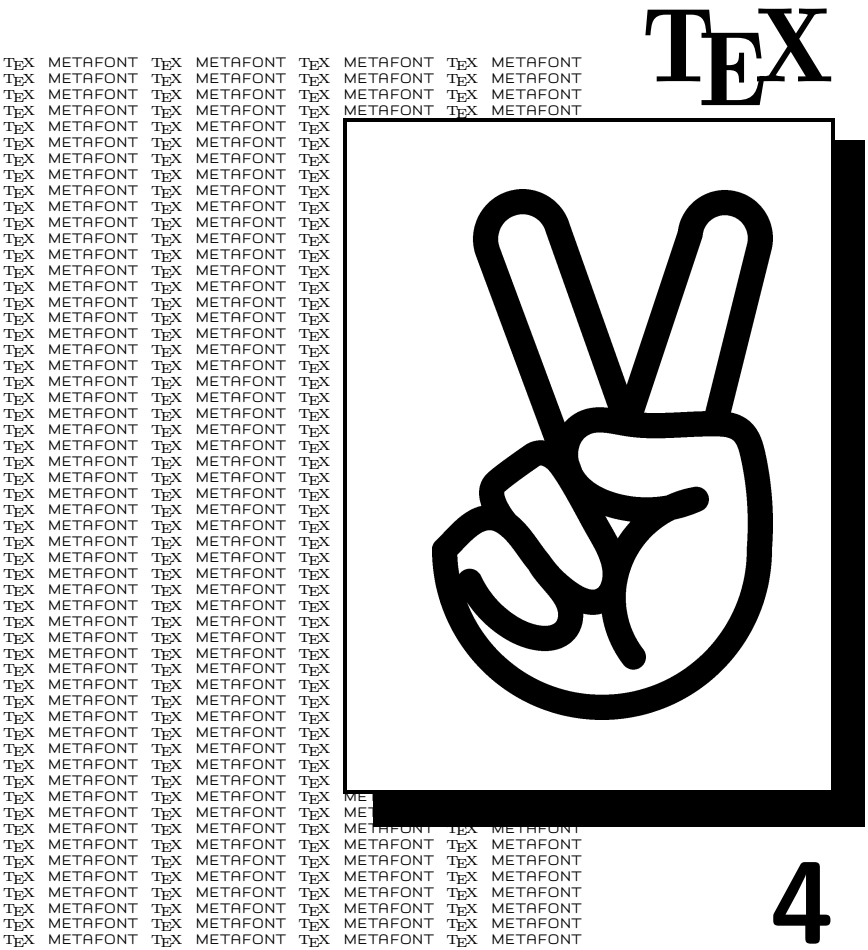


4

ZPRAVODAJ 93

ČESKOSLOVENSKÉHO SDRUŽENÍ UŽIVATELŮ TEXU

OBSAH

Grafika v TEXu (1)

Petr Sojka

Úvod

V dnešní době je stále větší pozornost kladena na přehlednost a instruktivnost vědeckých prací, knih, dokumentace; jak praví přísloví – jeden obrázek vydá za tisíc slov. Vznikají softwarové systémy specializované na vytváření grafiky (Adobe Illustrator 4.0, Aldus FreeHand 3.1, CorelDraw! 3.0, Graphics Editor 3.11, Fractal Design Painter 1.2 for Windows, Micrografix Designer 3.1, PhotoFinish 1.0, PhotoStyler 1.1a, Picture Publisher 3.1, Publisher's Paintbrush 2.02)[1]) a také většina dalších softwarových balíků (Mathematica, Lotus 1-2-3, Statgraphics, ...) umožňuje výstup prezentační grafiky v některém z grafických elektronických formátů pro další použití.

Některé výtky na adresu systému TEX se týkají integrace grafiky do textu dokumentu. Proto v tomto a v dalším pokračování TEXové filipiky budeme věnovat pozornost integraci grafiky do dokumentu sázeného TEXem a také související problematice jazyka většiny kvalitních výstupních zařízení Postscript.

Filosofie a počátky TEXu se datují do konce 70. let, kdy jsme v ČSFR začínali tisknout české znaky na řádkových tiskárnách přetiskem z apostrofů a kdy i ve světě myšlenka přímé integrace grafiky a textu byla téměř kacířská. TEX byl koncipován v době, kdy nebyly rozšířeny osobní počítače a pracovní stanice s grafickými displeji, laserové tiskárny, ani standardy jako Postscript či SGML. Přes své stáří TEX spolu s programem METAFONT a dalšími podpůrnými programy tvoří ucelený (a v mnoha aspektech nepřekonaný) softwarový balík pro vysoce kvalitní publikační činnost.

Obrázky do TEXového dokumentu můžeme integrovat několika způsoby:

1. pomocí balíků maker přímo v TEXu či (LATEXu),
2. pomocí programu METAFONT,

Upravená verze série tří autorových článků psaných pro Zpravodaj ÚVT MU Brno.

[1]) Více o těchto systémech viz např. časopis Bajt 8/1992.

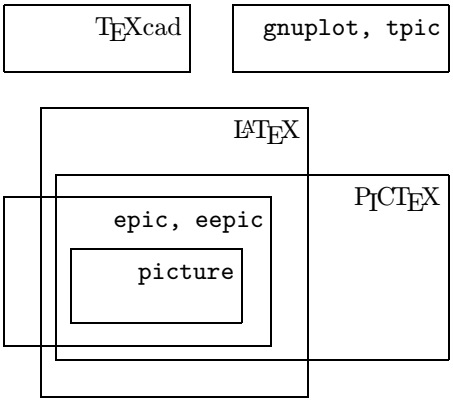
3. integrací grafiky vytvořené specializovanými grafickými systémy.

1. Grafika pomocí maker v TEXu

Základní sada písem TEXu (rodina písem Computer Modern) obsahuje sadu znaků pro sazbu čar (různých délek a konečného počtu sklonů) a kruhů a kružnic různého poloměru. Z této omezené sady znaků („stavebních kostek“) lze skládat obrázky pomocí poměrně složitých maker.

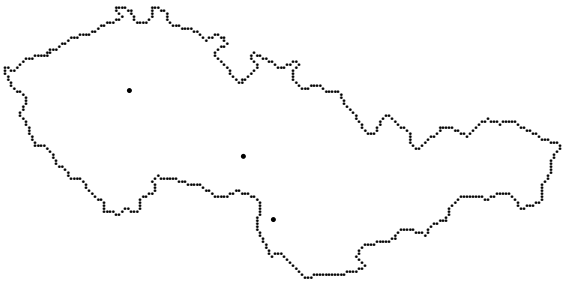
Výhodou tohoto způsobu je dokonalá přenositelnost a jednoduchost a to, že nevyžaduje žádné další programové prostředky. Nevýhodou je vysoká pracnost při ručním popisu obrázku, příp. náročnost na kapacitní možnosti TEXu či ovladačů (náročné jsou zejména části obrázku s bodovou grafikou).

LATEX. Přímo ve standardním LATEXu je okolí `picture`, které umožňuje skládat obrázek umisťováním jednotlivých elementů na uživatelem definované souřadnice. Například obrázek 1 (viz níže) vznikl tak, že uživatel musel zadat explicitně pomocí souřadnic každý obdélník.



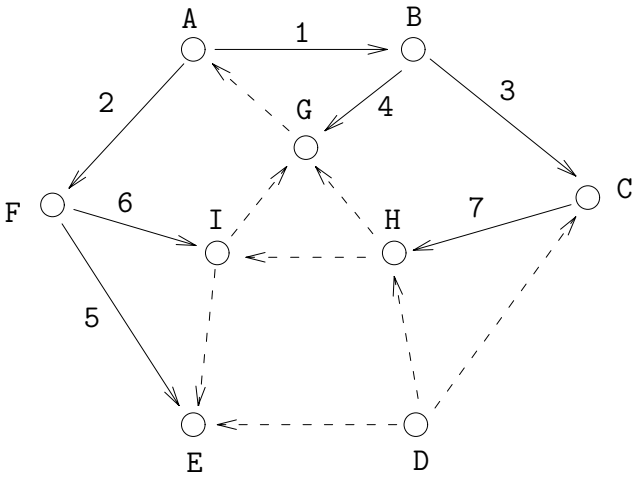
Obr. 1. Grafika v TEXu

Jako jiný (spíše netypický a archaický) příklad slouží mapa ČSFR (obrázek č. 2), která vznikla pomocí maker imitujících pohyb pera na milimetrovém papíře.



Obr. 2. Mapa ČSFR

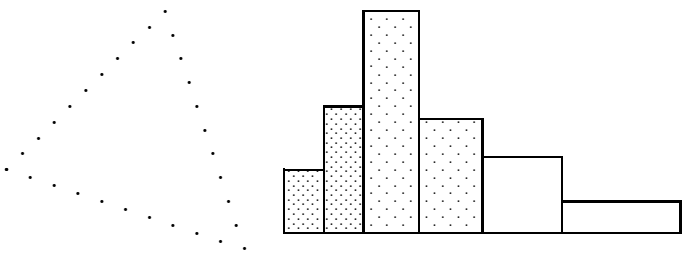
Makra `epic`, `eepic`. Omezení standardního okolí `picture` na kresbu jen základních objektů, na počet směrů čar a kapacitu paměti podmínila vznik nové sady maker, definující makra na sazbu širší škály objektů (různých typů čar, mřížek – makra `epic`) a umožňující lepší využití dostupných prostředků, zejména možností laserových tiskáren (makra `eepic`). Ukázka možností maker je na obrázku 3.



Obr. 3. Příklad použití maker `eepic`

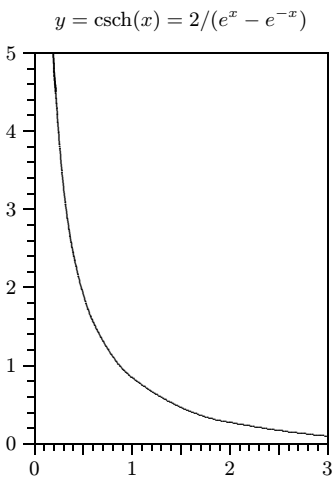
PICTEX. PICTEX je sada maker (*public domain*, pro plainTEX i LATEX) pro vytváření obrázků jako histogramy, grafy funkcí se souřadným systémem, diagramy obsahující kružnice, elipsy apod. Makra umožňují celkem jednoduše popsat obrázek pomocí zavedení souřadného systému a (dvourozměrných) objektů do něj umisťovaných. Pro vytvoření čtenářovy představy o makrech a syntaxi zápisu uveďme příklad zadání obrázku 4 v PICTEXu:

```
\beginpicture
\setcoordinatesystem units <1mm,1mm>
\multiput {.} at 0 0
       *10 2 2  *10 1 -3  *10 -3 1 /
\setcoordinatesystem units <1mm,8mm>
                      point at -35 1
\setshadegrid span <0.6mm>
\shaderectangleson \sethistograms
\plot 0 0  5 1.0  10 2.0 /
\setshadegrid span <1mm>
\plot 10 0  17 3.5  25 1.8 /
\shaderectanglesoff
\plot 25 0  35 1.2  50 0.5 /
\endpicture
```



Obr. 4. Ukázky obrázků vytvořených PICTEXem

Jiný obrázek vytvořený pouze pomocí maker PICTEXu je na obr. 5. Tištěný manuál PICTEXu je možno objednat na adrese TUGu. Docela



Obr. 5. Graf funkce vytvořený PICTEXem

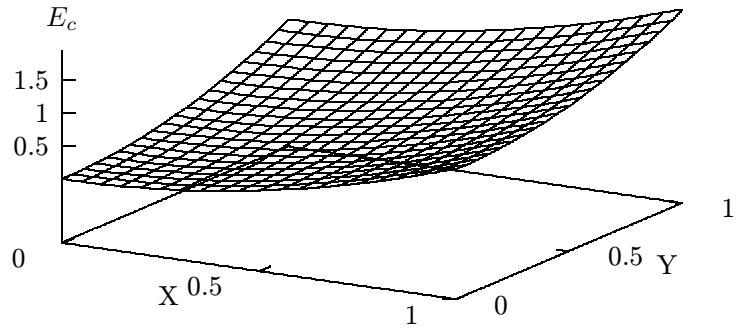
podrobný popis maker je možno nalézt v knize [1] (kapitola o více než 70 stranách). Makra jsou dostupná v elektronickém archivu ÚVT MU.[2])

Preprocesory. Protože syntaxe zápisu v okolí `picture` či makry PICTEXu je poměrně složitá, vzniklo několik preprocesorů, které umožňují (polo)automatické generování obrázků do těchto maker. Příkladem takových preprocesorů jsou programy TEXcad a `gnuplot`.

Program TEXcad. TEXcad je samostatný program, který pomocí nabídek ovládaných myší vytváří na obrazovce obrázek (z obdélníků, kružnic, ...). Výstupem tohoto programu je kód v TEXu (okolí `picture` LATEXu), který se vloží (editorem) do TEXového dokumentu. TEXcad běží pod operačním systémem MS DOS a naleznete jej jako součást standardní distribuce emTEXu.

Program gnuplot. Dalším preprocesorem umožňujícím výstup ve formátu dat pro TEX (též okolí `picture` – viz obr. 6) je program `gnuplot`.

[2]) Pomocí anonymního `ftp` na počítač `ftp.ics.muni.cs`.



Obr. 6. Příklad výstupu programu `gnuplot`

Je přenositelný a použitelný jak pod operačním systémem MS DOS, tak i pod systémem UNIX. Distribuční balík programu verze 3.2 naleznete také v elektronickém archivu ÚVT MU.

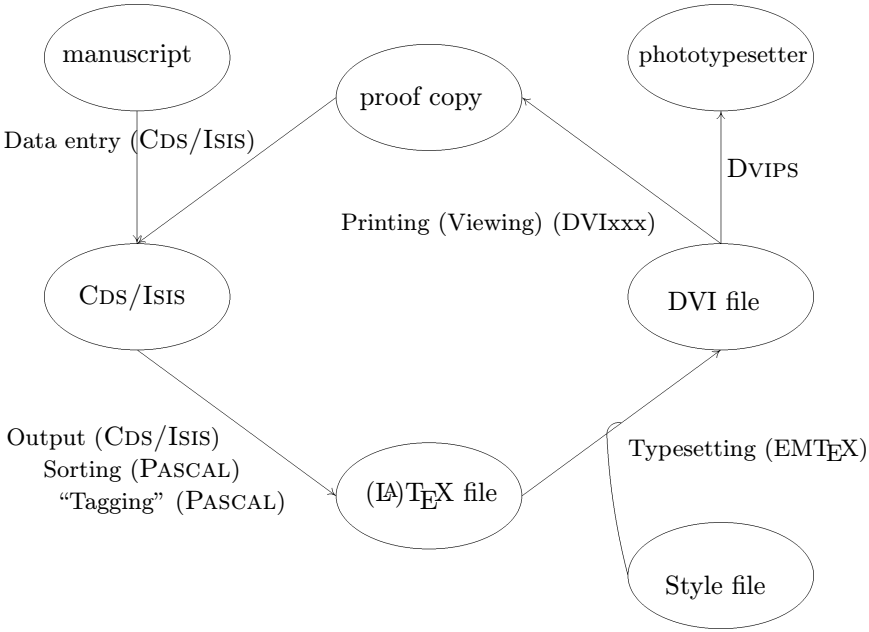
Následující část je věnována ukázce možností dalších balíků maker a zejména možnostem METAFONTu.

2. Grafika vytvářená programem METAFONT

alší prostředky, jak vytvářet grafiku (zejména nové znaky, fonty, loga či algebraicky snadno definovatelné křivky), poskytuje program METAFONT. Autorem tohoto programu není nikdo jiný než autor TEXu Don Knuth. Knuth použil METAFONT k vytvoření sad písem Computer Modern (písma běžně užívaná s TEXem, jimiž je také sázen tento Zpravodaj) a Concrete Roman (viz textové fonty odstavce s pohádkou v tomto článku). Znaky jsou definovány ve speciálním jazyce se syntaxí a částečně filosofií (makroprocesor) podobnou TEXu. Pro ilustraci uveďme kódy znaků ✓ a ✔.

$u\# := {}^1\!/_{10}pt\#;$

beginchar$(33, 100u\#, 100u\#, 0);$
$z_0 = (50u, 50u);\ \ z_1 = (15u, 35u);$
$z_2 = (25u, 7u);\ \ z_3 = (52u, 50u);$
$z_4 = (90u, 97u);$



Příklad obrázku vytvořeného pomocí maker `mfpic`

$z_{10} = (18u, 25u)$; $z_{20} = (25.5u, 4.5u)$;
pickup pencircle
xscaled $17.5u$ yscaled $10u$ rotated 30;
draw z_1 .. z_{10} .. z_2;
pickup pencircle xscaled $10u$ yscaled $3u$;
draw$(z_{20}$ .. z_3 .. $z_4)$;
endchar;

beginchar(34, $100u\#$, $100u\#$, 0);
$z_0 = (50u, 50u)$; $z_1 = (15u, 50u)$;
$z_2 = (18u, 20u)$;
$z_3 = (55u, 50u)$; $z_4 = (90u, 90u)$;
$z_{10} = (15u, 45u)$; $z_{20} = (19.5u, 20u)$;
$z_{21} = (25u, 25u)$; $z_{40} = (87.5u, 88u)$;

pickup pencircle xscaled $27.5u$
yscaled $17.5u$ rotated 45;
draw $z_1 \,..\, z_{10} \,..\, z_2$;
pickup pencircle xscaled $18.5u$
yscaled $7.5u$ rotated 270;
draw$(z_{20} \,..\, z_{21} \,..\, z_{40} \,..\, z_4)$;
endchar;

Máme-li k dispozici dostatečně bohatou sadu znaků, můžeme psát i obrázkové pohádky:

Byl jednou jeden ♂(◁) a ♀(◁). Babička jim ✍ ✉, že je ▷, že ji tak dlouho nenavštívili. Tak ♡ babičce ☎, že ji přijdou navštívit, aby napekla koláče. Byly 🕒, když se vydali na cestu. ▷ svítilo, ❀ kvetly a děti si ♪♩♫ pochodové písně. Našly i ❀. Přebrodily ≋, když se začlo smrákat a vyšel ☽. Setmělo se a na obloze se objevil i ♂, ‹ a ›. Chtěly se ⟳, ale nevěděly již cestu zpět. Strhla se ϟ. Už si myslely, že ✝, když ...[3]). A zazvonil ♠ a pohádky je ✌.

METAFONT si svou složitostí nezadá s TEXem, pro běžné smrtelníky však postačí základní informace, jak vygenerovat žádané fonty pro dané výstupní zařízení v potřebné rozlišovací schopnosti.

Makra `mfpic`. Pro ty, kteří nechtějí vnikat hluboko do tajů METAFONTu, se nabízejí jednoduchá makra s názvem `mfpic`. Jsou napsána v plainu, ale pro uživatele LATEXu je k dispozici jednoduchý soubor, který usnadní jejich použití.

Zadávání takového obrázku je podobné jako u maker `epic` a `eepic` či v okolí `picture` (viz ukázku) s tím rozdílem, že makra generují zdrojový text pro METAFONT. Po prvním překladu TEXem musíme spustit METAFONT, abychom v dalším průchodu TEXu již měli metrické informace spočítány; obrázek je pak automaticky vysázen na požadovaném místě. O tom, že nejde o nic složitého, svědčí i skutečnost, že vyčerpávající manuál k makrům `mfpic` čítá 9 stran.

```
\opengraphsfile{eurotex}
\begin{mfpic}[1]{-160}{150}{-150}{105}
\pen{1.5}
```

[3]) Dopsání pohádky ponecháme laskavému čtenáři jako cvičení.

```
\ellipse{(-110,90),70,40}
  \label{-140}{85}{manuscript}
\ellipse{(0,75),70,40}
  \label{-30}{70}{proof copy}
\ellipse{(0,-75),70,40}
  \label{-30}{-77}{\LTEX\ file}
\ellipse{(-110,0),70,40}
  \label{-137}{-5}{\ISIS}
\ellipse{(110,0),70,40}
  \label{90}{-5}{DVI file}
....
\end{mfpic}
```

Tento způsob generování obrázků je lehce přenositelný a nevyžaduje rozsáhlé znalosti METAFONTu.

Závěr. Všechny znaky a fonty použité v tomto článku jsou vytvořeny v METAFONTu a jsou k dispozici (jako public domain nebo freeware) v univerzitním elektronickém archivu `ftp.muni.cz` v adresáři `pub/tex/fonts`. V podadresáři `mfsrc` jsou zdrojové texty pro METAFONT. Speciální znaky použité v ‚pohádce' najdete v balících `wasy.zip` a `bbding.zip`. V pokladnici TEXových fontů dále najdete arabštinu, řečtinu ($\alpha, \ldots, \omega$), ‘цырилици’, ſvabach a desítky dalších písem. A to keště nezdůrazňujeme, že TEX je schopen používat i libovolná PostScriptová písma. Seznam fontů dostupných v TEXových archívech je možno nalézt v souboru `ftp.muni.cz:/pub/tex/faqs/mflist.faq`, krátký úvod do METAFONTu lze nalézt tamtéž v souboru `mf.faq`.

TEX a jízdní řády MHD

ing. Josef Veselý

Zavedení výpočetní techniky do zpracování tvorby grafikonu linek hromadné dopravy zcela jasně vyvolalo diskuse o řešení tiskového výstupu informací pro cestující veřejnost. Porovnáním množství informací na straně jedné a plošnými rozměry formátu papíru jsme dospěli k závěru, že tento úkol je nad možnosti (v té době v DPmB dostupných) tiskáren. Tehdejší situaci navíc ovlivňovala výpočetní technika na úrovni PC-AT 286.

V této situaci však zazářila TEXová hvězda, na kterou nás upozornil RNDr. Petr Sojka z ÚVT MU. Podle našich požadavků sestavil první verzi stylu a vzor zdrojového textu. Vlastní styl byl však náročný na paměť TEXu a vyžadoval překlad prostřednictvím BTEXu. Proto byl sestaven jednoduší styl, který respektoval omezení daná procesorem 286. Program, který zajistil vstup dat (časy odjezdů z konečných stanic), výpočet průjezdů zastávkami a následně sestavení tehdy ještě souboru `*.ED`, byl řešen v TurboBasicu. Verze zastávkových jízdních řádů byla pro velký úspěch doplněna o verzi služebního a sešitového jízdního řádu.

Toto provizorium bylo úspěšně provozováno bezmála 3 roky a způsobilo naši „závislost“ na TEXu.

V současné době dochází v rámci komplexního řešení dopravní problematiky v DPmB k přechodu na nový software mj. i pro tvorbu grafikonů. Zpracovatel softwaru — firma CHAPS — přijal již zavedenou koncepci oddělení výstupů od vlastního programu. Vysoce ocenil variabilní design tiskových výstupů přes dohodnuté rozhraní, kterým je zdrojový soubor `*.TEX`.

Pro nový software byla sestavena nová generace stylů umožňujících sazbu 2 verzí služebních jízdních řádů, 3 verzí jízdních řádů pro veřejnost a vyvinut speciální manažer nabízející spouštění jednotlivých programových produktů TEXu.

Uplatněním TEXu při sazbě jízdních řádů došlo zcela jednoznačně ke zkvalitnění prezentace podniku v dané oblasti. Nyní již zbývá pouze zajistit dodržování jízdních řádů ze strany provozních pracovníků, což je zatím velké sousto i na TEX.

Na dalších třech stranách přinášíme ukázky brněnských jízdních řádů.

Vranov - Útěchov - Soběšice - Semilaso / Štefánikova čtvrť - Malinovského náměstí / Husovice

zastávka \ *č.linky*	43	57	43	57	43	57	43	57	43	57	43	57
Vranov, myslivna	04.30	...	04.30	...	04.30	...	04.30	...	04.30	...	04.30	...
Útěchov	04.36	...	04.36	...	04.36	...	04.36	...	04.36	...	04.36	...
Útěchov, rozc.	04.38	...	04.38	...	04.38	...	04.38	...	04.38	...	04.38	...
U buku	04.45	...	04.45	...	04.45	...	04.45	...	04.45	...	04.45	...
U jezírka	04.47	...	04.47	...	04.47	...	04.47	...	04.47	...	04.47	...
Lesní školka	04.49	...	04.49	...	04.49	...	04.49	...	04.49	...	04.49	...
Soběšice	04.51	04.56	04.51	04.56	04.51	04.56	04.51	04.56	04.51	04.56	04.51	04.56
Semilaso (příj.)	05.12	⟨	05.12	⟨	05.12	⟨	05.12	⟨	05.12	⟨	05.12	⟨
Lesná, Brechtova	...	05.10	...	05.10	...	05.10	...	05.10	...	05.10	...	05.10
Lesná, Haškova	...	⟨	...	⟨	...	⟨	...	⟨	...	⟨	...	⟨
Štefánikova čtvrť	...	05.13	...	05.13	...	05.13	...	05.13	...	05.13	...	05.13
Husovice, Nováčkova (příj.)	...	05.20	...	05.20	...	05.20	...	05.20	...	05.20	...	05.20
Malinovského náměstí (příj.)	...	...	...	...	...	...	...	...	...	...	...	...

1

Řečkovice – Pisárky

Odjezdy ze zastávky Husitská směr PISÁRKY

	jen ve STŘEDU	jen ve čtvrtek	jen v pátek
0	43	43	43
1	43	43	43
2	43	43	43
3	43	43	43
4	43	43	43
5	43	43	43
6	43	43	43
7	43	43	43
8	43	43	43
9	43	43	43
10	15 25 35 45 55	15 25 35 45 55	15 25 35 45 55
11	15 25 35 45 55	15 25 35 45 55	15 25 35 45 55
12	15 25 35 45 55	15 25 35 45 55	15 25 35 45 55
13	15 25Ř 35 45 55	15 25Ř 35 45 55	15 25Ř 35 45 55
14	15Ř 25 35 45 55	15Ř 25 35 45 55	15Ř 25 35 45 55
15	15 25 35 45Ř 55	15 25 35 45Ř 55	15 25 35 45Ř 55
16	15 25 35 45 55	15 25 35 45 55	15 25 35 45 55
17	15 25 35 45 55	15 25 35 45 55	15 25 35 45 55
18	05 interval 5 - 6 min.	05 interval 5 - 6 min.	05 interval 5 - 6 min.
19		15 25 35 45 55	
20		15 25 35 45 55	
21		15 25 35 45 55	
22		15 25 35 45 55	
23	15 25 35 45 55		15 25 35 45 55

ŘEČKOVICE
Filkukova
Kořískova
Hudcova
Tylova
Semilasso
• Husitská
2 Jungmannova
4 Veterinární škola
6 Šumavská
8 Hrnčířská
10 Pionýrská
12 Antonínská
14 Moravské náměstí
17 Malinovského náměstí
20 Hlavní nádraží
23 Nové sady
24 Hybešova
26 Václavská
29 Mendlovo náměstí
31 Výstaviště
32 Vinařská (o)
33 Lipová (o)
34 PISÁRKY

o : na znamení od 20 do 5 hodin
z : na znamení

Vysvětlivky :

Ř ... jede jen z Řečkovic

Platí 19. listopadu 1993

linka:
kód stojanu:
soubor:
datum: 1
strana:

ŘEČKOVICE
Filkukova
Kořískova
Hudcova
Tylova
Semilasso
▽ **Husitská**
2 Jungmannova
4 Veterinární škola
6 Šumavská
8 Hrnčířská
10 Pionýrská
12 Antonínská
14 Moravské náměstí
17 Malinovského náměstí (z)
20 Hlavní nádraží
23 Nové sady
24 Hybešova
26 Václavská
29 Mendlovo náměstí
31 Výstaviště
32 Vinařská (o)
33 Lipová (o)
34 PISÁRKY

o : na znamení od 20 do 5 hodin
z : na znamení

REKREAČNÍ LINKA
V PROVOZU
OD 1.7. DO 31.8.1992

1

Husitská

PÁTEK

0	**43**
1	**43**
2	**43**
3	**43**
4	**43**
5	**43**
6	**43**
7	**43**
8	**43**
9	**43**
10	**15 25 35 45 55**
11	**15 25 35 45 55**
12	**15 25 35 45 55**
13	**15 25Ř 35 45 55**
14	**15Ř 25 35 45 55**
15	**15 25 35 45Ř 55**
16	**15 25 35 45 55**
17	**15 25 35 45 55**
18	**05** interval 5 - 6 min.
19	
20	
21	
22	
23	**15 25 35 45 55**

DPmB **Platí 19. listopadu 1993**

Poslední ukázka byla zmenšena na $\frac{11}{13}$ *pomocí programu* `bm2font`.

K čemu slouží WEB?

ZDENĚK WAGNER

K napsání tohoto článku mě vyprovokoval příspěvek Martina Bílého v loňském třetím čísle Zpravodaje. Tam je WEB popsán z hlediska „místního čaroděje“, který musí instalovat TEX ze zdrojových textů. V tomto článku bych se pokusil vysvětlit, jaké výhody přináší WEB programátorům.

Dlouho jsem uvažoval, zda mám vůbec tento článek napsat a zda se hodí do Zpravodaje. Nakonec mě k tomu přiměly dvě skutečnosti:

- Při psaní textu v LATEXu jsem se dostal do problémů, které jsem vyřešil pomocným programem v C++. Tento program by mohl být užitečný, a aby mohl být publikován, je vhodné jej napsat ve WEBu.
- Podobná filosofie programování se dá využít i při psaní rozsáhlých souborů TEXových maker.

M. Bílý označil WEB jako jazyk. Z hlediska matematické lingvistiky je to jistě pravda. Z praktického hlediska bych však raději označil WEB jako programovací nástroj. Není ovšem pravda, že WEB se podobá Pascalu — to vysvětlíme později.

Před časem jsem se ptal v elektronické konferenci `cstex`, zda již existuje oficiální překlad termínu „literate programming“. Nedostal jsem žádnou odpověď, a proto se sám pokusím o překlad. Chtěl bych se na tomto místě omluvit všem, kteří jsou vzdělanější než já. Mnoho věcí jsem studoval z anglických knih a nikdy mě nenapadlo přemýšlet nad tím, jak se výrazy jako „pointer“, „carriage return“, „box“, „glue“ správně přeloží do češtiny. Z toho vznikají mé současné problémy.

Slovo „literate“ lze přeložit jako „gramotný“ nebo „kultivovaný“, zatímco pro opačné slovo „illiterate“ jsem ve slovníku našel pouze jediný ekvivalent: „negramotný“. Raději bych tedy překládal „literate programming“ jako „kultivované programování“.

Základním konceptem kultivovaného programování je zápis zdrojového kódu programu i programové dokumentace do téhož souboru. Obrovská výhoda tohoto přístupu spočívá v tom, že program i dokumentace se udržují v synchronizaci. Dokumentace pak není o několik verzí

pozadu, jak se obvykle stává při „illiterate programming“. Pro předzpracování textu se používají dva překladače: `tangle`, jehož produktem je vstupní soubor pro kompilátor určitého programovacího jazyka, a `weave`, který vytváří formátovaný soubor pro vytištění dokumentace a zdrojového programu, obvykle pomocí TEXu.

Kultivovaný program se skládá ze sekcí. Každá sekce má tři části: část dokumentační, část definiční a část programovou. Kterákoliv z těchto částí může být prázdná. V dokumentační části se uvádí popis dané části programu v jazyce, který je srozumitelný lidským bytostem. Definiční část není nutná při použití jazyka C. Zde je totiž jinými prostředky implementována céčková direktiva `#define`. Programová část obsahuje vlastní program v příslušném programovacím jazyce. Speciálním příkazem lze do programové části vložit jinou sekci. Lze to přirovnat buď k preprocesoru jazyka C, nebo (abych se více držel TEXu) definici makra, v níž je voláno další makro. Programátor tak získává další úroveň strukturovanosti, což umožňuje převést lineární strukturu programu do logičtějšího členění. V každém programu musí být jedna bezejmenná sekce, kterou `tangle` bude považovat za začátek programu. Ostatní sekce mají svá jména.

Před okamžikem bylo uvedeno, že dokumentace se tiskne obvykle TEXem a programová část je psána v příslušném programovacím jazyce. Z toho je vidět, že označení WEB je naprosto nejednoznačné. Existuje více systémů, které implementují stejnou filosofii. Liší se jednak tím, zda podporují více programovacích jazyků, nebo se specializují na jeden z nich (C, Pascal, apod.), a také, jakým programem se pak tiskne dokumentace připravená překladačem `weave`. Původní WEB byl napsán pro Pascal a z toho právě plyne mylná představa, že „jazyk WEB se podobá Pascalu.“

V následujícím textu se omezíme pouze na `CWEB`. Je určen pro jazyk C a ve verzi 3.0 i pro C++. Pro tisk dokumentace slouží plain TEX, od verze 3.0 lze použít i LATEX. Budeme si jej demonstrovat na kompletním programu, který rutinně používám, a z didaktických důvodů si až v závěru vysvětlíme, proč jsem vlastně tento program napsal. Předtím je ovšem vhodné uvést alespoň některé příkazy `CWEBu`. Ve vytištěném textu sice nebudou vidět, ale usnadní nám výklad dalších výhod WEBu.

Všechny příkazy `CWEBu` začínají znakem „@“. Chceme-li tedy použít tento znak v textu, musíme jej zdvojit, tj. napíšeme „@@“.

Jedním z nejdůležitějších příkazů je definice jména sekce. Jméno sekce píšeme mezi `@<` a `@>`. Očíslování sekcí zařídí `weave`. Programová část

sekce začíná uvedením jména, za nímž následuje znak „=“. Do programové části začleníme jinou sekci tím, že mezi příkazy jazyka C uvedeme její jméno.

Programovou část bezejmenné sekce musíme uvést speciálním příkazem. Je jím `@c`. Kvůli kompatibilitě s původním WEBem pro Pascal je povolen i `@p`.

`Weave` provádí formátování zdrojového textu tak, aby byl čitelnější. Pro tento účel si rozkládá jednotlivé příkazy a různé části pak tiskne různými fonty. Za konec příkazu se považuje středník. Jazyk C je ovšem nedůsledný v tom smyslu, že v určitých příkazech se středník nepíše. Aby překladač `weave` nebyl zmaten, použijeme v takovém případě příkaz `@;`.

Zdrojový text je formátován tak, aby každý programový příkaz začínal na novém řádku. To není vždy žádoucí. K potlačení přechodu na nový řádek slouží příkaz `@+`.

To samozřejmě není úplný výčet toho, co musíte znát, chcete-li použít `CWEB` pro psaní vlastních programů. Podrobnější informace naleznete v případě zájmu v dokumentaci, která je součástí balíku `CWEB` a podobně jako TEX patří mezi volně šiřitelné programy.

Nyní již následuje slíbený program.

1. PROGRAM PERCENT. Účelem tohoto programu je usnadnit psaní balíků (LA)TEXových maker dokumentovaných pomocí DOC.STY a DOCSTRIP.TEX. Dokumentace k makrům se zapisuje do komentářů, makra se zapisují mezi `%␣␣␣␣␣\begin{macrocode}` a `%␣␣␣␣␣\end{macrocode}`. Program umožňuje konverzi mezi pohodlnějším zápisem a tvarem pro DOC.STY.

Pro snadnější zápis dokumentace existují následující příkazy, které musí být psány na samostatném řádku od prvního sloupce:

`%+`	před všechny následující neprázdné řádky bude přidáno procento a mezera.
`%−`	ruší funkci %+.
`%%+`	před všechny následující neprázdné řádky budou přidány dva znaky procento a mezera.
`%%−`	ruší funkci %%+.
`%+++`	expanduje na `%␣␣␣␣␣\begin{macrocode}`.
`%−−−`	expanduje na `%␣␣␣␣␣\end{macrocode}`.

Tato funkce se vyvolá příkazem:

`percent +<`*filename*`>`

kde *filename* je plné jméno souboru. Soubor bude nahrazen konvertovanou verzí, původní soubor bude uschován v souboru s příponou `.bak`.

Opačná konverze se provede příkazem:

`percent −<`*filename*`>`

Zatímco v původním souboru můžete libovolně míchat příkazy %+++ a %−−− s `%␣␣␣␣\begin{macrocode}` a `%␣␣␣␣\end{macrocode}`, při zpětné konverzi budou všechny výskyty `%␣␣␣␣\begin{macrocode}` i `%␣␣␣␣\end{macrocode}` převedeny na %+++ a %−−−.

2. Program bude mít následující části.

⟨ Hlavičkové soubory 3 ⟩
⟨ Prototypy 5 ⟩
⟨ Globální proměnné 10 ⟩
⟨ Hlavní program 4 ⟩
⟨ Funkce 6 ⟩

3. Jak později uvidíme, budeme potřebovat následující hlavičkové soubory. (Samozřejmě bychom tuto sekci mohli rozdělit na mnoho malých fragmentů a načítat každý hlavičkový soubor vždy, když se použije funkce v něm definovaná. Podle mého názoru to ale k přehlednosti programu nepřispěje.)

⟨ Hlavičkové soubory 3 ⟩ ≡

```
#include <dir.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <fstream.h>
#include <ibmanip.h>
#include <useful.h>
#include <stdlib.h>
#include <ctype.h>
```

Tento kód je použit v sekci 2.

4. Hlavní program musí přečíst parametr z příkazového řádku. Všimněte si, že mezi znakem + či − a jménem souboru není mezera. Vše se tedy přečte jako jediný parametr, takže *argc* musí mít hodnotu 2. První znak parametru rozhodne o tom, jakou funkci máme provést, zbytek se pošle dál jako jméno souboru.

```
⟨ Hlavní program 4 ⟩ ≡
  void main(int argc, const char *argv[])
  {
    if (argc ≠ 2) {
      cout ≪ "Nesprávný␣počet␣parametrů\n";
      return;
    }
    switch (*argv[1]) {
    case '+': AddPerCent(argv[1] + 1); break;
    case '-': RemovePerCent(argv[1] + 1); break;
    default: cout ≪ "Neznámá␣funkce;␣zvol␣+␣nebo␣-.\n";
    }
  }
```

Tento kód je použit v sekci 2.

5. Použité funkce musí mít své prototypy.

```
⟨ Prototypy 5 ⟩ ≡
  void AddPerCent(const char *);
  void RemovePerCent(const char *);
  int MakeBakFile(const char *);
```

Tento kód je použit v sekci 2.

6. V programu budeme potřebovat tři funkce (tu třetí jsme předčasně uvedli v sekci prototypů).

```
⟨ Funkce 6 ⟩ ≡
  ⟨ Přidej procento 8 ⟩
  ⟨ Odstraň procento 26 ⟩
  ⟨ Udělej záložní soubor 7 ⟩
```

Tento kód je použit v sekci 2.

7. Nejprve si vytvoříme posledně jmenovanou funkci. Ta vezme jméno souboru jako parametr a rozdělí jej na jednotlivé části. Pak vytvoří jméno, které má příponu `.bak`. Původní soubor s příponou `.bak` se zruší a zdrojový soubor je přejmenován. Tento soubor je pak otevřen a funkce vrátí hodnotu rukojeti souboru (file handle).

```
⟨ Udělej záložní soubor 7 ⟩ ≡
  int MakeBakFile(const char *fn)
  {
    char myfn[MAXPATH], fullfn[MAXPATH], dr[MAXDRIVE], dir[MAXDIR],
         fname[MAXFILE], ext[MAXEXT];
    _fullpath(fullfn, fn, sizeof fullfn);
    fnsplit(fullfn, dr, dir, fname, ext);
    fnmerge(myfn, dr, dir, fname, ".bak");
    unlink(myfn);
    if (rename(fullfn, myfn)) {
      cout ≪ "chyba␣vstupu/výstupu,␣pravděpodobně␣nee\
          xistující␣soubor\n";
      exit(EXIT_FAILURE);
    }
    return open(myfn, O_RDONLY | O_TEXT);
  }
```

Tento kód je použit v sekci 6.

8. Následující funkce bude konvertovat soubor do formátu pro DOC.STY, tj. stručně řečeno, bude přidávat procento na začátek řádků.

```
⟨ Přidej procento 8 ⟩ ≡
  void AddPerCent(const char *fn)
  {
    ⟨ Otevři soubory 9 ⟩
    ⟨ Add: inicializuj lokální proměnné 12 ⟩
    ⟨ Add: zpracuj soubor 13 ⟩
    ⟨ Zavři soubory 11 ⟩
  }
```

Tento kód je použit v sekci 6.

9. Při otvírání vstupního souboru použijeme vlastní buffer pro zrychlení čtení a zápisu. Jako první parametr v konstruktoru použijeme rukojeť souboru, kterou vrátí funkce *MakeBakFile*.

⟨ Otevři soubory 9 ⟩ ≡
```
ifstream inf(MakeBakFile(fn), inbuf, sizeof inbuf);
ofstream outf(fn);
```
Tento kód je použit v sekcích 8 a 26.

10. Buffer musíme deklarovat jako globální proměnnou.

⟨ Globální proměnné 10 ⟩ ≡
```
char inbuf[16384];
```
Viz též sekce 23, 24 a 25.

Tento kód je použit v sekci 2.

11. Zavírání souborů nepotřebuje bližší vysvětlení.

⟨ Zavři soubory 11 ⟩ ≡
```
outf.close(); inf.close();
```
Tento kód je použit v sekcích 8 a 26.

12. V této funkci budeme především potřebovat proměnnou, do níž budeme načítat vstupní řádky. Mlčky budeme předpokládat, že řádek nebude mít více než 255 znaků. Kromě toho potřebujeme logické proměnné s následujícím významem:

one	zpracovává se text za příkazem %+.
two	zpracovává se text za příkazem %%+.
macro	zpracovává se "macrocode". V určitém okamžiku se dočasně využívá i pro jiné účely, což bude na příslušném místě dokumentováno.
empty	poslední zpracovaný řádek byl prázdný.

Všechny logické proměnné se na začátku naplní hodnotou 0.

⟨ Add: inicializuj lokální proměnné 12 ⟩ ≡
```
char Line[256];
int one, two, macro, empty;
one ← two ← macro ← empty ← 0;
```
Tento kód je použit v sekci 8.

13. Dokud jsou data ve vstupním souboru a do výstupního souboru lze psát, budeme číst při každém průchodu vždy jeden řádek, zjistíme, zda je prázdný, a dále provedeme podrobnější analýzu a zpracování.

```
⟨ Add: zpracuj soubor 13 ⟩ ≡
  while (inf ∧ outf ∧ ¬inf.eof()) {
    ⟨ Add: přečti řádek 14 ⟩
    ⟨ Add: zpracuj prázdný řádek 15 ⟩
    ⟨ Add: analyzuj a zpracuj řádek 16 ⟩
  }
```

Tento kód je použit v sekci 8.

14. Některé editory nechávají na konci řádků mezery, což by znemožnilo správné rozeznávání prázdných řádků. Proto po načtení řádku a přeskočení všech následujících znaků až do `\n` včetně zavoláme funkci, která odstraní koncové mezery.

```
⟨ Add: přečti řádek 14 ⟩ ≡
  inf.get(Line, sizeof Line); inf ≫ endl; CutStg(Line);
```

Tento kód je použit v sekci 13.

15. V TEXu prázdný řádek znamená konec odstavce. Dva či více po sobě následující prázdné řádky nemají žádný smysl, a proto je odstraníme. Jestliže najdeme prázdný řádek a předchozí řádek byl také prázdný, skočíme rovnou na konec smyčky.

```
⟨ Add: zpracuj prázdný řádek 15 ⟩ ≡
  if (¬∗Line) {
    if (¬empty) outf ≪ Line ≪ endl;
    empty ← 1;
    continue;
  }
  else empty ← 0;
```

Tento kód je použit v sekci 13.

16. Při analýze a zpracování řádku budeme nejprve hledat začátek prostředí macrocode. Je-li nastaven přepínač *macro*, budeme hledat konec prostředí macrocode. V opačném případě zjistíme typ příkazu a provedeme výstup.

```
⟨ Add: analyzuj a zpracuj řádek 16 ⟩ ≡
  ⟨ Hledej begin macrocode 17 ⟩
  if (macro) {
    ⟨ Hledej end macrocode 18 ⟩
  }
  else {
    ⟨ Zjisti typ příkazu 19 ⟩
    ⟨ Zapiš upravený řádek 20 ⟩
  }
```

Tento kód je použit v sekci 13.

17. Zde případně expandujeme %+++. Pokud objevíme začátek prostředí macrocode, nastavíme přepínač *macro*. Současně si nadefinujeme *beginmac* pro pohodlnější programování.

```
#define beginmac "%␣␣␣␣\\begin{macrocode}"
⟨ Hledej begin macrocode 17 ⟩ ≡
  if (¬macro ∧ ¬strcmp(Line, "%+++")) {
    strcpy(Line, beginmac); macro ← 1;
  }
  if (¬macro ∧ ¬strcmp(Line, beginmac)) macro ← 1;
```

Tento kód je použit v sekci 16.

18. Při zpracování prostředí macrocode musíme nejprve otestovat a případně expandovat %−−−. Poté zapíšeme řádek do výstupního souboru, a pokud prostředí končí, vynulujeme *macro*. I zde použijeme definici makra pro usnadnění života.

```
#define endmac "%␣␣␣␣\\end{macrocode}"
⟨ Hledej end macrocode 18 ⟩ ≡
  if (¬strcmp(Line, "%---")) strcpy(Line, endmac);
  outf ≪ Line ≪ endl;
  if (¬strcmp(Line, endmac)) macro ← 0;
```

Tento kód je použit v sekci 16.

19. Nyní vyzkoušíme, zda řádek obsahuje některý ze zbývajících příkazů. Dočasně použijeme proměnnou *macro* pro jiný účel. Hodnota $+1$ bude znamenat zahájení příslušného typu zpracování, hodnota -1 znamená jeho konec. Současně se nastaví proměnné *one* a *two*. Přepínač *macro* bude vynulován v následující sekci.

```
⟨ Zjisti typ příkazu 19 ⟩ ≡
  if (¬strcmp(Line, "%+")) {
    macro ← 1; one ← 1; two ← 0;
  }
  if (¬strcmp(Line, "%-")) {
    macro ← −1; one ← two ← 0;
  }
  if (¬strcmp(Line, "%%+")) {
    macro ← 1; one ← 0; two ← 1;
  }
  if (¬strcmp(Line, "%%-")) {
    macro ← −1; one ← two ← 0;
  }
```

Tento kód je použit v sekci 16.

20. Je-li nastaven přepínač *macro*, je na současném řádku formátovací příkaz a celý řádek se ignoruje. V opačném případě zapíšeme úvodní procenta podle nastavení proměnných *one* a *two* a samozřejmě celý řádek.

```
⟨ Zapiš upravený řádek 20 ⟩ ≡
  if (macro) macro ← 0;
  else {
    if (one) outf ≪ "%␣";
    else if (two) outf ≪ "%%␣";
    outf ≪ Line ≪ endl;
  }
```

Tento kód je použit v sekci 16.

21. Zpětná konverze je poněkud složitější. Naprogramujeme ji tedy jako stavový automat s následujícími deseti stavy:

0: normální výstup.
1: nalezen jeden řádek začínající znakem %.
2: nalezeny dva řádky začínající znakem %.
3: nalezen jeden řádek začínající znaky %%.
4: nalezeny dva řádky začínající znaky %%.
5: přerušení výstupu řádků typu %.
6: přerušení výstupu řádků typu %%.
7: prostředí macrocode uvnitř normálního výstupu.
8: prostředí macrocode během výstupu řádků typu %.
9: prostředí macrocode během výstupu řádků typu %%.

Je zřejmé, že stavy 1–4 brání vytváření hloupých sekvencí

```
%+
Jednořádkový komentář
%−
```

Nadefinujeme si tedy odpovídající prahovou hodnotu.

```
#define threshold 3
```

22. Stavový automat musí rozeznat následující typy řádků:

0: prázdný řádek.
1: řádek začínající znakem %.
2: řádek začínající znaky %%.
3: `%␣␣␣␣\begin{macrocode}`.
4: `%␣␣␣␣\end{macrocode}`.
5: jiný.

23. V každém stavu se provede jedna z následujících procedur:

⟨ **_PA_** ⟩	zapiš vše, vynuluj *idx*.
⟨ **_KA_** ⟩	inkrementuj *idx*.
⟨ **_C1_** ⟩	zapiš všechny řádky s odstraněným znakem %, vynuluj *idx*.
⟨ **_MC1_** ⟩	zapiš příkaz %+ a proveď ⟨ _C1_ ⟩.
⟨ **_C2_** ⟩	zapiš všechny řádky s odstraněnými znaky %% a vynuluj *idx*.
⟨ **_MC2_** ⟩	zapiš příkaz %%+ a proveď ⟨ _C2_ ⟩.
⟨ **_KP_** ⟩	zapiš všechny řádky s výjimkou posledního, zkopíruj řádek[*idx*] do řádku[0] a nastav *idx*=1.
⟨ **_K1_** ⟩	zapiš příkaz %− a proveď ⟨ _KP_ ⟩.
⟨ **_K2_** ⟩	zapiš příkaz %%− a proveď ⟨ _KP_ ⟩.
⟨ **_S1_** ⟩	zapiš příkaz %− a proveď ⟨ _PA_ ⟩.
⟨ **_S2_** ⟩	zapiš příkaz %%− a proveď ⟨ _PA_ ⟩.
⟨ **_ERR_** ⟩	zobraz chybovou zprávu a proveď ⟨ _PA_ ⟩.

Definici enumerací musíme zařadit mezi globální proměnné, protože je budeme potřebovat později v definici stavového automatu.

```
⟨ Globální proměnné 10 ⟩ +≡
  enum ProcSteps {
    _PA_, _KA_, _C1_, _MC1_, _C2_, _MC2_, _KP_, _K1_, _K2_, _S1_, _S2_,
        _ERR_
  };
```

24. Stavový automat bude definován jako pole struktur. Struktura musí mít dvě položky: číslo nového stavu a označení procedury, která se má provést. Deklaraci odpovídající struktury přidáme k definici globálních proměnných, přestože zde pouze definujeme typ bez alokace paměti.

```
⟨ Globální proměnné 10 ⟩ +≡
  struct StateMachine {
    int NewState, ProcStep;
  };
```

25. Stavový automat je definován následující tabulkou. Ke každému stavu a typu vstupního řádku přísluší číslo nového stavu a označení požadované procedury (viz sekce 23).

Stav	prázdný řádek	%	%%	%+ + +	%− − −	jiný
0	0 ⟨ _PA_ ⟩	1 ⟨ _KA_ ⟩	3 ⟨ _KA_ ⟩	7 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩
1	0 ⟨ _PA_ ⟩	1 ⟨ _KA_ ⟩	3 ⟨ _KP_ ⟩	7 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩
2	0 ⟨ _PA_ ⟩	5 ⟨ _MC1_ ⟩	3 ⟨ _KP_ ⟩	7 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩
3	0 ⟨ _PA_ ⟩	1 ⟨ _KP_ ⟩	4 ⟨ _KA_ ⟩	7 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩
4	0 ⟨ _PA_ ⟩	1 ⟨ _KP_ ⟩	6 ⟨ _MC2_ ⟩	7 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩
5	5 ⟨ _PA_ ⟩	5 ⟨ _C1_ ⟩	3 ⟨ _K1_ ⟩	8 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _S1_ ⟩
6	5 ⟨ _PA_ ⟩	1 ⟨ _K2_ ⟩	6 ⟨ _C2_ ⟩	9 ⟨ _PA_ ⟩	0 ⟨ _ERR_ ⟩	0 ⟨ _S2_ ⟩
7	7 ⟨ _PA_ ⟩	7 ⟨ _PA_ ⟩	7 ⟨ _PA_ ⟩	7 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩	7 ⟨ _PA_ ⟩
8	8 ⟨ _PA_ ⟩	8 ⟨ _PA_ ⟩	8 ⟨ _PA_ ⟩	8 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩	8 ⟨ _PA_ ⟩
9	9 ⟨ _PA_ ⟩	9 ⟨ _PA_ ⟩	9 ⟨ _PA_ ⟩	9 ⟨ _ERR_ ⟩	0 ⟨ _PA_ ⟩	9 ⟨ _PA_ ⟩

Pro lepší hospodaření se zásobníkem budeme automat definovat globálně.

```
⟨ Globální proměnné 10 ⟩ +≡
  StateMachine RM[][6] ← {
      {{0, _PA_}, {1, _KA_}, {3, _KA_}, {7, _PA_}, {0, _ERR_}, {0, _PA_}},
    /* stav=0 */
  {{0, _PA_}, {2, _KA_}, {3, _KP_}, {7, _PA_}, {0, _ERR_}, {0, _PA_}},
    /* stav=1 */
  {{0, _PA_}, {5, _MC1_}, {3, _KP_}, {7, _PA_}, {0, _ERR_}, {0, _PA_}},
    /* stav=2 */
  {{0, _PA_}, {1, _KP_}, {4, _KA_}, {7, _PA_}, {0, _ERR_}, {0, _PA_}},
    /* stav=3 */
  {{0, _PA_}, {1, _KP_}, {6, _MC2_}, {7, _PA_}, {0, _ERR_}, {0, _PA_}},
    /* stav=4 */
  {{5, _PA_}, {5, _C1_}, {3, _K1_}, {8, _PA_}, {0, _ERR_}, {0, _S1_}},
    /* stav=5 */
  {{6, _PA_}, {1, _K2_}, {6, _C2_}, {9, _PA_}, {0, _ERR_}, {0, _S2_}},
    /* stav=6 */
  {{7, _PA_}, {7, _PA_}, {7, _PA_}, {7, _ERR_}, {0, _PA_}, {7, _PA_}},
    /* stav=7 */
```

```
  {{8, _PA_}, {8, _PA_}, {8, _PA_}, {8, _ERR_}, {5, _PA_}, {8, _PA_}},
    /* stav=8 */
  {{9, _PA_}, {9, _PA_}, {9, _PA_}, {9, _ERR_}, {6, _PA_}, {9, _PA_}}
    /* stav=9 */
  };
```

26. Podobně jako při opačném procesu, i nyní nejprve otevřeme soubory, provedeme inicializaci proměnných, zpracujeme soubor a nakonec všechny soubory zavřeme.

```
⟨ Odstraň procento 26 ⟩ ≡
  void RemovePerCent(const char *fn)
  {
    ⟨ Otevři soubory 9 ⟩
    ⟨ Rem: inicializuj proměnné 27 ⟩
    ⟨ Rem: zpracuj soubor 28 ⟩
    ⟨ Zavři soubory 11 ⟩
  }
```

Tento kód je použit v sekci 6.

27. Do proměnné *Line* budeme načítat vstupní řádky a *idx* bude obsahovat počet načtených řádků. Další proměnné budou obsahovat stav automatu a typ procedury, která se má provést. Jejich přesný význam bude jasnější později, nyní pouze některé z nich vynulujeme.

```
⟨ Rem: inicializuj proměnné 27 ⟩ ≡
  char Line[threshold][256];
  int idx, state, empty, linetype, newstate, proctype, maxstate;
  maxstate ← sizeof (RM)/sizeof (RM[1]);
  idx ← state ← empty ← 0;
```

Tento kód je použit v sekci 26.

28.

```
⟨ Rem: zpracuj soubor 28 ⟩ ≡
  while (inf ∧ outf ∧ ¬inf.eof()) {
    ⟨ Rem: načti vstupní řádek 29 ⟩
    ⟨ Zkontroluj stav 30 ⟩
    ⟨ Rem: zjisti typ řádku 31 ⟩
    ⟨ Rem: zjisti nový stav a typ procedury 32 ⟩
    ⟨ Rem: zpracování prázdného řádku 33 ⟩
    ⟨ Proveď příslušnou proceduru 35 ⟩
    ⟨ Nastav nový stav 34 ⟩
  }
```

Tento kód je použit v sekci 26.

29. Po načtení řádku a přeskočení všech znaků po `\n` včetně opět odstraníme případné koncové mezery.

```
⟨ Rem: načti vstupní řádek 29 ⟩ ≡
  inf.get(Line[idx], sizeof (Line[idx])); inf ≫ endl;
  CutStg(Line[idx]);
```

Tento kód je použit v sekci 28.

30. Nyní ověříme, zda je automat v přípustném stavu. Kontrola je vlastně zbytečná, neboť do nepřípustného stavu se automat může dostat pouze tehdy, je-li program poškozen (např. virem).

```
⟨ Zkontroluj stav 30 ⟩ ≡
  if (state < 0 ∨ state ≥ maxstate) {
    cout ≪ "Nepřípustný␣stav␣" ≪ state ≪ endl;
    cout ≪ "Program␣je␣poškozen\n";
    exit(EXIT_FAILURE);
  }
```

Tento kód je použit v sekci 28.

31. Dalším krokem je zjištění typu řádku tak, jak je uvedeno v sekci 22. Zjišťování se provádí jednoduchými příkazy **if**, které nepotřebují bližší komentář.

```
⟨ Rem: zjisti typ řádku 31 ⟩ ≡
  linetype ← 5;
  if (¬Line[idx][0]) linetype ← 0;
  else {
    if (¬strcmp(Line[idx], beginmac)) {
      strcpy(Line[idx], "%+++"); linetype ← 3;
    }
    else if (¬strcmp(Line[idx], endmac)) {
      strcpy(Line[idx], "%---"); linetype ← 4;
    }
    else if (Line[idx][0] ≡ '%') {
      if (Line[idx][1] ≠ '%') linetype ← 1;
      else if (Line[idx][2] ≠ '%') linetype ← 2;
    }
  }
```

Tento kód je použit v sekci 28.

32. Z tabulky zjistíme nový stav automatu a typ procedury, kterou máme provést.

```
⟨ Rem: zjisti nový stav a typ procedury 32 ⟩ ≡
  newstate ← RM[state][linetype].NewState;
  proctype ← RM[state][linetype].ProcStep;
```

Tento kód je použit v sekci 28.

33. Je-li *linetype* $\equiv 0$, znamená to, že poslední řádek je prázdný. Dva a více prázdných řádků nemají v TEXu význam, proto budeme nadbytečné prázdné řádky ignorovat. Po prázdném řádku si nastavíme switch *empty*. Je-li tento switch již nastaven, znamená to, že i předchozí řádek byl prázdný. V tom případě nastavíme nový stav automatu a skočíme rovnou na konec smyčky. Jestliže poslední řádek není prázdný, musíme vynulovat *empty*.

```
⟨ Rem: zpracování prázdného řádku 33 ⟩ ≡
  if (linetype ≡ 0) {
    if (empty) {
      ⟨ Nastav nový stav 34 ⟩
      continue;
    }
    empty ← 1;
  }
  else empty ← 0;
```

Tento kód je použit v sekci 28.

34. Nastavení nového stavu je triviální.

```
⟨ Nastav nový stav 34 ⟩ ≡
  state ← newstate;
```

Tento kód je použit v sekcích 28 a 33.

35. Typ procedury máme uložen v proměnné *proctype*. Nejprve provedeme kontrolu, zda je procedura povolena. Chyba je obvykle způsobena špatným vnořením prostředí „macrocode“. Pak následuje rozhodovací blok pro jednotlivé typy procedur, které byly definovány v sekci 23.

```
⟨ Proveď příslušnou proceduru 35 ⟩ ≡
  if (proctype ≡ _ERR_) {
    cout ≪ "Chyba␣ve␣vstupním␣souboru;␣zkontroluj␣p\
        rostředí␣macrocode\n";
  }
  char *s;
  int j;
  switch (proctype) {
  case _ERR_: case _S1_: case _S2_: case _PA_: ⟨ Proc P 36 ⟩ break;
  case _K1_: case _K2_: case _KP_: ⟨ Proc K 37 ⟩ break;
  case _MC1_: case _MC2_: ⟨ Proc M 38 ⟩
  case _C1_: case _C2_: ⟨ Proc C 39 ⟩ break;
  case _KA_: idx++; break;
  default: ⟨ Default proc 40 ⟩
  }
```

Tento kód je použit v sekci 28.

36. Tyto procedury musí především zapsat všechny uschované řádky. Procedury ⟨ _S1_ ⟩ a ⟨ _S2_ ⟩ navíc zapíší příkaz pro ukončení příslušného typu komentáře.

```
⟨ Proc P 36 ⟩ ≡
  if (proctype ≡ _S1_) outf ≪ "%-\n";
  else if (proctype ≡ _S2_) outf ≪ "%%-\n";
  for (j ← 0; j ≤ idx; j++) outf ≪ Line[j] ≪ endl;
  idx ← 0;
```

Tento kód je použit v sekci 35.

37. Tyto procedury se podobají předchozím, pouze pořadí příkazů je odlišné a navíc nezapisujeme poslední řádek.

```
⟨ Proc K 37 ⟩ ≡
  for (j ← 0; j < idx; j++) outf ≪ Line[j] ≪ endl;
  if (proctype ≡ _K1_) outf ≪ "%-\n";
  else if (proctype ≡ _K2_) outf ≪ "%%-\n";
  if (idx) strcpy(Line[0], Line[idx]);
  if (idx) idx ← 1;      /* vynechat, pokud bylo idx ≡ 0 */
```

Tento kód je použit v sekci 35.

38. Tyto procedury zapíší příkaz pro zahájení komentáře. V příkazu **switch** nejsou ukončeny příkazem **break**, takže pokračují do procedur ⟨ _C1_ ⟩, ⟨ _C2_ ⟩.

```
⟨ Proc M 38 ⟩ ≡
  if (proctype ≡ _MC1_) outf ≪ "%+\n";
  else outf ≪ "%%+\n";
```

Tento kód je použit v sekci 35.

39. Zde odstraníme jeden nebo dva znaky procento (podle typu řádku) a všechny následující mezery.

```
⟨ Proc C 39 ⟩ ≡
  for (j ← 0; j ≤ idx; j++) {
    s ← Line[j] + ((proctype ≡ _C1_ ∨ proctype ≡ _MC1_) ? 1 : 2);
    while (*s ∧ isspace(*s)) s++;
    outf ≪ s ≪ endl;
  }
  idx ← 0;
```

Tento kód je použit v sekci 35.

40. Toto je opět situace, která za normálních okolností nemůže nastat. Pokud program dojde do tohoto stavu, znamená to, že je poškozen.

```
⟨ Default proc 40 ⟩ ≡
  cout ≪ "Program␣je␣poškozen\n";
  exit(EXIT_FAILURE);
```

Tento kód je použit v sekci 35.

CWEB má ještě další užitečné prostředky, které jsme zde nepoužili. Nejsou-li potlačeny, generuje weave automaticky rejstřík všech proměnných a obsah. U rozsáhlých programů to představuje nespornou výhodu.

Kultivovaně napsaný program má ještě další výhodu. Obvykle v něm po prvním napsání bývá málo chyb, takže nevyžaduje zdlouhavé ladění. Programátor totiž vše vysvětluje lidskou řečí v dokumentační části a to jej vede k podrobnější analýze algoritmu, který píše. Tím odhalí různá úskalí dříve, než se jeho počítač zacyklí v nekonečné smyčce.

Ladění se ovšem zřejmě nevyhneme u žádného většího programu. Tehdy nezbývá, než najít předpokládanou příčinu chyby, provést modifikaci zdrojového programu a zkusit znovu. Chyb ale může být více a při ladění se můžeme splést. Tak se stává, že najednou chceme cosi vrátit do původního stavu, jenže nemáme použitelnou kopii a původní stav jsme již zapomněli.

Zde se vyplatí další nástroj, který nám WEB nabízí — a to tzv. změnový soubor. Jak název napovídá, jsou v něm obsaženy změny, které hodláme provést. Jeho struktura je následující:

```
@x
Původní řádky
@y
Nové řádky
@z
```

„Původní řádky“ i „Nové řádky“ mohou obsahovat libovolný text s výjimkou příkazů @x, @y a @z. Text, který předchází @x nebo následuje za @z (až do @x) je ignorován. Můžete si zde tudíž poznamenat, proč příslušnou změnu provádíte.

Když je program konečně odladěn, je vhodné začlenit změny do zdrojového textu. K tomu lze použít program wmerge.

Změnový soubor hraje důležitou roli při implementaci TEXu, ale i jiných programů, kde je kladen důraz na přenositelnost. Představte si, že

chcete implementovat nějaký program na neobvyklém počítači s obskurním operačním systémem. Prostudujete příslušný program a uděláte si změnový soubor. Po nějakém čase dá autor programu k dispozici novou verzi. Implementace této verze bude nyní představovat jen malou (nebo dokonce žádnou!) úpravu změnového souboru.

Při podrobnějším pohledu na program PerCent zjistíte, že se vám jej nepodaří přeložit. Je to dáno tím, že nemáte soubory `ibmanip.h` a `useful.h` a navíc linker nenajde funkci *CutStg* ani operátor *endl* pro **istream**. Potřebujete tedy změnový soubor, v němž nejprve vyřadíme příkazy načítající neexistující soubory:

```
@x
#include␣<ibmanip.h>
#include␣<useful.h>
#include␣<stdlib.h>
@y
#include␣<stdlib.h>
@z
```

Dále musíme nadefinovat prototypy operátoru *endl* a funkce *CutStg*.

```
@x
@<Prototypy@>=
@y
@<Prototypy@>=
istream&␣operator␣endl;
int␣CutStg␣(char*);
@z
```

Nakonec ještě musíme příslušné funkce implementovat. Pro jednoduchost pouze přidáme odpovídající příkazy do sekce „Funkce“. Lepší by ovšem bylo, kdybychom přidali i dokumentaci.

```
@x
@<Funkce@>=
@<Přidej␣procento@>@;
@<Odstraň␣procento@>@;
@<Udělej␣záložní␣soubor@>@;
@y
@<Funkce@>=
```

```
@<Přidej␣procento@>@;
@<Odstraň␣procento@>@;
@<Udělej␣záložní␣soubor@>@;

int␣CutStg␣(char␣*s)␣{
for␣(int␣j=strlen(s)-1;␣␣j>=0␣&&␣s[j]>=0␣&&␣isspace(s[j]);␣␣j--);
s[++j]␣=␣'\0';
return␣j;
}

istream&␣endl␣(istream&␣is)␣{
char␣c;
do␣{␣is.get(c);␣}␣while␣(c␣!=␣'\n'␣&&␣!is.eof()␣&&␣is);
return␣is;
}
@z
```

Ukázali jsme si, že změnový soubor je zcela nezávislý na programovacím jazyku. Lze jej tedy použít naprosto k čemukoliv. Chcete-li např. upravit pro své potřeby balík TEXových maker, můžete si napsat změnový soubor, a protože zde nelze použít `tangle` ani `weave` z `CWEBu`, vytvoříme nový balík pomocí `wmerge`.

Rozsáhlejší balík TEXových maker se svou složitostí vyrovná programům v jiných jazycích. I zde je tedy vhodné využít koncept kultivovaného programování. Můžeme použít některý z WEBů, který není závislý na programovacím jazyce, např. `noweb` nebo `nuweb`. To má ovšem jednu nevýhodu: uživatel TEXu nemusí nutně být programátor, a nebude tudíž umět implementovat WEB na svůj počítač. Každý uživatel TEXu má však TEX. To je využito v balíku DOC.STY, který vytvořil Frank Mittelbach. Definice maker se píší obvyklým způsobem, pouze se vkládají mezi řádky s příkazy `%␣␣␣␣␣\begin{macrocode}` a `%␣␣␣␣␣\end{macrocode}`. Všimněte si čtyř mezer — ty jsou v uvedených příkazech nezbytné. Dokumentace maker se uvádí v komentářích, tedy v řádcích uvedených znaky %. Pokud získáte takový balík maker, můžete jej použít bez úprav tak, jak je. Máte-li DOC.STY, můžete si LATEXem vytisknout dokumentaci.

Tímto způsobem jsem napsal do Zpravodaje článek o vytváření rejstříků. Tak bylo zajištěno, že příklad rejstříku byl vytvořen přesně stejnými makry, která jsem popisoval.

Teď se konečně dostáváme k důvodu, proč jsem psal program PERCENT. Dokumentace pro DOC.STY se nesnadno píše, pokud musíme pro-

centa doplňovat ručně. Navíc se tím ztěžují dodatečné zásahy a TEXspell nehledá překlepy v komentářích. Ve slušném editoru sice lze vytvořit klávesová makra, která přidávají nebo ruší procenta. Jenže my nechceme doplnit procenta do celého textu, ale do mnoha kratších kousků. Navíc musíme přesně dodržet zmíněné čtyři mezery — a to vše bez trochy automatizace vede k mnoha chybám. Program PERCENT jsem skutečně vytvořil proto, abych mohl napsat článek o tvorbě rejstříků, ovšem s vědomím, že jej budu používat i v dalších projektech. Tím jsme se od WEBu vrátili k tomu, čím se tento zpravodaj zabývá především, tj. k TEXu.

wagner@csearn
wagner@earn.cvut.cz

Recenze s přívažkem

JIŘÍ VESELÝ

Na rozdíl od knížek, které čtenáře varují na začátku první kapitoly a vybízejí ho k tomu, aby si přečetl předmluvu, já upozorňuji na odstavec začínající znamením •. Tam teprve recenze knížky

> George Grätzer: *Math into TEX. A Simple Introduction to AMS-LATEX*, Birkhäuser, Boston 1993, xxix + 294 str., 1 disk, brož., cena neznámá, ISBN 0-8176-3637-4 a také ISBN 3-7643-3637-4

opravdu začíná. Úvahy před můžete bez následků přeskočit, i když se týkají věcí kolem hlavního a jediného objektu recenzované knížky — AMS-LATEXu; snažím se upozornit na alespoň některé důležité souvislosti.

Kdybych byl stylový, psal bych tuto recenzi v AMS-LATEXu jako Grätzer svoji knížku. Konec konců každý zkušený uživatel TEXu však ví, že z kvality tištěné podoby článku nikdo nepozná, zda použitý „macro package“ je kvalitní — nakonec by to byla tedy jen výpověď o tom, že Karel Horák je schopen zvládnout přechod z tohoto poněkud exotického křížence dvou dosti odlišných „supermaker“ do stylu našeho Zpravodaje

(a byli by i tací, kteří by si byli jisti, že to nakonec přepsal, ale to by byli ti, co Karla málo znají). Co vlastně určuje kvalitu makra? Předcházející dvojí použití uvozovek signalizuje nedostatek přesnější terminologie: tak jako existují programy o několika řádcích i opravdu mamutí obludy okupující na disku desítky MB, lze i v TEXu produkovat užitečná i neužitečná makra extrémních velikostí. Z těch větších s širším použitím bylo vybráno několik k zařazení do současně šířené verze CSTEXu a mezi těmi několika naleznete i AMS-LATEX. Kolektivní rozhodnutí mu tedy přisoudilo přinejmenším punc nepominutelnosti.

LATEX Leslieho Lamporta alespoň částečně ovlivnil Michaela Spivaka při tvorbě AMS-TEXu (viz předmluvu k manuálu „*The Joy of TEX*“), nicméně obě makra se v mnoha ohledech liší. Obě mají svoje vášnivé zastánce i kritiky a jsou nesporně **velmi** užitečná. AMS-TEX prodělal nedávno omlazovací kůru (verze 2.0 a vyšší), na LATEXu 3 se už řadu měsíců intenzivně pracuje; zatímco AMS-TEX neopustil ruce svého tvůrce, mezi nejaktivnější členy teamu, formujícího budoucí LATEX 3, patří Frank Mittelbach a Rainer Schöpf, spolutvůrci AMS-LATEXu. Tolik k obsazení rolí.

Jestliže přiznávám svoji ne příliš velkou náklonnost k LATEXu, je na místě říci proč. To, co mne od něj odpuzuje, je jistá strnulost a jiná představa o tom, jak vypadá hezky sázený text. Musím ale říci, že je mi blízká část jeho filozofie. Není to přesně známé přirovnání výsledného produktu k rodinnému sedanu, ale souvislost LATEXu s logickou strukturou dokumentů. Myslím, že nebudu sám, kdo se na novou verzi těší, i když mu do dneška LATEX nijak nechyběl. Na druhé straně často píši texty v AMS-TEXu a pro psaní matematických článků ho považuji za výborný nástroj. Konstatuji, že při jeho poslední modernizaci byl učiněn krok směrem k zvýšení vazby na strukturu textu. To, co mi na něm nejvíce chybí, poskytuje AMS-LATEX. Kdybych musel v současné době psát delší text typu skript nebo učebnice, jistě bych o něm vážně uvažoval: je totiž k dispozici a je k němu nyní i slušný manuál. Má to ale přeci jen háček: zkrátka a dobře, i dvě velmi užitečné věci nemusí jít zcela ideálně dohromady, pokud jejich tvůrci pracovali nejen nezávisle, ale i s odlišnou koncepcí. Přesto si však myslím, že AMS-LATEX **spojuje** výhody obou. Jsem ale zároveň přesvědčen o tom, že LATEX 3 z něj udělá slabšího konkurenta či dokonce slepou uličku vývoje TEXu jako takového. O úspěšnosti projektu LATEX 3 nepochybuji, i když některé časové odhady beru s rezervou (ostatně původní Knuthův záměr byl věnovat TEXu sabathical year, rozhodně ne deset let života), a problém kompa-

tibility směrem dolů může nejen nové zájemce získat (typu těch, kteří LaTeXu dosud vzdorovali), ale jiné i trošku odradit.

• Grätzerova knížka vznikla v prostředí, odkud jsme již jednu populární pomůcku získali: Grätzer je kolegou Michaela Dooba na University of Manitoba, Canada. Ten mu také přispěl mnoha připomínkami stejně tak jako recenzenti pro nakladatele (Downes, Mittelbach, Freese).

Knížka má tři části: I. Krátký kurs, II. Volitelný kurs a III. Uživatelské úpravy. Doprovodná disketa 3,5″ má v odpovídajících adresářích cca 100 kB sw, na který má autorské právo vydavatelství Birkhäuser (myslím si, že jde o chybnou politiku). Jde o textové soubory, používané ve zmíněných částech; je patrně nejjednodušší přiblížit obsah jejich výčtem:

Part I obsahuje soubory: `article.tex, article.tpl, gallery.tex, gg.tpl, math.tex, mathB.tex, note1.tex, note1B.tex, note2.tex, topmat.tpl`

Part II obsahuje soubory: `article1.bib, article1.tex, inbibl.tpl, multicol.sty, multline.tpl, template.bib, textenv.tpl`

Part III obsahuje soubory: `article2.bbl, article2.bib, article2.tex, cform.tex, cform12.tex, distr.bat, macros02.tex, times.sty, timesC.sty`

Vedle toho obsahuje disketa ještě cca 1 MB sw v adresáři Mainz. Jsou v něm některá nová a rozšiřující makra pro LaTeX. Jde o soubory s © 1989–1992 Frank Mittelbach and Rainer Schöpf distribuované (naštěstí) za obvyklých podmínek. To se ale dozví čtenář teprve po rozlepení obalu diskety, takže se domnívá, že disketa obsahuje **pouze** sw, který bez písemného souhlasu nakladatelství nesmí býti kopírován (kromě záložních disket). Obsah a aktuálnost přiblíží znalcům LaTeXu označení verzí

```
array-v2_1b, doc-v1_7k, ftnright-v1_0d,
multicol-v1_4m, nfss-sep92, theorem-v2_1c
```

A nyní k obsahu knížky: je v podstatě čitelná bez jakékoli předchozí vědomosti o TeXu. Obsah má 8 stran, úvod obsahuje základní informace: jak si má počínat začátečník, jak pokročilejší čtenář — v částech II a III jsou graficky vyznačena nepominutelná fakta a také části pro letmé čtení. Kromě toho je zde krátký ukázkový článek včetně zdrojového souboru.

Stručný úvod (Part I) o 57 stranách nutně obsahuje mnoho základních informací o TeXu. Poměrně velká část (14 stran) je věnována instalaci;

popisuje se instalace PCTEXu (zvládá $\mathcal{AMS}$-LATEX od verze 3.14) na PC a Textures pro Macy, což pro prostředí v U.S. je patrně stále dosti typické. Druhá kapitola je návodem pro napsání prvního článku (mimochodem, pro $\mathcal{AMS}$ zůstává patrně převažujícím stylem $\mathcal{AMS}$-TEX).

Další část (Part II) má přes 130 stran. Nepominutelná fakta netvoří velkou část, protože však jsou graficky vyznačeny pouze začátky příslušných částí, je obtížné celkový rozsah odhadnout. Po pravdě řečeno, nedokázal bych asi napsat článek v $\mathcal{AMS}$-LATEXu pouze po přečtení první části, nemusí to však být stejné u všech matematických disciplín.

Další část (Part III) je věnována makrům (na uživatelské úrovni). Jedna kapitolka je věnována makrům v TEXu. V této části se projevuje poněkud zvláštní autorova touha po absolutní stručnosti (např. předefinování všech akcentů typu `\let\av=\v% check accent, ...`, „nadefinování řecké klávesnice“ typu

`\newcommand{\ga}{\alpha}`, ...
`\newcommand{\gy}{\psi}`,

apod.). Příjemné je vědět, užití kterých plainTEXových řídících slov se máme vyvarovat; příslušný výčet je proveden.

Dodatky opět nejlépe přiblíží výčet: A. Tabulky matematických symbolů, B. Tabulky textových symbolů, C. Základ $\mathcal{AMS}$-LATEXu (směs historie maker spolu s popisem prohlížení a tištění), D. Udržování kontaktů (o UNIXu, zacházení se sítí, TUGu a TUGboatu apod.), E. PostScriptové fonty.

Dodatek F se mi zdá velmi příjemný: obsahuje konverzní pokyny pro transformace textů pod $\mathcal{AMS}$-LATEX z plainTEXu, LATEXu a $\mathcal{AMS}$-TEXu. Nemyslím, že by byli uživatelé toho či onoho makra ve velké výhodě. Na ukázku si uveďme transformaci z plainTEXu. Je třeba:

- vzít v potaz již zmíněný výčet plainTEXových řídicích slov, kterých se musíme vyvarovat (45 položek, z nichž jsem však několik v životě nepoužil a některé se možná vloudily omylem: `\tabset`);
- přepsat úvodní část článku (nadpis, ...);
- nahradit všechny `$$` symboly `\[` a `\]`; jednoduché `$` lze — ne nutně — nahradit z estetických (?) důvodů `\(` a `\)`;
- předělat nadpisy sections, subsections ... a upravit (křížové) odkazy;
- předělat literaturu (i s ohledem na odkazy);
- předělat deklarované položky;
- předělat víceřádkové formule.

Dodatek G stojí rovněž za několik samostatných řádek. Obsahuje popis toho, co v knížce popsáno **není**. Nezbylo např. místo pro NFSS, ale čtenář je poučen o tom, kde se k informacím dostane. Vynechané věci jsou opět popsány ve skupinách (co z $\mathcal{AMS}$-LaTeXu, co z LaTeXu, co z TeXu). Krátce jsou popsány další knížky vhodné ke studiu. Čtenář tu nalezne i seznam archivů, kde lze hledat TeXware; informace o stuttgartském archivu neobsahuje žádnou informaci o emTeXu. Literatura zahrnuje 28 položek. Kvalitní rejstřík má 24 strany, o doprovodné disketě byla již shora řeč.

It's not a bird, it's not plain, it's .....

LaTeX2e – Preliminary Release Available

Leslie Lamport and the LaTeX3 project team
21 December 1993
The new release of LaTeX
is now available for testing.

LaTeX2e is the new standard version of LaTeX – prepared and supported by the LaTeX3 project team. It is upwardly compatible with LaTeX 2.09 documents, but contains new features.

Over the years many extensions of LaTeX have been developed. This is, of course, a welcome development, since it shows that the LaTeX system is in a healthy state. It has, however, had one unfortunate consequence: there are now several incompatible systems, in the sense of format (`.fmt`) files, all claiming to be LaTeX. Therefore, in order to process documents coming from various places, a site maintainer needs to provide several format files: LaTeX (with and without NFSS), SliTeX, $\mathcal{AMS}$LaTeX, and so on. In addition, when looking at a source file it is not always clear for which format the document was written.

LaTeX2e puts an end to this unsatisfactory situation—it will give access to all such extensions based on a single format and thus end the proliferation of mutually incompatible dialects of LaTeX2.09.

It uses an enhanced version (NFSS2) of the New Font Selection Scheme. Files such as `amstex.sty` (formerly the $\mathcal{AMS}$-LaTeX format) or `slides.sty` (formerly the SliTeX format) will become extension packages, all working with this single format.

The introduction of this new version will also make it possible to add a small number of often-requested features (such as extended versions of `\newcommand`).

To summarize:

- Standardisation: a single format incorporating NFSS2, to replace the present multiplicity of incompatible formats (NFSS, lfonts, pslfonts, etc.)
- Maintenance: a standardised system supported by a reliable maintenance policy.

LaTeX2e adheres, as far as possible, to the following principles:

- Unmodified version 2.09 document files can be processed with LaTeX2e.
- All new features of LaTeX2e conform to the conventions of version 2.09, making it as easy as possible for current users to learn to use them.

LaTeX2e is described in a new edition of 'LaTeX: A Document Preparation System' by Leslie Lamport (to appear during 1994) and 'The LaTeXCompanion' by Goossens, Mittelbach and Samarin, both published by Addison-Wesley.

LaTeX2e will be distributed twice a year. This distribution is a preliminary test release, and doesn't contain all of the files that will be part of the full release. In particular, it does not contain the planned extensions in the area of graphics inclusion.

The first full release will be available in Spring 1994.

This is a test release, so please get it and test it on as many different systems as possible!

LaTeX2e can be retrieved by anonymous ftp from the CTAN archives:

```
ftp.tex.ac.uk          /tex-archive/macros/latex/distribs/latex2e-test
ftp.shsu.edu           /tex-archive/macros/latex/distribs/latex2e-test
ftp.uni-stuttgart.de   /tex-archive/macros/latex/distribs/latex2e-test
```

Please report any problems with LaTeX2e by using the report-generating program `latexbug.tex`, included in the LaTeX2e distribution. Error reports can be sent to the following mail address:

`latex-bugs@rus.uni-stuttgart.de`

Note that no one on the programming team will be reachable until January 1994 (to give them a chance to see their families again). All mail sent to the above address will be answered then.

Seasons greetings to you all!
For the LaTeX3 Project

Johannes Braams
David Carlisle
Alan Jeffrey
Frank Mittelbach
Chris Rowley
Rainer Schöpf

Tisk v 600 dpi

Pokud jste si pořídili novou tiskárnu HP LaserJet IV a máte problémy s tiskem v hustotě 600 dpi podobně jako řada účastníků emTEXového listu, můžete si v následujících odstavcích přečíst postup, který radí autor programu `dvihplj` Eberhard Mattes. Pokud jste ovšem rovnou investovali trochu víc a zakoupili tiskárnu s interpretem PostScriptu, můžete samozřejmě k tisku použít převod do PostScriptu např. pomocí programu `dvips`.

Pokud PostScript nemáte, nebo z jiného důvodu potřebujete tisknout pomocí programu `dvihplj`, měl by do té doby, než EM vypustí už dlouho slibovanou verzi 1.4t, pomoci tento postup:

Nejprve vytvoříme soubor `600dpi.txt` s textem

```
ESC "&u600D"
```

z něhož vytvoříme soubor `600dpi.pcl` programem `makedot`

```
makedot -b 600dpi.txt 600dpi.pcl
```

V konfiguračním souboru (např. `lj.cnf`) nebo na příkazovou řádku (příp. do odpovídajícího batche) programu `dvihplj` použijeme následující volbu

```
/r600 /og600 /or2 /ob+ /pi:600dpi.pcl
```

To zapříčiní tisk v grafickém módu bez použití downloadu fontů na tiskárnu. Tato možnost by měla být umožněna až ve zmíněné nové verzi `dvihplj 1.4t`.

Pokud si budete generovat bitové mapy METAFONTem, je možno použít následující `mode`

```
mode_def laserjetIV =  % HP LaserJet IV (600 DPI)
proofing:=0;          % no, we're not making proofs
fontmaking:=1;        % yes, we are making a font
tracingtitles:=0;     % no, don't show titles in the log
pixels_per_inch:=600;
blacker:=0;           % The LaserJet is black enough
fillin:=.2;           % and it tends to fill in diagonals
o_correction:=.6;     %
enddef;
```

Karel Horák

Obsahy 3.–4. ročníku TUGboatu

Na následujících stránkách se po delší době opět vracíme k historickým číslům amerického TUGboatu. Dal jsem si konečně tu práci a přehlédl soubor s obsahy všech čísel prvních 12 ročníků (obsah 13. a právě ukončeného 14. ročníku ještě nemám v elektronické podobě úplný), doplnil chybějící definice a zformátoval je pro potřeby našeho časopisu. Tentokrát jsem si dal snad větší pozor na zrádnosti typu `\advance \hsize by 3pc`, které mě poněkud vypekly při tisku čísla 3/92. Celý soubor předám k distribuci do našich TEXových archivů. Po vytištění na laserové tiskárně dostanete svazeček o 60 stranách.

Karel Horák

TUGBOAT 3 (2) October 1982

TUGBOAT 4 (1) April 1983

TUGBOAT 4 (2) September 1983

Z obsahu příštího čísla

Petr Sojka: Grafika v TEXu (2)
Petr Olšák: Kouzla s programem MNU

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Bohumil Bednář
Počet výtisků:	600
Tisk:	HBT-Jet PRESS Neratovice
Adresa:	CSTUG MÚ UK, Sokolovská 83, 186 00 Praha 8

Podávání novinových zásilek povoleno
Ředitelstvím pošt Praha č.j. NP 320/1994 ze dne 10.2. 1994