TEXtutor

První krůčky v (plain)TEXu

Kamil Toman

ČS Armády 1930
390 01 Tábor
Česká republika

E-mail: toman@horac.ta.jcu.cz

25. června 1999

Předmluva

TEX je typografický systém vhodný pro elektronické zpracování textů. TEXtutor je text, který se tímto systémem zabývá a který vznikl z nedostatku krátkých volně šiřitelných článků v češtině zejména na internetu. V textu se snažím TEX přiblížit i lidem, kteří se s TEXem setkali jen okrajově nebo se s ním ještě nesetkali vůbec.

Snažil jsem se o co největší stručnost, ale doufám, že se to příliš neodrazilo ve srozumitelnosti textu. I přes tuto snahu nebude asi celý text hladce čitelný a občas bude zřejmě nutné se podívat při čtení na některou pasáž zpět. Jako zdroj dalších informací lze použít i vlastní zdrojové soubory TEXtutoru nebo níže uvedenou literaturu.

Při psaní jsem se často inspiroval anglickými texty o TEXu. Převzal jsem z nich i některé tabulky a přehledy, přeložené nebo s mírnými úpravami je pak používám v textu. Především matematická a také poslední část je inspirovaná skvělou knihou D. E. Knutha – **The TEXbook**.

Na rozdíl od většiny ostatních dokumentů podobného zaměření, TEXtutor není a ani se nesnaží být nějakou vyčerpávající sbírkou informací o TEXu, ani klasickou učebnicí. Jde spíš o takové letmé „nahlédnutí pod pokličku“.

Rád bych také na tomto místě poděkoval všem, kteří mi při přípravě tohoto textu pomohli. Zvláště děkuji panu P. Olšákovi za množství cenných připomínek, které přispěly k celkovému zlepšení kvality textu.

Seznam knih použitých při psaní a doporučených k přečtení:

- The TEXbook (D. E. Knuth)
- A Gentle Introdution to TEX(Michael Doob)
- Typografický systém TEX(Petr Olšák)
- TEXbook naruby (Petr Olšák)
  `ftp://math.feld.cvut.cz/pub/olsak/tbn`
- TEXfor the Beginner (Wynter Snow)
- Typografický manuál (Vladimír Beran)

Mnoho dalších zajímavých odkazů, programů a dokumentace, která nějak s TEXem souvisí můžete nalézt na webové stránce sdružení $\mathcal{CS}$TUG: `http://www.cstug.cz`.

Cože to ten TEX je

Nový sázecí systém zaměřený na vytváření nádherných knih
– a hlavně knih, co obsahují spoustu matematiky

D. E. Knuth

Nezbývá než s autorem TEXu souhlasit. TEX [čti tech] je velmi kvalitní systém pro počítačovou sazbu. Je široce rozšířený zejména v univerzitním prostředí, má širokou podporu na internetu a existuje téměř na všech počítačových platformách. Celý systém včetně zdrojových souborů v jazyce WEB, C nebo Pascal je *public domain* (tedy zcela zdarma).

Existuje více obecně podporovaných formátů TEXu. Základem všeho je plainTEX (krátce plain) navržený Donaldem Knuthem. Tento dokument popisuje základy práce právě s tímto formátem[1]. Nad tímto základním formátem je vytvořena spousta dalších nadstaveb, z nichž nejznámější jsou LATEX a $\mathcal{AMS}$TEX.

LATEX vytvořil Leslie Lamport. Zjednodušuje práci v TEXu, používá jednotný styl dokumentu a umožňuje automatizované vytváření obsahů, rejstříků, křížových odkazů, záhlaví, práci s obrázky apod. Je natolik úplný a komplexní, že v podstatě ani nevyžaduje znalost plainTEXu. $\mathcal{AMS}$TEX, navržený Michaelem Spivakem, umožňuje především jednoduché zadávání složitých matematických formulí. Z množství dalších bych rád jmenoval např. $\mathcal{AMS}$-LATEX, L$\mathcal{AMS}$TEX, texinfo...

Použití

K některým věcem se TEX zvlášť hodí:

- k psaní matematických publikací, článků, skript apod. (tady asi TEX nenajde vážnějšího soupeře),
- k vytváření velmi rozsáhlých textů,
- k psaní málo členitého textu s důrazem na kvalitní zpracování (beletrie),
- k sazbě krátkých informačních textů s důrazem na kvalitní zpracování (popisky, vizitky),
- k vytváření textů pro tisk na osvitových zařízeních.

Méně se TEX hodí k vytváření členité sazby, ve které jsou jako základní prvky na stránce rozmístěny obrázky a text se „napouští“ mezi ně.

[1] Pokud se někde v dokumentu zmiňuji o TEXu, pak mám na mysli právě formát plain, pro české dokumenty pak csplain. To je formát plain mírně modifikovaný tak, aby bylo možné pracovat s českými a slovenskými texty.

Něco je jinak

Samozřejmě bychom měli říkat „s uchy“
protože jsou to ucha jako třeba u hrníčku.
Ale „s ušima“ je libozvučnější.

J. Matoušek, J. Nešetřil

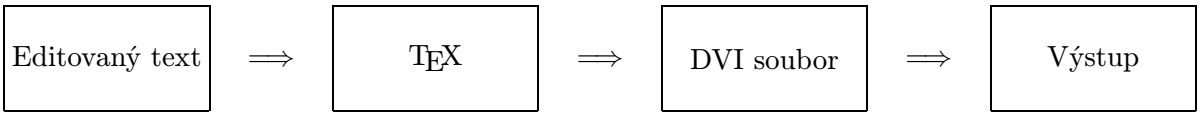
Rád bych upozornil na jednu malou skutečnost. Práce v TEXu je poněkud odlišná od práce v jiném DTP systému. TEX pracuje v dávkovém režimu a funguje podobně jako překladač. Přeloží zdrojový text do nějakého výstupního tvaru. Z toho plyne, že k napsání zdrojového tvaru dokumentu můžeme použít svůj oblíbený textový editor za předpokladu, že

- nebude do vytvářeného textu přidávat žádné pro něj specifické formátovací značky,
- nebo umí export do souboru pro TEX.

Nás bude ale zajímat hlavně první možnost.

Vytvoření dokumentu

Nejdříve vytvoříme zdrojový tvar dokumentu. Předáme ho TEXu jako vstupní soubor. TEX si náš text přečte a podle něj vytvoří na disku výsledný dokument – soubor s příponou ‘`.dvi`’ (DeVice Independent). Ten už si můžeme s pomocí nějakého programu vytisknout na tiskárně nebo třeba jen prohlédnout na obrazovce. Celé to vypadá asi takhle:



Je možné průběžně sledovat výstup TEXu v nějakém prohlížeči, a přitom ve vedlejším okně modifikovat zdrojový text a TEX jednoduše spouštět na pozadí. Na starších operačních systémech něco takového nebylo možné. Příprava textu pro TEX může být výrazně rychlejší než v běžném WYSIAWYG[1] editoru. Můžeme v klidu napsat vlastní text v jednoduchém textovém editoru a teprve nakonec se zabývat formátováním dokumentu. Můžeme se tedy soustředit na myšlenku a nemusíme se hned rozptylovat hledáním formy dokumentu. Protože zdrojový text lze psát v jakémkoli editoru, můžeme jej přenášet na libovolný počítač a upravovat jej v podstatě kdekoli. Teprve pro konečné zpracování dokumentu potřebujeme počítač s instalovaným TEXem[2].

[1] What You See Is Almost What You have Got

[2] Na většině operačních systémů s příkazovým řádkem se typicky TEX spouští podle jména použitého formátu. Řekněme například, že potřebujeme zpracovat zdrojový dokument ‘`dokument.tex`’. Anglický dokument zpracujeme příkazem ‘`tex dokument`’ (automaticky se zavede formát `plain`). Pokud je náš dokument v češtině, napíšeme ‘`cstex dokument`’ případně ‘`csplain dokument`’ (v obou případech se automaticky zavede modifikovaný formát `csplain`). Všimneme si, že přípona ‘`.tex`’ se na příkazovém řádku psát nemusí. Za povšimnutí také stojí, že příponu ‘`.tex` mají často i zdrojové soubory, které nejsou napsány pro formát `plain` (například soubory pro LATEX). Při pokusu o zpracování takového souboru výše uvedenými příkazy ohlásí TEX pravděpodobně chybu. Zdrojové soubory psané v LATEXu můžeme zpracovat například příkazem ‘`latex dokument`’ nebo (pokud jde o český dokument) ‘`cslatex dokument`’.

První text

Ono před několika málo sty lety málo lidí umělo psát.
Ale ti, co uměli psát, ti psát uměli. To se dá číst i teď.

V+W

Nejlépe bude patrně začít jednoduchým příkladem. Pokuste se do svého počítače do souboru `pokus.tex` přepsat následujících několik řádek pomyslného inzerátu.

```
\chyph % inicializace českého dělení slov
{\bf Hledáme:} schopného archeologa, který je schopen pracovat i v~extrémních
podmínkách na vykopávkách v~okolí tzv. {\it Čertovy jámy}. Doprava i veškeré
záležitosti týkající se dlouhodobého  pobytu jsou náležitě zajištěny.

Finanční a organizační záležitosti  řeší pobočka naší firmy v~Dolní Lhotě.
Na další spolupráci se těší členové výzkumného týmu firmy
{\it Belzebub a synové, Pekelná 24.}
\bye
```

Pokud jsme někde neudělali chybu, měli bychom vidět v prohlížeči následující text:

Hledáme: schopného archeologa, který je schopen pracovat i v extrémních podmínkách na vykopávkách v okolí tzv.*Čertovy jámy.* Doprava i veškeré záležitosti týkající se dlouhodobého pobytu jsou náležitě zajištěny.

Finanční a organizační záležitosti řeší pobočka naší firmy v Dolní Lhotě. Na další spolupráci se těší členové výzkumného týmu firmy *Belzebub a synové, Pekelná 24.*

Jistou představu o tom, jak vypadá zdrojový dokument TEXu tedy máme. Jedná se o jednoduchý text, který je poněkud obohacen o *příkazy*, které dále určují výsledný vzhled dokumentu. Jejich bližší vysvětlení a rozbor dalších detailů této ukázky si ponecháme na později. Zatím si jen všimneme, že TEXu příliš nezáleží na mezerách, které jsme do dokumentu vložili. Jedinou výjimkou je prázdný řádek, který, jak vidíme, způsobil rozdělení textu na dva samostatné odstavce[1].

Pojďme teď zkusit o něco složitější příklad a ukažme si na něm některé z možností a hlavních rysů TEXovských dokumentů. Ačkoliv se ukázka tentokrát daleko více podobá jakési změti znaků, překonejme počáteční odpor a pokusme se ještě jednou pečlivě[2] přepsat následující ukázku do souboru `pokus2.tex`[3]. Pak tento soubor zpracujme TEXem, formátem csplain.

```
%%%%%%%% Zde začíná "oblast definic" %%%%%%%%%%
\input ctimes                  % rodina Times Roman
\font\titlefnt=\fontname\tenbf\space scaled\magstep1  % větší font
\newdimen\indskip \indskip=15pt % výčty budou odsazeny 15pt
\def\sbitem{\par\hangindent=\indskip -- }  % definice zkratky \sbitem
\def\di#1{{\it #1\/}}          % vzhled definovaného pojmu
\def\title#1\par{              % definice nadpisu:
   \removelastskip\bigskip     %   odmaže poslední vert. mezeru a přidá vlastní
   \centerline{\titlefnt #1}   %   centrovaný text nadpisu větším fontem
```

[1] Tentýž efekt lze docílit vložením příkazu '`\par`'.

[2] včetně prázdných řádků

[3] Texty za znakem '%' přepisovat nemusíte, jsou to opravdu jen komentáře.

```
   \par\nobreak\medskip}       %   konec řádku, zakázaný zlom, menší mezera
\def\definition{               % vzhled definice
   \removelastskip\medskip     %   odmaže poslední vert. mezeru a přidá vlastní
   {\bf Definice: }}           %   začne definici
\let\itemskip=\medskip         % kolem výčtu bude jen menší mezera
\def\alphaOneToM{\alpha_1\ldots\alpha_m} % výčet znaků alfa s indexy 1...m
\def\betaOneToN{\beta_1\ldots\beta_n} % výčet znaků beta s indexy 1...n

%%%%%%%% Následuje "vlastní text" dokumentu %%%%%%%%%

\title Ukázka druhá

Definujme si několik fundamentálních pojmů teorie jazyků.  \definition
Řekneme, že $A$ je \di{Abeceda}, je-li $A$ libovolná konečná množina.
Prvky abecedy $A$ budeme nazývat \di{písmena}, resp. \di{symboly}.
Každou konečnou posloupnost písmen nazveme \di{slovem}, její délku pak
\di{délkou slova}. Dále:

\itemskip
\sbitem \di{Prázdné slovo}, tj. slovo délky~0, značíme $\Lambda$.
\sbitem Množinu všech slov nad abecedou $A$ značíme $A^*$.
\sbitem Libovolnou podmnožinu $L$ množiny $A$ nazveme \di{jazykem} nad $A$
        ($L \subseteq A^*$).
\sbitem Pro slova $\alpha = \alphaOneToM$ a~$\beta = \betaOneToN$
        definujeme operaci \di{conc} tak, že
        $conc(\alpha,\beta)=\alpha\beta=\alphaOneToM\betaOneToN$
(pro $n,m \in N$). Jedná se tedy o~složení řetězců, resp. slov.
\itemskip

\bye
```

Za naši pečlivost a trpělivost budeme odměněni následujícím výsledkem:

Ukázka druhá

Definujme si několik fundamentálních pojmů teorie jazyků.

Definice: Řekneme, že A je *Abeceda*, je-li A libovolná konečná množina. Prvky abecedy A budeme nazývat *písmena*, resp. *symboly*. Každou konečnou posloupnost písmen nazveme *slovem*, její délku pak *délkou slova*. Dále:

– *Prázdné slovo*, tj. slovo délky 0, značíme Λ.
– Množinu všech slov nad abecedou A značíme A^*.
– Libovolnou podmnožinu L množiny A nazveme *jazykem* nad A ($L \subseteq A^*$).
– Pro slova $\alpha = \alpha_1 \ldots \alpha_m$ a $\beta = \beta_1 \ldots \beta_n$ definujeme operaci *conc* tak, že $conc(\alpha, \beta) = \alpha\beta = \alpha_1 \ldots \alpha_m\beta_1 \ldots \beta_n$ (pro $n, m \in N$). Jedná se tedy o složení řetězců, resp. slov.

Komentář k formě příkladu

Podívejme se blíže na strukturu celého příkladu. V části označené „vlastní text dokumentu“ jsou použity značky `‘\title’`, `‘\definition’`, `‘\sbitem’` a `‘\itemskip’`, které ohraničují jen **logické části** dokumentu (vymezení nadpisu, oblasti definice, uvedení dalšího bodu ve výčtu prvků, uvedení skupiny výčtu prvků a podobně).

Konkrétní formátovací informace jsou umístěny mimo vlastní text v „oblasti definic“. Tady je definován vzhled a velikost fontů, mezer, tvarů různých značek atd.

Právě takové oddělení formy od obsahu je velice užitečné a žádoucí.[4]

[4] Především začátečníkům to ovšem obvykle působí značné potíže. Obsah se (nevhodně) mísí s formou, dokumenty jsou pak velice nepřehledné a jen velmi těžko se mění či opravují.

Vysvětlení použitých značek v příkladu

Jednotlivé značky, které řídí formátování a vymezují strukturu dokumentu se nazývají *řídící sekvence* a jsou (typicky) ve tvaru ‘\⟨*nějaké slovo*⟩’. Řídící sekvence bývají také nazývány *příkazy*, neboť jimi přikazujeme provedení nějaké akce.

Část řídících sekvencí, které jsme v ukázce použili, je definována už v samotném TEXu resp. v použitém formátu csplain. Zbylé řídící sekvence jsme definovali sami v „oblasti definic“ příkazem ‘`\def`’. Podívejme se na ně teď podrobněji:

- ‘`\title`’ značí nadpis; text nadpisu následuje za sekvencí a je ukončen novým odstavcem[5].
- ‘`\titlefnt`’ přepne na větší font; o který font půjde, se definuje o něco výše příkazem ‘`\font`’.
- ‘`\sbitem`’ je sekvence, která uvozuje položku ve výčtu prvků. Postará se o grafickou podobu výčtu prvků (odsadí text, uvodí jej pomlčkou).
- ‘`\itemskip`’ vytvoří vertikální mezeru, která oddělí výčet prvků od ostatního textu. Používá se na začátku i na konci výčtu.
- ‘`\definition`’ je sekvence uvozující oblast matematické definice
- ‘`\di`’ je sekvence, která určuje definovaný pojem.

Ostatní řídicí sekvence jsou definovány v (cs)plainu nebo přímo zabudovány v TEXu a hlouběji se s nimi seznámíme v dalších kapitolách.[6] Poznamenejme snad jen něco k užitým speciálním znakům:

- Znak ‘`~`’ (vlnka) znamená nedělitelnou mezeru. Používá se na místech, kde nechceme nebo není povoleno zalomení řádku.
- Znak ‘`%`’ uvozuje komentář až do konce řádku.
- Znak ‘`$`’ přepíná do matematického režimu a zpět (bude podrobně vysvětleno v některé z následujících kapitol).
- Znak ‘`^`’ v matematickém režimu uvozuje horní index (exponent).
- Podobně znak ‘`_`’ v matematickém režimu uvozuje dolní index.
- Znaky ‘`{`’ a ‘`}`’ (složené závorky) mají v TEXu tři mírně odlišné významy.
  1. Obklopují těla definic za příkazem ‘`\def`’.
  2. Obklopují parametry některých řídicích sekvencí. Například v příkazu ‘`\di{Prázdné slovo}`’ bude parametrem ‘`Prázdné slovo`’. Kdybychom složené závorky vynechali, bylo by parametrem příkazu ‘`\di`’ pouze slovo ‘`Prázdné`’.
  3. Samotné znaky ‘`{`’ a ‘`}`’ vymezují *skupiny*, ve kterých jsou veškerá nastavení lokální. Viz například text ‘`{\it Čertovy jámy}`’, kde je se lokálně zapíná kurzívové písmo. Složené závorky ‘`{`’ a ‘`}`’ musejí vždy párovat.

Všimněme si také, že na začátku každé z ukázek je příkaz ‘`\chyph`’. Tato řídící sekvence, definovaná ve formátu `csplain`, inicializuje české dělení slov[7]. Při vytváření českých dokumentů bychom ji nikdy neměli vynechat!

Změna formátu dokumentu

Experimentujme dál. Předpokládejme, že budeme chtít upravit dokument druhé ukázky tak, aby měl grafickou podobu podobnou textům stejného zaměření.

Přepneme tedy na písmo Computer Modern, které je běžné v matematických textech a manuálech programů. Tato rodina je po v TEXu nastavena implicitně, stačí tedy vymazat řádek ‘`\input ctimes`’, který zapínal písmo Times Roman. Jednotlivé položky uvodíme je puntíky

[5] Připomeňme si, že odstavce oddělujeme prázdným řádkem (nebo, je-li to nutné, příkazem ‘`\par`’).

[6] Naštěstí se dá většinou poměrně přesně odhadnout už z názvu, co daný příkaz znamená.

[7] Podobně ‘`\shyph`’ inicializuje slovenské vzory pro dělění slov

namísto pomlček a nebudeme mezi nimi dělat mezery. Pro definované pojmy použijeme raději polotučné písmo. Naopak, nápis ‘definice’ vysázíme kurzívou. Ještě můžeme zkusit o něco zvětšit nadpis (použijeme při definování fontu ‘`\magstep2`’ namísto ‘`\magstep1`’). Do vlastního textu nebudeme zasahovat, vzhled dokumentu je determinován „oblastí definic“.

```
%%%%%%%% Zde začíná "oblast definic" %%%%%%%%%%
\chyph                          % inicializace českého dělení slov v csplainu
\font\titlefnt=\fontname\tenbf\space scaled\magstep2  % větší font
\newdimen\indskip \indskip=15pt % výčty budou odsazeny 15pt
\def\defitem#1{{\bf #1}}        % vzhled definovaného pojmu
\def\title#1\par{               % definice nadpisu:
   \removelastskip\bigskip      %   odmaže poslední vert. mezeru a přidá vlastní
   \centerline{\titlefnt #1}    %   centrovaný text nadpisu větším fontem
   \par\nobreak\medskip}        %   konec řádku, zakázaný zlom, menší mezera
\def\definition{                % vzhled definice
   \removelastskip\medskip      %   odmaže poslední vert. mezeru a přidá vlastní
   {\noindent\it Definice: }}           %   začne definici
\let\itemskip=\smallskip        % kolem výčtu bude jen drobná mezera

\def\alphaOneToM{\alpha_1\ldots\alpha_m} % výčet znaků alfa s indexy 1 ... m
\def\betaOneToN{\beta_1\ldots\beta_n} % výčet znaků alfa s indexy 1 ... n
```

Dobře. Do dalších úprav jako automatického číslování všech definic se prozatím pouštět nebudeme. Podívejme se, jaká je grafická podoba dokumentu teď:

Ukázka druhá

Definujme si několik fundamentálních pojmů teorie jazyků.

Definice: Řekneme, že A je **Abeceda**, je-li A libovolná konečná množina. Prvky abecedy A budeme nazývat **písmena**, resp. **symboly**. Každou konečnou posloupnost písmen nazveme **slovem**, její délku pak **délkou slova**. Dále:

- **Prázdné slovo**, tj. slovo délky 0, značíme Λ.
- Množinu všech slov nad abecedou A značíme A^*.
- Libovolnou podmnožinu L množiny A nazveme **jazykem** nad A ($L \subseteq A^*$).
- Pro slova $\alpha = \alpha_1 \ldots \alpha_m$ a $\beta = \beta_1 \ldots \beta_n$ definujeme operaci **conc** tak, že $conc(\alpha, \beta) = \alpha\beta = \alpha_1 \ldots \alpha_m\beta_1 \ldots \beta_n$ (pro $n, m \in N$). Jedná se tedy o složení řetězců, resp. slov.

Často užívané definice lze umístit do jiného souboru než vlastní text dokumentu. Na začátku je pak stačí načíst příkazem ‘`\input` ⟨*jméno souboru*⟩’. Pokud se později rozhodneme změnit styl našich dokumentů, změníme část definic a pouhým opětovným zpracováním TEXem dostaneme zbrusu nové verze dokumentů, které reflektují náš nový typografický názor.

Zdrojový text

People hardly ever use TEX's primitive control sequences in their manuscripts, because the primitives are... well... so primitive

D. E. Knuth

Vzhled výsledného dokumentu a chování TEXu tedy můžeme ovlivnit různými příkazy, tzv. řídícími sekvencemi. Řídicí sekvence může být dvou typů: Řídící slovo nebo řídící znak.

Řídící slovo

Řídící slovo je uvozeno zpětným lomítkem[1] a skládá se z jednoho nebo více písmen anglické abecedy (ne čísel nebo jiných znaků[2]), za ním následuje mezera nebo nebo nějaký jiný znak (mimo písmen abecedy). TEX tak bude schopen snadno odlišit řídící slovo a jeho případné parametry od okolního textu. Například:

```
\input MS
```

TEX zpracuje jako řídící sekvenci `input` s parametrem `MS`, která znamená, že se má načíst soubor `MS.tex`. (Protože v parametru příkazu není uvedena přípona souboru, doplní se automaticky přípona `'.tex'`). Pokud napíšeme jen

```
\inputMS
```

TEX to pochopí jako řídící sekvenci ze sedmi písmen a ohlásí chybu (nedefinovaná sekvence `inputMS`).

Mezera za řídícím slovem se tedy bere jako *oddělovník*, ne jako „normální“ mezera. Uvidíme, že je to velice užitečná vlastnost, ale může dělat i potíže. Třeba v následující ukázce se zdá být všechno v pořádku,

```
\TeX používá mezeru za řídícím slovem jako oddělovník.
```

ale výstupní dokument svědčí o opaku.

TEXpoužívá mezeru za řídícím slovem jako oddělovník.

Taková chyba je o to záludnější, že není ve zdrojovém souboru dobře vidět a snadno ji přehlédneme i při zběžném čtení výsledného dokumentu. Ale jak tedy vložit do textu mezeru hned za řídící slovo? Jedna z možností je použít řídící znak `'\ '` (*escape znak* následovaný mezerou).

```
\TeX\ používá mezeru za řídící sekvencí jako oddělovník.
```

dá výsledek

TEX používá mezeru za řídícím slovem jako oddělovník.

[1] Přesněji tzv. *escape znakem*.

[2] V csplainu může řídicí slovo obsahovat i písmena s háčky a čárkami.

Řídící znak

Druhým typem řídících sekvencí je řídící znak. Za *escape znakem* následuje jediný znak, kterým *není* písmeno. Příklad:

```
o~50 \% více
```

TEX zpracuje text `‘o 50 ’`, pak zjistí řídící sekvenci `‘\%’`, za ní *mezeru* a za ní slovo `‘více’`. Všimneme si, že mezera za řídicím znakem se na rozdíl od mezery za řídicím slovem objeví ve výstupu. Je to proto, že TEX předem ví, že řídící sekvence bude dlouhá jenom jeden znak a následující znak už má další význam. Poznamenejme, že řídicí sekvence `‘\%’` je v plainu definována tak, že se vytiskne znak %, který běžně nelze do textu napsat přímo, protože uvozuje komentář ve zdrojovém textu.

Struktura řídících sekvencí

Plain TEX zná ve svém základním slovníku asi 900 řídících sekvencí. Naštěstí jejich názvy jsou natolik logické, že si toho příliš pamatovat nemusíme.

Přibližně 300 řídících sekvencí jsou tzv. primitiva. To jsou úplně nejnižší dále nedělitelné příkazy TEXu (např. `‘\input’`, `‘\def’`, `‘\vskip’`[3]), ze kterých jsou všechny zbylé řídící sekvence vystaveny. Takové složené řídící sekvence se nazývají makra.

Zjistit složení řídících sekvencí, jejich definici, můžeme pomocí příkazu `‘\show’`. Řekněme například, že jsme se od zkušenějšího kolegy dozvěděli o existenci příkazu `‘\thinspace’`. Ten má podle našeho kolegy vytvořit jakousi malou mezeru. Podívejme se na toto makro zevrubněji.

Na příkazovou řádku operačního systému napíšeme jen `‘cstex’` (nebo `‘csplain’`). Objeví se dvě hvězdičky a TEX očekává, že zadáme jméno souboru ke zpracování. My nic takového ovšem zadávat nechceme, a tak odpovíme `‘\relax’`. TEX zavede základní formát a vypsáním hvězdičky nám oznámí, že je připraven s námi přímo komunikovat:

```
*\show\thinspace
```

vypíše na standardní výstup definici `‘\thinspace’`.

```
> \thinspace=macro:
->\kern 1.6667em
```

Naproti tomu

```
*\show\kern
> \kern=\kern
```

což značí, že se jedná o primitivum TEXu. Pokud nevíme, co `‘\kern’` dělá, žádné `‘\show’` nám nepomůže. Tady nezbývá než vyhledat příslušnou část v dokumentaci TEXu nebo některé z referenčních příruček, kde je uveden seznam všech primitivních příkazů spolu s jejich popisem[4]. Tam se pak dočteme, že pokud TEX dostane ke zpracování příkaz `‘\kern’` následovaný délkovým parametrem, vytvoří nedělitelný výplněk uvedené velikosti ve vodorovném směru (v odstavci) nebo ve směru svislém (mezi dvěma odstavci). Teď už jsou obě definice jasné.

[3] Vloží vertikální mezeru dané délky.

[4] Takový seznam je uveden například v knize TEXbook naruby.

Práce s mezerami a prázdnými řádky

TEX nerozlišuje na vstupu mezeru od tabulátoru. Za normálních okolností znamená výskyt (i vícenásobný) kteréhokoliv ze jmenovaných znaků mezi slovy pouze to, že se má do výsledného dokumentu vložit jediná mezera. Nebo, jak jsme již viděli, mezera za řídícím slovem slouží pouze jako oddělovač – na výstup se nedostane. Mezery nebo tabulátory na začátku každého nového řádku se ignorují (předpokládá se, že jde o součást odsazení textu ve zdrojovém dokumentu). Můžeme tedy kdykoli bez obav začít psát text odsazený několik mezer od kraje. To podstatně zvyšuje přehlednost zdrojového textu a přitom to neovlivní formátování výstupu.

Konec řádku je TEXem rovněž interpretován jeko mezera. Prázdný řádek ale znamená ukončení aktuálního odstavce. Obsahuje-li řádek jenom mezery a tabulátory, je považován za prázdný. Více prázdných řádků za sebou se chová jako jediný prázdný řádek, prázdné řádky před prvním odstavcem v dokumentu nemají na výstup žádný vliv.[5]

Mezera, která vzniká na konci každého řádku se dá „zamaskovat“ komentářovým znakem. Viz následující ukázku:

```
Tady bude mezera mezi
A~a B, ale před%
    C nebude.
```

Tady bude mezera mezi A a B, ale předC nebude.

Tento příklad není příliš praktický. Je na něm ovšem dobře vidět mechanismus skrývání mezer, který se může později velmi hodit například pro přehledné vkládání poznámek pod čarou (viz příkaz `‘\footnote’`). Všimněme si také třetího řádku naší ukázky. Mezery z kraje jakoby na něm vůbec nebyly.

Speciální znaky, kategorie

Samotné řídící sekvence nebudou pro řízení TEXu stačit. Jako většina podobných systémů potřebuje i TEX nějaké speciální znaky. Každý znak se tedy nemusí ze zdrojového souboru bez úhony dostat až do výstupu (vzpomeňme například komentářový znak `‘%’`). Může mít zvláštní význam nebo naopak může významem splývat se znakem jiným. V předchozím odstavci jsme se seznámili s jedním takovým „splývajícím“ znakem: tabulátor, který se chová stejně jako mezera.

TEX pracuje s různými významy vstupních znaků (tzv. kategoriemi). Rozlišujeme tyto kategorie:

Číslo	*Význam*	*Kterým znakům je tento význam přidělen*
0	Uvození řídicí sekvence	`‘\’`
1	Začátek skupiny	`‘{’`
2	Konec skupiny	`‘}’`
3	Matematický přepínač	`‘$’`
4	Oddělovač v tabulkách	`‘&’`
5	Konec řádky	`‘<enter>’`, přesněji znak s kódem 13
6	Parametr v definicích	`‘#’`
7	Horní index v matematice	`‘^’`
8	Dolní index v matematice	`‘_’`
9	Ignorovaný znak	znak s kódem 0, v textu nemá co dělat
10	Mezera	mezera nebo tabelátor
11	Písmeno	`[A-Z|a-z]`, v csplainu i s akcenty
12	Jiný znak	vše, co nepatří do kategorií 0-11 a 13-15

[5] Explicitně lze takové chování TEXu potlačit např. příkazy `‘\obeylines’` a `‘\obeyspaces’`.

13	Aktivní znak	`‘~’`
14	Uvození komentáře	`‘%’`
15	Zakázaný znak	znak s kódem 127, v textu nemá co dělat

V průběhu zpracování dokumentu je možné významy vstupních znaků měnit změnou čísla jejich kategorie. K tomu slouží příkaz `‘\catcode’`. Například `‘\catcode‘\*=14’` způsobí, že nejen procento, ale i hvězdička bude uvozovat komentář. Obráceně `‘\catcode‘\%=12’` zařadí procento mezi „normální znaky“. Nebo `‘\catcode‘\\=12’` způsobí, že ztrácíme jediný znak pro uvození řídicích sekvencí a už nebudeme v dokumentu schopni zadat žádnou řídicí sekvenci. To není užitečný příklad. Na druhé straně tento příklad dokumentuje, že není radno si s kategoriemi zahrávat, pokud důkladně nevíme, co děláme. Rozsah tohoto textu neumožňuje uvést všechny záludnosti kategorií v TEXu, takže se budeme raději snažit (pokud možno) výchozí významy znaků neměnit.

Skupiny

TEX má ještě další zajímavou a maximálně užitečnou vlastnost. D. E. Knuth totiž do návrhu TEXu zabudoval základní vlastnost strukturovaného programování. Jsou to skupiny. Skupina se v textu uzavírá do dvojice složených závorek: `‘{`⟨*vlastní skupina*⟩`}’`.

V TEXu pro ně platí obdobná pravidla jako ve většině strukturovaných programovacích jazyků. To, co platí ve vnější skupině, bude platit i ve vnitřní. Naopak to, co platí pouze ve vnitřní skupině, po jejím ukončení platit přestane. V předchozí kapitole jsme v ukázce uvnitř skupin použili fontové přepínače `‘\bf’`, `‘\it’` a `‘\tt’` v kombinaci se složenými závorkami tak, aby nastavení fontu bylo pouze lokální.

Do skupin se často sdružují řídící sekvence, ale i samotné části textu. Dají se využít i k různým trikům. Jako příklad můžu uvést další dvě řešení při potížích s mezerami za makrem `‘\TeX’`.

```
{\TeX} používá mezeru za řídícím slovem jako oddělovník.
```

nebo

```
\TeX{} používá mezeru za řídícím slovem jako oddělovník.
```

Funguje to velice jednoduše. Jako oddělovník se použijí složené závorky, přičemž tím, že jsme dali makro `‘\TeX’` do určité skupiny, výstup nezměníme. Triviální. Jenom poznamenám, že prázdná skupina nic neprodukuje, slouží ve valné většině případů jako jakási logická zarážka.

Prázdná skupina se dá také použít také k potlačení automatického vyrovnání mezer mezi znaky[6]. Srovnejme ‘VLTAVA’ (`‘VLTAVA’`) s nesprávným ‘VLTAVA’ (`‘V{}L{}T{}A{}V{}A’`).

Aktivní znak, nechceme rozdělit řádek

Aktivní znak se v TEXu chová stejně jako řídicí sekvence. Dá se definovat jako makro a může mít spoustu dalších významů. Plain implicitně nastavuje jediný znak jako aktivní, sice znak `‘~’` (vlnka). Přiděluje mu význam mezery, která bude stejně velká, jako kterákoli jiná mezera mezi slovy, ale je v ní zakázáno zlomit řádek. Příklady použití takové mezery:

```
V~případě neslabičných předložek.
1.~prosince 1998, 1.~12.~1998, K.~H.~Borovský, písmeno~A, MS~Windows.
```

Psaním vlnek za neslabičné předložky se při pořizování textu zabývat nemusíme, dají se doplnit později některou z jednoduchých utilit, které bývají součástí distribuce TEXu. Na ostatní případy musíme myslet sami.

[6] používá se také termín *kerning*.

Někdy se hodí do sazby vložit užší nedělitelnou mezeru. Doporučuji definovat na začátku dokumentu řídicí sekvenci ‘\,’ jako ‘\thinspace’ pomocí příkazu ‘\let\,=\thinspace’ a dále psát jen:

```
1\,mm, 100\% muž, ale práce na 100\,\%.
```

Jak dopravit speciální znaky na výstup

Jak víme z předchozího textu, do dokumentu v TEXu nelze napsat přímo následující znaky: `\ { } $ & # ^ _ ~ %`. Tyto problémové znaky rozdělujeme do tří skupin:

- Znaky `$ & # _ %` lze vytisknout pomocí řídicích znaků `\$ \& \# \_ \%`. Tyto znaky budou správně zobrazeny ve všech fontech s výjimkou znaku $ v kurzívě fontu Computer Modern[7]. Tam se dolar změní v libru: £.
- Znaky `\ { }` jsou speciální ASCII znaky, které nemají v mnoha fontech rodiny Computer Modern svůj obraz. Najdeme je jedině ve strojopisu (‘\tt’, viz níže) a v matematických fontech (viz kapitola o matematice).
- Znaky `^` (stříška) `~` (vlnka) jsou ve fontech Computer Modern většinou kresleny jako akcenty ˆ a ˜. Chceme-li je vykreslit samostatně (jako zde), pišme akcentové sekvence pro „prázdné písmeno“: `\^{}` a `\~{}`.

Každý znak můžeme vytisknout příkazem `\char‘\`⟨*znak*⟩. Například `{\tt\char‘\\}` vede na \. Pozor: pokud bychom nepřepnuli do strojopisu, dostaneme znak pro obrácené uvozovky “, protože tento znak je v antikvě Computer Modern na ASCII pozici obráceného lomítka.

Kromě znaků: `\ { }` též další ASCII znaky: `| _ > <` ␣ nemají ve fontech Computer Modern na pozicích podle ASCII stejnou kresbu. Když napíšeme v ‘\rm’ fontu třeba ‘>cosi’, dostaneme ‘¿cosi’, ačkoli znak ‘>’ není nijak speciální. Všechny ASCII znaky jsou správně zaneseny jen ve variantě fontu ‘\tt’. Proto přepnutí do ‘\tt’ a zápis ‘>cosi’ vede správně na ‘>cosi’.

Záměr autora fontu byl takový, že veškeré ASCII ukázky textu budou výhradně sázeny fontem ‘\tt’. Znaky, které odpovídají matematickým značkám, lze bez problému sázet v matematickém módu, například `$a>b$` vede na $a > b$.

Pokud chceme beze změny vytisknout část souboru v ASCII znacích (tzv. verbatim výstup), nezbude než nějakým makrem lokálně v rámci skupiny nastavit ‘\tt’ a změnit kategorie všech speciálních znaků.[8]

[7] Nejčastěji používaná rodina fontů v TEXu.

[8] Řešení najdeme například v [TBN] na straně 29.

Fonty

Však ony tě ty boží mlýny semelou, Bille!

pravil jeden můj kamarád při pohledu
na nadpis v knize vytvořené pomocí
nejmenovaného textového editoru,
přes který se skvěl tučný text:
ZÁLOŽKA NENÍ DEFINOVÁNA

Systém TEX implicitně používá vlastní rodinu písem *Computer modern* (také *public domain*) nebo může používat i libovolný jiný font, ke kterému lze získat metrické informace (např. komerční fonty *Type 1*).

Při běžné práci s variantami písem potřebujeme jen několik málo příkazů. Většinou vystačíme s řídícími sekvencemi

`\rm`	přepne na antikvu (roman)
`\sl`	přepne na skloněný typ písma (slanted)
`\it`	přepne na kurzívu (italic)
`\tt`	přepne na písmo stylu psacího stroje (typewriter)
`\bf`	přepne na polotučné písmo (bold)

Použití fontů

Typografické standardy dovolují používat skloněného písma jako alternativu ke kurzívě. Jeho výhodou je vyšší čitelnost, nevýhodou pak slabší rozpoznatelnost v textu. Každopádně by se fonty jako kurzíva, tučné nebo skloněné písmo měly používat pouze nezbytně dlouhou dobu a pak přejít na standardní typ písma. Není to žádná zkostnatělost, ale spíš logická věc – zkuste si číst dvě strany textu psané kurzívou nebo malým fontem „gothic“ a uvidíte, jak budou vaše oči protestovat.

Největší část textu bývá psána antikvou[1], proto TEX tento font nastaví po inicializaci jako standardní. Není tedy nutné na začátek každého dokumentu psát `‘\rm’`. Pro zvýraznění části textu je zase běžné změnit font jen v rámci podskupiny, takže příkaz `‘\rm’` pro navrácení k fontu roman není nutný.

```
Je to jednoduché, něco napíšu {\it kurzívou\/} nebo {\bf tučně}.
```

Je to jednoduché, něco napíšu *kurzívou* nebo **tučně**.

Povíme si o použité sekvenci `\/` na konci první skupiny. Říká TEXu, že má použít korekci pro mezeru závisející na předcházejícím písmeni. Pokud totiž používáme nějaký typ skloněného písma (kurzíva[2], skloněné[3]), pak může těsně následující „vysoký svislý znak“ překrývat poslední skloněné písmeno. Proto `\/` vkládá dodatečnou mezeru tak, aby k takovému překrytí nedošlo. V této ukázce sice následuje mezera, ale bez použití `\/` by tato mezera byla nesprávně menší. V ukázce v kapitole „První text“ jsme takový oddělovač na konci kurzívy nepoužili,

[1] roman

[2] italic

[3] slanted

protože následovala tečka („nízký znak“), která naopak nesmí být od posledního znaku kurzívy odsazena.

Korekce závisí samozřejmě na tvaru předchozího znaku – větší bude třeba za `‘f’` a menší za `‘x’`. Korekci má každé písmeno fontu (nejen kurzívového), takže ji v případě nutnosti můžeme samozřejmě použít.

Jednotky

Samozřejmě, že všechny rozměry nemusíme zadávat pouze v bodech nebo např. centimetrech. To by jistě nebylo příliš pohodlné.

Následující tabulka dává přehled o tom, které jednotky můžeme v TEXovských dokumentech používat. Protože označení každé jednotky je v TEXu tvořeno vždy ze dvou písmen původního anglického názvu, nebudu údaje v tabulce překládat.

`pt`	point (1 pt = 1/72,27 in ≐ 0,35146 mm)
`bp`	big point (72 bp = 1 in ≐ 0,35277 mm)
`pc`	pica (1 pc = 12 pt ≐ 4,22 mm)
`in`	inch (1 in = 72,27 pt ≐ 25,4 mm)
`cm`	centimeter (2,54 cm = 1 in = 10 mm)
`mm`	milimeter (10 mm = 1 cm)
`dd`	didot point (1157 dd = 1238 pt ≐ 435,1 mm)
`cc`	cicero (1 cc = 12 dd ≐ 5221,3 mm)
`sp`	scaled point (65536 sp = 1 pt)

k tomu má ještě dvě jednotky relativní k fontu

`em`	(šířka velkého ‘M’ – odpovídá např. šířce em-pomlčky)
`ex`	(výška malého ‘x’)

Poslední dvě se dají s výhodou použít, pokud například potřebujeme s některými znaky v textu trochu pohnout. Pomocí takových jednotek je například definován i nápis TEX (`‘\TeX’`).

Zavedení nového fontu

Nový font se do dokumentu zavede příkazem `‘\font’`. Ten načte informace o fontu ze souboru s příponou `‘.tfm’` (TEX font metric) a přiřadí mu pro další potřeby v dokumentu nový přepínač. Zavedení fontu má tvar:

`\font`⟨*přepínač*⟩`=`⟨*jméno souboru*⟩ ⟨*parametry zvětšení*⟩

kde ⟨*jméno souboru*⟩ se píše bez přípony a ⟨*parametry zvětšení*⟩ i rovnítko jsou nepovinné. Plain například zavádí antikvu Computer Modern fontu ve velikosti deseti typografických bodů ze souboru `‘cmr10.tfm’` a csplain pro tytéž účely používá počeštěný `‘csr10.tfm’`. Oba formáty pro tuto antikvu rezervují přepínač `‘\tenrm’` a font je zaveden ve formátu csplain takto:

```
\font\tenrm=csr10
```

Pro antikvu velikosti pěti bodů je použit přepínač `‘\fiverm’` a soubor `‘csr5.tfm’`:

```
\font\fiverm=csr5
```

Ve výše uvedených příkladech jsme nepoužili ⟨*parametry zvětšení*⟩, protože fonty jsou už v souborech `csr10.tfm` a `csr5.tfm` připraveny pro odpovídající velikost. Pokud bychom chtěli nějaký font zvětšit, pak použijeme v místě ⟨*parametry zvětšení*⟩ slovo `scaled` následované koeficientem zvětšení (násobeným tisícem) nebo slovo `at` následované požadovanou velikostí fontu. V následující ukázce oba řádky zavádějí font `csr5.tfm` zvětšený na dvojnásobek:

```
\font\magfiverm=csr5 scaled 2000
```

```
\font\magfiverm=csr5 at 10pt
```

Požití tohoto zvětšeného pětibodového fontu ze souboru `csr5.tfm` vedle skutečně desetibodového fontu ze souboru `csr10.tfm` ukazuje, že tyto soubory opravdu nabízejí různě velké fonty, že neobsahují jen geometrické zvětšení společné předlohy.

Kromě `‘\tenrm’` plain i csplain implicitně zavádějí fonty s přepínači `‘\tensl’`, `‘\tenit’`, `‘\tentt’` a `‘\tenbf’`. Tyto vnitřní přepínače jsou při sazbě desetibodovým písmem ztotožněny s běžně používanými `‘\sl’`, `‘\it’`, `‘\tt’` a `‘\bf’`. Při sazbě v jiné velikosti se ale doporučuje zavést fonty pod jinými vnitřními jmény a přepínače `‘\it’`, `‘\tt’`, atd. ztotožnit s těmito novými jmény. Uživatel pak používá tyto přepínače nezávisle na aktuální velikosti sazby.

Přepínače deklarované příkazem `‘\font’` fungují jen v textovém režimu. V matematickém režimu je situace podstatně složitější a přesahuje rámec tohoto textu (viz [TBN], kapitola 5).

Užitečné zkratky a doporučené postupy

V sazbě se obvykle pro nadpisy používá geometrické zvětšení fontů odstupňované násobky čísla 1,2. Abychom si nemuseli tyto násobky pamatovat, můžeme použít zkratky `‘\magstep0’` (normální velikost), `‘\magstep1’` (1,2 krát), `‘\magstep2’` (1,44 krát), `‘\magstep3’` ($1{,}2 \cdot 1{,}2 \cdot 1{,}2 = 1{,}728$ krát) ... až do `‘\magstep5’`. Pro jemnější dolaďování můžeme ještě použít `‘\magstephalf’` ($\sqrt{1{,}2}$ krát).

Font pro jednoduchý titulek bychom pak mohli zavést jako:

```
\font\titlefont=csr10 scaled\magstep3
```

To nám ale zavede zvětšenou antikvu v normálním duktu. Pokud bychom chtěli polotučnou antikvu, museli bychom pátrat po tom, jak se odpovídající metrický soubor (`tfm`) jmenuje. Výhodnější je využít už zavedeného přepínače `‘\tenbf’` a příkaz `‘\fontname’`, který transformuje následující přepínač fontu zpětně do názvu odpovídajícího metrického souboru. Za ním musíme zapsat mezeru (`‘\space’`) a za mezerou teprve slovo `scaled`. Tento způsob zvětšení fontu byl použit v kapitole „První text“:

```
\font\titulfont=\fontname\tenbf\space scaled\magstep2  % větší font
```

Výhodou této metody je, že pokud se někdy rozhodneme změnit celou rodinu fontů na něco jiného než Computer Modern, pak nemusíme přepisovat jednotlivě u každého příkazu `‘\font’` názvy souborů. To bylo prakticky ukázáno v kapitole „První text“, kde byla zavedena rodina Times `‘\input ctimes’`, což změnilo (mimo jiné) význam přepínače `‘\tenbf’`, takže i přepínač `‘\titulfont’` pracoval s příslušně zvětšeným polotučným řezem fontu Times.

Podívejme se, jaké máme v instalaci TEXu metrické soubory (hledejme je v adresáři `‘tfm’` kdesi v instalaci TEXu). Chceme vědět, jak vypadá k danému souboru odpovídající font? Není nic snadnějšího. Spusťme TEX na dokument s názvem `‘testfont’` a TEX s námi začne interaktivně komunikovat. Na dotaz `‘name of the font to test’` odpovíme názvem metrického souboru (bez přípony). Na hvězdičku reagujeme příkazem `‘\table’` a komunikaci ukončíme příkazem `‘\end’`. Vytvoří se dokument `‘testfont.dvi’` s úplnou tabulkou testovaného fontu.

Změna rodiny fontu

Při přepínání do jiné rodiny fontů než Computer Modern je potřeba postarat se i o patřičnou změnu přepínačů jednotlivých variant písma (`‘\rm’`, `‘\bf’`, `‘\it’`, `‘\sl’`, `‘\tt’`) a nezapomenout patřičně změnit i všechny fonty použité v nadpisech, citacích, popiscích a podobně, aby výsledný dokument působil konsistentním dojmem.

Jednoduché definice pro csplain můžeme najít v následujících souborech[4]:

- `cavantga.tex` – Avantgarde Book

[4] Soubor `ctime.tex` jsme využili v ukázce v kapitole „První text“

- `cbookman.tex` – Bookman
- `chelvet.tex` – Helvetica
- `cncent.tex` – New Century
- `cpalatin.tex` – Palatino
- `ctimes.tex` – Times Roman

Zvětšování celého dokumentu

Pokud chceme zvětšit celý dokument, stačí když napíšeme na začátek dokumentu řídící sekvenci ‘`\magnification=`⟨*faktor pro zvětšení*⟩’. Například:

```
\magnification=\magstep2
```

Tento příkaz zvětší velikost aktuálního fontu o hodnotu ‘`\magstep2`’, ale ponechá implicitní velikost tiskového zrcadla.

Pokud jsme to vyzkoušeli, zjistíme, že se zvětšilo opravdu všechno. Jedna z věcí, kterou bychom možná rádi ponechali, jsou původní jednotky velikosti. Kvůli tomu existuje v TEXu ještě speciální typ jednotek s prefixem ‘`true`’. Příkaz

```
\font\twelwept=csr10 at 12truept
```

zařídí požadovanou velikost fontu.

Kódy ve fontu a dvojitá stříška

Každý má nějakou klávesnici (já např. už přiměřeně nefunkční), ale opravdu málokdo na ní má jednu klávesu pro každý znak – tj. asi 256 kláves. V TEXu máme naštěstí možnost dostat na výstup libovolný znak fontu, pokud známe jeho číslo (kód). Následující příklad

```
\char98 l\char98\char101 c
```

ukazuje, že můžeme klidně napsat dopis blbci i když nemáme zrovna funkční příslušné klávesy (znak č. 98, resp č. 101 je samozřejmě ‘`b`’, resp. ‘`e`’).

Další možností je použití trojznakových sekvencí uvozené dvěma stříškami. Ty se hned na vstupu TEXu promění na jeden vstupní znak. Trojznakové sekvence začínají ‘`^^A`’ (control A, kód 1) a končí ‘`^^Z`’ (control Z, kód 26). Kromě toho se dají použít sekvence ‘`^^@`’ (kód 0), ‘`^^[`’ (27), ‘`^^\`’ (28), ‘`^^]`’ (29), ‘`^^^`’ (30), ‘`^^_`’ (31) a ‘`^^?`’ (127).

Za dvěma stříškami může být též hexadecimální vyjádření kódu znaku pomocí dvojice cifer (`0`–`9`, `a`–`f`). Například ‘`^^41`’ se promění v písmeno A (kód 65). Tento postup umožňuje zadat všech 256 znaků.

Jaký je rozdíl mezi ‘`\char`’ a použitím dvou stříšek? Příkaz ‘`\char`⟨*číslo*⟩’ čteme jako „tisk znaku z aktuálního fontu z pozice ⟨*číslo*⟩“, zatímco dvě stříšky se promění na jeden *vstupní znak*, který může podléhat speciálnímu významu v TEXu. Například ‘`^^25`’ může uvést komentář ve zdrojovém souboru stejně jako znak ‘`%`’.

Akcentová znaménka

Přestože budete určitě používat nějakou z počeštěných distribucí TEXu, můžeme se setkat i s tím, že budete potřebovat akcentové znaménko, které čeština nepoužívá. Pak se budou hodit předdefinované řídící sekvence pro akcenty

Typ	*Výsledek*	*Typ*	*Výsledek*
`\‘o`	ò	`\’o`	ó
`\^o`	ô	`\"o`	ö
`\~o`	õ	`\=o`	ō

`\.o`	ȯ	`\u o`	ŏ
`\v o`	ǒ	`\H o`	ő
`\t oo`	o͡o	`\c o`	o̧
`\d o`	ọ	`\b o`	o̲
`\oe, \OE`	œ, Œ	`\ae, \AE`	æ, Æ
`\aa, \AA`	å, Å	`\o, \O`	ø, Ø
`\l, \L`	ł, Ł	`\ss`	ß

Jemnosti sazby

Ještě několik věcí, které mají jistou souvislost s fonty a zaslouží trochu pozornosti. Zajisté jste si už všimli, že TEX za nás zařizuje například automatické vyrovnávání mezer nejen mezi slovy, ale dokonce i mezi jednotlivými písmeny[5]. TEX automaticky nahrazuje i předem vybrané skupiny znaků jedním znakem, tzv. slitkem (ligaturou). Srovnejme například 'filosofie' s nesprávným 'filosofie'.

Právě tato pečlivost v každém detailu sazby je pro TEX typická. Není tedy divu, že je tedy potřeba rozlišovat

- `běžný spojovník (je-li),`
  běžný spojovník (je-li),
- `pomlčku ve~větě~-- běžně používanou,`
  pomlčku ve větě – běžně používanou,
- `dlouhou pomlčku---používanou především v~anglických dokumentech,`
  dlouhou pomlčku—používanou především v anglických dokumentech,
- `znaménko minus ($-x$).`
  znaménko minus ($-x$).

A nejen spojovníky,

- `budeme přece používat \uv{české uvozovky}`
  budeme přece používat „české uvozovky“
- `a nikoliv ``anglické''`
  a nikoliv “anglické”
- `nebo "tyto uvozovky" -- to je úplně špatně!`
  nebo ”tyto uvozovky” – to je úplně špatně!

Pokud budeme chtít v textu psát výpustky, můžeme sice napsat opravdu tři tečky po sobě, ale pak se nám může někdy přihodit, že TEX za každou tečku přidá příliš malou mezeru a nebude to vypadat dobře. Tomu se můžeme snadno vyhnout, pokud budeme důsledně používat místo třech teček řídící sekvenci '`\dots`'. Rozlišit různé výpustky můžeme i v matematice.

- `podívejme se na $a = (a_1\ldots a_n)$,` a na `$x^n = \overbrace{x\cdots x}^{n-krát}$`
  podívejme se na $a = (a_1 \ldots a_n)$, a na $x^n = \overbrace{x \cdots x}^{n-krát}$

[5] Výsledky na výstupu jsou velice zdařilé, přesto nás TEX nezbavuje možnosti do vzhledu jednotlivých slov dále zasahovat (především pomocí primitiv '`\kern`', '`\raise`' a '`\lower`', o kterých bude ještě řeč).

Krabice

Roses are red,
violets are blue.
Rhymes can be typeset
with boxes and glue.

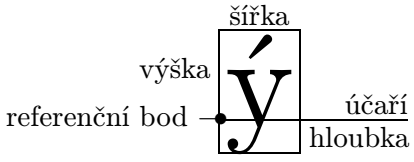
D. E. Knuth

Než se pustíme do vysvětlování, jak lze vzhled dokumentu dále upravovat, jak centrovat řádky, jak vynechat více mezer a podobně, nastíníme trochu způsob, jak TEX přistupuje k formátování textu na výstup.

TEX má zabudovány poměrně inteligentní postupy pro většinu (pro počítač) nepříjemných věcí, jako je rozdělování slov a řádků, ukončování stránek apod. Aby měl TEX trochu ulehčenou práci, definuje dva základní stavební prvky. Jsou to krabice (box) a lepidla (glue). Jak vidíte, přímý překlad zní poněkud komicky. Nadále budeme používat termíny *box* a *pružná mezera* pro glue. Každý znak si můžeme představit jako jeden box, nicméně my budeme termínem *box* nazývat jen složené boxy[1]. Pružná mezera (tj. glue) představuje výplň mezi boxy.

Parametry boxu

- Pozice vzhledem k účaří a referenční bod
- Výška boxu – vymezuje prostor nad účařím
- Hloubka boxu – vymezuje prostor pod účařím
- Šířka boxu – vymezuje šířku boxu (horizontálně)



Parametry pružné mezery

- Přirozená velikost – vymezuje ideální šířku nebo výšku mezery
- Schopnost roztáhnout se – vymezuje toleranci pro roztažení
- Schopnost smrštit se – vymezuje toleranci pro smrštění

Boxy

Boxy navíc mohou být buď horizontálního (hbox) nebo vertikálního (vbox) typu. Materiál v horizontálních boxech se skládá za sebe, kdežto ve vertikálních boxech pod sebe. TEX stavební prvky potom různě kombinuje, vkládá do sebe a sestavuje podle nich jednotlivé stránky výsledného dokumentu. S jednotlivými znaky zachází v podstatě stejně, nicméně chování TEXu se poněkud liší podle aktuálního módu (viz sekci Různé módy TEXu), ve kterém se právě nachází.

Boxy se dají jednoduše vytvářet pomocí řídících sekvencí `‘\hbox’`, `‘\vbox’` nebo `‘\vtop’`, případně `‘\vcenter’`[2].

```
\hbox {Koukni, krabice!}
```

vytvoří box osahující 14 boxů a 1 glue. Můžeme mu určit i velikost

```
\hbox to 10pt {Koukni, krabice!}
```

[1] Na složený box pohlížíme jako na nedělitelnou jednotku, se kterou chceme dále pracovat. Typicky se jedná o složení několika boxů a pružných mezer.

[2] Používá se v matematickém módu

nastaví nové rozměry boxu – toho dosáhne natažením nebo smrštěním pružných mezer uvnitř boxu. Můžeme také TEXu říct, aby přirozenou šířku boxu pozměnil jen trochu.

```
\hbox spread 5pt {Koukni, krabice!}
```

vytvoří hbox o 5pt širší než je jeho přirozená šířka.

Vboxy se konstruují zcela analogicky. Rozdíl mezi řídící sekvencí `'\vbox'` a `'\vtop'` spočívá v umístění referenčního bodu – zda se odvozuje od prvního nebo od posledního subboxu. U boxu zkompletovaného řídící sekvencí `'\vcenter'` bude výška boxu nad matematickou osou rovna hloubce boxu pod matematickou osou.

Možné použití těchto sekvencí je dobře vidět na jednoduchém příkladu. Představme si, že potřebujeme umístit vedle sebe na stránce nějaké informace ve vertikálních boxech. Pokud použijeme konstrukci s řídící sekvencí `'\vbox'`, budeme mít s největší pravděpodobností ve stejné výšce poslední řádky obou vboxů (díky pozici referenčního bodu). Naproti tomu (ze stejného důvodu) při použití `'\vtop'`, budeme mít ve stejné výšce nejspíše první řádky obou vboxů. `'\vcenter'` se používá tehdy, potřebujeme-li, aby oba vboxy ležely na společné horizontální ose (sazba matic).

Různé módy TEXu

TEX se vždy nachází v jednom ze šesti různých režimů, módů.

- Hlavní vertikální mód (vertical mode)
- Vnitřní vertikální mód (internal vertical mode)
- Odstavcový horizontální mód (horizontal mode)
- Vnitřní horizontální mód (internal horizontal mode)
- Vnitřní matematický mód (math mode)
- Display matematický mód (display math mode)

Pro nás není nutné zabývat se módy TEXu nijak zvlášť podrobně. Vnitřní módy mají (až na jistá omezení, jak už název napovídá) po dvojicích v tom pořadí, jak jsou uvedeny pod sebou, velmi podobné vlastnosti. Můžeme tedy situaci často zjednodušit a mluvit jen o vertikálním, horizontálním či matematickém módu.

Po startu je TEX vždy ve stavu hlavního vertikálního módu. Z něj při zpracování dokumentu v případě nutnosti přepíná do módů ostatních. Po sestavení příslušného materiálu *pod sebe* se opět vynořuje zpět. Nedostane-li se zpět do hlavního vertikálního módu (například při předčasném ukončení dokumentu příkazem `'\end'` v rámci nějaké podskupiny), ohlásí chybu.

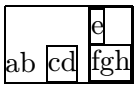
Do odstavcového módu lze vstoupit jen z hlavního vertikálního módu. Zde TEX zpracovává horizontální seznam materiálu, jednotlivé boxy se tedy ukládají *vedle sebe*. Je-li odstavec ukončen (například prázdným řádkem, tj. novým povelem `'\par'`), veškerý kompletovaný materiál se rozláme na jednotlivé řádky, TEX opět přejde do hlavního vertikálního módu a pokračuje v sestavování strany.

Vstup do odstavcového módu může být *explicitní* (pomocí příkazů pro odsazení odstavce `'\indent'` nebo `'\noindent'`) nebo *implicitní*. Protože některé akce nejsou ve vertikálním módu povoleny, je implicitní vstup do odstavcového módu velice častý. Například ve vertikálním módu vůbec nelze provést povel pro sazbu znaku. To je také jeden z hlavních důvodů, proč se tento automatický přechod do horizontálního módu zavádí. Typicky se tato akce provádí při započetí každého nového odstavce. Jsou ale situace, kdy toto přepnutí není tak úplně na první pohled vidět. Například

```
\vbox{ab
      \hbox{cd}
      \vbox{\hbox{e}
            \hbox{f}}}
```

výsledek tohoto příkladu by mohl mnohé překvapit.

e
ab cd fgh



Víme-li ale o implicitním přepínání módů, žádné překvapení nás nečeká. Ukázka začne přechodem do vnitřního vertikálního módu. Potom TEX načte znak `'a'`, tedy povel k sazbě. Ten ale nelze ve vertikálním módu provést, dojde proto k implicitnímu přepnutí na horizontální mód. Vysází se písmeno `'a'`, `'b'` a box obsahující `'cd'`. Následuje příkaz `'\vbox'`, dojde proto k opětovnému přepnutí na vertikální mód. Tentokrát sazbě hboxů nic nebrání, sestavení probíhá normálně ve vertikálním módu.

Hodně povelů TEXu se chová různě v různých módech. Například povel `'\par'` v odstavcovém módu ukončí načítání odstavce, sestaví všechny jeho jednotlivé řádky a přepne do vertikálního módu. Naproti tomu ve vertikálním nebo vnitřním horizontálním módu módu neudělá vůbec nic[3]. V matematickém módu jej použít nelze, TEX vypíše chybové hlášení.

Jako poznámku uveďme seznam všech příkazů, které způsobí implicitní přechod z vertikálního do odstavcového módu: přímá sazba znaku (nebo s pomocí `'\char'` či sekvencí definovanou `'\chardef'`), primitivy `'\accent'`, `'\discretionary'`, `'\hskip'`, `'\hss'`, `'\hfil'`, `'\hfill'`, `'\hfilneg'`, `'\unhbox'`, `'\unhcopy'`, `'\vrule'`, `'\valign'`, `'\noboundary'`, `'\␣'`. Do odstavcového módu TEX implicitně přejde také, pokud ve vertikálním módu narazí na přepínač do matematického módu.

Pružné mezery

S vlastnostmi pružných mezer (glue) je to jednodušší než s boxy. Budeme je používat pořád, i když o tom třeba ani nevíme. Většina automaticky i ručně vkládaných mezer se chová jako *pružná mezera* (představme si především typické mezery mezi slovy či odstavci). Výjimku tvoří některé speciální tuhé výplňky (typicky přesně definované pevné délky[4]).

Nastavování glue můžeme provádět přímo nebo nepřímo (doporučeno). Pro lepší pochopení začneme méně častým přímým nastavením. K tomu můžeme použít řídící sekvence `'\hskip'` a `'\vskip'`. Zkusme to tedy:

```
\vskip 2cm plus 10pt minus 5pt
```

přikáže TEXu vložit „pokud možno“ dvoucentimetrovou mezeru s tím, že je povoleno ji o něco prodloužit nebo zkrátit. Parametry `'plus'` a `'minus'` nejsou povinné a pokud je vynecháme, nastaví se automaticky na 0 pt.

O tom, jak budou mezery skutečně vypadat, rozhoduje poměrně spletitý algoritmus, jehož popis by ale zbytečně zabral mnoho místa[5]. Ve většině případů naštěstí vystačíme s jednodušší představou. Pokusím se postup naznačit na následujícím příkladu.

```
\hbox to 10cm{vlevo\hskip 2cm plus 1cm uvnitř\hskip 4mm plus 0.5cm vpravo}
\hbox to 10cm{vlevo\hskip 2cm plus 1cm uvnitř\hskip 4mm plus 4cm vpravo}
```

Výsledek vypadá možná trochu překvapivě.

vlevo uvnitř vpravo
vlevo uvnitř vpravo

Není na tom ale nic nelogického. TEX vytvoří box s předepsanou šířkou 10 centimetrů a pokusí se ho vyplnit textem a námi určenými mezerami (glue). Zjistí ovšem, že délka textu a celková $2\,\mathrm{cm} + 4\,\mathrm{mm} = 2,4\,\mathrm{cm}$ délka ideálních mezer není dostačující.

[3] To je také důvod, proč se více prázdných řádků chová jako jeden.

[4] Takovou mezeru vloží například příkaz pro odsazení začátku odstavce `'\indent'`.

[5] Podrobně je popsán například v knize The TEXbook, případně TEXbook naruby.

Pokusí se tedy námi udané mezery mezi slovy trochu roztáhnout. Místo, které je potřeba ještě vynechat, rozdělí v poměru, který udávají hodnoty `plus` u každé z mezer (tj. 2 : 1 v prvním a 1 : 4 ve druhém případě). Každou mezeru potom o příslušnou délku roztáhne.

Přesto v prvním případě není TEX spokojen.

```
Underfull \hbox (badness 3323) detected at line 1266
\tenrm vlevo uvnitř vpravo
```

Při svých výpočtech totiž ještě počítá speciální hodnotu tzv. `'badness'`[6], která značí cosi jako estetickou chybu, která při operaci vznikne. Chyba `3323` je již příliš vysoká, a tak nás TEX na vzniklou situaci upozorní. Obdobný postup používá TEX i pro smršťování mezer.

Nepřímé nastavení se provádí podobně, je ovšem ve většině případů výhodnější. Například řídící sekvence `'\vskip'` se dá ve většině případů nahradit jednou z řídících sekvencí `'\smallskip'`, `'\medskip'` nebo `'\bigskip'`. `'\smallskip'` vynechá místo, které se dělává mezi odstavci, `'\medskip'` vynechá místo dvakrát větší než `'\smallskip'` a `'\bigskip'` zase dvakrát větší než `'\medskip'`.

Jde tedy o obecnější typ formátování, při kterém nás zajímá jen celkové rozložení textu, ne přesné číselné údaje. Typicky se jedná také o zarovnání textu vlevo či vpravo, jeho vycentrování apod. Bylo by ovšem hodně nepohodlné určovat případ od případu, kolik místa musíme vynechat. Právě pro tyto účely jsou v TEXu vytvořeny speciální řídící sekvence. První z nich je `'\hfil'`. Pokud použijeme někde v textu řídící sekvenci `'\hfil'`, znamená to „*vynech 0pt s tím, že se může mezera libovolně zvětšovat*“. Podobné je to s řídící sekvencí `'\hfill'`. Funguje přibližně stejně, jen je „nekonečně pružnější“. Pro představu: řekněme, že pokud `'\hfil'` znamená deset, pak `'\hfill'` bude znamenat nejméně milión...

V následujícím příkladu budeme potřebovat řídící sekvenci `'\line'`. Pojďme se nejprve podívat, jak funguje:

```
> \line=macro:
->\hbox to\hsize .
```

Vytvoří tedy obyčejný box. Místo klasické číselné hodnoty pro šířku hboxu používá ale proměnnou `'\hsize'`. V té je uložena celková šířka odstavce. Velmi elegantní a jednoduché.

Teď tedy slíbený příklad:

```
\line{Tohle bude vlevo\hfil}
\line{\hfil Tohle bude uprostřed\hfil}
\line{\hfil Tohle bude vpravo}
\line{Tady bude něco vlevo\hfil a něco vpravo}
\line{\hfil\hfil Tady bude vlevo dvakrát tolik místa jako vpravo\hfil}
\line{\hfill Tohle bude také vpravo \hfil}
```

Tohle bude vlevo

Tohle bude uprostřed

Tohle bude vpravo

Tady bude něco vlevo a něco vpravo

Tady bude vlevo dvakrát tolik místa jako vpravo

Tohle bude také vpravo

Z příkladu je vidět, že účinek `'\hfill'` je daleko silnější než `'\hfil'`. TEX se pak chová, jako kdyby se ve formátovaném textu žádná sekvence `'\hfil'` vůbec nevyskytovala. To má

[6] Podrobněji bude rozebrána v části o formátování odstavců

svůj význam při řešení různých krizových situací v makrech, v normálním případě se doporučuje používat řídící sekvence `‘\hfil’`.

Celkové pojetí výše uvedených řídících sekvencí odpovídá tomu, jak k nim TEX přistupuje. TEX definuje speciální „nekonečné“ jednotky `‘fil’`, `‘fill’` a dokonce i `‘filll’`[7]. Zápis `‘\hfil’` nebo `‘\hfill’` je v jádru to samé jako

```
\hskip 0pt plus 1fil
\hskip 0pt plus 1fill
```

Stejným způsobem lze definovat v případě potřeby i sekvenci `‘\hfilll’`.

Další velice potřebná řídící sekvence se jmenuje `‘\hss’` (horizontal stretch or shrink). Funguje podobně jako `‘\hfil’`, ale dovoluje nejen nekonečné roztažení textu. ale i jeho nekonečné smršťování.[8]

```
\line{\hss Tohle bude uprostřed\hss}
```

Je v podstatě ekvivalentní zápisu

```
\line{\hskip 0pt plus 1fil minus 1fil
Tohle bude uprostřed
\hskip 0pt plus 1fil minus 1fil}
```

Poslední zajímavou řídící sekvencí pro obecnější formátování je `‘\hfilneg’`. Používá se poměrně zřídka, ale je velice praktická, protože dokáže zrušit účinky `‘\hfil’`. Její definice by mohla vypadat třeba takhle

```
\hskip 0pt plus -1fil
```

Opět platí, že téměř všechny operace, které jsou definovány na hboxech a horizontálních pružných mezerách, lze většinou provádět i na vboxy a vertikální pružné mezery. Většinou stačí nahradit písmenko ‘h’ písmenkem ‘v’ v názvu příkazu. Tedy `‘\vfil’` místo `‘\hfil’` atd.

Přehled dalších řídících sekvencí pro práci s mezerami

- `\noindent` - zakáže odsazení textu
- `\indent` - odsadí text (prázdný box velikosti `‘\parindent’`)
- `\quad` - vloží horizontální mezeru (glue) 1 em (rozdělitelnou)
- `\qquad` - vloží horizontální mezeru (glue) 2 em (rozdělitelnou)
- `\enskip` - vloží horizontální mezeru (glue) 0.5 em (rozdělitelnou)
- `\enspace` - vloží horizontální mezeru (kern) 0.5 em (nerozdělitelnou)
- `\thinspace` - vloží horizontální mezeru (kern) 0.16667 em (nerozdělitelnou)

Řádkování

Dosud jsme vystačili s jednoduchou představou, že TEX do sebe jenom skládá jednotlivé boxy a glue, které mu zadáme. To je trochu nepřesné. Ve skutečnosti je to o maličko složitější.

O čem to mluvím? Zkusme si představit stránku, kde by byly různě široké mezery mezi řádkami, podle toho, jestli je na dané řádce nějaké vysoké písmeno (jako `‘t’`, `‘A’` apod.) anebo není. Musí tu být ještě jednoduchý mechanismus, jak takovým věcem zabránit. Funguje celkem jednoduše. Když TEX zařazuje do vertikálního seznamu nový box, podívá se, jaká je vzdálenost mezi horním okrajem nového boxu a dolním okrajem boxu předchozího. Pokud je všechno v pořádku, nastaví mezeru mezi řádkami na velikost `‘\baselineskip’`. Pokud byla ale vzdálenost menší než `‘\lineskiplimit’`, přičte TEX ještě hodnotu `‘\lineskip’`.

[7] Všechny představují typ `‘\dimen’`.

[8] A to i do záporných velikostí

Mechanismus tedy není nijak složitý, přitom funguje velmi dobře. Zkusme trochu experimentovat! Můžeme například nastavit určité tolerance rozestupů mezi řádky, nebo řádkování naopak úplně zakázat příkazem `‘\offinterlineskip’` (nastaví nulovou meziřádkovou mezeru pro všechny následující boxy) či `‘\nointerlineskip’` (vloží následující box bez meziřádkové mezery). Návrat do původního stavu lze vynutit příkazem `‘\normalbaselines’`.

```
> \normalbaselines=macro:
->\lineskip=\normallineskip
->\baselineskip=\normalbaselineskip
->\lineskiplimit=\normallineskiplimit .
```

S mezerami mezi slovy je to o maličko složitější. TEX pro ně používá podobný mechanismus[9], ale navíc ještě rozlišuje dva režimy, které šířku mezer ovlivňují. První, `‘\frenchspacing’`, je jednodušší a zavádí všechny mezery jako sobě rovné. Druhý, sofistikovanější režim, je `‘\nonfrenchspacing’`. TEX se pak snaží rozlišovat jednotlivé typy mezer a vkládá větší či menší mezeru podle situace. Dělá větší mezeru za tečkou (pokud ovšem před ní nebylo velké písmeno indikující s největší pravděpodobností zkratku) apod.

Jemnosti

K opravdovým jemnostem patří i možnost velmi dobrého přístupu k jednotlivým písmenům (díky tomu, že je každé jednotlivé písmeno uloženo v boxu). Hodí se to především tam, kde je skvělý vzhled hlavní podmínkou (firemní logo, text na přebalu knihy apod.).

Každý typ boxu má své specifické řídící sekvence, kterými s ním lze jednoduše manipulovat. S hboxy se manipuluje příkazy `‘\raise’` a `‘\lower’` (upravují polohu vertikálně), s vboxy lze manipulovat příkazy `‘\moveleft’` a `‘\moveright’` (upravují polohu horizontálně). Další možností je využití primitiva `‘\kern’`. S jeho pomocí je například definováno logo TEXu:

```
> \TeX=macro:
->T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX.
```

Standardně jsou definovány také řídící sekvence, které umožňují překryv boxů. Jako příklad si můžeme uvést velice šikovně a elegantně definovaná makra `‘\rlap’` a `‘\llap’`

```
> \rlap=macro:
#1->\hbox to\z@ {#1\hss }.
```

a jeho obraz

```
> \llap=macro:
#1->\hbox to\z@ {\hss #1}.
```

Řídící sekvence `‘\z@’` je jednoduchá zkratka pro 0 a 0pt. Uveďme si jedno z možných použití:

```
Sesunutím dvou boxů dohromady se dá docílit pěkných efektů: o\llap{/}
```

dá výsledek

Sesunutím dvou boxů dohromady se dá docílit pěkných efektů: ∅

[9] Pomocí řídících sekvencí `‘\spaceskip’` a `‘\xspaceskip’`.

Formátování textu

Just as people get into different moods, TEX gets into different "modes." (Except that TEX is more predictable than people.)

D. E. Knuth

Jednou z hlavních starostí každého typografického systému je vizuální rozčlenění souvislého textu na jednotlivé strany a rozlámání jednotlivých řádků odstavců tak, aby odpovídaly předepsané velikosti. Podívejme se nejprve, jak TEX přistupuje k formátování odstavců a řádkovým zlomům.

Formátování odstavce

Charakteristickou úpravu odstavce vytváří TEX automaticky. Nejprve vloží nad začátek odstavce vertikální mezeru velikosti `'\parskip'`[1], odsadí první řádek o `'\parindent'`[2] a začne načítání textu odstavce. V okamžiku, kdy načte prázdný řádek nebo řídící sekvenci `'\par'`, je už jisté, že odstavec je již načten celý, a tak započne s jeho formátováním.

Odsazení prvního řádku odstavce lze potlačit příkazem `'\noindent'`, naopak vynutit odsazení lze příkazem `'\indent'`.

TEX se normálně snaží urovnat text tak, aby okraj vlevo i vpravo byl zarovnaný a aby vypadal co nejlépe. Pokud ovšem máme např. zapnutý větší font nebo pokud je v odstavci příliš málo míst vhodných k zalomení řádku[3], nebude TEX s to uspokojivě vyřešit vzniklou situaci a ohlásí chybu. Takové chybové hlášení by mohlo vypadat následovně:

```
Overfull \hbox (137.15346pt too wide) in paragraph at lines 2--0
[]\tenrm tohlejetakdlouhýkompaktnířádekažejejTeXnedokážerozumněupravit
```

TEX oznamuje přetečení textu (*overfull*) o 137.1546 bodu, což je o něco méně než jeden centimetr. Navíc přesahující řádek v textu označí černým obdélníčkem. Nastat může samozřejmě i opačný případ. Pak to vypadá nějak takhle

```
Underfull \hbox (badness 10000) detected at line 2
\tenrm příliš kratký řádek
```

Vzniklá situace se dá řešit opravdu různě. Především si ale musíme uvědomit, kde chyba vznikla. Většinu problémů vzniklých při formátování odstavců můžeme naštěstí snadno odstranit tím, že TEXu pomůžeme s rozdělováním slov, ukážeme mu vhodná místa k zalomení „neposlušných“ řádků apod.

Všimneme si především údaje *badness* v předchozí ukázce. Tento údaj představuje jakousi *míru násilí spáchaného na pružných mezerách*. Tuto „estetickou chybu“ si TEX průběžně počítá při sestavování odstavce a pokud je její hodnota příliš velká, informuje o tom uživatele.

[1] Implicitní nastavení je `'\parskip=0pt plus 1pt'`.

[2] Implicitně 20pt.

[3] Která to jsou, si ukážeme později.

Řádkový zlom

Hlavní roli v tom, jak budou řádkové zlomy nakonec vypadat, hraje právě právě hodnota *badness*. Každé potenciální zalomení řádku s sebou nese hodnotu zvanou *penalty*, která určuje, o jak vhodné místo pro lom řádku se jedná. Hodnota penalty významným způsobem ovlivňuje právě výpočet hodnoty badness.

Nejprve se podívejme, ve kterých místech je řádkový zlom povolen:

a) pružná mezera (pokud před ní není další taková, `‘\leaders’`[4], `‘\kern’`, `‘\penalty’`, mimo úsek v matematickém režimu)
b) `‘\kern’`, pokud je za ním `‘glue’` (mimo úsek v matematickém režimu)
c) `‘\penalty’` (i automaticky vložená)
d) u rozdělovníku slova

Hodnota penalty bude v případech (a), (b) nulová, v případě (c) je hodnota penalty příslušně upravena.

Vlastní algoritmus řádkového zlomu postupně probíhá v jednom až třech průchodech. TEX se snaží rozmístit slova v odstavci tak, aby celkové uspořádání vyhovovalo všem požadavkům, které na výsledný text klademe.

- V prvním průchodu TEX vyhodnotí všechny řádkové zlomy při nichž je
  (i) hodnota badness menší nebo rovna hodnotě `‘\pretolerance’`[5]
  (ii) a zároveň se nepoužije dělení slov.
  Pokud takové řešení existuje, zvolí se takové rozložení slov, které bude vypadat nejlépe a algoritmus pro řádkový zlom bude ukončen[6]. Pokud tyto požadavky nejsou splněny[7] pokračuje druhým průchodem.
- Ve druhém průchodu jsou vyhodnoceny všechny možné řádkové zlomy takové, že
  (i) všechny řádky mají požadovanou šířku
  (ii) a hodnota badness je zároveň menší nebo rovna velikosti `‘\tolerance’`[8].
  Dělení slov je už povoleno.

V tomto místě typicky algoritmus řádkového zlomu končí, výsledkem bude nejlepší dosavadní uspořádání odstavce[9]. Pokud však ani tady nemůže nalézt požadované řešení, pokračuje posledním třetím průchodem.

- Jak jsem již poznamenal, třetí průchod je málo častý a spíše „zachraňuje situaci“ V tomto průchodu TEX postupuje obdobně jako v průchodu druhém, ale ke každé mezeře připočte k původnímu natažení ještě hodnotu `‘\emergencystretch’`[10] (implicitně ovšem nastavena na 0pt!). Mezery se tedy mohou více roztáhnout a to ovlivní i výpočet badness. Je-li nová hodnota badness menší nebo rovna velikosti `‘\tolerance’`, výsledkem bude nejlepší dosavadní uspořádání odstavce a algoritmus řádkového zlomu se ukončí.

Po provedení řádkového zlomu dochází k případné další úpravě vstupního materiálu. Pokud TEX zalomí řádek v místě, kde byla pružná mezera, poslední element na řádku bude ten, který

[4] Podrobněji se touto řídící sekvencí budeme zabývat v poslední kapitole.

[5] V plainu je implicitně nastavena na hodnotu 100.

[6] Kvalita vzhledu se určuje z další tzv. demerits. Podrobný popis výpočtu: viz např. [TBN] na straně 228.

[7] Nastavením `‘\pretolerance’` až na hodnotu 10000 zajistíme to, že první průchod bude vždy úspěšný a TEX se ani nepokusí o rozdělování slov (ovšem mezery mezi slovy mohou vypadat příšerně). Naopak, nastavením `‘\pretolerance=-1’` vynutíme úplné přeskočení prvního průchodu, TEX se bude pokoušet slova rozdělovat okamžitě.

[8] Implicitní nastavení plain TEXu je 200.

[9] Vybírá se opět podle hodnoty demerits.

[10] Ta se v případném varovném hlášení ale nijak neprojevuje.

ji bezprostředně předcházel. Na novém řádku bude první element takový, který tuto pružnou mezeru následuje po přeskočení všech dalších po sobě jdoucích pružných mezer nebo jiných tzv. odstranitelných elementů[11].

Úpravy řádkového zlomu

Pokud nebylo dosaženo výsledku, TEX označí přesahující řádky značkou *overfull box* (černý obdélníček) a úpravu odstavce je potřeba udělat ručně.

Nejběžnějším a také nejjednodušším způsobem je změna hodnoty `‘\penalty’` (případ (c) přehledu) na nějakém vhodném místě. Pokud odstavec stále nedopadá podle našich představ můžeme příkaz změny penalty vložit na více míst a vyzkoušet, jak si to TEX přebere. Následující

```
... část odstavce před přijatelným zlomem \penalty 100 a za ním...
```

řekne TEXu, že by to bylo docela slušné místo, kde by se řádek mohl zlomit, ale hodnota 100 se započte. Nebo

```
... část odstavce před vítaným zlomem \penalty -200 a za ním...
```

udělí TEXu dokonce bonus, pokud řádek zlomí zrovna tam. Pokud chceme rovnou zakázat nebo naopak nařídit na nějakém místě řádkový zlom, stačí nastavit penalty na dostatečně vysokou hodnotu. Řídící sekvence `‘\nobreak’` resp. `‘\break’` nastavují penaltu na maximální hodnotu `‘\penalty=10000’`, resp. `‘\penalty=-10000’`.

```
> \nobreak=macro:
->\penalty \@M .
> \break=macro:
->\penalty -\@M .
```

Řídící sekvence `‘\break’` se v kombinaci s `‘\hfil’` obvykle také používá pro zalomení řádku, který nemusí zabírat plnou šířku odstavce[12].

```
\noindent Některé kratší řádky \hfil\break
nemusí zabírat \hfil\break
plnou šířku odstavce.
```

Některé kratší řádky
nemusí zabírat
plnou šířku odstavce.

Pokud tato technika nedostačuje, můžeme zkusit, zda potíže nedělá rozdělování některých slov a případně s tím TEXu trochu pomoci[13]. Pokud ani zde nebyl problém, nezbývá než se pokusit nějakým způsobem pozměnit nastavení řídících sekvencí, které ovlivňují chod algoritmu řádkového zlomu. Například:

```
\tolerance=1600
```

nastaví chování TEXu tak, aby toleroval řádky s hodnotou badness menší než 1600 (povolí tedy o trochu širší nebo kratší mezery mezi slovy než je standardní „dobře vypadající“ nastavení.

Dobrá, řekněme, že díky tomu se už ve zpracovávaném odstavci žádný řádek s větší hodnotou badness neobjeví. Z textu tak konečně zmizely černé obdélníčky značící přetečení. Přesto to ale nemusí znamenat, že už bude TEX úplně spokojen. Přetečení je jen jedním extrémem. Nesmíme ovšem zapomenout, že TEXu vadí i příliš málo textu na řádku (*underfull*). Pokud hodnota

[11] discardable elements

[12] Zkusme si uvědomit, jak by vypadal výstup následující ukázky, kdybychom před `‘\break’` nevložili příkaz `‘\hfil’`.

[13] Viz sekci „rozdělování slov“.

`'\badness'` některého z boxů překročí hodnotu speciální proměnné `'\hbadness'`[14], bude TEX hlásit chybu (i když jinou) i nadále.

Speciální případ nastane, pokud nastavíme `'\tolerance'` na hodnotu 10000. Na tak vysokou hodnotu TEX pohlíží, jako kdyby byla nekonečná. Proto při toleranci 10000 TEX nikdy nevyprodukuje *overfull box*, pokud nenastane nějaká speciální situace, jako například nerozdělitelné slovo delší než šířka odstavce. Podobným způsobem se takto můžeme zbavit i chybových hlášení o nedostatečném naplnění textem (*underfull*). To je ovšem metoda, která by se měla používat jenom v krajních případech! Takové nastavení akceptuje cokoliv, neodliší ještě přijatelné uspořádání textu od uspořádání absolutně nesmyslného. Zkusme nejprve nastavit `'\emergencystretch'` a zkusit ještě třetí průchod. Právě od toho tam přeci je!

Rozdělování slov

Ukažme si teď, jakými mechanismy můžeme ovlivňovat rozdělování slov. Nejjednodušší možností je jednoduše TEXu předepsat, jak má slovo správně rozdělit. Můžeme to udělat jednou na začátku dokumentu příkazem `'\hyphenation{roz-dělovník}'`[15] nebo pokaždé pokaždé (třeba jako výjimku) vepsáním řídícího znaku `'\-` přímo do daného slova (`'roz\-dělovník'`).

Obecněji lze rozdělení slova řídit vždy příkazem

`\discretionary{`⟨*pre-break text*⟩`}{`⟨*post-break text*⟩`}{`⟨*no-break text*⟩`}`

⟨*pre-break*⟩ udává část slova před rozdělovníkem
⟨*post-break*⟩ udává část slova za rozdělovníkem
⟨*no-break text*⟩ udává, jak by slovo vypadalo bez rozdělení

Dají se tak rozdělovat i některé speciální případy slov. Např. německé `'backen'` se dá rozdělit jako `'bak-ken'`

```
ba\discretionary{k-}{k}{ck}en
```

nebo si můžeme vypomoci makrem (pokud bychom to potřebovali častěji)

```
\def\ck/{\discretionary{k-}{k}{ck}}
```

a psát jen

```
ba\ck/en.
```

Rozdělení slova rozhodně není z estetického hlediska žádoucí. Proto TEX dělení slov penalizuje. Pokud je nutné nějaké slovo rozdělit, připočte při výpočtu penaltu velikosti `'\hyphenpenalty'` (pokud není část ⟨*pre-break text*⟩ prázdná), nebo `'\exhyphenpenalty'` (pokud je část ⟨*pre-break text*⟩ prázdná). Implicitně je `'\hyphenpenalty=50'` a `'\exhyphenpenalty=50'`.

Úprava formátování odstavce

O sestavování jednotlivých řádků už máme poměrně dobrou představu. Co jsem ale ještě nezmínil, je způsob, jakým TEX formátuje poslední řádek odstavce. Těsně předtím, než TEX určí všechny řádkové zlomy, udělá ještě dvě věci: (a) Pokud je poslední prvek horizontálního seznamu pružná mezera, TEX ji ze seznamu vyřadí. (b) Na konec aktuálního horizontálního seznamu odstavce vloží řídící sekvenci `'\penalty 10000'` (zakáže řádkový zlom) a příkazy `'\hskip\parfillskip'` (přidá závěrečnou mezeru) a `'\penalty -10000'` (vynutí závěrečný řádkový zlom) seznam ukončí. Implicitně má TEX nastavenou hodnotu `'\parfillskip=0pt plus1fill'`. Konec každého odstavce tedy může být vyplněn libovolným množstvím bílého místa.

[14] Implicitně je nastavena na 1000.

[15] `'\hyphenation'` rozšiřuje tabulku toho jazyka, který je právě aktivní.

Velké množství řídících sekvencí, které výsledný formát odstavce ovlivňují, není v tomto případě na škodu. Pouhá změna některých parametrů může zcela změnit vzhled odstavce. Přitom vlastní text odstavce nemusíme vůbec měnit. Například změnou parametrů `‘\parfillskip=0pt’`, `‘\parindent=0pt’` dosáhneme toho, že všechny řádky odstavce (včetně prvního a posledního) budou zarovnané vlevo i vpravo. Šířku odstavce můžeme o něco zúžit příkazem `‘\narrower’`, pokud to nestačí, můžeme přímo nastavit velikost místa, které je potřeba vynechat na levé nebo na pravé straně. To zařídí `‘\leftskip’` a `‘\rightskip’`. Přímé nastavení šířky odstavce lze docílit změnou hodnoty parametru `‘\hsize’`.

Pro lepší kontrolu vzhledu odstavce můžeme ještě zkusit experimentovat s nastavením parametru `‘\hfuzz’`. Pokud některý box přesahuje okraj odstavce o hodnotu menší nebo rovnou `‘\hfuzz’` jednoduše to ignoruje[16]. Plain TEX implicitně nastavuje tuto hodnotu `‘\hfuzz` na 0.1pt. Mějte ovšem na paměti, že nastavení tohoto parametru na vysokou hodnotu výrazně zvyšuje „zubatost“ pravého okraje textu.

Odstavcové speciality

V předchozí části jste si mohli prohlédnout, jak ovlivňovat vzhled odstavců pomocí jednotlivých parametrů, které TEX při formátování používá. Možností je ale ještě více. Velmi praktická je například dvojice řídících sekvencí `‘\hangindent’` a `‘\hangafter’`. `‘\hangafter`=⟨*číslo*⟩`’` udává, které řádky se budou odsazovat, `‘\hangindent`=⟨*velikost*⟩`’` doplní informaci, o kolik se budou odsazovat.

Označme si x resp. n hodnoty ‘`\hangindent`’ resp. ‘`\hangafter’`. Pak, jestliže $n \geq 0$, odsazení platí pro řádky $n+1$, $n+2$, … odstavce; jestliže $n < 0$, odsazení platí pro řádky 1, 2, …, n odstavce. Odsazením se rozumí, že příslušné řádky budou mít namísto `‘\hsize’` šířku ‘`\hsize`$-|x|$’. Pokud je $x \geq 0$, řádky se odsazují zleva; pokud je $x < 0$, odsazují se zprava.[17]

Podívejme se například, jak se vysází následující povídka od pana A. U. Thora:

```
\parindent=0pt\hangindent=4pc\hangafter=-3
Once upon a time, in a distant
galaxy called \"O\"o\c c,
there lived a computer
named R.~J. Drofnats.
Mr.~Drofnats---or ‘‘R. J.,’’ as
he preferred to be called---% error has been fixed!
was happiest when he was at work
typesetting beautiful documents.
```

> Once upon a time, in a distant galaxy called Ööç, there lived a computer named R. J. Drofnats. Mr. Drofnats—or “R. J.,” as he preferred to be called—was happiest when he was at work typesetting beautiful documents.

Libovolný tvar odstavce

Existuje ale i úplně obecná konstrukce, kterou lze vytvořit jakkoliv atypický odstavec. Řídící sekvence, která to zařídí, se jmenuje `‘\parshape’`. Jako parametr si bere počet dvojic čísel, které budou následovat. První číslo každé dvojice znamená, kolik místa je potřeba pro daný řádek vynechat, druhé pak, jaká bude délka řádku. Dají se tak vytvořit i opravdu zajímavé tvary…

```
\hbadness=10000 \parindent=0pt \parfillskip=0pt
```

[16] Udává tedy hodnotu, o kolik může nějaký box „beztrestně“ přesáhnout pravý okraj odstavce.

[17] Tento mechanismus používá TEX například pro řídící sekvence `‘\item’` a `‘\itemitem’`.

```
\parshape 7 6.3cm 3.3cm  5.8cm 4.4cm  5.4cm 5.2cm  5.0cm 6.0cm
           4.6cm 6.8cm  4.2cm 7.6cm  3.8cm 8.5cm

Once upon a time, in a distant galaxy called \"O\"o\c c,
there lived a computer named R.~J. Drofnats.
Mr.~Drofnats---or ``R. J.,'' as
he preferred to be called---% error has been fixed!
was happiest when he was at work typesetting beautiful documents.
```

Once upon a time,
in a distant galaxy called
Ööç, there lived a computer na-
med R. J. Drofnats. Mr. Drofnats—
or "R. J.," as he preferred to be called—
was happiest when he was at work typesetting
beautiful documents.

Seznamy a přehledy

Speciálním typem odstavců jsou i jednoduché seznamy a přehledy. Přestože stejného výsledku můžeme dosáhnout i při použití obecnější struktury – tabulky – má pro ně TEX navíc i jednoduché řídící sekvence.

```
\item{1.} Tohle bude první řádek přehledu. Odsazení si podrží až do konce
odstavce
    \itemitem{a)}
    První \uv{podpoložka}
    \itemitem{b)}
    Druhá \uv{podpoložka}

\item{2.}
Další bod...
```

1. Tohle bude první řádek přehledu. Odsazení si podrží až do konce odstavce
   a) První „podpoložka“
   b) Druhá „podpoložka“
2. Další bod…

Formátování stránek

Algoritmus pro stránkový zlom je obdobný algoritmu zajišťujícímu řádkový zlom, přičemž ale platí, že (především kvůli vysoké paměťové náročnosti použitých algoritmů) nelze propočítat všechny možné kombinace a pak vybrat tu nejlepší. Je tedy poněkud omezen na pouhé smršťování či roztahování mezer, případně ukončí sazbu odstavce a zbytek textu umístí do odstavce nového na následující straně.

Většinou to stačí, a pokud ne, můžeme příkazem `‘\eject’` vynutit okamžité ukončení stránky. Často je užívaná také konstrukce:

```
\vfill\eject
```

kde `‘\vfill’` nejprve vyplní zbytek stránky vertikální pružnou mezerou, `‘\eject’` pak stránku ukončí. Pokud ale potřebujeme stránku ukončit přímo uprostřed odstavce, ale jeho sazbu nechceme přerušit, je situace složitější. Uveďme si bez dalšího rozboru[18], jak toho lze docílit:

```
\vadjust{\eject}
```

[18] Podrobný popis tohoto gramatického pravidla `‘\vadjust’` lze nalézt například v [TBN] na straně 451.

Jinak můžeme pro formátování stránek používat skoro stejné řídící sekvence jako pro odstavce. Samozřejmě fungují i příkazy jako `‘\penalty’`, `‘\nobreak’`, `‘\smallbreak’` (penalty −50), `‘\medbreak’` (penalty −100), `‘\bigbreak’` (penalty −200).

Z nových řídících sekvencí jsou zajímavé hlavně `‘\goodbreak’` a `‘\filbreak’`. Příkaz `‘\goodbreak’` zařadí danou pozici na seznam míst, kde by bylo docela výhodné stránku ukončit. TEX z nich potom pokusí (podle situace) nějakou vybrat.

```
> \goodbreak=macro:
->\par \penalty -500 .
```

Zajímavé makro je i `‘\filbreak’`. Pokud v místě `‘\filbreak’` bude proveden stránkový zlom, zbylé místo se vyplní vertikální mezerou `‘\vfil’`. Jestliže se na stránku ještě vejde nějaký další text (ukončený například dalším `‘\filbreak’`), řídící sekvence se ignoruje.

I definice je udělaná velmi elegantně:

```
> \filbreak=macro:
->\par \vfil \penalty -200\vfilneg .
```

Což znamená: pokud se zalomí stránka podle `‘\penalty-200’`, `‘\vfil’` vyplní zbytek stránky bílým místem a řídící sekvence `‘\vfilneg’` už bude patřit na novou stránku (kde žádný účinek nemá); pokud stránkový zlom nenastane, `‘\vfilneg’` zruší účinek `‘\vfil’`.

Parametry vzhledu stránky

Vzhled stránky lze nastavit především následujícími parametry:

- `‘\hsize’` – celková šířka odstavce
- `‘\vsize’` – celková výška tiskového zrcadla
- `‘\topskip’` – automaticky vkládaná vertikální mezera před prvním elementem na stránce
- `‘\maxdepth’` – maximální hloubka posledního řádku na stránce
- `‘\hoffset’` – horizontální odsazení od levého okraje
- `‘\voffset’` – vertikální odsazení od horního okraje

Řídícími sekvencemi `‘\hsize’` a `‘\vsize’` se typicky nastavuje velikost stránky dokumentu, `‘\hoffset’` a `‘\voffset’` určují posun textu vůči referenčnímu bodu stránky, který je vždy umístěný palec[19] od levého a palec od horního okraje papíru. Kladné hodnoty `‘\hoffset’` resp. `‘\voffset’` posouvají text dokumentu[20] doprava resp. dolů. Podobně, pro záporné hodnoty se text dokumentu posouvá doleva resp. nahoru.

Na horní okraj stránky mají ještě vliv speciální parametry `‘\topskip’` a `‘\maxdepth’`. Dosti často je totiž třeba produkovat dvojice stran, jejichž horní či dolní účaří musí být v předem určených pozicích (například ve stejné výši). Z toho důvodu TEX před první box nebo linku na stránce umisťuje ještě speciální mezeru. Tato mezera je rovna parametru `‘\topskip’`, ovšem s tou výjimkou, že přirozená velikost mezery je zmenšena o výšku prvního boxu, případně je rovna nule, pokud by tato hodnota měla být záporná. Plain TEX nastavuje automaticky hodnotu `‘\topskip=10pt’`.

Řekněme například, že `‘\topskip=70pt plus 10pt minus 5pt’` a že první box na stránce má výšku `20 pt`. Potom TEX vloží přímo nad něj vertikální mezeru velikosti `‘\vskip 50pt plus 10pt minus 5pt’`.

Ze stejného důvodu je v TEXu zaveden i parametr `‘\maxdepth’`. TEX po každém vložení boxu nebo linky znovu kontroluje, zda nebyla hloubka vkládaného elementu větší než `‘\maxdepth’`[21]

[19] 2.54 cm

[20] Přesněji levý horní roh ukládaného boxu

[21] Informace o výšce strany se zjišťují z hodnot `‘\pagedepth’` a `‘\pagetotal’`.

Obrázky a speciální objekty

Papír snese všechno.

Cicero

TEXovský dokument často nebývá jen strohý text. Autoři do textu občas přidají nějakou ilustraci nebo poznámku. Přestože TEX v sobě nemá zabudovány žádné algoritmy pro práci s grafikou, máme několik možností, jak se se zařazením obrázku do textu vypořádat.

Většina ostatních DTP programů má obrázky na prvním místě a „nedůležitým“ textem se zabývá až později. Vložíme obrázek a program sám zařídí, aby text dokumentu kopíroval jeho geometrii (obtékání obrázků). V TEXu je na prvním místě text. Proto je potřeba zabývat se tím, jak text upravit, kde vynechat na obrázek místo a tak podobně.

Plovoucí objekty

Začneme tím jednodušším. Představme si, že potřebujeme do textu umístit nějaký obrázek, který se do dokumentu přidá až později (například ručně domaluje nebo vlepí).

Jednoduchá cesta, jak toho docílit je:

```
\topinsert <"výplň" ve vertikálním režimu> \endinsert
```

TEX se pokusí umístit „výplň“ na horní okraj stávající stránky. Pokud na této stránce už nezbývá žádné místo, umístí „výplň“ na stránku následující. Uvnitř „výplně“ může být i odstavec textu, který třeba doplňuje ilustraci.

```
\topinsert
\vskip 4cm    % posun 4 cm
\hsize=6cm    % šířka odstavce popisku
\raggedright  % povoluje zubatý pravý okraj
\noindent {\it Obrázek 1.\/} Tohle je krásný popisek k~prvnímu obrázku.
4~cm místa nahoře nechávají místo pro~uměleckou ilustraci.
\endinsert
```

účinek `‘\hsize’` a `‘\raggedright’` se automaticky ruší ukončením skupiny mezi řídícími sekvencemi `‘\topinsert’`, `‘\endinsert’`. Nakonec TEX ještě automaticky přidá pod obrázek mezeru `‘\bigskip’`. Velmi podobně se obsluhují i příkazy `‘\pageinsert’` a `‘\midinsert’`.

Chceme-li obrázek zařadit „někam do textu“, musíme změnit tvar příslušného odstavce. Pokud se jedná jen o to, zařadit obrázek do pravoúhlého výřezu zprava či zleva, můžeme s výhodou použít řídících sekvencí `‘\hanginsert’` a `‘\hangafter’`. Ve složitějších případech je nutno kombinovat práci s mezerami a boxy, případně použít řídící sekvenci `‘\parshape’`.

Pérovky

Jednoduché obrázky především geometrického typu, například geometrické obrazce, grafy funkcí, náčtrky apod., nazýváme pérovky. Při vytváření pérovek můžeme využít program původně určený především k návrhu a digitalizaci písem – METAFONT. Program METAFONT vytvořil rovněž D. E. Knuth a také se společně s TEXem distribuuje.

V obyčejném ASCII editoru vytvoříme soubor (typicky s příponou `‘.mf’`), ve kterém pomocí METAFONTových příkazů a maker obrázek popíšeme. Typicky se jeden obrázek definuje jako jeden znak nového fontu. Potom s pomocí METAFONTu celý font digitalizujeme do bitmapové

podoby a vytvoříme k němu metrické informace (tento krok obvykle zařídí ovladače automaticky).

S takovým obrázkem pak zacházíme jako s obyčejným boxem. To má své výhody i nevýhody. Obrázku můžeme například relativně snadno přidělit nějakou stálou pozici vůči poloze nějakého konkrétního odstavce, na druhou stranu přidělení nějakého stálého místa vzhledem k poloze papíru může být obtížné.

Další problém vznikne, pokud potřebujeme někam do obrázku umístit nějaké popisky. I to se dá naštěstí poměrně snadno zvládnout. Můžeme například využít toho, že TEX dovoluje, aby se boxy překrývaly. Obrázek (jedno písmeno fontu) vložíme do boxu a využijeme příkazu `‘\rlap’`, případně `‘\llap’`. S pomocí primitivních příkazů `‘\kern’`, `‘\lower’`, `‘\raise’`, `‘\moveleft’` a `‘\moveright’` pak popisek v boxu přesně umístíme.

Příklad:

```
\font\obr=obrazky  % Zaveď pro font s obrázky přezdívku ‘\obr’.
\newdimen\unit     % Definuj nový registr typu <dimen> pod názvem ‘\unit’.

\def\placedescription (#1;#2) [#3] {% Definujeme nové makro
                                    % dáváme pozor na zavlečené mezery
   \rlap{\kern#1\unit               % Využijeme překryvu boxů
         \lower#2\unit
   \hbox {\it #3\/}}}               % Popisek umístíme do boxu
                                    % a zapneme kurzívu.

\hbox{\unit=5mm%    Jedna jednotka bude 5mm
 \placedescription (0;10) [Popisek 1]% Umístíme popisek 1
 \placedescription (5;8) [Popisek 2]% Umístíme popisek 2
 \rlap{\obr\char0}%  Vložíme obrázek uložený pod znakem č. 0
}
```

Nejprve si tedy zavedeme font, ve kterém máme obrázky uloženy. Potom si zadefinujeme makro[1] `‘\placedescription’` se třemi parametry, které umí popisek kurzívou vysázet na specifikované souřadnice. Všimněme si, že jednotky těchto souřadnic si můžeme specifikovat až dodatečně. Zadáme souřadnice popisků a umístíme obrázek (se souřadnicemi jednotlivých popisků budeme muset nejspíš chvíli experimentovat, než je umístíme přesně tam, kam jsme chtěli).

Programů, které dokáží s METAFONTem spolupracovat je dokonce celá řada. Nemusíme se tedy omezovat na jednoduché ASCII editory, dokonce nemusíme ani příkazy METAFONTu znát. Programy jako `xfig`, TEXcad nebo např. `gnuplot`, vytvářejí METAFONTové soubory automaticky.

K popsání obrázku můžeme dokonce použít i příkazů samotného TEXu. Např. balík maker `mfpic` nabízí celou řadu příkazů, kterými lze takový jednoduchý obrázek snadno popsat. Zpracujeme-li takový dokument TEXem, vytvoří se zároveň i METAFONTový soubor s popisem obrázku a dál se o nic nemusíme starat. Soubory `mf` pak můžeme poslat ke zpracování spolu se zdrojovým textem dokumentu. Takové řešení má ještě tu výhodu, že je celý dokument i nadále nezávislý na dalším zpracování.

Může se ale stát, že narazíme na kapacitní limity samotného programu METAFONT. Ten byl původně určen čistě k vytváření fontů a s výrobou obrázků se při jeho tvorbě nepočítalo. Velikost jednoho znaku je METAFONTem omezena na 8192×8192 pixelů, tzn. při rozlišení 1300 dpi plochu přibližně 15 centimetrů.

[1] Podrobněji se definováním maker a registry budeme nabývat později.

Naštěstí to ale není tak velký problém. Obrázek je možno celkem jednoduše rozdělit do více znaků, ze kterých potom TEX sestaví výslednou kresbu. Práci si můžeme dále zjednodušit tím, že použijeme některý z balíků METAFONTových a TEXovských maker, které se problémem velkých obrázků a popisky zabývají.

Využití PostScriptu

Druhou zajímavou možností, jak dostat do textu obrázky, je využití jazyka PostScript. Přestože jazyk PostScript je o mnoho mladší než TEX, existuje poměrně hodně možností, jak spolupráci TEXu a PostScriptu zařídit. Nemusí jít zdaleka jen o vkládání obrázků. Pozdějším zpracováním dokumentu s využitím jazyka PostScriptu lze dosáhnout i různých modifikací textu, změny podkladu apod. Velmi dobrým příkladem toho, jak může vypadat spolupráce PostScriptu a TEXu je utilita `dvips` Toma Rokickima. Pomocí této utility a mechanismu tzv. virtuálních fontů TEXu, můžeme například využívat k sazbě některá z komerčních písem, od kterých máme k dispozici metrické údaje.

Velká část maker, ovladačů a utilit využívá pro podobné účely řídící sekvenci `'\special'`. Toto primitivum TEXu přenáší beze změn (jako znakový řetězec) svůj parametr přímo do výsledného `dvi` souboru bez dalšího zpracování TEXem. Ovladače pak mohou tyto „vzkazy“ nějak využít, nebo je mohou prostě ignorovat.

Encapsulated PostScript (zapouzdřený PostScript, obvyklá přípona `'.eps'`) dovoluje dokonce, aby řídící příkazy PostScriptu byly umístěny v nějakém vnějším prostředí, ve kterém se celý dokument vytváří. Podle normy, popis takovéhoto dokumentu nesmí obsahovat příkazy, které například manipulují s absolutní změnou měřítka, mění velikost objektu apod. Předpokládá se, že tyto parametry budou předem stanoveny „zvenku“ a nebudou se v průběhu zpracování měnit. Navíc takovýto objekt o sobě musí předávat informace, které jsou potřebné pro různé sázecí systémy. Z hlediska spolupráce s TEXem je takovýto systém vkládání obrazové informace téměř ideální. Celý postup samozřejmě předpokládá, že použijeme takový ovladač `dvi`, který má výstup do PostScriptu, např. `dvips`.

Makro epsf.tex

Na výše uvedeném principu pracuje systém maker `epsf.tex`, který navrhl Tom Rokick ve spolupráci s profesorem D. E. Knuthem.

Protože jazyk `eps` podporuje práci s barvou, nic nebrání tomu, aby v textu byly umístěny i barevné obrázky nebo například digitalizované fotografie apod. Nejprve je nutno obrázek převést do formátu `eps` (většina rozumných grafických programů naštěstí takový výstup podporuje), potom příkazem `'\input epsf'` nahrajeme systém maker pro práci s PostScriptem. Nyní už máme k dispozici všechny potřebné příkazy pro vkládání obrázků.

Nejdůležitější z nich jsou řídící sekvence `'\epsfbox'`, `'\epsfxsize'` a `'\epsfysize'`. S jejich pomocí už můžeme s obrázkem dostatečně manipulovat.

Příklad:

```
\epsfxsize=10cm
\epsfbox{obrázek.eps}
```

Vytvoří se box (na šířku) velikosti 10 centimetrů a obrázek se roztáhne tak, aby jej vyplnil celý. Výška boxu bude odpovídat tomu, jak velký musí být obrázek, aby v této ose nepodléhal žádné deformaci.

Tento postup má ještě jednu výhodu. Příslušný PostScriptový RIP[2] bývá totiž často vybaven speciálními rasterizačními a korekčními algoritmy s ohledem na vlastnosti zařízení, na kterém pracuje. Takový výstup pak většinou můžeme použít i pro profesionální účely.

[2] Raster Image Procesor

Speciální objekty

Může se stát, že krom běžného textu, formátovaného do odstavců, budeme potřebovat vložit i nějaký speciální objekt. Klasickou poznámku „pod čarou“ můžeme do dokumentu vložit řídící sekvencí `‘\footnote’`.[3]

```
...a do textu pak napíšeme nějakou
hezkou\footnote*{Záleží ovšem na vkusu} poznámku...
```

Záhlaví a pata stránky

K úpravě záhlaví a paty stránky se používají řídící sekvence[4] `‘\headline’` a `‘\footline’`. Jejich použití je poměrně intuitivní. Například

```
\headline={\centerline{\it Název knihy}}
```

nastaví záhlaví dokumentu tak, že bude na každé straně vycentrovaný název knihy vysázený kurzívou.

[3] Uvnitř popisků obrázků tahle RS použít nejde, ale můžeme ji nahradit ekvivalentem pro vertikální mód `‘\vfootnote’`.

[4] Registry typu `tokens`.

Mathematicians do it right!

Caution: This chapter is rather long one. Why don't you stop reading now, and come back fresh tomorrow?

D. E. Knuth

Konečně se dostávám k tomu nejlepšímu, co nám TEX může nabídnout. V TEXu jdou sázet matematické formulky s perfektním výsledkem a navíc jednodušeji než všechno dosud.

Matematický mód

Do matematického módu i zpět se přepnete pomocí znaku `'$'` nebo pomocí `'$$'`. Příkaz `'$$'` navíc oproti `'$'` přeruší sazbu odstavce, udělá dostatečné mezery pro odlišení formule od ostatního textu, vycentruje ji a pak pokračuje sázením odstavce.

První, čeho si v matematickém módu všimnete, bude nejspíš fakt, že TEX ignoruje většinu mezer, které jste do formule vložili. Matematické texty a formulky mají svojí typickou úpravu, kterou TEX udělá jednoduše za vás. Máte samozřejmě možnost změn, pokud se vám výstup nelíbil, ale většinou to není potřeba.

V matematických formulích se sázejí proměnné a neznámé kurzívou, číslice, různé funkce apod. antikvou a zbytek už bývají jen různé druhy matematických symbolů. Výsledný text je pak velice slušně čitelný a přitom snadno odlišitelný od okolního textu.

Na psaní matematických symbolů existuje v TEXu velké množství speciálních řídících sekvencí. Většinou jde o normální anglické názvy (nebo jednoduché zkratky) pro symbol, který chceme napsat. Aby se předešlo nepříjemným chybám ve výstupu, nepovoluje TEX většinou speciální příkazy pro matematiku mimo matematický mód. Příklad:

```
$$ x \neq y $$                    x ≠ y
$$ a \approx b + (x-1)$$          a ≈ b + (x − 1)
```

Pokud bychom např. na posledním řádku minulého příkladu zapomněli zapnout matematický mód, TEX ohlásí chybu:

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   \approx
l.1911 a \approx
                b + (x-1)
```

Speciální znaky

Pro matematický mód zbyly kromě `'$'` ještě dva rezervované znaky. Znak `'^'` se používá pro horní index, znak `'_'` pro dolní index. Oba indexy jdou samozřejmě používat zároveň. Na jejich pořadí nezáleží. Pokud použijeme více stejných indexů zároveň a není jasný význam, musíme jednotlivé indexy rozdělit do skupin (tzn. uzavřít do složených závorek). Zápis

```
$ x^i^j $
```

by mohl znamenat

```
$ x^{i^j} $                       x^{i^j}
```

a nebo klidně také

```
$ x^{ij} $
```
x^{ij}

V takovém případě nás TEX upozorní na nekorektnost v zápisu a bude použití závorek vyžadovat. Většinou se ale vyplatí ozávorkovat i formule, kdy TEX použití závorek explicitně nevyžaduje.

```
$ {({(x^2)}^3)}^4 $
```
$({(x^2)}^3)}^4$

vypadá ve výsledku lépe než jenom

```
$ ((x^2)^3)^4 $
```
$((x^2)^3)^4$

Pokud potřebujeme u nějakého symbolu horní i dolní index zároveň, může se nám stát, že TEX neumístí indexy zcela symetricky. Pokud se nám to nelíbí a chceme oba indexy symetricky mít, můžeme TEX přesvědčit, aby oba indexy zarovnal podle pomyslné skupiny znaků.

```
$ P^{1}_{2} $
```
P^{1}_{2}

```
$ P{}^{1}_{2} $
```
$P{}^{1}_{2}$

Někdy je dobré prázdnou skupinu `'{}'` použít, aby nedošlo k nedorozumění. Typická situace vzniká, pokud potřebujeme udělat index před, nikoliv za symbolem.

```
$ a\ {}_1b_2 $
```
$a\ {}_1b_2$

Přitom kromě normálních čísel a písmen můžeme používat jako indexy i různé háčky, obloučky, hvězdičky, vlnovky, puntíky apod.

```
$ y^{\prime\prime} $
```
$y^{\prime\prime}$

nebo a můžeme také psát

```
$ y'' $
```
y''

Složitější vzorce

Složitější vzorce vypadají sice poměrně nepřehledně, ale jejich psaní problémy nedělá. Syntax výrazů je naštěstí tak jednoduchá a logická, že i složité formule většinou napíšeme na první pokus.

```
$\sqrt2$
```
$\sqrt2$

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

```
$\underline{x-y}$
```
$\underline{x-y}$

```
$\overline x+ \overline y$
```
$\overline x+ \overline y$

```
$x^{\overline{y+m}}$
```
$x^{\overline{y+m}}$

```
$\sqrt{x^3+\sqrt{\alpha+b}}$
```
$\sqrt{x^3+\sqrt{\alpha+b}}$

Na posledním řádku příkladu je hezky vidět jedna maličkost. Totiž, že TEX automaticky zjišťuje velikost objektu pod odmocninami (a podobnými symboly). To je jistě užitečné, na druhou stranu výstup nemusí vždycky vypadat podle očekávaní. Je vidět, že

```
$\sqrt{\alpha}+\sqrt{b}+\sqrt{y}$
```
$\sqrt{\alpha}+\sqrt{b}+\sqrt{y}$

velikost odmocnin na řádku bude různá. Aby všechny odmocniny byly stejně vysoké, můžeme použít pomocnou řídící sekvenci `'\mathstrut'`[1]. Výška `'\mathstruct'` se vztahuje k největšímu objektu v použité sadě znaků. Nejčastěji to bývá kulatá závorka.

```
$\sqrt{\mathstrut \alpha} +
 \sqrt{\mathstrut b} +
 \sqrt{\mathstrut y}$
```
$\sqrt{\mathstrut \alpha} +
 \sqrt{\mathstrut b} +
 \sqrt{\mathstrut y}$

[1] Což není nic jiného než **hbox** s nulovou šířkou a s pevně danou výškou a hloubkou

Přehled důležitých matematických symbolů

TEX disponuje obrovským množstvím různých matematických symbolů nejrůznějších druhů. Jejich použití je ale velice jednoduché. Přepis většiny TEXovských matematických formulí totiž odpovídá velmi často tomu, jak by se taková matematická formule vyslovovala anglicky.

Řecká písmena

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

Velmi často je potřeba v matematickém módu odlišit určitý symbol přidáním nějakého znaménka nebo značky. Můžeme např. použít některý z matematických akcentů[2].

Matematické akcenty.

$\hat{o}$	`\hat o`	$\check{o}$	`\check o`	$\tilde{o}$	`\tilde o`
$\acute{o}$	`\acute o`	$\grave{o}$	`\grave o`	$\dot{o}$	`\dot o`
$\ddot{o}$	`\ddot o`	$\breve{o}$	`\breve o`	$\bar{o}$	`\bar o`
$\vec{o}$	`\vec o`	$\widehat{abc}$	`\widehat {abc}`	$\widetilde{abc}$	`\widetilde {abc}`

Poznámka: Aby se daly znaky ‘`i`’ a ‘`j`’ dobře označovat akcenty, má TEX zavedené speciální řídící sekvence ‘`\imath`’ a ‘`\jmath`’, které definují ‘$\imath$’ a ‘$\jmath$’ bez tečky nad znakem.

Velmi časté jsou i tzv. relační operátory. Následující tabulka obsahuje řídící sekvence pro některé z nejpoužívanějších. Jejich negaci lze většinou získat předřazením ‘`\not`’ těsně před relační operátor.

Nejpoužívanější symboly pro relace

$\leq$	`\leq`	$\not\leq$	`\not\leq`	$\geq$	`\geq`	$\not\geq$	`\not\geq`
$\equiv$	`\equiv`	$\not\equiv$	`\not\equiv`	$\sim$	`\sim`	$\not\sim$	`\not\sim`
$\simeq$	`\simeq`	$\not\simeq$	`\not\simeq`	$\approx$	`\approx`	$\not\approx$	`\not\approx`
$\subset$	`\subset`	$\not\subset$	`\not\subset`	$\subseteq$	`\subseteq`	$\not\subseteq$	`\not\subseteq`
$\supset$	`\supset`	$\not\supset$	`\not\supset`	$\supseteq$	`\supseteq`	$\not\supseteq$	`\not\supseteq`
$\in$	`\in`	$\notin$	`\not\in`	$\ni$	`\ni`	$\not\ni$	`\not\ni`
$\parallel$	`\parallel`	$\nparallel$	`\not\parallel`	$\perp$	`\perp`	$\not\perp$	`\not\perp`

Bezesporu nejběžnějším typem operátorů jsou binární operátory. Pokud na ně TEX narazí někde uprostřed matematického módu, postará se sám o přidání malé mezery na obě strany binárního operátoru.

Nejpoužívanější binární operátory

$\cdot$	`\cdot`	$\times$	`\times`	$*$	`\ast`	$\star$	`\star`
$\circ$	`\circ`	$\bullet$	`\bullet`	$\div$	`\div`	$\diamond$	`\diamond`

[2] Řídící sekvence používané v normálním textu nejdou v matematickém módu používat a naopak.

$\cap$	`\cap`	$\cup$	`\cup`	$\vee$	`\vee`	$\wedge$	`\wedge`
$\oplus$	`\oplus`	$\ominus$	`\ominus`	$\otimes$	`\otimes`	$\odot$	`\odot`

Krom operátorů „normální“ velikosti existuje v TEXu ještě třída „velkých“ operátorů. TEX tyto operátory nabízí ve dvou odlišných velikostech: pro matematický styl uprostřed textu i pro speciální matematický styl (display mode).

„Velké“ operátory

$\sum\ \sum$ `\sum`	$\prod\ \prod$ `\prod`	$\coprod\ \coprod$ `\coprod`	$\bigodot\ \bigodot$ `\bigodot`
$\int\ \int$ `\int`	$\bigvee\ \bigvee$ `\bigvee`	$\bigwedge\ \bigwedge$ `\bigwedge`	$\bigoplus\ \bigoplus$ `\bigoplus`
$\oint\ \oint$ `\oint`	$\bigcup\ \bigcup$ `\bigcup`	$\bigcap\ \bigcap$ `\bigcap`	$\bigotimes\ \bigotimes$ `\bigotimes`

Stále ještě zbývá poměrně veliké množství znaků a symbolů, které nepatří do žádné z předchozích kategorií. Přesto se řada z nich poměrně hodně používá. V následující tabulce jsou některé z nich.

Různé

$\aleph$	`\aleph`	ℓ	`\ell`	$\Re$	`\Re`	$\Im$	`\Im`
∂	`\partial`	∞	`\infty`	$\|$	`|`	$\angle$	`\angle`
∇	`\nabla`	$\backslash$	`\backslash`	$\forall$	`\forall`	$\exists$	`\exists`
$\neg$	`\neg`	$\flat$	`\flat`	$\sharp$	`\sharp`	$\natural$	`\natural`

Další symboly

Ani psaní složitějších symbolických zápisů, zjevně nedělá v TEXu žádné potíže. Jednoduše vypadají i zápisy zlomků, binomických nebo multinomických koeficientů, sum, integrálů ...

```
$$\prod_{0\leq i<100}(Z_i^3)$$
```
$$\prod_{0\leq i<100}(Z_i^3)$$

```
$${p\choose2}x^2y{p-2}-{1\over{1-x}}$$
```
$${p\choose2}x^2y{p-2}-{1\over{1-x}}$$

```
$$\int_0^{\pi/2}{1\over \sin^n{x}} dx$$
```
$$\int_0^{\pi/2}{1\over \sin^n{x}} dx$$

```
$$a\over{b+1\over b-1}$$
```
$$a\over{b+1\over b-1}$$

Přesto se občas vyplatí zamyslet a podívat se na to, co právě píšeme. Například poslední formule předchozího příkladu by patrně lépe vypadala takhle:

```
$$a\over{(b+1)/(b-1)}$$
```
$$a\over{(b+1)/(b-1)}$$

Kromě normálních zlomků a kombinačních čísel můžeme používat i doplňující řídící sekvence jako `‘\above’` (s délkovým parametrem) a `‘\atop’`. Příkaz `‘\atop’` funguje stejně jako `‘\over’`, ale vynechává zlomkovou čáru. Řídící sekvence `‘\above’` ji naopak udělá tak tlustou, jakou zadáme velikost.

Z dalších užitečných řídících sekvencí si ukažme například `‘\scriptstyle’` a `‘\displaystyle’`. Jimi se jednoduše mění velikost použitého fontu podle matematického stylu.

```
$$\sum_{\scriptstyle0\le i\le m
\atop\scriptstyle0<j<n}P(i,j)$$
```
$$\sum_{\scriptstyle0\le i\le m\atop\scriptstyle0<j<n}P(i,j)$$

U některých velkých symbolů, jako jsou sumační znaménko nebo integrál, můžeme ovlivnit, jak se budou zobrazovat horní a dolní indexy. K tomu účelu zavádí TEX řídící sekvence ‘\limits’ a ‘\nolimits’.

```
$$ \sum\limits_{n-1}^m $$
```

$$\sum_{n-1}^{m}$$

```
$$ \sum\nolimits_{n-1}^m $$
```

$$\sum\nolimits_{n-1}^{m}$$

Symboly s proměnlivou velikostí (delimiters)

Další skupinou znaků v TEXu jsou matematické symboly s proměnlivou velikostí tzv. delimiters (omezovače).

Symboly s proměnlivou velikostí (delimiters)

(	‘(’	)	‘)’	[	‘[’, ‘\lbrack’
]	‘]’, ‘\rbrack’	{	‘\{’, ‘\lbrace’	}	‘\}’, ‘\rbrace’
⌊	‘\lfloor’	⌋	‘\rfloor’	⌈	‘\lceil’
⌉	‘\rceil’	<	‘<’, ‘\langle’	>	‘>’, ‘\rangle’
/	‘/’	|	‘|’, ‘\vert’	‖	‘\\|’, ‘\Vert’
↑	‘\uparrow’	⇑	‘\Uparrow’	↓	‘\downarrow’
⇓	‘\Downarrow’	↕	‘\updownarrow’	⇕	‘\Updownarrow’

a speciální (používají se ve velkých velikostech za ‘\big’ a ‘\Big’)

⏐	\Big\arrowvert	‖	\Big\Arrowvert	⎪	\Big\bracevert
⎧	\Big\lgroup	⎫	\Big\rgroup	⎰	\Big\lmoustache
⎱	\Big\rmoustache				

Všechny tyto symboly se automaticky zvětšují, pokud před nimi použijeme řídící sekvence ‘\left’ nebo ‘\right’. Ty dohlédnou na to, aby omezovač byl vždycky dostatečně velký vůči formuli uvnitř.

TEX ovšem vyžaduje, aby příkazy ‘\left’ a ‘\right’ byly zadávány v párech. Jako nepárové můžeme použít řídící sekvence ‘\big’, ‘\bigl’, ‘\bigr’, ‘\biggl’ a ‘\biggr’ (seřazeno podle velikosti výstupu). Případně můžeme použít i ‘\bigm’ a ‘\biggm’, které jsou navrženy pro neobvyklé použití *uvnitř* výrazu. Navíc pro každou tuto řídící sekvenci existují ještě o polovinu větší alternativy ‘\Big’, ‘\Bigl’, ‘\Bigr’, ‘\Bigm’ o 2.5 krát větší ‘\Biggl’, ‘\Biggr’ a ‘\Biggm’ (vztaženo k ‘\bigl’, ‘\bigr’ a ‘\bigm’).

Vhodné použití těchto operátorů může formuli velice zpřehlednit:

```
$$C = \Bigl(x\in A(n)\bigm\Vert x\in B(n)\Bigr)$$
```

$$C = \Bigl(x \in A(n) \bigm\| x \in B(n)\Bigr)$$

```
$$\biggl({\partial^2 F\over\partial x\partial y}-
{\partial^2F\over\partial y\partial x}\biggr) = 0$$
```

$$\biggl(\frac{\partial^2 F}{\partial x \partial y} - \frac{\partial^2 F}{\partial y \partial x}\biggr) = 0$$

Matematické funkce

V matematických textech se často objevují různé funkce. Ve výrazech jako např. $\sin 2x = 2\sin x \cos x$ jsou trigonometrické funkce 'sin' a 'cos' vysázeny antikvou, nikoliv kurzívou.

Speciální funkce

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

Styly pro matematiku

TEX musí mít samozřejmě systém, který v matematickém módu udržuje správnou velikost indexů, určuje, kdy se použije který font a hlídá, co zrovna povoleno je a co není. TEX disponuje celkem čtyřmi základními režimy (styly).

– **display style** (pro matematické formule na samostatné řádce)
– **text style** (pro matematické formule uvnitř v textu)
– **script style** (pro horní a dolní indexy)
– **scriptscript style** (pro horní a dolní indexy druhého řádu (indexy indexů))

Každý styl má svoje specifické vlastnosti, především velikost použitých fontů. O přepínání stylů se TEX stará automaticky, ale pokud potřebujeme, můžeme si příslušný styl zapnout vlastnoručně. Jde to dokonce i uprostřed rozdělané matematické formule. Řídící sekvence pro přepnutí se jmenují stejně, jako příslušný styl (tzn. '`\displaystyle`', '`\textstyle`', '`\scriptstyle`', '`\scriptscriptstyle`'.

Tipy

Pište formule pokud možno jednoduše. Je zbytečné psát

```
$ y = {\rm arccos}(x) $
```
$y = \arccos(x)$

když můžeme použít už hotové makro

```
$ y = \arccos (x) $
```
$y = \arccos(x)$

Dejte pozor i na menší záludnosti. TEX má například dvě různé řídící sekvence pro zápis dělení se zbytkem (operace modulo). Řídící sekvence '`\pmod`' se používá jako binární operátor, kdežto '`\bmod`' se používá pro zápis Gaußových kongruencí. Pokud se použijeme špatnou verzi, výsledek může být velmi matoucí.

```
$ \gcd(m,n) = \gcd(n,m \bmod n) $
```
$\gcd(m,n) = \gcd(n,m \bmod n)$

```
$ x \equiv y+1 \pmod{m^2} $
```
$x \equiv y+1 \pmod{m^2}$

Nemusíme se bát ani používání používání tučných fontů, kaligrafických znaků, speciálních symbolů, závorek nad a pod výrazy, případně podtrhávání, pokud to text náležitě zpřehlední a zjednodušší.

```
$ \alpha \cdot {\bf y} = \bf (x + z)$
```
$\alpha \cdot {\bf y} = \bf (x + z)$

```
$\cal A,\ B,\ C$
```
$\cal A,\ B,\ C$

```
$\ell = 1 + x$
```
$\ell = 1 + x$

```
$\underbrace{x+\cdots+x}
 _{k\rm\;krát}$
```
$\underbrace{x+\cdots+x}_{k\rm\;krát}$

```
$\underline{\limsup{x}}$
```
$\underline{\limsup x}$

Používat můžeme i většinu řídících sekvencí pro práci s pružnými mezerami a boxy.

```
$$ F_n=F_{n-1}+F_{n-2},
 \qquad n\ge2. $$
```
$$F_n = F_{n-1} + F_{n-2}, \qquad n \ge 2.$$

Speciální řídící znaky

Pro jemné dolaďování vzhledu můžeme použít i různé řídící znaky pro mezery

`\,` (1/6 quad)
`\>` (2/9 quad)
`\;` (5/18 quad)
`\!` (-1/6 quad)

Jejich aplikace vypadá typicky takhle:

`$(2n)!/\bigl(n!\,(n+1)!\bigr)$`	$(2n)!/(n!\,(n+1)!)$
`$\sqrt2\,x$`	$\sqrt{2}\,x$
`$\sqrt{\,\log x}$`	$\sqrt{\log x}$
`$O\bigl(1/\sqrt n\,\bigr)$`	$O(1/\sqrt{n}\,)$
`$[\,0,1)$`	$[\,0,1)$
`$\log n\,(\log\log n)^2$`	$\log n\,(\log\log n)^2$
`$x^2\!/2$`	$x^2/2$
`$n/\!\log n$`	$n/\log n$
`$\Gamma_{\!2}+\Delta^{\!2}$`	$\Gamma_2+\Delta^2$
`$R_i{}^j{}_{\!kl}$`	$R_i{}^j{}_{kl}$
`$\int_0^x\!\int_0^y dF(u,v)$`	$\int_0^x\!\int_0^y dF(u,v)$

Výčtové typy

Ani s psaním složitějších struktur nejspíš nebudeme mít v TEXu problémy. Většinu úprav za nás udělají předdefinované řídící sekvence.

`$M = \{\,x\mid x>5\,\}$` $M = \{\,x \mid x > 5\,\}$

Pro složitější strukturu můžeme použít příkaz `‘\cases’`, kde znak `‘&’` slouží jako zarážka (tabulátor).

```
$$ |x| =\cases{
      x,& pokud je $x \ge 0$;\cr
     -x,& jinak.\cr}$$
```

$$|x| = \begin{cases} x, & \text{pokud je } x \ge 0; \\ -x, & \text{jinak.} \end{cases}$$

Matice

Velmi podobně se zapisují i matice. O jejich vytváření se starají řídící sekvence `‘\matrix’`, `‘\pmatrix’` (s kulatými závorkami okolo) a `‘\bordermatrix’` (s kulatými závorkami a s dalšími údaji okolo závorek matice).

```
$$A=\left|
\matrix{4-\lambda  &1          &-1 \cr
        0          &8-\lambda &-1 \cr
        -2          &0          &5-\lambda\cr}
\right|.$$
```

$$A = \begin{vmatrix} 4-\lambda & 1 & -1 \\ 0 & 8-\lambda & -1 \\ -2 & 0 & 5-\lambda \end{vmatrix}.$$

Výrazy

Je docela příjemné, že jsou v TEXu definovány i řídící sekvence pro práci s výrazy. Nejsou sice nezbytné, ale pokud o nich víme, může nám to ušetřit spoustu práce a času.

```
$$X_n=X_k\qquad\hbox{právě tehdy když}\qquad
  Y_n=Y_k\quad\hbox{a}\quad Z_n = Z_k.$$
```

$X_n = X_k$ právě tehdy když $Y_n = Y_k$ a $Z_n = Z_k$.

S pomocí řídících sekvencí ‘`\eqno`’ a ‘`\leqno`’ se dají výrazy i jednoduše číslovat. ‘`\eqno`’ opatří formuli číslem na pravé straně, ‘`\leqno`’ na levé. Syntax je následující:

```
$$ <výraz> \eqno <výraz>$$
```

Příklad:

```
$$ y = 2x^7 - 2x^5 + 3x^4 -15x^2 +16x - 5 \eqno(1) $$
```

$$y = 2x^7 - 2x^5 + 3x^4 - 15x^2 + 16x - 5 \tag{1}$$

Víceřádkové (ne)rovnice je možné zarovnat podle ‘`=`’ (‘`\eq`’) nebo ‘`\neq`’. K tomuto účelu slouží ‘`\eqalign`’. Místo zarovnání je označeno znakem ‘`&`’.

```
$$\eqalign{X_1+\cdots+X_p &=m, \cr
           Y_1+\cdots+Y_p &=n \cr}$$
```

$$\begin{aligned} X_1+\cdots+X_p &= m,\\ Y_1+\cdots+Y_p &= n \end{aligned}$$

Samozřejmě si můžeme nechat očíslovat i jednotlivé řádky. Můžeme si dokonce vybrat, na kterou stranu chceme číslování umístit. ‘`\eqalignno`’ umisťuje číslování klasicky na pravou stranu výrazu, ‘`\leqalignno`’ na levou. Syntax je pro obě řídící sekvence stejná.

```
$$\eqalignno{(x+y)(x-y) &= x^2 -xy +xy -y^2\cr
                        &= x^2 - y^2;&(2)\cr
     (x+y)^2&= x^2 + 2xy +y^2.&(3)\cr}$$
```

$$\begin{aligned} (x+y)(x-y) &= x^2 - xy + xy - y^2\\ &= x^2 - y^2; && (2)\\ (x+y)^2 &= x^2 + 2xy + y^2. && (3) \end{aligned}$$

Pro práci se složitými formulemi lze použít řídící sekvenci ‘`\displaylines`’. Složitý výraz rozepíšeme jako několik nezávislých jednodušších formulí, které pak s pomocí ‘`\displaylines`’ spojíme dohromady.

```
$$\displaylines{⟨matematická formule_1⟩\cr
                ⟨matematická formule_2⟩\cr
                    ⋮
                ⟨matematická formule_n⟩\cr}$$
```

Makra

Circular reference: See also Reference, circular.
Reference, circular: See also Circular reference.

The New Hacker's Dictionary

Makra jsou velice výkonnou, ale také poměrně spletitou částí TEXu. Pojednává o nich několik velice dobrých a poměrně obsáhlých knih. Nemá smysl, abych se snažil směstnat tak rozsáhlé téma do několika málo stránek. Zmíním jen to nejdůležitější nebo to, co se používá hodně často.

Definice

Základní řídící sekvence pro definování nových maker je `'\def'`. Definice mají tvar

`\def`⟨*RS*⟩⟨*text parametrů*⟩`{`⟨*text pro nahrazení*⟩`}`

⟨*text parametrů*⟩ nesmí obsahovat žádné závorky (`'{'`, `'}'`) a ⟨*text pro nahrazení*⟩ musí mít všechny závorky správně vnořeny do sebe. Parametry nejsou povinnou součástí definice maker.

Příklad:

```
\def\proclaim #1. #2\par{\medbreak
  \noindent{\bf#1.\enspace}{\sl#2}\par\medbreak}
```

volání makra pak vypadá[1]:

```
\proclaim Lemma 1. \TeX\ má výkonná makra. \par
```

Symboly `'#`⟨*číslo*⟩`'` v definici definují parametry makra. Makro může mít až 9 parametrů. ⟨*text pro nahrazení*⟩ makra se použije až tehdy, když volání makra plně splňuje definici makra. V příkladu nahoře bude onen text za volání makra považován pouze v případě, že za `'\proclaim'` následuje nějaký text, pak tečka, mezera, nějaký další text a `'\par'` (neboli prázdný řádek). Teprve potom bude příslušná část substituována za ⟨*text pro nahrazení*⟩. Znak `'#'` musí být v definici makra vždy následován číslem parametru a nebo dalším znakem `'#'` (pokud chceme dostat znak `'#'` na výstup tak, jak je).

Trik s tokeny

Následující definice makra bez parametrů demostruje hezký trik.

```
\def\ck/{\discretionary{k-}{k}{ck}}
```

Makro využívá toho, že tokeny, které jsou v ⟨*textu parametrů*⟩, budou muset následovat po definovaném názvu makra. De facto se stávají součástí názvu makra. Můžeme tedy psát

```
ba\ck/en
```

Takto definované makro má tu výhodu, že není při jeho použití slovo opticky zcela rozděleno mezerou a text odstavce bude o něco přehlednější.

[1] Namísto sekvence `'\par'` budete jistě používat prázdný řádek pro optické oddělení od znění lemmatu...

Dlouhá makra

V textech, které se ukládají do parametru makra, nesmí být přítomná sekvence `‘\par’`. TEX považuje makra s parametry s délkou větší než odstavec implicitně za chybná (neefektivně používají zdroje a paměť TEXu). Pokud budeme chtít takové makro opravdu vytvořit, nezbývá než použít pomocnou řídící sekvenci `‘\long’`. Následující makro

```
\long\def\bold#1{{\bf#1}}
```

dovoluje sázet celé odstavce tučným písmem.

Lepším řešením je použití řídících sekvencí `‘\begingroup’` resp. `‘\endgroup’`. Tyto speciální řídící sekvence označují začátek resp. konec skupiny podobně jako složené závorky. Na rozdíl od složených závorek, ale TEX nevyžaduje, aby párovaly. Výše uvedené makro, tedy může být nahrazeno dvojicí,

```
\def\beginbold{\begingroup\bf}
\def\endbold{\endgroup}
```

které pracují daleko efektivněji. Poznamenejme, že ovšem platí dál, že se skupiny nesmí překrývat. Zápis

```
{ \begingroup } \endgroup
```

samozřejmě není korektní.

Působnost

Všechny definice jsou automaticky považovány za lokální. Tzn. platí pouze v aktuální a ve všech vnořených skupinách. Toto chování lze změnit předřazením `‘\global’` v definici makra, případně lze psát zkratku `‘\gdef’`.

```
\def\A{A1}
\def\B{B1}
{\def\A{A2}
  {\def\B{B2}
   \gdef\MAC{\A\ \B}
   \MAC}}
\MAC
```

Makro `‘\MAC’` ve vnitřní skupině expanduje podle očekávání na `‘A2 B2’`. Ve vnější skupině ovšem mají makra `‘\A’` a `‘\B’` jiný význam. Proto poslední vyvolání makra `‘\MAC’` dává výsledek `‘A1 B1’`, nikoliv `‘A2 B2’`.

Tento příklad je velice jednoduchý. Přesto výsledek možná nebyl vidět na první pohled. Při použití složitějších maker může být chování takového systému maker nejasné, ne-li nerozlušťitelné. Proto se doporučuje používat podobných konstrukcí uvážlivě.

Přiřazování řídících sekvencí

Občas potřebujeme volat totéž makro vícekrát pod různými jmény. Prosté definování toho samého makra několikrát, když se liší jen jménem, ale není příliš elegantní. Možné řešení by bylo vložit volání jednoho makra do ostatních. To může mít svoje nevýhody. Např. můžeme narazit na nepříjemné problémy při předávání parametrů.

Vzniklá situace se dá naštěstí snadno vyřešit za pomoci řídící sekvence `‘\let’`. Ta provede jednoduše přiřazení jiného identifikátoru už existujícímu makru nebo řídící sekvenci. Můžeme pomocí ní udělat i trik přehození významu řídících sekvencí.

```
\let\a=\b \let\b=\c \let\c=\a
```

Proměnné

Možnosti TEXu by byly značně omezené, kdyby nebyly k dispozici žádné proměnné. Samozřejmě, že jsou. Základní proměnné mohou být čtyř typů:

```
\count<číslo>  = <číslo>
\dimen<číslo>  = <dimen>
\skip<číslo>   = <glue>
\muskip<číslo> = <muglue>
```

Každý typ definuje 256 registrů číslovaných 0–255. Pokud parametr není `<číslo>`, musí u něj být vždy uvedeny jednotky (cm, pt, cc, ...). Čísla 0–9 bývají vyhrazena pro interní makra nebo je TEX používá např. pro číslování stránek. Existují ještě dva speciální typy registrů[2]:

```
\toks<číslo> = <posloupnost tokenů>
\box<číslo> = <box>
```

přičemž token je buď řídící sekvence nebo nějaký znak opatřený informací o kategorii.

Do registru typu tokens můžeme uložit libovolnou posloupnost tokenů. Existují dokonce speciální pojmenované registry tohoto typu. Jako příklad uveďme registr `‘\everypar’`, jehož obsah je vložen do vstupní fronty při každém vstupu do odstavcového režimu.

Druhým speciálním typem jsou registry pro ukládání obsahu boxů. Na rozdíl od ostatních registrů do registrů typu box můžeme zapisovat pouze příkazem `‘\setbox’`. Pro práci těmito registry se používají převážně příkazy `‘\box<number>’` (vloží box a globálně vyprázdní obsah registru) a `‘\copy<number>’` (vloží box, obsah registru ponechá)[3].

Základní operace s proměnnými

TEX definuje i jednoduché aritmetické operace. Rozeberme následující:

```
\advance\count0 by \count1
\advance\count1 by -10
\multiply\count0 by \count2
\divide\count0 by 6
```

k hodnotě registru `‘\count0’` se přičte jednička, hodnota registru `‘\count1’` se sníží o 10, do registru `‘\count0’` se uloží násobek jeho původní hodnoty a hodnoty registru `‘\count2’`, poslední operace hodnotu registru `‘\count0’` šestkrát zmenší.

Alokace registrů

Tento způsob práce, jen s číselnými registry, není příliš příjemný. Z toho důvodu je v plain TEXu definovaný mechanismus alokace registrů. Registry si můžeme nejen pojmenovat, ale zároveň i snadno zjistit, jestli někde nevzniká nějaká kolize a tím se vyhnout zbytečným chybám.

Prosté pojmenování registru lze udělat následovně:

```
\countdef\chapternumber=28
```

vytvoří přezdívku `‘\chapternumber’` pro registr `‘\count28’`.

[2] Podrobně o registrech viz např. [TBN] str. 72.

[3] Podrobně o registrech typu box viz např. [TBN] str. 82

Používanější jsou ovšem příkazy ‘`\newcount`’, ‘`\newdimen`’, ‘`\newskip`’, ‘`\newmuskip`’ a ‘`\newtoks`’, které zajišťují automatickou alokaci registru daného typu. Příkaz přiřadí některý z volných registrů a nás nezajímá, který to byl. Pokud je budeme používat ve svých makrech, máme jistotu, že se makra o registry nebudou „tlouct“. Takhle definovaným proměnným pak můžeme přiřazovat libovolné hodnoty daného typu, dělat aritmetické výpočty apod. Například

```
\newdimen\halfhsize
\halfhsize=0.5\hsize
```

uloží do nově vytvořené proměnné typu ‘`dimen`’, polovinu hodnoty proměnné ‘`\hsize`’.

Čísla a číselné soustavy

TEX podporuje i zápis v jiných číselných soustavách. Čísla lze zadávat:

Klasicky	`24`
	`765.7865`
Hexadecimálně	`"C2` (odpovídá `0xc2` v jazyku C)
Oktalově	`'34` (odpovídá `034` v jazyku C)

Tabulky

Keyboard error:
Press F1 to continue.

PC BIOS

Většina běžných dokumentů obsahuje kromě normálního textu často i různé vsuvky, přehledy apod. Typicky se takovéto věci umisťují do tabulek. TEX disponuje poměrně obecnými a výkonnými prostředky, kterými lze této specifické úpravy dosáhnout.

Zarážky (tabulátory)

Nejjednodušší je nastavit zarážky (odrážky, tabulátory) a nechat TEX, ať text tabulky na stránce rovnoměrně rozvrhne. Každá řídící sekvence `'\+'` začíná nový řádek tabulky, každá zarážka (tabulátor) znamená nový sloupec, sekvence `'\cr'` řádek ukončuje.

```
\settabs 4 \columns
\+&&Text začíná ve třetím sloupci\cr
\+&Pak ve druhém\cr
\+&&opět ve třetím sloupci\cr
\+V prvním sloupci, &&&ve čtvrtém\cr
\+&&&&a za ním\cr
```

Text začíná ve třetím sloupci
Pak ve druhém
opět ve třetím sloupci
V prvním sloupci, ve čtvrtém
a za ním

Rozložení textu v tabulce lze samozřejmě dále upravovat. Můžeme k tomu používat většinu řídících sekvencí, které jsme mohli vidět v předchozích kapitolách.

```
\settabs 2 \columns
\+\hfill Text zarovnaný vpravo&
  \hfill Centrovaný text \hfill&\cr
\+\hfill na~první polovině.&
  \hfill na~druhé polovině.\hfill&\cr
```

Text zarovnaný vpravo — Centrovaný text
na první polovině. — na druhé polovině.

Vždycky se nehodí mít všechny sloupce stejně široké. Pokud si chceme nastavit šířky sloupců po svém, můžeme použít vzorovou řádku. Stačí za řídící sekvenci `'\settabs'`, která nastavuje šířky jednotlivých sloupců, přidat ještě `'\+'` a napsat vzor.

```
\settabs\+indent& Max. délka 1.sloupce\quad& Délka 2.sloupce\cr % vzor
\+&Nářadí -- Krumpáč &Strana 15\cr
\+&Nářadí -- Hoblík  &Strana 18\cr
\+&Nářadí -- Svěrák  &Strana 19\cr
```

Nářadí – Krumpáč Strana 15
Nářadí – Hoblík Strana 18
Nářadí – Svěrák Strana 19

Opravdové tabulky

Pro opravdové tabulky se nejčastěji používá řídící sekvence ‘`\halign`’. Její syntax se od syntaxe řídícího znaku ‘`\+`’ příliš neliší.

```
\halign{\indent#\hfil&\quad#\hfil\cr
Nářadí -- Krumpáč &Strana 15\cr
Nářadí -- Hoblík  &Strana 18\cr
Nářadí -- Svěrák  &Strana 19\cr}
```

Nářadí – Krumpáč Strana 15
Nářadí – Hoblík Strana 18
Nářadí – Svěrák Strana 19

První řádek určuje budoucí vzhled tabulky. Definuje vynechávané místo a umístění textu v něm (symbol ‘`#`’). V tomto příkladu obsahuje dvě šablony (pro první a druhý sloupec). Všimněme si, že díky mechanismu šablon není potřeba žádného vzorového řádku.

Příkaz ‘`\halign`’, na rozdíl od ‘`\+`’, zjišťuje maximální délku sloupce automaticky (nemusíme tedy vědět maximální délku sloupců a pracně odhadovat vzorový řádek). Naproti tomu, ‘`\+`’ si pamatuje nastavení zarážek tak dlouho, dokud nejsou explicitně změněny. Mezi jednotlivými částmi tabulky vytvořené s pomocí ‘`\+`’ tedy může být libovolné množství odstavců, dokonce i libovolně mnoho řídících sekvencí ‘`\halign`’.

Triky

I při tak nudné činnosti, jakou je vypisování tabulek, se dá přijít na malý zlepšovák.

```
\halign to \hsize{\indent Nářadí -- #\hfil&\quad Strana 1#\cr
Krumpáč &5\cr
Hoblík  &8\cr
Svěrák  &9\cr}
```

Nářadí – Krumpáč Strana 15
Nářadí – Hoblík Strana 18
Nářadí – Svěrák Strana 19

Vtip je v tom, že si do formátovací řádky umístíme i část textu, která se v tabulce na všech řádcích opakuje. Velikost tabulky určuje konstrukce ‘`\halign to\hsize`’. Problém byl byl, kdybychom museli v takhle napsané tabulce najednou nějaký údaj pozměnit. Zdá se, že by nezbývalo nic jiného, než celou tabulku přepsat. Ale naštěstí to není pravda. Pokud se opravdu změní jen několik řádek, můžeme pro korekci použít řídící sekvenci ‘`\omit`’. Ta nahradí šablonu pro daný řádek triviálním ‘`#`’.

Lepší formátování tabulek

Protože tabulky vždycky nemusí být takhle jednoduché, podporuje TEX chytrý způsob, jak do tabulky vložit nějaký jiný prvek. Stačí na příslušné místo napsat konstrukci:

`\noalign{`⟨*něco ve vertikálním režimu*⟩`}`

⟨*něco ve vertikálním režimu*⟩ může být opravdu skoro cokoliv. Do tabulky se tak dají vkládat čáry, mezery, vboxy nebo dokonce i další tabulky…

Vodiče (leaders)

Vodiče se často používají nejenom v tabulkách, ale i v různých přehledech, v obsahu knih a podobně. Zápis pro vodič je jednoduchý:

`\leaders` ⟨*box nebo glue*⟩`\hskip`⟨*glue*⟩

Vodič se bude chovat podobně jako **‘\hskip’**, ale místo glue se použije výplň z ⟨*box nebo glue*⟩. Typické použití vypadá nějak takhle:

```
\def\leaderfill{\leaders\hbox to 0.7em{\hss.\hss}\hfill}
\vbox{ \line{Kapitola desátá\leaderfill str. 128}
\line{Začátek\leaderfill Konec} }
```

Kapitola desátá . . . . . str. 128
Začátek . . . . . Konec

Linky

TEX rozlišuje linky dvou typů. Obě mají téměř stejný zápis:

```
\h(v)rule width<dimen> height<dimen> depth<dimen>
```

Parametry `width`, `height` ani `depth` nejsou povinné.

Pokud budeme linkami oddělovat jen kusy souvislého textu, nebudeme mít nejspíš žádné problémy. Pokud budeme chtít linky používat i uvnitř složitějších kusů (uvnitř tabulek apod.), musíme dát trochu pozor. První, co musíme udělat, když chceme tabulky s linkami, je vypnout automatické vkládání mezer (glue) mezi řádky v tabulce. To zařídí řídící sekvence **‘\offinterlineskip’**. Pak vložíme do šablony **‘\vrule’**. Když potřebujeme omezit délku čáry tak, aby byla stejně dlouhá jako formátovaný text, musíme navíc všechno vložit do boxu.

A na úplný závěr jedna lahůdka v podobě hrozivých čísel o naší populaci:

```
\def\BC{\hbox to4em{ \bf př.n.l\hss}}%
\def\AD{\hbox to4em{ \bf n.l.\hss}}%
\vbox{\offinterlineskip
\hrule
\halign{&\vrule#&
  \strut\quad\hfil#\quad\cr
height2pt&\omit&&\omit&\cr
&Rok\hfil&&Světová populace&\cr
height2pt&\omit&&\omit&\cr
\noalign{\hrule}
height2pt&\omit&&\omit&\cr
&8000\BC&&5,000,000&\cr
&50\AD&&200,000,000&\cr
&1650\AD&&500,000,000&\cr
&1850\AD&&1,000,000,000&\cr
&1945\AD&&2,300,000,000&\cr
&1980\AD&&4,400,000,000&\cr
height2pt&\omit&&\omit&\cr}
\hrule}
```

Rok	Světová populace
8000 **př.n.l**	5,000,000
50 **n.l.**	200,000,000
1650 **n.l.**	500,000,000
1850 **n.l.**	1,000,000,000
1945 **n.l.**	2,300,000,000
1980 **n.l.**	4,400,000,000