. Uvnitř textu makra se mohou vyskytovat další řídicí sekvence ve významu makra a ty se také expandují. T_EX při zpracování textu provádí úplnou expanzi všech maker.

S makrem tedy pracujeme ve dvou etapách. Nejprve pomocí `\def` makro *definujeme*, tj. T_EX si uloží do své paměti text makra. Potom přímým zápisem definované řídicí sekvence makro *použijeme*. Toto použití se může odehrát libovolněkrát. Při opakovaném použití se makro samozřejmě expanduje na stále stejný výsledek, dokud není předefinováno jinou definicí.

Závorky `{...}` při definování makra vymezují hranice obsahu makra a neslouží k zahájení a ukončení skupiny. Pokud chceme vytvořit makro například s přepínačem fontu uvnitř skupiny, musíme v textu makra napsat další závorky, které vymezují skupinu:

```
\def\Firma{{\bf SeQeFy}}
Oznamujeme Vám, že naše firma \Firma\ Vám dodá požadované zboží již zítra.
```

Příkaz `\def` nekontroluje, zda byla nově definovaná řídicí sekvence už dříve definována nebo má význam primitivního příkazu či čehokoli jiného. Kontrolní sekvenci prostě předefinuje. To může začátečníkům působit poměrně potíže, protože neznají všechny řídicí sekvence, které mají v T_EXu klíčový význam a jejich předefinování může natropit nečekané škody. Například po `\def\hbox{ahoj}` přestane fungovat mnoho maker i vnitřních rutin T_EXu, protože tyto rutiny zhusta příkaz `\hbox` používají. V podstatě se sazba zcela rozsype. Pro první krůčky s definováním maker tedy doporučuji používat třeba slova začínající velkým písmenem. Všechny klíčové řídicí sekvence ovlivňující interní algoritmy T_EXu mají totiž svůj název složený jen z malých písmen.

Těsně za `\def` (*sequence*) může být zapsáno pravidlo, podle kterého bude makro číst své parametry. Nejprve uvedeme příklad makra s jedním parametrem. Pravidlo má tvar `#1` (první a jediný parametr) a definice vypadá třeba následovně:

```
\def\makro #1{Text makra. Může obsahovat použití parametru #1.}
Když nyní napíšeme \makro A, \TeX{} to expanduje na:
Text makra. Může obsahovat použití parametru A.
Můžeme taky napsat třeba \makro\TeX, což se expanduje na:
Text makra. Může obsahovat použití parametru \TeX.
Konečně po zápisu \makro{nějaký parametr} dostaneme po expanzi:
Text makra. Může obsahovat použití parametru nějaký parametr.
```


Uvedená ukázka ilustruje různé použití makra s jedním parametrem. Tímto parametrem může být jediný token (znak nebo řídicí sekvence) nebo libovolně dlouhý text ohraničený závorkami `{...}`. \TeX za všech okolností při čtení parametrů nebo obsahu maker respektuje vnoření závorek `{ a }` libovolné úrovně. Pouze vnější závorky ohraničující parametr nebo obsah makra nejsou do parametru nebo obsahu makra zahrnuty.

Předchozí tvrzení o libovolně dlouhém textu jako parametru není zcela přesné. Do parametru se nesmí dostat `\par`, neboli prázdný řádek. Tvůrce \TeX předpokládal, že toto je typický překlep při psaní textu: nepárují závorky jak mají, přitom autor dokumentu typicky nechce v parametru makra přečíst více odstavců. \TeX tedy preventivně ohlásí chybu. Chcete-li vytvořit makro, které dovolí načíst do parametru i několik odstavců najednou, je potřeba před `\def` napsat prefix `\long`, tedy například `\long\def\makro#1{...}`.

Následuje ukázka makra `\trydef`, které pracuje stejně jako `\def`, ale odmítne definovat řídicí sekvenci, která už nějaký význam má.

```
\def\trydef#1{%
  \ifx#1\undefined \def\next{\def#1}%
  \else \errmessage{trydef: the \noexpand#1 is already defined.}%
  \def\next{\def\next}%
  \fi \next
}
```

Funkci tohoto makra si podrobně vysvětlíme, abychom ukázali, jak se při psaní \TeX ových maker přemýšlí. Předpokládejme použití `\trydef\cosi{text}`. Makro `\trydef` si sejme jeden parametr `\cosi` a expanduje na:

```
\ifx\cosi\undefined \def\next{\def\cosi}%
\else \errmessage{trydef: the \noexpand\cosi is already defined.}%
  \def\next{\def\next}%
\fi \next
```

Tento kód nyní začne \TeX vykonávat. Příkaz `\ifx\cosi\undefined` ověří, zda je řídicí sekvence `\cosi` nedefinovaná. Pokud je podmínka splněna (sekvence není definována), provede se kód před sekvencí `\else`. Pokud není podmínka splněna (sekvence je definována), provede se kód mezi `\else` a `\fi`. Předpokládejme nejprve, že `\cosi` není definována. V takovém případě se provede

```
\def\next{\def\cosi}
```

a následně se přeskočí vše za `\else` až po `\fi` a provede se `\next`, za kterým následuje `{text}` (kdo zapomněl, odkud se vzal `{text}`, připomene si, že použití `\trydef` bylo ve tvaru `\trydef\cosi{text}`). \TeX má tedy za úkol zpracovat následující kód:

```
\def\next{\def\cosi}\next{text}
```

Nejprve tedy \TeX definuje makro `\next` s významem `\def\cosi` a pak expanduje `\next`. Po expanzi `\next` tedy dostáváme `\def\cosi{text}`, což bylo cílem.

Nyní předpokládejme, že je `\cosi` již definované. V takovém případě se provede kód za `\else`, tedy:

```
\errmessage{trydef: the \noexpand\cosi is already defined.}%
\def\next{\def\next}\next{text}
```

Příkaz `\errmessage` oznámí na terminál chybu. Při té příležitosti vypíše text, který následuje ve svorkách, ovšem po úplné expanzi. V textu ale není žádoucí, aby bylo expandováno makro `\cosi`. Tomu je zabráněno příkazem `\noexpand`, který danému makru předchází. Na terminálu se tedy objeví zpráva:

```
trydef: the \cosi is already defined.
```

Dále se definuje `\next` jako `\def\next`, takže následné `\next{text}` expanduje na kód `\def\next{text}`. Takže je nakonec místo makra `\cosi` definováno makro `\next`, které je v `plainTeXu` a `CSplainu` používáno jako přechodné makro ve smyslu: „definuji a vzápětí použiji“. Je tedy naprosto jedno, jak je nakonec makro `\next` definováno.

8.2 Tanec s parametry

Makro v `TeXu` může mít více parametrů. V pravidle parametrů musejí být všechny parametry uvedeny pomocí znaku `#` a číslovány postupně počínaje číslem 1. Maximální počet parametrů je 9. Například:

```
\def\makro#1#2#3{První: #1, druhý: #2, třetí: #3.}
```

V pravidle parametrů se mohou za čísla parametrů vyskytovat znaky, sekvence nebo celé texty, které pak slouží jako *separátory* parametru. Při použití makra se do takto separovaného parametru nechte jediný token, ale veškerý text až po uvedený separátor. Příklad:

```
\def\makro#1 #2{první: #1, druhý: #2. }
1. \makro aha xyz          => první: aha, druhý: x. yz
2. \makro aha {xyz}        => první: aha, druhý: xyz.
3. \makro aha{xyz} uf      => první: aha{xyz}, druhý: u. f
4. \makro {aha xyz}u f     => první: {aha xyz}u, druhý: f.
5. \makro {aha {x}yz} {uf} => první: aha {x}yz, druhý: uf.
```

Makro v této ukázce je definováno s prvním parametrem separovaným mezerou a druhý parametr je neseparovaný. Poprvé je makro použito na řádce s číslem 1. Tam se do prvního parametru načte text `aha` až po první mezeru a do druhého parametru se načte jen `x`. Za expanzí makra tedy pokračují písmena `yz`. Ve druhém použití makra se do prvního parametru načte `aha` a do druhého `xyz`, protože je parametr obehnan svorkami. Ve třetím použití makra je prvním parametrem až po mezeru text `aha{xyz}` a druhým parametrem je znak `u`. Znak `f` pokračuje po expanzi. Ve čtvrtém použití je vidět, že pokud dáme text do svorek, separátor (v tomto případě mezera) uvnitř svorek nezabírá. Zabere až mezera za znakem `u`. Platí totiž obecné pravidlo, že do parametru makra `TeX` načte za všech okolností jen text, ve kterém svorky (s libovolnou úrovní vnoření) vzájemně párují. Poslední ukázka ilustruje vlastnost „mizení vnějších svorek“ z parametru, třebaže je parametr separován. O dalších vychytávkách se separovanými a neseparovanými parametry je možné se dočíst v TBN v sekci 2.1.

● **Cvičení 12** ● Před voláním makra `\makro` nastavte `\tracingmacros=1` a vyzkoušejte si výše uvedený příklad. Sledujte, co je o expanzi maker psáno v `.log` souboru. Rovněž můžete přidat další experimenty: `\makro {aha xyz}{}` `\makro {aha}{xyz}`.

Smysluplný příklad makra se separovaným parametrem je

```
\def\titulek#1\par{\bigskip \noindent{\bf #1}\par\nobreak \medskip}
```

Autor pak píše třeba `\titulek Moje první dojmy` a text nadpisu ukončí prázdným řádkem. Prázdný řádek se v `TeXu` interně promění v `\par` a tato sekvence v tomto případě slouží jako separátor parametru.

Je-li parametr neseparovaný, `TeX` přeskočí při jeho čtení případné mezery napsané před parametrem. Příklad:

```

\def\makro #1#2#3{...} % všechny tři parametry jsou neseparované
\makro abc ... dá stejný výsledek jako ... \makro a b c
\makro {první}{druhý}{třetí} je totéž jako \makro {první} {druhý} {třetí}
ale: \makro {}{ }{ } třetí ... #1 je prázdný, #2=<mezera>, #3=<mezera>třetí

```

Příkaz `\def` definuje řídicí sekvenci lokálně. To znamená, že pokud je příkaz `\def` ve skupině, po ukončení skupiny \TeX ochotně a rád definici zapomene. Pokud chcete, aby si ji pamatoval napříč skupinami, je nutné místo `\def` použít `\gdef` (globální `\def`).

Příkaz `\def<sekvence>{<text>}` přečte obsah makra `<text>` bez toho, aby v době definice tento obsah expandoval. K expanzi dojde až v případě použití makra. Někdy je ale potřebné přinutit \TeX k expanzi obsahu `<text>` už v době definice. K tomu slouží místo `\def` příkaz `\edef`, který provede úplnou expanzi obsahu makra během definice. Takže:

```

\def\a{původního}
\def\b{cosi \a} % \b je makro s obsahem cosi \a
\edef\c{cosi \a} % \c je makro s obsahem cosi původního
\def\a{jiného}
Nyní \b expanduje na cosi jiného, zatímco \c expanduje na cosi původního.

```

Pro globální `\edef` je v \TeX u připraven příkaz `\xdef`.

V závěru této sekce ukážeme ještě jednu vlastnost při definování maker, tzv. „zdvojení křížů“. V textu definice se smí symbol `#` vyskytovat jen tak, že za ním následuje číslo a tato dvojice znaků určuje místo, kam se má vložit přečtený parametr. Pokud ale v definici makra potřebujeme definovat interní makro se svými vlastními parametry, je potřeba zdvojit kříže. Jako příklad uvedeme makro `\jevseznamu <znak>{<seznam>}`, které odpoví, zda `<znak>` se vyskytuje v `<seznamu>`. Tedy například při `\jevseznamu A{abcxyz}` odpoví NE a při `\jevseznamu c{abcxyz}` odpoví ANO.

```

\def\jevseznamu#1#2{% #1 = znak, #2 = seznam
  \def\tmp##1#1##2\end{\ifx^##2^message{NE}\else\message{ANO}\fi}%
  \tmp#2#1\end}

```

Makro si podrobně vysvětlíme. Předpokládáme nejprve užití `\jevseznamu A{abcxyz}`. V takovém případě je `#1=A` a `#2=abcxyz` a makro expanduje na:

```

\def\tmp#1A#2\end{\ifx^#2^message{NE}\else\message{ANO}\fi}%
\tmp abcxyzA\end

```

Vidíme, že v rámci této první expanze se do míst jednoduchých křížů umístily aktuální parametry a dvojité kříže se pak proměnily v jednoduché. Nyní se tedy definuje přechodné makro `\tmp` jako makro s prvním parametrem `#1` separovaným znakem `A` a s druhým parametrem `#2` separovaným sekvencí `\end`. Následně se `\tmp` použije. Přitom se do prvního parametru vloží `abcxyz` a druhý parametr (mezi dvěma separátory) je prázdný. Makro `\tmp` spustí test `\ifx^#2^`, což je test na to, zda je parametr `#2` prázdný (viz sekci 8.4). On je v tomto případě skutečně prázdný, takže se provede `\message{NE}` a neprovede `\message{ANO}`. Příkaz `\message` vypíše svůj argument na terminál, takže si tam můžeme přechojit výstup algoritmu, slovo „NE“.

Předpokládejme nyní použití `\jevseznamu c{abcxyz}`. První expanze vypadá takto:

```

\def\tmp#1c#2\end{\ifx^#2^message{NE}\else\message{ANO}\fi}%
\tmp abcxyzc\end

```

Makro `\tmp` nyní má první parametr separován znakem `c` a druhý sekvencí `\end`, takže do prvního parametru přečte `ab` a do druhého `xyzc`. Protože druhý parametr není prázdný, odpoví makro `\tmp` zprávou „ANO“.

8.3 Numerické výpočty

Při použití klasického (nerozšířeného) \TeX u je pro programátora maker připravena jen ne příliš pohodlná možnost provádět numerické výpočty. Nejprve je potřeba deklarovat numerický registr a pak je možné jej používat, tj. dosazovat do něj hodnoty, přičítat, násobit, dělit a vyzvedávat si z něj hodnotu.

Řídící sekvenci deklarujeme jako numerický registr makrem `\newcount<sekvence>`. Registr může uchovávat celočíselné hodnoty v rozsahu cca ± 2 miliardy. K dispozici jsou následující příkazy, které modifikují numerický registr.

<code><registr> = <hodnota></code>	% uložení <i><hodnoty></i> do <i>registru</i>
<code>\advance<registr> by <hodnota></code>	% přičtení <i><hodnoty></i> k <i><registru></i>
<code>\multiply<registr> by <hodnota></code>	% násobení <i><registru></i> <i><hodnotou></i>
<code>\divide<registr> by<hodnota></code>	% celočíselné dělení <i><registru></i> <i><hodnotou></i>

Přitom *<hodnota>* může být také numerický registr, nebo to je celočíselná konstanta (numerický údaj). Podrobněji se o numerických údajích píše v dodatku C.

Příklady:

<code>\newcount\promennaP</code>	% deklarace P
<code>\newcount\promennaQ</code>	% deklarace Q
<code>\promennaP = 25</code>	% P := 25, uložení hodnoty do registru
<code>\promennaQ = \promennaP</code>	% Q := P, uložení hodnoty do registru
<code>\advance \promennaP by 1</code>	% P := P + 1, přičtení jedničky k registru
<code>\advance \promennaP by \promennaQ</code>	% P := P + Q
<code>\advance \promennaP by -9</code>	% P := P - 9, odečítání
<code>\message{\the\promennaP}</code>	% zobrazení hodnoty registru na terminál
<code>\the\promennaP</code>	% vytištění hodnoty registru v dokumentu

Pokud jste příklad správně sledovali, jistě víte, že se zobrazí hodnota 42.

Kromě celočíselných hodnot \TeX pracuje ještě s rozměry (viz také dodatek C). Metrický registr se deklaruje makrem `\newdimen`, je možné do něj vkládat metrické údaje a modifikovat jej pomocí `\advance`, `\multiply` a `\divide`. V případě `\advance` musí být *<hodnota>* metrický údaj a v případě `\multiply` a `\divide` je *<hodnota>* numerický registr nebo celočíselná konstanta. Vytisknout rozměrový registr lze pomocí `\the`. V tomto případě se \TeX vyjádří v jednotkách pt (monotypové body) a tuto jednotku za číselný výraz připojí. Daleko přirozenější je vypočítaný metrický údaj použít v sazbě, například v parametrech `\vskip`, `\hskip`, `\vrule`. Pak se rozměr přímo projeví v dokumentu třeba jako velikost mezery.

Metrický údaj je možné převést na celočíselnou hodnotu pomocí `\number<registr>`. Je-li *<registr>* roven například 2 cm, pak `\number<registr>` expanduje na 3729359. Obecně `\number` expanduje na násobek jednotky sp, přitom $1 \text{ pt} = 2^{16} \text{ sp} = 65536 \text{ sp}$.

Příkaz `\multiply` násobí metrický registr jen celočíselným násobkem a tudíž není moc užitečný. Naproti tomu zápis *<registr>* = *<násobek>**<registr>* umožňuje vynásobit rozměr i desetinným číslem. Příklad:

<code>\newdimen\rozmerU</code>	% deklarace U
<code>\rozmerU = 2.1cm</code>	% U := 2.1 cm
<code>\rozmerU = 3.1415\rozmerU</code>	% U := 3.1415 * U
<code>\parindent=0.7\parindent</code>	% \parindent := 0.7 * \parindent
<code>\hsize = 5.15\rozmerU</code>	% \hsize := 5.15 * U

Tento přehled není zdaleka kompletní. Úplný přehled práce s \TeX ovými registry je v TBN v sekci 3.3.

Je potřeba mít stále na paměti, že veškerá přiřazení v \TeX u jsou lokální v rámci skupin. Takže veškeré modifikace registrů se dějí lokálně a po ukončení skupiny se vracejí ke svým původním hodnotám. Chcete-li mít hodnotu uloženou nebo modifikovanou trvaleji napříč skupinami, je potřeba použít prefix `\global`, například:

```
\global\promennaP = 25
\global\advance\promennaP by-1
\global\multiply\promennaP by3
```

Příkazy `\advance`, `\multiply` a `\divide` čtou mezi $\langle\text{registrem}\rangle$ a $\langle\text{hodnotou}\rangle$ slovo `by`, které je nepovinné včetně mezer kolem tohoto slova. Takže se můžete setkat i s makry, která toto slovo zatajují, a tím snižují srozumitelnost:

```
\advance\promennaP 1
\advance\hsize\parindent
\multiply\promennaQ3
```

8.4 Větvení

\TeX nabízí několik příkazů ve tvaru `\if... $\langle\text{podmínka}\rangle$:`

Porovnávání celočíselných hodnot:

```
\ifnum $\langle\text{číslo1}\rangle$ = $\langle\text{číslo2}\rangle$ 
\ifnum $\langle\text{číslo1}\rangle$ < $\langle\text{číslo2}\rangle$ 
\ifnum $\langle\text{číslo1}\rangle$ > $\langle\text{číslo2}\rangle$ 
```

Lichost čísla:

```
\ifodd $\langle\text{číslo}\rangle$ 
```

Porovnávání rozměrů:

```
\ifdim $\langle\text{rozměr1}\rangle$ = $\langle\text{rozměr2}\rangle$ 
\ifdim $\langle\text{rozměr1}\rangle$ < $\langle\text{rozměr2}\rangle$ 
\ifdim $\langle\text{rozměr1}\rangle$ > $\langle\text{rozměr2}\rangle$ 
```

Porovnávání dvou tokenů:

```
\if $\langle\text{token1}\rangle$  $\langle\text{token2}\rangle$  % test shody ASCII hodnoty, expanduje
\ifcat $\langle\text{token1}\rangle$  $\langle\text{token2}\rangle$  % test shody kategorií, expanduje
\ifx $\langle\text{token1}\rangle$  $\langle\text{token2}\rangle$  % test úplné shody (ASCII i kategorie) nebo
% test shody významu řídicích sekvencí, neexpanduje
```

Dotaz na stav:

```
\ifhmode % zpracování probíhá v horizontálním módu
\ifvmode % zpracování probíhá ve vertikálním módu
\ifmmode % zpracování probíhá v matematickém módu
\ifinner % mód je vnitřní (horizontální, vertikální, matematický)
\ifeof $\langle\text{soubor}\rangle$  % test, zda je soubor na konci čtení
```

Větvení výpočtu do více případů:

```
\ifcase $\langle\text{číslo}\rangle$ 
```

Všechny tyto příkazy `\if... s výjimkou \ifcase mají formální strukturu:`

```
\if... $\langle\text{podmínka}\rangle$   $\langle\text{text, je-li test pravdivý}\rangle$ \fi
nebo:
\if... $\langle\text{podmínka}\rangle$   $\langle\text{pravda}\rangle$ \else $\langle\text{nepravda}\rangle$ \fi
```

V případě `\ifcase` je konstrukce následující:

```
\ifcase $\langle\text{číslo}\rangle$   $\langle\text{případ 0}\rangle$ \or $\langle\text{případ 1}\rangle$ \or $\langle\text{případ 2}\rangle$ \or... \or $\langle\text{případ }n\rangle$ \fi
nebo:
\ifcase $\langle\text{číslo}\rangle$   $\langle\text{případ 0}\rangle$ \or $\langle\text{případ 1}\rangle$ \or... \or $\langle\text{případ }n\rangle$ \else $\langle\text{jinak}\rangle$ \fi
```

Text $\langle\text{případ }0\rangle$ se provede, je-li $\langle\text{číslo}\rangle=0$, dále $\langle\text{případ }1\rangle$ se provede, je-li $\langle\text{číslo}\rangle=1$, atd.

Příkazy `\if...` je možno vnořovat do sebe, například:

```
\ifnum \pageno>20
  \ifodd\pageno strana lichá větší než 20
  \else strana sudá větší než 20
\fi
\else
  \ifodd\pageno strana lichá menší než 20
  \else strana sudá menší nebo rovna 20
\fi
\fi
```

Všechny příkazy `\if...` s výjimkou `\ifx` expandují text podmínky. Provádějí přitom úplnou expanzi. Takže třeba funguje:

```
\def\podminka{\pageno>20 }
\ifnum\podminka strana větší než 20\fi
\def\objekt{\pageno}
\ifnum\objekt>20 objekt větší než 20\fi
```

Příkazy `\if...` samotné podléhají rovněž expanzi. To může při vnořování `\if...` působit některým programátorům potíže: příkazy `\if` napsané později v *⟨podmínce⟩* se vyhodnotí dříve než úvodní příkaz `\if`. Je tedy potřeba přesně oddělit konec *⟨podmínky⟩* od dalšího textu. Pověšme si, že za čísla je proto žádoucí psát mezeru. Naopak při porovnávání dvou tokenů příkazy `\if` nebo `\ifx` se podmínka skládá ze dvou tokenů a každý další token (včetně mezery) je součástí textu, který se vykoná při shodě tokenů.

Test na prázdný parametr v makru můžeme provést dvěma způsoby.

```
\def\makro#1{...
  \def\tmp{#1}\ifx\tmp\empty parametr prázdný\else neprázdný\fi
}
\def\makro#1{...
  \ifx^#1^parametr prázdný\else neprázdný\fi
}
```

První způsob využívá toho, že je v plain \TeX u předdefinováno makro `\empty` způsobem `\def\empty{}` a pomocné makro `\tmp` je proti tomuto makru porovnáváno. Druhý způsob je daleko zajímavější a jeho výhodou je, že proběhne celý na úrovni procesu expanze. Je-li parametr prázdný, pak se test realizuje jako `\ifx^^` a to je pravda, protože první zobák je skutečně roven druhému. Je-li parametr neprázdný (třeba obsahuje `abc`), pak se test realizuje jako

```
\ifx^abc^parametr prázdný\else neprázdný\fi
```

Podmínka není splněna, protože zobák se nerovná písmenu `a`. Následující text ve tvaru: `bc^parametr prázdný` se přeskočí a vykoná se to, co následuje za `\else`. Je zřejmé, že tento test může selhat, pokud je prvním znakem parametru zobák. Máme-li tyto obavy, je potřeba místo zobáku zvolit takový znak (nebo řídící sekvenci), o kterém víme, že jej uživatel v parametru jako první znak nikdy nepoužije.

Programátor maker může deklarovat vlastní `\ifcosi` pomocí makra `\newif\ifcosi`. Deklarovaná řídící sekvence musí začínat na `\if...` (například `\ifcosi`) a je při deklaraci automaticky propojena s makry `\cositrue` a `\cosifalse`. Proběhlo-li `\cositrue`, je test `\ifcosi` vyhodnocen jako pravdivý a proběhlo-li naposled `\cosifalse`, je test `\ifcosi` vyhodnocen jako nepravdivý. Jako příklad pro tuto techniku rozšíříme makro `\jevseznamu` ze sekce 8.2 tak, aby začalo sloužit dalším makrům.

```

\newif\ifincluded
\def\jevseznamu#1\in#2{% #1 = slovo, #2 = seznam
  \def\tmp##1,##2\end{\ifx^##2^\includedfalse\else\includedtrue\fi}%
  \tmp,#2,#1,\end
}
\def\dalsimakro #1{... % makro, které využije makro \jevseznamu
  \jevseznamu #1\in{ha,bha,uff,tuf}%
  \ifincluded <vykonej něco, když #1 leží v seznamu>
  \else      <vykonej něco, když #1 neleží v seznamu>
  \fi
}

```

Makro `\jevseznamu` má místo původních příkazů `\message{NE}` a `\message{ANO}` vloženo `\includedfalse` a `\includedtrue`. Po provedení makra se tedy můžeme pomocí `\ifincluded` zeptat, jak situace dopadla, a podle toho větvit další výpočet. Nová verze makra `\jevseznamu` je navíc mírně vylepšena: má separován první parametr sekvencí `\in`, takže je možné do prvního parametru napsat (až po `\in`) celý text, například nějaké slovo. Makro odpoví kladně, právě když je dané slovo v seznamu slov oddělených čárkou.

V závěrečném příkladu této sekce ukážeme tisk aktuálního data a času zpracování dokumentu. `TeX` je vybaven interními numerickými registry `\day`, `\month` a `\year`, které obsahují aktuální den v měsíci, číslo měsíce a rok. Vytisknout aktuální datum tedy není problém: `\the\day.\the\month.\the\year`. Chceme-li měsíc tisknout slovy, použijeme `\ifcase` (jak je ukázáno v následující ukázce). Zajímavější to je s tiskem aktuálního času, který je uložen v registru `\time` v minutách od poslední půlnoci. Je tedy potřeba spustit nejprve drobný výpočet:

```

\newcount\minuty \newcount\hodiny
\def\caszpracovani{\minuty=\time
  \divide\minuty by60 \hodiny=\minuty
  \multiply\minuty by-60 \advance\minuty by\time
  \the\hodiny.\ifnum\minuty<10 0\fi\the\minuty
}
\def\datumzpracovani{\the\day.~%
  \ifcase\month\or ledna\or února\or března\or dubna\or května\or června\or
    července\or srpna\or září\or října\or listopadu\or prosince\fi
  \space\the\year
}

```

Pokud chcete znát aktuální čas s přesností na sekundy, můžete použít příkaz z `pdfTeXu` `\pdfcreationdate`, který expanduje na formát data a času popsany v sekci 11.2 u údaje `/CreationDate`.

8.5 Cykly

Cyklus vytvoříme pomocí `plainTeX`ového makra `\loop`. Použití má následující strukturu:

```

\loop
  <základní příkazy vykonávané v cyklu>
  \if... <podmínka opakování cyklu>
    <další příkazy vykonávané v cyklu>
  \repeat

```

Je-li hned napoprvé podmínka opakování cyklu nepravdivá, provedou se jedenkrát jen základní příkazy v cyklu a cyklus končí. Je-li podmínka opakování cyklu pravdivá, provedou

se ještě další příkazy v cyklu a cyklus se opakuje. Základní nebo další příkazy mohou také zcela chybět. Chybí-li obojí, nemá cyklus smysl.

Povšimneme si, že makro `\loop` poněkud mění způsob použití `\if...` v podmínce cyklu: následuje sice text, který se při splnění podmínky provede, ale není ukončen `\fi` a není možné použít `\else`. Místo toho je ukončen sekvencí `\repeat`.

Jako příklad cyklu je ukázka makra `\opakuj{<co>}{<kolikrát>}`. Makro vytiskne do dokumentu `<co>` zopakované `<kolikrát>`, takže třeba `\opakuj{ahoj}{3}` vytiskne ahojahojahoj.

```
\newcount\opaknum
\def\opakuj#1#2{\opaknum=#2
  \loop #1% opakovaný text
    \advance\opaknum by-1 % zmenšení čítače cyklu
    \ifnum \opaknum>0 \repeat
}
```

Makro ovšem nefunguje správně, je-li počet opakování roven nule. Náprava může vypadat takto:

```
\newcount\opaknum
\def\opakuj#1#2{\opaknum=#2
  \loop
    \ifnum \opaknum>0 #1\advance\opaknum by-1 \repeat
}
```

Cykly se dají do sebe vnořovat, ale vnořený cyklus `\loop...\repeat` musí být schován do svorek. Cyklus `\loop` je v plainTeXu implementován jako rekurzivní makro.

8.6 Další příkazy, bez nichž se opravdový makroprogramátor neobejde

Zde je uvedeno jen stručné shrnutí některých dalších příkazů, aby čtenář získal povědomí a přehled, jaké další příkazy existují. Pro podrobnější výklad a další ukázky je třeba sáhnout například po TBN.

Příkazem `\let<token1>=<token2>` (nebo také stručněji `\let<token1><token2>`) se přiřadí `<token1>` stejný význam, jako má `<token2>`. Přitom `<token1>` musí být řídicí sekvence nebo aktivní znak. Nechť je například `\makro` nějak definované makro. Pak po použití `\let\sekvence=\makro` se stává `\sekvence` stejně definovaným makrem. Pokud dále je `\makro` předdefinováno, `\sekvence` si stále drží svou původní hodnotu makra. Chcete-li, aby přiřazení proběhlo globálně (napříč skupinami), je potřeba předřadit prefix `\global`, tedy `\global\let<token1>=<token2>`.

Příkaz `\futurelet<sekvence><token1><token2>` se v TeXu interně převede na kód tvaru `\let<sekvence>=<token2><token1><token2>`. Typicky je `<token1>` makrem, které se tedy provede po přiřazení významu `<sekvenci>` podle `<token2>`.

Příkaz `\expandafter<token1><token2>` expanduje na `<token1><expandovaný token2>`. Provede jen expanzi první úrovně, nikoli úplnou expanzi. Typicky jsou `<token1>` a `<token2>` makra nebo expandovatelné příkazy. Často se stává, že například makro `<token1>` potřebuje přečíst své parametry nikoli ve formátu `<token2>`, ale chce mít ten `<token2>` nejprve expandovaný a pak z něj bude číst parametry. Každý programátor TeXových maker postupně dospěje do stádia, kdy se bez příkazu `\expandafter` (a nejen jednoho) ve svých makrech neobejde.

Podléhá-li nějaký text úplné expanzi (při `\edef`, v argumentech `\message`, `\write` atd.), je občas potřebné potlačit expanzi některého makra nebo expandovatelného příkazu. K tomu slouží prefix `\noexpand`.

Příkaz `\string`*(sekvence)* rozloží následující *(sekvenci)* na posloupnost tokenů uvozenou tokenem `\` (jako obyčejný znak), který je následován jednotlivými písmeny z názvu *(sekvence)*. Každé písmeno je samostatný token. Takže třeba `\ahoj` je jediný token, sice řídící sekvence. Ale `\string\ahoj` vytvoří token `\` následovaný čtyřmi tokeny `a`, `h`, `o`, `j`. I příkaz `\string` se dá použít v podobných případech, jako `\noexpand`. Rozdíl je ovšem v tom, že výsledná posloupnost tokenů už nebude v budoucnu interpretována jako řídící sekvence. Na druhé straně `\noexpand` nechává řídící sekvenci nezměněnou a může se později projevit. Příklad:

```
\def\acosi\def\bttest
\edef\c\b:\noexpand\ac % c je makro obsahující test: \a
\edef\d\b:\string\ac % d je makro obsahující test: \a
\c % vytiskne test: cosi
\d % vytiskne test: \a
```

Příkaz `\string` se dá použít i následovaný *(tokenem)*, který není řídící sekvencí. V takovém případě mu pouze změní kategorii na kategorii obyčejného znaku (12).

Dvojice příkazů `\csname`*(text)*`\endcsname` vytvoří řídící sekvenci. Název této řídící sekvence se bude skládat ze znaků, které vzniknou úplnou expanzí *(textu)*. Tato konstrukce umožní dopravit do názvu řídící sekvence jakékoli znaky (nejen písmena) včetně znaků, které jsou výsledkem expanze typu `\the`*(registr)*. To umožní implementovat pole nebo slovníky. Příklady:

```
\newcount\tmpnum \tmpnum=0
\def\pole[#1]{\csname pole:#1\endcsname}
\def\napln#1{\advance\tmpnum by1
\expandafter\def\csname pole:\the\tmpnum\endcsname{#1}}
\napln{první} \napln{druhý} \napln{třetí}
\pole[2] % expanduje na druhý
\pole[3] % expanduje na třetí

\def\ulozdoslovníku #1#2{%
\expandafter\def\csname slov:#1\endcsname{#2}}
\def\vyzvednizeslovníku #1{\csname slov:#1\endcsname}
\ulozdoslovníku {já jsem} {ich bin}
\ulozdoslovníku {ty jsi} {du bist}
\ulozdoslovníku {on je} {er ist}
\vyzvednizeslovníku {ty jsi} % expanduje na du bist
```

V prvním příkladu podrobně vysvětlíme makro `\napln`. Nejprve je tam zvětšen čítač `\tmpnum` o jedničku. Ten představuje *(index)* prvku pole, do kterého chceme údaj uložit. Údaj `#1` uložíme tak, že definujeme `\def\pole:(index){#1}`. Definovaná řídící sekvence je zde pro názornost zakreslena v rámečku. Nestačí jen tuto sekvenci obklopit `\csname... \endcsname`, protože pak by příkaz `\def` definoval `\csname`. Je tedy potřeba použít `\expandafter`, který požádá příkaz `\csname` o expanzi první úrovně, tedy `\csname` ve spolupráci s `\endcsname` vyrobí řídící sekvenci. Například při `\tmpnum=3` vyrobí řídící sekvenci `\pole:3` a tuto sekvenci teprve definuje příkaz `\def`.

Pokud je `\csname`*(text)*`\endcsname` nedefinovaná řídící sekvence, nedojde při její expanzi v tomto případě k chybě, ale tato sekvence bude interpretována jako `\relax` (což je příkaz „nic nedělej“). Zatímco tedy test na nedefinovanost řídící sekvence se typicky pro-

vádí pomocí `\ifx\sekvence\undefined ...`, chceme-li testovat nedefinovanost sekvence konstruované pomocí `\csname<text>\endcsname`, je třeba psát:

```
\expandafter \ifx \csname<text>\endcsname \relax není definována
\else je definována
\fi
```

V tomto příkladě nejprve `\expandafter` „štouchne“ do `\csname` a požádá ho o expanzi první úrovně, tedy o proměnu kódu `\csname<text>\endcsname` v řídicí sekvenci. Ta se stane jediným tokenem. Za příkazem `\ifx` se tedy nyní nalézají dva tokeny, první je výstupem z činnosti `\csname <text>\endcsname` a druhý je `\relax`. Porovnávají se jejich významy. Není-li řídicí sekvence vytvořená pomocí `\csname <text>\endcsname` definovaná, má význam `\relax`. Makro `\vzvednizeslovníku` z předchozí ukázky je tedy možno vylepšit takto:

```
\def\vzvednizeslovníku #1{%
\expandafter\ifx\csname slov:#1\endcsname\relax
\message{Varování: slovo #1 není v seznamu slov.}%
\else \csname slov:#1\endcsname \fi
}
```

Příkaz `\aftergroup<token>` zařadí `<token>` za konec skupiny, ve které se příkaz vyskytl. Nejčastěji je `<token>` nějaké makro, které se provede po ukončení skupiny, tedy:

```
{... \aftergroup\makro ...}
je totéž jako:
{... ...}\makro
```

Příkaz `\afterassignment<token>` zařadí `<token>` po vykonání následujícího přiřazení. Přiřazením v \TeX u je vložení hodnoty do registru, dále též vykonání příkazu `\def` nebo `\let`. Například

```
\afterassignment\makro \let\cosi=\dalsi
je totéž jako:
\let\cosi=\dalsi \makro
```

Makroprogramátor výjimečně použije i jiný kontejner na texty než představují makra bez parametru. Jsou to registry typu `<tokens>`. Deklarace registru se provede makrem `\newtoks<sekvence>`, uložení do registru příkazem `<sekvence>={<text>}` a vyzvednutí textu z registru pomocí `\the<sekvence>`. Vzhledem k tomu, že při ukládání do registru je rovnítko nepovinné, je možné tyto registry použít na vyplňování formulářových údajů. Například:

```
\newtoks\jméno \newtoks\bydliště \newtoks\zaměstnání

\jméno {Ferdinand Mravenec}
\bydliště {Nad paloukem 1}
\zaměstnání {Práce všeho druhu}
```

Specialitou registrů typu `<tokens>` je, že na jejich obsah neúčinkuje úplná expanze prováděná při `\edef` a při zápisu do souborů (`\message`, `\write`). Expanduje se jen `\the<sekvence>` na obsah registru, ale obsah registru zůstává dále nezměněn. I tuto vlastnost je někdy vhodné využít.

Příkaz `\relax` (nic nedělej) vkládají zkušeni makroprogramátoři za takové příkazy, které mají neúplné parametry a hrozilo by, že uživatel omylem ve svém textu parametr doplní. Například píše na konec svého makra `\vskip 6pt\relax`, protože parametr může pokračovat třeba `plus2pt minus1pt`. Příkaz `\relax` takovému pokračování zabrání. Podrobněji se o tom píše v TBN na str. 70.

Kapitola 9

Boxy, linky, mezery

Následující náčrtek ukazuje, co si představit pod příkazem `\hbox{pokusný text}`.



Příkaz `\hbox{<text>}` zapouzdří `<text>` do nedělitelného neviditelného obdélníku, jenž má výšku (nad účařím) shodnou s nejvyšším elementem `<textu>`, hloubka pod účařím je rovněž odvozena podle elementu s největší hloubkou a šířka je rovna šířce `<textu>`. Čáry vyznačující na obrázku hranici boxu a účaří v sazbě pochopitelně nebudou. Box jako celek bude do sazby zařazen v horizontálním módu jako jediné písmeno a ve vertikálním módu obsadí samostatný řádek.

Kromě příkazu `\hbox{<text>}` je k dispozici ještě `\vbox{<sazba>}`. Zatímco `<text>` je při plnění `\hboxu` zpracován ve vnitřním horizontálním módu, `<sazba>` je při sestavování `\vboxu` zpracována ve vnitřním vertikálním módu. To znamená, že jednotlivé elementy sazby (boxy, linky, nikoli samotné znaky) se kladou pod sebe podobně jako při sestavování celé strany textu. Výskyt písmena nebo jiného znaku v `<sazbě>` zahájí uvnitř `\vboxu` odstavcový mód a prázdný řádek (tedy `\par`) nebo konec `\vboxu` ukončí odstavec a vloží zalomené řádky pod sebe do `\vboxu`. Případné samostatné `\hboxy` uvnitř `\vboxu` budou vloženy jako samostatné řádky pod sebe. Sestavený `\vbox` bude mít účaří totožné s účařím svého posledního řádku, hloubku shodnou s hloubkou posledního řádku a vše nad účařím tvoří výšku `\vboxu`. Šířka `\vboxu` odpovídá šířce nejširšího řádku v `<sazbě>`. Podle stejného pravidla jako `\hbox` je i `\vbox` jako celek vložen do horizontálního módu jako jediné písmeno a do vertikálního módu jako jediný řádek.

Vnitřek `\hboxu` i `\vboxu` bude vždy sestaven uvnitř lokální skupiny. Můžete tam tedy provést nastavení, která budou platit jen uvnitř boxu.

Boxy typu `\hbox` a `\vbox` je možné libovolně do sebe vnořovat. Například

```
\hbox{X\quad\vbox{\hbox{a}\hbox{bbb}\hbox{gg}}\quad Y\quad
\vbox{\hspace=2cm\noindent pp qq rr ss tt vv ww}\quad Z}
```

vytvoří:

```

a
bbb      pp qq rr ss tt
X  gg    Y  vv ww      Z
```

Toto samotné ještě není nic oslnujícího. Síla \TeX u při používání `\hboxů` a `\vboxů` se projeví až tehdy, když do těchto boxů začnete vkládat linky, které pruží na celkovou výšku/šířku/hloubku boxu, dále pružné mezery a pevné mezery a když budete specifikovat, na jakou celkovou šířku/výšku se má box vytvořit.

V \TeX u při sestavování sazby je možné skoro vše považovat za box. Písmeno je elementární box se svou výškou, hloubkou a šířkou. V horizontálním módu tyto boxy \TeX klade společně s mezerami vedle sebe a pak je rozlomí na řádky, tedy boxy s šířkou `\hspace`, které klade pod sebe. Celou sazbu pak \TeX rozlomí a vloží do boxů o výšce `\vspace`, přidá ke každému takovému boxu box se záhlavím a box se zápatím a tím vzniká box s celou stranou, který vystupuje do DVI nebo PDF.

9.1 Pružné a pevné mezery

Uvnitř boxů často využijeme pro vytvoření mezer následující příkazy.

V horizontálním módu (tj. v odstavcovém nebo vnitřním horizontálním módu):

```
\kern<velikost> ... pevná horizontální mezera,  
\hskip<velikost> plus<roztážení> minus<stažení> ... pružná horizontální mezera,  
\hfil ... nulová mezera s ochotou se roztáhnout (jako \hskip Opt plus1fil),  
\hfill ... jako \hfil, ale nekonečně pružnější (jako \hskip Opt plus1fill),  
\hss ... nulová mezera s ochotou se roztáhnout nebo stlačit do  
záporných hodnot (jako \hskip Opt plus1fil minus1fil).
```

Ve vertikálním módu:

```
\kern<velikost> ... pevná vertikální mezera,  
\vskip<velikost> plus<roztážení> minus<stažení> ... pružná vertikální mezera,  
\vfil ... nulová mezera s ochotou se roztáhnout (jako \vskip Opt plus1fil),  
\vfill ... jako \vfil, ale nekonečně pružnější (jako \vskip Opt plus1fill),  
\vss ... nulová mezera s ochotou se roztáhnout nebo stlačit do  
záporných hodnot (jako \vskip Opt plus1fil minus1fil).
```

Příkaz `\kern<velikost>` vloží horizontální nebo vertikální mezera v závislosti na kontextu, v jakém módu byl použit. Stejnou mezera vytvoří příkazy `\hskip<velikost>` nebo `\vskip<velikost>` (bez nepovinné části parametru `plus...` a `minus...`). Rozdíly jsou dva:

- Mezera z `\hskip` a `\vskip` podléhá řádkovému nebo stránkovému zlomu (pokud není uvnitř boxu). Naproti tomu mezery z `\kern` jsou nedělitelné.
- Příkazy `\hskip` a `\vskip` kladou mezera jen ve svém módu (horizontálním nebo vertikálním) a pokud jsou použity v nesprávném módu, dovedou si vynutit přechod do svého módu. Tj. `\hskip` ve vertikálním módu nejprve zahájí odstavec a `\vskip` v odstavcovém módu nejprve ukončí odstavec. Naproti tomu mezera z `\kern` je vertikální nebo horizontální podle módu, ve kterém byla použita. Nikdy nemůže obsahovat pružnost.

Parametry pružnosti za slovy `plus` a `minus` mohou obsahovat metrický údaj, vymezující zhruba mez roztážitelnosti a stlačitelnosti. Nebo mohou obsahovat údaj `1fil` (ochota roztáhnout se nebo stlačit se s nekonečnou tolerancí). V sousedství takové mezery všechny mezery s konečnou tolerancí (včetně mezislovních) zaujmou jen svou základní velikost. Dále je možno použít jiné násobky `fil` (například pro „šišaté centrování“) nebo více písmen `l` (`fill` a `filll`), což přebíjí pružnost mezer s pouhým `fil`. Více viz TBN v sekci 3.5.

Za slovem „to“ při konstrukci boxu je možné specifikovat požadovanou šířku `\hbox` nebo výšku `\vbox` takto:

```
\hbox to<šířka>{<text>} ... box bude mít specifikovanou šířku  
\vbox to<výška>{<sazba>} ... box bude mít specifikovanou výšku
```

Pokud `<text>` nebo `<sazba>` neobsahuje tolik materiálu anebo pružných mezer, aby mohl \TeX vyhovět specifikovanému rozměru boxu, ohlásí na terminál a do `.log` souboru `Overfull` nebo `Underfull` `\hbox/\vbox` a sazbu nechá přechýat z boxu (nebo vloží hrozivé mezery).

9.2 Centrování

Centrování textu na řádku provedeme pomocí

```
\hbox to\hsize{\hfil<text>\hfil}
```

Pružné mezery `\hfil` na obou stranách mají stejnou „sílu pružinky“ a roztáhnou se stejně. Příklad:

```

\letfont\titulfont=\tenbf at14pt
\def\titul #1\par {\hbox to\hsize{\hfil\titulfont #1\unskip\hfil}}

\titul Nadpis veledíla

Text veledíla.
\bye

```

V tomto příkladu byl na prvním řádku připraven fontový přepínač `\titulfont`, který zahájí sazbu v tučné variantě fontu ve velikosti 14 pt. Na druhém řádku je definováno makro `\titul` s jedním parametrem `#1`, přitom tento parametr je ukončen sekvencí `\par`, která se v \TeX u interně zjeví na každém prázdném řádku. Uživatelí makra `\titul` tedy řekněte, že má toto makro použít s parametrem, který je ukončen prázdným řádkem.

Uživatel použil `\titul Nadpis veledíla` (*prázdný řádek*), takže se vymezený parametr `Nadpis veledíla` zkopíroval do parametru `#1` a makro `\titul` expanduje na:

```
\hbox to\hsize{\hfil\titulfont Nadpis veledíla \unskip\hfil}
```

Tato konstrukce vytvoří box na celou šířku sazby, ve kterém centruje uvedený text. Text je navíc vysázen v poněkud větším fontu `\titulfont` (lokálně uvnitř boxu).

Dodatečné vysvětlení vyžaduje použití příkazu `\unskip`, který odebírá případnou předcházející mezeru. Parametr `Nadpis veledíla` má totiž za posledním písmenem „a“ mezeru, která tam vznikla při proměně konce řádku v mezeru (viz sekci 2.1). Příkaz `\unskip` ji odebere. Bez použití tohoto příkazu by nebyl nadpis naprosto přesně centrováný.

Stejné centrování bychom dosáhli také konstrukcí `\hbox to\hsize{\hss<text>\hss}`. Tato varianta navíc toleruje `<text>` širší než `\hsize`. Takový text bude přechýlávat v obou směrech přes okraj sazby stejně. Původní případ (s mezerami `\hfil`) způsobí při `<textu>` přesahujícím přes `\hsize` hlášení `Overfull \hbox` a k centrování nedojde.

Plain \TeX definuje pro zápis `\hbox to\hsize` zkratku `\line`. Dále plain \TeX definuje `\centerline{<text>}` jako `\line{\hss<text>\hss}`. Takže makro `\titul` by mohlo využít uvedené zkratky plain \TeX u takto:

```
\def\titul #1\par {\centerline{\titulfont #1\unskip}}
```

● **Cvičení 13** ● Prozkoumejte v souboru `plain.tex` definice maker `\line`, `\centerline`, `\leftline` a `\rightline`. Vysvětlete chování maker `\leftline` a `\rightline`.

● **Poznámka** ● \LaTeX definuje makro `\line` pro naprosto jiné účely. To je jedna z věcí, kvůli které nelze tvrdit, že \LaTeX pouze rozšiřuje makra plain \TeX u.

9.3 Boxy s vyčnívající sazbou

V \TeX u se občas hodí vytvořit `\hbox to0pt{<text>}`, resp. `\vbox to0pt{<sazba>}`. Použijeme to, pokud chceme vysunout sazbu do okraje, pokud chceme sazbu speciálním způsobem centrovat, pokud chceme překrýt sazbu jinou sazbou nebo pokud chceme umístit sazbu na specifikované místo vzhledem k místu, kde zrovna je aktuální bod sazby. Pochoptelné je potřeba, aby `<text>` nebo `<sazba>` obsahovaly „návratovou mezeru“, která vrátí bod sazby uvnitř boxu do výchozího místa a dovolí tak \TeX u vytvořit box nulových rozměrů. Takovou návratovou mezerou je typicky `\hss`, resp. `\vss`.

Jako příklad vytvoříme makro `\item{<odrážka>}`, které vysune text odrážky vlevo od odstavce v prvním řádku. Skupinu odstavců s odrážkami bude autor vkládat do prostředí vymezeném příkazy `\startitems` a `\stopitems` třeba takto:

```

\startitems
\item{1.} První odstavec.
\item{2.} Druhý odstavec.
...
\item{9.} Devátý odstavec.
\item{10.} Desátý odstavec.
\stopitems

```

Uvedená makra lze definovat následovně:

```

\def\item#1{\par \noindent \hbox to0pt{\hss#1\kern.2em}\ignorespaces}
\def\startitems{\medskip\bgroup\advance\leftskip by\parindent}
\def\stopitems{\par\egroup\medskip}

```

Makro dává tento výsledek:

1. První odstavec.
2. Druhý odstavec. ...
9. Devátý odstavec.
10. Desátý odstavec.

Schéma sazby odrážky:

$$\overbrace{\langle \text{odrážka} \rangle}^{\text{\hss (záporná pružná mezer)}} \leftarrow \text{box nulových rozměrů}$$

$$\text{\kern.2em}$$

Vidíte, že začátky odstavců jsou přesně pod sebou a tečky za čísla taky. Funkci maker si podrobně vysvětlíme. Makro `\startitems` vloží vertikální mezeru o velikosti půlky řádku pomocí `\medskip` a zahájí skupinu pomocí `\bgroup`. Uvnitř skupiny zvětší `\leftskip`¹⁾ o `\parindent`²⁾. Makro `\item` nejprve ukončí předchozí odstavec pomocí `\par`, protože se nedá předpokládat, že bude uživatel sám vkládat mezi odstavce s odrážkami prázdné řádky. Dále makro zahájí nový odstavec pomocí `\noindent`. Parametr #1 je vysunut z nulového boxu doleva a odsunut od okraje odstavce o mezeru velikosti `.2em`, tedy 0,2 krát velikost písma. Na konci makra je použit příkaz `\ignorespaces`, který ignoruje následující mezeru. Uživatel totiž nebude asi psát `\item{1.}První odstavec`, protože to nepůsobí esteticky. Jeho mezeru (kterou tam vložil z estetických důvodů) je třeba ignorovat.

Makro `\stopitems` nejprve ukončí odstavec pomocí `\par` a potom ukončí skupinu pomocí `\egroup`. Nakonec vloží vertikální mezeru ve velikosti půlky řádku (`\medskip`).

● **Cvičení 14** ● Uvedená makra si vyzkoušejte a zkuste si rozmyslet, k jaké chybě by došlo, kdyby byla v makru `\stopitems` skupina ukončena dříve než odstavec.

Makra jsou navržena tak, že je možné prostředí `\startitems` a `\stopitems` do sebe vnořovat. Vnitřní dvojice `\startitems` a `\stopitems` vytvoří novou skupinu (uvnitř skupiny) a v ní znovu zvětší `\leftskip` o `\parindent`, takže nyní bude už levé odsazení rovno dvojnásobku `\parindent`. Po ukončení vnitřní dvojice se \TeX vrací k hodnotě `\leftskip` před zahájením skupiny, takže se vrací k odsazení o jeden `\parindent`.

● **Poznámka** ● Plain \TeX definuje zkratky: `\llap{\langle text \rangle}` je `\hbox to0pt{\hss\langle text \rangle}` (tj. `\langle text \rangle` vysunutý doleva) a `\rlap{\langle text \rangle}` je `\hbox to0pt{\langle text \rangle\hss}` (tj. `\langle text \rangle` vysunutý doprava). Při využití těchto zkratk lze makro `\item` definovat stručněji:

```

\def\item#1{\par\noindent\llap{#1\kern.2em}\ignorespaces}

```

¹⁾ mezeru vkládaná na začátek každého řádku

²⁾ velikost odstavcové zarážky


PlainTeX rovněž definuje své makro `\item`, které se chová poněkud odlišně od makra zde uvedeného, protože nepředpokládá odsazení odstavce pomocí `\leftskip`. Jeho definici je možné samozřejmě dohledat v souboru `plain.tex`. Není to uděláno příliš šikovně. Pragmatik asi použije hotové řešení z OPmac, viz sekci 7.5.

9.4 Linky

Linka v TeXu je černý obdélník, který má jako box svou výšku nad účařím, hloubku pod účařím a šířku. V horizontálním módu je možné použít linku `\vrule`, která má implicitní šířku 0,4 pt a implicitní výšku a hloubku má stejnou, jako je výška a hloubka boxu, ve kterém se nalézá. Je to tedy typicky svislá linka, která se natahuje na výšku a hloubku vnějšího boxu.

Ve vertikálním módu je možné použít `\hrule`, která má implicitní výšku 0,4 pt, implicitní hloubku 0 pt a implicitní šířku má stejnou, jako je šířka boxu, ve kterém se nalézá. Je to tedy typicky vodorovná linka, která se natahuje na šířku vnějšího boxu.

Implicitní hodnoty je možné potlačit hodnotami specifikovanými za slovy `height`, `depth` a `width`. Takže plně určená `\vrule` vypadá takto:

```
\vrule height<výška> depth<hloubka> width<šířka> % například:
\vrule height7pt depth3pt width30pt % obdélník 10x30 pt: 
```

Parametry je možné psát v libovolném pořadí a libovolný parametr může chybět (pak se použije implicitní hodnota).

Následující příklad vykreslí obrys `<textu>`.

```
\def\obrys#1{\vbox{\hrule\hbox{\vrule#1\vrule}\hrule}}
\obrys{tento text bude obtažen linkou}
```

Na výstupu dostaneme: `tento text bude obtažen linkou`, což asi není zcela uspokojivé. Chtěli bychom nelepit čáry těsně k textu, ale nechat drobný odstup, třeba 1 pt. V takovém případě už makro bude složitější:

```
\def\obrys#1{\vbox{\hrule
                    \hbox{\vrule
                        \vbox{\kern1pt
                            \hbox{\kern1pt #1\kern1pt}%
                            \kern1pt}%
                        \vrule}%
                    \hrule}%
}
\obrys{čára kolem textu má odstup 1 pt}
```

Dostáváme: `čára kolem textu má odstup 1 pt`. To nás asi taky neuspokojí, protože text nerespektuje účaři. Pochopitelně: vnější `\vbox` má poslední element `\hrule` a tímto posledním elementem prochází účaři vytvořeného boxu. Nicméně jako procvičení tvorby složených boxů a využití linek ty příklady dobře posloužily. Rozmyslete si, proč nelze použít jen dva do sebe vnořené boxy, když chceme implementovat odstup čáry od textu.

Další vylepšení tohoto makra předvede technologii vložené podpěry (tzv. `\strut`). Je to neviditelná svislá linka (má tedy šířku 0 pt), která má určenou výšku a hloubku tak, aby přesáhla všechny běžné výšky a hloubky písmen v textu. Její účel je podpírat hranice boxu (jako neviditelný obr Atlás), tedy vynutit výšku a hloubku vnějšího boxu podle jejích rozměrů. Tím budou mít všechny obrysy stejnou výšku i hloubku a v makru se můžeme pomocí záporného `\kern` vrátit na účaři.


```

\def\obrys#1{\vbox{\hrule
                    \hbox{\vrule\kern1pt #1\strutrule\kern1pt\vrule}
                    \hrule
                    \kern-3.4pt}}
\def\strutrule{\vrule height8pt depth3pt width0pt} % podpěra
\obrys{podepřený text}

```

Nyní dostáváme `podepřený text`, což už je docela uspokojivý výsledek. Makro `\strutrule` je podpěra výšky 8 pt a hloubky 3 pt, která způsobí, že `\hbox`, který tuto podpěru obsahuje, bude mít stanovenou výšku a hloubku. Tím je vytvořeno dostatek místa nad a pod textem, takže stačí vložit `\kern1pt` vlevo a vpravo od textu. Pomocí závěrečného `\kern-3.4pt` se vracíme na účarí. Je totiž známo, že hloubka je 3 pt a tloušťka čáry `\hrule` je 0,4 pt, dohromady tedy 3,4 pt.

● **Cvičení 15** ● Kdykoli napíše uživatel `\obrys{cosi}` uvnitř odstavce, dostane obtažený text (který samozřejmě nepodléhá řádkovému zlomu). Na krátké texty to postačí. Vyzkoušejte makro `\obrys` použít zcela na začátku odstavce. Zdůvodněte, co se stalo, na základě znalostí ze sekce 4.4 a z úvodu této kapitoly. Jak makro opravit? Definujte `\def\obrys#1{\leavevmode...}` (a dále už stejně jako dříve).

9.5 Další manipulace s boxy

V této sekci je uveden stručný přehled dalších možností při práci s boxy. Čtenář tak může získat povědomí o vlastnostech \TeX u v souvislosti s boxy a bude rozumět některým konstrukcím v hotových makrech. Pro hlubší pochopení ovšem bude potřeba čerpat informace z TBN, sekce 3.5.

Stejně jako `\vbox{< sazba >}` pracuje příkaz `\vtop{< sazba >}`. Rozdíl je pouze v tom, že účarí výsledného boxu neprochází posledním boxem v `< sazba >`, ale prvním boxem. Takže celý zbytek `< sazby >` se shromáždí pod účarím a vymezuje hloubku výsledného boxu.

Kromě `\vbox{< sazba >}` a `\vtop{< sazba >}` je možné použít (ovšem pouze v matematickém módu) `\vcenter{< sazba >}`. To vytvoří box stejně jako `\vbox{< sazba >}`, ale účarí výsledného boxu prochází mírně pod jeho středem tak, že případné znaky $+$ a $-$ v okolním vzorci mají vodorovnou osu shodnou s vodorovnou osou boxu `\vcenter`.

Boxy (`\hbox`, `\vbox` a `\vtop`) se ve vnějším prostředí chovají jako písmena (v horizontálním módu) nebo jako řádky (ve vertikálním módu). Je možné je ale „vysunout z řady“ pomocí `\lower`, `\raise` (v horizontálním módu) a `\moveleft` a `\moveright` (ve vertikálním módu). Ve všech případech je prvním parametrem příkazu velikost, o kolik posunout, a dále následuje box. Příklad: `\lower.5ex\hbox{E}` je součástí definice loga \TeX u. Totéž by šlo napsat jako `\raise-.5ex\hbox{E}`.

Boxy je možné ukládat do očíslovaných registrů pomocí `\setbox<číslo>=\hbox{...}` (nebo `\vbox`, `\vtop`). V místě vytvoření boxu se v `sazbě` v takovém případě nic neobjeví. Později je možno z paměti přesunout box do `sazby` pomocí `\box<číslo>`. Tím se hodnota registru vyprázdní. Pokud chcete hodnotu registru použít opakovaně, je třeba psát `\copy<číslo>`. Čísla se používají v rozsahu 0 až 9, pro další čísla je vhodné použít alokační makro `\newbox<sekvence>` a psát `\setbox<sekvence>`, `\box<sekvence>`, atd.

Boxům uloženým v registrech je možné změřit (nebo i nastavit) jejich výšku příkazem `\ht<číslo>`, jejich hloubku příkazem `\dp<číslo>` a jejich šířku příkazem `\wd<číslo>`. S jejich využitím plain \TeX definuje makro `\smash{<text>}`, které vytvoří `\hbox{<text>}`, ale s nulovou výškou a hloubkou.

V následujícím příkladě vytvoříme makro `\shiftacute<znak>`, které umístí akcent acute (tj. čárku) nad `<znak>`, přitom ji posune od středu doprava o `\shiftacuteright` a od obvyklé pozice nahoru o `\shiftacuteup`.


```

\newdimen\shiftacuteright \newdimen\shiftacuteup
\def\shiftacute#1{\setbox0=\hbox{#1}%
\leavevmode
\ vbox{\hbox to\wd0{\hss \kern2\shiftacuteright \char19 \hss}%
\kern-1ex \kern\shiftacuteup \nointerlineskip
\box0}}

```

Například při nastavení `\shiftacuteright=-.14em` a `\shiftacuteup=0pt` vytvoří `\shiftacute` b písmeno \acute{b} . Ukázka si zaslouží podrobnější vysvětlení. Nejprve jsou pomocí `\newdimen` deklarovány registry `\shiftacuteright` a `\shiftacuteup`, které pojmu metrický údaj. Na začátku makra `\shiftacute` je `\langle znak \rangle` vložen do `\box0`. Potřebujeme mu totiž změřit šířku. Sestavení akcentu probíhá ve `\vboxu`, který obsahuje nejprve `\hbox` stejné šířky, jako má `\langle znak \rangle`. Uvnitř tohoto prvního boxu je akcent (znak z pozice 19 fontu Computer Modern) centrován pomocí `\hss`, ale vlevo od akcentu je vložen dvojnásobný kern `\shiftacuteright`, který vychyluje centrování o požadovaný údaj doprava. Pod `\hboxem` je `\kern-1ex`, protože akcent samotný je ve fontu nakreslen ve výšce 1 ex. Dále je vložen `\kern`, který posunuje akcent nahoru. Makrem `\nointerlineskip` je zabráněno, aby se následující box umístil s účařím podle `\baselineskip`. Místo toho bude následující box vložen bez jakékoli další vertikální mezery. Nakonec je umístěn `\box0` z paměti, v něm je `\langle znak \rangle`. Je vhodné poznamenat, že příkaz `\accent` (zmiňovaný v sekci 3.2 v cvičení 7) umísťuje akcenty jen centrované (a případně s ohledem na skloněnou osu fontu). Chcete-li mít nad akcenty větší kontrolu, je nutné použít makro podobné makru v této ukázce.

Obsahy boxů uložených v registrech se dají zpětně vyzvrátit do sazby pomocí příkazu `\unhbox\langle číslo \rangle` (vrátí obsah `\hboxu`) nebo `\unvbox\langle číslo \rangle` (vrátí obsah `\vboxu`). Na rozdíl od `\box\langle číslo \rangle` uvedené operace do sazby nevloží box jako celek, ale jeho obsah (tj. jednotlivé elementy sazby, které jsou uschovány uvnitř boxu).

Obsah `\vboxu` uloženého v registru lze krájet pomocí příkazu `\vsplit` do `\vboxů` specifikované výšky. Obsah `\hboxu` uloženého v registru je možno rozdělit do řádků odstavce pomocí `\unhbox\langle číslo \rangle\par`.

Příkaz `\leaders\hrule\hfil` vytvoří pružnou mezeru stejně jako `\hfil`, ovšem nebude vyplněna prázdným místem, ale linkou o výšce 0,4 pt. Tato čárová mezera se chová stejně jako obvyklá mezera, tj. nejen pruží, ale podléhá též řádkovému zlomu. Je-li třeba umístit čáru jinak než na účaři, stačí specifikovat výšku a hloubku `\hrule`. Například `\leaders\hrule height3pt depth-2.5pt \hfil` vytvoří linku ve vzdálenosti 2,5 pt nad účařím a tloušťce 0,5 pt. Místo `\hfil` je možné specifikovat mezeru libovolných parametrů pomocí `\hskip\langle velikost \rangle plus\langle roztažení \rangle minus\langle stažení \rangle`. Když napíšete `\vskip`, vytvoříte vertikální mezeru.

Příkaz `\leaders` může mít jako svůj první parametr místo `\hrule` nějaký box. Obsah tohoto boxu pak je v mezeře opakován těsně vedle sebe tolikrát, aby byla mezera tímto opakováním boxem vyplněna. Například `\leaders\hbox to5pt{\hss.\hss}\hfill` vytvoří tečky, které se typicky používají v obsahu. Od této funkce je příkaz pojmenován: (leaders..... vedou oči). Pro vyplňování mezer tečkami slouží také makro `\dotfill`.

Příkaz `\halign{\langle deklarace \rangle \cr \langle data \rangle \cr}` vytvoří tabulku boxů tak, že v každém sloupci tabulky jsou boxy daného sloupce roztaženy na šířku nejširšího z nich. Boxy jsou plněny `\langle daty \rangle`, přitom každý box jednou položkou `\langle dat \rangle`. Položky v `\langle deatce \rangle` uživatel odděluje znakem `&` a nový řádek položek zahájí pomocí `\cr`. V `\langle deklaraci \rangle` je uveden způsob, jak jsou plněny položky v jednotlivých sloupcích. Tento příkaz nabízí velké možnosti, ale je uživatelsky poměrně komplikovaný a je mu věnována celá sekce 4.3 v TBN.

Pro ilustraci uvedeme druhou možnost, jak vytvořit makro `\shiftacute`, tentokrát pomocí `\halign`.

```

\def\shiftaccute#1{\leavevmode\vbox{\offinterlineskip
\halign{\hfil##\hfil\cr
\kern2\shiftacuteright\hbox to0pt{\hss\char19\hss}\cr
\noalign{\kern-1ex\kern\shiftacuteup}#1\cr}}}

```

Toto makro dělá to samé, jako první verze makra `\shiftacute`. Idea je nyní postavena na tom, že `\halign` v tomto případě sestaví tabulku s jedním sloupcem a se dvěma boxy pod sebou. Boxům nastaví šířku nejširšího z nich, tj. znaku na druhém řádku tabulky.

Je vidět, že přímé použití příkazu `\halign` nelze asi očekávat od laických autorů textů, kteří se nechtějí zatěžovat nějakými deklaracemi pomocí křížů a pružnými mezerami. Těm je určeno makro `\table` z OPmac (viz sekci 7.10).

Kapitola 10

Co se nevešlo jinem

10.1 Vertikální mezery a stránkový zlom

Následující přehled obsahuje registry \TeX u, podle kterých se řídí meziřádkové mezerování. Vedle registru je uvedena jeho výchozí hodnota nastavená $\text{plain}\TeX$ em.

```
\baselineskip = 12pt    ... vzdálenost mezi účarími řádků
\lineskip = 1pt        ... mezera, není-li použito \baselineskip
\lineskiplimit = 0pt   ... podmínka, kdy se nepoužije \baselineskip
\parskip = 0pt plus1pt ... dodatečná mezera mezi odstavci
\topskip = 10pt        ... vzdálenost účarí prvního řádku od stropu sazby
\abovedisplayskip = 12pt plus3pt minus9pt ... mezera nad display rovnicí
\belowdisplayskip = 7pt plus3pt minus4pt ... mezera pod display rovnicí
\abovedisplayskip, \belowdisplayskip ... analogie k předchozím
```

Jsou-li řádky na výšku „dostatečně malé“ tak, že se vejdou do \baselineskip , je mezera mezi řádky dopočítána tak, aby vzdálenost účarí byla \baselineskip . Je-li ale tato dopočítaná mezera menší než \lineskiplimit , pak \TeX místo ní použije \lineskip a k \baselineskip pro danou dvojici řádků nepřihlíží. Při nastavení registrů podle $\text{plain}\TeX$ u druhý případ znamená, že se řádky „opírají o sebe“, ovšem ne zcela, je mezi nimi mezera 1 pt. Registr \lineskiplimit lze nastavit na záporné hodnoty, pak bude \baselineskip dodržováno i za cenu toho, že budou některé vyšší části řádků zasahovat do sousedních řádků.

Účarí prvního řádku sazby na stránce¹⁾ je od stropu sazby²⁾ posunuto dolů o \topskip , je-li první řádek na výšku „dostatečně malý“ a nepřesáhne při takovém umístění strop sazby. Jinak se první řádek svou horní hranou opírá o strop sazby.

Makro \offinterlineskip přenastaví hodnoty registrů \baselineskip , \lineskip a \lineskiplimit tak, že se řádky budou opírat jeden o druhý s nulovou meziřádkovou mezerou. To je užitečné například v makrech pro tabulky, kde jsou řádky podepřeny pomocí \strut .

Mezi odstavce je možné vložit další mezery z následujících maker:

```
\bigskip    ... mezera velikosti jednoho řádku
\medskip    ... mezera velikosti poloviny řádku
\smallskip  ... mezera velikosti čtvrtiny řádku
```

Uvedené vertikální mezery obsahují rovněž toleranci k mírnému stlačení nebo roztahení. Libovolnou mezeru mezi odstavci lze vložit pomocí příkazu \vskip . Příkaz (i makra z předchozího přehledu) ukončí nejprve případný rozepsaný odstavec a pod něj vloží požadovanou mezeru.

Mezeru nebo i jiný vertikální materiál lze dopravit mezi řádky odstavce bez toho, aby byl odstavec ukončen, pomocí příkazu $\text{\vadjust}\{<\textit{materiál}>\}$. Tento příkaz umístí do odstavce bezrozměrnou značku, která si pamatuje obsah argumentu. Jakmile je později takový odstavec ukončen a rozlámán na řádky a tyto řádky začne \TeX umisťovat pod sebe, značka přidá pod svůj řádek obsah svého argumentu. Pro ukázkou byl to tohoto odstavce vložen $\text{\vadjust}\{\text{\line{\dotfill}}\}$ na konec první věty.

¹⁾ Je tím míněn první řádek běžného textu, záhlaví se do toho nezapočítává.

²⁾ Strop sazby je místo na stránce umístěné ve vzdálenosti \offset+1in od horního okraje papíru.

\TeX v horizontálním i vertikálním směru vytváří materiál, ve kterém se zhruba opakuje „box, penalty, glue“, což překládáme jako „box, penalta, mezera“. Ve vertikálním směru je střídání těchto tří objektů skoro pravidelné: box tvoří řádek, pod ním je penalta a pod ní je mezera. Pak následuje další řádek, další penalta a další mezera atd.

Makra vkládají penaltu do sazby příkazem `\penalty<číslo>` a dále \TeX vloží automaticky před mezery mezi řádky penaltu podle hodnot následujících registrů. Vedle registru je v přehledu uvedena výchozí hodnota z nastavení plain \TeX em.

```
\interlinepenalty = 0      % penalta před mezerou mezi řádky odstavce
\clubpenalty = 150        % penalta za prvním řádkem odstavce
\widowpenalty = 150       % penalta za předposledním řádkem odstavce
\brokenpenalty = 100      % penalta za řádkem s rozděleným slovem
\predisplaypenalty = 10000 % penalta před mezerou nad display rovnicí
\postdisplaypenalty = 0    % penalta před mezerou pod display rovnicí
```

Uvedené penalty se pod řádkem sčítají. Je-li například rozdělené slovo v prvním řádku odstavce, bude pod ním penalta rovná součtu z `\interlinepenalty`, `\clubpenalty` a `\brokenpenalty`.

Penalta před mezerou je potenciální trest za zlomení sazby v této mezeře, se kterým interně pracuje algoritmus řádkového i stránkového zlomu. Algoritmus hledá řešení s nejmenším celkovým trestem a chybovostí¹⁾ řádků/sloupců, takže dává přednost zlomu v mezerách s menšími penaltami. Při zlomu mezera ze sazby zcela mizí bez ohledu na to, jak byla veliká. Pokud penalta před mezerou chybí, je to stejné jako nulová penalta a mezera se láme „s normální ochotou“. Kladné hodnoty penalt udávají míru neochoty mezery se zlomit. Hodnota penalty 10 000 a více pak zaručí, že se mezera nezlomí nikdy. Záporné hodnoty penalt zvyšují šanci, že se v této mezeře provede zlom. Hodnota $-10\,000$ a méně značí vynucený zlom, který se provede za jakýchkoli okolností.

Makra `\break`, `\nobreak` a `\allowbreak` jsou definována jako:

```
\def\nobreak {\penalty10000 } % zabrání zlomu v následující mezeře
\def\break {\penalty-10000 }  % vynutí si zlom
\def\allowbreak {\penalty0 }  % umožní zlom
```

\TeX může zlomit v penaltě, i když za ní žádná mezera neexistuje. Takže pro ukončení stránky je možné psát `\vfill\break`. Tady je pružná mezera `\vfill` výjimečně před penaltou a v sazbě zůstane. Vyplní prázdný prostor do konce stránky.

Za zlomem všechny následující mezery a penalty mizí až po první box. Takže pokud chcete zahájit novou stránku například po `\vfill\break` vertikální mezerou, není účinné psát `\vskip2cm`, protože tato mezera zmizí. Je vhodné v tomto případě místo příkazu `\vskip` použít makro `\vglue`, které má stejnou syntaxi i význam jako `\vskip`, ale na začátku stránky nemizí.

\TeX se snaží každou stránku „vypružit“ tak, že je vzdálenost mezi prvním a posledním řádkem vždy stejná a výška této sazby²⁾ je `\vsize`. Když se podíváme na výchozí nastavení plain \TeX u na první přehled v této sekci, vidíme, že veškerá pružnost je soustředěna do `\parskip` (mezi odstavce) a do mezer nad a pod rovnicí. Další pružnost zřejmě dodají makra pro titulky, když použijí `\bigskip`, `\medskip` a `\smallskip`, protože tato makra rovněž implementují do vertikální mezery mírnou pružnost.

Není-li na stránce dostatek pružného materiálu, \TeX ohlásí varování `Underfull vbox while output is active`. Je možné vzdát se požadavku na stejnou výšku sazby na každé

¹⁾ Chybovost neboli *badness* zhruba vyjadřuje stupeň vypružení mezer, tedy míru odchylky od jejich přirozené velikosti v daném řádku nebo sloupci.

²⁾ Měřeno od stropu sazby před mezerou z `\topskip` po účaří posledního řádku.

stránce pomocí makra `\raggedbottom`, které stačí napsat do začátku dokumentu. Od této chvíle bude \TeX každou stránku dole zakončovat (pokud možno co nejmenší) pružnou mezerou a ostatní pružnost mezi řádky zcela zmizí.

Makro `\filbreak` umožní za odstavcem stránkový zlom, při kterém je aktuální stránka doplněna dole vertikální mezerou. Pokud ale se pod `\filbreak` vejde další text, zlom se neprovede a výsledek je takový, jakoby tam žádné `\filbreak` nebylo.

Pro kvalitní sazbu třeba beletrie je žádoucí dodržovat tzv. *řádkový rejstřík*, tj. řádky na stránkách jsou všechny v jednom shodném řádkování a je jich na každé zcela vyplněné straně stejný počet. Lze tušit, že toto je pro sázecí automat dost komplikovaný požadavek, zejména při dodržení dalšího typografického pravidla: neodlamovat první a poslední řádek odstavce, tj. nastavit `\clubpenalty=10000` a `\widowpenalty=10000`. Jak se za této situace dá řešit v \TeX u řádkový rejstřík, je popsáno v TBN na stranách 242 a 243.

Pro programátory maker jsou připravena ještě následující makra `plain \TeX` u, která vkládají mezi odstavce mezery nebo penalty:

```
\goodbreak      % dobré místo zlomu za odstavcem, penaltou -500
\bigbreak       % vloží mezeru jako \bigskip podpořenou penaltou -200
\medbreak       % vloží mezeru jako \medskip podpořenou penaltou -100
\smallbreak     % vloží mezeru jako \smallskip podpořenou penaltou -50
\removelastskip % odstraní naposledy vloženou mezeru
```

Příklad naprogramování titulku:

```
\def\titulek#1\par{\removelastskip\bigbreak
  {\noindent\bf #1\par}\nobreak\medskip}
```

Před titulkem je jednořádková mezeru ochotná se zlomit (`\bigbreak`). Tato mezeru zruší případnou předchozí mezeru (`\removelastskip`). Za titulkem nejprve ukončíme odstavce (bez toho by byl `\nobreak` v horizontálním módu, což nechceme) a pak pomocí `\nobreak\medskip` vytvoříme půlřádkovou mezeru, ve které je zakázáno zlomit sazbu.

OPmac mimoto nabízí pro tvorbu titulků makra `\remskip` a `\norempenalty`. Jejich smysl začne být zřejmý, když budeme chtít pod hlavní titulek dát hned vedlejší titulek a pak teprve text. Pod hlavním titulkem se nesmí zlomit sazba ale nad vedlejším titulkem je naopak doporučeno v obvyklých případech sazbu zlomit. Prioritu má ale první požadavek, což nelze docílit prostým přidáváním mezer a penalt pod sebe. Makro `\remskip` vloží specifikovanou mezeru (jako při `\vskip`) která je pro další užití označena jako *rem*-mezera (určena k případnému odstranění). Následuje-li těsně za tím `\norempenalty<číslo>`, pak se *rem*-mezera odstraní a nová penaltu se nevloží. Pokud ale není před `\norempenalty` žádná *rem*-mezera, pak `\norempenalty<číslo>` vloží penaltu `<číslo>`. Když například pod větším titulkem je `\par\nobreak\remskip 6pt\relax` a nad menším titulkem je `\norempenalty-150\vskip7pt`, pak pokud se pod větším titulkem objeví menší, bude mezeru 6 pt odstraněna a zůstane jen mezeru 7 pt stále nezlomitelná kvůli `\nobreak`. Jiná penaltu se do sazby nevloží. Když ale se menší titulek objeví samostatně, je nad ním penaltu -150 následovaná mezerou 7 pt, která jeví ochotu se zlomit.

10.2 Plovoucí objekty

V odborných textech se často vyskytují tabulky nebo obrázky například ve formě grafů. Ne vždy je možné tabulku či podobný objekt umístit tam, kde se o ní zrovna píše. Typické je dilema, kdy po vložení tabulky stránka přeteče a přesunutím tabulky na další stránku původní stránka zůstává nezaplněna. V odborných textech se na tabulku typicky odkazuje číslem (viz příkaz `\caption` v sekci 7.4). Není tedy nutné, aby byla tabulka přesně tam,

kde se o ní píše. Přesunutím tabulky nebo obrázku „někam poblíž“ je pak možné uvedené dilema vyřešit. \TeX to dělá automaticky, pokud použijete makro:

```
\midinsert
<tabulka nebo obrázek včetně popisku>
\endinsert
```

Makro se snaží nejprve umístit tabulku nebo obrázek tam, kde je. Pokud se do strany nevejde, je tabulka nebo obrázek přesunuta na začátek příští strany, ale text pokračuje na stávající stránce. Místo `\midinsert` lze také použít `\topinsert`, což umístí tabulku vždy na začátek stránky (bez ohledu na to, kde je umístěna ve zdrojovém textu). Objektům vytvořeným pomocí `\midinsert` nebo `\topinsert` říkáme *plovoucí objekty*, protože jejich umístění v sazbě není přesně stanoveno. Při využití maker `\table`, `\caption`, `\label` a `\ref` z OPMac odborné texty typicky vypadají takto:

```
... jak je vidět z tabulky~\ref[zavislosti].
\midinsert
  \centerline{\table{rl}{věk  & hodnota ... \cr
                      ...    & ... }}
  \medskip
  \label[zavislosti]
  \caption/t Závislost závislosti na počítačích na věku.
\endinsert
Kromě toho se výzkum zabýval...
```

Vejde-li se plovoucí objekt do textu, makra přidají nad a pod objekt mezeru `\bigskip`. Rovněž objekt na začátku strany je od dalšího textu oddělen mezerou `\bigskip`.

10.3 Dekorace stránek, výstupní rutina

Tiskový materiál, který \TeX odlomil při stránkovém zlomu, je zabalen do boxu a zaslán výstupní rutině (`\output`), ve které může programátor maker přidat dekoraci, která se na každé straně víceméně opakuje: záhlaví, zápatí, barevný podklad atd. Programování výstupních rutin daleko překračuje rámec tohoto úvodního textu. Zde se pouze naučíme používat základní prvky výstupní rutiny, která je připravena v `plain \TeX` . Tato rutina vytvoří stránku zhruba ze tří objektů: box nahoře pro záhlaví, pod ním box se sazbou a pod ním box pro zápatí. Záhlaví i zápatí mají své boxy v šířce `\hsize` a jsou od sazby odděleny vhodnými vertikálními mezerami.

Pro obsah boxu záhlaví je připraven registr `\headline` a pro obsah boxu zápatí registr `\footline`. Oba registry jsou typu *(tokens)* a jsou v `plain \TeX` naplněny těmito výchozími hodnotami:

```
\headline={\hfil}
\footline={\hss\tenrm\folio\hss}
```

Vidíme tedy, že záhlaví je implicitně prázdné (`\hfil`) a v zápatí se uprostřed (mezi dvěma `\hss`) vytiskne `\folio`, což je makro expandující na číslo strany `\the\pageno`. Pokud nechcete tisknout stránkovou číslici, stačí psát:

```
\footline={\hfil}
```

Pro toto nastavení má `plain \TeX` zkratku `\nopagenumbers`. Ta se hodí použít například na začátku knihy, kde jsou stránky s titulem, obsahem atd. Tam číslo strany chybí, ale strana je započítána. Jakmile později napíšete `\footline={\hss\tenrm\folio\hss}`, nebude první vypsané číslo mít sice hodnotu jedna, ale to je z typografického pohledu správně.

Pokud chcete mít v záhlaví nějaký stálý text, pište třeba:

```
\headline={\tenit Stejný text v záhlaví na každé straně\hss}
```

Jestliže si přejete mít záhlaví odděleno čarou (to je obvyklá úprava záhlaví), lze psát:

```
\headline={\tenit Text záhlaví\strut\vadjust{\hrule}\hfil}
```

Chcete-li mít jiné záhlaví na levých stránkách a jiné na pravých (lichých), můžete vyzkoušet třeba toto:

```
\headline={\strut\vadjust{\hrule}\tenit  
  \ifodd\pageno \hfil \chaptitle\else Titul knihy\hfil\fi}
```

Tento příklad vyžaduje, aby makro pro sazbu titulku kapitoly uložilo nejdříve text titulku do pomocného makra `\chaptitle`.

Další příklad ukazuje, jak zařídit, že budou čísla stran na levých stránkách vlevo a na pravých (s lichými čísly) vpravo:

```
\footline={\tenrm \ifodd\pageno \hfill \fi \the\pageno \hfil}
```

Tento příklad si zaslouží drobné dovysvětlení. Je-li nejprve stránka lichá, je vytištěno `\hfill \the\pageno \hfil`, ale protože má `\hfill` nekonečně větší pružnost než `\hfil`, mezera vpravo splaskne a vlevo se natáhne. Je-li naopak stránka sudá, je vytištěno `\the\pageno\hfil` a mezera vpravo se natáhne.

Chcete-li dopravit do záhlaví titulky sekcí, o kterých nevíte přesně, na kterých se vyskytnou stranách (protože nejsou odděleny `\vfill\break`), je potřeba v makru pro titulek sekce použít příkaz `\mark{<text>}`, který uloží `<text>` do paměti T_EXu. V záhlaví (tj. v `\headline`) je pak možné využít některý z těchto příkazů:

```
\topmark    % expanduje na <text> posledního \mark předchozí strany  
\firstmark  % expanduje na <text> prvního \mark na této stránce  
\botmark    % expanduje na <text> posledního \mark této stránky
```

Není-li na sledované straně žádný `\mark`, příkazy expandují na poslední použitý `\mark` z předchozích stran.

OPmac implicitně plovoucí záhlaví neřeší. Je ovšem možné se inspirovat OPmac trikem¹⁾, ve kterém je tvorba plovoucího záhlaví popsána.

Někdy existuje požadavek napsat titulky v záhlaví velkými písmeny. K tomu lze použít příkaz `\uppercase{<text>}`, který promění `<text>` na text s velkými písmeny. Pokud se v `<textu>` vyskytují řídicí sekvence, nejsou konverzí dotčeny. Je-li například v makru `\chaptitle` připraven text titulku malými písmeny, je nutné toto makro pro příkaz `\uppercase` nejprve expandovat, tedy psát `\uppercase\expandafter{\chaptitle}`.

Analogicky jako `\uppercase` funguje také příkaz `\lowercase{<text>}`, který proměňuje `<text>` na malá písmena. Vzhledem k tomu, že pravidlo tohoto zobrazení je možné pro každý znak zvlášť specifikovat příkazem `\lccode'<vzor>='<obraz>`, používá se často příkaz `\lowercase` v makrech na nejrůznější konverze textů, ne nutně na konverzi na malá písmena. Analogický příkaz k `\lccode` je `\uccode'<vzor>='<obraz>`, kterým lze specifikovat konverzi znaků při použití `\uppercase`.

V zahraniční odborné literatuře nebo ve studentských závěrečných pracích jsou často úvodní stránky číslovány římskými čísly a hlavní text je číslován arabsky znovu od strany 1. K tomu účelu slouží makro `\folio` plainT_EXu, které expanduje na římskou podobu čísla, je-li registr `\pageno` záporný, a na arabskou podobu, je-li `\pageno` kladné.

¹⁾ <http://petr.olsak.net/opmac-tricks.html#headline>

Výstupní rutina plainTeXu během přechodu na další stránku při záporném `\pageno` odečítá jedničku a při kladném přičítá jedničku. Na počátku úseku číslovaném římsky tedy stačí psát `\pageno=-1` a na počátku úseku číslovaném arabsky pak `\pageno=1`. Makro `\folio` je definováno takto:

```
\def\folio{\ifnum\pageno<0 \romannumeral-\pageno \else \the\pageno \fi}
```

Pro konverzi na římské číslování je použit místo `\the` příkaz `\romannumeral` (*číslo*). Ten vypíše číslo malými písmeny. Chcete-li mít číslo vytištěno velkými písmeny, je třeba psát `\uppercase\expandafter{\romannumeral-\pageno}`.

OPmac i CSpain umožňují poměrně snadno zvětšovat a zmenšovat fonty, jak bylo ukázáno v sekcích 5.4 a 7.1. V takovém případě `\tenrm` nebo `\tentit` může mít v dokumentu na různých místech různou velikost (základní font, citace, drobné postřehy atd.). Není ovšem správné, aby se podle toho měnila velikost písma pro číslo strany nebo záhlaví na různých stránkách. Doporučujeme tedy místo `\tenrm` a `tenit` v kódech pro `\headline` a `\footline` použít `\fixrm` a `\fixit` definované pomocí `\let\fixrm=\tenrm` a `\let\fixit=\tenit` na začátku dokumentu. Nebo (při použití OPmac) je možné psát ve `\footline` a `\headline` například `\tenrm\thefontsize[10]`.

Někdy je součástí typografického návrhu pravidlo, že kapitoly začínají vždy na pravé straně v otevřené knize i za cenu toho, že na levé straně vznikne prázdná stránka, neboli *vakát*. Vakát pak musí být dle typografických pravidel zcela prázdný bez záhlaví i bez čísla strany. Toto pravidlo někteří lidé z neznalosti nedodržují, ale my to dodržovat budeme. Navrhujeme makro `\nextoddpag`, které odstraní stránku a pokud by následující strana byla sudá, přidá navíc vakát:

```
\def\nextoddpag {\vfill\break % odstránkování
\ifodd\pageno \else % jsme-li na sudé straně
{\footline={\hfil} % lokálně ve skupině bez čísla stránky
\headline={\hfil} % a bez záhlaví
\hbox{} \vfill\break} % pošleme ven prázdný box
\fi}
```

10.4 Čtení a zápis textových souborů

TeX začíná číst *hlavní soubor* podle parametru na příkazovém řádku. V tomto souboru se může objevit příkaz `\input` (*soubor*) (přímo nebo v makru). V takovém místě začíná TeX číst specifikovaný *soubor*. V něm se může znovu vyskytnout `\input` atd., TeX se tedy může zanořit při čtení i do dalších souborů. Na konci souboru nebo v místě příkazu `\endinput` TeX ukončí čtení daného souboru a pokračuje ve čtení souboru nadřazeného. To provádí až do dosažení příkazu `\end`, který způsobí ukončení činnosti TeXu. Místo `\end` se často používá makro `\bye`.

Nenarazí-li TeX na `\end` a dokončí čtení hlavního souboru, přechází na čtení z terminálu. Očekává tedy interakci uživatele, který mu může psát další příkazy (nebo části dokumentu) „ručně“ a ukončí to příkazem `\end`. To není příliš obvyklé, takže je daleko vhodnější na `\end` nebo `\bye` v dokumentu nezapomínat.

Kromě výstupu na terminál, do `.log` souboru a do PDF resp. DVI souboru je TeX schopen zapisovat textové informace do dalších specifikovaných souborů. Typicky to dělá proto, aby tyto soubory mohl při následujícím spuštění načíst a dozvědět se tak některé údaje z předchozího běhu. Důvody pro tuto techniku jsou nejčastěji dva:

- Dopředné odkazy nelze zpracovat bez znalosti cíle odkazu.
- Odkazy na stránky nelze zpracovat přímo.

Oba důvody rozebereme podrobněji. Důvod první: dopředným odkazem je odkaz na místo v dokumentu, které je uvedeno později než odkaz. V místě cíle odkazu je slovníkovým způsobem (podobně jako v ukázce v sekci 8.6) propojen `<lejblík>` s vygenerovaným číslem (sekce, kapitoly, tabulky atd.). Jenomže v místě odkazu ještě není toto číslo známo, tam je znám jen `<lejblík>`. Je tedy potřeba, aby \TeX ukládal tyto slovníkové údaje (propojení mezi `<lejblíkem>` a číslem) do pracovního souboru a v následujícím zpracování je nejprve načtl a uvědomil si tím data pro všechny použité `<lejblíky>`. Pozorování: pokud bychom používali jen zpětné odkazy (na místa již dříve vytištěná), není nutné externí soubor používat.

Důvod druhý: jak bylo vysvětleno v sekci 4.4, \TeX zpracovává řádek odstavce v jiném čase, než kdy ukládá rozlomené řádky odstavce do sloupců, které pak zalamuje do stran. Zpracování probíhá asynchronně. V době expanze maker a zpracování textu v odstavci se ještě vůbec neví, na jakou stranu se dané místo textu dostane. Nelze tedy místo v textu propojit s číslem strany pomocí maker napsaných v textu.

Pro tyto účely \TeX disponuje asynchronním příkazem `\write<soubor>{<text>}`, který v místě použití uloží do sazby jen neviditelnou značku. Ta se probudí k činnosti až v době, kdy výstupní rutina ukládá kompletovaný box se sazbou strany do PDF nebo DVI. To provede výstupní rutina příkazem `\shipout`. V tuto chvíli je už jasné, jak vypadá registr `\pageno` označující číslo strany. Probuzené značky vytvořené v sazbě strany pomocí `\write` teprve nyní expandují `<text>` a ukládají jej do výstupního textového souboru. V argumentu `<text>` příkazu `\write` se typicky (mimo jiné) vyskytuje `\the\pageno`, což po expanzi v pravou chvíli (až v době `\shipout`) dá správné číslo strany.

Od těchto starostí je uživatel odstíněn, pokud použije OPMac. Na druhé straně aspoň základní povědomí o způsobu vytváření odkazů, obsahů a rejstříků je dobré mít třeba proto, že si budete chtít nějaké makro přečíst, porozumět mu a upravit podle vlastní představy. Také je tím vysvětleno, proč je nutné někdy dokument \TeX ovat vícekrát. Je-li obsah dokumentu vpředu, pak je nutné \TeX ovat dokonce třikrát: po prvním běhu je přední stránka s obsahem prázdná. Po druhém běhu je několik stran s obsahem vytvořeno, ale číslování stran se celkově kvůli tomu posunulo, takže obsah odkazuje na nesprávné stránky. Po třetím běhu je teprve vše v pořádku.

Ukážeme si nástroje, které se používají v makrech v souvislosti s `\write`. Pomocí makra `\newwrite<sequence>` se deklaruje `<sequence>` jako identifikátor souboru pro zápis. Pomocí příkazů `\immediate\openout<sequence>=<název souboru>` se soubor připraví k zapisování. Pokud na disku existoval, vymaže se a založí nově prázdný. Pak mohou následovat příkazy `\write<sequence>{<text>}`, které zapisují do stanoveného souboru `<text>`. Tento zápis proběhne se zpožděním až v okamžiku, kdy je strana zkompletována výstupní rutinou. Není-li toto zpoždění žádoucí, je možné před `\write` napsat prefix `\immediate`.

Techniku práce s externím souborem ukážeme na jednoduchém příkladě sestavení obsahu. Předpokládejme, že titulky kapitol jsou rozmístěny kdekoli na stránkách (ne nutně na začátku stránky) a tudíž není v době sazby titulku jasné, na které straně se titulek objeví. Je tedy nutné použít externí soubor a `\write` se zpožděním. Založíme soubor `\jobname.toc`, tedy soubor se stejným názvem jako dokument, ale s příponou `.toc`, a jednotlivé jeho řádky budou obsahovat `\udaj{<text titulk>}{<číslo strany>}`.

```
\newwrite\tocfile
\def\udaj#1#2{\line#1 \dotfill\ #2} % příprava ke čtení souboru
\noindent{\bf Obsah}\par\medskip % titulek obsahu
\softinput \jobname.toc % načtení pracovního souboru a vytvoření obsahu
\def\titulek#1\par{\bigbreak \noindent
  \write\tocfile{\string\udaj{#1}{\the\pageno}}% zápis údajů do souboru
  {\bf #1}\par\nobreak\medskip % tisk titulku
```

```

}
\immediate\openout\tocfile = \jobname.toc % otevření souboru k zápisu

... dokument
\end

```

Na této ukázce je důležité pořadí, v jakém jsou jednotlivé úkoly řešeny. Nejprve je definováno makro `\udaj`, které se vyskytuje v pomocném souboru. Pak je vytvořen obsah tím, že je pomocný soubor přečten (pokud z předchozího běhu existuje). Pak je pomocí `\immediate\openout` pomocný soubor promazán a připraven k zápisu. Nakonec se při čtení dokumentu do tohoto souboru zapisuje jednotlivými `\write`, které jsou součástí makra pro titulky kapitol. Toto pořadí činností nelze měnit.

Chceme-li tisknout obsah dokumentu na jeho konci, musíme si obsah souboru na začátku zpracování zapamatovat a poté psát `\immediate\openout`, dále přečíst dokument a v závěru vyvrhnout obsah z paměti. Pro zapamatování obsahu může sloužit nějaké pomocné makro nebo box. Například:

```

\newbox\obsahbox
% ... další deklarace a definice \udaj, \titulek jsou stejné
\setbox\obsahbox = \vbox {\softinput \jobname.toc } % uložení dat do boxu
\immediate\openout\tocfile = \jobname.toc % otevření souboru k zápisu
... dokument
\bigskip\noindent{\bf Obsah}\par\nobreak\medskip % titulek obsahu
\unvbox\obsahbox % tisk zapamatovaného obsahu z boxu

```

Pokud máme jistotu, že obsah na konci dokumentu je na samostatné stránce, můžeme si přečíst pomocný soubor ve stejném běhu \TeX u. K tomu je potřeba nejprve zapisovaný soubor `\tocfile` uzavřít příkazem `\immediate\closeout\tocfile` a následně se do něj pustit příkazem `\input`:

```

% ... deklarace a definice \udaj, \titulek jsou stejné
\immediate\openout\tocfile = \jobname.toc % otevření souboru k zápisu
... dokument
\vfil\break % ukončení poslední strany před obsahem
\immediate\closeout\tocfile % uzavření pracovního souboru
\noindent{\bf Obsah}\par\medskip % titulek obsahu
\input \jobname.toc % načtení pracovního souboru a vytvoření obsahu

```

V předchozích ukázkách jsme se dopustili jedné nepřesnosti. Nebylo vysvětleno ani definováno makro `\softinput`. Toto makro přečte soubor jen tehdy, pokud soubor existuje. Makro není součástí plain \TeX u ani O Π mac, ale je možné je převzít z TBN ze strany 288:

```

\newread\testin
\def\softinput #1 {\let\next=\relax \openin\testin=#1
  \ifeof\testin \message{Warning: the file #1 does not exist}%
  \else \closein\testin \def\next{\input #1 }\fi
\next}

```

Je v zásadě jedno, zda neviditelnou značku, vytvořenou příkazem `\write`, umístíte nad řádek s titulkem, do řádku s titulkem nebo pod něj. V ukázce na předchozí stránce ji makro `\titulek` vkládá do řádku jako první objekt řádku hned po `\noindent`. Pokud ji chcete mít nad řádkem, je nutné za ni napsat `\nobreak`, protože následně se vloží mezera podle `\parskip` a `\baselineskip` a ta nesmí podlehnout stránkovému zlomu. Jinak by stránkový odkaz obsahoval chybnou stránku. Pokud ji chcete mít pod řádkem, pak ji vložte okamžitě za příkazem `\par` a pak teprve přidávejte `\nobreak\medskip`. Příklady:

```
% nad řádkem:
\bigbreak
\write\tocfile{...}\nobreak
\noindent ... Titulek ... \par
\nobreak\medskip

% pod řádkem:
\bigbreak
\noindent ... Titulek ... \par
\write\tocfile{...}
\nobreak\medskip
```

Příkaz `\write` při zápisu do souboru svůj parametr plně expanduje. Pokud se tedy v titulku objeví nějaké makro, do souboru se toto makro „rozsype“ a může působit potíže při opakovaném čtení. OPmac tento problém řeší pomocí deklarace `\addprotect{makro}`, viz sekci 7.6.

10.5 Ladění dokumentu, hledání chyb

Možnosti hledání chyb v dokumentu nejsou v \TeX u bohužel příliš komfortní. Často se setkáváme se zavlečenými chybami, kdy zpracování zkolabuje poněkud později, než v místě, kde se autor dopustil chyby. Uživatel si musí postupně zvyknout na hlášení, která \TeX o zpracování podává na terminál a do `.log` souboru.

Při chybě se běh \TeX u typicky zastaví, zobrazí se řádek dokumentu rozdělený na část přečtenou a nepřečtenou a také jsou vypsány obsahy maker, která v místě chyby expandují. Rozsah tohoto zobrazení lze ovlivnit hodnotou registru `\errorcontextlines`. Dále se na terminálu objeví otazník. Na to může uživatel interaktivně reagovat, typicky zmáčkne Enter a tím poběží \TeX dále. Odpoví-li na otazník otazníkem, dozví se, jaké má další možnosti. Za zmínku stojí možnost napsat `h` a dozvědět se, co si o chybě myslí \TeX . Písmeno `x` ukončí zpracování dokumentu \TeX em. Písmeno `s` zařídí, že se \TeX přestane na dalších chybách zastavovat. Zastavování na chybách je možné vypnout též vložním příkazu `\scrollmode` do dokumentu nebo vhodným přepínačem na příkazovém řádku.

Je dobré porozumět typickému způsobu oznamování chyb. Ukážeme si to na příkladu. Dejme tomu, že jste napsali odstavec, ve kterém je lichý počet znaků `$`. Tedy na konci odstavce je \TeX v matematickém módu, což je špatně, protože uvnitř matematického módu nesmí končit odstavec. \TeX vám to takto polopaticky neoznámí, místo toho řekne:

```
! Missing $ inserted.
<inserted text>
$
<to be read again>
\par
1.42
?
```

Můžete si to nechat více objasnit tím, že vložíte písmeno `h`, ale moc dalšího světla to nemusí přinést:

```
? h
I've inserted a begin-math/end-math symbol since I think
you left one out. Proceed, with fingers crossed.
```

Je tedy třeba si zvyknout na to, že \TeX obvykle nepíše o příčině chyby, ale o způsobu, jak se s ní chce vyrovnat. V uvedeném příkladě narazil na prázdný řádek pod odstavcem, ten

řádek je interpretován jako sekvence `\par`. \TeX shledal, že pokud má pokračovat dále, musí vložit ukončovací znak matematického módu `$` před tuto sekvenci `\par`. V rámci chyby se často vypisuje zpráva `<to be read again>`, za kterou je příkaz, na který \TeX v rámci chyby narazil a který odložil v rámci zotavení se z chyby do vstupní fronty. Jak oznamuje, vloží do vstupní fronty `$` a poté znovu přečte `\par`. K problému došlo na (prázdném) řádku 42. Toto je typický způsob uvažování \TeX u při výskytu chyby.

Je dobré si všimnout i varování `Overfull/Underfull \hbox/\vbox`. Pokud je box podtečený (`Underfull`), znamená to, že mezery v boxu jsou nataženy více, než by odpovídalo jejich přirozené toleranci a je překročena jistá estetická mez. Závažnější problém je přetečení boxu (`Overfull`). Na příslušném místě se objeví černý obdélníček a řádek přechází ze sazby. Není-li zbytí, je možné převést problém `Overfull` na `Underfull` nastavením registru `\emergencystretch` na dostatečně velký rozměr, viz sekci 4.3.

Uvedením makra `\tracingall` v dokumentu lze docílit toho, že \TeX o sobě do terminálu i do `.log` souboru řekne vše, co dělá. Je to ale velmi nepřehledné a obsáhlé. Je také možné zapnout jen některé části tohoto výpisu nastavením následujících registrů na kladnou hodnotu:

<code>\tracingcommands</code>	% všechny primitivní příkazy, které \TeX vykonává
<code>\tracingmacros</code>	% expanze všech maker
<code>\tracingparagraphs</code>	% vnitřnosti algoritmu zlomu řádků
<code>\tracingpages</code>	% vnitřnosti algoritmu zlomu strany
<code>\tracingoutput</code>	% obsahy boxů vystupujících do výstupu
<code>\tracingrestores</code>	% údaje o registrech měnících hodnoty na konci skupiny
<code>\tracingstats</code>	% údaje o využití \TeX ovské paměti
<code>\tracingonline</code>	% co píše do <code>.log</code> souboru, bude vypisovat i na terminál

Výpisy obsahů boxů jsou omezeny hloubkou vnoření boxů a délkou tiskového seznamu. Obojí se dá zvětšit pomocí nastavení registrů `\showboxdepth` a `\showboxbreadth` na dostatečně velkou hodnotu. Výpis obsahu jednotlivého boxu se provede příkazem `\showbox<číslo boxu>`. Výpis vytvořeného tiskového materiálu v aktuálním místě sazby lze získat příkazem `\showlists`. Příkaz `\show<token>` vypisuje význam `<tokenu>`. Je-li `<token>` makrem, dozvíme se i obsah makra. Příkaz `\showthe<registr>` vypíše hodnotu registru. Při práci s příkazy `\showbox`, `\showlists`, `\show` a `\showthe` je praktické mít nastaven `\tracingonline` na kladnou hodnotu.

Při ladění maker se často používají i ladicí tisky pomocí `\message{<text>}`. Příkaz vypíše svůj argument do `.log` souboru i na terminál expandovaný. V argumentu příkazu `\message` je možné mimo jiné použít `\meaning<sekvence>`, což vypíše to samé, jako `\show<sekvence>`, tedy význam dané řídicí sekvence. Také lze použít `\the<registr>`, což v argumentu `\message` vypíše totéž, jako přímé použití `\showthe<registr>`.

Kapitola 11

Možnosti pdfTeXu

PdfTeX rozšiřuje možnosti klasického TeXu o přímý výstup do PDF formátu. Tím ale jeho vlastnosti zdaleka nekončí. Formát PDF totiž nenabízí jen tisk dokumentu na papír. Při prohlížení v počítači je navíc možné používat hyperlinky, rozklikávací obsahy po straně prohlížeče (záložky), a mnoho dalšího. PdfTeX tedy nabízí rozšířenou sadu primitivních příkazů, kterými je možné v PDF výstupu tyto vlastnosti nastavit. Navíc obsahuje další rozšíření: vkládání externí grafiky do PDF výstupu, možnost přímé tvorby grafických prvků na úrovni elementárních příkazů podle specifikace PDF formátu a v neposlední řadě přidává nenápadná, ale užitečná mikrotypografická rozšíření. Protože TBN se pdfTeXem vůbec nezabývá, a přitom pdfTeX je rozšíření dosažitelné na všech TeXových distribucích ve stabilní verzi a je všeobecně používané (například ve formátu `pdfcsplain`), rozhodl jsem se tomuto tématu věnovat aspoň zde.

Tato kapitola chce být v mezích možností stručná a přehledová. Proto nepopisuje všechny vlastnosti pdfTeXu, ale uvádí jen ty podstatné. O dalších možnostech pdfTeXu (např. vkládání zvuku a videa) se lze dočíst v [5]. Úplný přehled možností poskytne specifikace PDF formátu [24]. Jisté kusy PDF kódů se totiž vyskytují jako argumenty příkazů pdfTeXu a lze tam zapsat cokoli, o čem se dočtete v PDF specifikaci.

11.1 Základní parametry

Registr `\pdfoutput` je pdfTeXem implicitně nastaven na nulu, což znamená, že pdfTeX vytváří DVI výstup a chová se tedy jako klasický TeX. Například `pdfcsplain` nastavuje tento registr na jedničku, takže pdfTeX vytvoří PDF výstup. Všechny další příkazy a registry pdfTeXu zmíněné v této kapitole mají smysl jen při `\pdfoutput=1`.

Následuje přehled dalších registrů pdfTeXu a jejich obvyklé výchozí hodnoty:

<code>\pdfpagewidth</code>	= 210mm	... šířka média (nastaveno podle formátu A4)
<code>\pdfpageheight</code>	= 297mm	... výška média (nastaveno podle formátu A4)
<code>\pdfhorigin</code>	= 1in	... poloha výchozího bodu, který odpovídá hornímu
<code>\pdfvorigin</code>	= 1in	levému rohu sazby při <code>\hoffset=0pt</code> , <code>\voffset=0pt</code>
<code>\pdfcompresslevel</code>	= 9	... úroveň komprese PDF výstupu (v rozmezí 0--9)
<code>\pdfdecimaldigits</code>	= 3	... přesnost reálných souřadnic v PDF výstupu
<code>\pdfimageresolution</code>	= 72 (dpi)	... pro výpočet velikosti rastrové grafiky, není-li specifikován její rozměr

Například OPmac mění hodnoty registrů `\pdfpagewidth` a `\pdfpageheight` v makru `\margins` (viz sekci 7.2). Toto makro nastavuje okraje a formát papíru.

Je asi rozumné zůstat kompatibilní s klasickým TeXem a ponechat `\pdfhorigin` a `\pdfvorigin` nastavené na „Knuthův bod“ ve vzdálenosti 1in od okrajů a podle toho upravit `\hoffset` a `\voffset`. Chcete-li ale hodit ne zcela koncepční Knuthův bod za hlavu a nastavovat `\hoffset` a `\voffset` přirozeně, je zde možnost `\pdfhorigin` a `\pdfvorigin` nastavit na nulu. OPmac ponechává Knuthův bod na svém místě a podle něj v makru `\margins` přepočítává `\hoffset` a `\voffset`.

11.2 Dodatečné informace k PDF

Formát PDF může obsahovat textové informace o autorovi, názvu díla, programu, který toto PDF stvořil, atd. Tyto informace jsou dosažitelné například v Acroreaderu v záložce File/Properties/Description. Na příkazové řádce se k informacím dostanete pomocí programu `pdfinfo`.

Informace jsou slovníkového charakteru: /Klíč (Hodnota) a sada klíčů je přesně vymezena a nelze ji rozšiřovat. Odpovídající hodnoty dále v některých případech musejí mít předepsaný formát.

PdfTeX nabízí k vyplnění těchto informací příkaz `\pdfinfo`. Následuje ukázka, kde v argumentu příkazu `\pdfinfo` jsou uvedeny všechny přípustné klíče.

```
\pdfinfo {/Author (<jméno autora>)
          /Title (<název díla>)
          /Subject (<stručná informace o díle>)
          /Keywords (<seznam klíčových slov>)
          /Creator (<program, který stvořil toto PDF>)
          /Producer (<doplňující informace o stvořiteli>)
          /CreationDate (D:<datum stvoření>)
          /ModDate (D:<datum modifikace>)}
}
```

Ne všechny údaje ve formě /Klíč (Hodnota) musíte vyplňovat. Údaj /Creator má implicitně hodnotu TeX a /Producer obsahuje implicitně pdfTeX včetně čísla verze pdfTeXu. Konečně údaj /CreationDate se nastaví implicitně shodně jako /ModDate a obsahuje datum a čas zpracování dokumentu.

Příkaz `\pdfinfo` je možno použít opakovaně dle principu „poslední vyhrává“, ovšem přepisují se jen explicitně uvedené údaje. Takže je možné údaje přidávat i postupně.

Hodnoty pro /CreationDate a /ModDate mají speciální formát:

D:<rok><měsíc><den><hodina><minuta><sekunda>

Údaj <rok> je čtyřciferný a ostatní údaje jsou dvouciferné. Například

```
\pdfinfo { /CreationDate (D:20130813120000) }
```

zanese údaj o vytvoření díla v roce 2013, 13. 8. v pravé poledne.

Chcete-li nastavit /CreationDate odlišně od /ModDate, nastavte jen „datum prvního stvoření“ v /CreationDate (jako v předchozí ukázce). Údaj /ModDate se doporučuje nenastavovat: pak se automaticky doplní podle data a času posledního zpracování dokumentu.

Údaje v kulatých závorkách jsou *stringy* podle specifikace PDF formátu. Ty je možné kódovat dvěma způsoby: Buď podle tzv. „PdfDocEncoding“ (to je ASCII a ISO-8859-1 s přidáním některých znaků, ale pro češtinu nepoužitelné) nebo v „UTF-16BE Unicode“. Jak vypadá toto kódování, je popsáno v [17]. Máte tedy dvě možnosti. Buď psát údaje jednoduše bez hacku a carek, například `\pdfinfo{/Author (Petr Olsak)}`, nebo použít soubor `pdfuni.tex`, který definuje konverzní makro `\pdfunidef<sekvence>{<text>}`. Po provedení `\pdfunidef` bude <sekvence> makrem expandujícím na <text> v kódování UTF-16BE Unicode. Je tedy možné psát:

```
\input pdfuni
\pdfunidef\tmp{Božena Němcová} \pdfinfo{/Author (\tmp)}
\pdfunidef\tmp{Růžové poupě} \pdfinfo{/Title (\tmp)}
```

11.3 Nastavení výchozích vlastností PDF prohlížeče

Specifikace PDF formátu umožňuje nastavit některé vlastnosti PDF prohlížeče. Zejména lze vymezit, v jakém stavu se prohlížeč zjeví po prvním načtení PDF souboru, a také se mu dá vnutit speciální interpretace stránek v dokumentu. Tyto údaje se vkládají do tzv. *katalogu* v PDF formátu a pdfTeX pro ně nabízí příkaz `\pdfcatalog`. Podobně jako u příkazu `\pdfinfo` se dají argumenty zadávat dohromady při jednom zavolání `\pdfcatalog` nebo postupně. Seznamovat se s nimi budeme postupně.

```
\pdfcatalog {/PageMode <výchozí stav prohlížeče>}
  přitom <výchozí stav prohlížeče> je jedna z možností:
  /UseOutlines ... zobrazí po straně rozklikávací obsah (záložky)
  /UseThumbs   ... zobrazí po straně náhledy stránek
  /UseNone     ... nezobrazí po straně nic
  /FullScreen  ... zobrazí dokument na celé obrazovce
```

```
\pdfcatalog {/PageLayout <uspořádání stránek>}
  přitom <uspořádání stránek> je jedna z možností:
  /SinglePage   ... jednotlivé stránky
  /OneColumn    ... stránky pod sebou navazují
  /TwoColumnRight ... stránky vedle sebe jak v rozevřené knize
  /TwoColumnLeft ... stránky vedle sebe opačně než v knize
```

Implicitní stav prohlížeče (není-li údaj specifikován) záleží na použitém prohlížeči. Některé prohlížeče taky nemusejí implementovat všechny vlastnosti.

```
\pdfcatalog {/ViewerPreferences << /HideToolbar <true nebo false>
                                         /HideMenubar <true nebo false>
                                         /HideWindowUI <true nebo false>
                                     >>>}
```

Nastavením uvedených hodnot na **true** můžete potrápit uživatele některých prohlížečů, kteří pak marně budou hledat v okénku prohlížeče nabídku na ovládání prohlížeče. Například `\pdfcatalog {/ViewerPreferences << /HideMenubar true >>>}` schová hlavní nabídku. Implicitně mají uvedené parametry hodnotu **false**.

```
\pdfcatalog {/PageLabels << /Nums [ <způsob stránkování> ] >>>}
```

Tento údaj umožní interpretovat jednotlivé stránky jinak než vzestupně od jedné. Prohlížeč totiž nechte stránkovou číslici v sazbě (ostatně, ani by nevěděl, kde ji má hledat, a někdy zcela chybí). Přesto je vhodné mu sdělit, že například prvních 10 stránek je číslováno římskými čísly a pak začíná arabské číslování. Zrovna tento požadavek by se zapsal takto:

```
\pdfcatalog {/PageLabels << /Nums [ 0 <</S/r>> 10 <</S/D>> ] >>>}
```

Údaj *<způsob stránkování>* je seznam ve tvaru:

```
<absolutní strana> <<typ stránkování>> <absolutní strana> <<typ stránkování>> ...
```

přitom *<absolutní strana>* je počáteční číslo strany, kterého se týká *<typ stránkování>*. Tato dvojice tvoří jeden blok stránek, který je ukončen následující *<absolutní stranou>* nebo koncem dokumentu. Číslování *<absolutních stran>* začíná od nuly (aby se to trochu pletlo) a je průběžné v celém dokumentu. Konečně *<typy stránkování>* mohou vypadat takto:

```
nic ... stránky nejsou číslovány
/S/D ... číslování arabsky v desítkové soustavě
/S/r ... číslování římsky malými písmeny
/S/R ... číslování římsky velkými písmeny
/S/a ... označení stránek abecedně: a, b, c, ...
/S/A ... označení stránek abecedně: A, B, C, ...
```

Implicitně číslování každého bloku začíná od jedné. Chcete-li ve vybraném bloku jiné číslo startovní stránky, připište do *<typu stránkování>* údaj */St <číslo>*. Například místo `<</S/D>>` v předchozím příkladu pište `<</S/D /St 11>>` a budete mít blok „arabských“

stránek číslován od jedenácti. Konečně můžete do *<typu stránkování>* přidat */P(<text>)* a tím deklarujete *<text>* jako statický prefix vkládaný před každou stránkovou číslicí.

Za argumentem `\pdfcatalog` může následovat klíčové slovo `openaction`, za kterým je specifikován odskok na konkrétní místo v dokumentu. Odskoky a cíle v dokumentu budou vysvětleny později v sekci 11.4, takže zde je třeba se spokojit jen s příkladem, který způsobí, že (některé prohlížeče) po přečtení PDF dokumentu skočí rovnou na stranu 42:

```
\pdfcatalog {} openaction goto page 42 {/XYZ}
```

11.4 Hyperlinky

• **Vymezení cíle v dokumentu** • Cíl, kam má prohlížeč skočit při kliknutí na hypertextový odkaz, je bezrozměrný a neviditelný objekt v sazbě vymezený příkazem

```
\pdfdest name{<identifikátor>} <typ doskoku>
```

Stejný *<identifikátor>* se použije v místě hypertextového odkazu. Jako identifikátor může sloužit libovolný text včetně číslic a mezer. Jeden *<identifikátor>* se může v roli cíle použít v celém dokumentu jen jednou. Hypertextových odkazů na stejný cíl může být více.

Pomocí *<typu doskoku>* se specifikuje, jak se má po kliknutí na odkaz v místě cíle PDF prohlížeč zachovat. Nejběžnější asi bude *<typ doskoku>* ve tvaru *xyz*, což je pokyn pro prohlížeč, aby zachoval stávající zvětšení náhledu a posunul se na místo cíle nejlépe tak, aby cíl lícovál s horní hranou okna prohlížeče. Když tedy napíšeme `\pdfdest` v horizontálním módu, máme cíl na účarí, takže řádek, který obsahuje cíl, bude z větší části schován za horní hranou prohlížeče a nebude vidět. Je tedy rozumné cíl specifikovat například takto:

```
Tady je velmi zajímavý text%
\vbox to0pt{\vss \pdfdest name{<identifikátor>} xyz \kern1.5em}
a tady pokračuje text.
```

V tomto příkladu je cíl vystrčen nad účarí o 1,5 em. OPmac používá právě tento typ cíle a pro velikost vystrčení nad účarí používá makro `\destheight`.

Další *<typy doskoku>* obsahují případné zvětšení/zmenšení náhledu dle specifikace:

```
fit          ... celá strana s cílem se vejde do okna
fith         ... šířka strany se vejde do okna
fitv        ... výška strany se vejde do okna
fitb        ... popsaný text strany (bez okrajů) se vejde do okna
fitbh, fitbv ... analogie k fith a fitv
fitr height<výška> depth<hloubka> width<šířka> ... virtuální
              obdélník se celý vejde do okna s maximálním zvětšením
```

• **Obecná specifikace hyperlinku** • Aktivní plocha, na kterou je možné najet myší a při kliknutí se něco stane (typicky prohlížeč odskočí na místo cíle), se vymezí pomocí

```
\pdfstartlink height<výška> depth<hloubka> <atributy>
               <kam skočit> <text>\pdfendlink
```

O vymezení údaje *<kam skočit>* pojednáme později. Údaj *<výška>* a *<hloubka>* vymezuje velikost virtuální podpěry, která ve skutečné sazbě nepřekáží, ale vymezuje výšku a hloubku obdélníku s aktivní plochou odkazu. Šířka tohoto obdélníku bude stejná, jako šířka *<textu>*. Obdélník tedy typicky ohraničuje *<text>*. Aktivní plocha nemusí být jen jediný obdélník: pokud se *<text>* rozlomí na více řádků, pak obdélníky s aktivní plochou ohraničují části *<textu>* na každém řádku. Parametr *<text>* může zahrnovat i více odstavců a může přejít i přes hranici stránky.

Nepovinný parametr *<atributy>* umožní zviditelnit hranici obdélníku s aktivní plochou a je tvaru `attr{/C[<red> <green> <blue>] /Border[0 0 <tloušťka>]}`. Hranice bude mít barvu namíchanou jako RGB podle parametrů *<red>* *<green>* *<blue>* (jsou to desetinná čísla v rozsahu 0 až 1) a *<tloušťka>* vymezuje tloušťku rámečku v bp. Například:

```
\pdfstartlink height<výška> depth<hloubka> attr{/C[.9 0 0] /Border[0 0 .6]}
      <kam skočit> <text>\pdfendlink
```

vytvoří červený rámeček kolem *<textu>* s tloušťkou .6 bp. Rámečky tohoto typu při tisku mizí. Některé prohlížeče rámečky nezobrazí, je-li jejich tloušťka menší než 0,5 bp.

● **Interní odkaz** ● Údaj *<kam skočit>* v parametrech příkazu `\pdfstartlink` může obsahovat požadavek na odskok do místa cíle deklarovaného pomocí `\pdfdest` nebo odskok na stránku deklarovanou jejím číslem. Jsou tedy tyto možnosti:

```
goto name{<identifikátor>}
goto page <číslo> {<typ doskoku>}
```

Například

```
\pdfstartlink height1em depth.5em goto name{cosi123} TEXT\pdfendlink
```

vytvoří interní hypertextový odkaz (aktivní bude TEXT), který odkazuje na místo cíle ve stejném dokumentu specifikované třeba pomocí `\pdfdest name{cosi123} xyz`.

Při `goto page` je třeba specifikovat *<typ doskoku>* podobně, jako se to dělá v příkazu `\pdfdest`. Ovšem forma specifikace je mírně odlišná. Používají se značky */XYZ*, */Fit*, */FitH*, */FitV*, */FitB*, */FitBH*, */FitBV*, které korelují postupně s údaji *xyz*, *fit*, *fith*, *fitv*, *fitb*, *fitbh* a *fitbv*. Například klik na TEXT vymezený pomocí

```
\pdfstartlink height1em depth.5em goto page 42 {/FitB} TEXT\pdfendlink
```

způsobí skok na stranu 42, kterou prohlížeč zobrazí ve svém okně celou bez okrajů.

● **Odkaz do jiného PDF souboru** ● Je-li jiný PDF soubor ve stejném adresáři, kde je PDF prohlížeč najde, je možné do něj odkazovat pomocí specifikace údaje *<kam skočit>* ve tvaru:

```
goto file {<jméno>.pdf} name {<identifikátor>}
```

nebo

```
goto file {<jméno>.pdf} page <číslo> {<typ doskoku>}
```

Prohlížeč při kliku na takový odkaz automaticky načte soubor *<jméno>.pdf* a zobrazí ho v místě specifikovaného cíle. Při použití `name {<identifikátor>}` musí být v souboru *<jméno>.pdf* tento identifikátor použit v příkazu `\pdfdest`.

● **Odkaz na WWW stránku** ● Údaj *<kam skočit>* lze zapsat ve formátu:

```
user{/Subtype/Link /A << /Type/Action /S/URI /URI (<url>) >>}
```

V tomto případě se PDF prohlížeč pokusí (je-li vhodně nakonfigurován) komunikovat s existujícím webovým prohlížečem v systému. Je-li takový prohlížeč spuštěn, je požádán o načtení příslušného *<url>*, což může být WWW stránka nebo odkaz na dokument. Není-li webový prohlížeč spuštěn, je po kliku na odkaz spuštěn a je mu předán pokyn k načtení *<url>*. Například

```
\pdfstartlink height1em depth.5em user{/Subtype/Link
  /A << /Type/Action /S/URI /URI (http://petr.olsak.net) >>}
TEXT\pdfendlink
```

způsobí zobrazení uvedené stránky po kliknutí na TEXT.

11.5 Klikací obsahy po straně prohlížeče

Jeden řádek klikacího obsahu po straně prohlížeče (tj. záložku) lze vytvořit příkazem:

```
\pdfoutline <kam skočit> count <číslo> {\text záložky}
```

Zde <kam skočit> je tvaru `goto <něco>`, jak je uvedeno v předchozí sekci v odstavcích „interní odkaz“ nebo „odkaz do jiného PDF souboru“. Dále <číslo> označuje počet potomků ve stromové struktuře záložek (viz dále) a <text záložky> je PDF string, který se zjeví v prohlížeči jako záložka. Vzhledem k tomu, že to je PDF string, platí o něm totéž, co bylo řečeno o kódování stringu v sekci 11.2. Tedy české texty je nutné psát bez háčeků a čárek nebo je kódovat v UTF-16BE Unicode. Příklad:

```
\pdfoutline goto name{cosi123} count 0 {Tu je zalozka}
```

nebo:

```
\input pdfuni  
\pdfunidef\tmp{Tu je záložka} \pdfoutline goto name{cosi123} count 0 {\tmp}
```

Cíl deklarovaný v odkazu <kam skočit> pochopitelně musí existovat. Je-li tedy tvaru `goto name{<identifikátor>}`, pak musí být <identifikátor> deklarován příkazem `\pdfdest`.

Údaj za `count` značí počet potomků umožní deklarovat stromovou strukturu klikacího obsahu. Ovšem musíme se smířit s poněkud nešikovně stanovenou PDF specifikací, která vyžaduje uvést počet přímých potomků záložky, které jsou následně vytvořeny dalšími příkazy `\pdfoutline`. Opmac kvůli tomu prochází při použití makra `\outlines` obsah dokumentu (přečtený z `.ref` souboru) dvakrát. Při prvním průchodu si spočítá, kolik má který údaj potomků (tj. kolik má každá kapitola sekcí a kolik má každá sekce podsekcí) a v druhém průchodu teprve tyto údaje použije při sestavení klikacího obsahu postupným voláním příkazu `\pdfoutline`.

Má-li záložka potomky, je možné pomocí znaménka čísla za `count` označit, zda ve výchozím zobrazení bude položka otevřená (tj. potomci budou viditelní) nebo zavřená (potomci se ukáží, až uživatel klikne na odpovídající grafický symbol). V prvním případě píšeme `count <počet potomků>` a ve druhém `count -<počet potomků>`.

11.6 Lineární transformace sazby

Veškerá sazba v pdfTeXu může podléhat lineární transformaci, která je daná transformační maticí `\pdfsetmatrix{<a> <c> <d>}`. Tato matice se v lineární algebře zapisuje do dvou řádků:

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}, \quad \text{např. zvětšení: } \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix}, \quad \text{nebo rotace: } \begin{pmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{pmatrix}.$$

Příkaz `\pdfsave` uloží stávající transformační matici a aktuální bod sazby. V době vykonání příkazu `\pdfrestore` se matice vrátí do původní podoby a aktuální bod sazby v té době musí být na stejném místě, jako byl v době `\pdfsave`, jinak se nám sazba rozjede a pdfTeX o tom varuje na terminálu a v `.log` souboru. Aby se sazba sešla ve stejném bodě, je potřeba použít například konstrukci `\pdfsave...\rlap{\text}\pdfrestore`. Transformační matice se nastavují pomocí `\pdfsetmatrix`. Opakované použití `\pdfsetmatrix` způsobí pronásobení transformační matice novou maticí, takže to funguje jako skládání zobrazení. Opmac nabízí dvě užitečná makra `\pdfscale{<vodorovně>}{<visle>}` a `\pdfrotate{<úhel>}`. Parametr <úhel> je ve stupních (včetně možnosti zapsat desetinné číslo). Tato makra provedou odpovídající `\pdfsetmatrix`.

Aplikujeme-li více matic za sebou, je potřeba vědět, že výchozí text prochází transformací jednotlivých matic „odzadu dopředu“, takže například:

```
První: \pdfsave \pdfrotate{30}\pdfscale{-2}{2}\rlap{text1}\pdfrestore
      % text1 je zvětšen dvakrát a překlopen podél svislé osy,
      % dále je otočen o 30 stupňů doleva a konečně je vytištěn.
druhý: \pdfsave \pdfscale{-2}{2}\pdfrotate{30}\rlap{text2}\pdfrestore
      % text2 je otočen o 30 stupňů doleva, dále zvětšen a překlopen
      % podél svislé osy, nakonec vytištěn.
třetí: \pdfsave \pdfrotate{-15.3}\pdfsetmatrix{2 0 1.5 2}\rlap{text3}%
      \pdfrestore % nejprve zkosení, pak otočení o 15.3 stupňů doprava
```

Ukázka dává následující výsledek. První druhý: *text3*

Následující poněkud praktičtější ukázka vkládá obrázek otočený o 90 stupňů na střed sazby pomocí `\centerline`. Protože \TeX si myslí, že box vytvořený pomocí `\centerline` má výšku rovnou výšce obrázku bez otočení (otočení se totiž děje na úrovni PDF kódu), je potřeba velikost obrázku změřit (po vložení do `\boxu0`) a před `\centerline` dát mezeru rovnou šířce minus výšce neotočeného obrázku. Obrázek je otočen podle levého dolního rohu a \TeX si pouze myslí, že tam je bezrozměrný box, který by umístil na střed. Je tedy potřeba tento bezrozměrný box posunout o půlku výšky neotočeného obrázku doprava, tedy vložit kern vlevo v `\centerline` o velikosti celé výšky obrázku.

```
\picw=4cm \setbox0=\hbox{\inspic hodiny.jpg }
\par\nobreak \vskip\wd0 \vskip-\ht0
\centerline {\kern\ht0 \pdfsave\pdfrotate{90}\rlap{\box0}\pdfrestore}
\nobreak\medskip
\caption/f Již známé hodiny otočené o 90$^\circ$
```

Uvedený příklad vytvoří



Obrázek 11.6.1 Již známé hodiny otočené o 90°

Obecnější transformace boxů včetně výpočtu jejich nových rozměrů jsou řešeny na <http://petr.olsak.net/opmac-tricks.html#transformbox>.

11.7 Vkládání externí grafiky

Pdf \TeX vkládá externí grafiku (ze souborů png, jpg, jbig2 nebo pdf) do výstupního PDF ve dvou krocích. Nejprve při použití příkazu

```
\pdfximage height<výška> width<šířka> {\jméno}.<přípona>}
```

pdfTeX nahlédne do souboru $\langle jméno \rangle . \langle přípona \rangle$ a ujasní si potřebné údaje o obrázku. Obrázek ale nezobrazí. Místo toho na něj prozradí interní odkaz ve formě čísla v globálním registru `\pdflastximage`.

V místě zobrazení obrázku je následně třeba použít `\pdfrefximage<číslo odkazu>`. První výskyt takového příkazu vloží data obrázku do výstupního PDF souboru a vykreslí obrázek. Každý další výskyt `\pdfrefximage` se stejným $\langle číslem odkazu \rangle$ pouze vykreslí obrázek pomocí odkazu (bez nového vkládání dat).

Pokud chceme obrázek použít jen jednou, pak je obvyklé psát

```
\pdfximage height<výška> width<šířka> {\langle jméno \rangle . \langle přípona \rangle}%
\pdfrefximage\pdflastximage
```

zatímco pro opakované použití se doporučuje psát

```
\pdfximage height<výška> width<šířka> {\langle jméno \rangle . \langle přípona \rangle}%
\mathchardef\obrazek=\pdflastximage
```

Tento kód pouze připraví odkaz pro obrázek do znakové konstanty (zde v příkladě označené jako `\obrazek`). Pro zobrazení obrázku je možné pak psát `\pdfrefximage\obrazek`. Takové zobrazení je možné psát opakovaně v různých místech dokumentu, a přitom to (skoro) nezvětšuje množství dat ve výstupním PDF souboru. Zobrazení je totiž realizované odkazem na stále stejné místo s daty.

Parametry `height<výška>` i `width<šířka>` příkazu `\pdfximage` mohou chybět. Chybí-li oba, bude mít obrázek přirozenou velikost. Chybí-li jeden, podle druhého se nastaví požadovaná velikost tak, že se obrázek nedeformuje. Při nastavení obou hodnot bude mít obrázek zadané rozměry a je vysoce pravděpodobné, že bude narušen jeho přirozený poměr výška/šířka¹). Výsledný obrázek (po provedení příkazu `\pdfrefximage<číslo>`) se v sazbě chová jako box uvedených rozměrů.

Příkaz `\pdfximage` při načítání vícestránkového PDF dokumentu pojme jako „obrázek“ implicitně první stránku tohoto dokumentu. Pokud ale za případné parametry `height<výška>` a `width<šířka>` uvedete parametr `page<číslo strany>`, bude načtena specifikovaná strana. Po přečtení aspoň jedné strany takového PDF dokumentu je možné v registru `\pdflastximagepages` zjistit celkový počet stran čteného dokumentu, takže je možné spustit cyklus a postupně přečíst všechny strany.

Následující příklad implementuje makro `\adddocument{\langle jméno \rangle}`, které připojí k vytvářenému dokumentu všechny stránky externího dokumentu $\langle jméno \rangle . pdf$. Například je tím možné připojit do dokumentu přílohy z dokumentu, který byl vytvořen třeba zcela jiným softwarovým nástrojem nikterak nesouvisejícím s TeXem.

```
\newcount\strana
\def\adddocument#1{\vfil\break
  \bgroup
    \strana=1 \voffset=-1in \hoffset=-1in \nopagenumbers
    \loop
      \pdfximage width\pdfpagewidth page\strana {#1.pdf}
      \vbox to0pt{\pdfrefximage\pdflastximage \vss}
      \vfil\break
      \ifnum\strana<\pdflastximagepages \advance\strana by1 \repeat
    \egroup
  }
```

Makro nejprve odstraní a celý proces vložení externího dokumentu provede ve skupině (`\bgroup... \egroup`). Nastaví `\voffset` a `\hoffset` na -1 in , takže se dostaneme

¹⁾ Jak je občas možno vidět například při nesprávně nastaveném aspektu u některých televizí.

těsně k levému a hornímu okraji papíru. V cyklu nejprve načteme stránku (tj. aspoň jedna stránka bude vždy načtena), modifikujeme její šířku na `\pdfpagewidth` a stránku vytiskneme. Nyní se už můžeme zeptat na hodnotu registru `\pdflastximagepages`, takže víme, kolik má dokument stránek. Je-li číslo vytištěné strany menší než celkový počet stran, zvětšíme číslo strany o jedničku a proces opakujeme.

11.8 Stránková montáž pdfT_EXem

Stránková montáž je rozmístění jednotlivých stránek dokumentu na velké archy papíru, které projdou tiskařskými stroji s oboustranným tiskem, pak vlezou do speciálního automatu, který je šikově poskládá a ze tří stran ořízne tak, že vzniká jeden svazek pro vazbu knihy. Jednotlivé svazečky kladou knihaři vedle sebe a tím postupně vzniká kniha. Na jeden arch se typicky vejde 8 stránek. Arch má rub a líc, takže dohromady nese 16 stránek. Samozřejmě, že stránky na archu nejdou po řadě, ale vše je potřebné předřídit způsobu skládání archu ve zmíněném skládacím automatu. Knihaři používají na rozmístění stránek do archu speciální schémata.

Z vlastností vyložených na konci předchozí sekce je zřejmé, že pdfT_EX může posloužit i pro stránkovou montáž. Navíc během montáže může přidat ohybové, ořezové i pasovací značky tam, kde si to knihaři přejí. Přitom vše je v rukou programátora, kterému stačí umět v T_EXu k sobě sestavovat příslušné boxy a používat příkaz `\pdfximage`.

Následující příklad je ukázkou této technologie, ale využijete ho i s normální laserovou tiskárnou, do které nenacpete větší arch než formátu A4. Cílem bude na tento formát rozmístit vždy dvě stránky dokumentu vedle sebe tak, že až to projede laserovou tiskárnou s oboustranným tiskem, dostanete svazek papírů, který stačí přehnout v půli a vzniká sešitek formátu A5 se správným pořadím stránek. Kromě toho, jsou-li původní stránky dokumentu příliš velké, pdfT_EX je automaticky zmenší, aby měly rozměr podle formátu A5. Stačí použít soubor `pdfa5.tex` s tímto obsahem:

```
\def\document{navrh-rozpocet} % Jméno zpracovávaného dokumentu bez přípony
\nopagenumbers % nebude další stránkování
\pdfpagewidth=297mm \pdfpageheight=210mm % Arch = formát A4 naležato
\pdfhorigin=0pt \pdfvorigin=0pt % Knuthův bod hodíme za hlavu
\def\pageswidth{width.5\pdfpagewidth} % Šířka stránek je půlka šířky archu

\pdfximage \pageswidth {\document.pdf} % Čteme první stránku, abychom se
\mathchardef\firstpage=\pdflastximage % dozvěděli \pdflastximagepages

\def\putpage#1{% vložení stránky číslo #1 do archu
  \ifnum#1>\pdflastximagepages \hbox{\vrule\pageswidth}\else % vakát
    \ifnum#1=1 \pdfrefximage\firstpage % první strana
    \else \pdfximage \pageswidth page#1 {\document.pdf}% % normální strana
    \pdfrefximage\pdflastximage
  \fi\fi}

\newcount\al \newcount\ar \newcount\bl \newcount\br
\al=\pdflastximagepages
\advance\al by3 \divide\al by4 \multiply\al by4 % Zaukrouhlení na 4N nahoru
\ar=1 \bl=2 \br=\al \advance\br by-1 % Drobné další výpočty
\loop
  \hbox{\putpage\al \putpage\ar}\vfil\break % Líc archu
  \hbox{\putpage\bl \putpage\br}\vfil\break % Rub archu
  \advance\ar by2 \advance\al by-2
```

```

\advance\bR by-2 \advance\BL by2
\ifnum \aL>\aR \repeat
\end

```

Představme si nejprve pro jednoduchost, že máme čtyřstránkový dokument. Na líc archu se vytisknou strany [4|1] a na rub strany [2|3]. To se skutečně dá přeložit v půli s požadovaným výsledkem. Funguje to i pro více stránek. Na líc se obecně tisknou strany $[\backslash aL|\backslash aR]$ a na rub strany $[\backslash bL|\backslash bR]$. Makro `\putpage` se větví podle tří možností: při požadavku tisku stránky mimo rozsah vytiskne vakát, při tisku strany 1 využije již dříve načtená data a v ostatních případech vytiskne požadovanou stranu.

• **Poznámka** • Než tento příklad vyzkoušíte, nezapomeňte si nastavit na tiskárně duplexový tisk s převrácením přes kratší stranu.

11.9 Využití elementárních PDF příkazů pro grafiku

Pd_fT_EX disponuje příkazem `\pdfliteral{⟨PDF kód⟩}`, kterým lze vložit do výstupního PDF souboru libovolný kód. Tímto způsobem lze nastavovat barvy, kreslit křivky, dělat výplně a rozličné obrázky. Stačí vědět, jakým *⟨PDF kódem⟩* požadovaný grafický efekt realizovat. Protože tato možnost odkrývá netušené obzory, věnujeme jí tuto poněkud obsírnější sekci. Text je z významné části převzat z článku [16].

K tomu, abychom mohli psát užitečné *⟨PDF kódy⟩*, nemusíme hned studovat mnohastránkovou PDF specifikaci [24]. Zpočátku stačí znát následující užitečné elementární PDF příkazy:

```

q                % zahájení skupiny pro nastavení grafického stavu
Q                % ukončení skupiny, návrat k původnímu grafickému stavu
⟨num⟩ g          % (Grey) nastavení stupně šedi pro plochy
⟨num⟩ G          % (Grey) nastavení stupně šedi pro tahy
⟨r⟩ ⟨g⟩ ⟨b⟩ rg   % nastavení barvy v RGB pro plochy
⟨r⟩ ⟨g⟩ ⟨b⟩ RG   % nastavení barvy v RGB pro tahy
⟨c⟩ ⟨m⟩ ⟨y⟩ ⟨k⟩ k % (cmyK) nastavení barvy v CMYK pro plochy
⟨c⟩ ⟨m⟩ ⟨y⟩ ⟨k⟩ K % (cmyK) nastavení barvy v CMYK pro tahy
⟨width⟩ w       % (Width) nastavení šířky čáry
⟨typ⟩ j         % nastavení typu lámání čáry, 0 s hranami, 1 kulatě, 2 s ořezem
⟨typ⟩ J         % nastavení typu zakončení čáry, 0 hranatý, 1 kulatý, 2 s přesahem
⟨a⟩ ⟨b⟩ ⟨c⟩ ⟨d⟩ ⟨e⟩ ⟨f⟩ cm % (Concatenate Matrix) pronásobení transformační matice
⟨x⟩ ⟨y⟩ m       % (Moveto) nastavení polohy kreslicího bodu
⟨dx⟩ ⟨dy⟩ l     % (Lineto) přidání úsečky
⟨x1⟩ ⟨y1⟩ ⟨x2⟩ ⟨y2⟩ ⟨x3⟩ ⟨y3⟩ c % (Curveto) přidání Bézierovy křivky
⟨x⟩ ⟨y⟩ ⟨dx⟩ ⟨dy⟩ re % (Rectangle) připraví obdélník
h              % uzavření postupně budované křivky
S              % (Stroke) vykreslení připravené křivky čarou
s              % stejné jako h S
f              % (Fill) vyplnění oblasti dané uzavřenou připravenou křivkou
B              % (Fill and Storke = Both) vyplní oblast a obtáhne ji čarou
W n           % nastavení připravené uzavřené křivky jako omezující (clipping)

```

Uvedené příkazy si za chvíli ukážeme v příkladech podrobněji.

Příkaz `\pdfliteral{⟨PDF kód⟩}` z hlediska T_EXu neudělá se sazbou nic. Tj. následující sazba pokračuje tam, kde předchozí skončila. Přitom *⟨PDF kód⟩* může obsahovat elementární příkazy pro PDF rasterizér například na změnu grafického stavu nebo na vykreslení nějaké grafiky.

● **Nastavení barev** ● Nejprve vysvětlíme a na příkladech ukážeme příkazy na změnu barvy `g`, `G` (šedá), `rg`, `RG` (RGB), `k` a `K` (CMYK). Jsou zde dvě varianty (malá a velká písmena) pro každý barevný prostor. Příkaz s malými písmeny ovlivní použití barvy při vyplňování uzavřených křivek (`Fill`) a při sazbě textu. Pochopitelně, protože kresba jednotlivých písmen v textu probíhá taky vyplňováním uzavřených křivek. Příkaz pro změnu barvy s velkými písmeny ovlivní barvu při kresbě podél křivek (`Stroke`). Tato dvě nastavení jsou na sobě nezávislá, lze tedy nastavit vyplňování zelené a obtahování červené. Je třeba také vědět, že pdfTeX řeší vykreslení `\vrule` a `\hrule` pomocí `Stroke`, je-li objekt tenčí nebo roven 1 bp. A vyplní obdélník pomocí `Fill`, má-li oba rozměry větší než 1 bp. Z tohoto pohledu nastavení barvy pomocí malých písmen se týká textu a „tlustých“ `\vrule` a `\hrule`, zatímco nastavení barvy pomocí velkých písmen „tenkých“ `\vrule` a `\hrule`.

Poznamenejme, že nastavování barev v OPmac je vyloženo v sekci 7.7 a že OPmac nabízí rozlišení těchto dvou „barevných druhů“ pomocí prefixu `\linecolor`.

Příklad přepnutí do červené sazby může vypadat takto:

Tady je černý text.

```
\pdfliteral{1 0 0 rg}Tady je červený.\pdfliteral{0 0 0 rg}
```

Tu je zase černý.

```
\pdfliteral{0 1 1 0 k}Tu znovu červený.\pdfliteral{0 g}
```

A zpátky černý.

Dostaneme tento výsledek: Tady je černý text. **Tady je červený.** Tu je zase černý. **Tu znovu červený.** A zpátky černý.

Argumenty příkazů pro nastavování barvy jsou obecně desetinná čísla (s desetinou tečkou) v rozsahu od nuly do jedné. Například `\pdfliteral{0.7 g}` znamená třicetiprocentní šedou. Nebo třeba `\pdfliteral{0.25 0.3 0.75 rg}` nastaví barvu smíchanou z 25 % červené, 30 % zelené a 75 % modré v aditivním barevném prostoru RGB.

Na příkladu vidíme, že je v podstatě jedno, jaký barevný prostor použijeme. Barvy ovšem nejsou přesně stejné, protože CMYK prochází korekcemi vhodnými pro tisk. Dále je možné uzavřít nastavení barvy do skupiny `\pdfliteral{q}...\pdfliteral{Q}`. Po uzavření skupiny se sazba vrátí k původní barvě. Ovšem sazba se taky vrátí do původního bodu sazby, což často není žádoucí. Proto je lepší ukončit sazbu v barvě příkazem `\pdfliteral{0 g}`, který jednoduše vrátí barvu černou.

S nastavováním barev souvisí poměrně komplikovaný problém, který se typicky neprojeví, je-li barva nastavena lokálně pro objekt, který nikdy nepřekročí hranici stránky. Změna barvy je totiž pokyn pro PDF rasterizér, o kterém TeX nemá tušení. Máte-li změnu barvy na první stránce a návrat k černé na některé další stránce, pak PDF rasterizér změní barvu od místa změny po konec stránky. Pro každou stranu zakládá PDF rasterizér novou skupinu `q...Q`, takže na konci stránky barva mizí. Přitom se obarví i zápatí, tedy číslo strany. Na další stránce rasterizér zakládá novou skupinu a vůbec neví, že má pokračovat ve speciální barvě. Pokračuje tam barvou černou.

Tento problém řeší pdfTeXové příkazy pro tzv. `colorstack`. Ty ale nejsou bohužel dokumentovány, nicméně jsou použity např. v L^ATeXovém balíku `color.sty`. Makro OPmac je nepoužívá právě proto, že nejsou dokumentovány, a řeší uvedený problém ve vlastní režii jen na úrovni maker pomocí `.ref` souboru.

● **Kresba křivek** ● Křivku je potřeba pomocí elementárních PDF příkazů nejprve připravit (`Moveto`, `Lineto`, `Curveto`) a poté podél připravené křivky můžeme vést čáru (`Stroke`) nebo, je-li uzavřená, můžeme vyplnit vnitřek křivky (`Fill`).

Argumenty příkazů pro přípravu křivky jsou desetinná čísla v jednotkách, které jsou implicitně nastaveny na bp (typografický bod, 1/72 palce) a souřadnicový systém implicitně

prochází aktuálním bodem sazby, tj. místem, kde je použit příslušný příkaz `\pdfliteral`. První souřadnicová osa x směřuje doprava a druhá y nahoru. Toto implicitní chování je možné změnit změnou transformační matice, o čemž pojednáme později.

Následuje příklad, který vytvoří zelený trojúhelník a modrý půldisk.



```
\pdfliteral{q      % uchování grafického stavu
  0 1 0 RG 0 0 1 rg % nastavení barvy pro čáry (zelená) a výplně (modrá)
  3.2 w           % (Width) šířka čáry bude 3.2 bp
  0 0 m           % (Moveto) pero položíme do počátku
  30 30 1         % (Lineto) přidáme úsečku z 0 0 do 30 30
  30 0 1         % (Lineto) přidáme úsečku a 30 30 do 30 0
  h              % uzavření křivky,
  S              % (Stroke) kresba křivky čarou v dané šířce a barvě
  50 0 m         % (Moveto) nastavení pera do bodu 50 0
  50 10 60 20 70 20 c % (Curveto) první čtvrtina disku
  80 20 90 10 90 0 c % (Curveto) druhá čtvrtina disku
  h              % uzavření křivky
  f              % vyplnění uzavřené křivky barvou
  Q              % návrat k původním hodnotám grafického stavu
}
```

Kresba se zjeví v aktuálním bodě sazby a nebude zabírat žádné místo. Takže abychom tímto obrázkem nepřekreslili předchozí text, bylo potřeba připravit místo pro obrázek manuálně. V tomto konkrétním příkladě jsem rozměry pro obrázek odhadl a napsal:

```
\nobreak\vskip2cm\centerline{\hss\pdfliteral{\předchozí kód}}\hskip4cm\hss}
```

Při kresbě tímto způsobem je potřeba mít na paměti následující pravidla:

- Nastavení barvy a tloušťky čáry je vhodné dělat mezi `q` a `Q`.
- Před vykreslením pomocí `S` nebo `f` nebo `B` je nutné připravit křivku.
- Příprava křivky musí začínat příkazem `m`, který nastavuje aktuální bod kresby.
- Křivka se připravuje příkazy `l` nebo `c` které budují křivku postupně z částí. Každý další příkaz `l` nebo `c` připojí další úsek křivky k již sestavené a posune na její konec aktuální bod kresby.
- Během přípravy křivky je možné použít další příkaz `m` a tím vznikne křivka nesouvislá.
- Příprava křivky ještě neznamená její vykreslení. To je možné provést pomocí `S` nebo `f` nebo `B`.
- Po vykreslení křivky její data z paměti zmizí.
- Neuzavřou-li se souvislé části křivky pomocí `h` a použije-li se příkaz `f` nebo `B`, provede rasterizér uzavření každé jednotlivé části (oddělené příkazy `m`) samostatně.

Bézierova křivka tvořená pomocí $\langle x1 \rangle \langle y1 \rangle \langle x2 \rangle \langle y2 \rangle \langle x3 \rangle \langle y3 \rangle$ `c` je určena počátečním bodem $\langle x0 \rangle \langle y0 \rangle$, který je roven aktuálnímu bodu kresby, dále koncovým bodem $\langle x3 \rangle \langle y3 \rangle$ a dvěma kontrolními body $\langle x1 \rangle \langle y1 \rangle$ a $\langle x2 \rangle \langle y2 \rangle$. Jak vypadá chování takové křivky, doporučuji čtenáři zjistit v nějakém interaktivním editoru pro vektorovou grafiku. Je vhodné vědět, že spojnice $\langle x0 \rangle \langle y0 \rangle$ -- $\langle x1 \rangle \langle y1 \rangle$ je tečnou křivky v počátečním bodě a stejně tak spojnice $\langle x2 \rangle \langle y2 \rangle$ -- $\langle x3 \rangle \langle y3 \rangle$ je tečnou křivky v koncovém bodě. Počátečním a koncovým bodem křivka prochází a kontrolními body obvykle neprochází (ty křivku jen „přitahují“). Matematicky je křivka grafem parametricky zadaného polynomu třetího stupně (Bézierova kubika).

PDF rasterizér disponuje jedním složeným příkazem $\langle x \rangle \langle y \rangle \langle dx \rangle \langle dy \rangle \text{ re}$, který je zkratkou za $\langle x \rangle \langle y \rangle \text{ m } \langle x+dx \rangle \langle y \rangle \text{ l } \langle x+dx \rangle \langle y+dy \rangle \text{ l } \langle x \rangle \langle y+dy \rangle \text{ l h}$ a používá se k přípravě obdélníka.

• **Transformační matice** • O transformační matici byla již zmínka v sekci 11.6 v souvislosti s příkazem `\pdfsetmatrix`. Tento příkaz pracuje s maticí 2×2 a dovoluje jen lineární transformace. Na druhé straně PDF elementární operátor

$\langle a \rangle \langle b \rangle \langle c \rangle \langle d \rangle \langle e \rangle \langle f \rangle \text{ cm}$

pracuje s maticí 3×3 a umožňuje lineární transformace a posunutí. Matice se doplní na třetím řádku čísly $0 \ 0 \ 1$. Bod se souřadnicemi (x, y) se transformuje na bod se souřadnicemi (x', y') dle následujícího maticového násobení:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Vidíme, že údaje a, b, c, d realizují lineární transformaci a dále se provede posunutí o vektor (e, f) . Následující matice provádějí jednoduché transformace:

```
1 0 0 1  $\langle e \rangle$   $\langle f \rangle$  cm    % posunutí o vektor ( $\langle e \rangle$ ,  $\langle f \rangle$ )
0 1 -1 0 0 0 cm           % rotace o 90 stupňů v kladném směru
 $\langle a \rangle$  0 0  $\langle d \rangle$  0 0 cm    % škálování  $\langle a \rangle$  krát ve směru x a  $\langle d \rangle$  krát ve směru y
-1 0 0 1 0 0 cm           % zrcadlení podle osy y
1 0 0 -1 0 0 cm           % zrcadlení podle osy x
 $\langle \cos \alpha \rangle$   $\langle \sin \alpha \rangle$  - $\langle \sin \alpha \rangle$   $\langle \cos \alpha \rangle$  0 0 cm % rotace o úhel  $\alpha$  v kladném směru
```

PDF rasterizér udržuje v paměti aktuální transformační matici a každá další aplikace operátoru `cm` způsobí pronásobení aktuální matice zleva maticí sestavenou z parametrů operátoru `cm`. To odpovídá skládání jednotlivých zobrazení.

Existují dva pohledy na aplikaci transformační matice. Podle jednoho pohledu každý bod s danými souřadnicemi transformujeme podle výše uvedeného maticového násobení a dostáváme souřadnice, kam máme bod nakreslit. Druhý pohled interpretuje transformaci jako změnu souřadnicového systému. Matice aplikovaná pomocí `cm` změní souřadnicový systém následovně: v prvních dvou sloupcích matice čteme směrové vektory nových os (jednotky na těchto osách odpovídají velikosti směrových vektorů) a v posledním sloupci přečteme souřadnice nového počátku. Dále si představíme nový souřadnicový systém a veškeré údaje příkazů `m`, `l`, `c` nyní vztahujeme k tomuto novému souřadnicovému systému.

Oba pohledy ilustrujeme na matici

```
72 0 0 -72 0 0 cm
```

První pohled: Například bod o souřadnicích $(2, 3)$ se transformuje na bod o souřadnicích $(144, -216)$. Druhý pohled: Původní souřadný systém měl jednotku $1/72$ palce. Nový souřadný systém má jeden směrový vektor $(72, 0)$ a ten tvoří novou jednotku v nové ose x . Ta má stejný směr, jako původní osa x , tedy doprava. Druhý směr je $(0, -72)$, takže nová osa y má stejnou jednotku, ale je orientovaná nikoli nahoru ale dolů. Počátek souřadného systému zůstává na stejném místě. Máme-li nyní nakreslit bod o souřadnicích $(2, 3)$, provedeme to přímočaře v novém souřadném systému, v nových jednotkách a směrech, tedy v palcích: dva palce doprava a tři dolů. Oba pohledy samozřejmě vedou ke stejnému výsledku.

Jako příklad uvedeme možnost změnit souřadný systém pro kresbu z původních jednotek `bp` na `TEX`ovsky přítulnější jednotky `pt`. Vytvoříme makro `\koso{ $\langle velikost \rangle$ }`, které

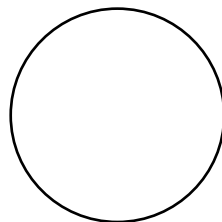
vytvoří čtverec o straně *⟨velikost⟩* otočený o 45 stupňů. Pochopitelně to lze udělat jednoduše pomocí `\vrule` otočené o 45°, ale z důvodu ukázky, jak mohou T_EXové jednotky spolupracovat s jednotkami PDF rasterizéru, zapomeneme na chvíli na to, že máme primitivní příkaz `\vrule`. Uživatel může použít makro `\koso{2mm}` nebo `\koso{\parindent}` a nemůžeme ho nutit, aby nám své rozměry přepočítával do jednotek bp. O převod se musí postarat makro. Jakýkoli rozměr v T_EXu můžeme uložit třeba do `\dimen0` a pak pomocí `\the\dimen0` jej vypsat. To nám T_EX ochotně udělá v jednotkách pt a tuto jednotku navíc připojí. My potřebujeme symbol pt jednak odpojit a jednak nastavit transformační matici tak, aby odpovídající rozměry bylo možno zadávat přímo v pt.

```
\def\koso#1{\dimen0=#1\relax
  \hbox to1.4142\dimen0{\hss\ vbox to1.4142\dimen0{\vss \kosoA}\hss}}
\def\kosoA{\pdfliteral{q                % pdfsave
  0.9963 0 0 0.9963 0 0 cm              % přechod z bp na pt
  0.7071 0.7071 -0.7071 0.7071 0 0 cm  % otočení o 45 stupňů
  0 0 \nopt\dimen0, \nopt\dimen0, re f % nakreslení obdélníka
  Q                                      % pdfrestore
}}
\def\nopt#1,{\expandafter\ignorept\the#1 }
{\lccode'\?='\p \lccode'\!='\t \lowercase{\gdef\ignorept#1?!{#1}}}
```

Vidíme, že převod souřadnic probíhá na úrovni příkazu cm, přitom číslo 0,9963 je přibližně rovno zlomku 72/72,27. To souvisí s tím, že pt má rozměr 1/72,27 palce a bp má rozměr 1/72 palce. Zbýlý kód makra obsahuje již jen drobné triky. Především v makru `\koso` je třeba T_EXovsky vytvořit box potřebné šířky a výšky, což je uděláno pomocí vnořených `\hbox` a `\vbox`, které mají výšku i šířku $\sqrt{2}$ krát větší než zadaný rozměr strany čtverce. Vlastní kresba se pak odehrává dole uprostřed tohoto boxu jako makro `\kosoA`. V registru `\dimen0` je uložen požadovaný rozměr. Je-li tímto rozměrem třeba 2mm, T_EX pomocí `\the\dimen0` vypíše 5.69054pt. My ale potřebujeme odstranit písmena pt, která by v PDF kódu překážela, a vložit jen 5.69054. Od toho slouží makro `\ignorept` (jeho definice je převzata z O_Pmac). Makro `\nopt`, které je nakonec v PDF kódu použito, vezme registr typu *⟨dimen⟩* až po čárku a odebere mu jednotku pt. Mezera za čárkou už je platná a odděluje parametry v PDF kódu.◆

V dalším příkladu vytvoříme kružnici. Kresba kružnic nebo jejich částí není podpořena přímo PDF elementárním příkazem a je nutno ji nahradit pro každou čtvrtinu kružnice příkazem `c` (curveto) s vhodnou polohou kontrolních bodů. Matematicky není možno dosáhnout Bézierovou kubikou přesné kružnice, přesto je následující aproximace tak dokonalá, že to oko nevidí:

```
\def\circle{.5 0 m
  .5 .276 .276 .5 0 .5 c
  -.276 .5 -.5 .276 -.5 0 c
  -.5 -.276 -.276 -.5 0 -.5 c
  .276 -.5 .5 -.276 .5 0 c
}
\pdfliteral{q 80 0 0 80 0 0 cm .0125 w \circle S Q}
```



V makru `\circle` je připravena kružnice o průměru 1. Před jejím použitím je pomocí transformační matice realizováno zvětšení, takže kružnice bude mít průměr 80 bp. Aby měla čáru tloušťky 1 bp, musíme zadat pro příkaz `w` převrácenou hodnotu zvětšení. Nakreslit kružnici libovolného průměru T_EXovým makrem je dále jednoduchým cvičením pro zručného T_EXistu.

• **Rámeček s oblými kouty** • Pro nakreslení rámečku **s oblými kouty** by se hodilo, kdybychom mohli argumenty příkazů `lineto` a `curveto` zapisovat relativně k aktuálnímu bodu kresby, nikoli k počátku. Tedy například místo $\langle x \rangle \langle y \rangle 1$ by byl užitečnější příkaz $\langle dx \rangle \langle dy \rangle dl$, který by vedl úsečku z bodu $\langle x0 \rangle \langle y0 \rangle$ (aktuálního bodu kresby) do bodu $\langle x0+dx \rangle \langle y0+dy \rangle$. To ale PDF rasterizér neumí. O přepočítání do absolutních souřadnic se tedy musí postarat \TeX . Připravíme si makra

```
\dmoveto \langle x \rangle, \langle y \rangle, % nastavení aktuálního bodu kresby x0 y0
\dlineto \langle dx \rangle, \langle dy \rangle, % úsečka relativně k aktuálnímu bodu kresby x0 y0
\dcurveto \langle dx1 \rangle, \langle dy1 \rangle, \langle dx2 \rangle, \langle dy2 \rangle, \langle dx3 \rangle, \langle dy3 \rangle, % křivka relativně k x0 y0
```

Makra předpokládají, že rasterizér pracuje v souřadnicích s jednotkou pt, takže je potřeba předřadit příslušnou matici transformace (z bp do pt) a využít makra `\nopt` z předchozího příkladu.

```
\newdimen\cpX \newdimen\cpY % souřadnice aktuálního bodu kresby
\def\dmoveto #1,#2,{\cpX=#1\cpY=#2\pdfliteral{\nopt\cpX,\nopt\cpY,m}}
\def\dlineto #1,#2,{\advance\cpX by#1\advance\cpY by#2%
\pdfliteral{\nopt\cpX,\nopt\cpY,l}}
\def\dcurveto #1,#2,#3,#4,#5,#6,{%
{\advance\cpX#1\advance\cpY#2\pdfliteral{\nopt\cpX,\nopt\cpY,}}%
{\advance\cpX#3\advance\cpY#4\pdfliteral{\nopt\cpX,\nopt\cpY,}}%
\advance\cpX#5\advance\cpY#6\pdfliteral{\nopt\cpX,\nopt\cpY,c}}
```

Údaje pro kontrolní body při `\dcurveto` jsou přepočítány uvnitř skupiny, takže jsou všechny relativní k počátku křivky $\langle x0 \rangle \langle y0 \rangle$, nikoli relativní jeden k druhému.

Vlastní rámeček se zaoblenými kouty je dán následujícími parametry, které může uživatel měnit:

```
\newdimen\rfR \rfR=5pt % poloměr zaoblených rohů
\newdimen\rfM \rfM=1pt % okraje mezi boxem a čarou rámečku
\def\rfType{1 1 0 rg 1 0 0 RG 1 w} % barvy plochy a čáry a šířka čáry
```

Následuje kód makra `\roundedframe{<text>}`, který vykreslí rámeček. Rámeček je zahájen kresbou levého horního rohu (jeho spodní částí). K tomu účelu musíme umístit počáteční bod na souřadnice 0 $\langle výška \rangle$, kde tato $\langle výška \rangle$ je rovna výšce boxu plus velikost okraje `\rfM` mínus poloměr zaoblení `\rfR`. Parametr $\langle výška \rangle$ je připraven v `\dimen2`. Podobně jsou v `\dimen1` a `\dimen3` předpočítány další parametry kresby.

```
\def\roundedframe#1{\setbox0=\hbox{\strut#1}%
\hbox{\drawroundedframe \kern\rfM \box0 \kern\rfM}}
\def\drawroundedframe{\dimen0=\rfR \advance\dimen0 by-\rfM
\dimen1=\wd0 \advance\dimen1 by-2\dimen0 % délka vodorovné linky
\dimen2=\ht0 \advance\dimen2 by-\dimen0 % výška počátečního bodu
\dimen3=\dp0 \advance\dimen3 by-\dimen0
\advance\dimen3 by\dimen2 % délka svislé linky
\pdfliteral{q 0.996264 0 0 0.996264 0 0 cm \rfType}% parametry
\dmoveto 0pt,\dimen2,% výchozí bod
\dcurveto 0pt,.5\rfR, .5\rfR,\rfR, \rfR,\rfR,% levý horní roh
\dlineto \dimen1,0pt,% vodorovná linka
\dcurveto .5\rfR,0pt, \rfR,-.5\rfR, \rfR,-\rfR,% pravý horní roh
\dlineto 0pt,-\dimen3,% svislá linka
\dcurveto 0pt,-.5\rfR, -.5\rfR,-\rfR, -\rfR,-\rfR,% pravý dolní roh
\dlineto -\dimen1,0pt,% vodorovná linka
\dcurveto -.5\rfR,0pt, -\rfR,.5\rfR, -\rfR,\rfR,% levý dolní roh
\pdfliteral{h B Q}}% close fill+stroke
```

Rámeček `\roundedframe{<text>}` se z hlediska \TeX u chová jako `\hbox{<text>}`, takže je možné jej použít třeba pro vyznačení tlačítka v textu odstavce. Pokud chceme do rámečku schovat celý `\vbox`, je třeba psát `\roundedframe{\vbox{<text>}}`.

● **Návaznost na Inkscape** ● Můžeme třeba řešit typografický požadavek na vkládání jednoduchých piktogramů do sazby. Existují dvě možná řešení, která člověka napadnou:

- Nakreslit piktogramy nějakým grafickým editorem a vložit je do sazby pomocí příkazu `\pdfimage`.
- Použít \TikZ [20] nebo \METAPOST [4] nebo něco podobného a obrázky naprogramovat přímo v makrech \TeX u (nebo \METAPOST u).

První způsob má *nevýhodu*, že vzniká sada externích souborů, se kterými je třeba při sazbě nějak manipulovat: umístit je na potřebné místo, kde je pdf \TeX najde, archivovat je společně s makry, atd.

Druhý způsob má *nevýhodu*, že programování výtvarně pojatých obrázků bez viditelných matematických zákonitostí je poněkud šílené, mnohdy až nemožné. Zejména, pokud si člověk uvědomí, že v grafickém interaktivním editoru má totéž vytvořeno za pár minut.

Doporučuji tedy postup třetí, který vylučuje nevýhody obou předchozích postupů a spojuje jejich výhody. Nakreslete si potřebný piktogram v grafickém editoru Inkscape¹). Pak proveďte export do `.eps`. Když tento EPS soubor otevřete textovým editorem, shledáte, že tam je celý obrázek nakreslen klasickým PDF kódem. Stačí tedy vyhledat první `q` a jemu odpovídající poslední `Q` a tento blok přesunout do argumentu `\pdfliteral` v makrech, která se starají o ty piktogramy. A je vymalováno. Doslova. Žádné načítání složitých maker typu \TikZ , žádné „programování“ obrázků, žádné starosti s externími obrázky. \TeX ová makra řeší piktogramy ve vlastní režii.

● **Poznámka** ● V článku [16] je uvedeno daleko více možností souvisejících s tvorbou grafiky v pdf \TeX u: opakované *<PDF kódy>* řešené odkazem, barevné přechody, ořezy podle deklarované křivky, grafika závislá na poloze textu atd.

11.10 Mikrotypografická rozšíření

Mikrotypografická rozšíření v pdf \TeX u jsou dvojího druhu:

- možnost vystrčení vybraných znaků o vybrané hodnoty do okraje a stanovení dalšího speciálního chování vybraných znaků zejména v návaznosti na mezery,
- možnost mírné deformace písma v mezích stanovené tolerance, která kompenzuje při formátování textu do bloku pružnost mezer, takže mezery nemusejí být příliš široké nebo příliš úzké.

Tato rozšíření (na rozdíl od všech předchozích) nemají přímou návaznost na výstupní PDF formát a jsou přidanou hodnotou pdf \TeX u. Autor pdf \TeX u Hàn Thê Thành v rámci výzkumu těchto typografických vlastností obhájil dizertaci na Masarykově univerzitě v Brně²). Uvedená rozšíření zde probereme jen encyklopedicky. Případní zájemci si podrobnosti vyhledají v manuálu k pdf \TeX u [21].

● **Vystrčení vybraných znaků** ● Příkazem `\rppcode{}<kód znaku>=<velikost>` se přidělí znaku z daného *<fontu>* následující vlastnost: vyskytne-li se při zlomu odstavce daný znak na konci řádku, je vysunut do okraje (doprava) o stanovenou *<velikost>*. Pro splnění tohoto požadavku je celý řádek formátován s jinak vypruženými mezerami. Analogicky

¹) <http://inkscape.org/>

²) Školitelem byl prof. Zlatuška.

`\lpcode<kód znaku>=<velikost>` vysune uvedený znak, vyskytuje-li se jako první na řádce, do okraje (doleva) o stanovenou *<velikost>*. Údaj *<velikost>* je celé číslo vyjadřující velikost v tisícinách em daného fontu a je v rozsahu -1000 až 1000 (větší či menší hodnoty se interpretují jako krajní mez). Nastavení `\lpcode` a `\rprcode` se projeví až po vložení kladné hodnoty do registru `\pdfprotrudechars` (je implicitně nastaven na nulu). Při `\pdfprotrudechars=1` pdfTeX provádí dodatečnou korekci jednotlivých řádků po vyhodnocení odstavce a při `\pdfprotrudechars=2` navíc algoritmus na vyhledávání řádkového zlomu s nastavenými hodnotami `\lpcode` a `\rprcode` přímo počítá.

Předpokládejme, že jsme zjistili, že ve fontu `\tenrm` zabírají české uvozovky velikost 0,5em a chceme je mít (při jejich výskytu na začátku či konci řádku) zcela vystrčeny do okraje. Pak můžeme psát:

```
\rprcode\tenrm\crqq=500
\lpcode\tenrm\clqq=500
\pdfprotrudechars=2
```

Rozměr znaku lze zjistit vložení znaku do boxu a změřením jeho šířky. Je asi vhodné vytvořit si makro, které vkládá údaje `\rprcode` a `\lpcode` relativně vzhledem k šířce daného znaku:

```
\newcount\tmpnum \newdimen\tmpdim % deklarace z OPmac
\def\setcharcode#1#2#3{% #1: \whatcode, #2: char, #3: factor
  \setcharcodeforfont \tenrm #1#2{#3}%
  \setcharcodeforfont \tenbf #1#2{#3}%
  \setcharcodeforfont \tenit #1#2{#3}%
  \setcharcodeforfont \tenbi #1#2{#3}%
}
\def\setcharcodeforfont#1#2#3#4{%
  \setbox0=\hbox{#1#3}\tmpdim=#4\wd0 \tmpnum=\tmpdim
  \tmpdim=\fontdimen6#1% velikost 1em
  \divide\tmpdim by50 \multiply\tmpnum by20 \divide\tmpnum by\tmpdim
  #2#1\ifcat#3\clqq\else'\fi#3=\tmpnum
}
\chardef\hyph=\hyphenchar\font
\setcharcode \lpcode\clqq {.9}
\setcharcode \rprcode\crqq {.9}
\setcharcode \rprcode . {1}
\setcharcode \rprcode , {1}
\setcharcode \rprcode\hyph {0.7}
\pdfprotrudechars=2
```

Uvedený kód nastaví vystrkování uvozovek na 0,9 jejich šířky a dále vystrkování tečky, čárky a znaku pro dělení slov. To jsou obvyklé znaky, které po vystrčení způsobí, že odstavec vypadá opticky v bloku vyrovnaněji. Vyzkoušejte si to.

Ve stejném formátu jako `\rprcode<kód znaku>=<velikost>` je možné v pdfTeXu stanovit další vlastnosti znaků:

```
\knbscode ... přidaná velikost mezery těsně následující za znakem
\stbscode ... přidaná roztážitelnost mezery těsně následující za znakem
\shbscode ... přidaná stažitelnost mezery těsně následující za znakem
\knbccode ... přidaný kern před každým výskytem znaku
\knaccode ... přidaný kern za každým výskytem znaku
```

Aby se projevilo nastavení `\knbscode`, `\stbscode` a `\shbscode`, je třeba dát registru `\pdfadjustinterwordglue` kladnou hodnotu. Aby se projevilo nastavení `\knbccode`, je

třeba dát kladnou hodnotu registru `\pdfprependkern` a totéž platí o `\pdfappendkern` v souvislosti s nastavením `\knaccode`. Příklad:

```
\setcharcode \knbccode ? {.2} \setcharcode \knbccode ! {.4}
\setcharcode \knbccode : {.3} \setcharcode \knbccode ; {.3}
\pdfprependkern=1
```

vytvoří nálitky před znaky otazník, vykřičník, dvojtečka a středník.

• Deformace písma v mezích mírného pokroku • Příkazem

```
\pdffontexpand<font><roztahení> <stažení> <krok> autoexpand
```

se dá pdf_T_EXu najevo, že se má (při kladném registru `\pdfadjustspacing`) pokusit kromě mezer v řádcích odstavce deformovat ``. Údaje o maximálním `<roztahení>` a `<stažení>` jsou v tisícinách. Tj. hypoteticky při `<roztahení>` rovném 1000 je možné font deformovat až na dvojnásobnou šířku. Roztažení ani stažení se neděje spojitě, ale v diskrétních krocích. Údaj `<krok>` je taky v tisícinách. Příklad:

```
\pdffontexpand\tenrm 30 20 10 autoexpand \pdfadjustspacing=2
```

Po tomto nastavení se pdf_T_EX pokusí font `\tenrm` kromě přirozené velikosti používat v roztaženějších řádcích v šířkách o 1, 2 nebo 3 procenta větší a ve stlačenějších řádcích o 1 nebo 2 procenta menší. Více viz [21].

Některé znaky při deformaci vypadají tak, že si toho oko všimne dříve. Je tedy možné pomocí `\efcode<kód znaku>=<promile>` stanovit pro daný znak výjimku. Například při `<promile>=500` bude znak podléhat poloviční deformaci než všechny ostatní, které nemají nastaveno `\efcode`.

Cílem této úpravy pdf_T_EXu je jen taková deformace, *kteou oko nepostřehne*, ale pomůže k lepšímu řešení mezerislovních mezer.

• Prostrkaný font • Příkazem

```
\letterspacefont<nový font><font><velikost>
```

je možné deklarovat `<nový font>` jako původní ``, ale mezi znaky budou vloženy dodatečné mezery specifikované jako `<velikost>` (v tisícinách em). Údaj `<velikost>` může být i záporný, tj. znaky budou více přisazeny k sobě. V dávných dobách se prostrkaný text používal k vyznačování (zejména na psacích strojích), ale dnes pět typografů z šesti to nedoporučuje.

Dodatek A

Generování formátů

T_EX se při zpracování dokumentů nespouští samostatně, ale typicky s nějakým předgenerovaným formátem. Například T_EX s formátem C_Splain spouštíme příkazem `csplain`, T_EX s formátem L^AT_EX spouštíme příkazem `latex`. Výjimkou z tohoto pravidla je T_EX s formátem plain (Knuthův původní formát k T_EXu), který se spustí příkazem `tex`, nikoli `plain`. Jednotlivé formáty je potřeba předgenerovat, což dělá obvykle distribuce T_EXu automaticky bez přispění uživatele. Následující informace dávají čtenáři přehled o koncepci T_EXových formátů a umožní mu generovat je ručně.

A.1 Módy INITEX a VIRTEX

Binární programy `tex`, `pdftex`, `xetex` a `luatex` pracují ve dvou módech: INITEX a VIRTEX. V módu INITEX tyto programy připravují formát a v módu VIRTEX formát použijí a zpracují dokument. K rozlišení těchto módů se typicky používá přepínač `-ini` na příkazové řádce: je-li přítomen, program pracuje v módu INITEX, a není-li přítomen, program pracuje v módu VIRTEX.

V módu INITEX je program při svém startu vybaven schopností interpretovat jen primitivní příkazy a interní registry. Těch je zhruba 300. Během čtení souborů se T_EX „učí“ další makra, deklarované registry atd. a tím rozšiřuje repertoár řídicích sekvencí, kterým rozumí. Po přečtení souboru `plain.tex` je takových řídicích sekvencí zhruba 900. Nepřesně říkáme této činnosti *proces učení maker*. Přesněji jde o následující činnosti.

- Definice maker, deklarace registrů a znakových konstant. Toto je možné v obou módech INITEX i VIRTEX.
- Čtení fontových metrických údajů příkazem `\font` a deklarace přepínačů fontů. I toto je možné v obou módech INITEX i VIRTEX.
- Čtení vzorů dělení slov pro různé jazyky příkazem `\patterns`. Toto je možné jen v módu INITEX.

Generování formátu v módu INITEX je ukončeno příkazem `\dump`. V takovém okamžiku T_EX uloží nabyté vědomosti včetně dosud načtených fontů a vzorů dělení slov do binárního souboru s příponou `.fmt` a se jménem shodným se jménem hlavního souboru (není-li toto jméno pozměněno přepínačem `-jobname`) a ukončí činnost.

Zpracování dokumentu pak probíhá v módu VIRTEX tak, že T_EX nejprve načte předgenerovaný binární formát, takže na začátku zpracování dokumentu vychází ze znalostí, které nabyl v okamžiku příkazu `\dump`. Původní smysl rozdělení procesu učení do těchto dvou fází (generování formátu a zpracování dokumentu) vycházel zřejmě z požadavku na rychlost zpracování. Skutečně, před desítkami let trvalo i několik minut, než byl formát vygenerován. Dnes je tento požadavek asi irelevantní, protože formát je vygenerován ve zlomku sekundy. A tak zde máme relikť z minulosti: dvě fáze procesu učení T_EXu.

V následujícím příkladě předpokládáme, že soubor `cosi.ini` obsahuje definice maker a znakových konstant, případně také příkazy `\font` pro zavedení fontů a příkazy `\patterns` pro zavedení vzorů dělení slov. Soubor je ukončen příkazem `\dump`.

```
tex -ini cosi.ini          ... vygenerování formátu cosi.fmt v módu INITEX
tex -fmt cosi dokument    ... užití formátu cosi.fmt v módu VIRTEX
```

Jak bylo řečeno, ve VIRTEX módu se T_EX nejprve shání po formátu `.fmt`. Buď je jeho jméno stanoveno za přepínačem `-fmt` nebo, není-li tento přepínač uveden, je jméno formátu shodné s názvem, pod kterým je program spuštěn. Například:


```

tex dokument                ... přečte tex.fmt a následně dokument.tex
pdftex dokument            ... přečte pdftex.fmt a následně dokument.tex
pdftex -fmt csplain dokument ... přečte csplain.fmt a následně dokument.tex

```

Protože je soubor s formátem binární (je to vlastně „dump“ interní paměti \TeX u na konci generování formátu), není možné předložit tento formát ke čtení jiné verzi \TeX u, než která verze formát vytvořila. Pokud se o to pokusíte, dočkáte se hlášení:

```
Fatal format error; I'm stymied.
```

Vzhledem k tomu, že v současné době je \TeX implementován různými programy (`tex`, `pdftex`, `luatex`, `xetex`), není uvedena chyba bohužel vyloučena. Pravidlo zní: pouze stejný program ve stejné verzi, jaký formát vygeneroval, jej může použít. Spuštěný program ve VIRTEX módu se pokusí nejprve najít soubor `.fmt` v aktuálním adresáři a pokud tam není (což je typické), najde jej v distribuci v rezervovaném adresáři. Jak se tento adresář přesně jmenuje a jaký je systém prohledávání takových adresářů záleží na použité distribuci a její konfiguraci. Typicky jsou v distribuci rezervovány různé adresáře pro různé \TeX ové programy, aby nedošlo k výše uvedené chybě.

Odlišnosti módů INITEX a VIRTEX lze shrnout následujícím způsobem. Mód INITEX rozumí na rozdíl od módu VIRTEX příkazu `\patterns` pro čtení vzorů dělení slov a příkazu `\dump` pro uložení vygenerovaného formátu. Mód VIRTEX umí na rozdíl od módu INITEX číst předgenerované formáty a vždy nějaký formát přečte. Oba módy dokáží vytvořit dokument, ale v módu INITEX to nebývá obvyklé, ačkoli to je možné, jak ukazuje druhý řádek příkladu.

```

tex -ini plain.tex "\dump"                ... vytvoří plain.fmt (formát)
tex -ini plain.tex "\input story.tex \end" ... vytvoří přímo dokument

```

V \TeX ových distribucích je soubor `plain.fmt` typicky přejmenován na `tex.fmt`, takže program `tex` spuštěný v módu VIRTEX bez použití přepínače `-fmt` přečte tento formát. Dále je v distribucích zařízeno, aby třeba příkaz `latex` ve skutečnosti spustil program `tex` (nebo nověji `pdftex`). Tento program je spuštěn v módu VIRTEX a vyhledá formát podle názvu volání, tedy vyhledá `latex.fmt`.

A.2 Generování formátu \mathcal{C} splain

Pojem \mathcal{C} splain používáme ve třech významech:

- je to balík `maker` a vzorů dělení slov a další podpory pro český a slovenský jazyk rozšiřující možnosti `plain \TeX` u, nebo
- je to příkaz (`csplain` nebo `pdfcsplain`), kterým zpracováváme dokument. Tento příkaz spustí \TeX , který použije
- binární soubor `csplain.fmt` nebo `pdfcsplain.fmt`, tedy předgenerovaný formát.

\TeX ová distribuce by měla mít obě mutace formátu \mathcal{C} splain (tj. \mathcal{C} splain a `pdf \mathcal{C} splain`) předgenerovány nebo by je měla umět automaticky vygenerovat v okamžiku prvního užití. Měla by nabízet příkaz `csplain`, který spustí `pdf \TeX` s formátem `csplain.fmt` (s implicitním výstupem do DVI), a dále příkaz `pdfcsplain`, který spustí `pdf \TeX` s formátem `pdfcsplain` (s implicitním výstupem do PDF). Uživatel by se tedy nemusel zajímat o to, jak formáty vygenerovat ani kam je umístit, aby vše fungovalo.

Pokud podmínky předchozího odstavce nejsou splněny nebo pokud má uživatel speciální požadavky, pak je možné vygenerovat formáty pomocí nějakého nástroje v použité \TeX ové distribuci nebo pomocí příkazového řádku. Popíšeme druhou možnost.

Pro vygenerování formátu

```

1 csplain.fmt      ... vstup: UTF-8, výstup: DVI, stroj: pdfTeX+encTeX,
2 pdfcsplain.fmt  ... vstup: UTF-8, výstup: PDF, stroj: pdfTeX+encTeX,
3 pdfcsplain.fmt  ... vstup: UTF-8, výstup: PDF, stroj: LuaTeX,
4 pdfcsplain.fmt  ... vstup: UTF-8, výstup: PDF, stroj: XeTeX,

```

je potřeba použít následující příkaz:

```

1 pdftex -jobname csplain -ini -enc csplain-utf8.ini
2 pdftex -jobname pdfcsplain -ini -enc csplain-utf8.ini
3 luatex -jobname pdfcsplain -ini csplain.ini
4 xetex -jobname pdfcsplain -ini -etex csplain.ini

```

Tyto příkazy uloží formát `csplain.fmt` nebo `pdfcsplain.fmt` do aktuálního adresáře, odkud je pak \TeX dovede číst. Ovšem asi nebudete chtít generovat v každém adresáři formáty znova, takže bude vhodné je umístit „někam do distribuce“ \TeX u, odkud je \TeX také dovede číst. Přesná lokace záleží na použité distribuci a její konfiguraci a koncepci. Například \TeX live má „někde“ adresář `web2c` a pod ním podadresáře `pdftex`, `luatex`, `xetex` a do nich patří umístit vygenerované formáty (podle názvu stroje, který jej vygeneroval). Pak je nutné spustit příkaz `texhash` pro aktualizaci vyhledávacích tabulek.

Příkaz pro zpracování souboru `dokument.tex` při použití výše uvedených formátů vypadá takto:

```

1 pdftex -fmt csplain dokument      nebo      csplain dokument
2 pdftex -fmt pdfcsplain dokument  nebo      pdfcsplain dokument
3 luatex -fmt pdfcsplain dokument
4 xetex -fmt pdfcsplain dokument

```

Je dobré si všimnout, že formáty 1 a 2 jsou vygenerované stejným strojem a jeho implicitní výstup (DVI nebo PDF) se nastaví automaticky jen podle názvu formátu, který je při generování specifikován přepínačem `-jobname`. Jsou-li první tři písmena názvu `pdf`, bude implicitní výstup v PDF, jinak bude výstup v DVI.

● **Cvičení 16** ● Prozkoumejte `.log` soubory vzniklé při generování formátů `csplain` a `pdfcsplain` a najdete rozdíl.

● **Poznámka** ● Běžně užívané příkazy `csplain`, resp. `pdfcsplain` bývají obvykle zkratkami za `pdftex -fmt csplain` resp. `pdftex -fmt pdfcsplain`. Způsob implementace těchto zkratk závisí na operačním systému a na \TeX ové distribuci.

● **Cvičení 17** ● Prozkoumejte, jak jsou implementovány uvedené příkazy v použité distribuci.

● **Poznámka** ● $\text{Xe}\TeX$ a $\text{Lua}\TeX$ nemá smysl nutit vytvářet DVI, protože taková DVI nejsou rozumně zpracovatelná v okamžiku, kdy se použije font v kódování Unicode. Přitom češtinu nebo slovenštinu nelze v těchto rozšířeních provozovat jinak než s takto kódovými fonty. Viz též dodatek B a důležitou poznámku na straně 118.

● **Poznámka** ● UTF-8 vstup je při použití $\text{pdf}\TeX$ u možný jen tak, že se při generování formátu aktivuje jeho rozšíření `encTeX` [14], viz též dodatek E. To je zařízeno přepínačem `-enc` na příkazové řádce a dále v souboru `csplain-utf8.ini` je nastaveno UTF-8 kódování pomocí `\let\enc=u`. Pokud byste chtěli vygenerovat formát se vstupem jiným než UTF-8, můžete si uvedený soubor `csplain-utf8.ini` zkopírovat do jiného souboru a místo `\let\enc=u` psát `\let\enc=w` (pro kódování CP1250), `\let\enc=p` (pro kódování CP852) nebo nic (pro kódování ISO-8859-2). Takto upravený \mathcal{C} splain ovšem neumožňuje přepnout kódování fontů v dokumentu pomocí `\input t1code`.

● **Poznámka** ● Formát \mathcal{C} splain je možné vygenerovat s více vzory dělení, než jen čeština, slovenština a angličtina. O tom pojednává dodatek G.

Dodatek B

Všeliká rozšíření T_EXu

Autor T_EXu Donald Knuth zmrazil v roce 1989 vývoj T_EXu, neboť považuje za užitečné vytvořit jistý „pevný bod v čase“. Podrobnější argumenty najdete na jeho [www stránce](#). Posléze vznikla různá rozšíření T_EXu, která se nesmějí (dle přání autora) přímo nazývat T_EX, ale slovo T_EX mohou ve svém názvu obsahovat. Vesměs všechna rozšíření přidávají další sadu primitivních příkazů a rozšiřují vlastnosti T_EXu. Mezi první taková rozšíření patří eT_EX (též zvaný ε -T_EX), viz sekci B.1. Velkým přínosem byl vznik pdfT_EXu, který umožňuje přímý výstup do PDF formátu (viz kapitolu 11). Dále vznikl encT_EX (viz přílohu E) a konečně dnes asi nejvíce hýbou T_EXovým světem X_YT_EX (viz sekci B.2) a LuaT_EX (viz sekci B.3). Poslední dvě rozšíření interně pracují v Unicode, umožňují přímé použití fontů ve formátu OTF a vystupují do PDF. O jednotlivých rozšířeních je zde uvedeno jen základní shrnutí. Podrobnější informace je třeba vyhledat v dokumentaci [7, 9, 14, 18, 21].

B.1 eT_EX

Rozšiřující primitivní příkazy eT_EXu jsou dnes využívány standardně v makrech L^AT_EXu. Jinak řečeno, L^AT_EX si nevystačí se základním Knuthovým T_EXem. Formát C_Splain eT_EX nepotřebuje, ale je možné jej vygenerovat i s podporou eT_EXu a využít tím další možnosti.

Rozšíření eT_EX je dnes běžnou součástí binárních programů **pdftex**, **xetex** a **luatex** ve většině T_EXových distribucí. Aby bylo možno eT_EX využít, je potřeba jej „probudit k životu“ při generování formátu, obvykle přepínačem **-etex** nebo hvězdičkou na začátku názvu vstupního souboru. Není nutné znovu psát přepínač **-etex** při použití takto vygenerovaného formátu, eT_EX je už připraven přímo k použití. Binárky **pdftex** a **xetex** se tedy chovají standardně „knuthovsky“ a s nedostupným eT_EXem, pokud se při generování formátu nenapíše příslušný přepínač. V LuaT_EXu se aktivace eT_EXu provede jiným způsobem (viz sekci B.3).

Je-li eT_EX probuzen, nabízí řadu dalších příkazů. Nejdůležitější z nich jsou zmíněny v následujícím textu. Počet alokovatelných registrů typu `count`, `dimen` atd. je v klasickém T_EXu roven 256, zatímco eT_EX rozšiřuje jejich počet na 32 tisíc.

Pomocí příkazů eT_EXu `\numexpr` a `\dimexpr` je možné zapisovat výpočet numerických a metrických hodnot přímočařeji. Možnosti nejsou větší, než co se dá umlátit pomocí příkazů `\advance`, `\multiply` a `\divide`, ale zápis je přehlednější. Například

```
\numexpr 3*(\pageno-10)\relax
```

dá trojnásobek čísla strany nejprve zmenšený o 10. Je tedy možné použít závorky a znaky `+`, `-`, `*` a `/`. Výraz je ukončen `\relax` nebo jiným objektem, který syntakticky neodpovídá konstrukci výrazu. Značná výhoda je v tom, že výpočet proběhne na úrovni expanze makra, zatímco příkazy `\advance`, `\multiply`, `\divide` neexpandují. Takže třeba po `\edef\alpha{\the\numexpr\pageno/7}` máme v makru `\alpha` přímo hodnotu sedminy aktuální strany (zaokrouhlenou od poloviny nahoru, jinak dolů). Nebo je možné pomocí `\ifdim\dimexpr\hsize+\hoffset+1in>20cm` vyhodnotit, zda je pravý okraj sazby vzdálen od levého okraje papíru o více než 20 cm. Výrazy, jak je vidět z příkladů, mohou obsahovat konstanty nebo registry.

Je dovoleno též využít vnořené výrazy vymezené znovu pomocí příkazů `\numexpr` nebo `\dimexpr` a ukončené `\relax`. Následuje poněkud komplikovanější příklad, v němž je definováno makro `\dividedimen(<čítatel>/<jmenovatel>)`, které vypočítá poměr dvou metrických údajů, tedy poměr velikostí.

```

\def\dividedimen (#1/#2){\expandafter\ignorept\the % \ignorept je makro z OPmac
\dimexpr \numexpr \number\dimexpr#1\relax*65536 / \number\dimexpr#2\relax
\relax sp\relax
}
% Například:
\dividedimen (12cm/5cm) % expanduje na 2.4
\dividedimen (\hsize/1cm) % expanduje na 15.92, když je \hsize = 15.92cm

```

Makro `\dividedimen` využívá toho, že přechodné numerické registry alokované pomocí `\numexpr` pracují s 64bitovou aritmetikou, takže nedojde k přetečení numerické hodnoty v čitateli a výsledek je poměrně přesný. Navíc vše proběhne na úrovni expandování maker. Je tedy možné mít například nějakou změřenou velikost v registru `\tmpdim` a požadovanou velikost v `#1` a na základě toho vypočítat poměr zvětšení. Zvětšit jiný registr takovým poměrem lze pak snadno, například: `\fontdim=\dividedimen(#1/\tmpdim)\fontdim`.

Podmínky typu `\if...` mohou být v eTeXu uvozeny příkazem `\unless`, což způsobí negaci vyhodnocované podmínky. Takže lze psát:

```

\unless \ifx\makro\undefined zde je definováno \else a zde není definováno\fi

```

Dále eTeX zavádí novou podmínku `\ifdefined` makro, takže výše uvedenou podmínku lze zapsat přímočařeji. Dále `\ifcsname... \endcsname` testuje, zda je řídicí sekvence `\csname... \endcsname` definovaná. Pomocí `\iffontchar` `(font)<slot>` se dá zjistit, zda font obsahuje na daném slotu znak.

Příkaz `\unexpanded{<text>}` potlačí expanzi `<textu>` při `\edef`, `\write`, `\message` atd. Expanduje na `<text>`, který zůstane neexpandovaný. Tuto vlastnost je možno v klasickém TeXu dosáhnout protažením `<textu>` přes registr typu `<tokens>`. Například

```

\newtoks\mytoks \mytoks={text, který se nemá expandovat, např. \makro.}
\edef\af{\the\mytoks}
% zatímco v eTeXu stačí jednoduše:
\edef\af{\unexpanded{text, který se nemá expandovat, např. \makro.}}

```

Prefix `\protected` před příkazy `\def`, `\edef`, `\gdef`, `\xdef` způsobí v eTeXu, že makro je definováno jako neexpandovatelné v době `\edef`, `\write`, `\message` atd. Toto zabrání expanze není v klasickém TeXu přímočaře řešeno. Makrobalíky to řeší rozličným způsobem. Třeba L^AT_EX předhazuje před taková makra `\protect`, což je makro obsahující `\noexpand`. OPmac místo toho umožní uživateli použít `\addprotect\makro` a zařídí jeho neexpanzi ve vyjmenovaných situacích pomocí přechodného `\let\makro=\relax`.

Příkaz `\scantokens{<text>}` expanduje na `<text>` s aktuálně nastavenými kategoriemi všech jeho znaků. V klasickém TeXu je toto možné udělat jen zápisem `<textu>` do řádku pracovního souboru a následně přečtením tohoto textu příkazem `\input`.

Příkaz `\detokenize{<text>}` expanduje na `<text>` jako při `\scantokens{<text>}` a při nastavení kategorií všech znaků na 12 s výjimkou mezery, která má kategorii 10.

V eTeXu je připraven interní registr typu `<tokens>` `\everyeof`, který připojí svůj obsah na konec každého souboru čteného pomocí `\input`. Je výhodné tam dát zarážku pro načtení souboru do parametru makra:

```

\everyeof = {EndOfFile!}
\long\def\scanfile#1EndOfFile!{\def\filecontent{#1}}
\expandafter\scanfile \input file
\everyeof = {}

```

Nyní je v makru `\filecontent` uložen obsah celého souboru. Číst makrem až do konce souboru v klasickém TeXu nelze. Tam je třeba číst po jednotlivých řádcích a ptát se na konec souboru pomocí `\ifeof`.

Pokud čtený soubor neobsahuje nic, co by mohl \TeX expandovat, pak existuje přímočařejší řešení pro načtení celého souboru, využívající toho, že konec souboru označený pomocí `\noexpand` nezlobí:

```
{\everyeof={\noexpand}\xdef\filecontent{\input file }}
```

Pomocí `\lastlinefit` je možné v $e\TeX$ u nastavit poměr stlačení nebo roztažení mezer posledního řádku odstavce ve srovnání s předposledním řádkem. Poměr je dán v tisícinách, tj. při `\lastlinefit=1000` bude poslední řádek deformován stejně jako předposlední, při `\lastlinefit=500` bude mít poloviční deformaci mezer a při `\lastlinefit=0` nebude mít žádnou deformaci mezer. Poslední případ je implicitní nastavení a je to také jediné možné chování posledního řádku odstavce v klasickém \TeX u.

Klasický \TeX umísťuje stanovené penalty `\clubpenalty` a `\widowpenalty` pod první řádek odstavce a před poslední. Dále přidá `\interlinepenalty` mezi každý řádek odstavce. Naproti tomu $e\TeX$ nabízí možnost deklarovat penalty mezi všemi řádky odstavce pomocí pole penalt `\clubpenalties`, `\widowpenalties` a `\interlinepenalties`. Například nastavení:

```
\clubpenalties 2 10000 10000
\widowpenalties 2 10000 10000
```

způsobí zakázaný zlom za prvním i druhým řádkem odstavce a před posledním i předposledním řádkem odstavce.

$e\TeX$ nabízí do matematických výrazů obklopených `\left...\right` přidávat libovolné množství příkazů `\middle` (*závorka*). Tato závorka bude stejně veliká, jako obklopující závorky vytvořené v `\left...\right`.

Klasický \TeX počítá místa pro dělení slov po konverzi textu na malá písmena pomocí aktuálního nastavení `\lccode`. Naproti tomu $e\TeX$ umožní uložit nastavení `\lccode` společně se vzory dělení (pro každý jazyk třeba jinak) do formátu a pak použít konverzi na malá písmena podle toho a ne podle aktuálních hodnot `\lccode`. K tomu slouží registr `\savingshyphencodes`. Popsané chování se aktivuje po nastavení tohoto registru na kladnou hodnotu.¹⁾

$e\TeX$ dále obsahuje implementaci přepínání na pravolevý směr sazby. Vyšel z rozšíření \TeX u zvané \TeX - $X_{\mathcal{T}}$, ale nazval svůj modul pro změnu \TeX - $X_{\mathcal{T}}$. Směr sazby se nastavuje příkazy `\beginL`, `\endL`, `\beginR`, `\endR`. Arabové mají radost.

Konečně $e\TeX$ nabízí více možností pro programování plovoucích záhlaví (více než jeden příkaz `\mark`) a dále obohacuje ladicí výpisy (`\tracing...`) o některé další informace.

S $e\TeX$ em souvisí soubor maker `etex.src`, který mírně rozšiřuje původní `plainTeX` o některé další vlastnosti: možnost načtení a využití vzorů dělení slov více jazyků (definuje k tomu makra `\addlanguage` a `\uselanguage`), rozšířené využití registrů `\tracing...`, rozšířená makra pro alokaci $e\TeX$ ových registrů typu `\newcount`, `\newdimen` a možnost kontrolovaného načítání kódů maker pomocí makra `\load`. Svým přístupem minimálně rozšířit Knuthův `plainTeX` se dá tento soubor maker přirovnat k $\mathcal{C}\mathcal{S}\mathcal{P}\mathcal{L}\mathcal{A}\mathcal{I}\mathcal{N}$ u. Makro `etex.src` se načítá v \TeX ových distribucích jako výchozí formát pro `pdfTeX`, `X\mathcal{T}TeX` i `LuaTeX`. To jinými slovy znamená, že když napíšeme třeba příkaz `xetex` bez parametru `-ini` nebo `-fmt`, spustí se `X\mathcal{T}TeX` s formátem `xetex.fmt`, který je v distribuci generován za použití maker `etex.src`.

¹⁾ Detekuje-li $\mathcal{C}\mathcal{S}\mathcal{P}\mathcal{L}\mathcal{A}\mathcal{I}\mathcal{N}$ přítomnost $e\TeX$ u, je popsaná vlastnost zapnuta.

B.2 X_YTeX

X_YTeX (vyslovujeme zítech) vytvořil v roce 2004 Jonathan Kew. Nejprve pro MAC OS X a po dvou letech i pro Linux, odkud pramenil další port na MS Windows. X_YTeX obsahuje všechny vlastnosti eTeXu (pokud jsou inicializovány při generování formátu přepínačem `-etex`) a navíc přidává dvě další zásadní odlišnosti od klasického TeXu:

- Interně pracuje v Unicode, na vstupu předpokládá výhradně UTF-8 kódování. Zatímco klasický TeX umožňuje pracovat jen se znaky v rozsahu `\char0` až `\char255`, v případě X_YTeXu máme rozsah úplné Unicode tabulky, tj. `\char0` až `\char1114111`, tedy `\char"10FFFF`.¹⁾
- Je slinkován s knihovnou operačního systému, která umožňuje aplikacím přistupovat k fontům instalovaným v systému. X_YTeX umí pracovat jednak jako klasický TeX s fonty v TeXové distribuci, ale kromě toho lze za příkazem `\font` napsat i název fontu instalovaného v operačním systému. Toto byla tak trochu revoluce v TeXových vodách, protože dosud se TeX tvářil jako zarputile nezávislý na operačním systému se svým vlastním mechanismem práce s fonty na hony vzdáleném od toho, jak k fontům přistupují jiné aplikace a operační systém.

V souvislosti s druhou vlastností je třeba si uvědomit některá pro a proti. Nevýhodou je, že to uživatele svádí využít fonty v systému, které ovšem nemusí být instalovány v jiném systému, kam třeba bude potřeba zdrojový text dokumentu přemístit a pokračovat v TeXování. A i když tam stejné fonty budou, pak možná budou s jinými metrickými informacemi a dalšími fontovými atributy (protože v jiné verzi fontů nebo knihovny), takže jednou formátovaný text se může podruhé formátovat zcela jinak. Na druhé straně klasický TeX a pdfTeX se opírají jen o fonty v TeXových instalacích. Tam najdeme sice ne příliš bohatou, ale zato stabilní sadu fontů, která nepodléhá dalšímu vývoji. Naprosto fixovány (se zárukou stoprocentní neměnnosti) jsou Knuthovy fonty Computer Modern a i další běžné fonty z TeXových instalací dnes již nepodléhají změnám.

Jednoznačnou výhodou užití přímo systémových fontů je konec frustrací při instalování např. nově zakoupeného fontu do TeXové distribuce. To byla a je práce pro TeXového odborníka. Naproti tomu uživateli X_YTeXu pouze stačí instalovat font do operačního systému a může jej začít používat. Navíc má k dispozici všechny fontové atributy (font features), které nabízejí nejnovější fonty ve formátu OpenType. Operační systém rovněž obvykle nabízí interaktivní katalog nainstalovaných fontů, takže jej stačí rozkliknout a přepsat název fontu za příkaz `\font` do zdrojového textu dokumentu.

X_YTeX umožňuje využití příkazu `\font` jako v klasickém TeXu a kromě toho i rozšířené použití ve tvaru:

```
\font\prepinac="⟨jméno fontu⟩:⟨atributy⟩" ⟨at nebo scaled⟩
```

Přitom `⟨atributy⟩` (s dvojtečkou před) stejně jako `⟨at nebo scaled⟩` (s mezerou před) jsou nepovinné parametry. V následujícím příkladu pracujeme s fontem **Linux Libertine 0**.

```
% font Linux Libertine 0 ve velikosti 12.5pt:
\font\fnormal="Linux Libertine 0" at12.5pt
% Linux Libertine 0 zvětšený 1,2x:
\font\fscaled="Linux Libertine 0" scaled1200
% Kurzíva fontu Linux Libertine 0:
\font\fitalics="Linux Libertine 0 Italics"
% Kurzíva fontu Linux Libertine 0, jako v předchozím případě:
\font\fital="Linux Libertine 0/I"
```

¹⁾ Toto umí i LuaTeX, oba v tomto případě vycházejí ze společného prazákladu, projektu Omega.

```
% Zapnutý atribut smcp aktivující malé kapitálky:
\font\fcaps="Linux Libertine 0:+smcp"
% Atributy hlig;dlig;salt: historické, kontextové ligatury a variantní znaky:
\font\fspec="Linux Libertine 0:+hlig;+dlig;+salt"
% Vypnutý atribut liga, tj. font bez ligatur fi, fl atd:
\font\fnolig="Linux Libertine 0:-liga"
```

Jméno `Linux Libertine 0 Italics` je úplné jméno kurzívy daného fontu. Z příkladu je vidět, že stačí též uvést jen jméno rodiny následované modifikátorem `/I`. Podobně fungují modifikátory `/B` a `/BI` pro **Bold** a **BoldItalics**. Za těmito modifikátory může následovat dvojtečka a za ní atributy oddělené středníkem a uvozené symbolem `+` (vlastnost atributu zapínáme) nebo `-` (vlastnost atributu vypínáme). Atributy OpenType fontů jsou popsány například na webové stránce¹⁾. Jaké atributy jsou k dispozici v konkrétním fontu a jak jsou implicitně nastaveny, závisí na rozhodnutí písmolijny, která font vytvořila. Je ovšem možné použít příkaz `otfinfo -f soubor.otf` ke zjištění seznamu dostupných atributů. Další informace je možné najít na stránce seriálu o \TeX u²⁾.

Chceme-li, aby fungovaly \TeX ové ligatury (např. `--` se promění na `—`, `---` na `—`), je třeba doplnit atribut `mapping=tex-text`. Toto výjimečně není atribut implementovaný přímo ve fontu, ale rozumí mu \XTeX . Takže třeba:

```
\font\ftal="Linux Libertine 0/I:mapping=tex-text" at13pt
```

zavede kurzívu s \TeX ovými ligaturami. Přitom ligatury `fi`, `fl` fungují také, protože atribut `liga`, který je aktivuje, je implicitně zapnutý.

Místo názvu fontu je možné použít jméno souboru (bez přípony `.otf`). Název souboru je třeba obklopit hranatými závorkami, takže soubor `lmroman10-regular.otf` lze do \XTeX u zavést například pomocí

```
\font\frm="[lmroman10-regular]:mapping=tex-text" at13pt
```

Soubor vyhledá \XTeX v aktuálním adresáři nebo v \TeX ové distribuci v adresáři `.../fonts/otf/`.

• **Důležitá poznámka** • Při tvorbě českých nebo slovenských dokumentů je potřeba si uvědomit, že \XTeX (ani $\text{Lua}\TeX$) není jinak použitelný než se zavedenými fonty v Unicode. Je to tím, že znaky naší abecedy jsou ze vstupního souboru v UTF-8 převedeny do interního Unicode (jiná možnost neexistuje), přitom tyto znaky v Unicode jsou mimo rozsah 0–255. Na druhé straně Němci, Francouzi nebo Španělé nemají problém, jejich abeceda je v Unicode tabulce celá uvnitř rozsahu 0–255, takže jim stačí natáhnout klasický \TeX ový font v kódování T1. Jinak řečeno, na rozdíl od němčiny, pro češtinu v \XTeX u nutně potřebujeme OpenType font. Z toho taky plyne, že příkaz:

```
xetex -fmt pdfcsplain
```

sice spustí \XTeX s \mathcal{S} plainem, ale v něm jsou zavedeny jen klasické 8bitové fonty, takže čeština bez dalšího zavedení fontů nemůže fungovat. Je potřeba na začátek dokumentu minimálně napsat:

```
\input ucode % inicializace maker pro Unicode
\input lmfonts % nebo \input cs-termes atd., zavedení fontu v Unicode
\chlyph % zavedení vzorů dělení, nyní dle Unicode
```

¹⁾ <http://www.microsoft.com/typography/otspec/featurelist.htm>

²⁾ <http://www.abclinuxu.cz/clanky/tex-7-opentype-fonty>

Bohužel není možné načíst OTF fonty do formátu a pak je přímo použít. Do formátu umí X_YTeX načíst jen klasické 8bitové T_EXové fonty. V makru `xeplain.ini` (které je součástí balíčku `CSplain`) je učiněn pokus tuto nevýhodu obejít pomocí příkazu `\everyjob`, jehož parametr se spustí na začátku zpracování při každém spuštění T_EXu. Je to ale třeba považovat za velmi nouzové řešení. Je lepší seznámit uživatele s tím, že si musí v dokumentu pro X_YTeX zavést fonty explicitně.

X_YTeX neobsahuje rozšíření pdfT_EXu, protože interně vystupuje do modifikovaného DVI, které je ve stejném běhu okamžitě zpracováno postprocesorem `xdvipdfmx` do výstupního PDF. Toto modifikované DVI uživatel na disku neuvidí, protože postprocesor přebírá data X_YTeXu přímo v paměti počítače¹). Veškerou manipulaci s PDF formátem (načítání obrázků, barvy, transformace) je tedy třeba řešit pomocí vzkazů pro postprocesor. T_EX je vybaven příkazem `\special{<text>}`, který vloží do sazby neviditelnou značku se vzkazem `<text>` pro DVI postprocesor. Sadu vzkazů, kterým případný postprocesor rozumí, můžeme považovat za další jazyk na řízení sazby. Manuál pro `xdvipdfmx` najdete v `dvipdfm.pdf`²). V tomto manuálu je možné se dočíst, jak přepínat barvy, vkládat obrázky, nastavovat hyperlinky, záložky atd. Makro `OPmac` řeší tuto disproporci mezi pdfT_EXem a X_YTeXem tak, že veškerou manipulaci s PDF dělá pomocí primitivních příkazů pdfT_EXu a pokud zjistí, že místo pdfT_EXu je použit X_YTeX, načte pomocný soubor `opmac-xetex.tex`. Tam jsou chybějící pdfT_EXové primitivní příkazy dodefinovány pomocí příkazů `\special` pro `xdvipdfmx` a dále je tam dorešen X_YTeX-specifický způsob vkládání obrázků pomocí příkazů `\XeTeXpdffile` nebo `\XeTeXpicfile`. Uživatel `OPmac` se nemusí o nic starat, ten používá stále stejná makra na řízení grafiky a vkládání obrázků.

Další specialitou X_YTeXu je možnost automatického vkládání tokenů mezi dvojice tokenů určitých tříd. Toto samočinné vkládání pracuje na úrovni vkládání tokenů do sazby, ne na úrovni expandování maker. Nejprve je třeba každému znaku, který má být počten možností vytvořit dvojici se samočinným vkládáním, přiřadit číslo, tzv. třídu znaku. Tato čísla se alokují makrem `\newXeTeXintercharclass`. Pak je třeba pomocí příkazů

```
\XeTeXcharclass<kód znaku> = <třída>
\XeTeXinterchartoks<třída><třída> = {\<tokeny>}
```

přidělit znakům třídy a oznámit, co se bude vkládat mezi dvojice znaků jakých tříd. Konečně je potřeba pomocí `\XeTeXinterchartokenstate=1` aktivovat vkládání tokenů. Příklad:

```
\newXeTeXintercharclass \mycharclassbf
\XeTeXcharclass 'a =\mycharclassbf
\XeTeXcharclass 'e =\mycharclassbf
\XeTeXcharclass 'i =\mycharclassbf
\XeTeXcharclass 'o =\mycharclassbf

\XeTeXinterchartoks 0 \mycharclassbf = {\bgroup\bf}
\XeTeXinterchartoks \mycharclassbf 0 = {\egroup}
\XeTeXinterchartoks 255 \mycharclassbf = {\bgroup\bf}
\XeTeXinterchartoks \mycharclassbf 255 = {\egroup}

\XeTeXinterchartokenstate = 1

Nektere samohlasky jsou nyní tue.
\bye
```

¹) Unixoví uživatelé tomu říkají roura (pipe).

²) Ta písmena x v názvu postprocesoru naznačují, že se jedná o verzi původního programu rozšířenou směrem k manipulaci s OTF fonty a s interním Unicode.

Příklad alokuje jednu třídu vybraných samohlásek a předpokládá, že tyto budou vloženy do textu s ostatními znaky, které mají implicitně třídu 0. Nebo se vyskytnou na začátku či konci slova, kde je uvažován virtuální hraniční znak vestavěné třídy 255.

B.3 LuaTeX

LuaTeX propojuje vlastnosti TeXu s procedurálním jazykem Lua¹). Autoři LuaTeXu dospěli zřejmě k názoru, že makrojazyk TeXu je místy obtížně použitelný a že je rozumné TeX rozšířit o jazyk, v němž by se daly dělat věci lépe. Jazyk Lua je v LuaTeXu prorostlý do původních algoritmů TeXu a je možné tyto algoritmy na různých úrovních zpracování modifikovat nebo dokonce přeprogramovat pomocí Lua kódů. LuaTeX bez použití Lua kódu se chová jako TeX.

Výhodou LuaTeXu je skutečnost, že nabízí mocný nástroj na zpracování sazby odvozený z TeXu a navíc se v něm dá programovat ve srozumitelném procedurálním jazyku.

Nevýhodou LuaTeXu je fakt, že jeho dokumentace je mnohastránková a aby ji člověk mohl začít číst, musí stejně nejprve dokonale zvládnout a pochopit vnitřní algoritmy klasického TeXu. Další nevýhodou je fakt, že se zde míchají jazyky na různých úrovních zpracování. Jak nakonec celý systém funguje, je pro běžného uživatele mnohonásobně hůře uchopitelné.

Hans Hagen zveřejnil v roce 1996 svou první verzi balíku maker nad TeXem zvanou ConTeXt MKII, která pracovala s pdfTeXem. Další jeho verze ConTeXt MKIV²) z roku 2007 je již zcela vystavěná na využití LuaTeXu. Nyní je vývoj ConTeXtu natolik vzájemně ovlivněn vývojem LuaTeXu a naopak, že je možné oba projekty považovat ze projekt jediný. Probíhá nepřetržitý vývoj maker ConTeXtu, Lua kódů i LuaTeXu samotného.

LuaTeX je při svém startu v INITEX módu vybaven jen primitivními příkazy TeXu a příkazem `\directlua{<text>}`, kde `<text>` je kód v jazyce Lua a příkaz je zpracován na úrovni expanze maker. Během expanze se interpretuje `<text>` v jazyce Lua.

LuaTeX disponuje rozšířeními pdfTeX, eTeX, Unicode a přidává svá další rozšíření. K aktivaci těchto rozšíření je třeba v Lua kódu spustit k tomu speciálně určenou funkci `tex.enableprimitives(<prefix>, <seznam>)` se seznamem všech dalších příkazů, které chceme mít k dispozici. Tento seznam je výstupem funkce `tex.extraprimitives()` a je velmi rozsáhlý. Uvedená aktivace rozšiřujících příkazů je typicky provedena při generování formátů v souboru `luatexiniconfig.tex`. Vypadá to tam zhruba takto:

```
\directlua{ tex.enableprimitives('', tex.extraprimitives()) }
```

Jak bylo řečeno, jazyk Lua v LuaTeXu disponuje speciálními funkcemi a datovými strukturami, které přímo spolupracují s interními algoritmy a interními registry TeXu. V tomto krátkém textu není možno ani ukázat základní vlastnosti jazyka Lua ani uvést seznam všech rozšiřujících primitivních příkazů, kterých je kolem pěti set a nové v rámci vývoje vznikají. Následuje tedy jen jednoduchý příklad, který ukazuje propojenost Lua kódu s makrojazykem při výpočtu průměru hodnot. Makro `\average{<číslo>}` přečte čísla oddělená čárkou a expanduje na jejich aritmetický průměr.

```
\def\average#1{\directlua{ t = {} }\averageA #1,,}
\def\averageA#1,{\ifx,#1,\averageB
    \else \directlua{ table.insert(t, #1) }%
    \expandafter\averageA\fi
}
```

¹) <http://www.lua.org/>

²) <http://wiki.contextgarden.net/>

```

\def\averageB{\directlua{
    sum = 0;
    for i,v in ipairs(t) do sum = sum + v end;
    tex.print(sum/table.getn(t))
}}
% Test:
\message{::: \average{2,3,4,5,6,7,11}}
\end

```

Makro `\average` inicializuje tabulku `t` jako prázdnou a spustí `\averageA` *<číslo>*,,. Dále makro `\averageA` čte postupně jednotlivá čísla oddělená čárkou a ukládá je do tabulky `t` pomocí `table.insert()`. Také na konci volá samo sebe, takže v cyklu přečte všechna čísla. Jakmile dospěje ke koncové dvojčárce `,,` přečte prázdný parametr, který pozná pomocí testu `\ifx,#1,`. V takovém případě zavolá závěrečný Lua kód napsaný v makru `\averageB`. Tento kód pomocí cyklu `for` projde naplněnou tabulku, spočítá součet jejich hodnot `sum` a do vstupní fronty \TeX u vrátí pomocí funkce `tex.print()` tento součet dělený délkou tabulky. Příklad si můžete vyzkoušet pomocí `luatex pokus.tex`.

V závěru této krátké zmínky o Lua \TeX u se zaměříme na možnosti zavedení OTF fontů v Lua \TeX u. Poznamenejme nejprve, že pro Lua \TeX rovněž platí důležitá poznámka ze strany 118. Primitivní příkaz `\font` pracuje implicitně normálně, jako v klasickém \TeX u. Je ovšem možné zavolat soubor `luaotfload.sty`, který pomocí `\directlua` přeprogramuje primitivní příkaz `\font` tak, aby byl schopen číst OTF fonty. Tutéž práci jako `luaotfload.sty` dělá i soubor `maker luafonts.tex` z \CSplain u. Protože je tento soubor z \CSplain u daleko přehlednější (neodkazuje na desítky dalších souborů `maker`), doporučujeme používat `\input luafonts`.

Po `\input luafonts` je primitivní příkaz `\font` vybaven stejnými schopnostmi, jako v $X_{\mathcal{T}}\TeX$ u, takže funguje třeba ukázka zavedení fontu `Linux Libertine` ze strany 117. Syntaktické možnosti při zavádění OTF fontů po `\input luafonts` zahrnují nejen stejné syntaktické možnosti jako v $X_{\mathcal{T}}\TeX$ u, ale přidávají některé možnosti navíc. Doporučujeme tyto rozšiřující možnosti nepoužívat, protože člověk nikdy neví, kdy bude chtít svůj dokument místo v Lua \TeX u použít třeba v $X_{\mathcal{T}}\TeX$ u.

\TeX ové ligatury `--`, `---` atd. v OTF fontech se v $X_{\mathcal{T}}\TeX$ u aktivují pomocí atributu `mapping=tex-text`, zatímco v Lua \TeX u pomocí `+tlig`. Protože fontový zavaděč tiše ignoruje atributy, kterým nerozumí nebo které nejsou součástí fontu, je možné psát oba atributy současně, aby zavedení fontu fungovalo stejně v $X_{\mathcal{T}}\TeX$ u i Lua \TeX u.

Lua \TeX se na rozdíl od $X_{\mathcal{T}}\TeX$ u neopírá o knihovnu pro práci s fonty v operačním systému, ale řeší interpretaci všech vlastností OTF fontů ve své vlastní režii pomocí Lua kódů. Rovněž spolupracuje se speciálními skripty, které se rozhlíží po operačním systému a ukládají si do interních tabulek poznámky o těchto fontech a umožňují pak primitivu `\font` z Lua \TeX u nalézt OTF font nejen v \TeX ové distribuci, ale i v operačním systému. Další možnosti týkající se fontů v Lua \TeX u jsou popsány na webové stránce `Con \TeX tu1)`.

¹⁾ http://wiki.contextgarden.net/Fonts_in_LuaTeX

Dodatek C

Numerické a metrické údaje

V parametrech příkazů a jako hodnoty registrů se často vyskytují numerické a metrické údaje. Například příkaz `\chardef\⟨něco⟩=⟨hodnota⟩` deklaruje novou znakovou konstantu `\⟨něco⟩` a přiřadí ji `⟨hodnotu⟩`. Nebo `\parindent=⟨hodnota⟩` vloží do registru `\parindent` `⟨hodnotu⟩`. V prvním příkladě je `⟨hodnota⟩` celočíselný numerický údaj a ve druhém příkladě metrický údaj. \TeX nabízí několik variantních možností, jak zapsat `⟨hodnotu⟩`.

C.1 Numerický údaj

Nejběžnější způsob zápisu je prostě číslo zapsané svými ciframi v desítkové soustavě s případným znaménkem mínus před takovým zápisem (pro záporná čísla). Tedy: 191, -4 atd. Kromě toho je možné zapsat numerický údaj hexadecimálně tak, že musí předcházet znak " a cifry A-F se píší velkými písmeny. Třeba "3B. Další možností je extrahovat `⟨hodnotu⟩` jako kód zapsaného znaku. To se provede prefixem `‘`, který předchází danému znaku.¹⁾ Například `‘A` je rovno 65, protože písmeno A má v ASCII (to \TeX interně používá) kód 65. Před znaky, které mají v \TeX u speciální význam, je nutné přidat zpětné lomítko. Tím vznikne řídicí sekvence `\⟨znak⟩`, která v kontextu čtení `⟨hodnoty⟩` ve tvaru `\⟨znak⟩` nevyvolá žádnou speciální aktivitu, pouze se promění v kód `⟨znaku⟩`. Třeba `%` zahajuje až do konce řádku komentář, který je \TeX em ignorován. Ovšem `‘%` je hodnota 37. Konečně v místě `⟨hodnoty⟩` může být napsaná dříve deklarovaná znaková konstanta nebo numerický registr, v obou případech může předcházet znak mínus. Podrobný popis syntaxe numerického údaje najdete v TBN na str. 325 u hesla `⟨number⟩`. Příklady:

```
\pageno = 13           % registr \pageno (číslo strany) je nastaven na 13
\chardef\% = 37        % znaková konstanta \% je deklarována jako 37
\chardef\% = "25       % znaková konstanta \% je deklarována jako 37
\chardef\% = ‘\%       % znaková konstanta \% je ASCII kód znaku %, tedy 37
\chardef\a = \pageno   % znaková konstanta \a má hodnotu aktuální strany
\clubpenalty = 5000    % registr \clubpenalty (trest za vytvoření sirotka)
                        % je nastaven na 5000
```

Za numerickým údajem je možné vložit nepovinnou mezeru, která se nevytiskne, ale odděluje údaj od dalších informací.

Aby toho nebylo málo, je v \TeX u možné při zápisu jakéhokoli přiřazení vynechat `symbol = i` mezery kolem. Takže nelze vyloučit, že se někdy setkáte i s takovými zápisy: `\pageno13`, `\chardef\%‘\%`. Pokud k tomu ale není pádný důvod, je vhodné v makrech rovnítko pro zvýšení přehlednosti používat.

C.2 Metrický údaj

Takový údaj vyjadřuje rozměr a je obvykle zadán desetinným číslem následovaným jednotkou. Je-li číslo celé, desetinnou tečku není nutno psát. Je-li číslo v absolutní hodnotě menší než 1, nulu před desetinnou tečku není nutno psát. Jednotky jsou zapsány dvěma malými písmeny. V \TeX u jsou rezervovány následující metrické jednotky:

¹⁾ Prefixový znak `‘` je *zpětný apostrof*, na klávesnici vlevo nahoře.

mm	milimetr	$1\text{ mm} = 1/25,4\text{ in} \doteq 2,84528\text{ pt} \doteq 2,6591\text{ dd}$
cm	centimetr	$1\text{ cm} = 10\text{ mm} \doteq 28,4528\text{ pt}$
in	palec (inch, coul)	$1\text{ in} = 72,27\text{ pt} = 25,4\text{ mm}$
pt	monotypový bod	$1\text{ pt} = 1/72,27\text{ in} \doteq 0,35146\text{ mm} \doteq 0,93457\text{ dd}$
pc	pica	$1\text{ pc} = 12\text{ pt} \doteq 4,21752\text{ mm}$
bp	počítačový bod	$1\text{ bp} = 1/72\text{ in} \doteq 0,35278\text{ mm}$
dd	Didotův bod	$1\text{ dd} = 1238/1157\text{ pt} \doteq 1,07\text{ pt} \doteq 0,376\text{ mm}$
cc	cicero	$1\text{ cc} = 12\text{ dd} \doteq 4,512\text{ mm}$
sp	přesnost \TeX u	$1\text{ sp} = 1/65536\text{ pt} = 2^{-16}\text{ pt} \doteq 5,36 \cdot 10^{-9}\text{ m}$
em	velikost písma	typicky šířka velkého M aktuálního fontu
ex	střední výška písma	typicky výška malého x aktuálního fontu

Příklady:

```
\parindent=15pt      % odstavcová zarážka je nastavena na 15 pt
\vskip 2.5cm         % příkaz \vskip vloží vertikální mezeru 2,5 cm
\baselineskip=13pt   % registr \baselineskip (řádkování) je nastaven na 13 pt
\hskip 1.5em         % příkaz \hskip vloží horizontální mezeru 1,5 em
```

První a třetí řádek ukázky obsahuje přiřazení $\langle hodnoty \rangle$ do registru, zatímco druhý a čtvrtý řádek ilustruje použití příkazu s parametrem, kterým je metrický údaj. Za příkazem musí následovat parametr přímo bez rovnítka, zatímco za registrem je možné použít (nepovinné) rovnítko označující přiřazení.

Před i za jednotkou můžete psát nepovinnou mezeru, která se netiskne. Místo jednotky se v zápise metrického údaje může vyskytnout nějaký registr nesoucí metrický údaj. \TeX pak provede násobení desetinného čísla s registrem. Takže je možné psát třeba:

```
\vskip .5\baselineskip % příkaz \vskip vloží mezeru velikosti půlky řádku
\hskip \parindent      % příkaz vloží mezeru velikosti jednoho \parindent
\baselineskip=1.2\baselineskip % nové řádkování bude 1,2 krát větší
```

● **Poznámka** ● Příkazy `\hskip`, `\vskip` a registr `\baselineskip` jsou schopny pracovat s komplikovanějším parametrem, než je jen metrický údaj. Dokáží připojit údaj o roztažitelnosti a stlačitelnosti mezery. Viz sekci 9.1.

Přesná specifikace zápisu metrického údaje je v TBN na str. 319 u hesla $\langle dimen \rangle$. Pro formální popis údajů s roztažitelností a stlačitelností hledejte v TBN heslo $\langle glue \rangle$.

Dodatek **D**

Dvouzobáková konvence

Objeví-li se na vstupu čtveřice znaků `^^xy`, kde `xy` jsou cifry hexadecimálního zápisu kódu (cifry `a–f` je nutné tentokrát psát s malými písmeny), `TeX` promění tuto čtveřici okamžitě po přečtení v jediný znak s daným kódem. Takže třeba `^^41` se promění ve znak `A`. Teprve po této proměně se znaku přiřadí kategorie a podle kategorie se znak chová jako normální nebo speciální. Aby tato proměna fungovala, musí mít znak zobáku `^` nastavenou kategorii 7, což je obvykle splněno.

Důvod této konvence je následující. Ne všechny znaky v rozsahu kódů 0–255 jsme schopni jednoduše napsat na klávesnici. Dvouzobáková konvence nám dává možnost toto obejít a vložit hexadecimální kód vstupního znaku.

Kromě hexadecimálního kódu je možné ještě používat `Ctrl`- znaky: `Ctrl-A` má kód 01, `Ctrl-B` kód 02 atd. K tomu stačí psát `^^A`, `^^B` až `^^Z`. Je to totéž, jako kdybychom napsali `^^01`, `^^02`, až `^^1a`.

`TeX` nechce rozbořit terminál při výstupu svých zpráv do `.log` souboru a na terminál. Proto při těchto výpisech používá pro potenciálně nebezpečné znaky, které nemají grafickou podobu v ASCII (kódy 0–31 a 127–255), výše uvedenou dvouzobákovou konvenci. Formát `CSplain` ruší tuto konverzi znaků pro kódy 128–255, takže akcentované znaky české a slovenské abecedy vidíme v lokalizovaném terminálu přímo a správně.

V `XYTeXu` a `LuaTeXu` funguje navíc možnost zápisu `^^^xyuv`, kde `xyuv` jsou cifry hexadecimálního zápisu podle Unicode. Takže například `^^^010d` se promění ve znak `č`, protože tento znak má v tabulce Unicode hodnotu `U+010D`. Je-li hexadecimálních cifer více, je třeba použít odpovídající počet zobáků. Znak s maximálním kódem tabulky Unicode tedy zapíšeme jako `^^^^^^10ffff`.

Dodatek E

Vstupní kódování, encTeX, UTF-8

Již klasický TeX je vybaven překódovacími vektory `xord` a `xchr`. První jmenovaný překódovává byte na byte při vstupu a druhý by měl být nastaven jako inverzní a překódovává při zápisu na terminál nebo do textových souborů během příkazů `\write` a `\message`. Tuto věc zařadil autor do TeXu v rámci svého rozhodnutí, že uvnitř se bude TeX chovat na všech počítačových systémech jednotně podle ASCII, ale systémy existovaly tehdy ve dvou rozšířených kódováních: ASCII a EBCDIC. Později, v 90 letech minulého století, byla čeština také používána na různých operačních systémech v různých 8bitových kódováních (ISO-8859-2, PC-Latin2, CP1250, Kameničtí, KOI8-cs atd.). V té době bylo rozhodnuto, že C_Splain bude uvnitř pracovat s češtinou a slovenštinou podle ISO-8859-2 (kódování C_Sfontů), zatímco jeho implementace v konkrétním operačním systému pracujícím třeba v jiném 8bitovém kódování bude prováděna správným nastavením `xord` a `xchr` vektorů. Toto nastavení bylo možné upravit během kompilace TeXu nebo ve vylepšených verzích pomocí speciálních `tcx` tabulek. Nebo také pomocí encTeXu.

E.1 EncTeX

EncTeX je rozšíření pdfTeXu, které se probouzí k životu přepínačem `-enc` v INITEX módu při generování formátu a které se stará o překódování při čtení vstupního souboru a o zpětné překódování při zápisu na terminál a do textového souboru pomocí příkazů `\write` a `\message`. EncTeX překóduje vstup dříve, než se v TeXu uplatní jakékoli další algoritmy zpracování vstupu, tj. před přidělením kategorií znakům, před sestavením řídicích sekvencí a před expandováním maker.

EncTeX umí nastavit jednotlivé hodnoty `xord` vektoru pomocí `\xordcode`, dále hodnoty `xchr` vektoru nastavuje pomocí `\xchrcline` a konečně příkazem `\xprncode` nastavuje tisknutelnost znaků, tj. zda znaky vystupující do terminálu a do souborů pomocí `\message` a `\write` mají nebo nemají být převedeny na dvouzobákovou notaci. Nastavením na kladnou hodnotu, `\xprncode'\langle znak\rangle=1`, bude znak tištěn do souborů a na terminál v nezměněné podobě, bez převádění na dvouzobákovou notaci. Klasický TeX implicitně převádí na dvouzobákovou notaci vše s výjimkou ASCII viditelných znaků z rozsahu kódů 32 až 126.

Od verze encTeXu z roku 2003 je možno nejen nastavovat uvedené byte-to-byte překódovací vektory, ale je možné pomocí příkazů `\mubyte` a `\endmubyte` zajistit překódování více vstupních bytů na jeden byte nebo na řídicí sekvenci. Tento byte nebo tato řídicí sekvence se při `\message` a `\write` převádějí zpět na původních více bytů. Protože kódování UTF-8 je vícebytové, umožňuje tato verze encTeXu překódovávat vstup napsaný v UTF-8. Přitom počet UTF-8 kódů, které dovede zpracovat, není omezen jen počtem různých interních bytů v pdfTeXu (těch je jen 256). Je totiž možné mapovat UTF-8 kódy na řídicí sekvence, kterých může být libovolně mnoho.

Překódovací informace se do encTeXové tabulky zanesou pomocí

```
\mubyte<byte> <více bytů>\endmubyte  
nebo  
\mubyte<řídicí sekvence> <více bytů>\endmubyte
```

Aby encTeX tuto tabulku na vstupu použil, je třeba dále nastavit `\mubytein` na kladnou hodnotu, tedy typicky `\mubytein=1`. Aby encTeX kodoval zpětně takto označené `<byte>` a `<řídicí sekvence>` na `<více bytů>` při zápisu do souborů pomocí `\write`, je třeba

nastavit `\mubyteout` větší nebo rovno třem. Aby dělal totéž i při zápisu na terminál a do `.log` souborů pomocí `\message`, je třeba nastavit na kladnou hodnotu `\mubytelog`. Více informací o `encTeXu` je uvedeno v dokumentaci [14].

E.2 EncTeX v Csplainu

Od verze Csplainu Dec. 2012 je jeho implicitní vstupní kódování nastaveno na UTF-8. Používá-li Csplain `pdfTeX`, je toto kódování zpracováno `encTeXem`. To znamená, že formát je vygenerován s přepínačem `-enc` a při generování formátu je přečten soubor `csenc-u.tex`, ve kterém je nastavena překódovací tabulka pro české a slovenské znaky i pro další obvyklé řídicí sekvence a konečně jsou nastaveny registry `\mubytein`, `\mubyteout` a `\mubytelog` na příslušné kladné hodnoty.

Rovněž je během generování formátu přečten soubor `utf8unkn.tex`, ve kterém je zajištěno, že pokud `encTeX` narazí na neznámý UTF-8 kód, převede ho na řídicí sekvenci `\warntwobytes` nebo `\warnthreebytes`. Tyto sekvence jsou makra, která vypíší varování na terminál a do `.log` souboru ve tvaru (například):

```
WARNING: unknown UTF-8 code: '€ = ^^e2^^82^^ac' (line: 42)
```

a do sazby umístí černý čtvereček. Je možné v Csplainu dodefinovat chybějící UTF-8 kódy například takto:

```
\mubyte\eurochar ^^e2^^82^^ac\endmubyte % kód znaku € mapován na \eurochar
\def\eurochar{{\eurofont e}}           % definice \eurochar
\font\eurofont=feymr10 \regfont\eurofont % použitý font
```

Jak vidíme v ukázce, je vhodné údaj *⟨více-bytů⟩* mezi `\mubyte` a `\endmubyte` rozepsat pomocí zobákové konvence, kterou navíc vidíme přímo ve vypsaném varování. Není-li pro daný znak k dispozici potřebný font, je možné makrem řešit sestavení znaku z komponent nebo provést jakoukoli jinou aktivitu.

Pokud `encTeX` zjistí zásadní chybu v UTF-8 kódování, vypíše prostřednictvím makra `\badutfinput` chybové hlášení:

```
UTF-8 INPUT IS CORRUPTED ! Maybe you are using another input encoding.
```

Jak je zde řečeno: `encTeX` se domnívá, že je použito nějaké jednobytové kódování. Je-li to pravda a je-li toto kódování shodné s vnitřním kódováním v `TeXu` (tj. ISO-8859-2), je možné na začátek dokumentu napsat `\input utf8off`. Toto makro deaktivuje `encTeX`, takže žádné překódování ani kontrola kódování se nekoná. Je možné také místo toho zavolat trikové makro `\input mixcodes`, které ponechá `encTeX` aktivní, ale dokáže akceptovat a správně interpretovat na vstupu nejen UTF-8 kódy, ale také znaky české abecedy z kódování ISO-8859-2 nebo CP1250. Veškerý mix těchto kódů zpracuje správně a pomocí `\write` zapisuje do souborů v UTF-8 kódování.

Csplain po vygenerování formátu (bez zavedení dalších maker) je schopen správně interpretovat pouze UTF-8 kódy české a slovenské abecedy a kódy znaků ze seznamu na straně 15 (`\ss` až `\promile`). Pokud jsou zavedeny pomocí fontových souborů jiné fonty (tj. `\input ctimes`, `\input cs-termes` atd.), jsou navíc k dispozici znaky uvedené na straně 133 dole (`\currency` až `\frq`) a jejich UTF-8 kódy jsou také správně interpretovány. To mimo jiné znamená, že problém s chybějícím znakem Euro, který byl součástí předchozí ukázky, není potřeba řešit po zavedení těchto fontů, protože znak Euro je v těchto fontech obsažen.

Balíček Csplain nabízí k použití soubory `utf8lat1.tex` a `utf8lata.tex`, jejichž zavedení způsobí rozšíření správně interpretovaných UTF-8 kódů na odpovídající úseky tabulky Unicode:

```
\input utf8lat1 % ... Latin-1 Supplement U+0080--U+00FF
\input utf8lata % ... Latin Extended-A U+0100--U+017F
```

Je možné najít v těchto souborech inspiraci a případně vytvořit další takové podobné soubory.

CSplain nabízí soubor maker `exchars.tex`, která propojí běžně používané fonty s jejich alternativami v kódování TS1 (viz tabulku F.3.3 na straně 134). Použití tohoto souboru maker je vysvětleno v závěru dodatku F.3 na straně 135. Soubor maker `exchars.tex` sice přidává možnost využití mnoha dalších znaků, ovšem nepřidává těmto znakům mapování na UTF-8 kódy. Pokud chcete takové znaky použít „nativně“, je třeba nejen zavést `exchars.tex`, ale také mapovat UTF-8 kódy, například:

```
\mubyte \exnumero ^^e2^^84^^96\endmubyte
\mubyte \exonehalf ^^c2^^bd\endmubyte
\mubyte \exflorin ^^c6^^92\endmubyte
...
```

V CSplainu je připraven soubor `cyrchars.tex`, který přidává azbuku a funguje podobně jako `exchars.tex`, tedy azbuka je závislá na aktuální verzi a velikosti základního písma. Na rozdíl od `exchars.tex` ale soubor pro azbuku navíc mapuje potřebné UTF-8 kódy, takže není nutno nic dalšího mapovat, není nutno přepínat fonty a stačí psát:

```
\input cyrchars
Tady je normální text.
A zde: это наше дело, \it kurzívou: это наше дело, \bf tučně: это наше дело.
\bye
```

A dostaneme: Tady je normální text. A zde: это наше дело, *kurzívou: это наше дело*, **tučně: это наше дело**.

E.3 Vstupní kódování v XeTeXu a LuaTeXu

XeTeX ani LuaTeX nedisponují rozšířením `encTeX`. Oba jsou implicitně nastaveny tak, že čtou vstup v kódování UTF-8, který převádějí do vnitřního kódu podle Unicode a zpětně do textových souborů zapisují podle UTF-8. XeTeXu je možné říci, že má interpretovat jiné vstupní kódování pomocí příkazu `\XeTeXinputencoding`. Ovšem interní kódování je jediné Unicode. LuaTeX může mít teoreticky naprogramován vlastní input procesor pomocí Lua kódu.

Vzhledem k tomu, že oba zmíněné TeXové programy předpokládají vnitřní kódování podle Unicode, je nezbytné pro češtinu nebo slovenštinu na to navázat odpovídajícím fontem v kódování Unicode ve formátu OTF. Klasické 8bitové TeXovské fonty je sice možné také načíst, ale pro češtinu nebo slovenštinu v Unicode jsou nepoužitelné. Viz též důležitou poznámku na straně 118.

Dodatek F

Fonty v T_EXové distribuci

T_EX (s výjimkou variant X_ƎT_EX a LuaT_EX) nespolupracuje s fonty instalovanými v operačním systému. Používá jen fonty, které jsou speciálním způsobem instalovány přímo v T_EXové distribuci. V první části se podíváme, jak se v těchto fontech dá aspoň trochu orientovat, a v druhé části ukážeme, jak přidat do T_EXové distribuce nový font.

F.1 Základní přehled T_EXových fontů

Problém T_EXových fontů je často v tom, že zde existují už desítky let a v době jejich vzniku se velmi dbalo na to, aby jejich názvy obsahovaly maximálně 8 písmen (plus tři písmena přípony). Protože jediné tak mohly být tyto fonty použitelné v libovolném tehdy provozovaném operačním systému. Z toho důvodu autoři vymýšleli pro své fonty nejružnější obskurní zkratky.

Knuth rozhodl v případě rodiny fontů Computer Modern, že první dvě písmena v názvu budou **cm**, pak následuje zkratka varianty (maximálně 4 písmena) a ke konci je připojen údaj o designované velikosti (jedno- nebo dvouciferné číslo). Č_Sfonty kopírují přesně názvy odpovídajících Computer Modern fontů, jen první dvě písmena jsou **cs** místo **cm**. Zkratky variant jsou ve fontech Computer Modern následující:

r	... Roman (neboli normal)	17 12 10 9 8 7 6 5
b	... Bold (tučný)	10
bx	... Bold extended (tučný, širší)	12 10 9 8 7 6 5
ti	... Text italics (kurzíva)	12 10 9 8 7
bxti	... Bold extended text italics (tučná kurzíva)	10
tt	... Typewriter (strojopis)	12 10 9 8
itt	... Italics typewriter (strojopis v kurzívě)	10
sl	... Slanted (skloněný, nepravá kurzíva)	12 10 9 8
sltt	... Slanted typewriter	10
ss	... Sans serif (bez serifů, tj. grotesk)	17 12 10 9 8
ssi	... Sans serif italic (skloněný)	17 12 10 9 8
ssdc	... Sans serif demi condensed (polotučný)	10
ssbx	... Sans serif Bold extended	10
csc	... Caps and Small Caps (malé kapitálky)	10
tcsc	... Typewriter caps and small caps	10
u	... Upright (napřímená kurzíva)	10
mi	... Math italics (matematická kurzíva)	12 10 9 8 7 6 5
mib	... Math italics bold (tučná mat. kurzíva)	10
sy	... Math symbols (matematické symboly)	10 9 8 7 6 5
bsy	... Bold math symbols (tučné mat. symboly)	10
ex	... Extended math symbols (velké mat. symboly)	10

V pravém sloupci je seznam připravených designovaných velikostí pro danou variantu. *Designovaná velikost* fontu je velikost, pro kterou je font navržen. Tvar znaků v jednotlivých designovaných velikostech Tato není odvozen jen pomocí geometrického zvětšování/zmenšování z velikostí jiných. Vyberáte-li mezi designovanými velikostmi, pak vybíráte výchozí *přirozenou velikost* písma. Naopak pomocí klíčových slov **at** a **scaled** zvětšujete font jen geometricky (lineárně). Srovnajte:

<code>\font\fa=cmr5</code>	<code>{\fa tady je text}</code>	tady je text
<code>\font\fb=cmr10</code>	<code>{\fb tady je text}</code>	tady je text
<code>\font\fc=cmr5 at10pt</code>	<code>{\fc tady je text}</code>	tady je text

Rozličné designované velikosti Knuth připravil zejména pro fonty častěji používané, zatímco méně často používané varianty mají designovánu jen základní velikost 10 bodů a do jiné velikosti je třeba je zvětšit/zmenšit lineárně. \TeX ové makro by mělo znát seznam designovaných velikostí pro každou variantu a jakmile si uživatel přeje nějakou velikost, makro by mělo rozhodnout o nejbližší designované velikosti a tu použít. Například:

```
\font\font = csr17 at 16pt % uživatel chce 16bodový CM Roman
\font\font = csr8 at 8.3pt % uživatel chce 8,3bodový CM Roman
```

Touto vlastností je obdařen soubor maker `OPmac` po načtení `ams-math.tex` (viz sekce 6.2 a 7.1) a dále v \LaTeX u se o to stará makrobalík `NFSS`.

Fonty Computer Modern mají své znaky umístěny jen v části tabulky se sloty 0 až 127. \TeX sice umí od roku 1989 pracovat s 8bitovými fonty, ale Knuth se rozhodl v každém svém fontu nechat zaplněnu jen polovinu slotů. Takže zbylé sloty 128–255 zůstaly volné. Tuto druhou část tabulky využívají například \mathcal{C} Sfonty, které tam přidávají znaky české a slovenské abecedy podle kódování ISO-8859-2. Viz též tabulku F.3.1 v sekci F.3.

Kódování \mathcal{C} Sfontů je poněkud svérázné. Na viditelných ASCII slotech se poměrně shoduje s ASCII (až na výjimky popsané v sekci 3.7). Na dalších slotech s pozicí menší než 128 jsou vloženy ligatury nebo matematické symboly stejně jako v Computer Modern fontech. Druhá část tabulky je obsazena řídce jen znaky české a slovenské abecedy.

\TeX oví uživatelé se na konferenci v Corku v roce 1992 dohodli, že vytvoří kódování \TeX ových fontů, které plně obsadí tehdy možných 256 slotů, přitom bude v první části tabulky kopírovat přesně ASCII (tj. zruší Knuthovy výjimky popsané v sekci 3.7). Mimo viditelné ASCII znaky vloží jen jednotlivé akcenty a ligatury (tj. opustí matematické znaky, které budou přemístěny jen do matematických fontů) a do slotů 128–255 umístí akcentované znaky jazyků západní Evropy podle ISO-8859-1. Konečně do volných míst vměstnají znaky dalších jazyků evropských národů písícičkou latinkou, tedy i češtiny a slovenštiny. Tak vzniklo kódování, které se poněkud nešťastně jmenuje T1 a říká se mu také „kódování DC fontů“ nebo „EC fontů“ nebo „kódování podle Corku“. Viz též tabulku F.3.2 v sekci F.3. V kódování T1 byly vygenerovány stovky fontů pro \TeX . Typicky mají ve zkratce svého názvu dvojici znaků `8t`.

● **Cvičení 18** ● Vyhledejte na disku všechny soubory s názvem `*8t*.tfm`. Namátkou se do některého fontu podívejte pomocí `pdftex testfont`.

Ačkoli to tedy vypadá, že v \TeX ové distribuci jsou kromě Computer Modern stovky dalších fontů, není možné jásat. Jejich užitečnost je často sporná. Nejstarší fonty mají jen METAFONTové zdroje¹⁾, což dnes není příliš použitelné. Mírně mladší fonty jsou sice i ve formátu Type1, ale pro kódování T1 jsou v distribuci instalovány vesměs pomocí nástroje `fontinst`, který využívá schopnost \TeX u pracovat s tzv. *virtuálními fonty*. To je sice silný nástroj, ale v uvedeném případě je použit na doplnění chybějících akcentovaných znaků v jednotlivých fontech pomocí kompozitů, což má neblahé následky: ve výsledném PDF s takovým fontem nelze vyhledávat český text ani kopírovat text z takového PDF přes schránku (clipboard) do jiných aplikací. Taky je už desítky let v konfiguraci `fontinst` chyba, která způsobila, že znaky `ď` a `ť` vypadají obludně. Z toho důvodu \mathcal{C} Splain nepodporuje mnoho dalších fontových rodin s výjimkou těch, které byly zmíněny v sekci 5.3. Fonty použité při `\input ctimes` až `cpalatin` jsou postiženy uvedeným problémem při nastaveném kódování T1. Při implicitním kódování podle \mathcal{C} Sfontů je ovšem použita jiná implementace zmíněných rodin fontů a ta popsanou chybu s `ď` a `ť` neobsahuje a dovolí v PDF dokumentu vyhledávat a kopírovat český text.

¹⁾ METAFONT je rodná sestra \TeX u poskytující mocný programovací jazyk na tvorbu fontů. Mezi tvůrci fontů se tato sestra pro svou složitost příliš neprosadila.

Místo přehazování vidlemi a hledání perly v kupce sena mezi fonty s volnou licencí z \TeX ové distribuce je často účelnější si zakoupit kvalitní český font. Jak jej pak zařadit do \TeX ové distribuce je řečeno v následující sekci F.2.

Do kódování T1 byly převedeny rovněž fonty Computer Modern. V této formě se přechodně jmenovaly DC fonty a později EC fonty. Tyto projekty dnes považujeme za slepou vývojovou větev, protože se opíraly o stále méně užívaný METAFONT. Byly nahrazeny *Latin Modern fonty*. To jsou fonty vycházející z Computer Modern, nabízející se v soudobých formátech Type1 (PostScript) a OTF (OpenType, Unicode). Pro \TeX jsou předgenerovány v nejrůznějších kódováních:

```
ec-lmr10.tfm ... Latin Modern Roman at10pt v T1 kódování
ts1-lmr10.tfm .. Latin Modern v TeX-companion kódování (doplňkové)
cs-lmr10.tfm ... Latin Modern Roman at10pt v kódování CSfontů
qx-lmr10.tfm ... Latin Modern Roman at10pt v polském kódování
t5-lmr10.tfm ... Latin Modern Roman at10pt ve vietnamském kódování
l7x-lmr10.tfm .. Latin Modern Roman at10pt v litevském kódování
rm-lmr10.tfm ... Latin Modern Roman at10pt regular math
texansi-lmr10.tfm      ... kódování podle Y&Y
lmroman10-regular.otf ... Latin Modern Roman at10pt, Unicode font
```

Zkratky názvů fontů tak, aby se vešly do 8 znaků, dovedl k dokonalosti Karl Berry, který pro nové fonty v \TeX u navrhl zkratkové schéma ve tvaru:

```
<písmolijna><rodina><varianta><modifikace><kódování><šířka>
např.: pzcmi8z = p(PostScript Adobe) zc(Zapf-Chancery) mi(MediumItalic)
          8z(kódování CSfontů)
nebo:  uhvbc8tn = u(URW) hv(Helvetica) b(Bold) c(SmallCaps)
          8t(kódování T1) n(Narrow).
```

Je dobré vědět, že 8t značí T1 kódování a 8z kódování $\text{\textit{CSfontů}}$. Detailní popis tohoto zkratkového schématu je k nalezení v [2]. Toto schéma nepočítá s různými designovanými velikostmi pro varianty, předpokládá se jen běžná velikost 10 bodů.

Některé nově instalované fonty opouštějí konečně zkratky a jejich názvy souborů jsou více informativní. Záleží hodně na autorovi fontu, jak se rozhodne.

Někdy se plain \TeX oví uživatelé mohou setkat s \LaTeX ovým dokumentem, ve kterém je zajímavý font, který by chtěli využít. Bohužel, pohled do zdrojového souboru \LaTeX ového dokumentu nenapoví, jak se jmenuje soubor .tfm, který je třeba použít v příkazu `\font` při sazbě v plain \TeX u. Mezi příkazem `\font` a \LaTeX ovým uživatelem leží totiž tlustá vrstva NFSS, která je natolik nepřehledná, že se obtížně dá trasovat, jaký příkaz `\font` je na primitivní úrovni v \LaTeX u nakonec použit. Zkušenosti ukazují, že prohledávání desítek makrosouborů \LaTeX u je ztráta času. Doporučuji tedy pro takový případ uchýlit se k několika trikům. Do místa \LaTeX ového dokumentu, kde je provedena sazba vyhlídnutým fontem, můžete napsat `\fontname\font`. To vytiskne parametry příkazu `\font` aktuálně použitého fontu, tj. název souboru a případné zvětšení. Nebo se podívejte do výsledného DVI \LaTeX ového dokumentu příkazem `dvitype <soubor>`. Tam je seznam použitých fontů. Stejný pohled do PDF příkazem `pdf fonts` nemusí být dostatečně informativní, protože pdf \TeX mohl už použitý font transformovat pomocí tzv. virtuálních fontů na něco úplně jiného.

F.2 Instalace nového OTF fontu

Nový font dnes pořídíte ve formátu OpenType (přípona `otf`). Zaměříme se tedy na tento formát.

Nová rozšíření \TeX u ($\text{X}\mathfrak{e}\text{L}\mathfrak{A}\text{T}\mathfrak{E}\text{X}$ a $\text{L}\mathfrak{u}\mathfrak{A}\text{T}\mathfrak{E}\text{X}$) dokáží číst pomocí příkazu `\font` přímo OTF fonty (v Unicode). O této možnosti pojednává dodatek B. V následujícím textu je popis, jak instalovat OTF font pro klasický \TeX nebo $\text{p}\mathfrak{d}\mathfrak{f}\mathfrak{T}\mathfrak{E}\mathfrak{X}$.

Instalaci si můžete vyzkoušet na volně dostupném kvalitním fontu LidoSTF ze Střešovické písmolijny¹⁾, který se hodně podobá písmu Times, ale není to Times. Po stažení uvedené rodiny fontů máte na disku soubory

```
LidoSTF.otf LidoSTFBold.otf LidoSTFItalic.otf LidoSTFBoldItalic.otf
```

kteřé obsahují čtyři varianty uvedeného písma ve formátu OpenType. Dále použijete volně dostupný nástroj `otftotfm`²⁾.

Při převodu OpenType fontu pro 8bitový \TeX je třeba nejprve rozhodnout, v jakém kódování bude font instalován. V následujícím příkladě bylo zvoleno kódování podle $\mathcal{C}\mathfrak{s}$ fontů, takže je třeba z \TeX ové distribuce do aktuálního adresáře zkopírovat soubor `xl2f.enc` s popisem uvedeného kódování. Pak na příkazový řádek napíšete

```
otftotfm --no-virtual -e xl2f LidoSTF.otf -fkern -fliga LidoSTF-8z
```

V parametru příkazu je nejprve řečeno, že nechcete font instalovat pomocí mechanismu virtuálních fontů. Dále je uvedeno kódování, pak zdrojový soubor a pomocí přepínačů `-fkern` a `-fliga` je vysloveno přání, že chcete do výsledné metriky pro \TeX zakomponovat informace o kerningových párech a o ligaturách, což je standardní požadavek. V závěru je uveden požadovaný název výstupní metriky pro \TeX . Uvedený příkaz vytvoří soubory:

```
LidoSTF.pfb LidoSTF-8z.tfm
```

Kromě těchto dvou souborů vznikly ještě další, které neupotřebíte. Soubor `LidoSTF.pfb` obsahuje konverzi vstupního `LidoSTF.otf` do Type1 formátu, se kterým umějí pracovat 8bitové \TeX y. Tento soubor obsahuje kompletně všechny znaky původního `LidoSTF.otf`, nejen ty, které jsou vyjmenovány v použitém kódování. Soubor `LidoSTF-8z.tfm` obsahuje metrické údaje vybraných maximálně 256 znaků podle zvoleného kódování.

Nyní je třeba tyto dva soubory zkopírovat do \TeX ové distribuce. První někam mezi ostatní Type1 fonty a druhý mezi ostatní `tfm` soubory. Po instalaci těchto souborů je třeba spustit příkaz `texhash`, aby se s novými soubory \TeX seznámil a uměl je najít. Kromě toho je třeba do souboru `pdfTeX.map`, který je přítomen někde v distribuci, přidat řádek:

```
LidoSTF-8z LidoSTF "XL2encodingF ReEncodeFont" <xl2f.enc <LidoSTF.pfb
```

První slovo je název \TeX ové metriky, druhé slovo je název fontu, pak následuje požadavek na překódování podle `xl2f.enc` a nakonec je jméno `pfb` souboru. První dva údaje tohoto řádku vypíše program `otftotfm` na terminál. On vypíše ten řádek kompletní, ale v případě překódování nabízí možnost použít nový kódovací soubor, který pro účely jen tohoto fontu vytvořil. To je vhodné ignorovat, protože je lepší nezanášet si distribuci balastem.

Uvedený řádek se typicky do souboru `pdfTeX.map` nepřidává ručně, ale prostřednictvím nástroje `updmap`. To mírně komplikuje situaci. Je třeba vytvořit soubor `neco.map` s uvedeným řádkem (nebo více podobnými řádky) a v adresáři s tímto souborem spustit příkaz `updmap --enable Map=neco.map`. Nyní můžete písmo vyzkoušet třeba v takovém testovacím souboru:

```
\font\f = LidoSTF-8z
\f Tady je zkouška nového písma.
\end
```

¹⁾ <http://www.stormtype.com>

²⁾ <http://www.lcdf.org/type/>

Analogicky lze konvertovat ostatní soubory `LidoSTFBold.otf`, `LidoSTFItalic.otf` a `LidoSTFBoldItalic.otf`. Dejme tomu, že jsou pro ně vytvořeny stejnojmenné metriky, jen s připojením `-8z` k názvu, aby bylo zřejmé, že jsou v kódování podle \mathcal{S} fontů. Konečně můžete vytvořit jednoduchý fontový soubor `cs-lido.tex`:

```
\message{FONT: LidoSTF - \string\rm, \string\it, \string\bf, \string\bi.}

\font\tenrm = LidoSTF-8z          \sizespec
\font\tenbf = LidoSTFBold-8z      \sizespec
\font\tenit = LidoSTFItalic-8z    \sizespec
\font\tenbi = LidoSTFBoldItalic-8z \sizespec
\tenrm

\ifx\font\corkencoded \else \input chars-8z \fi
\ifx\mathpreloaded X\else \input tx-math \fi
\endinput
```

a v dokumentu lze psát `\input cs-lido` a následně vesele přepínat mezi variantami fontů (`\rm`, `\bf`, `\it`, `\bi`) a nastavovat jejich zvětšování/zmenšování způsobem popsáním v sekci 5.4.

Je pochopitelně možné stejné fonty připravit v dalším kódování. Pokud zopakujete uvedený postup a místo souboru `xl2f.enc` použijete soubor `tex256.enc`¹⁾ a k názvu metrik budete připojovat `-8t` (místo `-8z`), dostanete další sadu čtyř metrik. Soubory `.pfb` budou příkazem `otftotfm` vygenerovány podruhé zcela shodně a není nutné je znovu instalovat. Skutečně, fonty `.pfb` obsahují mnohdy mnohem více než 256 znaků. Jednotlivé metriky `.tfm` je možné vnímat jako 256znakový výřez nad fontem a takových výřezů může být nad jedním `.pfb` fontem vytvořeno libovolně mnoho.

Do fontového souboru `cs-lido.tex` pak lze přidat výhybku, která vybere font podle kódování nastaveného uživatelem:

```
\message{FONT: LidoSTF - \string\rm, \string\it, \string\bf, \string\bi.}

\ifx\font\corkencoded \def\tmp{8t } \else \def\tmp{8z } \fi

\font\tenrm = LidoSTF-\tmp          \sizespec
\font\tenbf = LidoSTFBold-\tmp      \sizespec
\font\tenit = LidoSTFItalic-\tmp    \sizespec
\font\tenbi = LidoSTFBoldItalic-\tmp \sizespec
\tenrm

\ifx\font\corkencoded \else \input chars-8z \fi
\ifx\mathpreloaded X\else \input tx-math \fi
\endinput
```

Do fontových souborů se většinou přidává ještě jedna výhybka, která umožní přímé čtení OTF fontů $\mathcal{X}\mathcal{T}\mathcal{E}\mathcal{X}$ em nebo $\mathcal{L}\mathcal{u}\mathcal{a}\mathcal{T}\mathcal{E}\mathcal{X}$ em (`\ifx\font\unicoded`). Viz například soubor `cs-termes.tex`.

¹⁾ V souboru `tex256.enc` jsou ligatury na slotech 27 až 31 zapsány jako `/ff`, `/fi`, `/fl`, `/ffi`, `/ffl`, ale font `LidoSTF` má ligatury `/f.f`, `/f.i`, `/f.l`, `/f.f.i`, `/f.f.l`. Přejmenujte si tedy soubor `tex256.enc` podle svého a opravte v něm názvy ligatur a změňte název kódování. Teprve pak budou ve vygenerovaném fontu fungovat ligatury.

F.3 Kódování textových fontů

Je-li font ve formátu OpenType, je jeho kódování podle Unicode a pracovat s ním přímo mohou jen $\text{X}\text{\LaTeX}$ nebo $\text{Lua}\text{\LaTeX}$. Klasický $\text{T}\text{\LaTeX}$ nebo $\text{pdf}\text{\LaTeX}$ mohou sice taky s těmito fonty pracovat (po konverzi do PFB formátu, jak bylo ukázáno v předchozí sekci), ale načítají metriky, které obsahují jen nejvýše 256znakový výběr znaků z takového fontu. Takových výběrů jednoho fontu mohou načíst libovolně mnoho, tj. nakonec nemají omezení na množství dostupných znaků jednoho OpenType fontu¹⁾.

Tabulka F.3.1 vypisuje kódování používané jako implicitní v \LaTeX . Kódování je označováno XL2 a mírně rozšiřuje kódování IL2 originálních \LaTeX fontů (načtených ve formátu). Toto kódování je použito ve fontech čtených fontovými soubory `lmfonts.tex`, `ctimes.tex`, `cpalatin.tex`, `cs-terms.tex`, atd. (viz sekci 5.3) za předpokladu, že uživatel nespecifikuje kódování fontů odlišné od implicitního.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl
1x	ı	ı	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	ˆ	æ	œ	ø	Æ	Œ	Ø
2x	ˆ	!	”	#	\$	%	&	'	()	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	ı<	=	ı>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[“\]	^	ˆ
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	-{	—	”}	˜	ˆ
8x	...	†	‡	•	£	¶	€		™	©	®			% ₀₀	‹	›
9x									À		,		-	ˆ	«	»
Ax		À		É	Ê	Ë		§		Š		Ť			Ž	
Bx		à		é	ê	ë			à	š		ť			ž	
Cx	Ř	Á	Â		Ä	Í		Ç	Č	É		È	Ë	Í	Î	Ď
Dx			Ñ	Ó	Ô		Ö		Ř	Ů	Ú		Û	Ý		
Ex	í	á	â		ä	í		ç	č	é		è	ë	í	î	ď
Fx			ñ	ó	ô		ö		ř	ů	ú		û	ý	„	“

Tabulka F.3.1 Kódování XL2 označované též jako kódování \LaTeX fontů nebo kódování 8z. Kódovací soubor má název `x12.enc`. Na pozicích, kde jsou vytištěny dvě varianty, platí varianta první. Druhá odpovídá kódování XT2 označovaného též jako 8u. Toto kódování respektující ASCII se používá typicky pro fonty emulující strojpis.

Po načtení rodiny fontů v tomto kódování může uživatel použít (kromě sekvencí uvedených v sekci 3.3) další řídicí sekvence pro tisk jednotlivých znaků:

`\currency` ₤ `\sterling` £ `\euro` € `\trademark` ™ `\registered` ®
`\textbullet` • `\ellipsis` ... `\clq` , `\crq` ‘ `\flq` ‹ `\frq` ›

\LaTeX nabízí kromě implicitního kódování \LaTeX fontů možnost použít také kódování T1 (viz tabulku F.3.2) a v případě $\text{X}\text{\LaTeX}$ u nebo $\text{Lua}\text{\LaTeX}$ u i Unicode. Je-li na začátku

¹⁾ Ale jen uvnitř každé 256znakové sady zvlášť funguje automatické vyrovnání mezer mezi písmeny (kerning) a automatická tvorba ligatur.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	`	´	^	~	¨	”	°	˘	˙	–	·	ˆ	˜	,	‹	›
1x	“	”	„	«	»	—	—		o	l	J	ff	fi	fl	ffi	ffl
2x	□	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	-
8x	Ǻ	Ą	Ć	Č	Ď	Ě	Ě	Ǧ	Ǧ	Ł	Ł	Ń	Ń	Đ	Ő	Ŕ
9x	Ř	Ś	Ŝ	Ş	Ť	Ť	Ů	Ů	Ÿ	Ž	Ž	Ž	IJ	İ	đ	§
Ax	ǻ	ą	ć	č	ď	ě	ě	ğ	í	ı	ı	ń	ñ	η	ő	ŕ
Bx	ř	ś	ŝ	ş	ť	ť	ű	ű	ÿ	ž	ž	ž	ij	ı	ı	£
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Ø	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	œ	ø	ù	ú	û	ü	ý	þ	ß

Tabulka F.3.2 Kódování T1 označované též jako kódování podle Corku nebo kódování 8t. Kódovací soubor má název `tex256.enc`.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	`	´	^	~	¨	”	°	˘	˙	–	·	ˆ	˜	,		
1x			„			—	—		←	→	^	^	^	^		
2x	¸				\$			'			*		,	=	.	/
3x	o	1	2	3	4	5	6	7	8	9			‹	—	›	
4x														U		○
5x		Ö						Ω				ℓ		ℓ	↑	↓
6x	`		★	o o	+								☛	∞	♪	
7x		ø		f											~	=
8x	˘	˙	”	”	†	‡	‖	‰	•	°C	\$	¢	f	©	W	N
9x	G	P	£	R	?	ı	đ	™	‰	¶	B	№	ℓ	e	◦	SM
Ax	{	}	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	®	®	—
Bx	°	±	²	³	´	µ	¶	·	×	¹	º	√	¼	½	¾	€
Cx																
Dx							×									
Ex																
Fx							÷									

Tabulka F.3.3 Kódování TS1 označované též jako T_EX text companion symbols nebo kódování 8c. Kódovací soubor má název `fontools_ts1.enc`.


dokumentu `\input t1code`, dává tím uživatel najevo, že chce použít fonty v kódování T1. Je-li na začátku dokumentu napsáno `\input ucode`, bude sazba ve fontech v Unicode.

Po nastavení kódování fontů (`\input t1code` nebo `\input ucode`) *musí* následovat zavedení fontového souboru, který v takovém případě zavede fonty ve zvoleném kódování. Implicitně zavedená rodina \mathcal{C} Sfontů je totiž po změně kódování nepoužitelná. Chcete-li zůstat v rodině fontů vzhledově totožné rodině Computer Modern, je třeba použít `\input lmfons`.

Nastavení kódování musí také předcházet před použitím makra na inicializaci vzorů dělení slov (`\chyp`, `\shyp` případně další dle dodatku G). Je tedy vhodné dávat příkaz `\input t1code` jako úplně první příkaz do dokumentu a během zpracování dokumentu toto nastavení neměnit. Příklad:

```
\input t1code      % kódování fontů T1
\input cs-terms    % Termes z TeXGyre (podobný Times Roman) v kódování T1
\chyp             % vzory dělení slov češtiny
Tady je možné psát hezky česky ve fontu Termes.
\bye
```

Kódování T1 sice nabízí rozsáhlejší sadu znaků různých evropských jazyků píšících latinkou, ale bohužel chybí znak Euro, protože kódování bylo vytvořeno a uzavřeno dřív, než si Evropa vymyslela Euro. V \TeX u se nicméně mohou použít další metriky téhož fontu s jiným výběrem znaků – tzv. expertní sady. Nejčastěji se používá kódování TS1, které je zobrazeno v tabulce F.3.3.

Makra mohou být naprogramována tak, že když uživatel napíše řídicí sekvenci odpovídající znaku z expertní sady (například `\exleaf` pro  nebo `\exnumero` pro \mathbb{N}), tato sekvence zjistí, zda font s expertním kódováním pro aktuální verzi základního fontu je přítomen. Pokud ano, přechodně do něj přepne ve správné velikosti a verzi fontu, vytiskne znak a vrátí se do původního fontu. Tuto vlastnost mají v \mathcal{C} Splainu například makra ze souboru `exchars.tex`. Příklad:

```
\input ctimes      % Times Roman v kódování XL2
\input exchars      % Další znaky z expertního kódování

Tady vytisknu lísteček: \exleaf, {\it lísteček v kurzívě: \exleaf} a
{\bf lísteček tučně: \exleaf}.
\bye
```

Jaké jsou po načtení souboru `exchars.tex` k dispozici řídicí sekvence, lze zjistit pohledem do tohoto souboru. Všechny speciální znaky tam deklarované mají pro přehlednost v názvu předponu `ex`. V uvedeném souboru najdete taky návod, jak přidat další metriky s expertním kódováním a jak deklarovat další znaky podle vlastní potřeby.

Dodatek G

Více jazyků v $\text{\textit{Csplain}}$

Implicitně načítá $\text{\textit{Csplain}}$ při generování formátu následující vzory dělení slov:

- $\text{\textbackslash enPatt=0}$... (implicitní vzor z $\text{\textit{plainT\TeX u}}$) v ASCII kódování
- $\text{\textbackslash csILtwo=5}$... čeština v ISO-8859-2
- $\text{\textbackslash skILtwo=6}$... slovenština v ISO-8859-2
- $\text{\textbackslash csCork=15}$... čeština v T1 kódování
- $\text{\textbackslash skCork=16}$... slovenština v T1 kódování
- $\text{\textbackslash csUnicode=115}$... čeština v Unicode (jen pro $\text{\textit{X\TeX}}, \text{\textit{LuaT\TeX}}$)
- $\text{\textbackslash skUnicode=116}$... slovenština v Unicode (jen pro $\text{\textit{X\TeX}}, \text{\textit{LuaT\TeX}}$)

Jednotlivé vzory dělení se v dokumentu zapínají pomocí maker $\text{\textbackslash uslang}$, $\text{\textbackslash cslang}$ a $\text{\textbackslash sklang}$. Makro $\text{\textbackslash uslang}$ zapne také $\text{\textbackslash nonfrenchspacing}$ a ostatní přepínače jazyků zapínají $\text{\textbackslash frenchspacing}$. Jsou zachovány i staré názvy přepínačů: $\text{\textbackslash ehyphe}=\text{\textbackslash uslang}$, $\text{\textbackslash chyphe}=\text{\textbackslash cslang}$ a $\text{\textbackslash shyph}=\text{\textbackslash sklang}$. Přejít k novým názvům přepínačů se zkratkami jazyků podle ISO 639-1 je důsledkem nové možnosti v $\text{\textit{Csplain}}$ použít více než 50 různých jazyků.

V souboru $\text{\textit{hyphen.lan}}$ na řádcích 48 až 160 jsou za dvojtečkami skryty jazyky a kódování zhruba ve tvaru:

```
\let\csCork=\patterns      % Czech
\let\skCork=\patterns      % Slovak
:\let\deCork=\patterns     % German
:\let\frCork=\patterns     % French
:\let\plCork=\patterns     % Polish
:\let\cyCork=\patterns     % Welsh
:\let\daCork=\patterns     % Danish
...
:\let\saUnicode=\patterns  % Sanskrit
:\let\ruUnicode=\patterns  % Russian
:\let\ukUnicode=\patterns  % Ukrainian
:\let\hyUnicode=\patterns  % Armenian
:\let\asUnicode=\patterns  % Assamese
...
```

Jazyky, které mají abecedu kompletně obsaženou v T1 kódování (latinkou píšící evropské jazyky) mají vzory dělení připraveny v T1 (Cork) a v Unicode. Ostatní jazyky mají vzory dělení jen v Unicode. Vzory dělení v kódování IL2 (kódování $\text{\textit{Csfont}}$ ů) jsou navíc připraveny pouze pro češtinu a slovenštinu. Kódování IL2 a T1 funguje výhradně v $\text{\textit{T\TeX u}}$ a $\text{\textit{pdfT\TeX u}}$. Kódování Unicode funguje výhradně v $\text{\textit{LuaT\TeX u}}$ a $\text{\textit{X\TeX u}}$.

Stačí si vybrat jazyky, odstranit u vybraného názvu vzoru dělení dvojtečku a znovu vygenerovat formát $\text{\textit{Csplain}}$. Nebo můžete přidat vzor bez zásahu do souboru $\text{\textit{hyphen.lan}}$ vhodně zvolenou zprávou na příkazovém řádku při generování formátu. Například:

```
pdftex -jobname pdfcsplain -ini -enc "\let\plCork=y \input csplain-utf8.ini"
```

nebo

```
pdftex -jobname pdfcsplain -ini -enc "\let\allpatterns=y \input csplain-utf8.ini"
```

Jakmile je načten vzor dělení pro nový jazyk, je automaticky připraven k tomu odpovídající přepínač jazyka tvaru $\text{\textbackslash \langle zkratka \rangle lang}$, tedy například $\text{\textbackslash delang}$ po načtení

`\deCork`. Seznam všech načtených vzorů dělení je v makru `\pattlist`, takže pomocí `\show\pattlist` se můžete podívat, jaké vzory dělení jsou ve formátu načteny.

Na začátku zpracování dokumentu `CSplainem` je aktivován vzor `\enPatt` a jsou připraveny k použití vzory dělení `\<zkratka>ILtwo`. Takže přepínače `\<zkratka>lang` přepínají na tyto vzory dělení. Vzory dělení typu `\<zkratka>Cork` budou fungovat až po `\input t1code` a vzory dělení `\<zkratka>Unicode` budou fungovat až po `\input ucode`.

`CSplain` definuje pro každý jazyk s načtenými vzory dělení makro `\lan:<číslo>` jako `\<zkratku jazyka>`, například `\lan:5` stejně jako `\lan:15` expandují na `cs`. Programátor maker toho může využít. Pomocí konstrukce `\csname lan:\the\language\endcsname` se může dozvědět, jaký jazyk je zrovna aktivní. `TeX`ový registr `\language` je totiž nastaven na číslo zrovna používaného vzoru dělení slov.

Programátor maker dále může využít toho, že přepínače `\<zkratka>lang` volají makro `\initlanguage{<zkratka>}`. Toto makro implicitně neudělá nic, ale programátor si je může předefinovat dle svého. Protože je makro `\initlanguage` zavoláno těsně za přiřazením `\language=<číslo>` v kontextu:

```
\<zkratka>lang -> \language=<číslo>\relax
\initlanguage{<zkratka>}\frenchspacing
\lefthyphenmin=<lhm>\righthyphenmin=<rh>%
\message{<text>}
```

je možné, aby programátor maker odebral další inteligenci maker `\<zkratka>lang` a převzal je do vlastních rukou. Třeba definuje:

```
\def\initlanguage #1#2#3\message#4{#3\csname lg:#1\endcsname}
```

Toto řešení přebírá z makra `\<zkratka>lang` jen nastavení registrů `\lefthyphenmin` a `\righthyphenmin`¹⁾, ale ruší implicitní `\message` i nastavení `\frenchspacing`. Místo toho se předpokládá, že budou definována makra `\lg:<zkratka>`, ve kterých budou tyto věci (a možná mnoho dalších jako třeba nastavení implicitního fontu) řešeny pro každý jazyk individuálně.

Přepínání mezi kódováními vzorů dělení je řízeno makry `\iltwolangs`, `\corklangs` a `\unicodelangs`. Po spuštění takového makra jsou připraveny vzory dělení s odpovídajícím kódováním. Takže po `\iltwolangs` (což je výchozí nastavení) přepínače pracují takto:

```
\cslang % ... vzor dělení \csILtwo
\sklang % ... vzor dělení \skILtwo
```

Po použití `\corklangs` přepínače nyní pracují takto:

```
\cslang % ... vzor dělení \csCork
\sklang % ... vzor dělení \skCork
```

Konečně po použití `\unicodelangs` (v `XYTeXu` nebo `LuaTeXu`):

```
\cslang % ... vzor dělení \csUnicode
\sklang % ... vzor dělení \skUnicode
```

Makro `\corklangs` se spustí při čtení souboru `t1code.tex` a makro `\unicodelangs` se spustí v souboru `ucode.tex`. Proto uživateli zdůrazňujeme, aby přepínač vzorů dělení použili až po `\input t1code` resp. `\input ucode`.

¹⁾ Ta obsahují minimální povolený počet písmen při dělení slova zleva a zprava.

Dodatek **H**

Literatura a odkazy

- [1] ČSN 01 6910. Úprava písemností zpracovaných textovými procesory.
- [2] Karl Berry. *Filenames for T_EX fonts*. <http://tug.org/fontname/html/>.
- [3] Michael Doob. *Jemný úvod do T_EXu*. CSTUG, 1997.
<ftp://math.feld.cvut.cz/pub/cstex/doc/jemny.tar.gz>.
- [4] John D. Hobby, *METAPOST – a user’s manual*. Soubor `mpman.pdf` v distribucích T_EXu,
<http://www.tug.org/metapost.html>.
- [5] František Chvála, *O možnostech pdfT_EXu*, Zpravodaj CSTUG 1/2005.
- [6] Donald Ervin Knuth. *Computer & Typesetting A: The T_EXbook*. Addison Wesley, 1994.
- [7] LuaT_EX development team. *LuaT_EX Reference Manual*.
<http://www.luatex.org/documentation.html>.
- [8] David Nečas. *Yetiho typografický bestiář*.
<http://physics.muni.cz/~yeti/tex/>.
- [9] NTS team. *The eT_EX manual*, version 2, February 1998.
<http://ctan.org/pkg/etex/>.
- [10] Petr Olšák. *CSplain*, 1992–2014. <http://petr.olsak.net/csplain.html>.
- [11] Petr Olšák. *OPmac*, 2012–2014. <http://petr.olsak.net/opmac.html>.
- [12] Petr Olšák. *První setkání s T_EXem*.
<http://petr.olsak.net/ftp/cstex/doc/prvni.pdf>.
- [13] Petr Olšák. *T_EXbook naruby*. Konvoj Brno, 2001.
<http://petr.olsak.net/tbn.html>.
- [14] Petr Olšák. *encT_EX*. <http://www.olsak.net/enctex.html>.
- [15] Petr Olšák. *T_EX v jednoduchém UNIXovém prostředí*. Zpravodaj CSTUG 3/2012.
<http://petr.olsak.net/ftp/olsak/texloop/texunix.pdf>.
- [16] Petr Olšák. *Jednoduchá grafika PDF-primitivně*. Zpravodaj CSTUG 1/2013.
<http://petr.olsak.net>.
- [17] Petr Olšák. *PDFuni – akcenty v PDF záložkách*. Zpravodaj CSTUG 1/2013.
<http://petr.olsak.net>.
- [18] Will Robertson, Khaled Hosny. *The X_YT_EX reference guide*.
<http://ctan.org/pkg/xetexref>.
- [19] Pavel Satrapa. *L^AT_EX pro pragmatiky*.
<http://www.nti.tul.cz/~satrapa/docs/latex/>.
- [20] Till Tantau. *TikZ & PGF*, manual. Soubor `pgfmanual.pdf` v distribucích T_EXu,
<http://sourceforge.net/projects/pgf/>.
- [21] Hàn Thế Thành et al. *The pdfT_EX user manual*. Soubor `pdftex-1.pdf` v distribucích T_EXu, <http://www.tug.org/applications/pdftex/>
- [22] T_EXLive. <http://www.tug.org/texlive/>.
- [23] MikT_EX <http://miktex.org/>.
- [24] PDF reference, http://www.adobe.com/devnet/pdf/pdf_reference.html.

Dodatek I

Rejstřík řídicích sekvencí

V rejstříku je vedle každé řídicí sekvence zkratkou uveden její výchozí význam. Pak následuje stránka, kde je řídicí sekvence v textu vyložena nebo významně zmíněna. Nejsou uvedeny ostatní stránky, kde je řídicí sekvence jenom použita. Seznam zkratk:

p	primitivní příkaz <code>\TeXu</code>
e	expandovatelný primitivní příkaz <code>\TeXu</code>
r	interní registr <code>\TeXu</code>
pp	primitivní příkaz nebo registr <code>pdfTeXu</code>
pe	primitivní příkaz nebo registr <code>eTeXu</code>
px	primitivní příkaz <code>X_YTeXu</code>
pl	primitivní příkaz <code>\LuaTeXu</code>
pc	primitivní příkaz nebo registr <code>encTeXu</code>
mp	makro <code>plainTeXu</code>
mc	makro <code>\CSplainu</code>
mo	makro <code>OPmac</code>
ma	makro z <code>ams-math.tex</code> nebo <code>tx-math.tex</code>
ro	registr z <code>OPmac</code>
zp	znaková konstanta z <code>plainTeXu</code>
zc	znaková konstanta z <code>\CSplainu</code>
za	znaková konstanta z <code>ams-math.tex</code> nebo <code>tx-math.tex</code>
fp	přepínač fontu z <code>plainTeXu</code>

Rejstřík zatím není ve veřejně přístupném textu zařazen.