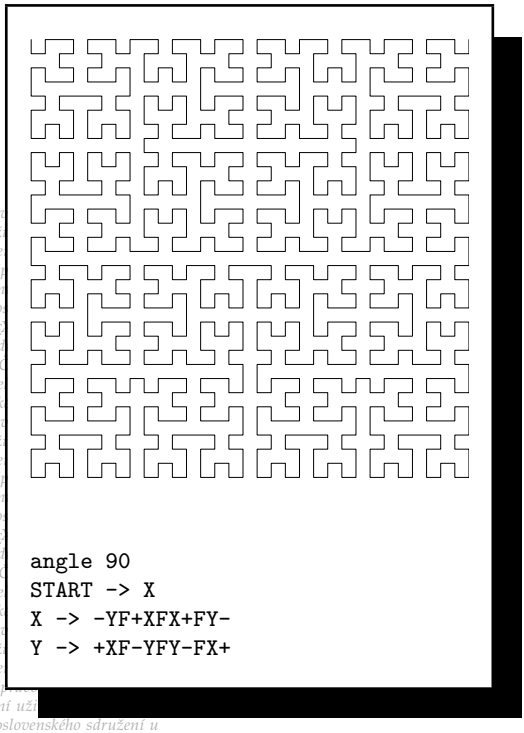
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1
2012

ISSN 1211-6661 Tištěná verze
ISSN 1213-8185 Online verze
Ročník 22

OBSAH

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupném archívu dostupném přes `http://www.cstug.cz/`.

Zpravodaj je zařazen do Seznamu recenzovaných neimpaktovaných periodik vydávaných v České republice, viz `http://www.vyzkum.cz/`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Postupujte podle instrukcí, které najdete na stránce `http://bulletin.cstug.cz/`. Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě, CD, či DVD na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3

`zpravodaj@cstug.cz`

Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Úvodník

Zdeněk Wagner

Právě se před vámi otvírají stránky prvního čísla Zpravodaje roku 2012. Sestavili jej z článků, dodaných v průběhu zmíněného a předchozího roku, členové mírně obměněné redakční rady a nový technický redaktor tohoto čísla.

V tomto čísle najdete jak články věnované přímo TEXu, tak články zabývající se grafikou. Jeden z nich je ponechán v původním, anglickém znění. Snahou redakce je především zveřejňování původních českých a slovenských článků, ale kvalitním textům v jazyce anglickém se nebráníme.

Při čtení článku Michela Charpentiera mě zmínka o tom, že PostScript je vytlačován formátem PDF, napadá trochu nostalgická myšlenka na dobu, kdy se vyráběly skutečné tiskárny, které měly PostScript implementován uvnitř, tímto jazykem komunikovaly s počítačem, a proto mohly spolupracovat s jakýmkoliv operačním systémem. Uživatel nemusel při koupi tiskárny přemýšlet o tom, zda pro jeho systém existuje ovladač a zda bude ovladač dostupný i v nové verzi operačního systému.

Statistiky užívání TEXu jednoznačně ukazují, že nejpreferovanějším formátem je LATEX. Přispívají k tomu i redakce odborných časopisů, které z TEXového světa přijímají výhradně LATEX a nabízejí balíčky pro své časopisy, některé jsou dostupné i na CTAN, odkud se dostávají do hlavních distribucí, jakými jsou TEXLive a MiKTEX. Petr Olšák však ukazuje, že formát CSplain vůbec není mrtvý, dokonce se vyvíjí, ale konzervativním způsobem, stále zachovává kompatibilitu. Ukazuje, jak lze přehledně napsaným makrem nahradit řadu LATEXových balíčků.

Časopis doplňuje kratší text předvádějící několik praktických postupů. Originál je opět cizojazyčný, ale tentokrát byl text přeložen do češtiny.

Redakce přeje všem příjemné okamžiky strávené čtením tohoto čísla.

Zdeněk Wagner
šéfredaktor

Použitie METAPOSTu ako knižnice

Martin Budaj

Článok popisuje použitie METAPOSTu ako knižnice (v tomto kontexte nazývanej MPlib) na tvorbu vektorovej grafiky pre programy písané v jazykoch C, C++ a Lua. V článku sú predstavené príklady, akým spôsobom je možné rozšíriť funkčnosť METAPOSTu (MPlib), ak je grafika vytvorená pomocou MPlib dodatočne spracovaná v aplikácii, ktorá MPlib používa (napr. opakované vzory, priehľadnosť, vrstvy, unikódové texty pomocou OpenType písiem).

Kľúčové slová: MPlib, METAPOST, C/C++, *post-processing* grafiky, knižnica.

Úvod

METAPOST je zvyčajne používaný ako samostatný program, ktorého výstupom sú grafické súbory vo formáte EPS alebo SVG. Od verzie 1.060, zverejnenej v roku 2008, je však možné použiť METAPOST ako knižnicu na spracovanie vektorovej grafiky pre programy napísané v jazykoch C, C++ alebo Lua. Na odlíšenie od názvu METAPOST je knižnica samotná nazvaná MPlib. Vývoj knižnice zabezpečuje Taco Hoekwater, projekt bol finančne podporený aj zo strany CSTUG.

Knižnica MPlib je v súčasnosti používaná samozrejme v METAPOSTe. Ďalším programom, ktorý používa MPlib je LuaTEX. V nasledovnom texte poukážeme na možnosti použitia MPlib vo vlastných programoch.

Základná dokumentácia k MPlib [4] popisuje rozhranie pre jazyky C a Lua. Dokument je stručný a nepokrýva všetky detaily použitia MPlib, môže byť preto často potrebné nahliadnuť aj do zdrojových súborov METAPOSTu. V poslednej časti článku ukážeme niekoľko tipov na použitie nezdokumentovaných častí MPlib.

V súčasnosti sú k dispozícii tri verzie METAPOSTu, resp. MPlib:

- *1.212:* stabilná verzia;
- *1.504 beta:* nepoužíva mem súbory s formátom (číta priamo definície v jazyku METAPOST, napr. `plain.mp`); statická alokácia pamäte je nahradená dynamickou;
- *1.750 pre-alpha:* ako alternatívu k aritmetike s pevnou desatinnou čiarkou umožňuje použiť aritmetiku s pohyblivou desatinnou čiarkou, čo rádovo zvyšuje rozsah prípustných číselných hodnôt.

Zo skúsenosti odporúčame použitie stabilnej verzie. Pri experimentovaní s použitím rôznych verzií MPlib v programe narazíme na drobné zmeny v API medzi jednotlivými verziami (viď posledná časť článku).

Princíp práce s MPlib je nasledovný: popis grafiky v jazyku METAPOST je knižnici odovzdaný v premennej typu reťazec, MPlib tento kód spracuje a (v prí-

 DOI: 10.5300/2012-1/2

pade, že je kód korektný) naplní štruktúry dostupné v jazyku C (resp. tabuľky v jazyku Lua), reprezentujúce kresbu. MPlib taktiež poskytuje podporné funkcie pre prácu s týmito štruktúrami.

Použitie knižnice

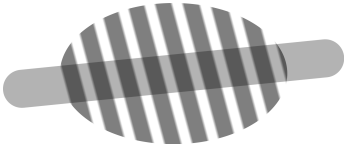
Jednou z výhod použitia MPlib ako grafickej knižnice vo vlastnom programe je nezávislosť na (ne)prítomnosti TEXu a METAPOSTu v systéme, prípadne na ich verziách.

Hlavnou výhodou však je, že v programe získame priamy prístup k interným grafickým štruktúram kresby. Je to dôležité najmä v prípade, ak nechceme použiť priamo grafické výstupy vo formáte EPS alebo SVG, ale chceme grafiku ďalej spracovať a zobraziť ju iným spôsobom. Dodatočné spracovanie grafiky môže slúžiť na doplnenie funkcionality, ktorú METAPOST nepodporuje (napr. opakované vzory, priehľadnosť objektov, gradientné výplne a pod.), resp. na konverziu do iného formátu.

Dodatočné spracovanie grafiky vytvorenej v METAPOSTe je možné aj pri tradičnom prístupe, kedy METAPOST zapíše grafiku do EPS súboru na disku. Ako príklad môžeme uviesť tvorbu máp jaskýň v programe Therion [3]. Mapa je v ňom rozdelená do mnohých fragmentov, z ktorých každý je v METAPOSTe spracovaný ako samostatný obrázok. Therion analyzuje vytvorené EPS súbory, doplní pravidelné vzory a priehľadnosti, konvertuje príkazy PostScriptu do PDF, fragmenty PDF vloží pomocou príkazu `\pdfliteral` do dokumentu spracúvaného pdfTEXom, ktorý vytvorí mapu alebo viacstranový atlas vo formáte PDF. Tento prístup, využívaný v Therione od jeho skromných začiatkov (por. [2]) až do súčasnosti, je však značne obmedzujúci. Použitie MPlib umožní omnoho efektívnejší priamy prístup k jednotlivým grafickým elementom kresby vytvorenej METAPOSTom.

Vlastné rozšírenie jazyka METAPOST môžeme dosiahnuť dvoma spôsobmi:

- použiť príkaz **special** *<text>*; ktorý v MPlib uloží *<text>* do špeciálneho objektu (a ktorý METAPOST zapíše na začiatok EPS súboru pri ukladaní na disk),
- v METAPOSTe 1.000 a novšom špecifikovať **withprescript** *<text>* alebo **withpostscript** *<text>* pri kresliacom príkaze (**fill**, **draw**). Na rozdiel od globálneho **special** *<text>*; je v tomto prípade *<text>* zapísaný tesne pred, resp. za konkrétny grafický element.



Obrázek 1: Elipsa vyplnená opakovaným vzorom s 50% priehľadnosťou. Šedá čiara je pod opakovaným vzorom.

Príklady rozšírenia funkčnosti METAPOSTu

Opakované vzory

Opakované vzory (*tiling patterns*) je možné v METAPOSTe používať vďaka inovatívnej práci P. Boleka [1]. Na označenie objektov, ktoré majú byť vyplnené vzorom, sú v ním navrhnutých makrách použité špeciálne farebné hodnoty (napr. $i \times \epsilon \times white$, kde $i = 1, 2, \ldots$ je poradové číslo vzoru a ϵ je najmenšie kladné číslo v METAPOSTe). Definície samotných vzorov a procedúry mapujúce špeciálne farby na opakované vzory sú zapísané na začiatok EPS súboru pomocou príkazu **special**. Vďaka skutočnosti, že jazyk PostScript umožňuje definovať a volať procedúry, môže interpreter jazyka PostScript pri zobrazení EPS súboru namapovať a zobraziť opakované vzory namiesto špeciálnych farieb, ktoré sú v súbore v skutočnosti použité.

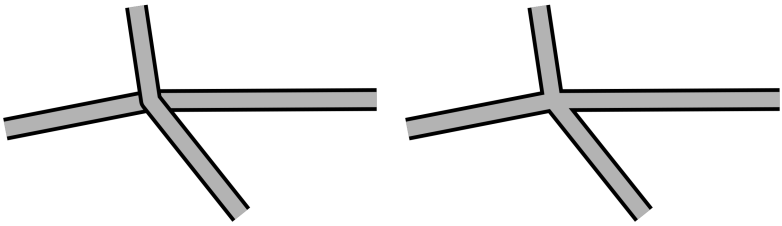
Túto metódu nie je možné analogicky použiť napríklad pri výstupnom formáte SVG, v ktorom nie je možné definovať procedúry zabezpečujúce mapovanie farieb na zodpovedajúce vzory. Pri použití MPlib môže bez problémov zabezpečiť mapovanie špeciálnych farieb na opakované vzory samotná aplikácia, ktorá MPlib využíva, a výsledok uložiť v ľubovoľnom formáte.

Používanie špeciálnych farieb kódujúcich použité vzory je v súčasnosti možné nahradiť napr. špecifikovaním **fill** <...> **withpostscript**(`"pattern:"` & *id*), kde *id* je reťazec s identifikátorom vzoru. Úvodná časť `pattern:` bude v tomto prípade volajúcej aplikácii slúžiť na identifikáciu typu špeciálneho reťazca. Tento prístup zároveň umožní nezávisle nastaviť opakovanému vzoru aj farbu.

Priehľadnosť

Ešte jednoduchším spôsobom je možné definovať priehľadnosť pre jednotlivé objekty, napríklad použitím **fill** <...> **withpostscript**(`"opacity:"` & decimal *o*), kde *o* je číselná hodnota. Opäť je na volajúcej aplikácii, aby v grafickej štruktúre vytvorenej METAPOSTom nahradila výskyty tohto špeciálneho označenia zodpovedajúcim nastavením priehľadnosti pre daný objekt.

Analogicky je možné vyznačiť aj skupiny priehľadnosti (*transparency groups*).



Obrázek 2: Križovatka dvoch ciest. V oboch prípadoch je najprv vykreslená východo-západná cesta, potom severo-juhovýchodná (vždy len jedenkrát). Obrázok vpravo využíva vrstvy na zmenu poradia grafických elementov v porovnaní s poradím v zdrojovom súbore.

Vrstvy

Dodatočné spracovanie môžeme využiť aj na podstatnejšie zásahy do vytvorenej grafiky ako len na jednoduché nahradenia uvedené vyššie. Zaujímavým príkladom môže byť automatické rozdelenie grafiky do vrstiev, ktoré môžu byť usporiadané v inom poradí, ako je poradie objektov vo vstupnom súbore.

Typickým príkladom použitia je mapový symbol pre cestu, tvorený šedým pásom s čiernymi okrajmi. Symbol je možné jednoducho vykresliť v dvoch krokoch: najprv širokým čiernym perom a následne tenším šedým perom po tej istej ceste, čím dosiahneme požadovaný grafický efekt. Problém nastáva pri križovaní dvoch ciest, kedy cesta vykreslená neskôr zanechá neželané artefakty (obr. vľavo). Pri použití preusporiadania sú najprv vykreslené široké čierne čiary pre obidve cesty, až potom tenšie šedé čiary (obr. vpravo).

Definícia symbolu v tomto prípade môže byť:

```
def cesta(expr p) =
    linecap := 0;
    draw p withpen pencircle scaled 3mm withcolor black
            withpostscript("level:1");
    draw p withpen pencircle scaled 2mm withcolor 0.7 * white
            withpostscript("level:2");
enddef;
```

Aplikácia spracúvajúca výstup MPlib musí samozrejme grafiku spracovať v toľkých prechodoch, koľko vrstiev bolo definovaných. V každom prechode vykreslí len tie objekty, ktoré do vrstvy prislúchajú.

Unikódové texty

MPlib nepodporuje zobrazenie textov uzavretých medzi **btex** ... **etex**, na sadzbu ktorých METAPOST (na rozdiel od MPlib) vie použiť (La)TEX. Veľmi dobrý výsledok je však možné dosiahnuť použitím niektorej zo špecializovaných knižníc na sadzbu unikódového textu, napr. Pango [6]. Pango nebude volané priamo z MPlib, ale z aplikácie, ktorá MPlib používa. Negatívom tohto prístupu je, že MPlib nemá k dispozícii informáciu o rozmeroch textu, takže nie je možné napríklad nakresliť okolo neho rámik (rozmery textu budú známe až volajúcej aplikácii, ktorá sa musí postarať o správne zarovnanie a ostatné transformácie textu).

Pri použití knižnice Pango je teda potrebné prostredníctvom operátoru **withpostscript** zapísať pomerne veľké množstvo parametrov: samotný text v kódovaní UTF-8, jeho polohu, zarovnanie, odsadenie, zväčšenie a použitý typ písma, ktoré naša aplikácia vyhodnotí a podľa ktorých text v knižnici Pango vysádza. Keďže v jazyku METAPOST musí byť operátor **withpostscript** viazaný na grafický objekt, môžeme namiesto textu zobraziť napríklad bodku, ktorú volajúca aplikácia síce bude ignorovať, avšak bude sa z nej dať zistiť informácia o prípadných transformáciách a použitej farbe.

Implementácia makra s podobným spôsobom použitia ako makro *label*, známe z Plain METAPOSTu, môže byť napríklad nasledovná:

```
vardef thepangolabel_@#(expr s, z) =
    save p; picture p;
    p = image(draw origin withpen pencircle scaled 1bp
        withpostscript("pangotext:" & s)
        withpostscript("pangooptions:"
            & decimal xpart z & ":"
            & decimal ypart z & ":"
            & decimal labxf_@# & ":"
            & decimal labyf_@# & ":"
            & decimal(labeloffset * xpart laboff_@#) & ":"
            & decimal(labeloffset * ypart laboff_@#) & ":"
            & decimal(10pt * defaultscale) & ":"
            & defaultfont));
    p
enddef;
def pangolabel = draw thepangolabel enddef;

defaultfont := "Linux Libertine O";
defaultscale := 1.2;

pangolabel_urt("Pango <i>text</i>", origin) rotated 30 withcolor red;
```

Značnou výhodou použitia knižnice Pango je jednoduchý a priamy prístup k OpenType fontom inštalovaným v systéme.

Grafický výstup

Aplikácia, ktorá používa MPlib, získa prístup k štruktúram obsahujúcim jednotlivé grafické elementy kresby. Po ich prípadnom dodatočnom spracovaní (ako bolo ukázané v príkladoch uvedených vyššie) je väčšinou potrebné kresbu zobraziť, alebo uložiť do grafického súboru.

Túto úlohu môže uľahčiť špecializovaná knižnica. Ako príklad môžeme uviesť knižnicu Cairo [5], ktorá umožňuje grafiku zobraziť alebo uložiť v rastrovom (PNG) alebo vektorovom (PDF, PS, SVG) formáte. Je zároveň dobre previazaná s knižnicou Pango.

Spojenie MPlib, Pango a Cairo umožňuje spracovanie komplexnej vektorovej grafiky pre programy v C alebo C++. Táto kombinácia bola použitá aj na vykreslenie ilustrácií v tomto článku.

Tipy a triky pri použití knižnice

Posledná časť článku poukazuje na nezdokumentované oblasti MPlib. Dokumentácia je miestami veľmi stručná a popisuje len stabilnú verziu. Pri použití knižnice je často potrebné nahliadnuť do zdrojových súborov `psout.w` alebo `svgout.w`.

Najmarkantnejším rozdielom verzie 1.750 oproti starším verziám je, že číselné hodnoty sú spravidla vyjadrené v PostScriptových bodoch. V starších verziách boli – kvôli použitiu celočíselnej aritmetiky – vyjadrené v *scaled points* (ktorých je 65536 v 1 PostScriptovom bode).

Ďalšou všadeprítomnou zmenou je zmena typu štruktúry, v ktorej MPlib ukladá (grafickú) cestu, z `mp_knot *` na `mp_gr_knot`, ako aj štruktúry samotnej (pre detaily viď súbory `mplib.h` a `mplibps.h`). Konverzia tejto štruktúry do podoby, v akej cestu zapisuje PostScript alebo SVG, nie je úplne triviálna; značnou pomocou môže byť inšpirácia funkciou `mp_gr_ps_path_out` a funkciami z nej volanými v súbore `psout.w`.

Rovnaká štruktúra je použitá aj na uloženie informácií o eliptickom pere, ak je ním cesta vykreslená. Spôsob získania týchto informácií najnázornejšie ilustruje funkcia `mp_svg_pen_info` zo súboru `svgout.w`. Parametre pera je však potrebné dodatočne upraviť (na vykreslenie eliptického pera sa používa lokálna transformácia súradnicového systému a je potrebné zabezpečiť, aby transformačná matica nebola singulárna). Na úpravu parametrov je možné použiť postup podľa funkcie `@<Tweak the transformation...@>` zo súboru `psout.w` (v stabilnej verzii METAPOSTu).

Zoznam literatúry

[1] Bolek, Piotr. *METAPOST and patterns*. TUGboat, 1998, roč. 19, č. 3, s. 276–283; dostupné na `http://www.tug.org/TUGboat/Articles/tb19-3/tb60bolek.pdf`

[2] Budaj, Martin. *METAPOST nielen na nakreslenie loga*. Zpravodaj Československého sdružení uživatelů TEXu, 1999, roč. 9, č. 4, s. 195–201. DOI 10.5300/1999-4/195

[3] Budaj, Martin – Mudrák, Stacho. *Therion – Digital Cave Maps*. Proceedings, 4th European Speleological Congress, Vercors, in: Spelunca, 2008, č. 33, s. 138 – 141, ISBN 978-2-900894-15-6; dostupné na `http://therion.speleo.sk/downloads/tharticle-vercors-2008.pdf`

[4] Hoekwater, Taco. *MPlib API documentation* 2008, 21 s. Dostupné v zdrojovej distribúcii METAPOSTu od verzie 1.090 (júl 2008).

[5] `http://cairographics.org`

[6] `http://pango.org`

Summary: Using METAPOST as a Library

The article informs about using METAPOST as a library (called MPlib in this context) for creating vector graphics in programs written in C, C++ or Lua languages. The article presents a few examples of extensions of the features of METAPOST (MPlib), which is possible if the graphics created by MPlib is post-processed in the application calling MPlib (e.g. tiling patterns, transparency, layers, unicode labels using OpenType fonts).

Key words: MPlib, METAPOST, C/C++, graphics post-processing, library.

Martin Budaj, `m.b@speleo.sk`
c/o CSTUG c/o FEL ČVUT, Technická 2
Praha, CZ-166 27, Czech Republic

Programujeme L-systémy v PostScriptu

Michel Charpentier

Abstrakt

I když se PostScript tradičně považuje za formát souborů pro popis grafiky, jedná se ve skutečnosti o plnohodnotný programovací jazyk rozšířený o grafické funkce. Jeho vyjadřovací schopnosti sahají mnohem dál než pouhý popis vektorové grafiky. PostScript umožňuje naprogramování řady různých druhů vypočtů, včetně složitých aritmetických operací. V tomto článku ukážeme jak používat rekurzivní funkce v PostScriptu k implementaci skupiny přepisovacích systémů nazývaných L-systémy. Pomocí těchto systémů můžeme psát jednoduché programy v PostScriptu, které kreslí jak klasické fraktály tak i zajímavé obrázky připomínající rostliny.

Klíčová slova: PostScript, L-systémy, fraktály.

PostScript as a Programming Language

PostScript [1] is a graphical description language which was widely used in the graphics and typesetting worlds but is slowly being supplanted by the newer *Portable Document Format* (pdf). One fascinating thing about PostScript is that, in addition to its graphical capabilities, it possesses many of the features found in more traditional programming languages, like variables, conditional and loops (which pdf does not). Indeed, there is enough in PostScript to compute everything that is computable (in more pedantic terms, the language is said to be "Turing-complete").

It is always fun to learn a new programming language and, in my case, it always begins by calculating prime numbers.[1] Although it is possible to use PostScript to generate tables of prime numbers—the fun[2] part being that the numbers are computed by the *printer*—the language is first and foremost a graphical language, which begs the question: What can be done with prime numbers *and* graphics? One answer is the Ulam Spiral [10], which places integers along a (squarish) spiral and marks prime numbers with dots to display interesting patterns.

Fig. 1 shows Ulam's Spiral with a dot for each prime number, in which diagonal patterns start to emerge. Fig. 2 shows a closeup of the center of the spiral. Both pictures are generated using a PostScript program [4] that draws the spiral, the

[1] I grew tired of saying *Hello* to a world that never said *Hello* back.

[2] It stops being fun as soon as there are enough people waiting for urgent printouts while the main department printer is busy crunching prime numbers.

doi: 10.5300/2012-1/9

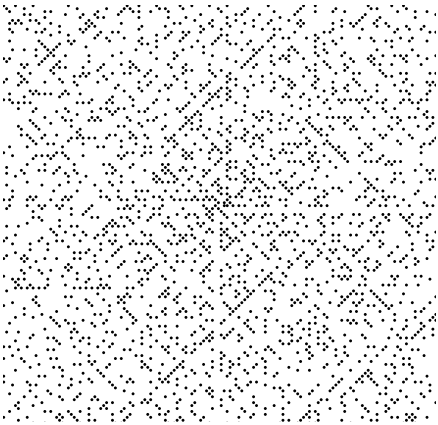


Figure 1: Ulam's Spiral.

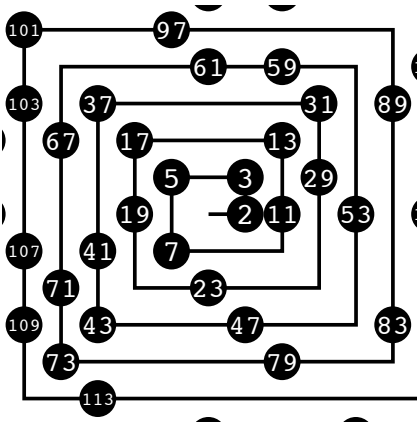


Figure 2: Ulam's Spiral (closeup).

dots, the numbers inside the dots and, more importantly, implements its own primality testing. Fig. 3 shows how trial division can be written in PostScript to implement a `prime?` function that tests whether a number is prime. PostScript also has arrays, which makes it possible to calculate prime numbers with the sieve of Eratosthenes, but the code is a bit longer (see [4] for details).

```
/prime? {
  /n exch def
  n 1 eq {
    false % 1 is not prime
  }{
    n 2 mod 0 eq {
      n 2 eq
    }{
      /divide? false def
      3 2 n sqrt {
        n exch div dup ceiling eq
          {/divide? true def exit} if
      } for
      divide? not
    } ifelse
  } ifelse
} bind def
```

Figure 3: Primality testing by trial division.

Without getting too much into PostScript syntax for now, a few things are worth noting. We see that the language has all the usual good stuff, like Booleans (`true`, `false`, `not`), conditionals (`if`, `ifelse`), loops (`for`, used here from 3 to

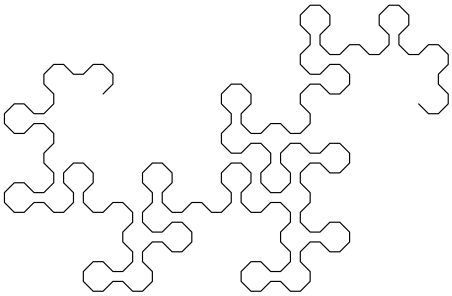
$\sqrt{n}$ by 2 increments), tests (`eq`) and variable names (`n`, `divide?`). We also see that it is based on a stack (`exch`, `dup`) and uses a postfix notation (as in (`n 1 eq`) or when pushing the condition, "then" block and "else" block on the stack before calling `ifelse`). This use of stack in a postfix manner does not make PostScript the easiest programming language to read.[3]

Fractals as L-Systems

An L-system [9, 7], named after the biologist Lindenmayer, is a model based on rewriting rules, most famously used to study the self-similarity found in plants [6]. Before going into plants (which require branching, as plants would), consider a (mythical) animal, the Dragon. The Dragon Curve [8] is a famous fractal that is obtained by repeatedly replacing a line segment by a "corner" made of two shorter line segments.

The Dragon Curve can be described as a L-system with three rewriting rules:[4] `X` $\rightarrow$ `-FX++FY-`, `Y` $\rightarrow$ `+FX--FY+` and `F` $\rightarrow \Lambda$ (where Λ represents the empty string to indicate that `F` symbols are removed). Starting with `X`, apply the rules N times, each time replacing *all* the `X`, `Y` and `F`. After N iterations, remove the remaining the `X` and `Y` and what is left is a string of `F`, `-` and `+`. If `F` means "move forward"; `-` means "turn 45° right"; and `+` means "turn 45° left", in the classic "turtle" interpretation, the string describes a curve. Intuitively, the rewriting rules implement a process by which a line segment is being removed (rule `F` $\rightarrow \Lambda$) and replaced by a "corner" below (rule `X`) or above (rule `Y`) it, in an alternating fashion. Fig 4 shows the first seven iterations, with the corresponding curves.

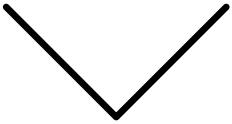
After seven iterations, and using "round corners" to make the curve easier to follow, it looks like this (and yes, this is a Dragon, not a French poodle):



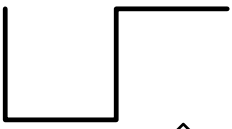
[3] Not that this would scare an experienced LaTeX user, I am sure.

[4] There are simpler L-systems for the Dragon Curve, described in [6], but the system chosen here has the benefit of drawing the Dragon always oriented in the same direction.

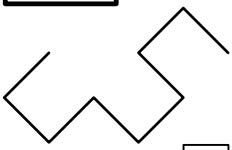
$N = 1$ `-F++F-`



$N = 2$ `-F++F-+++F-F+-`

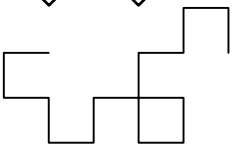


$N = 3$ `--F++F-+++F-F+-+++-F++F--+F-F++-`



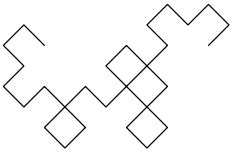
$N = 4$

```
--F++F-+++F-F+-+++-F++F--+F-F++-+++-F++F-+++
F-F+--+-F++F--+F-F+++-
```



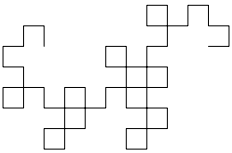
$N = 5$

```
---F++F-+++F-F+-+++-F++F--+F-F++-+++-F++F-++
+F-F+--+-F++F--+F-F+++-+++--F++F-+++F-F+-+++
-F++F--+F-F++--+-F++F-+++F-F+--+-F++F--+F-
F++++-
```



$N = 6$

```
---F++F-+++F-F+-+++-F++F--+F-F++-+++-F++F-+
++F-F+--+-F++F--+F-F+++-+++--F++F-+++F-F+-++
+-F++F--+F-F++--+-F++F-+++F-F+--+-F++F--+F-
-F++++-+++--F++F-+++F-F+-+++-F++F--+F-F++-+++
-F++F-+++F-F+--+-F++F--+F-F+++--+--F++F-+++
F-F+-+++-F++F--+F-F++--+-F++F-+++F-F+--+-F+
+F--+F-F++++-
```



$N = 7$

```
----F++F-+++F-F+-+++-F++F--+F-F++-+++-F++F-
+++F-F+--+-F++F--+F-F+++-+++--F++F-+++F-F+-+
++-F++F--+F-F++--+-F++F-+++F-F+--+-F++F--+F
-F++++-+++--F++F-+++F-F+-+++-F++F--+F-F++-++
+-F++F-+++F-F+--+-F++F--+F-F+++--+--F++F-++
+F-F+-+++-F++F--+F-F++--+-F++F-+++F-F+--+-F
++F--+F-F++++-+++---F++F-+++F-F+-+++-F++F--
+F-F++-+++-F++F-+++F-F+--+-F++F--+F-F+++-+++
--F++F-+++F-F+-+++-F++F--+F-F++--+-F++F-+++F
-F+--+-F++F--+F-F++++--+--F++F-+++F-F+-+++
-F++F--+F-F++-+++-F++F-+++F-F+--+-F++F--+F-
F+++--+--F++F-+++F-F+-+++-F++F--+F-F++--+-F
++F-+++F-F+--+-F++F--+F-F+++++-
```

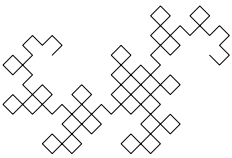


Figure 4: Dragon Curve, as generated by the system `X` $\rightarrow$ `-FX++FY-`, `Y` $\rightarrow$ `+FX-FY+`, `F` $\rightarrow \Lambda$.

Programming L-Systems in PostScript

Since PostScript is stack-based, it obviously supports recursive functions. The three rewriting rules of the Dragon Curve system can be implemented as three functions `X`, `Y` and `F`, where `X` and `Y` are mutually recursive. The PostScript program for the Dragon Curve is given below, with relevant comments.

First, the file starts with a special comment to indicate that it should be interpreted as PostScript.

```
%!PS
```

A variable `N` is then defined. This is the number of iterations we want to perform.

```
/N 7 def
```

Three functions `X`, `Y` and `F` are defined. They take as parameter the number of remaining iterations, which is pushed on the stack before each call. The idea is that, at the last iteration, when this counter reaches zero, the symbols (here, the function calls) are interpreted graphically: `F` draws a line segment while `X` and `Y` do nothing. When the counter is non-zero, the functions are interpreted as rewriting rules, triggering calls to more functions.

Function `X` tests the number of remaining iterations to see if it is zero. This is done by duplicating it, then comparing to zero. This way, if the top of the stack is $\langle 3 \rangle$, it becomes $\langle 3, \mathsf{true} \rangle$ and if it is $\langle 0 \rangle$, it becomes $\langle 0, \mathsf{false} \rangle$. If the count is zero, the function does nothing (it has no graphical counterpart). If the count is non-zero, `X` makes 7 function calls to functions `-`, `+`, `F`, `X` and `Y`. Functions `-` and `+` represent 45° rotations and do not need the iteration count. The count is thus pushed 4 times on the stack (for the 4 calls to `F`, `X` and `Y`) after having been decremented by one. After the calls, the function terminates by popping its parameter from the stack.

```
/X {
  dup 0 ne
  {1 sub 4 {dup} repeat - F X + + F Y -}
  if pop
} def
```

Function `Y` is similar to function `X`. Function `F` does nothing when the count is non-zero (corresponding to the $\mathtt{F} \to \Lambda$ rule). When the counter reaches zero, the function performs a graphical operation, namely drawing a horizontal line segment of length 10.

```
/Y {
  dup 0 ne
  {1 sub 4 {dup} repeat + F X - - F Y +}
  if pop
} def
```

```
/F {
  0 eq { 10 0 rlineto } if
} bind def
```

Although the line is always drawn horizontally, the graphics context is rotated by the `-` and `+` functions. Function `-` rotates it clockwise (right turn) and function `+` does it counterclockwise (left turn).

```
/- { -45 rotate } bind def
/+ { 45 rotate } bind def
```

Following are simple settings so lines have rounded tips and they join nicely.

```
1 setlinejoin
1 setlinecap
```

The call to `newpath` starts a new curve. We move to a carefully calculated location and we scale the picture by a factor equal to $\frac{50}{(\sqrt{2})^N}$, where N is the total number of iterations. Since the line segment have a constant length of 10, the Dragon ends up always having the same size, for any number of iterations.

```
newpath
220 180 moveto
50 N { 2 sqrt div } repeat dup scale
```

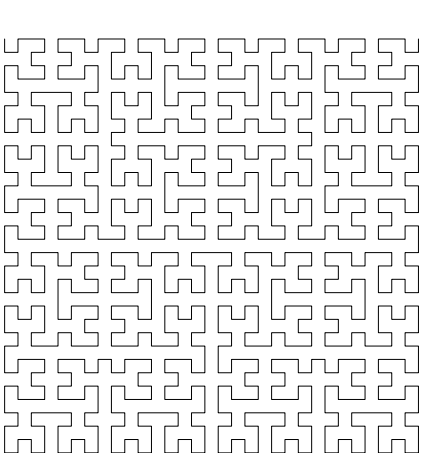
Finally, a 90° rotation places the page in landscape orientation and an initial call to function `X` is made with N on the top of the stack. The path that is built by the series of calls to `rlineto` is drawn as a line by `stroke` and the page is printed or displayed, depending on where the PostScript code is interpreted.

```
90 rotate
N X
stroke
showpage
```

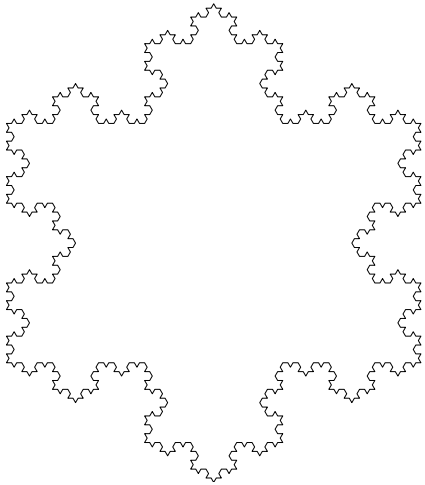
The full program can be downloaded from the Web [2]. Other well-known fractals, besides the Dragon Curve, can be represented as L-Systems with well-chosen rotation angles and rewriting rules. Many can be drawn with the same PostScript program by only changing a few lines. Fig. 5 shows the classic examples of Hilbert's curve and Koch's snowflake, with their associated L-systems.

Implementing Branching

Dragons—and, for that matter, French poodles—need trees and trees need branches. L-systems include a branching operator, usually represented with square brackets. Basically, '[' marks a point and ']' goes back to it. This allows the system to recursively build a branch (a subtree) before it continues from the trunk (or main branch) it left from. Fig. 6 shows examples of branching L-systems from [6] and their graphical representations. Note how some systems use nested branching. Colors are obtained here by using lighter shades of green



```
angle 90
START -> X
X -> -YF+XFX+FY-
Y -> +XF-YFY-FX+
```



```
angle 60
START -> +F--F--F
F -> F+F--F+F
```

Figure 5: Hilbert's curve and Koch's snowflake.

as the depth of the computation increases for a "realistic" effect, or are chosen randomly at branching points, each variant being straightforward to implement in PostScript (the language has a `rand` operator).

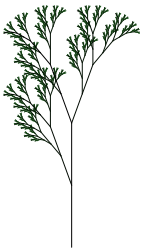
Branching can be implemented straightforwardly using PostScript's `gsave` and `grestore` operators. The first operator saves the current graphics context (including color and current point) by pushing it onto the stack; the second operator restores the graphics context from the stack. They result in the following PostScript implementation of '[' and ']' (square brackets are part of PostScript's syntax for arrays, so `B` and `E` are used instead):

```
/B { gsave } bind def
/E { stroke grestore } bind def
```

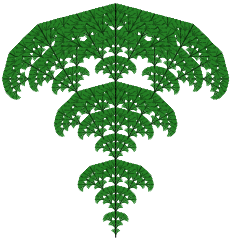
`B` saves the graphics context and `E` draws the current subtree (`stroke`) before restoring the context.

This implementation of branching in PostScript, however, turns out to be quite inefficient: When function `E` is invoked (at the tip of a branch), a path is being stroked all the way from the origin (the root of the tree). As a result, branches shared by many leaves are being drawn many times. This can be avoided by committing the path up to the branching point before branching. An alternate definition of `B` could be:

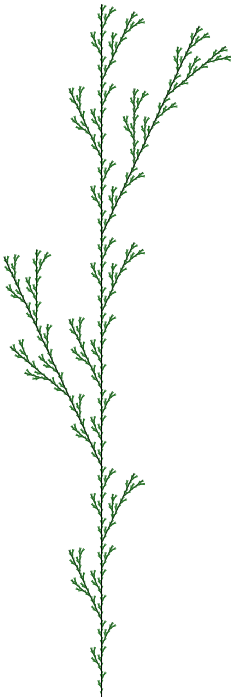
```
/B { currentpoint stroke moveto gsave } bind def
```



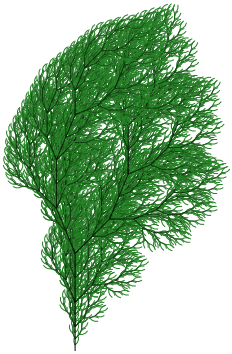
```
angle 20
START -> X
X -> F[+X]F[-X]+X
F -> FF
```



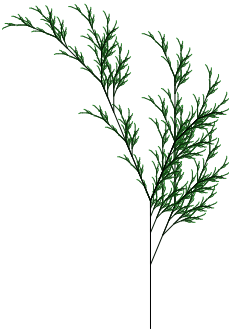
```
angle 30
START -> F
F -> F[+F[+F][-F]F][-F
     [+F][-F]F]F[+F][-F]F
```



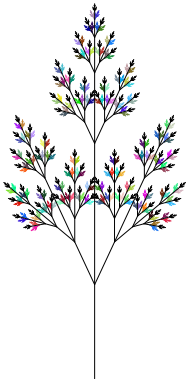
```
angle 25
START -> F
F -> F[+F]F[-F]F
```



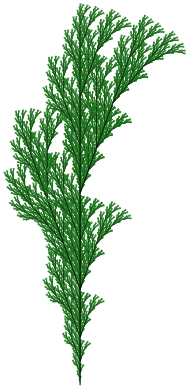
```
angle 22.5
START -> F
F -> FF-[-F+F+F]+[+F-F-F]
```



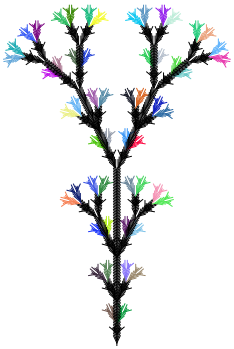
```
angle 22.5
START -> X
X -> F-[[X]+X]+F[+FX]-X
F -> FF
```



```
angle 25
START -> X
X -> F[+X][-X]FX
F -> FF
```



```
angle 20
START -> F
F -> F[+F]F[-F][F]
```



```
angle 25
START -> Y
X -> X[-FFF][+FFF]FX
Y -> YFX[+Y][-Y]
```

Figure 6: Plants as branching L-systems.

Figure 7: Single path versus multiple paths.

Function `B` now draws the path up to the current point and moves the origin of a new path to the branching point. The resulting program is faster, but branches are drawn as a series of successive strokes, which does not always look as nice. Fig. 7 shows a closeup of a branching point, with the new implementation of branching on the right. To give the user a choice between faster computations or nicer graphics, one can define a Boolean flag within the PostScript program. It could be inefficient, however, to test this flag at every branching point. An alternative approach is to use the flag to *build* variants of the branching function. This strategy can also be used to choose a coloring scheme once and for all, without the need for further testing when the drawing takes place. PostScript was to some extend inspired by Lisp and like Lisp, it offers ways to dynamically build blocks of code to be later evaluated (basically, a block is just an array that is flagged as executable). The resulting implementation of `B` is as follows:

```
/B [
  fast? {
    {currentpoint stroke moveto}
    aload pop
  } if
  {gsave} aload pop
  currentdict /color known {
    {dup color}
    aload pop
  } if
] cvx bind def
```

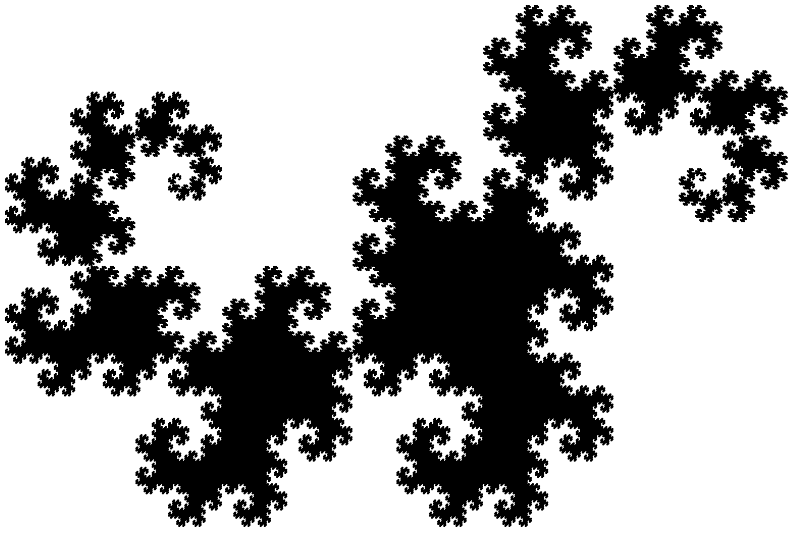
The code tests the Boolean `fast?` and looks up a `color` function to build an array that is then given the executable property by the `cvx` operator. There is no further testing of `fast?` or look-up of `color` when function `B` is executed

(i.e., when a branching point is reached). If no `color` function is defined, no function call takes place and the drawing remains black. A complete program, with branching and a few different color schemes can be downloaded from the Web [3].

With branching L-systems and imaginative coloring schemes, short PostScript programs can be written that produce impressive looking plants. One thing the examples from fig. 6 show is that L-systems that are almost identical can result in very different looking plants. One can therefore study the effects of mutation-like variations to an L-system to evaluate what happens to the corresponding fractal (see [5] for a system that implements such random mutations).

Conclusions

There is a lot more to L-systems, including fancy operators not described here, generalizations to 3D graphics, and theoretical studies of their expressive power. Sadly, though, PostScript is being superseded by the *Portable Document Format* which, in spite of its qualities as a document exchange format, lacks the programming capabilities that make PostScript so remarkable. The Web is full of Java applets that implement L-systems, but it is just not the same thing. When PostScript is gone, how will we use all those CPU cycles wasted in printers?



References

[1] Adobe Systems Incorporated. *PostScript Language Reference*. Addison-Wesley, third edition, February 1999.

[2] Michel Charpentier. Dragon Curve in PostScript. `http://www.cs.unh.edu/~charpov/Programming/L-systems/simple-dragon.ps`.

[3] Michel Charpentier. L-systems in PostScript. `http://www.cs.unh.edu/~charpov/Programming/L-systems/plant2.ps`.

[4] Michel Charpentier. Ulam's Spiral in PostScript. `http://www.cs.unh.edu/~charpov/Programming/PostScript-primes/primes-distribution.ps`.

[5] Jim Lund. DoodleTron (a L-system Iterator). `http://elegans.uky.edu/jiml/lsystem/ls_index.html`.

[6] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[7] Pavel Tišnovský. L-systémy: přírodní objekty i umělé artefakty. `http://www.root.cz/clanky/l-systemy-prirodni-objekty-i-umele-artefakty`.

[8] Eric W. Weisstein. Dragon Curve. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/PrimeSpiral.html`.

[9] Eric W. Weisstein. Lindenmayer Systems. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/LindenmayerSystem.html`.

[10] Eric W. Weisstein. Ulam's Spiral. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/PrimeSpiral.html`.

Summary: Programming L-Systems in PostScript

Although we tend to think of PostScript as a file format used to describe graphics, it is in reality a full-fledged programming language with graphical capabilities. Thus, the power of PostScript goes far beyond that of simple vector-graphics formats. All sorts of computations can be programmed, including complex arithmetic calculations. In this paper, we show how to use recursive functions in PostScript to implement a family of rewriting structures known as L-systems. Based on these systems, one can write short PostScript programs that draw classic fractals and beautiful plant-like pictures.

Keywords: PostScript, (Lindenmayer) L-systems, fractals, unconventional programming languages.

Michel Charpentier, `charpov@cs.unh.edu`
University of New Hampshire, Computer Science Department
Kingsbury Hall, rm N215A, Durham, NH 03824
The United States of America

OPmac – makra rozšiřující možnosti plainTEXu

Petr Olšák

OPmac je balík jednoduchých doplňujících maker k plainTEXu umožňující uživatelům základní LATEXovou funkcionalitu: změny velikosti písma, automatickou tvorbu obsahu a rejstříku, práci s `bib` databázemi, referencemi, možnost proložení referencí hyperlinkovými odkazy atd. V tomto článku jsou ukázány významné vlastností makra OPmac. Soubor maker a úplná uživatelská i technická dokumentace jsou ke stažení na `http://petr.olsak.net/opmac.html`.

Úvod

V říjnu roku 2012 jsem se rozhodl, že společně s novým $\mathcal{CS}$plainem zveřejním svá makra, která už léta používám pro nejrůznější účely. Stačilo je trochu pročistit, přidat k nim uživatelskou a technickou dokumentaci a zveřejnit je. Základním záměrem bylo nabídnout uživatelům plainTEXu sadu maker, která řeší nejčastěji se vyskytující úlohy. Aby nemuseli pořád nanovo vymýšlet kolo. Vedlejším efektem je demonstrace faktu, že TEXová makra lze dělat jednoduše a účelně, což je fenomén, který nebývá v LATEXu obvyklý.

Při tvorbě balíku OPmac jsem se řídil následujícími zásadami:

- V jednoduchosti je síla.
- Makra nejsou univerzální, ale jsou čitelná a srozumitelná.
- Uživatel si makra může snadno předefinovat k obrazu svému.

Každé makro je napsáno s cílem co největší srozumitelnosti pro lidi, kteří to budou chtít číst, porozumět tomu a měnit to. Balík nabízí čtenáři inspiraci, jak se programují TEXová makra. Z kódu maker je cítit jistá elegance. Technická část dokumentace k OPmac by mohla sloužit jako učebnice programování TEXových maker. To je zásadní rozdíl od koncepce LATEXu. Když se člověk podívá do LATEXového souboru `latex.ltx`, vystřeví se na něj množství zavináčů a makra, ze kterých je často cítit topornost a mnohdy nepochopení vnitřní koncepce TEXu. Skoro nikdo se v tom nevyzná. Soubor `latex.ltx` obsahuje 8000 řádků a schopnosti LATEXu jsou navíc ukryty v desítkách či stovkách různých dalších makro souborech. Naproti tomu v OPmac vidíte vše pohromadě a názorně v jednom malém souboru. V některých věcech OPmac výrazně překračuje možnosti běžného [1] i rozšířeného [2] LATEXu: generování rejstříků bez externího programu, práce s předgenerovanou databází bibliografických záznamů, listingy externích souborů, pohodlnost a účelnost, s jakou se makra ovládají na uživatelské úrovni.

 doi: 10.5300/2012-1/20

Balík OPmac nabízí podobně jako LaTeX autorům textů *rozhraní*, tj. smluvenou sadu značek na vymezení struktury dokumentu. Je jiná, než v LaTeXu, možná umožní napsat zdrojový text článku poněkud přehledněji a oku více lahodícím způsobem. Balík OPmac ovšem neřeší typografický vzhled dokumentu. Bez doplňujících maker vyleze jednoduchý střízlivý dokument. Předpokládá se, že autor dodatečných plainTeXových maker ušije vzhled dokumentu na míru konkrétnímu požadavku. Například pro nastavení vzhledu tohoto článku podle typografie časopisu Zpravodaj stačilo dopsat 30 doplňujících řádků maker.

Uživatelům OPmac nabízím konzultace po emailu a uvítám hlášení o chybách. Pojďme společnými silami odstranit případné mouchy a mušky.

V následujícím textu je přepsána větší část uživatelské dokumentace balíku OPmac. Vážní zájemci o OPmac si mohou přečíst i technickou dokumentaci dostupnou na síti, která podrobně popisuje interní řešení jednotlivých maker.

Záhlaví dokumentu

Na začátek dokumentu můžete napsat třeba toto:

```
\input opmac      % zavedení makra OPmac
\chyph            % zapnutí češtiny
\input lmfonts    % Latin Modern fonty (od verze CSplainu Nov.2012)
\typosize[12/14]  % nastavení základní velikosti sazby
```

Velikosti fontů a řádkování

Všechna makra popsaná v této sekci nastavují změny ve fontech a dalších parametrech jen lokálně, takže po ukončení skupiny se nastavení vrací k původním hodnotám.

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

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

Začátek dokumentu tedy může vypadat takto:

```
\input opmac \typosize[11.5/13]  % stupeň písma 11.5pt, řádkování 13pt
```

Makro `\typoscale[`⟨*faktor-font*⟩`/`⟨*faktor-řádkování*⟩`]` zvětší nebo zmenší velikost textových i matematických fontů resp. řádkování ⟨*faktor*⟩krát aktuální velikost fontů resp. řádkování. Faktor je celé číslo, přitom 1000 znamená jedničku

(jako za slovem `scaled` v příkazu `\font`). Je-li parametr prázdný, je to stejné, jako by byl roven 1000.

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

Tato makra můžeme použít pro nadpisy nebo poznámky s tím, že velikost bude relativní vzhledem k základnímu písmu celého dokumentu.

Pokud zavedete font příkazem `\font\prepinac=`⟨*metrika*⟩ (následovaný případným `at` nebo `scaled`) pak `\prepinac` přepíná do pevně stanoveného fontu, který není ovlivněn makry na nastavování velikosti. Ale co není, může být. Stačí font registrovat pomocí `\regfont\prepinac` a nyní i `\prepinac` přepíná do fontu podle velikosti nastavené pomocí `\typosize` nebo `\typoscale`. Příklad:

```
\font\zapfchan=pzcmi8z  \regfont\zapfchan
\typosize[20/] Taky \zapfchan přepne do Zapf-Chancery velikosti 20pt.
```

Při inicializaci $\mathcal{CS}$plainu je registrováno pět fontových přepínačů ukrytých v makrech `\rm`, `\it`, `\bf`, `\bi`, `\tt`. Takže tato makra implicitně nastavují font do stanovené velikosti.

Na místo `\typosize` a `\typoscale` je možno použít makra na změnu velikosti jen textových fontů `\textfontsize[`⟨*velikost*⟩`]` a `\textfontscale[`⟨*faktor*⟩`]`. Tato makra nemění matematické fonty ani řádkování.

Makra `\thefontsize[`⟨*velikost-fontu*⟩`]` nebo `\thefontscale[`⟨*faktor*⟩`]` změní velikost jen aktuálního fontu, nemění žádné jiné fonty a nemění řádkování. Tato makra používají interní fontový přepínač `\thefont`. Je možné si jej „vypůjčit“ příkazem `\let` pro pozdější použití:

```
\tenrm \thefontscale[700]\let\sevenrm=\thefont \tenrm
```

Díky přepínači `\tenrm` na konci ukázky sazba dále pokračuje neškálovaným fontem, ovšem škálovaný font máme uložen v přepínači `\sevenrm`.

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

Poslední poznámka se týká makra `\em`, které sice nepřepíná velikost fontů, ale přepíná jejich variantu. Je to kontextové makro, které pracuje v závislosti na aktuálně zvoleném fontu. Implicitně přepíná na `\it`. Pokud ale je aktuálním fontem `\it`, přepne na `\rm`. Je-li aktuálním fontem `\bf` přepne na `\bi` a obráceně. Makro navíc správně doplní italické korekce ke slovu před jeho použitím a za jeho použitím. Takže se o italické korekce není nutno starat. Příklad:

```
To je {\em zdůrazněný} text.     %jako: To je {\it zdůrazněný\/} text.
\it To je {\em zdůrazněný} text. %jako: To je\/ {\rm zdůrazněný} text.
\bf To je {\em zdůrazněný} text. %jako: To je {\bi zdůrazněný\/} text.
\bi To je {\em zdůrazněný} text. %jako: To je\/ {\bf zdůrazněný} text.
```

Členění dokumentu

Dokument se může skládat z kapitol, kapitola ze sekcí a sekce z podsekcí. Titul dokumentu vyznačte pomocí `\tit` ⟨*titul*⟩⟨*prázdný řádek*⟩, kapitolu zahajte `\chap`⟨*titul*⟩⟨*prázdný-řádek*⟩ a podobně novou sekci je možné zahájit makrem `\sec`⟨*titul*⟩⟨*prázdný-řádek*⟩ a podsekci `\secc`⟨*titul*⟩⟨*prázdný-řádek*⟩. Takže třeba:

```
\chap Brouci

\sec  Chrousti

\secc O nesmrtelnosti chroustů

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

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

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

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

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

Další číslované objekty a odkazy na ně

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

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

$$a^2 + b^2 = c^2 \eqmark$$

Je-li potřeba očíslovat jedním číslem více rovnic sestavených například pomocí `\eqalignno`, pak použijte `\eqmark` v posledním sloupci, například takto:

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

Ukázka dává tento výsledek:

$$\begin{aligned} a^2 + b^2 &= c^2 \\ c &= \sqrt{a^2 + b^2} \end{aligned} \tag{2}$$

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

```
\hfil\table{rl}{věk   & hodnota \crl\noalign{\smallskip}
                0-1  & neměřitelná \cr
                1-6  & projevující se \cr
                6-12 & výrazná \cr
               12-20 & extrémní \cr
               20-40 & mírnější \cr
               40-60 & různá \cr
               60-$\infty$ & umírněná} % vytvoření tabulky
\par\nobreak\medskip
\caption/t Závislost závislosti na počítačích na věku.
```

Tato ukázka vytvoří:

věk	hodnota
0–1	neměřitelná
1–6	projevující se
6–12	výrazná
12–20	extrémní
20–40	mírnější
40–60	různá
60–∞	umírněná

Tabulka 1 Závislost závislosti na počítačích na věku.

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

Způsob číslování lze změnit jinou definicí makra `\thednum` (pro rovnici), `\thetnum` (pro tabulky) a `\thefnum` (pro obrázky).

Makro OPmac vloží slovo „Tabulka“ v závislosti na nastaveném jazyce příkazem `\chyph`, `\shyph`, `\ehyph`. Při `\shyph` dostaneme „Tabuľka“ a při `\ehyph` máme

„Table“. Podobně se chovají slova „Obrázek/Obrázok/Figure“ a „Kapitola/Kapitola/Chapter“. Jiná automaticky generovaná slova OPmac nepoužívá.

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

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

LaTeXoví uživatelé jsou zvyklí, že jim tabulky a obrázky plavou v dokumentu, přičemž inklinují k horní části stránky. To se při použití OPmac implicitně neděje, ale je možno plavání zařídit pomocí plainTeXového makra `\topinsert` resp. `\midinsert`. Například:

```
\topinsert
   \hfil\table{rl}{...} % vytvoření tabulky
   \medskip
   \caption/t Závislost závislosti na počítačích na věku.
   \par
\endinsert
```

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

```
\label[chroust] \sec O nesmrtelnosti chroustů

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

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

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

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

Odrážky

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

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

Příklad:

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

vytvoří následující výstup:

1. Tady je první myšlenka.
2. A tady druhá, která je rozdělena na

   a) podmyšlenku
   b) a hned následuje další podmyšlenka,
   c) poslední podmyšlenka.

3. Tady je třetí myšlenka.

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

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

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

Tvorba obsahu

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

Nadpis pro obsah většinou nebudeme psát pomocí `\chap` nebo `\sec`, protože by se údaj o obsahu dostal do obsahu, což obvykle není žádoucí. Je proto potřeba použít vnitřního makra `\printsec` a před jeho použitím vyprázdnit obsah `\thesecnum`, abychom neměli tuto sekci číslovanou. Takže obsah vyrobíme třeba takto:

```
\def\thesecnum{}\printsec{\unskip Obsah}\maketoc
```

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

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

a mnoho dalších. Není možné ale předvídat všechno, co může uživatel nacpat do titulu sekce nebo kapitoly. Pokud se tedy „rozsype“ REF soubor, je potřeba si tímto způsobem zabezpečit používané makro.

Sestavení rejstříku

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

Pro zanesení slova do rejstříku slouží makro `\ii`. Je to zkratka za „insert to index“. Jeho parametr je ⟨*slovo*⟩ bez mezery ukončené mezerou (obecnější tvar parametru uvedeme později). Toto slovo se přepíše do rejstříku, ve kterém jsou

všechna takto deklarovaná slova seřazena podle abecedy a jsou k nim připojena čísla stránek, na kterých bylo použito odpovídající makro `\ii` ⟨*slovo*⟩. Příklad:

```
Tady mluvím o jistém
\ii rezistor
rezistoru, který provokoval moji zvědavost.
```

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

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

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

```
Zásady jsou \iid nestrannost , \iid pravdomluvnost a \iid odvaha .
```

To povede ke stejnému výsledku jako

```
Zásady jsou \ii nestrannost nestrannost,
\ii pravdomluvnost pravdomluvnost a \ii odvaha odvaha.
```

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

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

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

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

Dostaneme totéž jako při `\ii lineární~prostor \ii vektorový~prostor` a tato ukázka demonstruje ještě jednu věc: je-li potřeba do parametru `\ii` dostat mezeru, pište vlnku.

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

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

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

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

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

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

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

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

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

Do rejstříku musejí být zařazena jen „čistá“ slova, která neobsahují makra expandující na primitivní příkazy TEXu. Pokud chcete vytisknout v rejstříku něco komplikovanějšího, můžete sestavit slovník výjimek pomocí sady maker `\iis` ⟨*heslo*⟩⟨*mezera*⟩`{`⟨*tisk*⟩`}` (název makra můžeme číst jako `\ii` speciální). Funkci si vysvětlíme na příkladu:

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

Po takové deklaraci je možné psát `\ii relax` nebo `\ii chikvadrat` nebo třeba `\ii Goedelova/věta/o~neúplnosti,@`. OPmac abecedně řadí podle těchto hesel, ale když dojde na potřebu heslo vytisknout do rejstříku, vytiskne místo těchto hesel materiál, který je uveden na pravé straně slovníku. Příklad ukazuje, že tím lze řešit nejen tisk hesel, která je potřeba ošetřit speciálními makry (v příkladu slovo relax), ale také výjimky abecedního řazení (písmeno `\"o` nemáme v české abecedě, kdoví, kam by se to zařadilo). Slovník výjimek je možný zapsat kamkoli před `\makeindex`, typicky se píše na začátek dokumentu.

Výjimku z řazení dvojhlásky ch (například ve slově mochnátý, tj. mnohonohý) je možné zařídit pomocí tečky, která má stejně jako ostatní interpunční znaky,

nulovou řadicí platnost (OPmac hesla řadí, jakoby tam interpunkce nebyla). Takže třeba takto:

```
... \ii moc.hnátý ...
\iis moc.hnátý {mochnátý}
```

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

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

Barvy, vodoznaky

Makra uvedená v této sekci nastavují barvy jen při použití pdfTEXu a při výstupu do PDF. Jinak tato makra neudělají nic.

Barvu textu můžete nestavit pomocí přepínačů `\Blue`, `\Red`, `\Brown`, `\Green`, `\Yellow`, `\White`, `\Grey` a `\Black`. Je potřeba si uvědomit, že tyto přepínače pracují globálně nezávisle na skupině uvnitř boxu i mimo, prostě kdekoli. Barvu jinou než černou je potřeba nakonec vypnout přepínačem `\Black`. Pokud barvu jinou než černou zapnete na některé stránce a text v této barvě přeteče do stránek dalších, při opakovaném TEXování se objeví správná barva i na následujících stránkách. Tuto skutečnost si totiž TEX musí ujasnit prostřednictvím pomocného souboru s odkazy.

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

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

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

Následuje příklad, kterým vytvoříme podbarvený text:

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

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

Klikací odkazy

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

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

Poslední z uvedených odkazů je externí a bude mít barvu ⟨*color-out*⟩, zatímco ostatní čísla jsou interními odkazy a budou mít barvu ⟨*color-in*⟩. Příklad:

```
\hyperlinks{\Blue}{\Green}   % vnitřní odkazy modré, URL zelené
```

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

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

Implicitně tato makra nejsou definována, což znamená, že se rámečky netvoří.

Cíl odkazu lze specifikovat makrem `\dest[`⟨*typ*⟩`:`⟨*lejblík*⟩`]` a klikací text makrem `\link[`⟨*typ*⟩`:`⟨*lejblík*⟩`]{`⟨*color*⟩`}{`⟨*text*⟩`}`. Parametr ⟨*typ*⟩ je typ odkazu (toc, pg, cite, ref nebo další). Parametr ⟨*výška*⟩ určuje vzdálenost cíle nad účařím.

Makro `\url{`⟨*text*⟩`}` vytiskne odkaz do internetu. Text je psán strojopisem a může se lámat do řádků za lomítky. Například `\url{http://petr.olsak.net}` vytvoří `http://petr.olsak.net`. Je-li nastaveno `\hyperlinks`, stává se tento text aktivním vnějším odkazem.

Obsah dokumentu se dá přesunout do levé záložky PDF prohlížeče tak, že klikáním na něj se přechází v dokumentu na požadované místo. Ve specifikaci PDF se tomu říká „outlines“. Makro, které uvedenou věc zařídí, se jmenuje `\outlines{`⟨*úroveň*⟩`}`. Záložky budou implicitně rozevřeny do ⟨*úrovně*⟩ včetně. Takže při ⟨*úroveň*⟩=0 jsou vidět jen úrovně kapitol, při ⟨*úroveň*⟩=1 vidíme i sekce a při ⟨*úroveň*⟩≥2 vidíme rozevřeno všechno. Bohužel písmo v záložkách typicky

nezvládá správně česká a slovenská písmena. Proto OPmac konvertuje texty do záložek tak, že tam jsou bez hacku a carek. Chcete-li vypnout tuto konverzi, napište `\def\cnvhook#1{}`.

Samotný řádek do záložek vložíte makrem `\insertoutline{`⟨*text*⟩`}`. Text v tomto případě nepodléhá konverzi. V sazbě se neobjeví nic, jen se stane cílem, když uživatel na záložku s ⟨*textem*⟩ klikne. Obsah se do záložek vloží celý během činnosti makra `\outlines`, takže další řádky vložené pomocí `\insertoutline` tomuto obsahu předcházejí nebo následují podle toho, zda předcházejí nebo následují místu, kde je použito `\outlines`.

Verbatim texty

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

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

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

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

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

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

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

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

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

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

Verbatim listingy je možné tisknout z externího souboru. Například

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

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

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

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

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

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

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

Jednoduché tabulky

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

```
\table{||lc|r||}{\crl
                Měsíc     & zboží     & cena\hfil  \crli \tskip.2em
```

```
leden     & noťas     & 14 kKč     \cr
únor      & skejt     &  2 kKč     \cr
červenec  & jachtička & 3,4 MKč    \crl}
```

Uvedený příklad povede na následující výsledek:

Měsíc	zboží	cena
leden	noťas	14 kKč
únor	skejt	2 kKč
červenec	jachtička	3,4 MKč

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

Místo symbolu pro konec řádku `\cr` je možno použít `\crl` (přidá jednoduchou vodorovnou čáru) nebo `\crll` (přidá dvojitou čáru), `\crli` (přidá čáru přerušenou svislými dvojitými linkami) a `\crlli` (přidá dvojitou čáru přerušenou svislými dvojitými linkami). Těsně za `\cr`, `\crl` atd. může následovat `\tskip`⟨*dimen*⟩, což vytvoří vertikální mezeru velikosti ⟨*dimen*⟩, přitom se nepřeruší svislé čáry v tabulce.

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

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

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

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

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

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

OPmac definuje `\strut` závislý na zvoleném řádkování (při použití příkazu `\typosize`) zhruba takto:

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

Tip: zkuste `\def\tabiteml{$\enspace} \def\tabitemr{\enspace$}`. Ty dolary způsobí, že každá datová položka bude zpracována v matematickém módu. Makro `\table` se nyní podobá LaTeXovému prostředí `array`.

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

```
\begtt
\frame{\table{|c||l||r|}{\crl
  \multispan3\vrule\hss\bf Nadpis\hss \vrule\tabstrut \crl
  \noalign{\kern\hhkern}\crli
  první & druhý & třetí  \crlli
  sedmý & osmý  & devátý \crli}}
\endtt
```

Nadpis		
první	druhý	třetí
sedmý	osmý	devátý

Tloušťka všech čar je v TEXu implicitně 0,4pt. Makro OPmac umožňuje tuto implicitní tloušťku nastavit jinak pomocí `\rulewidth=`⟨*šířka*⟩, například `\rulewidth=1.5pt`.

Další příklad použití makra `\table` najdete na straně 24. Pokud potřebujete vytvořit komplikovanější tabulky, nezbude než prostudovat TBN, kapitolu čtvrtou.

Vkládání obrázků

Nejprve je potřeba nastavit šířku obrázku do registru `\picw` a pak je možné použít makro `\inspic` ⟨*jméno*⟩`.`⟨*přípona*⟩`"`. Obrázek se vloží jako `\hbox` dané šířky `\picw`. Registr `\picw` si ponechá svou hodnotu, takže další obrázek bude mít stejnou šířku, pokud ji nezměníte. Přípony souboru s obrázkem mohou být: `png`, `jpg`, `jbig2`, `pdf`.

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

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

```
\picw=.5\hsize \centerline{\inspic hodiny.jpg }\nobreak\medskip
\caption/f Hodiny na brněnském náměstí Svobody.
```

Makro není vhodné použít při opakovaném použití stejného obrázku v dokumentu (opakující se grafika na každé straně nebo obrázek jako odrážka ve výčtu položek). V takovém případě je vhodnější natáhnout obrázek do PDF dokumentu jen jednou pdfTEXovým příkazem `\pdfximage` a dále opakovat jeho zobrazení na různých místech dokumentu pomocí `\pdfrefximage`. Dokumentace k pdfTEXu řekne víc.

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

PDF transformace

Veškerá sazba v pdfTEXu může podléhat lineární transformaci, která je daná transformační maticí `\pdfsetmatrix{`⟨*a*⟩ ⟨*b*⟩ ⟨*c*⟩ ⟨*d*⟩`}`. Tato matice se v lineární algebře zapisuje do dvou řádků:

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}, \quad \text{např. zvětšení: } \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix}, \quad \text{nebo rotace: } \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}.$$

PdfTEXové primitivy `\pdfsetmatrix`, `\pdfsave` a `\pdfrestore` bohužel nejsou v dokumentaci pdfTEXu uvedeny, tak je musím dokumentovat aspoň zde. Příkaz `\pdfsave` uloží stávající transformační matici a aktuální bod sazby. V době konání příkazu `\pdfrestore` se matice vrátí do původní podoby a aktuální bod sazby v té době musí být na stejném místě, jako byl v době `\pdfsave`, jinak se nám sazba rozjede a pdfTEX nadává. Toho se dá docílit např. pomocí `\pdfsave...\rlap{<text>}\pdfrestore`. Transformační matice se nastavují pomocí `\pdfsetmatrix`. Opakované použití `\pdfsetmatrix` způsobí pronásobení transformační matice novou maticí, takže to funguje jako skládání zobrazení. OPmac nabízí 2 užitečná makra `\pdfscale{`⟨*vodorovně*⟩`}{`⟨*svisle*⟩`}` a `\pdfrotate{`⟨*úhel*⟩`}`. Parametr ⟨*úhel*⟩ je interpretován ve stupních. Tato makra provedou odpovídající `\pdfsetmatrix`.

Poznámky pod čarou a na okraji

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

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

Implicitní značkování je možné změnit předefinováním makra `\thefnote`. Například po

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

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

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

```
{\typoscale[/1200]\table{||lc|r||}{\crllT
      Měsíc     & zboží                & cena\hfil \crlM \tskip.2em
      leden     & noťas\fnotemark1     & 14 kKč    \cr
      únor      & skejt\fnotemark2     &  2 kKč    \cr
      červenec  & jachtička\fnotemark3 & 3,4 MKč   \crllB}}
\par\nobreak
\fnotetext{notebook}\fnotetext{scateboard}\fnotetext{jachta}
```

Čísla za slovy `\fnotemark` jsou vztažena vzhledem k jedné tabulce a nemusejí souviset se skutečným číslem poznámky. Například, je-li na stejné stránce nad tabulkou z ukázky normální `\fnote`, bude mít vytištěno číslo 1, odkazy v tabulce budou mít čísla 2, 3, 4 a případná další poznámka pod tabulkou na stejné stránce obdrží číslo 5.

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

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

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

Není ošetřen případ, kdy je `\mnote` víceřádková a je umístěna na úroveň například posledního řádku strany. Pak text poznámky přečuhuje poněkud dolů ze strany. Je tedy nutné `\mnote` použít jen na velmi krátké poznámky a případně si tento jev pohlídat a ošetřit před definitivní sazbou manuálně.

Bibliografické údaje

Pomocí `\cite[`⟨*lejblík*⟩`]` nebo `\cite[`⟨*lejblík1*⟩`,`⟨*lejblík2*⟩`,`⟨*lejblík3*⟩`]` (atd.) vytvoříme v textu odkazy na položky v seznamu literatury. V seznamu literatury je třeba uvést záznamy, které mají odkazované lejblíky. Tyto záznamy dostanou v seznamu automaticky vygenerovaná čísla a sekvence `\cite` se pak promění na číselné odkazy, například [27] nebo [18, 24, 42] (atd.). Souvislé řady čísel [1, 2, 3, 5, 6] se promění v intervaly [1–3, 5–6] jen tehdy, když je v úvodní deklaraci dokumentu napsáno `\shortcitations`.

Seznam literatury je možné vložit do dokumentu třemi různými způsoby:

- Manuálně: pomocí jednotlivých položek `\bib[`⟨*lejblík*⟩`]` přímo v dokumentu.
- S využitím BibTEXu makrem `\usebibtex{`⟨*bib-báze*⟩`}{`⟨*bst-styl*⟩`}`.

- Využitím jednou vygenerované databáze makrem `\usebbl/`⟨*typ*⟩ ⟨*bbl-báze*⟩.

Jednotlivé způsoby jsou níže probrány podrobněji.

Manuálně vložený seznam literatury v dokumentu vypadá například takto:

```
\bib[tbn] P. Olšák. {\it\TeX{}book naruby.} 468~s. Brno: Konvoj, 1997.
\bib[tst] P. Olšák. {\it Typografický systém \TeX.}
                    269~s. Praha: CSTUG, 1995.
```

Výše uvedená ukázka dá následující výstup:

[1] P. Olšák. *TEXbook naruby.* 468 s. Brno: Konvoj, 1997.

[2] P. Olšák. *Typografický systém TEX.* 269 s. Praha: CSTUG, 1995.

Využití BibTEXu. Předpokládá se, že uživatel má soubor ⟨*bib-báze*⟩`.bib`, ve kterém jsou nashromážděny bibliografické údaje ve formátu, v jakém je čte program BibTEX. V TEXové distribuci jistě najdete nějaký `*.bib` soubor, tak se do něj podívejte. Lejblíkem je první údaj u každého bibliografického záznamu. Soubor ⟨*bib-báze*⟩`.bib` by měl obsahovat bibliografické údaje, které jsou nadmnožinou toho, co potřebuje uživatel vypsat ve svém dokumentu. Na místo, kde budete chtít vypsat seznam literatury, vložte následující pokyn:

`\usebibtex{`⟨*bib-báze*⟩`}{`⟨*bst-styl*⟩`}`

Parametr ⟨*bib-báze*⟩ je jméno souboru bez přípony `.bib`, ve kterém jsou připraveny bibliografické záznamy. Parametr ⟨*bst-styl*⟩ je jméno stylového souboru bez přípony `.bst`, který použije BibTEX pro konverzi ze zdroje ⟨*bib-báze*⟩`.bib` do výstupu ⟨*dokument*⟩`.bbl`. Tento výstup pak bude makrem `\usebibtex` přečten a vložen do dokumentu. Typicky používané ⟨*bst-styly*⟩ jsou `plain`, `alpha`, `apalike`, `ieeetr`, `unsrt`. Styl `alpha` způsobí, že se místo čísel začnou zjevovat v dokumentu zkratky, a to jednak v seznamu literatury a jednak v místě výskytu příkazů `\cite`. V takovém případě pochopitelně `\shortcitations` nefunguje a pokud byste se o něj pokusili, tak makro havaruje. Existují desítky, možná stovky dalších `.bst` stylů, viz internet.

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

```
WARNING: .bbl file doesn't exist.
         Use the "bibtex ⟨dokument⟩" command.
```

Přejděte tedy na příkazový řádek a napíšeme `bibtex` ⟨*dokument*⟩. Tím se spustí program BibTEX, který přečte ze souboru ⟨*dokument*⟩`.aux` vstupní pokyny (kterou otevřít `.bib` databázi, který `.bst` styl a jaké lejblíky jsou požadovány) a na základě toho vygeneruje soubor ⟨*dokument*⟩`.bbl`, který obsahuje výběr jen těch záznamů, které byly uživatelem citovány pomocí `\cite`. Soubor ⟨*dokument*⟩`.bbl` je

navíc zkonvertovaný z `.bib` formátu do formátu čitelného TEXem. Tato konverze je řízena stylem `.bst`.

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

Seznam literatury obsahuje po použití BibTEXu jen citovaná dílka. Pokud chcete do seznamu zařadit další položky, které nejsou v textu explicitně odkazovány příkazem `\cite`, použijte `\nocite[`⟨*lejblík*⟩`]`. Toto makro dá BibTEXu pokyn, aby do seznamu zahrnul i položku s ⟨*lejblíkem*⟩, ale v místě použití tohoto makra se nevytiskne nic. Konečně pomocí `\nocite[*]` dáváme BibTEXu vzkaz, že chceme mít v seznamu literatury celou `.bib` databázi.

Zdroj bibliografických záznamů může být ve více `.bib` souborech. Pak stačí jejich názvy oddělit čárkou: `\usebibtex{`⟨*bib-báze1*⟩`,`⟨*bib-báze2*⟩`}{`⟨*bst-styl*⟩`}`.

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

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

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

1. Vytvořte si zvláštní dokument ⟨*mojebáze*⟩`.tex`, do kterého napíšete:

   `\input opmac  \genbbl{`⟨*bib-báze*⟩`}{`⟨*bst-styl*⟩`}  \end`

2. Po TEXování dokumentu ⟨*mojebáze*⟩`.tex` spusťte `bibtex` ⟨*mojebáze*⟩. Tím se vytvoří soubor ⟨*mojebáze*⟩`.bbl`.
3. Zpracujte TEXem soubor ⟨*mojebáze*⟩`.tex` ještě jednou. Vytvoří se seznam veškeré literatury, který byl v souboru ⟨*bib-báze*⟩`.bib`, přitom každá položka je označena svým ⟨*lejblíkem*⟩. Vytiskněte si tento výstup a dejte si jej na nástěnku.
4. Uložte soubor ⟨*mojebáze*⟩`.bbl` někam, kde jej umí přečíst TEX bez ohledu na to, v kterém pracujete adresáři.

5. Přejděte k editaci svého dokumentu, pište `\cite` nebo `\nocite` podle potřeby a v místě seznamu literatury dejte sekvenci `\usebbl/`⟨*typ*⟩ ⟨*mojebáze*⟩. Údaj ⟨*typ*⟩ má tyto možnosti:

```
\usebbl/a ⟨mojebáze⟩ % vypsat kompletně celou ⟨mojebaze⟩ (a=all),
\usebbl/b ⟨mojebáze⟩ % jen cite údaje řadit dle ⟨mojebaze⟩ (b=base),
\usebbl/c ⟨mojebáze⟩ % jen cite řadit podle pořadí citace (c=cite).
```

Kroky 2 až 4 budete muset opakovat pouze tehdy, když budete chtít přidat do ⟨*mojebáze*⟩`.bbl` další údaj, tj. po aktualizaci souboru ⟨*bib-báze*⟩`.bib`. Prudí-li různí odběratelé vaší vědecké činnosti požadavky na různé ⟨*bst-styly*⟩, stačí si vygenerovat podle různých stylů různé soubory `mybbl-plain.bbl`, `mybbl-ieeetr.bbl` a další.

Okraje

PlainTEX nastavuje levý okraj 1 in a šířku sazby (`\hsize`) nastavuje tak, aby i pravý okraj při formátu papíru „letter“ byl 1 in. Také horní a dolní okraj (do kterého přesahuje záhlaví a stránková číslice) jsou nastaveny na 1 in při formátu papíru „letter“ a tím je určena výška sazby (`\vsize`). $\mathcal{CS}$plain dělá totéž, tj. okraje jsou 1 in, ale formát papíru je A4.[1] OPmac umožňuje toto nastavení změnit příkazem:

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

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

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

[1] Přesněji: $\mathcal{CS}$plain nastavuje výšku sazby `\vsize=239.2mm`, což vede k dolnímu okraji o 7 mm většímu než 1 in.

Uživatel může před použitím `\margins` definovat vlastní ⟨*formát*⟩ papíru pomocí příkazu `\sdef{pgs:`⟨*formát*⟩`}{`⟨*šířka*⟩ ⟨*výška*⟩`}`. OPmac například implicitně definuje:

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

Celou sazbu na úkor okrajů je možno zvětšit nebo zmenšit pomocí makra `\magscale[`⟨*factor*⟩`]`. Například `\magscale[500]` zmenší sazbu na polovinu. Při této změně zůstává na místě „Knuthův bod“, tj. bod o souřadnicích 1 in, 1 in od horního a levého okraje. Sazba samotná je zalomena zcela stejně. Jednotky použité v dokumentu jsou od této chvíle relativní. Například po `\magscale[2000]` je použitá jednotka v dokumentu `1mm` ve skutečnosti `2mm`. Makro `\magscale` ponechává nezměněny jen rozměry stránek dané formátem stránek (A4, A3, atd.). Příklad použití makra: `\magscale[1414] \margins/1 a4 (,,,)mm` umístí sazbu, která je určena pro tisk na A5, doprostřed stránky A4 a odpovídajícím způsobem ji zvětší, aby se to korektorům lépe četlo.

Reference

[1] L. Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[2] M. Goossens, F. Mittelbach, a A. Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

Summary: OPmac — The Macro Package to Enrich the Possibilities of PlainTeX

The OPmac package is set of simple additional macros to plainTeX. It enables to users to take advantage of basic LaTeX functionality: the font size changing, automatic creation of table of contents and index, working with bibliography databases, tables, references including hyperlinks option, etc. In this paper, the significant properties of the OPmac are shown. The complete source of the macros and user and technical documentation are available on the `http://petr.olsak.net/opmac.html`.

Key words: OPmac, plainTeX extension macros, $\mathcal{CS}$plain.

petr@olsak.net

$\mathcal{CS}$plain z roku 2012

Petr Olšák

$\mathcal{CS}$plain existuje od roku 1994 a je to *jemné* rozšíření plainTEXu tak, aby v něm bylo možno pracovat v češtině a slovenštině. Toto platilo až do října 2012, kdy byla provedena významná revize a doplnění $\mathcal{CS}$plainu. Výchozí změna vyplynula z rozhodnutí implicitně nastavit vstupní kódování $\mathcal{CS}$plainu na UTF8. Kromě toho získal $\mathcal{CS}$plain mnoho dalších nových vlastností: zavádění všech dostupných vzorů dělení všemožných jazyků, schopnost spolupráce s 16bitovými TEXovými mašinami (LuaTEX, XeTEX), efektivní práce s fonty včetně matematických, snadné přepínání interního kódování včetně Unicode, uživatelsky přívětivá sada maker OPmac. V implicitní konfiguraci zůstává $\mathcal{CS}$plain stále jemným rozšířením plainTEXu zpětně kompatibilní s předchozími verzemi. Nové možnosti jsou snadno dostupné pomocí `\input` a při jejich použití už zdaleka není možné hovořit jen o rozšíření *jemném*. Naopak, je to silný konkurent všech možných poněkud přerostlých makro-nadstaveb TEXu, přitom vítězí v jednoduchosti, účelnosti řešení, přímočarosti a snadnosti použití. Nový $\mathcal{CS}$plain je přístupný na `http://petr.olsak.net/csplain.html`.

1. Úvod

V říjnu roku 2012 proběhla na diskusním listu `cstex` debata o konfiguraci vstupního kódování $\mathcal{CS}$plainu, ze které vyplynulo, že už mnoho let je $\mathcal{CS}$plain v distribucích pro MS Windows implicitně konfigurován špatně: s kódováním ISO-8859-2, které je v tomto operačním nesystému cizí. Zhrozil jsem se. Zjistil jsem posléze, že například studenti MFF, kteří pořádají korespondenční seminář, kvůli této chybě používají jakési své makro, které napravuje tuto chybu pomocí aktivních znaků. Zhrozil jsem se podruhé.

V diskusi jsem se snažil vysvětlit naše dávné rozhodnutí, že vstupní kódování $\mathcal{CS}$plainu musí být nastaveno v závislosti na použitém systému. Podstatné je, že když v editoru ve zdrojovém textu správně vidím české a slovenské znaky, $\mathcal{CS}$plain je správně vytiskne. Když je vidím blbě, nesmím se divit, že i z $\mathcal{CS}$plainu vylezou porouchané. Bohužel tato naše idea byla platná před deseti lety. Dnes jsou textové editory vybaveny inteligencí, která se snaží o autodetekci kódování a pokusí se cokoli zobrazit správně. V takovém prostředí výše uvedené pravidlo pozbývá smyslu. Na druhé straně tyto moderní editory zvládají kódování UTF8, takže jsme se rozhodli, že toto bude implicitně nastavené vstupní kódování $\mathcal{CS}$plainu pro všechny systémy. Ušetříme si tím starostí s různým nastavováním pro různé systémy. Samozřejmě je možné vygenerovat $\mathcal{CS}$plain postaru v nějakém zvoleném

 doi: 10.5300/2012-1/42

8bitovém vstupním kódování. Ale po instalaci TEXové distribuce bez dodatečného vrtání se v konfiguraci bude CSplain číst vstup v UTF8.

Konverze mezi UTF8 vstupními kódy a interním kódováním TEXu musí být přímočará na úrovni input procesoru, jako tomu bylo dosud. Proto je pro UTF8 kódovaný csplain v 8bitových TEXových mašinách (TEX, pdfTEX) použito rozšíření encTEX. Zda je toto rozšíření do TEXové mašiny zabudováno, poznáte pomocí přepínače `-enc`, kterému encTEXem poznamenané mašiny rozumějí. Není-li toto rozšíření k dispozici, není možné pro 8bitovou mašinu zprovoznit UTF8 kódovaný CSplain. Všechny obvyklé TEXové distribuce mají v 8bitových mašinách encTEX zabudován.

V době, kdy píšu tento článek (začátek prosince 2012) udržuji čilou komunikaci s Karl Berrym o zařazení nového CSplainu do TEXlive. Karl si už v listopadu sám od sebe všiml, že se něco děje, a oslovil mě. Je to tím, že mé FTP adresáře se automaticky zrcadlí do CTANu. Dá se tedy očekávat, že v nejbližší době bude nový CSplain součástí TEXlive. Až bude ukončeno jeho zařazení do TEXlive, oslovím Christiana Schenka a projednám s ním zařazení do MikTEXu.

2. Generování formátu

Formát CSplainu s UTF8 vstupem vygenerujeme následujícím způsobem:

```
pdftex -ini -enc "\let\enc=u \input csplain.ini"
pdftex -jobname pdfcsplain -ini -enc "\let\enc=u \input csplain.ini"
```

První řádek vygeneruje `csplain.fmt` s výstupem do DVI (místo mašiny pdfTEX lze použít i TEX). Druhý řádek vygeneruje `pdfcsplain.fmt` s výstupem do PDF. Makra při inicializaci CSplainu poznají podle názvu generovaného formátu (`csplain/pdfcsplain`), zda mají nastavit výstup do DVI nebo PDF. Přepínačem `-enc` je probuzen k životu encTEX a pokynem `\let\enc=u` je řečeno, že má encTEX nastavit vstupní kódování UTF8. Formát bude mít zavedeny anglické vzory dělení stejné jako v plainTEXu a dále české a slovenské vzory dělení, obě připraveny ve dvou kódováních: podle CSfontů a v T1 kódování (alias Cork). O možnosti zavést vzory dělení slov dalších jazyků pojednává sekce 10.

V konfiguračních souborech TEXových distribucí pro generování formátu se obvykle používá soubor `csplain-utf8.ini`, který obsahuje uvedený pokyn `\let\enc=u` a zavolá `csplain.ini`. Také je tam obvykle přidán přepínač `-etex`, který probudí k životu rozšíření eTEX. Makra CSplainu nikdy toto rozšíření nepoužívají, takže to není nutné rozšíření. Ovšem, pokud uživatel má zájem to použít, má i možnost. Takže v konfiguračních souborech pro generování formátu najdete obvykle analogii těchto příkazů:

```
pdftex -jobname csplain -ini -etex -enc csplain-utf8.ini
pdftex -jobname pdfcsplain -ini -etex -enc csplain-utf8.ini
```

Můžete samozřejmě také vygenerovat formát $\mathcal{CS}$plainu postaru s 8bitovým vstupem:

```
pdftex -ini -enc csplain.ini
pdftex -ini -enc "\let\enc=w \input csplain.ini"
pdftex -ini -enc "\let\enc=p \input csplain.ini"
```

Všechny tyto řádky generují `csplain.fmt` s rozšířením encTEX. První řádek vytvoří formát se vstupním kódováním ISO-8859-2, druhý s kódováním CP1250 (Windows) a třetí s kódováním CP852. Všechny varianty budou vystupovat do DVI. Chcete-li vytvářet PDF, přidejte `-jobname pdfcsplain`.

Při volbě 8bitového vstupu není nutné použít encTEX. Formát můžete generovat třeba takto:

```
pdftex -ini -translate-file=cp227.tcx csplain.ini
pdftex -ini -translate-file=cp1250cs.tcx csplain.ini
pdftex -ini -translate-file=cp852-cs.tcx csplain.ini
```

Jako prve: první řádek vytvoří formát se vstupním kódováním ISO-8859-2, druhý s kódováním CP1250 (Windows) a třetí s kódováním CP852. Jednotlivé možnosti nastavení (encTEX nebo TCX tabulka) nelze míchat dohromady.

Formát $\mathcal{CS}$plainu je možné vygenerovat i pro použití v 16bitových TEXových mašinách (LuaTEX a XETEX) pomocí následujících příkazů:

```
xetex -jobname pdfcsplain -etex -ini csplain.ini
luatex -ini csplain.ini
luatex -jobname pdfcsplain -ini csplain.ini
```

Vidíme, že na rozdíl od 8bitových mašin není nutné (a ani to není možné) použít rozšíření encTEX, protože UTF8 vstup je zde implicitní. Formát v tomto případě zavede vzory dělení češtiny a slovenštiny nejen podle kódování $\mathcal{CS}$fontů a Corku, ale také v Unicode. Program XETEX vystupuje pouze do PDF, takže můžete sice vygenerovat i `csplain.fmt` (bez předpony `pdf`), ale i ten bude vystupovat do PDF. Program LuaTEX se chová shodně jako pdfTEX a má tedy možnost přepnout mezi DVI a PDF výstupem. Implicitně nastavený výstup je tedy rozlišen podle názvu (`csplain/pdfcsplain`). Protože 16bitový výstup do DVI není dále snadno zpracovatelný, preferujeme výstup do PDF.

3. $\mathcal{CS}$plain – opakování

Nejprve stručně shrnu vlastnosti, které $\mathcal{CS}$plain odjakživa měl. V dalších sekcích jsou pak popsány novinky.

Formát csplain je inicializován tak, aby se implicitně choval jako plainTEX. To znamená, že je nastaveno anglické dělení slov a sekvence typu `\v`, `\'` expandují

na primitiv `\accent`. Rovněž je nastaveno `\nonfrenchspacing`. Rozdíl mezi plainTEXem a $\mathcal{CS}$plainem je jen ve velikosti implicitního rozměru zrcadla sazby, které je v $\mathcal{CS}$plainu nastaveno pro palcové okraje při formátu A4, zatímco v plainTEXu je nastaveno pro palcové okraje formátu Letter.

Volbu vzorů dělení slov a nastavení expanze sekvencí `\v`, `\'`, `\^`, `\‘`, `\"`, `\r` na přirozené znaky lze provést následující příkazy:

```
\chyph      % inicializuje české dělení slov a \frenschpacing
\shyph      % inicializuje slovenské dělení slov a \frenschspacing
\csaccents  % způsobí jiné chování \', \v, \^, \‘, \" a \r, které
            % od této chvíle expandují přímo na znaky podle CSfontů
```

Doporučení: První řádek dokumentu by měl být například:

```
\chyph  % use format csplain
```

Když bude uživatel takový dokument zpracovávat jiným formátem, který nemá definován příkaz `\chyph`, výše uvedený řádek se objeví v chybovém hlášení včetně komentáře, takže uživatel vidí, čím má dokument zpracovat.

Návrat k původnímu nastavení:

```
\ehyph      % americké dělení a \nonfrenchspacing
\cmaccents  % \', \v a spol.. expandují na primitiv \accent
```

Další řídicí sekvence jsou jen zkratkami k některým znakům v $\mathcal{CS}$fontech:

```
\clqq       % levá dvojitá česká uvozovka
\crqq       % pravá dvojitá česká uvozovka
\flqq       % levá dvojitá francouzská uvozovka
\frqq       % pravá dvojitá francouzská uvozovka
\promile    % znak pro promile
\uv         % text uvozený českými uvozovkami: \uv{text}
\ogonek a   % polské „a s ocáskem" (sestaveno z komponent)
```

4. UTF8 vstup při použití encTEXu

UTF8 kódovaný $\mathcal{CS}$plain s encTEXem napíše při každém spuštění do logu a na terminál toto hlášení:

```
The format: csplain <Nov. 2012>.
The cs-fonts are preloaded and A4 size implicitly defined.
The utf8->iso8859-2 re-encoding of Czech+Slovak alphabet
    activated by encTeX
```

Na vstupu se může vyskytnout desetitisíce různých UTF8 kódů. Nečekejte ovšem, že jsou všechny správně $\mathcal{CS}$plainem zpracovány bez starosti navíc. Vygenerovaný formát zaručí správné zpracování ASCII znaků. Dále je zaručeno správné zpracování znaků české a slovenské abecedy, tedy znaků:

Á á Ä ä Č č Ď ď É é Ě ě Í í Ĺ Í Ľ ľ Ň ň Ó ó Ö ö Ô ô Ŕ ŕ Ř ř Š š Ť ť Ú ú Ů ů Ü ü Ý ý Ž ž

Tyto znaky jsou z UTF8 kódu mapovány na interní jeden byte implicitně podle kódování $\mathcal{CS}$fontů, což je pro českou a slovenskou abecedu totéž jako kódování ISO-8859-2. Dále jsou mapovány UTF8 kódy všech znaků, které mají svou řídicí sekvenci definovanou v plainu nebo v $\mathcal{CS}$plainu. Jsou to tyto znaky:

```
plain: \ss ß  \l ł  \L Ł  \ae æ  \oe œ  \AE Æ  \OE Œ
       \o ø  \O Ø  \i ı  \j ȷ  \aa å  \AA Å
       \S §  \P ¶  \copyright ©  \dots …  \dag †  \ddag ‡
csplain: \clqq „  \crqq “  \flqq «  \frqq »  \promile ‰
```

Pokud je na vstupu jakýkoli jiný UTF8 kód, implicitně jej $\mathcal{CS}$plain nemá mapován, takže se vypíše na terminál a do logu následující varování:

```
WARNING: unknown UTF-8 code: `X = ^^xx^^xx' (line: ??)
```

a do tiskového výstupu se vloží na dané místo černý čtvereček. Uživatel si musí takový znak dodefinovat. Například po hlášení:

```
WARNING: unknown UTF-8 code: `Ñ = ^^c3^^91' (line: 42)
```

si uživatel do záhlaví dokumentu například doplní definici:

```
\mubyte\Ntilde ^^c3^^91\endmubyte % UTF8 kód mapován na \Ntilde
\def\Ntilde{\~N}                  % sekvence \Ntilde definována
```

Po této úpravě se při zpracování dokumentu už varování neobjeví a vstupní kód se zpracuje jako řídicí sekvence `\Ntilde`, která expanduje na `\~N`, takže v tiskovém výstupu se objeví Ñ.

Pokud se řídicí sekvence mapovaná pomocí `\mubyte` objeví v argumentu `\write` souboru, nebude expandovat, ale promění se při zápisu do souboru zpětně do odpovídajícího UTF8 kódu.

Během `\write` a při zápisu do logu se také zpět na UTF8 kódy proměňují interní byte, které byly mapovány. Implicitně tedy jsou to znaky české a slovenské abecedy vyjmenované výše. Ostatní nemapované interní byte s kódem větším jak 127 se přepisují podle dvouzobákové konvence např. takto: `^^ad`. Jestliže do deklarace dokumentu napíšete `\xprncodes=1`, budou se ostatní nemapované byty vypisovat přímo.

Příkazy `\mubyte` a `\endmubyte` jsou součástí encTEXu a jsou popsány v jeho dokumentaci. Stručně řečeno `\mubyte token string\endmubyte` zařadí do kódovací tabulky další údaj: `string` bude na vstupu převeden na interní `token` v TEXu a při činnosti `\write` bude `token` zpětně převeden na `string`.

Shrnuto: UTF8 kódy znaků české a slovenské abecedy jsou mapovány na interní byte a ostatní UTF8 kódy je třeba mapovat na kontrolní sekvence. Těch máme víceméně neomezeně mnoho, takže můžeme i v 8bitovém TEXu pracovat s rozsáhlým počtem různých vstupních UTF8 kódů.

Při použití jiných fontů pomocí `\input ctimes` (atd., viz sekci 7) je zavolán soubor `chars-8z.tex`, který mapuje další UTF8 kódy na řídicí sekvence, jež jsou pomocí `\chardef` propojeny se znakem fontu. Jsou to tyto znaky:

```
\euro €   \trademark ™   \registered ®   \ellipsis …
\textbullet •   \sterling £   \currency ¤
```

Soubor `chars-8z.tex` navíc předefinovává makra plainu `\P`, `\S`, `\dag`, `\ddag`, `\copyright`, `\lslash`, `\Lslash` pomocí `\chardef`, aby tyto řídicí sekvence vedly přímo na znak ve fontu.

V balíčku `enctex.tar.gz` (a podruhé i v `csplain.tar.gz`) jsou připraveny soubory, které mapují UTF8 kódy celých bloků UNICODE tabulky na řídicí sekvence a definují jejich výchozí chování. Můžete použít:

```
\input utf8lat1  % Latin-1 Supplement U+0080-U+00FF
\input utf8lata  % Latin Extended-A U+0100-U+017F
```

a možná v budoucnu i další. Po zavedení těchto souborů se nově deklarované UTF8 kódy mapují na řídicí sekvenci, např. ë se mapuje na `\edieresis` a tato sekvence expanduje na příslušné sestavení akcentu, v tomto příkladě na `\"e`. Na interní byte zůstávají mapovány jen znaky české a slovenské abecedy. Pro nově mapované řídicí sekvence jsou použity definice ze souborů `utf8lat*.tex` jen tehdy, pokud tyto řídicí sekvence nejsou definovány dříve.

Pokud předložíte $\mathcal{CS}$plainu (generovaném pro UTF8 kódování vstupu) dokument kódovaný jinak, než v UTF8, skoro jistě se dočkáte chybového hlášení:

```
! UTF-8 INPUT IS CORRUPTED !
  May be you are using another input encoding.
```

Je ovšem možné snadno přejít do módu, při němž UTF8 kódovaný $\mathcal{CS}$plain pracuje stejně jako $\mathcal{CS}$plain s kódováním ISO-8859-2. Stačí na začátek dokumentu napsat:

```
\mubytein=0  \mubyteout=0  \mubytelog=0  \xprncodes=1
```

Tuto práci vykoná také soubor `utf8off.tex`, tj. stačí napsat `\input utf8off`. Tento soubor navíc definuje makro `\clearmubyte`, které vymaže data deklarovaná pomocí `\mubyte...\endmubyte`.

Je dokonce možné mít na vstupu při sazbě jediného dokumentu různé soubory různě kódované. Není nutné při přechodu z jednoho kódování přepínat na druhé. Stačí na začátek dokumentu napsat:

```
\input mixcodes
```

a od této chvíle je možné na vstupu střídat texty kódované dle UTF8, ISO-8859-2 nebo CP1250 dle libosti. Všechny výstupy pomocí `\write` jsou kódovány podle UTF8 a jsou tedy opět připraveny k načtení.

V předchozím odstavci není zcela přesná formulace „střídat kódování dle libosti“. Platí to jen pro český text. Slovákům nefunguje »ľ« v CP1250. Pokud jim to vadí, napíšou `\shyph` před voláním `\input mixcodes`. Pak ale zase nefunguje »ž« v ISO-8859-2. Tyto dva znaky jsou v uvedených dvou kódováních poněkud v konfliktu. UTF8 vstup ale funguje bez problémů.

5. Interní kódování

Interní kódování TEXu je v $\mathcal{CS}$plainu implicitně nastaveno na kódování $\mathcal{CS}$fontů, které má znaky české a slovenské abecedy kódovány podle ISO-8859-2. Na interní kódování musejí správně navázat vzory dělení slov a použité fonty. Konverzi ze vstupního kódování do interního dělá input procesor TEXu, typicky encTEX.

$\mathcal{CS}$plain podporuje kromě kódování $\mathcal{CS}$fontů ještě další interní kódování: T1 (alias Cork) a Unicode. Posledně jmenované kódování je možné jen v 16bitových TEXových mašinách. Chcete-li přejít do kódování podle Corku nebo do Unicode, napište na začátek dokumentu:

```
\input t1code  % nastaveno kódování T1
\input ucode   % nastaveno kódování Unicode
```

Používáte-li 8bitovou TEXovou mašinu s encTEXem, pak `\input t1code` zařídí nejen správné nastavení `\lccode` atd., ale také přenastaví encTEXovou konverzní tabulku tak, že nyní se budou všechny UTF8 kódy, které odpovídají znakům v T1 kódování, převádět na interní byte v T1 kódování.

Používáte-li 16bitovou TEXovou mašinu, pak je bezvýhradně nutné na začátek dokumentu napsat `\input ucode`, protože tyto mašiny implicitně převádějí UTF8 kódy do Unicode a není jednoduché je přimět k jinému chování.

Po změně kódování pomocí `\input t1code` nebo `\input ucode` se automaticky změní chování příkazů `\chyph` a `\shyph` tak, že tyto příkazy zapnou vzory dělení slov ve správném kódování. Je ovšem nutné tyto příkazy psát *až poté*, co je změněno kódování.

Po změně kódování pomocí `\input t1code` nebo `\input ucode` se změní také vnitřní chování makra `\csaccents`: sekvence `\v`, `\'` atd. budou expandovat na znaky podle zvoleného kódování.

Změna kódování pomocí `\input t1code` nebo `\input ucode` neřeší zavedení tomu odpovídajících fontů. To znamená, že pokud neuděláte nic dalšího, zůstanou v $\mathcal{CS}$plainu zavedeny $\mathcal{CS}$fonty, jejichž kódování na `t1code` ani `ucode` nenavazuje. Takže ve výsledku uvidíte pomršenou češtinu nebo slovenštinu. Ovšem řešení je snadné: po změně kódování zavést třeba Latin Modern fonty příkazem `\input lmfonts`. Tento balíček maker má v sobě zabudovaný rozcestník podle zvoleného kódování, takže zavede ve všech třech případech fonty správně kódované:

```
\input lmfonts                  % Latin Modern v kodování CSfontů
\input t1code  \input lmfonts   % Latin Modern fonty v T1 kódování
\input ucode   \input lmfonts   % Latin Modern fonty v Unicode
\chyph      % aktivace vzorů dělení ve vybraném kódování
```

Podobnou vlastnost jako `lmfonts.tex` mají další „fontové soubory“, o kterých pojednává sekce 7.

Doporučuje se dodržovat pravidlo nejprve nastavit vnitřní kódování TEXu a pak řešit cokoli jiného. Například v sekci 4 byly zmíněny mapovací soubory `utf8lat1.tex` a `utf8lata.tex`. Pokud je zavedete dřív než `\input t1code`, budou znaky è, ã atd. mapovány na `\egrave`, `\atilde` atd. a tyto kontrolní sekvence budou definovány jako `\‘e`, `\~a` atd. Bude to sice fungovat, ale v T1 mají tyto znaky svůj vlastní kód. Možná by bylo tedy přirozenější mapovat znaky è, ã atd. v takovém případě přímo na jejich kódy. K tomu stačí, aby `\input t1code` *předcházel* před `\input utf8lat1`. Pak totiž makra v souboru `utf8lat1` budou vědět, že používáte T1 kódování, a vyřeší mapování přímočarým způsobem na interní byty. Dokonce tím vyřešíte znaky `\Eth`, `\Thorn` atd., které jsou implicitně v `utf8lat1.tex` definovány tak, že vypíší varování o nedostupnosti znaku, zatímco v T1 kódování jsou jednoduše i tyto znaky mapovány na své kódy.

Možná nyní někoho napadne otázka, co udělá `\input utf8lat1`, pokud předchází `\input ucode`. Nic. Unicode je možné použít jen v 16bitových mašinách, ale tyto mašiny nemají encTEX, takže v nich nefunguje mapování pomocí encTEXu. Ani není potřeba nic mapovat, protože v 16bitových mašinách asi zavedete rovnou Unicodový font.

6. Změna velikosti fontů

Nový *CS*plain nabízí elementárním způsobem možnost zvětšování nebo zmenšování fontů do požadované velikosti. Pomocí makra `\regfont` se registruje fontový přepínač, který má podléhat změně velikosti. Implicitně jsou registrovány přepínače `\tenrm` (Regular), `\tembf` (**Bold**), `\tenit` (*Italic*), `\tenbi` (***BoldItalic***), `\tentt` (`Mono`). Pak je potřeba definovat makro `\sizespec` jako „scaled1200“ nebo „at17pt“ a poté spustit makro `\resizeall`. To zavede pro každý registrovaný přepínač jeho původní font v nově požadované velikosti. Individuálně je možné zavést jedinému přepínači font v nové požadované velikosti (podle `\sizespec`) makrem `\resizefont`. Příklad:

```
\def\sizespec{scaled1200}
\resizeall % fonty \tenrm, \tenbf, \tenit, \tenbi, \tentt
           % nyní budou scaled 1200.
\def\sizespec{at8pt}
\resizeall % fonty \tenrm, \tenbf, \tenit, \tenbi, \tentt
           % nyní budou at 8pt.
```

```
\font\tenss=csss10 % nově zavedený font sans serif
\regfont\tenss     % registrujeme jej pro změnu velikostí
\def\sizespec{at13pt}
\resizeall % fonty \tenrm, \tenbf, \tenit, \tenbi, \tentt
           % a \tenss nyní budou at 13pt.
\def\sizespec{scaled700}
\resizefont\tenbi  % font \tenbi bude nyní scaled 700.
```

Veškerá nastavení z `\resizefont` a `\resizeall` jsou lokální v rámci skupiny. Můžete si definovat třeba makro `\small`, které přepne na 8bodové písmo takto:

```
\def\small{\def\sizespec{at8pt}\resizeall \tenrm}
Tady je normální text {\small a tady je zmenšený \it i v kurzívě.}
A tu je zase normálně veliké písmo.
```

Makro OPmac společně s makrem `ams-math.tex` rozvíjí tuto jednoduchou myšlenku změny velikostí fontů k téměř dokonalosti. Jednak nabízí uživateli pohodlnou změnu velikostí a nastavení třeba i relativně vzhledem k aktuální velikosti písma (což nemusí být 10bodů) a velikost správně nastavuje i všem matematickým rodinám včetně indexů a subindexů. Také interně pracuje s tabulkami designovaných velikostí metrik (např. `csr10 at12pt` je něco jiného než `csr12`) a zvětšuje/zmenšuje za použití metriky s vhodnou designovanou velikostí. Nabízí tedy srovnatelné možnosti jako NFSS, ale jednodušeji implementované a s pohodlnějším uživatelským rozhraním. Počet řádků kódu tohoto řešení je 5 (ve formátu $\mathcal{CS}$plain) a 40 v `ams-math.tex`. V jednoduchosti je síla.

Názvy fontových přepínačů `\tenrm`, `\tenbf` atd. zůstávají tedy shodné, jako v plainTEXu, ale jejich význam je typicky jiný. Nejsou to vždy fonty výhradně ve velikosti `\ten*`. Je třeba ten název chápat jako takovou trošičku pozůstalost z původního plainTEXu. Aspoň nám to signalizuje, že je to kontrolní sekvence, která je přepínačem nějakého fontu.

7. Fontové soubory

Fontový soubor je soubor, který zavádí fonty v jedné rodině, typicky v jednom kvartetu Regular, **Bold**, *Italic*, ***BoldItalic***. K přepínání do těchto variant jsou připravena makra `\rm`, `\bf`, `\it`, `\bi`. Tato makra volají fontový přepínač `\tenrm`, `\tenbf`, `\tenit` a `\tenbi`. Fontové soubory zavedou pro tyto fontové přepínače požadované fonty. Například soubor `ctimes.tex` nastaví rodinu fontů Times tak, že zavede font `\tenrm` jako ptmr8z (při zvoleném kódování $\mathcal{CS}$fontů), `\tenbf` jako font ptmb8z atd. Přitom `ptmr8z.tfm`, `ptmb8z.tfm` jsou metriky odpovídajících fontů rodiny Times. Kromě zmíněného kvartetu typicky fontové soubory zavádějí aspoň `\tentt` pro strojopis (makro `\tt`). Fontový soubor `ctimes.tex` pro tento účel použije Courier.

Zavedené fonty je možné dodatečně zvětšovat a zmenšovat způsobem popsaným v předchozí sekci.

„Klasické“ fontové soubory, které jsou součástí $\mathcal{CS}$plainu už mnoho let, jsou:

```
\input ctimes    % TimesRoman
\input chelvet   % Helvetica
\input cavantga  % AvantGarde
\input cncent    % NewCentury
\input cpalatin  % Palatino
```

Všechny tyto klasické fontové soubory zavádějí fonty v kódování $\mathcal{CS}$fontů[1] i v T1[2]. Unicode není v klasických fontových souborech podporován.

Některé fontové soubory připraví více než běžný fontový kvartet+tt. V každém případě se o tom zmíní na terminálu a v logu. Například po zavedení `chelvet.tex` se lze dočíst:

```
FONT: Helvetica - \rm, \it, \bf, \bi, \tt,
                  \cond\rm, ..., \cond\bi, \narrow
```

Je tedy zřejmé, že v případě Helveticy máme kromě běžných přepínačů `\rm`, až `\tt` ještě zúžené verze, do kterých se přepíná pomocí `\cond\rm`, `\cond\bf` atd. nebo je možné do nich jednorázově přepnout pomocí `\narrow`.

Vzhledem k tomu, že jedno písmeno `c` na začátku názvu nepůsobí příliš přehledně a může vzniknout soubor, který už v `texmf/` stromu je pro jiné účely, rozhodl jsem se nové fontové soubory pojmenovávat s předponou `cs-`. Jaké všechny fontové soubory jsou k dispozici zjistíte pomocí příkazu `tex cs-all` na příkazovém řádku. V tuto chvíli jsou připraveny k použití následující soubory:

```
\input cs-termes   % TeXgyre Termes (Times)
\input cs-heros    % TeXgyre Heros (Helvetica)
\input cs-cursor   % TeXgyre Cursor (Courier)
\input cs-adventor % TeXgyre Adventor (AvantGarde)
\input cs-bonum    % TeXgyre Bonum (Bookman)
\input cs-pagella  % TeXgyre Pagella (Palatino)
\input cs-schola   % TeXgyre Schola (NewCentury)
```

[1] Metriky všech těchto fontů jsem v říjnu 2012 kompletně přegeneroval. Původní verze řešila akcenty pomocí virtuálních fontů a kompozitů, což znamenalo, že ve výsledném PDF souboru nešlo vyhledávat česká slova a nefungovalo v PDF prohlížeči nabírání textu do clipboardu kvůli přenosu textu do jiné aplikace. Nové verze odkazují na znaky ve fontu přímo, a proto výsledné PDF uvedenými neduhy netrpí. V rámci přegenerování metrik jsem do nich přidal znaky Euro, Registered a Trademark.

[2] Metriky všech těchto fontů jsou už cca 15 let součástí TEXových distribucí a jsou vygenerovány programem fontinst. Fonty takto generované trpí chybou, která způsobuje bohužel jejich nepoužitelnost v češtině a slovenštině. Za znaky ď, ť a ľ se rozprostírá hrozivá mezera, která je dobře patrná zejména uprostřed slova. Tyto metriky asi přegenerovat nepůjde, protože jsou používány mezinárodní TEXovou komunitou. Karl Berry, se kterým jsem toto řešil, soudí, že pokud to českým a slovenským uživatelům 15 let nevadilo, není třeba spěchat s řešením.

```
\input lmfonts     % Latin Modern fonts
```

Všechny tyto uvedené fontové soubory zavedou fonty v kódování $\mathcal{CS}$fontů nebo T1 nebo Unicode (podle nastaveného vnitřního kódování). Tyto fontové soubory zavádějí kromě kvartetu+tt ještě celý kvartet ve verzi Caps and Small caps. Do těchto variant se přepíná pomocí `\caps\rm`, `\caps\it` atd.

Dále jsou k dispozici následující fontové soubory:

```
\input cs-antt    % Antykwa Torunska
\input cs-polta   % Antykwa Poltawskiego
\input cs-bera    % Bera
\input cs-arev    % ArevSans
\input cs-charter % Charter
```

a předpokládám, že si uživatelé budou schopni jednoduše dopsat další.

8. Matematika

Matematické fonty jsou řešeny v makru `ams-math.tex` nebo `tx-math.tex`. Soubor `ams-math.tex` zavádí Computer Modern fonty (nebo Latin Modern) pro základní matematické rodiny a $\mathcal{AMS}$fonty pro rozšiřující matematické rodiny. Nabízí kompletní sadu všech znaků z $\mathcal{AMS}$TEXu. Fonty jsou vizuálně kompatibilní s Computer Modern fonty. Proto je toto makro zavedeno fontovým souborem `lmfonts.tex`.

Soubor `tx-math.tex` zavádí TX fonty, které nabízejí nadmnožinu znaků známých z $\mathcal{AMS}$TEXu. Fonty jsou vizuálně kompatibilní s Times, ale snesou se i s dalšími rodinami fontů odvozených z dynamické antikvy. Proto je makro `tx-math.tex` použito pro matematiku ve vesměs všech ostatních fontových souborech. Následující text popisuje možnosti matematické sazby po zavedení souboru `ams-math.tex` nebo `tx-math.tex` ať už explicitně nebo prostřednictvím fontového souboru.

Je k dispozici příkaz `\setmathsizes[`⟨*text*⟩`/`⟨*script*⟩`/`⟨*scriptscript*⟩`]`, kterým se nastavují velikosti základní sazby, indexů a subindexů. Uvedené parametry jsou údaje v bodech (`pt`), ale tyto jednotky se tam nepíší. Po nastavení velikostí je možné použít `\normalmath` pro zavedení všech matematických rodin fontů v normálním duktu nebo `\boldmath` pro zavedení matematiky v tučném duktu. Například:

```
\setmathsizes[12/8.4/6]\normalmath % základní velikost 12pt
                                   % indexy 8.4pt a subindexy 6pt
```

Nastavení `\normalmath`, `\boldmath` je lokální v rámci skupiny. Následuje příklad pro zavedení matematiky pro případ, že uživatel napíše dolary s matematickou sazbou v nadpisu:

```
\def\titulfont{\def{at14pt}\resizefont\tenbf \tenbf
               \setmathsizes[14/9.8/7]\boldmath}
\def\titul#1\par{\centerline{\titulfont #1}}

\titul Kam konverguje $\int_x^\infty f(t){\rm d}t$?
```

Při použití makra OPmac je řešení nadpisů ještě snazší. V takovém případě ani tvůrce maker nemusí explicitně psát velikosti indexů a subindexů. Jednoduše použije příkaz `\typosize` nebo `\typoscale`. Indexy pak budou mít 70 % základní velikosti a subindexy 50 %.

Po `\normalmath` nebo `\boldmath` je zavedeno 12 matematických rodin (v případě `ams-math.tex`) resp. 14 matematických rodin (v případě `tx-math.tex`). Jedna matematická rodina sdružuje stejný font ve třech velikostech: základní, indexovou a subindexovou. Matematická rodina se zavádí buď proto, aby byla k dispozici další „matematická abeceda“ nebo proto, aby se daly použít další symboly typu $\int$, ∂ atd.

Jakmile mezi dolary začnete psát písmena bez použití matematického přepínače, je použita implicitní matematická abeceda `\mit`. Celkově jsou k dispozici následující abecedy:

`\mit`	`% matematická kurzíva`	$abc{-}xyz, ABC{-}XYZ$
`\it`	`% textová kurzíva`	$abc{-}xyz, ABC{-}XYZ$
`\rm`	`% textová antikva`	abc−xyz, ABC−XYZ
`\cal`	`% jednoduché cal. znaky`	$\mathcal{ABC}{-}\mathcal{XYZ}$
`\script`	`% kroucenější cal. znaky`	$\mathscr{ABC}{-}\mathscr{XYZ}$
`\frak`	`% fraktura`	$\mathfrak{abc}{-}\mathfrak{xyz}, \mathfrak{ABC}{-}\mathfrak{XYZ}$
`\bbchar`	`% zdvojené tahy`	$\mathbb{ABC}{-}\mathbb{XYZ}$
`\bf`	`% sans serif bold`	$\mathbf{abc{-}xyz, ABC{-}XYZ}$
`\bi`	`% sans serif bold slanted`	$\boldsymbol{abc{-}xyz, ABC{-}XYZ}$

Přepínač typu `\bf`, `\bi` může fungovat jinak v textovém a jinak v matematickém fontu. Takže skutečnost, že `\bi` přepíná v matematickém módu do sans serif bold slanted (vhodné pro sazbu vektorů nebo matic), zatímco v textovém módu do čtvrtého prvku fontového kvartetu (BoldItalic) není chyba, ale záměr.

V matematické sazbě jsou k dispozici stovky symbolů dostupných pomocí $\backslash\langle\mathit{něco}\rangle$, například `\alpha` α, `\geq` $\geq$, `\sum` $\sum$, `\sphericalangle` $\sphericalangle$, `\bumpeq`, $\bumpeq$. Seznam všech těchto symbolů najdete v dokumentaci k $\mathcal{AMS}$TEXu, která se typicky jmenuje `amsguide.ps`. Nebo v dokumentaci TX fontů `txfontsdoc.pdf`.

Makro `ams-math.tex` nastavuje jako implicitní abecedu matematickou kurzívu, která je mírně jinak kreslena než kurzíva textová. Naopak `tx-math.tex` nastavuje jako implicitní abecedu textovou kurzívu převzatou z „okolního“ textového fontu. Ta se ve vzorečcích spolu s textovým fontem bude esteticky snášet daleko lépe. Chcete-li toto implicitní chování změnit, je možné použít následující globální přepínače:

```
\itvariables    % implicitní abeceda bude textová kurzíva,
\mitvariables   % implicitní abeceda bude matematická kurzíva.
```

Chce-li uživatel zavést vlastní matematickou abecedu nebo nové symboly, má volné už jen dvě rodiny s čísly 14 a 15. Ostatní rodiny jsou obsazeny makrem `tx-math.tex`. Uživatel si vybere jedno z těchto dvou čísel (v následujícím příkladě si vybral 15) a napíše:

```
\input opmac  % zavede ams-math.tex a definuje makro \addto
\addto\normalmath {\loadmathfamily 15 ocrb10 } % OCRB metrika
\addto\boldmath   {\loadmathfamily 15 ocrb10 } % bold bohužel není
\def\ocrb{\fam15 } % definice matematického přepínače
\normalmath        % zavedení všech mat. fontů včetně nové rodiny
Zkouška: $a \cdot \ocrb x_{y_z}$
```

Možná se někomu může zdát tento příklad příliš komplikovaný. Ocení to asi pouze ti TEXisté, kteří se někdy pokusili zavést úplnou matematickou rodinu fontů v plainTEXu. A když to v potu tváře na mnoha řádcích nakonec vytvořili, nebylo to zvětšovací (pro nadpisy) ani zmenšovací (pro poznámky pod čarou). V tomto příkladě je nová rodina deklarována na dvou řádcích a je zvětšovací i zmenšovací.

9. Unicodové fonty

16bitové mašiny pracují interně v Unicode a je potřeba na to navázat Unicodovými fonty. XETEX disponuje rozšiřujícími možnostmi primitivu `\font`, který dovede rovnou načíst font formátu OpenType (přípona `.otf`). Takový font je Unicodový. Načtení fontu probíhá takto:

`\font\prepinac="[`⟨*filename*⟩`]:`⟨*fontfeatures*⟩`"` ⟨*sizespec*⟩

kde ⟨*filename*⟩ je název souboru (bez přípony `.otf`) a ⟨*sizespec*⟩ je obvyklé `at`⟨*dimen*⟩, `scaled`⟨*factor*⟩ nebo nic. Konečně ⟨*fontfeatures*⟩ jsou modifikátory fontu odděleny od sebe středníkem. Je potřeba vědět, jakými modifikátory font disponuje a kterým modifikátorům rozumí fontový loader. XETEXový fontový loader nabízí modifikátor `mapping=tex-text`, který způsobí aktivování TEXovských ligatur typu `---` → —. Obyčejné ligatury (fi, atd.) jsou aktivovány automaticky. Dále fonty disponují typickým modifikátorem `+smcp`, který způsobí, že font se chová jako Caps and small caps. Pokud napíšete neznámý nebo chybný modifikátor, nic se nestane, žádné chybové hlášení se nezjeví.

XETEX má svůj fontový loader slinkovaný s knihovnami operačního systému a nic dalšího neřeší, zatímco LuaTEX si vše dělá sám za pomocí programů v Lua. V LuaTEXu implicitně není primitiv `\font` vybaven schopností číst OpenType fonty, ale pomocí `\directlua` lze vložit Lua-kód, který to zařídí. Vyštípal jsem ze

souboru `luaotfload.sty` potřebné řádky Lua-kódu a vložil jsem je do souboru `luafonts.tex`. Nejsem si ale jist, zda vývojáři LuaTEXu neudělají v Lua-funkcích, které jsou tím Lua-kódem volány, nějakou nekompatibilní změnu a ono to přestane v budoucnu fungovat. Bohužel mi na konferenci k LuaTEXu nebyli schopni dát záruku trvanlivosti tohoto řešení.

Píšete-li svůj fontový soubor, je tedy potřebné načíst soubor `luafonts.tex`. Po provedení Lua-kódu z tohoto souboru se primitiv `\font` chová analogicky, jako v XeTEXu. V $\mathcal{CS}$plainu je soubor `unifam.tex`, který načítá jeden obvyklý kvartet OpenType fontů (plus tt a plus small caps). Můžete se podívat tam, jak je to uděláno.

Protože má LuaTEX odlišně implementovaný fontový loader, interpretuje bohužel jinak a jiné ⟨*fontfeatures*⟩. Implicitně nefungují v LuaTEXu v Unicodových fontech žádné ligatury, teprve po modifikátoru `+tlig` začnou fungovat ligatury typu `---` → — a dále modifikátor `script=latn` probudí k životu ligatury typu fi. Tyto a mnohé další ⟨*features*⟩ jsou popsány v dokumentaci `luaotfload.pdf`.

Jak bylo řečeno v sekci 7, $\mathcal{CS}$plain načítá v Unicode za jistých okolností Latin Modern fonty a TEXgyre fonty. Dělá to interně pomocí souboru `unifam.tex`. Než tyto fonty načtete (pomocí `\input lmfonts`, `\input cs-heros` atd.), je možné definovat makro `\fontfeatures` s modifikátory fontu. Není-li toto makro definováno, soubor `unifam.tex` si je dodefinuje takto:

```
\def\fontfeatures{mapping=tex-text;script=latn;+tlig}
```

10. Více jazyků

Implicitně načítá $\mathcal{CS}$plain při generování formátu následující vzory dělení:

- `\enPatt=0` ... (implicitní vzor z plainTEXu) v ASCII kódování
- `\csILtwo=5` ... čeština v ISO-8859-2
- `\skILtwo=6` ... slovenština v ISO-8859-2
- `\csCork=15` ... čeština v T1 kódování
- `\skCork=16` ... slovenština v T1 kódování
- `\csUnicode=115` ... čeština v Unicode (jen v 16bitové mašině)
- `\skUnicode=116` ... slovenština v Unicode (jen v 16bitové mašině)

Jednotlivé vzory dělení se v dokumentu zapínají pomocí `\enlang`, `\cslang` a `\sklang`. Příkaz `\enlang` zapne také `\nonfrenchspacing` a ostatní zapínají `\frenchspacing`. Jsou zachovány staré názvy přepínačů: `\ehyph=\enlang`, `\chyph=\cslang` a `\shyph=\sklang`. Přechod k novým názvům s aspoň dvěma písmenky pro jazyk je důsledkem nové možnosti v $\mathcal{CS}$plainu použít 54 různých jazyků. Jedním písmenkem je rozlišit nemůžeme.

V roce 2012 byl zcela přepracován soubor `hyphen.lan`, který je použit při generování formátu $\mathcal{CS}$plain a řeší čtení jednotlivých vzorů dělení slov. Na řádcích

48 až 160 tohoto souboru jsou za dvojtečkami skryty názvy vzorů dělení rozličných jazyků v kódováních T1 (Cork) nebo Unicode. Část výše jmenovaného souboru vypadá takto:

```
\let\csCork=y    % Czech
\let\skCork=y    % Slovak
:\let\deCork=y   % German
:\let\frCork=y   % French
:\let\plCork=y   % Polish
:\let\cyCork=y   % Welsh
:\let\daCork=y   % Danish
...
:\let\saUnicode=y   % Sanskrit
:\let\ruUnicode=y   % Russian
:\let\ukUnicode=y   % Ukrainian
:\let\hyUnicode=y   % Armenian
:\let\asUnicode=y   % Assamese
```

Stačí si vybrat jazyky, odstranit příslušnou dvojtečku a znovu vygenerovat formát $\mathcal{CS}$plain. Jazyky s abecedou, která se vejde do T1 kódování, mají připraveny vzory dělení `\*Cork` i `\*Unicode`, zatímco jazyky, které se do T1 kódování se svou abecedou nevejdou, mají připraveny vzory dělení jen `\*Unicode`.

Není nutné modifikovat soubor `hyphen.lan`. Požadované vzory dělení lze specifikovat na příkazové řádce při generování formátu pomocí `\let\vzor=y`. Třeba příkaz:

```
pdftex -ini -enc "\let\plCork=y \let\enc=u \input csplain.ini"
```

zavede navíc vzory polštiny v kódování T1. Nebo:

```
pdftex -ini -enc "\let\allpatterns=y \let\enc=u \input csplain.ini"
```

zavede navíc všechny dostupné vzory dělení `\*Cork`. Naproti tomu 16bitová mašina příkazem:

```
luatex -jobname pdfcsplain -ini "\let\allpatterns=y \input csplain.ini"
```

zavede všechny vzory dělení `\*Unicode`.

Jakmile je načten vzor dělení pro nový jazyk, je automaticky připraven k tomu odpovídající přepínač jazyka `\*lang`, tedy například `\delang` po načtení `\deCork` nebo `\pllang` po načtení `\plCork`.

Příkazy `\cslang` a `\sklang` implicitně přepínají do češtiny a slovenštiny s kódováním ISO-8859-2 a `\enlang` přepíná do angličtiny. Je-li zaveden další vzor dělení, který se vejde do ASCII, přepínač odpovídajícího jazyka také funguje, protože tyto vzory dělení pracují nezávisle na zvoleném kódování. Například `\itlang` (italština). Ostatní přepínače jazyků v kódování ISO-8859-2 nefungují.

Teprve po `\input t1code` začne `\cslang` a `\sklang` přepínat do vzorů kódovaných v T1 a budou fungovat i další jazyky se vzory dělení typu `\*Cork` načtenými při inicializaci formátu. Například `\delang`, `\pllang` atd. Konečně po `\input ucode` budou tyto přepínače jazyků přepínat do vzorů v Unicode.

Deklarace kódování pomocí `\input t1code` nastaví `\lccode` všech znaků, které jsou v T1 kódování definovány, takže všechny vzory dělení typu `\*Cork` budou fungovat. Na druhé straně `\input ucode` nastaví `\lccode` jen znaků české a slovenské abecedy. Používáte-li v 16bitové mašině další jazyk, je potřeba mu nastavit `\lccode` znaků jeho abecedy explicitně, jinak budou načtené a inicializované vzory dělení slov takového jazyka málo platné.

$\mathcal{CS}$plain definuje pro každý jazyk s načtenými vzory dělení makro `\lan:`⟨*číslo*⟩ jako ⟨*zkratku jazyka*⟩, například `\lan:5` stejně jako `\lan:15` expandují na `cs`. Programátor maker toho může využít. Programátor maker dále může využít toho, že přepínače jazyků `\czlang`, `\delang`, `\itlang`, `\frlang` atd. volají makro `\initlanguage{`⟨*zkratka jazyka*⟩`}`. Toto makro implicitně neudělá nic, ale programátor maker si je může předefinovat dle svého. Protože je `\initlanguage` zavolán těsně za `\language=`⟨*číslo*⟩ v kontextu:

```
\*lang ->  \language=⟨číslo⟩\relax
           \initlanguage{⟨značka⟩}\frenchspacing
           \lefthyphenmin=⟨lhm⟩\righthyphenmin=⟨rhm⟩%
           \message{⟨text⟩}
```

je možné, aby programátor maker odebral další inteligenci maker `\*lang` a převzal je do vlastních rukou. Třeba definuje:

```
\def\initlanguage #1#2#3\message#4{#3\csname lg:#1\endcsname}
```

Toto řešení přebírá z makra `\*lang` jen nastavení registrů `\lefthyphenmin` a `\righthyphenmin`, ale ruší implicitní `\message` i nastavení `\frenchspacing`. Místo toho se předpokládá, že budou definována makra `\lg:`⟨*značka*⟩, ve kterých budou tyto věci (a možná mnoho dalších jako třeba nastavení implicitního fontu, nastavení `\lccode`) řešeny pro každý jazyk individuálně.

$\mathcal{CS}$plain definuje kódovací přepínače pro vzory dělení slov: `\corklangs`, `\iltwolangs` a `\unicodelangs`. Funguje to takto: `\corklangs` ... `\cslang` (nyní je aktivní vzor dělení `\csCork`) a dále třeba ... `\iltwolangs` ... `\cslang` (nyní je aktivní vzor dělení `\csILtwo`). Makra `\iltwolangs`, `\corklangs`, a `\unicodelangs` jsou spuštěna při `\input il2code`, `\input t1code` a `\input ucode`, takže uživatel to typicky nemusí řešit.

Dále je potřeba vědět, že v 16bitové mašině můžete načíst vzory dělení `\*ILtwo` a `\*Cork` korektně jen pro češtinu a slovenštinu, protože další vzory dělení jsou čteny pomocí triku s aktivními znaky, který dekóduje 8bitový vstup. Nicméně typicky v 16bitovém TEXu není inicializován 8bitový vstup, takže se to poláme.

S volbou jazyka souvisí též správné nastavení automaticky generovaných slov, jako třeba Kapitola/Chapter, Obrázek/Figure. Tuto problematiku řeší makro OPmac a je to popsáno v technické dokumentaci k makru v sekci 3.5.

11. Balíček OPmac

Od prosince 2012 je součástí $\mathcal{CS}$plainu soubor maker `opmac.tex`. To neznamená, že bude tento soubor maker zaveden do formátu. Soubor `opmac.tex` bude jen přítomen v balíčku $\mathcal{CS}$plain a uživatelé jej budou moci jednoduše použít pomocí `\input opmac`. Momentálně je tento soubor maker ve fázi beta testování. Vlastnosti maker OPmac jsou popsány v samostatném článku a jsou také k dohledání na `http://petr.olsak.net/opmac.html`.

Summary: $\mathcal{CS}$plain of Year 2012

$\mathcal{CS}$plain existed since 1994 and it is a *gentle* extension of plainTEX so that it is possible to work in Czech and Slovak with it. This was true until October 2012, when the significant revisions and additions of $\mathcal{CS}$plain was made. The main change was done as a result of the decision to set the default input encoding of $\mathcal{CS}$plain to UTF8. Moreover $\mathcal{CS}$plain has many new properties now: the possibility of loading of all sorts of hyphenation patterns available for many languages, interoperability with 16-bit TEX engines (LuaTEX, XeTEX), an effective work with fonts including mathematical fonts, easy switching of internal encoding including Unicode, user-friendly macro package OPmac. In the default configuration $\mathcal{CS}$plain remains still as gentle extension of plainTEX backward compatible with previous versions. New options are easily accessible via `\input` and if they are used it is far to speak only about *gentle* extension. On the contrary, it is a strong competitor of all huge macro extensions of TEX and it wins by simplicity, efficiency of solutions, directness and ease of use. New $\mathcal{CS}$plain is available at `http://petr.olsak.net/csplain.html`.

Petr Olšák
petr@olsak.net

Jak na to – Adresní štítky

Peter Wilson

Abstrakt

Článek ukazuje, jak je možné použít a modifikovat LaTeXový balíček envlab k sazbě různých adresních štítků. Dále článek ukazuje, jak je možné vytvořit knížku s obrázky, kde rychlým otáčením stránek vzniká dojem animace.

Klíčová slova: LaTeX, balíček envlab, adresní štítky, kniha s animací.

Remember still that the loftier minde
That in this world doth seek to glister so,
Blowne on this rock by fonde vainglorious winde,
Falls headlong down to everlasting wo.

The Ship of Safegarde
Barnabe Googe, 1569

Cílem tohoto článku je ukázat čtenáři krátké kousky kódu, které mohou vyřešit některé z jeho problémů. Doufám, že situaci ještě více nezkomplikuji v důsledku mých chyb. Opravy, poznámky a návrhy na změny budou vždy vítány.

Addresses are given to us to conceal our whereabouts.

Reginald in Russia
Saki

1. Seznamy adres

Celý rok jsem manželce sliboval, že jí vytisknu štítky s adresami na naše vánoční pohlednice. Šel jsem dokonce tak daleko, že jsem si na to koupil program běžící pod operačním systémem z minulého století. Nakonec jsem se rozhodl použít LaTeX a balíček EnvLab Borise Veytsmana [1].

Po několika experimentech jsem vytvořil balíček myenvlab, který mi umožnil nastavovat parametry, jak jsem chtěl. Zjistil jsem, že musím použít příkaz `\SetLabel` balíčku EnvLab, aby pozice adresy odpovídala štítkům, které jsem chtěl použít. Také jsem si makra upravil tak, že každá adresa mohla být orámována makrem `\fbox`, abych si mohl adresy vytisknout na obyčejný papír a ověřit, zda jsou adresy umístěny přesně tak, jak mají být na štítcích. Další informace se dočtete v dokumentaci k balíčku.

Z anglického originálu *Glisterings* [2] přeložil Jan Šustek.

doi: 10.5300/2012-1/59

```
% soubor myenvlab.sty
\usepackage[avery5160label,
            noprintbarcodes,
            nocapaddress]{envlab}
\SetLabel{4.19in}{1.23in}{0.73in}{0.16in}%
         {0.19in}{2}{7}
\newif\ifboxlabel
\let\oldPrintLabel\PrintLabel
\renewcommand{\PrintLabel}[1]{%
  \ifboxlabel
    {\fboxsep-\fboxrule
    \fbox{\oldPrintLabel{#1}}}%
  \else
    \oldPrintLabel{#1}%
  \fi}
\boxlabeltrue
\endinput
```

Balíček myenvlab pak lze načíst v normálním souboru a tak vytisknout adresní štítky. Makro `\adresa` definuje formát jména a adresy na štítku.

```
% soubor vanoce.tex
\documentclass[12pt]{letter}
\usepackage{myenvlab}
%%\boxlabelfalse
\newcommand{\adresa}[1]{\mlabel{}{#1}}
\startlabels
\begin{document}
\adresa{Josef První \\
  Nějaká ulice \\
  Nějaké město \\
  12345}%
\adresa{Karel Druhý \\
  Jiná ulice \\
  Jiné město \\
  67890}%
\adresa{Martin Třetí \\
  Úplně jiná ulice \\
  Úplně jiné město \\
  13579}%
\end{document}
```

Když jsem vytiskl štítky s adresami, uvědomil jsem si, že pravděpodobně budu později potřebovat i jiné štítky. Manželka navíc říkala, že bychom mohli založit nový sešit s adresami, protože ten starý už byl skoro nečitelný, jak se v něm pořád přepisovalo, mazalo a přidávalo. Hrál jsem si s myšlenkou upravit BibTeX jako databázi adres, ale naštěstí jsem zaváhal dříve, než jsem začal v jazyku BibTeXu programovat. V současnosti mám všechny adresy uloženy v souboru, který vypadá následovně.

```
% soubor seznamadres.tex
%%% \adresa{jmeno}{adresa}{telefon}{email}{poznámky}
\newcommand{\PrvniJ}{\adresa{Josef První}%
  {Nějaká ulice \\
    Nějaké město \\
    12345}%
  {765 432 109}%
  {\url{josef1@nekde.cz}}%
  {Narozen 29.2.1996.}}
\newcommand{\DruhyK}{\adresa{Karel Druhý}%
  {Jiná ulice \\
    Jiné město \\
    67890}%
  {234 567 890}%
  {\url{karel2@jinde.cz}}%
  {Pes kouše.}}
\endinput
```

Sešit s adresami lze jednoduše vysázet vhodnou definicí makra `\adresa`. Například zde makro `\adresa` vkládá své argumenty do prostředí `minipage`.

```
% soubor sesitadres.tex
\documentclass[12pt,twocolumn]{article}
\usepackage{url}
\newcommand{\adresa}[5]{\noindent
  \begin{minipage}{\linewidth}\raggedright
    #1 \\
    #2 \\
    #3 \\
    #4 \\
    #5
  \end{minipage}%
  \bigskip}
\begin{document}
\input{seznamadres}
```

```
\DruhyK
\PrvniJ
\end{document}
```

Pokud makro `\adresa` předefinujeme tak, že bude používat pouze první dva argumenty, můžeme štítky na Vánoce vysázet následovně.

```
% soubor vanoce.tex
\documentclass[12pt]{letter}
\usepackage{url}
\usepackage{myenvlab}
\newcommand{\adresa}[5]{\mlabel{}{#1\\#2}}
\startlabels
\begin{document}
\input{seznamadres}
\DruhyK
\PrvniJ
\end{document}
```

> Now, *here*, you see, it takes all the running
> *you* can do, to keep in the same place.
> If you want to go somewhere else, you must
> run at least twice as fast as that.
>
> *Through the Looking Glass*
> Lewis Carroll

2. Knížky s animacemi

Při procházení svých starších souborů jsem našel jeden napsaný Jeremy Gibbonsem do jeho seriálu *Hey — it works!*, který ovšem nebyl publikován. Jeremy byl tak hodný, že mi dal svolení ke zveřejnění zde.

Když jsem byl malý, otec dělával malé knížečky, ve kterých byly obrázky, na každé stránce drobně pozměněné, aby rychlým procházením stránek vznikal dojem animace. Nedávno James Willans z Yorku měl dotaz na `comp.text.tex`, jak lze tohoto efektu dosáhnout v LaTeXu. Na dotaz odpověděl Michael Liebling. Tady ukáži zjednodušenou verzi Lieblingovy myšlenky.

V první řadě potřebujeme mít připraveno několik obrázků stejné velikosti. Udělal jsem v METAPOST u obrázky běžícího člověka v různých pozicích.

Soubor running.mp s těmito obrázky je možné stáhnout ze stránek TUGboatu http://tug.org/TUGboat/Contents/contents27-2.html.

Dále musíme zjistit, kolikátý obrázek se má vykreslit na aktuální straně. Jedná se o zbytek po dělení čísla stránky počtem obrázků.

```
\newcount\cislo
\def\spoctizbytek#1#2{%
  \cislo=#2\relax
  \divide\cislo by #1\relax
  \multiply\cislo by #1\relax
  \multiply\cislo by -1\relax
  \advance\cislo by #2\relax}
```

Například `\spoctizbytek{12}{\thepage}` spočítá číslo stránky modulo 12 a výsledek uloží do registru `\cislo`.

Také musíme určit, o jakou vzdálenost se má obrázek na další stránce posunout. Od šířky stránky odečteme šířku obrázku a tuto vzdálenost podělíme celkovým počtem stran.

```
\newcount\pocetstran
\pocetstran=100
\setbox0=\hbox{\includegraphics[scale=0.5]{running.0}}
\newdimen\vzdalenost
\vzdalenost=\textwidth
\advance\vzdalenost by -\wd0
\divide\vzdalenost by \pocetstran
```

Nakonec použijeme styl strany `plain` a na každé straně umístíme správný obrázek na správné místo.

```
\makeatletter
\newdimen\posun
\def\@oddfoot{%
  \posun=\vzdalenost
  \multiply\posun by \thepage
  \hskip\posun
  \spoctizbytek{12}{\thepage}%
  \includegraphics[scale=0.5]{running.\the\cislo}%
  \hfil}
\let\@evenfoot\@oddfoot
\makeatother
```

Zbývá už jen vygenerovat správný počet stránek.

```
\loop
  \mbox{}\newpage
\ifnum \pocetstran>0
  \advance\pocetstran by -1
\repeat
```

Seznam literatury

[1] Veytsman, Boris. Printing Envelopes and Labels in LaTeX 2_ε: EnvLab Package User Guide. June 1996. Dostupné na CTAN v adresáři `latex/macros/contrib/envlab`

[2] Wilson, Peter. Glisterings. *TUGboat*, 27(2):119–120, 2006.

Summary: Envelope Labels

This paper shows how to use and modify the envlab package to typeset address books. The paper also shows how to typeset animated booklets.

Key words: LaTeX, envlab package, address book, animated book.

Peter Wilson, herries.press@earthlink.net
18912 8th Ave. SW
Normandy Park, WA 98166 USA

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Michel Charpentier
Počet výtisků:	340
Uzávěrka:	1. 12. 2012
Odpovědný redaktor:	Zdeněk Wagner
Technický redaktor:	Petr Sojka
Redakční rada:	Zdeněk Wagner (šéfredaktor), Ján Buša, Jiří Demel, Tomáš Hála, Jaromír Kuben, Michal Růžička, Jiří Rybička, Petr Sojka, Pavel Stříž, Jan Šustek
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. +420 541 245 548
Adresa:	CSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6
Email:	`cstug@cstug.cz`

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz`

www server sdružení:
`http://www.cstug.cz`

CONTENTS