Motto: Certainly, if I were a publishing house, if I were in the publishing bussiness myself, I would have probably had ten different versions of TeX by now for ten different complicated projects that had come in. They would all look almost the same as TeX, but no one else would have this program—they wouldn't need it, they're not doing exactly the book that my publishing house was doing.

Donald E. Knuth, Prague, March 1996

EncTeX — změny konverzních tabulek TeXu

Petr Olšák

Článek popisuje jednoduchou úpravu TeXu, která umožňuje přímý přístup k interním vektorům `xord` a `xchr` používaným pro konverzi mezi vstupním kódováním a vnitřním kódováním TeXu. Například uživatelé emTeXu v DOSu a OS/2 mohou zasahovat do těchto vektorů prostřednictvím tzv. `tcp` tabulek, zatímco v UNIXu takto pružné řešení implicitně neexistovalo. Popsaná modifikace TeXu víceméně zastupuje vlastnosti TCP tabulek. Protože je tato modifikace provedena na úrovni změnového souboru `tex.ch` v jazyce WEB, je použitelná všude tam, kde se TeX implementuje ze zdrojových textů. Odzkoušena byla v UNIXových systémech s implementací TeXu `web2c`.

Na těchto implementacích TeXu jsme dosud měli dvě možnosti, jak ovlivnit kódovací algoritmy TeXu. První možností bylo použití záplaty pana Škarvady [3]. Toto řešení mělo přímo v sobě zabudováno několik kódových tabulek, ale tyto tabulky nebylo snadné měnit uživatelem, protože byly součástí zdrojového kódu k TeXu. Načtené kódování se neukládalo po inicializaci do formátu. To mi nepřipadalo příliš pružné.

Druhá možnost spočívala v použití tzv. `tcx` souborů, které byly obdobou známých `tcp` tabulek. Tato možnost ale byla ve zdrojovém kódu TeXu odkomentovaná se slovy: „`tcx` files are probably a bad idea, since they make TeX source documents unportable. Try the `inputenc` LaTeX package.“ S těmito slovy bych byl ochoten polemizovat. Myslím si, že kdyby uživatelé UNIXových instalací našli obdobu `tcp` tabulek, pak by to zvláště v našich zeměpisných šířkách mnozí uvítali. Autor zmíněného sdělení o „špatné ideji“ asi nepoužívá český jazyk. Kdyby jej používal, kdyby byl nucen číst `logy` plné nesrozumitelných stříšek nebo zapsané v odlišném kódování a kdyby nemohl psát kontrolní sekvence ve svém jazyce (například `\příkaz`), asi by si znovu rozmyslel, jakou funkci přidělil vektorům `xord` a `xchr` sám autor TeXu. Možná by pak přestal hovořit o „špatné ideji“. Já osobně mám z citovaného sdělení pocit, že se automaticky předpokládá, že TeX je používán vždy jen prostřednictvím LaTeXu. Myslím si, že právě to je „špatná idea“.

Bohužel, dokumentace k `tcx` souborům zcela chybí, takže jen letmým pohledem do zdrojového kódu lze tušit, že to také ukládalo kódovací vektory čtené z `tcx` souborů do formátu, a tím bylo možné instalovat formáty závislé na kódování, například `csplain.fmt` a `csplain-kam.fmt`.

Moje úprava zdrojového kódu `tex.ch` v instalaci `web2c` jde ještě dál, protože zahrnuje výhody druhého přístupu, a navíc jsou veškeré tabulky načítány za běhu

TEXu jednoduše prostřednictvím `\input`. Implementoval jsem totiž do TEXu tři nové primitivy `\xordcode`, `\xchrcode` a `\xprncode`, pomocí nichž je možný přímý přístup k vektorům `xord` a `xchr`. Novinkou je též skutečnost, že se stávají tyto vektory zcela nezávislými, což otevírá další široké možnosti, které v předchozích řešeních nebyly vůbec možné.

Technický úvod do problematiky

Vektory `xord` a `xchr` mají velikost 256 bytů a obsahují informaci o překódování znaku vstupujícího do TEXu nebo vystupujícího na terminál a do textových souborů. Jedná se o pole vestavěná do programu, přes která jsou filtrovány veškeré textové vstupní a výstupní informace. Má-li znak na vstupu kód x a chceme, aby měl uvnitř TEXu kód y, pak musí být nastaven vektor `xord` tak, aby $\texttt{xord}[x] = y$. Při zpětném výstupu znaku na terminál a do logu a do souborů zpracovávaných pomocí `\write` platí tato pravidla: Není-li znak s kódem y označen jako „tisknutelný", pak vystupuje pomocí přepisu `^^`*kód* y. Je-li tisknutelný, pak vystupuje s kódem $x = \texttt{xchr}[y]$.

Standardně bývají v systémech s kódem ASCII nastaveny hodnoty těchto vektorů tak, že $\texttt{xord}[i] = \texttt{xchr}[i] = i$ pro všechna $i \in 0 \ldots 255$. Na systémech, které nepoužívají ASCII, se může mapovat 94 tisknutelných ASCII znaků jinam. Mimoto je deklarována vlastnost „tisknutelnosti" znaku v ASCII takto: Znak je tisknutelný, pokud má kód $y \in 32 \ldots 126$. Ostatní znaky se považují za netisknutelné a TEX je standardně přepisuje pomocí dvojité stříšky.

Instalace balíčku encTEX byla odzkoušena na distribuci TEXu `web2c` verze 7. Máme-li k této distribuci zdrojové texty, pak stačí spustit `patch <enctex.patch` v adresáři se souborem `tex.ch` a překompilovat TEX pomocí `make tex`. Instalace je podrobněji popsána v souboru `INSTALL`.

Aplikací přiložené záplaty `patch` se pozmění jediný soubor `tex.ch`. Ostatní soubory včetně pomocných knihoven v jazyce C zůstávají beze změny. Při činnosti „`make tex`" se provede `tangle` na hlavní zdrojový soubor `tex.web` a pozměněný změnový soubor `tex.ch`. Tím vzniká Pascalský kód TEXu `tex.p`, který je pak konvertován do jazyka C skriptem `convert` a kompilován do spustitelné podoby.

Protože je veškerá změna TEXu implementována pouze do změnového souboru `tex.ch`, je vlastně toto řešení nezávislé na použitém operačním systému či implementaci TEXu. Na jiných implementacích je ale jiný výchozí soubor `tex.ch`, takže zde nelze jednoduše provést povel `patch`. Proto je v balíčku zahrnut soubor `enctex.ch`, který obsahuje ty části změnového souboru `tex.ch`, které se týkají encTEXu. Veškeré změny vycházejí ze standardního zdrojového kódu TEXu `tex.web` až na jedinou výjimku: ukládání vektorů `xord` a `xchr` do formátu a čtení z formátu. Zde jsem s výhodou použil již hotové pomocné funkce *dump_things* a *undump_things* napsané pro `web2c` přímo v jazyce C. Pokud by to někdo potřeboval v jiné implementaci, musel by zřejmě použít analogii k *dump_four_ASCII* z `tex.web`.

Po instalaci balíčku je možno přímo nastavovat a číst obsahy vektorů `xord` a `xchr` prostřednictvím primitivů `\xordcode` a `\xchrcode` a dále nastavovat vlastnost „tisknutelnosti" znaku pomocí primitivu `\xprncode`. Syntaxe všech tří nových primitivů je naprosto stejná, jakou známe například u primitivů `\lccode` a `\uccode`. Například:

```
\xordcode"AB="CD  \xchrcode\xordcode"AB="AB \the\xchrcode200
```

nastavuje `xord[0xAB]=0xCD; xchr[xord[0xAB]]=0xAB` a dále vytiskne hodnotu `xchr[200]`.

Na rozdíl od podobných primitivů `\catcode`, `\lccode`, `\sfcode` a dalších však nově zavedené primitivy mají jednu podstatnou výjimku. Reprezentují interní registry TEXu, které vždy mají globální platnost. Proto je nastavení `\xordcode` a `\xchrcode` uvnitř skupiny za všech okolností globální, ačkoli to explicitně nepíšeme. Ústupem z požadavku na možnost lokálního deklarování hodnot jsem dosáhl podstatně větší efektivity výsledného kódu programu.

Primitiv `\xprncode` umožňuje nastavovat vlastnost „tisknutelnosti“ znaku takto: Znak s kódem y je tisknutelný právě tehdy, když je $y \in \{32 \dots 126\}$ nebo `\xprncode`$y > 0$. Napíšeme-li například `\xprncode255=1`, bude tisknutelný znak s kódem 255. Na druhé straně, nastavení ``\xprncode`a`` třeba na nulu nemá na chování programu žádný vliv, protože kód znaku „`a`“ je v množině $\{32 \dots 126\}$. Tímto opatřením program vykazuje určitý pud sebezáchovy, protože zlý uživatel by mu mohl nastavit všechny znaky jako netisknutelné a program by ztratil schopnost se vyjadřovat. Hodnoty `\xprncode` lze nastavit jako u ostatních nových primitivů v rozsahu nula až 255, ovšem otázka tisknutelnosti je totožná s otázkou na kladnou hodnotu bez ohledu na to, jak velká tato hodnota je.

Výchozí hodnoty pro kódování v době iniTEXu jsou následující:

`\xordcode`$i = i$ pro $i \in \{128 \dots 255\}$,
`\xchrcode`$i = i$ pro $i \in \{128 \dots 255\}$,
`\xprncode`$i = 0$ pro $i \in \{0 \dots 31, 127 \dots 255\}$,
`\xprncode`$i = 1$ pro $i \in \{32 \dots 126\}$.

Hodnoty `\xordcode`i a `\xchrcode`i pro $i \in \{0 \dots 127\}$ jsou závislé na operačním systému. Pokud systém pracuje v rámci této množiny s ASCII kódováním (což je obvyklé), pak je `\xordcode`$i = i$ a `\xchrcode`$i = i$. Na jiných systémech jde hlavně o to, jak systém kóduje 95 základních tisknutelných znaků, které jsou v ASCII na pozicích $\{32 \dots 126\}$.

Hodnoty `\xordcode`, `\xchrcode` a `\xprncode` se v této úpravě TEXu ukládají do formátu `fmt`, z něhož jsou znovu načteny při běhu produkční verze TEXu.

Než budeme zasahovat do hodnot vektorů `xord` a `xchr`, je dobré si uvědomit několik souvislostí. K tomu použiji matematické značení a terminologii. Nechť $X = \{0 \dots 255\}$ je množina obsahující vtupní kódy a $Y = X$ je matematicky tatáž množina, ovšem budeme ji používat pro označení vnitřních kódů TEXu. Nechť dále $Y_p \subseteq Y$ je množina všech tisknutelných znaků. Je:

$$Y_p = \{y; \texttt{\char`\\xprncode}\, y > 0\} \cup \{32 \dots 126\}.$$

Je zřejmé, že nastavení hodnot vektoru `xchr` na množině $Y \setminus Y_p$ nemá pro chování programu žádný vliv. Dále označme vstupní zobrazení $I : X \to Y$ definované hodnotami vektoru `xord` a výstupní zobrazení $O : Y_p \to X$ definované hodnotami vektoru `xchr`. Výchozí nastavení je voleno tak, že I je prosté a na a O je prosté. Nastavíme-li `\xprncode` tak, že $Y_p = Y$, pak i zobrazení O je prosté a na. Navíc je inverzní k zobrazení I. Ovšem tento ideál platí pouze do prvního zásahu do vektorů `xord` a `xchr`. Nechť například máme $x \neq y$ a $x \in X$, $y \in Y$. Proveďme jedno elementární přehození:

$$\texttt{xord}[x] = y, \quad \texttt{xchr}[y] = x. \tag{1}$$

Nyní není ani zobrazení I ani O prosté. Je totiž $\texttt{xord[}x\texttt{]} = \texttt{xord[}y\texttt{]}$ a totéž platí pro vektor `xchr`. Aby byla po dalších n elementárních přehozeních naše zobrazení opět prostá, musí existovat posloupnost $x_0, \dots x_n$ taková, že

$$x_0 = x, \quad x_1 = y \quad \text{a} \quad \texttt{xord}[x_i] = x_{i+1} \quad \text{pro všechna } i \in \{0 \dots n-1\}$$

a dále musí $\texttt{xord[}x_n\texttt{]} = x$. Podobná vlastnost musí platit pro vektor `xchr`. Často se při prohazování kódů zaměřujeme jen na podmnožinu X (například abecední znaky, tisknutelné znaky, dosažitelné znaky v daném kódování). Pak se nesmíme divit, že výsledek našeho nastavení není prosté zobrazení a tím není splněna ani podmínka, že $O = I^{-1}$, přestože jsme důsledně prováděli elementární přehození podle (1) pro oba vektory vždy současně. Většinou nám ale skutečnost, že nejsou naše zobrazení prostá, v dané aplikaci moc nevadí.

Kódovací tabulky

Protože změna vektorů `xord` a `xchr` může totálně rozhodit chování TeXu zcela k nepoznání, doporučuji používat určité soubory, které nastaví požadované kódování, a dále s primitivy `\xordcode`, `\xchrcode` a `\xprncode` za běhu TeXu moc nelaškovat. V balíčku encTeX jsou k dispozici soubory, které změnu vektorů pro běžná kódování definují. Tyto soubory mají obvyklou příponu `tex`. Říkáme jim kódovací tabulky. Rozlišujeme dva typy kódovacích tabulek.

První typ kódovacích tabulek

První typ tabulek deklaruje vnitřní kódování TeXu ve vztahu ke kódování, které je běžně používané v hostitelském operačním systému. Máme-li například v systému kódování ISO-8859-2 a vnitřní kódování TeXu volíme podle Corku (kódování je označováno jako T1), pak tabulka musí předefinovat `xord` vektor tak, aby mapoval znaky z ISO-8859-2 do T1 a vektor `xchr` musí převádět zpátky z T1 do kódování systému.

Tento typ tabulek obsahuje v názvu souboru vstupní i cílové vnitřní kódování TeXu. Například tabulka `iso2-t1.tex` vypadá následovně:

```
%%% The encoding table, v.Aug.1997 (C) Petr Ol\v s\'ak
%%% input: ISO-8859-2, internal TeX: T1 alias Cork

\input encmacro  \input t1macro
```

```
%             input  TeX   lc   uc  cat       sequence
\setcharcode   "C1  "C1  "E1  "C1  11  \texaccent \'A
\setcharcode   "E1  "E1  "E1  "C1  11  \texaccent \'a
\setcharcode   "C4  "C4  "E4  "C4  11  \texaccent \"A
\setcharcode   "E4  "E4  "E4  "C4  11  \texaccent \"a
\setcharcode   "C8  "83  "A3  "83  11  \texaccent \v C
\setcharcode   "E8  "A3  "A3  "83  11  \texaccent \v c
\setcharcode   "CF  "84  "A4  "84  11  \texaccent \v D
...
\setcharcode   "A7  "9F    0    0  12  \texmacro  \S
\setactivecode "D7  "03   {$\times$}
...
\redefaccent \'
\redefaccent \v
...
```

Každá tabulka prvního typu čte soubor `encmacro` s definicemi maker `\setcharcode`, `\setactivecode`, `\texaccent`, `\texmacro` a `\redefaccent`.

• `\setcharcode #1 #2 #3 #4 #5` deklaruje TeXovské kódy pro jeden znak. Nastaví `xord[#1]=#2`, `xchr[#2]=#1`, `\xprncode#2=1` a postupně nastaví `\lccode`, `\uccode` a `\catcode` znaku s kódem `#2` na hodnoty `#3`, `#4` a `#5`. Je-li `#1` otazník, pak se `xord` a `xchr` nenastaví a `\xprncode=0`.

• `\setactivecode #1 #2 {`*definice*`}` nastaví rovněž hodnoty kódovacích vektorů `xord[#1]=#2`, `xchr[#2]=#1`, `\xprncode#2=1` a dále znak s kódem `#2` deklaruje jako aktivní a definuje jej podle *definice*. Je-li `#1` otazník, `\setactivecode` neprovede vůbec nic. Například v ISO-8859-2 je na pozici `"D7` znak „×“. My jsme v naší ukázce tomuto znaku přidělili vnitřní kód TeXu `"03` a znak bude expandovat na `$\times$`.

• `\texaccent #1` připravuje expanzi zápisu `#1` na znak s kódem `#2` z naposledy použitého `\setcharcode`. Například zápis `\v C` bude podle naší ukázky expandovat na znak s kódem `"83`. Pokud zápis pro akcent není v tabulce uveden, zůstává v původním významu, tj. třeba `\v g` expanduje na primitiv `\accent`, který usadí háček nad písmeno g. K aktivaci všech zápisů `\texaccent` dojde až po použití makra `\redefaccent` (viz níže).

• `\texmacro #1` deklaruje makro `#1` tak, že bude expandovat na znak s kódem `#2` z naposledy použitého `\setcharcode`. K předefinování makra `#1` dojde (na rozdíl od `\texaccent`) okamžitě. Například makro `\S` bude podle naší ukázky expandovat na znak s kódem `"9F`, protože na této pozici je podle Corku znak „§“.

• `\redefaccent #1` aktivuje expanzi zápisů podle `\texaccent` pro jeden konkrétní akcent `#1`.

Kromě toho je na začátku tabulky čten soubor definic závislých na kódování textového fontu TeXu. V naší ukázce jde například o soubor `t1macro`. Definují se tam sekvence `\promile`, `\clqq` a další. Jedná se o jednoduchou obdobu `fd` souborů z NFSS.

Může se stát, že nechceme uvedená makra použít, ale hodnoty z tabulky načíst chceme. Pak můžeme přistoupit k následujícímu triku: Definujeme si makra `\setcharcode` až `\redefaccent` sami a dále provedeme načtení tabulky takto:

```
\let\originput=\input \def\input #1 {}
\originput il2-t1
\let\input=\originput
```

V balíčku encTeX jsou připraveny tyto tabulky prvního druhu:

Název souboru	vstupní kódování	vnitřní kódování TEXu
`il2-csf.tex`	ISO8859-2	CS-font
`kam-csf.tex`	Kamenických	CS-font
`1250-csf.tex`	CP1250, MS-Windows	CS-font
`852-csf.tex`	CP852, PC Latin2	CS-font
`il2-t1.tex`	ISO8859-2	T1 alias Cork
`kam-t1.tex`	Kamenických	T1 alias Cork
`1250-t1.tex`	CP1250, MS-Windows	T1 alias Cork
`852-t1.tex`	CP852, PC Latin2	T1 alias Cork

Za zmínku stojí první uvedená tabulka `il2-csf.tex`, protože ta jediná ponechává vektory `xord` a `xchr` beze změny. Tuto tabulku je tedy možné použít i v TEXu, který neobsahuje rozšíření encTEX. Pro formát `csplain` by příslušný soubor `csplain.ini` mohl vypadat třeba takto:

```
\input csfonts    % re-defines primitive \font
\input plain      % format Plain
\restorefont      % original meaning of primitive \font
\input il2-csf    % input ISO8859-2, internal TeX: CS-font
\input hyphen.lan % czech / slovak hyphenation pattern
\input plaina4    % \hsize and \vsize for A4
\everyjob=\expandafter{\the\everyjob
   \message{The format: plain-il2-cs <Sep. 1997>.}
   \message{The cs-fonts are preloaded and A4 size predefined.}}
\dump
```

Takto generovaný formát by se od standardního formátu `csplain` neměl mnoho lišit. Pokud v tomto `ini` souboru zaměníme kupříkladu slovo `il2` za `1250`, pak vytvoříme identický formát, který je ale namísto UNIXu určen pro prostředí MS-Windows. V tomto případě ovšem už budeme potřebovat rozšíření TEXu s názvem encTEX.

Pokud bychom chtěli vytvořit analogii formátu `csplain` pro použití s DC/EC fonty, pak bych doporučoval tento postup:

```
\input noprefnt   % \font\preloaded is not preloaded
\input plain      % format Plain
\restorefont      % original meaning of primitive \font
```

```
\input dcfonts    % load text-style fonts
\input il2-t1     % input ISO8859-2, internal TeX: Cork
\input hyphen.lan % czech / slovak hyphenation pattern
\input plaina4    % \hsize and \vsize for A4
\everyjob=\expandafter{\the\everyjob
   \message{The format: plain-il2-dc <Sep. 1997>.}
   \message{The dc+cm-fonts are preloaded and A4 size predefined.}}
\dump
```

Necháme tedy matematické fonty načíst originálním makrem `plain` a potom předefinujeme textové fonty v souboru `dcfonts` jednoduše takto:

```
\font\tenrm=dcr10  \font\tenbf=dcbx10  \font\tenit=dcti10
\font\tentt=dctt10 \font\tensl=dcsl10  \tenrm
```

Upozorňuji, že tento formát může v sobě skrývat podstatné odlišnosti od originálního formátu `csplain`. Buďte při jeho používání opatrní.

Pokud budeme chtít vytvořit další tabulky, které mají stejné vnitřní kódování ale odlišné vstupní, pak většinou stačí vyjít z existující tabulky a pozměnit jen první sloupec.

Druhý typ kódovacích tabulek

Druhý typ tabulek provádí překódování pouze na vstupní straně TEXu. Poznáme je podle toho, že nemají na konci názvu značku pro vnitřní kódování TEXu (tj. `t1` nebo `csf`), ale značku používanou pro kódování operačního systému (např. `il2`, `kam`). Třeba tabulka `kam-il2.tex` provádí na vstupní straně konverzi z kódování kamenických do kódování ISO8859-2. Tento typ tabulek pozměňuje pouze vektor `xchr`, ale výstupní vektor `xord` ponechává beze změny. Takovou tabulku použijeme, pokud TEXem načítáme soubor, který je v jiném kódování, než běžně používáme na našem operačním systému. Přitom výstup do `log`, `aux` apod. ponecháme v kódování podle našeho systému. Tyto změny kódování je možné provádět i v průběhu zpracování jediného dokumentu.

Druhý typ tabulek navazuje na vstupní kódování deklarované dříve tabulkou prvního typu. Nastavení vnitřního kódování TEXu není vůbec druhým typem tabulek měněno. Uvedeme příklad. Při generování formátu jsme použili tabulku prvního typu `il2-t1.tex`, takže vnitřní kódování máme podle Corku. Nyní můžeme při zpracování dokumentu na přechodnou dobu vybrat některou z tabulek `*-il2.tex`, třeba:

```
\input kam-il2
\input dokument
\restoreinputencoding
nyní mohu pracovat v~původním kódování...
```

V době, kdy probíhá načítání souboru `dokument.tex` se provádí překódování z Kamenických do T1, uvnitř TEXu se vše zpracovává v T1 a výstup na terminál a do `logu` máme v ISO8859-2. V tomto kódování je také zapsán další text pod `\restoreinputencoding`. Tabulka totiž deklaruje toto makro, aby byl možný návrat k původnímu nastavení vektoru `xord`.

Při použití tabulek druhého typu musíme dát velký pozor, abychom něco neudělali špatně. V našem příkladě jsou všechny výstupy do souborů typu `aux` v ISO-8859-2, takže je při opakovaném spuštění TEXu nesmíme načítat v okamžiku, kdy máme nastaven vstupní kód podle Kamenických. To je také důvod, proč nedoporučuji generovat formát příkazem `\dump` v situaci, kdy máme načtenou tabulku druhého typu.

O kompatibilitě neboli slučitelnosti

Upravený TEX (encTEX) projde zcela bez problémů testem TRIP s výjimkou jediného případu: počet vložených „`multiletter control sequences`" je o tři větší, než v originálním TEXu.

Veškeré změny TEXu, které nemění dosavadní chování TEXu, ale pouze přidávají nové primitivy, jsou zpětně zcela kompatibilní s originálním TEXem. Tím se myslí, že dokumenty napsané pro originální TEX se budou v pozměněném TEXu chovat zcela stejně. Výjimkou je snad dokument s konstrukcí typu `\ifx\xordcode\undefined`, ale četnost výskytu takových konstrukcí v dokumentech pro originální TEX je prakticky nulová.

Je třeba si ale uvědomit, že v pozměněném TEXu lze psát dokumenty a makra, které nejsou s originálním TEXem kompatibilní. To je nevýhoda všech rozšíření, která přidávají primitivy. Je zcela jedno, zda jsou nové primitivy přidány do TEXu „natvrdo" (jako v případě encTEXu nebo pdfTEXu), nebo zda je přístup k novým primitivům otevřen až po speciální volbě na příkazovém řádku při inicializaci formátu (NTS, MLTEX). Zde záleží na tom, jak moc se používání nových primitivů rozšíří, a kdo bude dohlížet, aby rozšířená množina primitivů byla celosvětově standardizována. V případě mého jednoduchého rozšíření si nekladu žádný nárok na to, aby se to dostalo do nějakých standardů. Prostě jsem si to udělal pro svoje potřeby a pokud se to líbí ještě někomu jinému, má možnost mé rozšíření použít s vědomím, že chce-li psát přenositelné dokumenty, nebude rozšířené primitivy ani makra, která tyto primitivy volají, ve svém dokumentu používat.

Chce-li administrátor systému instalovat TEXovský formát včetně volby vhodného kódování, pak při inicializaci formátu může zavolat některou tabulku prvního typu a potom, těsně před povelem `\dump`, zakáže uživatelům sahat na kódovací vektory:

```
\let\xordcode=\undefined
\let\xchrcode=\undefined
\let\xprncode=\undefined
```

Tím má zaručeno, že uživatel nebude používat nestandardní rozšíření této implementace TEXu. Na veřejných sítích bych asi takové řešení doporučil. Tímto způsobem navíc vektory `xord` a `xchr` plní funkci, kvůli které je do TEXu už dávno zahrnul jeho autor: starají se o odstínění mezi kódovacími specifiky jednotlivých operačních systémů a pevně zvoleným vnitřním kódováním TEXu.

Původní záměr autora TEXu spočíval v tom, že nastavení kódovacích vektorů provede administrátor systému zásahem do změnového souboru `tex.ch` v době kompilace TEXu a tyto vektory budou pevně pro každou implementaci nastaveny. Bohužel, tato praxe se v případě implementací TEXu pro UNIX příliš nerozšířila, protože by to od administrátora vyžadovalo speciální aktivitu navíc, která navíc pro nepoučeného nebyla zcela triviální. Zavedení primitivů `\xordcode` a `\xchrcode` a tabulek prvního typu posunuje úpravu kódovacích vektorů až na dobu inicializace formátu, ale také podstatně zjednodušuje jejich nastavení. To bych považoval za výhodu mého řešení. Já jsem vlastně neudělal nic jiného, než že jsem z pozice administrátora systému pozměnil změnový soubor `tex.ch` a změnu jsem udělal tak, aby výsledek mé práce byl pokud možno flexibilní.

Můžeme si položit otázku, proč profesor Knuth už dávno primitivy sahající k vektorům `xord` a `xchr` nezavedl. Zřejmě chtěl, aby se všechna TEXovská makra chovala zcela stejně na všech implementacích. Přímý přístup k vektorům `xord` a `xchr` byl pro něj v takovém případě nepřijatelný. V konstrukcích typu `\ifnum\xordcode‘@=‘@` se totiž může makro větvit podle toho, zda hostitelský operační systém používá stejné kódování jako ASCII nebo ne.

Poznamenejme, že se autorovi TEXu nepodařilo zcela zajistit stejné chování maker na všech implementacích. Když si nechám vypsat do souboru znak, který je v jednom operačním systému tisknutelný a v druhém ne, pak mám dva různé výsledky: buď přímo znak nebo `^^`*kód*. Když znovu v druhém běhu TEXu takový soubor přečtu, ale nejprve nastavím `\catcode‘^=12`, pak se může mé makro větvit podle toho, zda je testovaný znak v dané implementaci tisknutelný nebo nikoli.

Závěrečná poznámka

Každý si může pro své potřeby upravit TEX přímo ze zdrojového kódu. Viz též úvodní motto. Přitom je to úkol jednodušší, než by se na první pohled mohlo zdát. Mě osobně stačilo jeden večer listovat v [1] a vše si důkladně rozmyslet. Pak jsem druhý den dopoledne dostal myšlenku do počítače a vše vyzkoušel a odpoledne jsem napsal tento články. A věc je hotova. To vše díky velmi dobře dokumentovanému programu TEX.

Reference

[1] Donald Knuth. *TEX: The program*, volume B of *Computers & Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[2] Petr Olšák. *encTEX*, balík rozšiřující TEX volně k dostání na `ftp://math.feld.cvut.cz/pub/olsak/enctex`

[3] Libor Škarvada. Záplata rozšiřující TEX volně k dostání na `ftp://ftp.muni.cz/pub/tex/local/cstug/skarvada`