verze Aug. 2013

OPmac – rozšiřující makra plainTEXu

Petr Olšák

www.olsak.net/opmac.html

Obsah

1 Úvod

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.

2 Uživatelská dokumentace

Uživatelská dokumentace je zatím v souboru `opmac-u.tex` a `opmac-u.pdf`. Do tohoto místa ji zahrnu později a prolinkuji ji s technickou dokumentací.

3 Technická dokumentace

Tato část dokumentace je určena pro tvůrce maker, kteří se chtějí zde uvedenými makry inspirovat a případně je přizpůsobit svému požadavku. Předpokládá se znalost TEXu, tj. například aspoň zběžná orientace v TEXbooku naruby. Na tuto knihu je na mnoha místech odkazováno pod zkratkou TBN.

3.1 Základní makra

Na začátku souboru `opmac.tex` zjistíme, zda není soubor čtený podruhé. V takovém případě čtení odmítneme. Ptáme se na to, zda je definováno makro `\OPmacversion`, které vzápětí definujeme. Je-li někdo překvapen, proč jsem nepoužil `\expandafter\endpinput\fi`, může si prostudovat TBN, stranu 358, heslo `\endinput`.

opmac.tex
```
7: \ifx\OPmacversion\undefined \else \endinput \fi
8: \def\OPmacversion{Aug. 2013}
```

Dva pracovní registry:

opmac.tex
```
14: \newcount\tmpnum % auxiliary count
15: \newdimen\tmpdim % auxiliary dimen
```

OPmac nebude nikdy hlásit chyby. Často ale bude psát pomocí `\opwarning` na terminál varování.

opmac.tex
```
17: \def\opwarning#1{\immediate\write16{l.\the\inputlineno\space OPmac WARNING: #1.}}
```

\OPmacversion: 3 \tmpnum: 15, 19, 22–23, 27–28, 35–36, 38–39, 42–43, 45 \tmpdim: 6, 8–9, 32, 41–43, 51 \opwarning: 3, 7, 12, 15–16, 18, 21, 23, 30, 32, 35–38, 40, 42–44, 46, 48–49, 51–52

Makro \addto ⟨*makro*⟩{⟨*tokeny*⟩} přidá na konec ⟨*makra*⟩ dané ⟨*tokeny*⟩.

opmac.tex
```
\long\def\addto#1#2{\expandafter\def\expandafter#1\expandafter{#1#2}}
```

V OPmac budeme pracovat se seznamem \protectlist, který bude obsahovat makra, jež chceme mít tzv. robustní, tj. chceme, aby se při \write v output rutině neexpandovala. Každému makru v seznamu předchází \doprotect, takže seznam \protectlist vypadá takto:

```
\doprotect⟨makro1⟩ \doprotect⟨makro2⟩ ...
```

Seznam budeme spouštět v output rutině s tím, že \doprotect tam bude mít význam makra, které zařídí, aby jeho parametr získal význam \relax. Tím bude zabráněno jeho expanzi. Naprogramujeme \addprotect ⟨*makro*⟩, které zařídí vložení ⟨*makra*⟩ do seznamu.

opmac.tex
```
\def\protectlist{}
\def\addprotect#1{\addto\protectlist{\doprotect#1}}
\addprotect~
```

Některá makra budou fungovat jen v pdfTEXu při nastaveném \pdfoutput=1. Připravíme si tedy test \ifpdftex, který pak použijeme při čtení souboru opmac.tex. Test nikdy nebudeme vkládat do maker, takže při čtení souboru opmac.tex už musí být jasné, zda bude výstup směrován do DVI nebo PDF. Pozdější změna \pdfoutput může způsobit potíže.

opmac.tex
```
\newif\ifpdftex  \pdftextrue
\ifx\pdfoutput\undefined \pdftexfalse
\else \ifnum\pdfoutput=0 \pdftexfalse \fi \fi
```

Makra \sdef a \sxdef umožňují pohodlně definovat kontrolní sekvence ohraničené pomocí \csname...\endcsname.

opmac.tex
```
\def\sdef#1{\expandafter\def\csname#1\endcsname}
\def\sxdef#1{\expandafter\xdef\csname#1\endcsname}
```

Makro \adef umožní nastavit znak na aktivní a rovnou ho definovat, což normálně uvnitř maker není jednoduché (TBN str. 25 a 26). Využijeme toho, že ~ je aktivní znak a pomocí \lccode a \lowercase jej přepíšeme na požadovaný znak. Dostaneme tím aktivní token s požadovanou ASCII hodnotou a tento token definujeme. Pomocí \lccodetiezero vrátíme po přiřazení (tj. po provedení definice) \lccode vlnky na původní hodnotu.

opmac.tex
```
\def\adef#1{\lccode`\~=`#1\catcode`#1=13
   \afterassignment\lccodetiezero
   \lowercase{\def~}%
}
\def\lccodetiezero{\lccode`\~=0 }
```

Makrem \isdefined {⟨*jméno*⟩}\iftrue se ptáme, zda je definovaná \csname⟨*jméno*⟩\endcsname. To závěrečné připojené \iftrue makro sežere, ale uživatel ho píše zejména z toho důvodu, aby mu tato konstrukce fungovala uvnitř vnořených \if..\fi

opmac.tex
```
\def\isdefined #1#2{\expandafter\ifx \csname#1\endcsname \relax
     \csname iffalse\expandafter\endcsname
   \else
     \csname iftrue\expandafter\endcsname
   \fi
}
```

Makro \isinlist ⟨*list*⟩{⟨*tokeny*⟩}\iftrue zjistí, zda ⟨*tokeny*⟩ jsou (jako string) obsaženy v makru ⟨*list*⟩. Přitom sežere \iftrue ze stejných důvodů, jak je uvedeno před chvílí.

opmac.tex
```
\def\isinlist#1#2#3{\def\tmp##1#2##2\end{\def\tmp{##2}%
   \ifx\tmp\empty \csname iffalse\expandafter\endcsname \else
                  \csname iftrue\expandafter\endcsname \fi}%
   \expandafter\tmp#1\endlistsep#2\end
}
```

Makro \isnextchar ⟨znak⟩{⟨co-dělat-při-ano⟩}{⟨co-dělat-při-ne⟩} pracuje poněkud odlišně od předchozích maker. Zjistí, zda následující znak je ⟨znak⟩ a pokud ano, vykoná vnitřek první závorky, jinak vykoná vnitřek druhé závorky. Pomocí \futurelet uloží zkoumaný znak do \next a spustí \isnextcharA.

opmac.tex
```
\def\isnextchar#1#2#3{\def\tmpa{#2}\def\tmpb{#3}%
   \let\tmp=#1\futurelet\next\isnextcharA
}
\def\isnextcharA{\ifx\tmp\next\expandafter\tmpa\else\expandafter\tmpb\fi}
```

Předefinujeme makro \uv z CSplainu. Tam je toto makro navrženo tak, aby mohlo mít za svůj parametr verbatim text. Důsledkem toho nefunguje správně kerning. Považuji za lepší mít správně kerning a případné uvozování verbatim textů řešit třeba pomocí \clqq...\crqq.

opmac.tex
```
\def\uv#1{\clqq#1\crqq}
```

Knuth v souboru plain.tex zanechal řídicí sekvenci \\ v provizorním stavu (cvičení: podívejte se v jakém). Domnívám se, že je lepší ji dát jednoznačný význam \undefined. Některým uživatelům totiž může OPmac připomínat LaTeX a není tedy vyloučeno, že je napadne psát \\. Měli by na to dostat jednoznačnou odpověď: undefined control sequence.

opmac.tex
```
\let\\=\undefined
```

Do pracovního souboru určeného k novému načtení budeme chtít vložit komentáře za znakem procento. K tomu potřebujeme mít procento jako obyčejný znak kategorie 12. Na tento znak se v našem kódu překlopí otazník, Takže \percent expanduje na znak procento s kategorií 12.

opmac.tex
```
{\lccode`\?=`\%  \lowercase{\gdef\percent{?}}}
```

Podobně je naprogramováno makro \bslash, které vytiskne obyčejné zpětné lomítko:

opmac.tex
```
{\lccode`\?=`\\  \lowercase{\gdef\bslash{?}}}
```

Obě makra chceme při \write do souboru nechat v původním stavu:

opmac.tex
```
\addprotect\percent \addprotect\bslash
```

Makro plainTeXu \, funguje jen v matematické sazbě. Uživatel bude chtít makro často použít například mezi číslem a jednotkou v textovém módu: 5\,mm, takže makro předefinujeme.

opmac.tex
```
\def\,{\ifmmode \mskip\thinmuskip \else \kern.166em \fi}
```

3.2 Globální parametry

Zakážeme vdovy a sirotky a dále nastavíme registry pro listingy tiskového materiálu na smysluplnější hodnoty, než jsou implicitní.

opmac.tex
```
\widowpenalty=10000
\clubpenalty=10000
\showboxdepth=7
\showboxbreadth=30
```

Následující makra a registry ovlivní chování klíčových maker OPmac způsobem, jak je popsáno v komentářích. Mnohé z těchto maker a registrů byly zmíněny v uživatelské dokumentaci.

opmac.tex
```
\newdimen\iindent  \iindent=\parindent
   % indentation of items, TOC, captions, list of bib. references
\newdimen\ttindent \ttindent=\parindent
   % indentation in \begtt...\endtt and \verbinput

\def\ttskip{\medskip}       % space above and below \begtt, \verbinput
\mathchardef\ttpenalty=100  % penalty between lines in \begtt, \verbinput
\def\tthook{}               % hook in \begtt, \verbinput
\def\intthook{}             % hook in in-text verbatim
```

\isnextchar: 5, 49 \isnextcharA: 5 \uv: 5 \percent: 5, 10–11, 34, 48 \bslash: 5, 34
\iindent: 5, 16–18, 22, 47–49 \ttindent: 5, 37–39 \ttskip: 37, 39 \ttpenalty: 37, 39
\tthook: 37–39 \intthook: 37

```
78:
79: \def\iiskip{\medskip}     % space above and below \begitems...\enditems
80: \def\bibskip{\smallskip}  % space between bibitems
81:
82: \def\tabstrut{\strut}     % strut in the \table
83: \def\tabiteml{\enspace}   % left material before each \table item
84: \def\tabitemr{\enspace}   % right material after each \table item
85: \def\vvkern{1pt}          % space between vertical lines
86: \def\hhkern{1pt}          % space between horizontal lines
87:
88: \def\multiskip{\medskip}      % space above and below \begmulti...\endmulti
89: \newdimen\colsep \colsep=2em  % space between columns
90:
91: \newdimen\mnoteindent \mnoteindent=10pt % ditance between mnote and text
92: \newdimen\mnotesize   \mnotesize=20mm   % the width of the mnote paragraph
93:
94: \def\picdir{}      % the directory with picture files
95: \def\bibtexhook{}  % hook in \usebibtex and \usebbl macros
96: \def\chaphook{}    % hook in \chap
97: \def\sechook{}     % hook in \sec
98: \def\secchook{}    % hook in \secc
99: \def\cnvhook{}     % hook before conversion of outlines
100: \def\pghook{}      % hook in \output rutine
101: \def\toclinehook{} % hook in \tocline
102: \def\mnotehook{}   % hook in \mnote to corret its vertical position
103: \def\captionhook#1{} % hook in \caption (#1 is "t" or "f")
```

3.3 Loga

V logu \OPmac je pomocí \thefontscale zvětšeno písmeno O. Logo \CS je přepsáno beze změny z CSTEXu. Tím snadno vytvoříme i logo \csplain.

opmac.tex

```
107: \def\OPmac{\leavevmode
108:    \lower.2ex\hbox{\thefontscale[1400]O}\kern-.86em P{\em mac}}
109: \def\CS{$\cal C$\kern-.1667em\lower.5ex\hbox{$\cal S$}}
110: \def\csplain{\CS plain}
```

Troufám si tvrdit, že logo \LaTeX (ačkoli je plainTEXisté asi moc nebudou potřebovat) je v následujícím kódu daleko lépe řešeno, než v samotném LaTEXu. Počítá totiž ve spolupráci s makrem \slantcorr i se sklonem písma při usazování zmenšeného A.

opmac.tex

```
112: \def\LaTeX{\tmpdim=.42ex L\kern-.36em \kern\slantcorr % slant correction
113:   \raise\tmpdim\hbox{\thefontscale[710]A}%
114:   \kern-.15em \kern-\slantcorr \TeX}
115: \def\slantcorr{\expandafter\ignorept\the\fontdimen1\the\font\tmpdim}
```

Loga se občas mohou vyskytnout v nadpisech. Zabezpečíme je tedy proti rozboření při zápisu do REF souboru.

opmac.tex

```
117: \addprotect\TeX   \addprotect\OPmac   \addprotect\CS  \addprotect\LaTeX
```

3.4 Velikosti fontů, řádkování

CSplain od verze ⟨*Nov.-2012*⟩ definuje makro \resizefont ⟨*fontselector*⟩, které změní velikost fontu daného svým přepínačem a tento změněný font si ponechá stejný přepínač. Změna velikosti je dána obsahem makra \sizespec. Tam může být například napsáno `at13pt` nebo `scaled800`. Dále CSplain definuje makro \resizeall, které změní velikost fontů s registrovanými přepínači. Registrování se provádí makrem \regfont. Implicitně jsou registrovány přepínače \tenrm, \tenit, \tenbf, \tenbi, a \tentt.

Do nových velikostí tedy půjdeme se starými názvy přepínačů `\ten`⟨*něco*⟩ a to slovo `ten` budeme chápat jen jako historický relikt, který nám ovšem napoví, že kontrolní sekvence je fontovým přepínačem.

OPmac si zjistí, zda je definovaný `\regfont`. Pokud ne, upozorní na starou verzi CSplainu na terminálu a potřebná makra si definuje. Je to kopie kódu ze souboru `csfontsm.tex` z balíčku CSplain.

opmac.tex

```
\ifx\regfont\undefined
  \opwarning{csplain version <Nov. 2012> or later is recommended}
  % macros from csplain, file csfontsm.tex:
  \font\tenbi=csbxti10 \def\bi{\tenbi}
  \def\sizespec{}
  \def\resizefont #1{\expandafter
    \font\expandafter#1\expandafter\resizefontskipat\fontname#1 \relax}
  \def\regfont#1{\expandafter\def\expandafter\resizeall\expandafter{%
    \resizeall \resizefont#1}}
  \def\resizeall{}
  \regfont\tenrm \regfont\tenit \regfont\tenbf \regfont\tenbi \regfont\tentt
\fi
```

Implicitně jsou zavedeny CSfonty, takže k nim přidáme AMS fonty z `ams-math.tex`, které vizuálně odpovídají. Později si může uživatel zavést jiné makro (např. `tx-math.tex`) a zavede si třeba i jiné textové fonty. To nezmění vlastnosti maker v OPmac, pokud nové soubory maker správně předefinují makra `\setmathsizes[`⟨*text*⟩`/`⟨*script*⟩`/`⟨*scriptscript*⟩`]`, `\normalmath` a `\boldmath`. Soubor `ams-math.tex` načteme jen tehdy, když není definováno `\normalmath`. Je totiž možné, že uživatel načetl matematické makro ještě před zavoláním `\input␣opmac`.

opmac.tex

```
\ifx\normalmath\undefined \input ams-math \fi  % ams-math.tex is in csplain package
```

Po načtení souboru `ams-math.tex` disponujeme makry `\regtfm` na registraci různých metrik pro různé designované velikosti fontů a `\whichtfm`, které expanduje na svůj parametr nebo na metriku, která je registrována pro velikost `\dgsize`. Registrace metrik CSfontů je rovněž provedena v souboru `ams-math.tex`. Jak bylo řečeno, makro `\resizefont`⟨*fontselector*⟩ CSplainu změní velikost fontu ⟨*fontselector*⟩ podle obsahu makra `\sizespec` (viz soubor `csfontsm.tex`). Toto makro `\resizefont` volá pomocné makro `\resizefontskipat` na odstranění „sizespec“ z názvu metriky. Toto pomocné makro je v OPmac předefinováno s využitím `\whichtfm`, takže když před voláním makra `\resizefont` připravíme správnou `\dgsize`, TEX použije metriku designovanou na požadovanou velikost. Nebyl-li načten `ams-math.tex`, není `\whichtfm` definováno a v takovém případě neprovedeme nic.

opmac.tex

```
\ifx\whichtfm\undefined \else
  \def\resizefontskipat#1 #2\relax{\whichtfm{#1} \sizespec\relax}
\fi
```

Makra `\typosize`, `\fontsizex`, `\textfontsize`, `\setbaselineskip` požadují zápis parametru bez jednotky. Jednotkou je `\ptunit`, která je nastavena na 1pt. Uživatel může jednotku změnit (např. `\ptunit=1mm` při návrhu plakátu). Dále `\fontdim` je registr, který udává aktuální velikost písma.

opmac.tex

```
\newdimen\ptunit    \ptunit=1pt
\newdimen\fontdim   \fontdim=10pt
```

Často budeme potřebovat odstranit jednotku `pt` ve výpisu `\the`⟨*dimen*⟩. Provedeme to pomocí `\expandafter\ignorept\the`⟨*dimen*⟩. Protože `\the` vyrábí znaky pt s kategorií 12, je makro `\ignorept` definováno trikem přes `\lowercase`. Z otazníku vznikne p kategorie 12 a z vykřičníku vznikne t.

opmac.tex

```
{\lccode`\?=`\p \lccode`\!=`\t  \lowercase{\gdef\ignorept#1?!{#1}}}
```

Makra `\typosize` a `\typoscale` změní velikosti a nastavují výchozí font `\tenrm` a výchozí matematiku `\normalmath`. Nehrajeme si na OFS nebo NFSS, které se snaží ctít naposledy nastavený duktus a variantu. Uživatel si variantu písma a tučný duktus pro matematiku musí nastavit až po zavolání makra na změnu velikosti fontu.

opmac.tex

```
\def\typosize[#1/#2]{\fontsizex[#1]\setbaselineskip[#2]\ignorespaces}
\def\typoscale[#1/#2]{\fontscalex[#1]\scalebaselineskip[#2]\ignorespaces}
```

`\regtfm` `\whichtfm`: 7 `\dgsize`: 7–9 `\resizefontskipat`: 7 `\ptunit`: 7–9 `\fontdim`: 7–9
`\ignorept`: 6–9, 43, 52 `\typosize`: 7–9, 32 `\typoscale`: 7, 9, 14, 43–44

Makro \fontsizex [⟨velikost⟩] předpokládá svůj parametr bez jednotky. Písmeno x v názvu značí, že makro není v uživatelské dokumentaci. Uživatel totiž může použít \typosize[⟨velikost⟩/] a třeba ho napadne si nějaké vlastní makro \fontsize definovat. Je-li parametr ⟨velikost⟩ prázdný, makro \fontsizex neudělá nic. Jinak pomocí \textfontsize nastaví velikost textových fontů. Dále zavolá \setmathsizes[\fontsize/.7\fontsize/.5\fontsize], ovšem v parametru musí odstranit jednotky a parametr přichystá pro makro \setmathsizes expandovaný. Příkazem \normalmath nakonec nastaví matematické fonty do nové velikosti.

opmac.tex

```
\def\fontsizex[#1]{\if$#1$\else
  \textfontsize[#1]%
  \tmpdim=0.7\fontdim \edef\tmpa{\expandafter\ignorept\the\tmpdim}%
  \tmpdim=0.5\fontdim \edef\tmpb{\expandafter\ignorept\the\tmpdim}%
  \edef\tmp{\noexpand\setmathsizes[\expandafter\ignorept\the\fontdim/\tmpa/\tmpb]}%
  \tmp \normalmath
  \fi
}
```

Makro \textfontsize [⟨velikost⟩] předpokládá svůj parametr bez jednotky. Připojí jednotku \ptunit, nastaví \dgsize a \sizespec a zavolá \resizeall, což je makro definované v CSplainu, které postupně volá \resizefont na všechny registrované fonty.

opmac.tex

```
\def\textfontsize[#1]{\if$#1$\else
  \fontdim=#1\ptunit
  \let\dgsize=\fontdim
  \edef\sizespec{at\the\fontdim}%
  \resizeall \rm \let\dgsize=\undefined
  \fi
}
```

Makro \setbaselineskip [⟨velikost⟩] předpokládá parametr bez jednotky. Připojí jednotku \ptunit a nastaví \baselineskip bez dodatečné pružnosti. Nastaví další registry, které s \baselineskip souvisejí. Záměrně není nastavena \topskip, \splittopskip, \above/belowdisplayskip. Tyto parametry (globální pro celý dokument) by si měl uživatel nastavit sám.

opmac.tex

```
\def\setbaselineskip[#1]{\if$#1$\else
  \tmpdim=#1\ptunit
  \baselineskip=\tmpdim \relax
  \bigskipamount=\tmpdim plus.33333\tmpdim minus.33333\tmpdim
  \medskipamount=.5\tmpdim plus.16666\tmpdim minus.16666\tmpdim
  \smallskipamount=.25\tmpdim plus.08333\tmpdim minus.08333\tmpdim
  \normalbaselineskip=\tmpdim
  \jot=.25\tmpdim
  \maxdepth=.33333\tmpdim
  \setbox\strutbox=\hbox{\vrule height.709\tmpdim depth.291\tmpdim width0pt}%
  \fi
}
```

Makro \withoutunit \makro⟨dimen⟩ odstraní jednotku z ⟨dimen⟩ a takto upravené číslo vloží do parametru \makro, které očekává údaj bez jednotky v hranaté závorce.

opmac.tex

```
\def\withoutunit#1#2{\expandafter#1\expandafter[\expandafter\ignorept\the#2]}
```

Makra \fontscalex ⟨factor⟩, \textfontscale ⟨factor⟩ a \scalebaselineskip ⟨factor⟩ přepočítají ⟨factor⟩ podle aktuálního \fontdim resp. \baselineskip na absolutní jednotku a zavolají odpovídající makro definované před chvílí.

opmac.tex

```
\def\fontscalex[#1]{\if$#1$\else
  \tmpdim=#1pt \divide\tmpdim by1000
  \tmpdim=\expandafter\ignorept\the\tmpdim \fontdim
  \withoutunit\fontsizex\tmpdim
  \fi
}
\def\textfontscale[#1]{\if$#1$\else
  \tmpdim=#1pt \divide\tmpdim by1000
```

\fontsizex: 7–8 \textfontsize: 7–9 \setbaselineskip: 7–9 \withoutunit: 8–9
\fontscalex: 7–8 \textfontscale: 8–9 \scalebaselineskip: 7, 9

```
  \tmpdim=\expandafter\ignorept\the\tmpdim \fontdim
  \withoutunit\textfontsize\tmpdim
  \fi
}
\def\scalebaselineskip[#1]{\if$#1$\else
  \tmpdim=#1pt \divide\tmpdim by1000
  \tmpdim=\expandafter\ignorept\the\tmpdim \baselineskip
  \withoutunit\setbaselineskip\tmpdim
  \fi
}
```

Makro \thefontsize si alokuje aktuální font do sekvence \thefont a tento nový fontový přepínač podrobí změně velikosti \resizefont. Makro \thefontscale přepočítá parametr na absolutní velikost a zavolá \thefontsize.

opmac.tex

```
\def\thefontsize[#1]{%
  \expandafter\let \expandafter\thefont \the\font
  \def\sizespec{at#1\ptunit}\def\dgsize{#1\ptunit}\resizefont\thefont
  \thefont \let\dgsize=\undefined \ignorespaces
}
\def\thefontscale[#1]{%
  \tmpdim=#1pt \divide\tmpdim by1000
  \tmpdim=\expandafter\ignorept\the\tmpdim \fontdim
  \withoutunit\thefontsize\tmpdim
}
```

PlainTEXový \magstep má na konci \relax, takže nefunguje jako pouze expandující makro. My ale \magstep očekáváme v parametrech příkazů \typoscale a podobných, proto v \magstep je nahrazeno \relax méně drsným \space. To separuje číselný parametr dostatečně.

opmac.tex

```
\def\magstep#1{\ifcase#1 1000\or1200\or1440\or1728\or2074\or2488\fi\space}
```

Makro \em přepíná kontextově do odpovídající varianty a ve spolupráci s makry \additcorr a \afteritcorr přidává italickou korekci. Makro \additcorr si pomocí \lastskip zapamatuje poslední mezeru, pak ji odstraní, vloží italickou korekci a nakonec vrátí tu odstraněnou mezeru. Makro \afteritcorr se probudí k činnosti na konci skupiny a přidá italickou korekci, pokud nenásleduje tečka nebo čárka.

opmac.tex

```
\def\em {\expandafter\ifx \the\font \tenit \additcorr \rm  \else
         \expandafter\ifx \the\font \tenbf \bi\aftergroup\afteritcorr\else
         \expandafter\ifx \the\font \tenbi \additcorr \bf  \else
         \it \aftergroup\afteritcorr\fi\fi\fi}
\def\additcorr{\ifdim\lastskip>0pt \skip0=\lastskip \unskip\/\hskip\skip0 \else\/\fi}
\def\afteritcorr{\def\tmp{\ifx\next..\else\ifx\next,,\else\/%
                          \expandafter\expandafter\expandafter\next\expandafter\fi\fi}%
   \afterassignment\tmp \let\next= }
```

Fontová makra zabezpečíme proti rozkladu v parametru \write.

opmac.tex

```
\addprotect\thefontsize   \addprotect\thefontscale
\addprotect\typosize      \addprotect\typoscale
\addprotect\textfontsize  \addprotect\textfontscale
\addprotect\em
```

3.5 Texty ve více jazycích

Makro \mtext ⟨*značka*⟩ je zkratkou za „multilingual text“. Toto makro si podle značky a aktuálního jazyka (dle registru \language) vyhledá, jaký text má vypsat.

opmac.tex

```
\def\mtext#1{\csname mt:#1:\csname lan:\the\language\endcsname\endcsname}
```

Jednotlivé texty definujeme pomocí \sdef{mt:⟨*značka*⟩:⟨*jazyk*⟩} takto:

opmac.tex
```
\sdef{mt:chap:en}{Chapter}  \sdef{mt:chap:cz}{Kapitola}  \sdef{mt:chap:sk}{Kapitola}
\sdef{mt:t:en}{Table}       \sdef{mt:t:cz}{Tabulka}      \sdef{mt:t:sk}{Tabu\v lka}
\sdef{mt:f:en}{Figure}      \sdef{mt:f:cz}{Obr\'azek}    \sdef{mt:f:sk}{Obr\'azok}
```

Některé texty jsou zapsány pomocí \v notace. Je lepší udělat to takto než vytvořit soubor opmac.tex závislý na kódování. Aby byla tato notace správně interpretována, spustíme \csaccents, což je makro CSplainu. Pokud někdo používá OPmac s jiným formátem, než CSplain, neprovede se nic, protože konstrukce \csname␣csaccents\endcsname se v takovém případě přerodí v \relax. Makro \csaccents spustíme jen tehdy, pokud je už uživatel nespustil před \input␣opmac. To poznáme podle toho, zda je definovaná sekvence \r.

opmac.tex
```
\ifx\r\undefined \csname csaccents\endcsname \fi
```

CSplain od verze Nov.␣2012 připravuje následující makra, která konvertují číslo \language na značku jazyka pro všechny jazyky, které mají nataženy vzory dělení slov. Pro jistotu (pokud je použita starší verze CSplainu) tuto koverzi „naučíme“ i makro OPmac:

opmac.tex
```
\sdef{lan:0}{en}  \sdef{lan:100}{en} \sdef{lan:101}{en}
\sdef{lan:5}{cz}  \sdef{lan:15}{cz}  \sdef{lan:115}{cz}
\sdef{lan:6}{sk}  \sdef{lan:16}{sk}  \sdef{lan:116}{sk}
```

3.6 REF soubor

OPmac používá pro všechny potřeby (obsah, reference, citace, rejstřík, poznámky na okraji) jediný soubor \jobname.ref (tzv. REF soubor). Navíc, pokud není potřeba, vůbec tento soubor nezakládá. Často totiž budeme chtít dělat s OPmac jen jednoduché věci a je únavné pořád na disku kvůli tomu uklízet smetí.

Je potřeba deklarovat souborové deskriptory \reffile a \testin:

opmac.tex
```
\newwrite\reffile
\newread\testin
```

Do souboru zapisujeme makrem \wref \⟨*sekvence*⟩{⟨*data*⟩}, které vloží do \reffile řádek obsahující \⟨*sekvence*⟩⟨*data*⟩. Implicitně ale není \reffile založeno, takže implicitní hodnota tohoto makra je \wrefrelax, tedy nedělej nic.

opmac.tex
```
\def\wrefrelax#1#2{}
\let\wref=\wrefrelax
```

Makro \inputref spustíme na konci čtení souboru opmac.tex, tedy v situaci, kdy už budeme mít definovány všechny kontrolní sekvence, které se v REF souboru mohou vyskytnout. Nyní si toto makro jen připravíme. Makro ověří existenci souboru \jobname.ref a pokud existuje, provede \input␣\jobname.ref. V takovém případě po načtení REF souboru jej otevře k zápisu a připraví \wref do stavu, kdy toto makro bude ukládat data do souboru.

opmac.tex
```
\def\inputref{
  \openin\testin=\jobname.ref
  \ifeof\testin \else
    \closein\testin
    \input \jobname.ref
    \fnotenum=0 \mnotenum=0
    \immediate\openout\reffile=\jobname.ref
    \def\wref##1##2{\write\reffile{\string##1##2}}
    \immediate\write\reffile {\percent\percent\space OPmac - REF file}
  \fi
```

Makro \openref kdekoli v dokumentu si vynutí založení souboru \jobname.ref. Toto makro neprovede nic, je-li REF soubor už založen. To pozná podle toho, že makro \wref nemá význam \wrefrelax. Jestliže soubor ještě není založen, makro jej založí, předefinuje \wref a vloží první řádek do souboru. Tím je zaručeno, že při příštím TEXování dokumentu je soubor neprázdný, takže jej OPmac rovnou přečte a znovu založí na začátku své činnosti. Nakonec se \openref zasebevraždí, aby se nemuselo při opakovaném volání obtěžovat vykonávat nějakou práci. Práce už je totiž hotova.

\reffile: 10–11 \testin: 10, 48 \wref: 10–11, 14, 18, 29, 31, 43–44, 47–48 \wrefrelax: 10–11
\inputref: 10, 52 \openref: 10–12, 18, 29, 35, 43–44, 46–48

opmac.tex
```
\def\openref{%
  \ifx\wref\wrefrelax
     \immediate\openout\reffile=\jobname.ref
     \gdef\wref##1##2{\write\reffile{\string##1##2}}%
     \immediate\write\reffile
       {\percent\percent\space OPmac - REF file (\string\openref)}%
  \fi
  \gdef\openref{}%
}
```

Pro zápisy do REF souboru používáme tuto konvenci: první kontrolní sekvence na řádku je vždy tvaru `\X`⟨*název*⟩, takže máme přehled, která kontrolní sekvence pochází z REF souboru.

3.7 Lejblíky a odkazy

K vytvoření zpětného odkazu provedeme tři kroky (v tomto pořadí):

- V místě `\label[`⟨*lejblík*⟩`]` si zapamatujeme ⟨*lejblík*⟩.
- V době vygenerování čísla (sekce, kapitoly, caption, atd.) propojíme ⟨*lejblík*⟩ s tímto číslem. Provedeme to pomocí `\sxdef{lab:`⟨*lejblík*⟩`}{`⟨*číslo*⟩`}`.
- V místě `\ref[`⟨*lejblík*⟩`]` vytiskneme `\csname␣lab:`⟨*lejblík*⟩`\endcsname`, tedy ⟨*číslo*⟩.

To je základní idea pro zpětné odkazy. V takovém případě nepotřebujeme REF soubor. Pokud ale chceme dopředné odkazy, je potřeba použít REF soubor zhruba takto:

- V době vygenerování čísla (sekce atd.) navíc uložíme informaci `\Xlabel{`⟨*lejblík*⟩`}{`⟨*číslo*⟩`}` do REF souboru.
- V době čtení REF souboru (tedy na začátku dokumentu) provede `\Xlabel` přiřazení pomocí `\sdef{lab:`⟨*lejblík*⟩`}{`⟨*číslo*⟩`}`.
- Nyní může přijít `\ref[`⟨*lejblík*⟩`]` se svým `\csname␣lab:`⟨*lejblík*⟩`\endcsname` kdekoli v dokumentu.

Přejdeme od idejí k implementaci. Makro `\label [`⟨*lejblík*⟩`]` si pouze zapamatuje ⟨*lejblík*⟩ do makra `\lastlabel`, aby s touto hodnotou mohlo později pracovat makro, které automaticky generuje nějaké číslo.

opmac.tex
```
\def\label[#1]{\xdef\lastlabel{#1}\ignorespaces}
```

Makro, které automaticky generuje nějaké číslo, má za úkol zavolat `\wlabel` ⟨*číslo*⟩. Toto makro propojí `\lastlabel` a ⟨*číslo*⟩ tak, že definuje sekvenci `\lab:\lastlabel` jako makro s hodnotou ⟨*číslo*⟩. Kromě toho zapíše expandované `\lastlabel` i ⟨*číslo*⟩ do REF souboru (jen, je-li otevřen, zpětné reference totiž fungují i bez souboru). Nakonec vrátí makru `\lastlabel` jeho původní nedefinovanou hodnotu, tj. lejblík už byl použit. Další makro automaticky generující číslo zavolá `\wlabel`, který nyní neprovede nic (pokud tedy uživatel nenapsal další `\label[`⟨*lejblík*⟩`]`).

opmac.tex
```
\def\wlabel#1{%
  \ifx\lastlabel\undefined \else
     \dest[ref:\lastlabel]%
     \edef\tmp{\wref\Xlabel{{\lastlabel}{#1}}}\tmp
     \isdefined{lab:\lastlabel}\iftrue \else
        \sxdef{lab:\lastlabel}{#1}%
     \fi
     \global\let\lastlabel=\undefined
  \fi
}
```

\label: 11, 33 \lastlabel: 11 \wlabel: 11, 15–16

Makro \ref [⟨lejblík⟩] zkontroluje definovanost \lab:⟨lejblík⟩. Je-li to pravda, vytiskne \lab:⟨lejblík⟩ (krz reflink, aby to bylo případně klikací). Jinak vytiskne dva otazníky a předpokládá, že v tomto případě jde o dopřednou referenci. Vynutí si tedy otevření REF souboru zavoláním \openref.

opmac.tex
```
\def\ref[#1]{\isdefined{lab:#1}%
  \iftrue \reflink[#1]{\csname lab:#1\endcsname}%
  \else ??\opwarning{label [#1] unknown. Try to TeX me again}\openref
  \fi
}
```

Makro \pgref [⟨lejblík⟩] dělá podobnou práci, jako \ref, jen s makrem \pgref:⟨lejblík⟩. Toto makro je definováno až při znovunačtení REF souboru makrem \Xlabel, protože ke správnému určení čísla stránky potřebujeme asynchronní \write.

opmac.tex
```
\def\pgref[#1]{\isdefined{pgref:#1}%
  \iftrue \pglink{\csname pgref:#1\endcsname}%
  \else ??\opwarning{pg-label [#1] unknown. Try to TeX me again}\openref
  \fi
}
\def\Xlabel#1#2{\sxdef{lab:#1}{#2}\sxdef{pgref:#1}{\the\lastpage}}
```

3.8 Kapitoly, sekce, podsekce

Od verze OPmac Jul. 2013 jsou zcela přepracována pomocná makra pro návrh typografie kapitol, sekcí a podsekcí. Nyní má autor typografického návrhu lépe pod vlastní kontrolou, co se vkládá do vertikálního seznamu, což je pro programování možných stránkových zlomů a nezlomů důležité.

Autor typografie dokumentu by měl definovat pro tisk kapitoly, sekce a podsekce makra \printchap, \printsec a \printsecc. Makra mají jeden parametr #1, který obsahuje text titulku. Typická struktura každého takového makra je:

```
\par
⟨penalta obvykle záporná, neboli bonus, pro zlomení stránky před nadpisem⟩
⟨mezera před nadpisem⟩
{⟨nastavení fontu⟩ \noindent \dotocnum{⟨značka⟩}#1\nbpar}
⟨případné vložení značky (insertmark) pro plovoucí záhlaví⟩
\nobreak ⟨mezera pod napisem⟩
```

Pro realizaci maker \printchap, \printsec a \printsecc může autor návrhu využít následujících interních maker OPmac:

- \dotocnum{⟨značka⟩} – umístí cíle odkazů, zařídí obsah, vytiskne ⟨značku⟩
- \thetocnum – ⟨značka⟩, např. 3.2.4 pro secc, 3.2 pro sec a 3 pro chap
- \insertmark{⟨text⟩} – vloží \mark s expandovaným \thetocnum a neexpandovaným ⟨textem⟩
- \nbpar – jako \par, ale mezi řádky je nezlomitelná mezera
- \remskip⟨velikost⟩ – mezera (pod nadpisem) odstranitelná následujícím \norempenalty
- \norempenalty⟨číslo⟩ – vloží penaltu ⟨číslo⟩ jen pokud nepředchází \remskip

Aby fungoval obsah a cíle odkazů, je *nutné* použít \dotocnum. Parametr makra \dotocnum nemusí obsahovat jen \thetocnum, ale také tečky a mezery kolem ⟨značky⟩. Předchází-li \nonum, makro \dotocnum nevytiskne celý svůj druhý prametr, tedy včetně případného „okolí“ značky. Návrh tisku sekce může vypadat takto:

```
\def\printsec#1{\par
  \norempenalty-500
  \vskip 12pt plus 2pt
  {\secfont \noindent \dotocnum{\thetocnum\quad}#1\nbpar}%
  \placemark{#1}%
  \nobreak \remskip 6pt plus 1pt
}
```

\ref: 11–12 \pgref: 12 \Xlabel: 11–12

V tomto návrhu bude nad nadpisem penalta −500 (bonus za zlomení nad nadpisem), dále je `12pt` mezera, pak je titulek `#1` vytištěný fontem `\secfont`. Před tímto titulkem je číselná značka oddělená od titulku mezerou `\quad`. Titulek může být na více řádcích. Protože je ukončen `\nbpar`, nebude povolen mezi jednotlivými řádky titulku řádkový zlom.

Vysvětlíme si nyní na příkladu činnost a smysl `\remskip` a `\norempenalty`. Předpokládejme pro ilustraci definici `\printsec`, jako je uvedená výše. Pokud je dále třeba definice podsedkce `\printsecc` zahájena příkazy

```
\par \norempenalty-200 \vskip 8pt plus2pt
```

pak se mohou dít tyto věci:

- Následuje-li podsekce těsně za sekcí, pak se vymaže spodní mezera od sekce `6pt␣plus1pt` a místo ní se vloží mezera `8pt␣plus2pt`. Mezery se tedy nesčítají. Navíc v tomto případě se nevloží penalta −200, takže mezi sekcí a podeskcí nedojde nikdy ke stránkovému zlomu.
- Následuje-li za sekcí obyčejný text, pak je pod sekcí a nad textem mezera `6pt␣plus␣1pt`, která je nezlomitelná.
- Předchází-li před podseskcí obyčejný text, pak se vloží před nadpisem podsekce `\penalty-200` následovaná `\vskip␣8pt␣plus2pt`. Tato mezera je ochotně zlomitelná (bonus −200), takže se může nadpis podsekce objevit na následující straně.

Je možné mezeru pod nadpisem složit ze dvou druhů:

```
\def\printsec{%
  ...
  \nobreak \vskip 2pt \remskip 4pt plus1pt}
```

V tomto příkladě se odstraní při následující podsekci z celkové mezery `6pt␣plus1pt` jen její část `4pt␣plus1pt`.

Defaultní hodnoty maker `\printchap`, `\printsec` a `\printsecc` vypadají v OPmac takto:

opmac.tex

```
294: \def\printchap#1{\vfil\break
295:   {\chapfont \noindent \mtext{chap} \dotocnum{\thetocnum}\par
296:    \nobreak\smallskip\noindent #1\nbpar}\mark{}%
297:   \nobreak \remskip\bigskipamount \firstnoindent
298: }
299: \def\printsec#1{\par \norempenalty-400 \bigskip
300:   {\secfont \noindent \dotocnum{\thetocnum\quad}#1\nbpar}\insertmark{#1}%
301:   \nobreak \remskip\medskipamount \firstnoindent
302: }
303: \def\printsecc#1{\par \norempenalty-200 \medskip
304:   {\seccfont \noindent \dotocnum{\thetocnum\quad}#1\nbpar}%
305:   \nobreak \remskip\medskipamount \firstnoindent
306: }
```

Příkazem `\firstnoindent` dávají tato makra najevo, že následující odstavec nebude mít odstavcovou zarážku.

Makro pro titul `\tit` počítá s tím, že bude titul na více řádcích. Sází ho tedy jako odstavec s pružnými `\leftskip` a `\rightskip`. Příkaz `\unskip` těsně za parametrem `#1` odstraní mezeru z konce řádku, která tam obvykle je. Teprve poté je nadpis řádně centrován.

opmac.tex

```
307: \def\tit#1\par{\vglue4em
308:   {\leftskip=0pt plus1fill \rightskip=\leftskip
309:    \titfont \noindent #1\unskip\par}%
310:    \nobreak\bigskip
311: }
```

Fonty pro titul, kapitoly a sekce `\titfont`, `\chapfont`, `\secfont` a `\seccfont` jsou definovány jako odpovídající zvětšení a nastavení tučného duktu. Ten je nastaven pomocí `\bfshape` jako `\bf` a navíc je ztotožněn `\tenit` s `\tenbi`, takže když nyní uživatel napíše `\it`, dostane tučnou kurzívu.

\printchap: 12–15 \printsec: 12–15 \printsecc: 12–15 \tit: 13 \titfont: 13–14
\chapfont: 13–14 \secfont: 13–14 \seccfont: 13–14 \bfshape: 14

opmac.tex
```
\def\titfont{\typoscale[\magstep4/\magstep4]\bf}
\def\chapfont{\typoscale[\magstep3/\magstep3]\bfshape}
\def\secfont{\typoscale[\magstep2/\magstep2]\bfshape}
\def\seccfont{\typoscale[\magstep1/\magstep2]\bfshape}
\def\bfshape{\let\tenit=\tenbi \boldmath \bf}
```

V další části této sekce je implemantace maker \chap, \sec a \secc. Pro číslování kapitol, sekcí a podsekcí potřebujeme čítače a další registry:

opmac.tex
```
\newcount\chapnum \newcount\secnum \newcount\seccnum \newcount\nonumnum
```

Makro \notoc je možno použít jako prefix před \chap, \sec, \secc s tím, že se kapitola, sekce, podsekce nedostane do obsahu. Makro \nonum je možnot použít jako prefix před stejnými makry s tím, že kapitola, sekce, podsekce nebude mít číslo. I nečíslovaní kapitola se může dostat do obsahu. Je-li obsah klikací, pořebuje mít svoje referenšní číslo pro vytvoření linku. K tomu slouží registr \nonumnum.

opmac.tex
```
\newif\ifnotoc \notocfalse  \def\notoc{\global\notoctrue}
\newif\ifnonum \nonumfalse  \def\nonum{\global\nonumtrue}
```

Makra \chap, \sec a \secc nastaví odpovídající čítače, dále vytvoří číslování pro tisk (sestávající z více čísel) v makrech \thechapnum, \thesecnum a \theseccnum. Aktuální čítač má vždy název \thetocnum. S touto hodnotou (nazávisle na tom, zde jde o kapitlu, sekci nebo podsekci, bude pracovat \dotocnum. Dále makra připraví obsah makra \dotocnumafter, což je proměnlivá část makra \dotocnum. Konečně uvedená makra \chap, \sec a \secc zavolají odpovídající makro \printchap, \printsec a \printsecc.

opmac.tex
```
\def\chap#1\par{\ifnonum\else \global\advance\chapnum by1 \fi
  \chaphook {\globaldefs=1 \secnum=0 \seccnum=0 \tnum=0 \fnum=0 \dnum=0}\relax
  \edef\thechapnum{\the\chapnum}\let\thetocnum=\thechapnum
  \def\dotocnumafter{\wcontents\Xchap{#1}}%
  \printchap{#1\unskip}\resetnonumnotoc
}
\def\sec#1\par{\ifnonum\else \global\advance\secnum by1 \fi
  \sechook {\globaldefs=1 \seccnum=0 \tnum=0 \fnum=0 \dnum=0}\relax
  \edef\thesecnum{\othe\chapnum.\the\secnum}\let\thetocnum=\thesecnum
  \def\dotocnumafter{\wcontents\Xsec{#1}}%
  \printsec{#1\unskip}\resetnonumnotoc
}
\def\secc#1\par{\ifnonum\else \global\advance\seccnum by1 \fi
  \secchook {}\relax
  \edef\theseccnum{\othe\chapnum.\the\secnum.\the\seccnum}\let\thetocnum=\theseccnum
  \def\dotocnumafter{\wcontents\Xsecc{#1}}%
  \printsecc{#1\unskip}\resetnonumnotoc
}
```

V proměnlivé části makra \dotocnum, tedy v \dotocnumafter, se řeší uložení informací o kapitole, sekci nebo podsekci do obsahu. K tomu je využito makro \wcontents, které provede

```
\write\reffile{\string#1{⟨expandovaný thetocnum⟩}{#2}{\the\pageno}}
```

opmac.tex
```
\def\wcontents#1#2{% #1: sequence to REF, #2: titletext
  \ifnotoc\else
      \expandafter\wref\expandafter#1\expandafter
        {\expandafter{\thetocnum}{#2}{\the\pageno}}%
  \fi
}
```

Makro \dotocnum {⟨text⟩} umístí cíl odkazu do místa, které je od účaří vzdáleno o \destheight. Toto místo se kryje s horní hranou okna prohlížeče po kliknutí na odkaz. Dále toto makro vygeneruje data pro obsah pomocí předpřipraveného \dotocnumafter. Pokud předchází \notoc, makro nezapíše nic do obsahu. Pokud předchází \nonum, makro nevytiskne svůj parametr, takže je titulek bez čísla.

Dále v takovém případě je pro hyperlinkové odkazy číslo \nonumnum uvozeno vykřičníkem, což navazuje v obsahu na makro \toclinkA.

opmac.tex
```
\def\dotocnum#1{%
  \leavevmode
     {\ifnonum \global\advance\nonumnum by1 \edef\thetocnum{!\the\nonumnum}\fi
      \wlabel\thetocnum \dest[toc:\thetocnum]%
      \dotocnumafter}\ifnonum\else#1\fi
  \global\let\dotocnumafter=\relax
}
```

V makrech \chap, \sec, \secc je po zavolání \printchap, \printsec nebo \printsecc voláno \resetnonunotoc, které vrátí hodnotu přepínačů \ifnonum a \ifnotoc na imlicitní hodnoty. Tím je zaručeno, že makra \nonum a \notoc ovlivní jen následující kapitolu, sekci nebo podsekci a fungují jako prefixy. Makro navíc kvůli přechodu na verzi OPmac Jul. 2013 kontroluje, zda makra \printchap, \printsec a \printsecc skutečně použila makro \dotocnum. Pokud ne, náležitě na to upozorní.

opmac.tex
```
\def\resetnonumnotoc{\global\notocfalse \global\nonumfalse
  \ifx\dotocnumafter\relax \else
    \opwarning{\noexpand\dotocnum unused in printchap/printsec/printsecc}\fi
}
```

Text titulu sekce a její číslo jsou vloženy do \mark, takže tyto údaje můžete použít v plovoucím záhlaví. Tato vlastnost není dokumentována v uživatelské části, protože je poněkud techničtějšího charakteru.

Makro \insertmark {⟨*text*⟩} vloží do \mark data ve formátu {⟨*thetocnum*⟩}␣{⟨*text*⟩}, takže je možno je použít přímo expanzí např. \firstmark, nebo je oddělit a zpracovat zvlášť. Parametru ⟨*text*⟩ je zabráněna expanze pomocí protažení tohoto parametru přes \toks, viz TBN str. 54 dole a strany 55–57. Příkaz \mark se totiž snaží o expanzi.

opmac.tex
```
\def\insertmark#1{\toks0={#1}\mark{{\thetocnum} {\the\toks0}}}
```

Příklad použití plovoucího záhlaví v \headline:

```
\headline{\expandafter\domark\firstmark\hss}
\def\domark#1#2{\llap{\it\headsize #1. }\rm\headsize #2}
\def\headsize{\thefontsize[10]}
```

Makro \headsize v této ukázce zaručí, že bude mít záhlaví vždy požadovanou velikost. Bez toho ta záruka není, pokud tedy uživatel v sazbě dokumentu střídá velikosti písma. Output rutina totiž může přijít náhle, třeba v okamžiku, kdy je zapnutá jiná velikost písma.

Pokud chcete kombinovat na levých a pravých stránkách plovoucí záhlaví z kapitol a sekcí, inspirujte se v TBN na stranách 259 a 260.

Makro \remskip ⟨*velikost*⟩ je implimentováno jako \vskip⟨*velikost*⟩ následované smluvenou nezlomitelnou penaltou 11333. Makro \norempenalty pak podle této hodnoty poslední penalty v seznamu větví svou činnost. V registru \remskipamount je uložena naposledy vložená mezera z \remskip.

opmac.tex
```
\newskip\remskipamount
\def\remskip{\afterassignment\remskipA \global\remskipamount}
\def\remskipA{\vskip\remskipamount \penalty11333 }
\def\norempenalty{\ifnum\lastpenalty=11333
   \vskip-\remskipamount \tmpnum=\else \removelastskip \penalty \fi}
```

Makro \othe pracuje stejně jako primitiv \the s tím rozdílem, že nezobrazí nic (a sejme následující tečku), pokud je hodnota registru nulová. Tímto způsobem lze tisnout dokument jen se sekcemi bez kapitol. Číslo kapitoly se pak nezobrazuje jako 0., protože se nezobrazuje vůbec.

opmac.tex
```
\def\othe#1.{\ifnum#1>0 \the#1.\fi}
\def\thechapnum{} \def\thesecnum{} \def\theseccnum{}
```

Makro \afternoindent potlačí odstavcovou zarážku pomocí přechodného naplnění \everypar kódem, který odstraní box vzniklý z \indent a vyprázdní pomocí \wipeepar registr \everypar. Makro

\resetnonunotoc \insertmark: 12–13, 15 \remskip: 12–13, 15 \norempenalty: 12–13, 15
\remskipamount: 15 \othe: 14–15 \afternoindent: 16, 37 \wipeepar: 16, 27, 37, 39

\firstnoindent je ztotožněno s \afternoindent, ale uživatel může psát \let\firstnoindent=\relax. Pak bude \afternoindent pracovat jen za verbatim výpisy.

opmac.tex
```
\def\afternoindent{\global\everypar={\wipeepar\setbox0=\lastbox}}
\def\wipeepar{\global\everypar={}}
\let\firstnoindent=\afternoindent
```

Nechceme, aby se nám odstavce tvořené z titulů kapitol a sekcí rozdělily do více stran. Proto místo příkazu \par je použito \nbpar. Konečně uživatel může odřádkovat například v titulu pomocí \nl. Tomuto makru později v \output rutině změníme význam na mezeru.

opmac.tex
```
\def\nbpar{{\interlinepenalty=10000\par}}
\def\nl{\hfil\break}
```

3.9 Popisky, rovnice

Nejprve deklarujeme potřebné čítače:

opmac.tex
```
\newcount\tnum  \newcount\fnum  \newcount\dnum
```

Výchozí hodnoty maker, které se vypisují v místě generovaného čísla jsou:

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

Makro \caption /⟨*typ*⟩␣ zvedne čítač \⟨*typ*⟩num o jedničku, dále nastaví pružné mezery s odsazením \iindent a s centrováním posledního řádku (viz TBN str. 234), propojí pomocí \wlabel číslo s případným lejblíkem a vytiskne zahájení popisku makrem \printcaption. Makro \caption tím končí svou činnost a dále je zpracován odstavec s nastavenými \leftskip, \rightskip. Sekvence \par je předefinována tak, že první výskyt \par (alias prázdného řádku) ukončí skupinu a tím se všechna nastavení vrátí do původního stavu.

opmac.tex
```
\def\caption/#1 {\isdefined{#1num}%
   \iftrue \global\advance \csname #1num\endcsname by1
   \else   \opwarning{Unknown caption /#1}%
   \fi
   \bgroup
      \leftskip=\iindent plus1fil
      \rightskip=\iindent plus-1fil
      \parfillskip=0pt plus2fil
      \def\par{\endgraf\egroup}%
      \captionhook{#1}\noindent
      {\destheight=2.1em \wlabel{\csname the#1num\endcsname}}%
      \printcaption{\mtext{#1}}{\csname the#1num\endcsname}%
}
```

Makro \printcaption {⟨*slovo*⟩}{⟨*číslo*⟩} vytiskne zahájení popisku tabulky a obrázku. Za číslem implicitně není žádná interpunkce, jen \enspace. Je možné předefinovat \printcaption s interpunkcí třeba takto: \def\printcaption#1#2{{\bf#1␣#2:}\space}

opmac.tex
```
\def\printcaption#1#2{{\bf#1 #2}\enspace}
```

Makro \eqmark zvedne \dnum o jedničku. V display módu pak použije primitiv \eqno, za kterým následuje \thednum. V interním módu (v boxu) vytiskne jen \thetnum. V obou případech propojí případný lejblík s číslem pomocí makra \wlabel.

opmac.tex
```
\def\eqmark{\global\advance\dnum by1
  \ifinner\else\eqno \fi
  {\destheight=2.1em \wlabel\thednum}\thednum
}
```

\firstnoindent: 13, 16 \nbpar: 12–13, 16 \nl: 16, 50 \tnum: 14, 16 \fnum: 14, 16
\dnum: 14, 16 \caption: 6, 16 \printcaption: 16 \eqmark: 16

3.10 Odrážky

Odsazení každé další vnořené úrovně odrážek bude o \iindent větší. Jeho hodnota je nastavena na \parindent v době čtení opmac.tex. Jestliže uživatel později změní \parindent, měl by odpovídajícím způsobem změnit \iindent. Kromě toho deklarujeme čítač pro odrážky \itemnum.

opmac.tex
```
\newcount\itemnum  \itemnum=0
```

Makro \begitems vloží \iiskip, zahájí novou skupinu, pronuluje \itemnum, zvětší odsazení o \iindent a pomocí \adef definuje hvězdičku jako aktivní makro, které provede \startitem. Makro \enditems ukončí skupinu a vloží \iiskip.

opmac.tex
```
\def\begitems{\par\iiskip\bgroup
  \itemnum=0 \adef*{\startitem}
  \advance\leftskip by\iindent
  \let\printitem=\normalitem
}
\def\enditems{\par\egroup\iiskip}
```

Makro \startitem ukončí případný předchozí odstavec, posune čítač, zahájí první odstavec jako \noindent a vyšoupne doleva text definovaný v \printitem, který je implicitně nastaven makrem \begitems na \normalitem.

opmac.tex
```
\def\startitem{\par \advance\itemnum by1
   \noindent\llap{\printitem}\ignorespaces}
\def\normalitem{$\bullet$\enspace}
```

Makro \style ⟨*znak*⟩ přečte ⟨*znak*⟩ a rozvine jen na makro \item:⟨*znak*⟩. Tato jednotlivá makra jsou definována pomocí \sdef. Není-li makro \item:⟨*znak*⟩ definováno, použije se \normalitem.

opmac.tex
```
\def\style#1{\expandafter\let\expandafter\printitem\csname item:#1\endcsname
  \ifx\printitem\relax \let\printitem=\normalitem \fi
}
\sdef{item:o}{\raise.4ex\hbox{$\scriptscriptstyle\bullet$} }
\sdef{item:-}{- }
\sdef{item:n}{\the\itemnum. }
\sdef{item:N}{\the\itemnum) }
\sdef{item:i}{(\romannumeral\itemnum) }
\sdef{item:I}{\uppercase\expandafter{\romannumeral\itemnum}\kern.5em}
\sdef{item:a}{\athe\itemnum) }
\sdef{item:A}{\uppercase\expandafter{\athe\itemnum}) }
\sdef{item:x}{\raise.3ex\fullrectangle{.6ex} }
\sdef{item:X}{\raise.2ex\fullrectangle{1ex}\kern.5em}
```

Čtvereček kreslíme jako \vrule odpovídajících rozměrů makrem \fullrectangle {⟨*dimen*⟩}.

opmac.tex
```
\def\fullrectangle#1{\hbox{\vrule height#1 width#1}}
```

Pro převod mezi numerickou hodnotou čítače a příslušným písmenem a, b, c atd. je vytvořeno makro \athe ⟨*number*⟩.

opmac.tex
```
\def\athe#1{\ifcase#1?\or a\or b\or c\or d\or e\or f\or g\or h\or i\or j\or k\or l\or
   m\or n\or o\or p\or q\or r\or s\or t\or u\or v\or w\or x\or y\or z\else ?\fi
}
```

3.11 Tvorba obsahu

Do \toclist budeme ukládat data pro obsah. Pomocí \ifischap se budeme ptát, zda v dokumentu jsou kapitoly.

opmac.tex
```
\def\toclist{} \newif\ifischap \ischapfalse
```

\itemnum: 17 \begitems: 6, 17 \enditems: 6, 17 \startitem: 17 \printitem: 17
\normalitem: 17 \style: 17 \fullrectangle: 17 \athe: 17 \toclist: 17–18, 35
\ifischap: 17–18

V době čtení REF souboru vložíme veškerá data obsahu do makra \toclist tak, že v tomto bufferu budeme mít za sebou sekvence \tocline následované pěti parametry. Makra \Xchap, \Xsec a \Xsecc mají parametry {⟨číslo⟩}{⟨text⟩}{⟨strana⟩} a jsou definovány následovně:

opmac.tex
```
\def\Xchap#1#2#3{\ischaptrue\addto\toclist{\tocline{0}{\bf}{#1}{#2}{#3}}}
\def\Xsec#1#2#3{\addto\toclist{\tocline{1}{\rm}{#1}{#2}{#3}}}
\def\Xsecc#1#2#3{\addto\toclist{\tocline{2}{\rm}{#1}{#2}{#3}}}
```

Makro \tocline {⟨odsazení⟩}{⟨font⟩}{⟨číslo⟩}{⟨text⟩}{⟨strana⟩} vytvoří řádek obsahu. Řádek tiskneme jako odstavec, protože ⟨text⟩ může být třeba delší. Registr \leftskip nastavíme jako součin ⟨odsazení⟩ krát \iindent. Pokud se v dokumentu vyskytují kapitoly, odsadíme ještě o další \iindent. Registr \rightskip nastavíme na 2\iindent, aby delší ⟨text⟩ se zalomil dřív než v místě, kde jsou stránkové číslice. Konec odstavce se stránkovou číslicí pak vytáhneme mimo tento rozsah pomocí \hskip-2\iindent. Odstavec pruží v \tocdotfill, protože tento výplněk má větší pružnost než \parfillskip. Registr \parfillskip má 1fil, zatímco \tocdotfill má pružnost 1fill. Makro \tocdotfill je implementováno pomocí \leaders jako opakované tečky, které budou lícovat pod sebou.

opmac.tex
```
\def\tocline#1#2#3#4#5{{\leftskip=#1\iindent \rightskip=2\iindent
   \ifischap\advance\leftskip by\iindent\fi
   \ifnum#1>1 \advance\leftskip by\iindent\fi
   \toclinehook \noindent\llap{#2\toclink{#3}\enspace}%
         {#2#4\unskip}\nobreak\tocdotfill\pglink{#5}\nobreak\hskip-2\iindent\null\par}}
\def\tocdotfill{\leaders\hbox to.8em{\hss.\hss}\hskip 1em plus1fill\relax}
```

Makro \maketoc jednoduše spustí \toclist, Pokud je \toclist prázdný, upozorní o tom adekvátním způsobem na terminál.

opmac.tex
```
\def\maketoc{\par \ifx\toclist\empty
      \opwarning{\noexpand\maketoc -- data unavailable, TeX me again}\openref
   \else \toclist \fi}
```

Makro \toclinkA je citlivé na vykřičník jako první znak argumentu. Pokud tam je, vytiskne jen mezeru velikosti 0.8em, jinak vytiskne argument. Vykřičník do argumentu dáme v případě kapitol a sekcí prefixovaných jako \nonum. V obsahu pak nechceme mít žádné číslo, jen příslušnou mezeru.

opmac.tex
```
\def\toclinkA#1{\def\tmp##1!##2\end{\if^##1^\kern.8em \else##1\fi}\tmp#1!\end}
```

3.12 Sestavení rejstříku

Slovo do rejstříku vložíme pomocí \iindex {⟨heslo⟩}. Protože výskyt slova na stránce není v době zpracování znám, je nutné použít REF soubor s asynchronním \write.

opmac.tex
```
\def\iindex#1{\openref\wref\Xindex{{#1}{\the\pageno}}}
```

Nyní naprogramujeme čtení parametru makra \ii ⟨slovo⟩,⟨slovo⟩,...⟨slovo⟩␣. Vzhledem k tomu, že za přítomnosti zkratky @ budeme potřebovat projít seznam slov oddělených čárkou v parametru ještě jednou, zapamatujeme si tento seznam do \tmp.

opmac.tex
```
\def\ii #1 {\leavevmode\def\tmp{#1}\iiA #1,,}
```

Makro \iiA sejme vždy jedno slovo ze seznamu. Podle prázdného parametru poznáme, že jsme u konce a neděláme nic. Při výskytu ⟨slovo⟩=@ (poznáme to podle shodnosti parametru s \iiatsign), spustíme \iiB, jinak vložíme údaj o slově do rejstříku pomocí \iindex. Nakonec makro \iiA volá rekurzivně samo sebe.

opmac.tex
```
\def\iiA #1,{\if$#1$\else\def\tmpa{#1}%
   \ifx\tmpa\iiatsign \expandafter\iiB\tmp,,%
   \else\iindex{#1}\fi
   \expandafter\iiA\fi}
\def\iiatsign{@}
```

\Xchap: 14, 18 \Xsec: 14, 18 \Xsecc: 14, 18 \tocline: 6, 18, 35 \tocdotfill: 18
\maketoc: 18 \toclinkA: 15, 18, 33 \iindex: 18–19 \ii: 18–19 \iiA: 18–19
\iiatsign: 18

Makro \iiB rovněž sejme vždy jedno slovo ze seznamu a na konci volá rekurzivně samo sebe. Toto makro ovšem za použití makra \iiC prohodí pořadí prvního podslova před prvním lomítkem se zbytkem. Není-li ve slově lomítko, pozná to makro \iiC podle toho, že parametr #2 je prázdný. V takovém případě neprovede nic, neboť slovo je už zaneseno do rejstříku v makru \iiA.

opmac.tex

```
\def\iiB #1,{\if$#1$\else
   \iiC#1/\relax
   \expandafter\iiB\fi
}
\def\iiC #1/#2\relax{\if$#2$\else\iindex{#2#1}\fi}
```

Makro \iid ⟨*slovo*⟩␣ pošle slovo do rejstříku a současně je zopakuje do sazby. Pomocí \futurelet a \iiD zjistí, zda následuje tečka nebo čárka. Pokud ne, vloží mezeru.

opmac.tex

```
\def\iid #1 {\leavevmode\iindex{#1}#1\futurelet\tmp\iiD}
\def\iiD{\ifx\tmp,\else\ifx\tmp.\else\space\fi\fi}
```

Při čtení REF souboru se vykonávají makra \Xindex {⟨*heslo*⟩}{⟨*strana*⟩}, která postupně vytvářejí makra tvaru \,⟨*heslo*⟩, ve kterých je shromažďován seznam stránek pro dané ⟨*heslo*⟩. Kromě toho každé makro \,⟨*heslo*⟩ je vloženo do seznamu \iilist. Na konci čtení REF souboru tedy máme v \iilist seznam všech hesel jako řídicí sekvence (to zabere nejmíň místa v TEXu). Každé \,⟨*heslo*⟩ na konci čtení REF souboru obsahuje dva údaje ve svorkách, první údaj obsahuje pomocná data a druhý obsahuje seznam stránek. Tedy \,⟨*heslo*⟩ je makro s obsahem {⟨*pomocná-data*⟩}{⟨*seznam-stránek*⟩}.

Seznam stránek není jen tupý seznam všech stránek, na kterých se objevil záznam \ii pro dané slovo. Některé stránky se totiž mohou opakovat a my je chceme mít jen jednou. Pokud stránky jsou souvisle za sebou: 13, 14, 15, 16, chceme navíc takový seznam nahradit zápisem 13–16. Makro \Xindex{⟨*heslo*⟩}{⟨*strana*⟩} je z tohoto důvodu poněkud sofistikovanější.

opmac.tex

```
\def\Xindex#1#2{\bgroup \def~{ }%
   \isdefined{,#1}\iftrue
     \expandafter\firstdata \csname,#1\endcsname \XindexA
     \ifnum#2=\tmpa % \ii on the same page
     \else
        \tmpnum=#2 \advance\tmpnum by-1
        \expandafter\seconddata \csname,#1\endcsname \XindexB
        \if\tmpb+% state: the pagelist ends by a pagenumber
           \ifnum\tmpnum=\tmpa  % the consecutive page
              \sxdef{,#1}{{#2/-}{\tmp\iiendash}}
           \else                % the pages drop
              \sxdef{,#1}{{#2/+}{\tmp, #2}}
           \fi
        \else    % state: the pagelist ends by --
           \ifnum\tmpnum=\tmpa  % the consecutive page
              \sxdef{,#1}{{#2/-}{\tmp}}
           \else                % the pages drop
              \sxdef{,#1}{{#2/+}{\tmp\tmpa, #2}}
           \fi
        \fi
     \fi
   \else % first occurrence of the index item #1
     \sxdef{,#1}{{#2/+}{#2}}
     \global \expandafter\addto \expandafter\iilist \csname,#1\endcsname
   \fi
   \egroup
}
\def\iilist{} \def\iiendash{--}
```

Činnost makra \Xindex si vysvětlíme za chvíli podrobněji. Nejprve ovšem definujeme pomocné makro \firstdata \,⟨*heslo*⟩ \⟨*cs*⟩, které expanduje na \⟨*cs*⟩ ⟨*první-datový-údaj-hesla*⟩&. Je-li třeba \,aa definováno jako {prvni}{druhy}, pak \firstdata \,aa \cosi expanduje na \cosi prvni&. Tím máme možnost vyzískat data z makra. Podobně makro \seconddata \,⟨*heslo*⟩ \⟨*cs*⟩ expanduje na \⟨*cs*⟩ ⟨*druhý-datový-údaj-hesla*⟩&.

\iiB: 18–19 \iiC: 19 \iid: 19 \iiD: 19 \Xindex: 18–20 \iilist: 19–21, 25–26
\firstdata: 19–21, 25 \seconddata: 19–21

opmac.tex
```
509: \def\firstdata#1#2{\expandafter\expandafter\expandafter #2\expandafter\firstdataA#1}
510: \def\firstdataA#1#2{#1&}
511: \def\seconddata#1#2{\expandafter\expandafter\expandafter #2\expandafter\seconddataA#1}
512: \def\seconddataA#1#2{#2&}
```

Než se pustíme do výkladu makra \Xindex, vysvětlím, proč pro hesla rejstříku tvořím jednu řídicí sekvenci, která je makrem se dvěma datovými údaji. Mohl bych jednodušeji pracovat se dvěma různými řídicími sekvencemi, např. \,⟨heslo⟩ a \:⟨heslo⟩. Důvod je prostý: šetřím paměť TEXu. Dá se totiž očekávat, že počet hesel v rejstříku můžeme počítat na tisíce a je rozdíl alokovat kvůli tomu tisíce kontrolních sekvencí nebo dvojnásobné množství takových sekvencí.

V makru \Xindex čteme \firstdata na řádku 482 a \seconddata na řádku 486. Čtení je provedeno makry \XindexA a \XindexB. První úsek dat je tvaru ⟨*poslední-strana*⟩/⟨*stav*⟩ a druhý úsek dat obsahuje rozpracovaný seznam stránek. Podíváme-li se na definice \XindexA a \XindexB, shledáme, že seznam stránek bude uložen v \tmp, dále ⟨*poslední-strana*⟩ bude v \tmpa a ⟨*stav*⟩ je v \tmpb.

opmac.tex
```
514: \def\XindexA#1/#2&{\def\tmpa{#1}\let\tmpb=#2}
515: \def\XindexB#1&{\def\tmp{#1}}
```

Rozlišujeme dva stavy: ⟨*stav*⟩=+, pokud je seznam stránek zakončen konkrétní stránkou. Tato konkrétní stránka je uložena v ⟨*poslední-strana*⟩. Druhým stavem je ⟨*stav*⟩=-, když je seznam stránek ukončen -- (přesněji obsahem makra \iiendash, které můžete snadno předefinovat) a v tomto případě ⟨*poslední-strana*⟩ obsahuje poslední stránku, na které byl zjištěn výskyt hesla. Tato strana nemusí být v seznamu stránek explicitně uvedena.

Makro \Xindex{⟨*heslo*⟩}{⟨*strana*⟩} tedy postupně vytváří seznam stran zhruba takto:

```
if (první výskyt \,⟨heslo⟩) {
  založ \,⟨heslo⟩ do iilist;
  ⟨seznam-stran⟩ = "⟨strana⟩"; ⟨stav⟩ = +; ⟨posledni-strana⟩ = ⟨strana⟩;
  return;
}
if (⟨strana⟩ == ⟨posledni-strana⟩) return;
if (⟨stav⟩ == +) {
  if (⟨strana⟩ == ⟨posledni-strana⟩+1) {
    ⟨seznam-stran⟩ += "--";
    ⟨stav⟩ = - ;
  }
else {
    ⟨seznam-stran⟩ += ", ⟨strana⟩";
    ⟨stav⟩ = + ;
  }
  else {
    if (⟨strana⟩ > ⟨posledni-strana⟩+1) {
      ⟨seznam-stran⟩ += "⟨posledni-strana⟩, ⟨strana⟩";
      ⟨stav⟩ = + ;
    }
  }
}
⟨poslední-strana⟩ = ⟨strana⟩;
```

Makro \makeindex nejprve definuje přechodný význam rekurzivního \act tak, aby byly uzavřeny seznamy stránek (tj. aby seznam nekončil znakem --) a do první datové oblasti každého makra typu \,⟨heslo⟩ vloží konverzi textu ⟨heslo⟩ do tvaru vhodném pro abecední řazení českých slov. Pomocí \expandafter␣\act␣\iilist␣\relax se požadovaná činnost vykoná pro každý prvek v \iilist. Dále makro \makeindex provede seřazení \iilist podle abecedy makrem \dosorting a nakonec provede tisk jednotlivých hesel. K tomu účelu znovu přechodně předefinuje \act a předloží mu \iilist.

\XindexA: 19–21 \XindexB: 19–21 \iiendash: 19 \makeindex: 20–22, 24

opmac.tex
```
\def\makeindex{\par
  \ifx\iilist\empty \opwarning{index data-buffer is empty. TeX me again}
    \else
    \bgroup
      \setprimarysorting
      \def\act##1{\ifx##1\relax \else
         \firstdata##1\XindexA \seconddata##1\XindexB
         \if\tmpb+%
            \preparesorting##1% converted item by sorting data in \tmpb
            \xdef##1{{\tmpb}{\tmp}}
          \else
            \preparesorting##1% converted item by sorting data in \tmpb
            \xdef##1{{\tmpb}{\tmp\tmpa}}
          \fi
          \expandafter\act\fi}
      \expandafter \act \iilist \relax
    \egroup
    \dosorting  % sorting is in progress
    \iiparparams
    \gdef\act##1{\ifx##1\relax \else \prepii##1%
             \seconddata##1\printiipages \expandafter\act \fi}
      \expandafter \act \iilist \relax
    \orippx \global\let\act=\undefined \global\let\orippx=\undefined
  \fi
}
```

Makro \printiipages sebere z ⟨*druhého-datového-údaje*⟩ seznam stránek a jednoduše je vytiskne.

opmac.tex
```
\def\printiipages#1&{ #1\par}
```

Makro „prepare index item“ \prepii \,⟨*heslo*⟩ odstraní prostřednictvím \prepiiA z názvu kontrolní sekvence backslash a čárku a zbytek tiskne pomocí \printii. Pokud ale je \,⟨*heslo*⟩ uloženo v seznamu \iispeclist, pak se expanduje sekvenci s názvem \\,⟨*heslo*⟩, ve které je uloženo, co se má místo hesla vytisknout. Data těchto výjimek jsou připravena makrem \iis

opmac.tex
```
\def\prepii #1{\isinlist \iispeclist #1\iftrue
   \expandafter\expandafter\expandafter \printii \csname\string#1\endcsname&%
   \else \expandafter\prepiiA\string #1&%
   \fi
}
\def\prepiiA #1#2#3&{\printii#3&}
```

Kontrolní otázka: proč se nedotazujeme jednoduše na to, zda je \\,⟨*heslo*⟩ definovaná řídicí sekvence? Odpověď: museli bychom ji sestavit pomocí \csname...\endcsname, ale to založí do TEXové paměti novou řídicí sekvenci pro každé heslo v rejstříku. My se snažíme počet těchto řídicích sekvencí redukovat na minimum. Počítáme s tím, že obyčejných hesel bude tisíce a výjimek jen pár desítek.

Makro \iis ⟨*heslo*⟩␣{⟨*text*⟩} vloží další údaj do slovníku výjimek pro hesla v rejstříku. Přesněji: vloží \,⟨*heslo*⟩ do \iispeclist a definuje sekvenci \\,⟨*heslo*⟩ jako ⟨*text*⟩.

opmac.tex
```
\def\iis #1 #2{\bgroup \def~{ }%
    \global\expandafter\addto\expandafter\iispeclist\csname,#1\endcsname
    \global\sdef{\expandafter\string\csname,#1\endcsname}{#2}%
  \egroup \ignorespaces
}
\def\iispeclist{}
```

Makro „print index item“ \printii ⟨*heslo*⟩& vytiskne jeden údaj do rejstříku. Makro projde prostřednictvím \printiiA jednotlivá podslova oddělená lomítkem a přepíše je do rejstříku odděleny mezerou. Přitom kontroluje, zda se podslova rovnají odpovídajícím podslovům z předchozího hesla, které je uloženo v \previi. Toto porovnání je protaženo krz \meaning, protože nechceme porovnávat kategorie, ale jen stringy. Pokud se stringy rovnají, místo podslova se vloží \iiemdash, což je pomlka. Na konci činnosti se nastaví \previi na \currii (nové slovo se pro další zpracování stává předchozím)

\printiipages: 21 \prepii: 21 \prepiiA: 21 \iis: 21 \iispeclist: 21 \printii: 21–22
\printiiA: 22 \previi: 21–22 \iiemdash: 22 \currii: 22

a vytiskne se seznam stránek. Makrem \everyii (implicitně je prázdné) dovolíme uživateli vstoupit do procesu tisku hesla. Může například psát \def\everyii{\indent}, pokud chce.

opmac.tex
```
\def\printii #1&{\gdef\currii{#1}\noindent\everyii
   \hskip-\iindent \ignorespaces\printiiA#1//}
\def\printiiA #1/{\if^#1^\let\previi=\currii \else
   \expandafter\scanprevii\previi/&\def\tmpb{#1}\edef\tmpb{\meaning\tmpb}%
   \ifx\tmpa\tmpb \iiemdash \else#1 \gdef\previi{}\fi
   \expandafter\printiiA\fi
}
\def\iiemdash{\kern.1em---\space}
\def\everyii{}
```

Makro \iiparparams nastavuje parametry sazby odstavce v rejstříku. Vlevo budeme mít \leftsip rovný \iindent, ale první řádek posuneme o -\iindent (viz řádek kódu 559) takže první řádek je vystrčen doleva. Vpravo máme pružnou mezeru, aby se seznam čísel stran mohl rozumně lámat, když je moc dlouhý. Makro \iiparparams si musí poznačit do makra \orippx původní údaje měněných hodnot. Není možné se totiž schovat do skupiny, protože rejstřík je obvykle tištěn pomocí \begmulti...\endmulti a toto makro občas ukončuje plnění boxu a spouští \flushcolumns. Kdybychom měli \makeindex ve skupině, pak by při \flushcolumns došlo ke křížení skupin. Na konci práce \makeindex na řádku 539 je makro \orippx zavoláno a tím jsou parametry odstavce vráceny do původní podoby.

opmac.tex
```
\def\iiparparams{%
   \xdef\orippx{\global\rightskip=\the\rightskip
               \global\leftskip=\the\leftskip
               \global\exhyphenpenalty=\the\exhyphenpenalty}
   \global\rightskip=0pt plus1fil
   \global\exhyphenpenalty=10000
   \global\leftskip=\iindent
}
```

Pomocné makro \scanprevii ⟨*expanded-previi*⟩& se podívá do \previi, odloupne z něj úsek před prvním lomítkem a tento úsek definuje jako \tmpa.

opmac.tex
```
\def\scanprevii#1/#2&{\def\previi{#2}\def\tmpa{#1}\edef\tmpa{\meaning\tmpa}}
```

Výchozí hodnota \previi před zpracováním prvního slova v rejstříku je prázdná.

opmac.tex
```
\def\previi{} % previous index item
```

3.13 Abecední řazení rejstříku

Nejprve se zaměříme na vytvoření makra \isAleB \,⟨*heslo1*⟩ \,⟨*heslo2*⟩, které rozhodne, zda je ⟨*heslo1*⟩ řazeno za ⟨*heslem2*⟩ nebo ne. Výsledek zkoumání můžeme prověřit pomocí \ifAleB.

Pro porovnání dvou údajů vyžaduje norma dva průchody. V prvním (primární řazení) se rozlišuje jen mezi písmeny A B C Č D E F G H Ch I J K L M N O P Q R Ř S Š T U V W X Y Z Ž. Pokud jsou hesla z tohoto pohledu stejná, pak se provede druhý průchod (sekundární řazení), ve kterém jsou řazena neakcentovaná písmena před přehlasovaná před čárkovaná před háčkovaná před stříškovaná před kroužkovaná a dále s nejnižší prioritou malá písmena před velká. Připravíme si tedy dvě sady \lccode dvojic: pro první průchod a pro druhý. Porovnávaná hesla zkonvertujeme pomocí \lowercase při nastavení \lccode odpovídajícího průchodu. Pak takto zkonvertovaná hesla teprve začneme porovnávat.

Makro \setprimarysorting připraví \lccode znaků české a slovenské abecedy pro první průchod a \setsecondarysorting pro druhý průchod. Makro \setprimarysorting expanduje \sortingdata a předhodí před takto expandovaná data \act. Povšimneme si, že pro první průchod dostanou stejný \lccode všechny znaky na společném řádku makra \sortingdata, zatímco v druhém průchodu budou mít všechny znaky z tohoto makra rozdílný \lccode, ve vzestupném pořadí. Je to tím, že v makru \setprimarysorting se zvedá \tmpnum jen v místě čárky, zatímco v \setsecondarysorting se \tmpnum zvedá pro každý znak. Nejnižší hodnotu má mezera vyznačená v \sortingdata pomocí {␣}. Tím je zaručeno, že kratší slovo je řazeno dřív než delší slovo se stejným začátkem, obsahující celé kratší slovo

\everyii: 22 \iiparparams: 21–22 \orippx: 21–22 \scanprevii: 22
\setprimarysorting: 21–23, 25 \setsecondarysorting: 22–23, 25 \sortingdata: 22–23

(ten tučňák≺tento). Je sice pravda, že ASCII hodnota mezery je ještě menší, ale my musíme mezeru někam šoupnout na jiný kód než 32, jinak by nám ji nepřečetlo makro s neseparovaným parametrem. Ovšem my budeme chtít mezeru přečíst.

opmac.tex

```
\def\sortingdata{%
  /,{ },-,&,@,%
  aA\"a\"A\'a\'A,%
  bB,%
  cC,%
  \v c\v C,%
  dD\v d\v D,%
  eE\'e\'E\v e\v E,%
  fF,%
  gG,%
  h^^HH,%
  ^^T^^U^^V,%
  iI\'i\'I,%
  jJ,%
  kK,%
  lL\'l\'L\v l\v L,%
  mM,%
  nN\v n\v N,%
  oO\"o\"O\'o\'O\^o\^O,%
  pP,%
  qQ,%
  rR\'r\'R,%
  \v r\v R,%
  sS,%
  \v s\v S,%
  tT\v t\v T,%
  uU\"u\"U\'u\'U\r u\r U,%
  vV,%
  wW,%
  xX,%
  yY,\'y\'Y,%
  zZ,%
  \v z\v Z,%
  0,1,2,3,4,5,6,7,8,9,'.%
}
\def\setprimarysorting {\csname sort:\csname lan:\the\language\endcsname \endcsname
  \def\act##1{\ifx##1.\else
     \ifx##1,\advance\tmpnum by1
     \else \lccode`##1=\tmpnum \fi
     \expandafter \act \fi}%
  \ifx\r\undefined
     \opwarning{\noexpand\csaccents is unused, falling back to ASCII sorting}%
     \gdef\sortingdata{.}\global\let\chsorting=n%
  \else
     \xdef\sortingdata{\sortingdata}% expand sorting data now
  \fi
  \tmpnum=133 \expandafter \act\sortingdata \setignoredchars
}
\sdef{sort:en}{\global\let\chsorting=n} % skipping ch processing in English language

\def\setsecondarysorting {\def\act##1{\ifx##1.\else
     \ifx##1,\else \advance\tmpnum by1 \lccode`##1=\tmpnum \fi
     \expandafter \act \fi}%
  \tmpnum=133 \expandafter \act\sortingdata \setignoredchars
}
```

Jedním z problémů českého řazení je dvojhláska ch. Tu potřebujeme proměnit v jediný znak. Pro potřeby řazení proměníme ch v `^^T`, Ch v `^^U` a CH v `^^V`. Uděláme to následujícím okultním kódem, který definuje makro `\preparesorting` `\,`⟨*heslo*⟩. Toto makro připraví pomocí do `\tmpb` ⟨*heslo*⟩ zkonvertované krz `\lowercase`, ovšem nejprve je dvojhláska ch nahrazena jedním znakem. Makro `\chsorting`

`\preparesorting`: 21, 24–25 `\chsorting`: 23–24

je implicitně nedefinované, což znamená, že pracujeme s dvojhláskou ch. Na řádku 631, 618 a 625 je ovšem nastaveno \chsorting jako n, což je vzkaz, že dvojhlásku ch nechceme interpretovat.

opmac.tex

```
638: \bgroup
639: \lccode‘4=‘c \lccode‘5=‘h \lccode‘6=‘C \lccode‘7=‘H
640: \lowercase{
641: \gdef\iiscanch #145#2\relax{#1\if$#2$\else^^T\iiscanch #2\relax\fi}
642: \gdef\iiscanCh #165#2\relax{#1\if$#2$\else^^U\iiscanCh #2\relax\fi}
643: \gdef\iiscanCH #167#2\relax{#1\if$#2$\else^^V\iiscanCH #2\relax\fi}
644: \gdef\preparesorting#1{\expandafter\preparesortingA\string#1&}
645: \gdef\preparesortingA#1#2#3&{\xdef\tmpb{#3}%
646:   \ifx\chsorting\undefined
647:     \xdef\tmpb{\expandafter\iiscanch\tmpb 45\relax}%
648:     \xdef\tmpb{\expandafter\iiscanCh\tmpb 65\relax}%
649:     \xdef\tmpb{\expandafter\iiscanCH\tmpb 67\relax}\fi
650:   \lowercase\expandafter{\expandafter\gdef\expandafter\tmpb\expandafter{\tmpb}}%
651:   \xdef\tmpb{\expandafter\removedot\tmpb.\relax}%
652: }}
653: \egroup
```

Tento kód je bohužel obtížněji čitelný, protože makra \iiscanch a další potřebují mít separátor ch ve stavu, kdy jednotlivá písmena mají \catcode 12. Pracujeme totiž s výstupem primitivu \string, který bohužel vše (až na mezeru) balí do tokenů s kategorií 12. Proto je celý kód obalen do \bgroup, \egroup a \lowercase. Tam jsou znaky 4, 5, 6, 7, které mají kategorii 12, šoupnuty na c, h, C, H. Po tomto dešifrování tedy vidíme, že makro \iiscanch je definováno takto:

```
\gdef\iiscanch #1ch#2\relax{#1\if$#2$\else^^T\iiscanch #2\relax\fi}
\iiscanch Schází hrách, který bych házel na stěnu.ch\relax
```

Uvedený příklad expanduje postupně na

```
#1<-S
#2<-ází hrách, který bych házel na stěnu.ch
=> s^^T\iiscanch #2\relax

#1<-ází hrá
#2<-, který bych házel na stěnu.ch
=> s^^Tází hrá^^T\iiscanch #2\relax

#1<-, který by
#2<- házel na stěnu.ch
=> s^^Tází hrá^^T, který by^^T\iiscan #2\relax

#1<- házel na stěnu.
#2<-
=> s^^Tází hrá^^T, který by^^T házel na stěnu.
```

a to nahradí všechny výskyty dvojhlásky ch znakem ^^T. Analogicky pracují makra \iiscanCh a \iiscanCH. Makro \preparesortingA nakonec zavolá všechna tři makra, takže máme nahrazeny všechny dvojhlásky ch, Ch i CH.

V rámci optimalizace rychlosti jsou před algoritmem na setřídění seznamu všechna hesla jednorázově zkonvertovaná podle pravidel prvního průchodu řazení a tato data jsou uložena v prvním datovém údaji hesla (ve druhém máme seznam stránek). Není tedy nutné dělat konverzi při každém porovnávání dvou hesel. Ovšem, pokud porovnání hesel vyjde bez rozdílu, je potřeba provést druhý průchod řazení (sekundární řazení). Ten nastavujeme jednotlivě jen pro takové dvojice hesel, kde to je potřeba. Pravděpodobnost, že to je vůbec někdy potřeba, je mizivá. Data pro primární řazení jsou tedy už připravena na řádcích 523 až 531 v makru \makeindex.

V českém řazení se nemá přihlížet na interpunkční znaky (tečka, středník, otazník, atd.). Hesla máme řadit tak, jako kdyby tam tyto znaky nebyly. Kdyby se dvě hesla podle tohoto pravidla nelišila,

\iiscanch: 24, 34 \iiscanCh: 24 \iiscanCH: 24 \preparesortingA: 24

norma předepisuje nasadit cca čtvrtý průchod, ve kterém se tyto znaky rozliší. Čtvrtý průchod implementován není: hesla lišící se jen interpunkčními znaky, jsou v OPmac při řazení nerozlišitelná, tj. jsou řazena v pořadí, v jakém vstupují do rejstříku. Ignorování interpunkčních znaků je provedeno tak, že všem těmto znakům je přidělen makrem \setignoredchars \lccode tečky a tečka je při zpracování v \setprimarysorting a \setsecondarysorting odstraněna makrem \removedot.

opmac.tex
```
\def\removedot #1.#2\relax{#1\if$#2$\else\removedot #2\relax\fi}
\def\setignoredchars{\setlccodes ,.;.?.!.:.'."|.(.).[.].<.>.=.+.{}{}}
```

Připravíme si \newif, kterým ohlásíme výsledek porovnání dvou hesel:

opmac.tex
```
\newif \ifAleB
```

Makro \isAleB \,⟨*heslo1*⟩ \,⟨*heslo2*⟩ spustí \testAleB ⟨*zkonvertované-heslo1*⟩&\relax ⟨*zkonvertované-heslo2*⟩&\relax \,⟨*heslo1*⟩ \,⟨*heslo2*⟩.

opmac.tex
```
\def\isAleB #1#2{%
  \edef\tmp{\firstdata#1\empty\relax\firstdata#2\empty\relax%
            \noexpand#1\noexpand#2}%
  \expandafter \testAleB \tmp
}
```

Idea makra \testAleB lexikograficky porovnávající dvě slova je v tom, že ze dvou stringů v parametru oddělených \relax postupně odlupuje vždy první znak #1 a #3 z každého stringu a ten porovnává a samozřejmě při rovnosti rekurzivně zavolá samo sebe. Pokud jsme se dostali na konec bez rozhodnutí, co je menší, narazíme na znak &. V takovém případě přestoupíme do sekundárního průchodu.

opmac.tex
```
\def\testAleB #1#2\relax #3#4\relax #5#6{%
  \if #1#3\if #1&\testAleBsecondary #5#6%
          \else \testAleB #2\relax #4\relax #5#6%
          \fi
  \else \ifnum `#1<`#3 \AleBtrue \else \AleBfalse \fi
  \fi
}
```

Makro \testAleBsecondary \,⟨*heslo1*⟩ \,⟨*heslo2*⟩ založí skupinu, v ní nastaví \lccode dle sekundárního řazení a pomocí \preparesorting připraví zkonvertovaná data do \tmpa a \tmpb. Na chvosty těchto dat přidám nulu a jedničku, aby porovnání vždy nějak dopadlo, a spustím \testAleBsecondaryX, což pracuje obdobně, jako \testAleB.

opmac.tex
```
\def\testAleBsecondary#1#2{%
  \bgroup
    \setsecondarysorting
    \preparesorting#1\let\tmpa=\tmpb \preparesorting#2%
    \edef\tmp{\tmpa0\relax\tmpb1\relax}%
    \expandafter\testAleBsecondaryX \tmp
  \egroup
}
\def\testAleBsecondaryX #1#2\relax #3#4\relax {%
  \if #1#3\testAleBsecondaryX #2\relax #4\relax
  \else \ifnum `#1<`#3 \global\AleBtrue \else \global \AleBfalse \fi
  \fi
}
```

Nyní můžeme pomocí \isAleB\,⟨*heslo1*⟩\,⟨*heslo2*⟩\ifAleB rozhodnout, který ze dvou daných parametrů má být řazen dříve. Stačí tedy už jen naprogramovat celkové řazení seznamu. Toto makro vycházející z algoritmu mergesort vytvořil můj syn Miroslav. Makro bylo poprvé použito v DocByTEXu, což je nástroj, kterým je například pořízena i tato dokumentace.

Makro \dosorting pomocí pomocného makra \act doplní za každý údaj v \iilist čárku a dále předloží makru \mergesort jako parametr obsah \iilist ukončený \end,\end, vyprázdní \iilist a spustí \mergesort.

\setignoredchars: 23, 25 \removedot: 24–25 \isAleB: 22, 25–26 \testAleB: 25
\testAleBsecondary: 25 \testAleBsecondaryX: 25 \dosorting: 20–21, 26

opmac.tex
```
\def\dosorting{%
   \message{Opmac: Sorting index...}
   \def\act##1{\ifx##1\relax\else \global\addto\iilist{##1,}%
             \expandafter\act\fi}
   \expandafter\removeiilist \expandafter\act \iilist\relax
   \expandafter\removeiilist \expandafter\mergesort \iilist \end,\end
}
\def\removeiilist{\gdef\iilist{}}
```

Makro \mergesort pracuje tak, že bere ze vstupní fronty vždy dvojici skupin položek, každá skupina je zatříděná. Skupiny jsou od sebe odděleny čárkami. Tyto dvě skupiny spojí do jedné a zatřídí. Pak přejde na následující dvojici skupin položek. Jedno zatřídění tedy vypadá například takto: dvě skupiny: `eimn,bdkz,` promění v jedinou skupinu `bdeikmnz,`. V tomto příkladě jsou položky jednotlivá písmena, ve skutečnosti jsou to kontrolní sekvence, které obsahují celá slova.

Na počátku jsou skupiny jednoprvkové (\iilist odděluje každou položku čárkou). Makro \mergesort v tomto případě projde seznam a vytvoří seznam zatříděných dvoupoložkových skupin, uložený zpětně v \iilist. V dalším průchodu znovu vyvrhne \iilist do vstupní fronty, vyprázdní ho a startuje znovu. Nyní vznikají čtyřpoložkové zatříděné skupiny. Pak osmipoložkové atd. V závěru (na řádku 704) je první skupina celá setříděná a druhá obsahuje `\end`, tj. všechny položky jsou už setříděné v první skupině, takže stačí ji uložit do \iilist a ukončit činnost. Pomocí \gobbletoend odstraníme druhé `\end` ze vstupního proudu.

opmac.tex
```
\def\mergesort #1#2,#3{% by Miroslav Olsak
   \ifx,#1                      % prazdna-skupina,neco,  (#2=neco #3=pokracovani)
      \addto\iilist{#2,}        % dvojice skupin vyresena
      \return{\fif\mergesort#3}%    % \mergesort pokracovani
   \fi
   \ifx,#3                      % neco,prazna-skupina,  (#1#2=neco #3=,)
      \addto\iilist{#1#2,}%     % dvojice skupin vyresena
      \return{\fif\mergesort}%      % \mergesort dalsi
   \fi
   \ifx\end#3                   % neco,konec (#1#2=neco)
      \ifx\empty\iilist             % neco=kompletni setrideny seznam
         \def\iilist{#1#2}%
         \return{\fif\fif\gobbletoend}%   % koncim
      \else                     % neco=posledni skupina nebo \end
         \return{\fif\fif \expandafter\removeiilist    % spojim \indexbuffer+necoa cele znova
               \expandafter\mergesort\iilist#1#2,#3}%
   \fi\fi                       % zatriduji: p1+neco1,p2+neco2, (#1#2=p1+neco1 #3=p2)
   \isAleB #1#3\ifAleB          % p1<p2
      \addto\iilist{#1}%        % p1 do bufferu
      \return{\fif\mergesort#2,#3}%         % \mergesort neco1,p2+neco2,
   \else                        % p1>p2
      \addto\iilist{#3}%        % p2 do bufferu
      \return{\fif\mergesort#1#2,}%         % \mergesort p1+neco1,neco2,
   \fi
   \relax % zarazka, na ktere se zastavi \return
}
```

Jádro \mergesort vidíme na řádcích 711 až 716. Makro \mergesort sejme ze vstupního proudu do `#1` první položku první skupiny, do `#2` zbytek první skupiny a do `#3` první položku druhé skupiny. Je-li `#1<#3`, je do výstupního zatříděného seznamu `\indexbuffer` vložen `#1`, ze vstupního proudu je `#1` odebrán a \mergesort je zavolán znovu. V případě `#3<#1` je do `\indexbuffer` vložen `#3`, ze vstupního proudu je `#3` odebrán a \mergesort je zavolán znovu. Řádky 695 až 701 řeší případy, kdy je jedna ze skupin prázdná: je potřeba vložit do `\indexbuffer` zbytek neprázdné skupiny a přejít na další dvojici skupin. Ostatní řádky makra se vyrovnávají se skutečností, že zpracování narazilo na zarážku `\end,\end` a je tedy potřeba vystartovat další průchod.

3.14 Více sloupců

Makro pro sazbu do více sloupců je převzato z TBN, kde je podrobně vysvětleno na stranách 224 až 245. Základní myšlenka makra spočívá v tom, že se naplní jeden velký `\vbox` (box6) jedním sloupcem a

\mergesort: 25–26 \gobbletoend: 26

\endmulti jej rozlomí do sloupců požadované výšky a strčí do sazby. Není k tomu nutno měnit výstupní rutinu. Makro z TBN je zde v OPmac ve dvou věcech přepracováno:

- Důslednější balancování sloupců vylučující možnost ztráty sazby a umožňující mít sazbu s nezlomitelnými mezerami mezi řádky.
- Makro měří kumulovanou sazbu a umožňuje při rozsáhlém množství tiskového materiálu přechodně přejít do režimu „vyprazdňování“.

Makra \begmulti, \endmulti, \corrsize, \makecolumns a \splitpart pracují zhruba tak, jak je popsáno v TBN.

opmac.tex

```
\def\corrsize #1{%% #1 := #1 + \splittopskip - \topskip
   \advance #1 by \splittopskip \advance #1 by-\topskip}

\def\begmulti #1 {\par\wipeepar\multiskip\penalty0 \def\Ncols{#1}
   \splittopskip=\baselineskip
   \setbox6=\vbox\bgroup\penalty0
   %% \hsize := Sirka sloupce = (\hsize+\colsep) / n - \colsep
   \advance\hsize by\colsep
   \divide\hsize by\Ncols  \advance\hsize by-\colsep
   \dimen0=0pt
   \def\par{\endgraf\advance\dimen0 by\the\prevgraf\baselineskip
            \ifdim\dimen0>.9\maxdimen \message{flushcolumns:}%
               \global\let\balancecolumns=\flushcolumns \expandafter \endmulti
            \fi}%
}
\def\endmulti{\vskip-\prevdepth\vfil\egroup \setbox1=\vsplit6 to0pt
   %% \dimen1 := the free space on the page
   \ifdim\pagegoal=\maxdimen \dimen1=\vsize \corrsize{\dimen1}
   \else \dimen1=\pagegoal \advance\dimen1 by-\pagetotal \fi
   \ifdim \dimen1<2\baselineskip
     \vfil\break \dimen1=\vsize \corrsize{\dimen1} \fi
   \dimen0=\ht6 \divide\dimen0 by\Ncols \relax
   %% split the material to more pages?
   \ifdim \dimen0>\dimen1 \splitpart
   \else \balancecolumns \fi  % only balancing
   \multiskip\relax}
\def\makecolumns{\bgroup % full page, destination height: \dimen1
   \vbadness=20000 \setbox1=\hbox{}\tmpnum=0
   \loop \ifnum\Ncols>\tmpnum
      \advance\tmpnum by1
      \setbox1=\hbox{\unhbox1 \vsplit6 to\dimen1 \hss}
   \repeat
   \hbox{}\nobreak\vskip-\splittopskip \nointerlineskip
   \line{\unhbox1\unskip}
   \egroup}
\def\splitpart{%
   \makecolumns % full page
   \vskip 0pt plus 1fil minus\baselineskip \break
   \dimen0=\ht6 \divide\dimen0 by\Ncols \relax
   \dimen1=\vsize \corrsize{\dimen1}\dimen2=\dimen1
   \advance\dimen2 by-\Ncols\baselineskip
   %% split the material to more pages?
   \ifvoid6 \else
      \ifdim \dimen0>\dimen2 \expandafter\expandafter\expandafter \splitpart
      \else \balancecolumns % last balancing
   \fi \fi
}
```

Výstup rozlomené sazby do sloupců probíhá ve dvou režimech: když je třeba sloupci zaplnit celou stránku, použijeme \makecolumns. Toto makro neřeší otázku, že může v kumulovaném boxu 6 zbýt nějaká sazba, protože se předpokládá, že lámání bude pokračovat na další straně. Pokud ale je na aktuální straně vícesloupcová sazba ukončena, použijeme propracovanější \balancecolumns. Toto makro si zazálohuje materiál z boxu 6 do boxu 7 a jme se zkoušeti rozlomit box 6 na sloupce s výškou \dimen0. Pokud ale po

\begmulti: 6, 22, 27–28 \endmulti: 6, 22, 27–28 \corrsize: 27 \makecolumns: 27
\splitpart: 27–28 \balancecolumns: 27–28

rozlomení není výchozí box 6 zcela prázdný, makro zvětší krapánek (o 0,2\baselineskip) požadovanou výšku, vrátí se k zálohované sazbě v boxu 7 a zkusí rozlomit znovu. To opakuje tak dlouho, dokud je box 6 prázdný.

opmac.tex

```
\def\balancecolumns{\bgroup \setbox7=\copy6 % destination height: \dimen0
   \ifdim\dimen0>\baselineskip \else \dimen0=\baselineskip \fi
   \vbadness=20000
   \def\tmp{%
      \setbox1=\hbox{}\tmpnum=0
      \loop \ifnum\Ncols>\tmpnum
         \advance\tmpnum by1
         \setbox1=\hbox{\unhbox1
              \ifvoid6 \hbox to\wd6{\hss}\else \vsplit6 to\dimen0 \fi\hss}
      \repeat
   \ifvoid6 \else
      \advance \dimen0 by.2\baselineskip
      \setbox6=\copy7
      \expandafter \tmp \fi}\tmp
   \hbox{}\nobreak\vskip-\splittopskip \nointerlineskip
   \hbox to\hsize{\unhbox1\unskip}%
   \egroup
}
```

Když je sazba plněna do boxu 6, může ji být tak moc, že tento box překročí maximální výšku boxu, která je v TEXu bohužel omezena na cca pět metrů (16383 pt). Proto na řádku 736 je předefinován \par, který přičítá do \dimen0 celkovou výšku postupně kumulované sazby. Jakmile tato výška dosáhne 0.9\maxdimen, předefinujeme makro \balancecolumns na \flushcolumns a spustíme předčasně \endmulti. Toto makro vyprázdní box pomocí opakovaného \splitpart, ovšem nevyprázdní ho celý. Jen tu část, která zaplní celé stránky. Jakmile bude chtít \splitpart přejít k vybalancování sazby pomocí \balancecolumns na řádku 770, spustí se místo běžného \balancecolumns makro \flushcolumns. Toto makro ignoruje zbytek činnosti \endmulti až po \relax na řádku 751, takže vyskočí z trojitě zanořeného \if. Musí tedy vrátit příslušné množství \fi a dále vystartuje nový \setbox6=\vbox. Uvnitř tohoto boxu nejprve vysype zbytek boxu 6, pomocí \unskip\unskip odstraní \vfil a \vskip-\prevdepth, který tam vložil \endmulti na řádku 741, vrátí se k zálohovanému významu makra \ibalancecolumns a znovu definuje \par obdobným způsobem jako v \begmulti. Pak pokračuje ve čtení sazby.

opmac.tex

```
\def\flushcolumns#1\relax{\fi\fi\fi
   \setbox6=\vbox\bgroup\penalty0
   \global\let\balancecolumns=\ibalancecolumns
   \dimen0=\ht6 \unvbox6 \unskip\unskip
   \advance\hsize by\colsep
   \divide\hsize by\Ncols  \advance\hsize by-\colsep
   \def\par{\endgraf\advance\dimen0 by\the\prevgraf\baselineskip
            \ifdim\dimen0>.9\maxdimen \message{flush-columns:}%
               \global\let\balancecolumns=\flushcolumns \expandafter \endmulti
            \fi}%
}
\let\ibalancecolumns=\balancecolumns
```

3.15 Barvy

Od verze pdfTEXu 1.40 nabízí tento program primitivy \pdfcolorstackinit a \pdfcolorstack. Makra v OPmac tyto primitivy nepoužívají, protože:

- Řešení v OPmac jsem vytvořil a použil podstatně dřív, než si programátoři pdfTEXu vůbec všimli, že existuje problém s přecházením barev na nové stránky.
- Primitivy \pdfcolorstackinit a \pdfcolorstack stále nejsou dokumentované.

Deklarujeme \ifwritecolor, což nastaveno na true způsobí, že makro \writecolor bude zapisovat do REF souboru informaci o zrovna nastavené barvě. Tuto informaci pak zužitkujeme ve výstupní rutině při nastavování barev, které přetékají ze strany na stranu. Dále deklarujeme \lastpage. Tento registr budeme potřebovat pro identifikaci jednotlivých stran při čtení REF souboru.

\flushcolumns: 22, 27–28 \ibalancecolumns: 28 \ifwritecolor: 29–30 \lastpage: 12, 29, 31–32, 45

opmac.tex
```
\newif\ifwritecolor \writecolortrue
\newcount\lastpage  \lastpage=0  % the last page of the document
```

Barvy se v PDF přepínají pomocí PDF speciálů ⟨*num*⟩␣⟨*num*⟩␣⟨*num*⟩␣⟨*num*⟩␣k (pro text a plochy) a ⟨*num*⟩␣⟨*num*⟩␣⟨*num*⟩␣⟨*num*⟩␣K pro linky. Pro oba typy barev připravíme barevná makra \Blue, \Red, \Brown, \Green, \Yellow, \Cyan, \Magenta, \White, \Grey, \LightGrey, \Black. Uživatel si může definovat další.

opmac.tex
```
\def\Blue{\setcmykcolor{1 1 0 0}}
\def\Red{\setcmykcolor{0 1 1 0}}
\def\Brown{\setcmykcolor{0 0.67 0.67 0.5}}
\def\Green{\setcmykcolor{1 0 1 0}}
\def\Yellow{\setcmykcolor{0 0 1 0}}
\def\Cyan{\setcmykcolor{1 0 0 0}}
\def\Magenta{\setcmykcolor{0 1 0 0}}
\def\White{\setcmykcolor{0 0 0 0}}
\def\Grey{\setcmykcolor{0 0 0 0.5}}
\def\LightGrey{\setcmykcolor{0 0 0 0.2}}
\def\Black{\setcmykcolor{0 0 0 1}}
```

Makro \setcmykcolor nastaví barvu textu (při \pdfK obsahující „k“) nebo barvu linek (při \pdfK obsahující „K“). Dále si toto makro globálně uloží nastavenou barvu do \currcolork, resp. \currcolorK. Také podle předchozí \currcolork, resp. \currcolorK testuje, zda je potřeba aktuální barvu změnit. Pokud ne, tak \special pro nastavení barvy nevloží.

opmac.tex
```
\def\setcmykcolor#1{%
   \def\tmp{#1}\expandafter \ifx\csname currcolor\pdfK\endcsname \tmp \else
      \special{PDF:#1 \pdfK}%
      \expandafter\xdef\csname currcolor\pdfK\endcsname{#1}%
      \writecolor\pdfK
   \fi}%
}
```

Problém barev v PDF je, že od výskytu speciálu pro barvu jsou změněné barvy až po jiný výskyt takového speciálu nebo po konec strany. Na každé nové straně začíná sazba v barvě černé. Nám ovšem někdy může obarvený text přetéci na další stranu. Pak ale musíme v \output rutině nastavit barvu, která přetekla. Ovšem jak poznáme, že něco přeteklo do další strany? Jedině pomocí asynchronního \write. Proto makro \setcmykcolor nevkládá do PDF jen požadovaný speciál, ale taky ukládá pomocí \writecolor ⟨*k-nebo-K*⟩ informaci do REF souboru ve formátu \Xpdfcolor{⟨*CMYK-barvy*⟩} nebo \XpdfcolorK{⟨*CMYK-barvy*⟩}.

opmac.tex
```
\def\writecolor#1{\ifwritecolor
      \openref
      \edef\act{\noexpand\wref\noexpand\Xpdfcolor{#1{\csname currcolor#1\endcsname}}}\act
   \fi
}
```

Makro \pdfK má implicitně hodnotu k (barva pro texty a plochy) a přechodně při použití \linecolor ⟨*přepínač-barvy*⟩ má hodnotu K.

opmac.tex
```
\def\pdfK{k}
\def\linecolor#1{{\def\pdfK{K}#1}}
```

Výchozí hodnoty maker \currcolork a \currcolorK jsou rovny barvě černé, neboli makru \pdfblackcolor.

opmac.tex
```
\def\pdfblackcolor{0 0 0 1}
\xdef\currcolork{\pdfblackcolor} \xdef\currcolorK{\pdfblackcolor}
```

Následuje výklad makra \localcolor. Uvědomíme si, co od něj vlastně očekáváme. Ukázka první:

```
\Blue základní text je modrý.
{\localcolor \Green Zelený, {\localcolor \Red červený,} tady zpátky zelený,
 \Grey Gandalf šedý} a tady zpátky základní modrý text.
```

Ukázka druhá:

```
\Blue základní text je modrý.
{\localcolor \Green \linecolor\Red Tady je zelený text a červené linky.}
Zde je zpátky text modrý a linky černé.
```

Z ukázky první plyne, že \localcolor si musí uložit informaci o právě nastavené barvě do zásobníku, ze kterého ji na konci skupiny vyzvedneme makrem \restorecolor. Toto makro pošleme na konec skupiny příkazem \aftergroup. Z ukázky druhé plyne, že do zásobníku musíme uložit nejen informaci o barvě textu (k) ale též informaci o barvě linek (K) a makro \restorecolor musí zrestaurovat oba typy barev.

Pro zásobník je rezervováno makro \savedcolors, do kterého jsou údaje vkládány vlevo a zleva jsou též vyzvedávány. Jeden údaj je ve tvaru ⟨*vzkaz*⟩{⟨*barva-k*⟩}{⟨*barva-K*⟩}. Výchozí hodnota zásobníku je prázdná.

opmac.tex

```
\xdef\savedcolors{}
```

Často se stává, že barvy uvnitř skupin jsou současně barvami v boxech a pak máme jistotu, že barva nepřeteče to další strany. Je tedy zbytečné ukládat informaci o přechodu barev do REF souboru. Takže makro \localcolor nastavuje lokálně uvnitř dané skupiny \writecolorfalse. Pokud uživatel pracuje se skupinou, která má tendenci utéci na další stranu, použije místo \localcolor makro \longlocalcolor. Makra \localcolor a \longlocalcolor tedy vypadají takto:

opmac.tex

```
\def\localcolor{\aftergroup\restorecolor \writecolorfalse
   \xdef\savedcolors{0{\currcolork}{\currcolorK}\savedcolors}}
\def\longlocalcolor{\aftergroup\restorecolor
   \ifwritecolor\else \opwarning{\noexpand\longlocalcolor inside
      \string\localcolor. Something wrong}\fi
   \writecolortrue
   \xdef\savedcolors{1{\currcolork}{\currcolorK}\savedcolors}}
\let\locpgcolor=\relax  % for backward compatibility
```

Vidíme, že ⟨*vzkaz*⟩=1 v zásobníku \savedcolors znamená, že je třeba změnu barvy provedenou na konci skupiny zapsat do REF souboru.

Makro \restorecolor usazené za koncem skupiny pomocí \aftergroup si vyzvedne potřebné tři údaje ze zásobníku. K tomu definuje makro \tmp, které to provede. Dále do \tmpa vloží ⟨*barvu-k*⟩ a do \tmpb vloží ⟨*barvu-K*⟩ a testem proti \currcolork, resp. \currcolorK zjistí, zda je vůbec potřeba barvu měnit. Pokud ne, nedělá nic. Jinak zapíše potřebný \special a přenastaví makro \currcolork, resp. \currcolorK. Konečně, při ⟨*vzkaz*⟩=1, zapíše nově nastavenou barvu do REF souboru. Celou práci vykoná uvnitř skupiny, takže lokální změny se vracejí po ukončení práce makra k původním hodnotám.

opmac.tex

```
\def\restorecolor{{\def\tmp##1##2##3##4\end{\xdef\savedcolors{##4}%
   \def\tmpa{##2}\def\tmpb{##3}\writecolortrue
   \ifx\tmpa\currcolork \else \special{PDF:##2 k}\xdef\currcolork{##2}%
      \ifnum##1=1 \writecolor k\fi\fi
   \ifx\tmpb\currcolorK \else \special{PDF:##3 K}\xdef\currcolorK{##3}%
      \ifnum##1=1 \writecolor K\fi\fi}%
   \expandafter\tmp\savedcolors\end}}
```

Přepínače barev stejně jako makra \localcolor nebo \longlocalcolor se mohou vyskytnout v nadpise. Takže je potřeba je zabezpečit proti rozsypání.

opmac.tex

```
\addprotect\setcmykcolor  \addprotect\localcolor  \addprotect\longlocalcolor
```

Aby mohla output rutina správně obsloužit barvy, je třeba učinit několik opatření, která jsou zhruba načrtnuta v následujícím schématku.

\savedcolors: 30 \longlocalcolor: 30–31, 33 \restorecolor: 30

```
\output = {
\begoutput  % Záloha aktuálních barev, nastavení \currcolorkK=černá
...
\makeheadline % Uživatel může měnit barvy, ale musí se vrátit k černé.
\pagecontents = {
  ...\topins     % Výchozí barva je černá, k černé je třeba se vrátit.
  \preboxcclv    % Nastavení barvy, která přetekla na tuto stranu.
  \unvbox256     % Sazba pro tuto stranu.
  \postboxcclv   % Návrat k barvě černé.
  ...\footins    % Poznámky pod čarou.
}
\makefootline % Výchozí barva je černá, uživatel může nastavit cokoli.
...
\endoutput  % návrat \currcolorkK k původním zálohovaným barvám
}
```

Při nastavování barev v `\headline` a `\footline` je možné použít `\localcolor`, protože zásobník se tím při práci output rutiny posune a také znovu splaskne na původní hodnotu a neovlivní tedy stav v běžné sazbě (mimo output rutinu). Ovšem běžná sazba má nastaveny nějak hodnoty `\currcolork` a `\currcolorK` a není žádoucí, aby byly tyto hodnoty output rutinou měněny. Proto jsou v makru `\begoutput` tyto hodnoty zálohovány a v makru `\endoutput` se k nim output rutina vrátí. Uvnitř output rutiny potlačíme všem přepínačům barev jejich tendenci ukládat něco do REF souboru, takže je v makru `\begoutput` použito `\writecolorfalse`.

opmac.tex

```
\def\begoutput{\writecolorfalse  \let\longlocalcolor=\localcolor
   \edef\restoreoutputcolor{%
      \xdef\noexpand\currcolork{\currcolork}\xdef\noexpand\currcolorK{\currcolorK}}%
   \xdef\currcolork{\pdfblackcolor}\xdef\currcolorK{\pdfblackcolor}%
   \immediate\wref\Xpage{{\the\pageno}}%
}
\def\endoutput{\restoreoutputcolor}
```

Poněkud složitější je makro `\preboxcclv`, které má nastavit barvu, která přetekla z předchozí strany. Abychom se k funkci tohoto makra dobrali, vraťme se k makrům `\Xpdfcolork` {⟨*CMYK-barva*⟩} a `\XpdfcolorK` {⟨*CMYK-barva*⟩}, která jsou uložena v REF souboru pro každé (potenciálně dlouhé) přepnutí barvy. Povšimněte si, že output rutina ukládá do REF souboru pro každou stranu makrem `\begoutput` údaj `\Xpage`{⟨*číslo-strany*⟩}. Tyto údaje tvoří oddělovače mezi jednotlivými stránkami.

Příkazy `\Xpdfcolork` a `\XpdfcolorK` ukládají při čtení REF souboru do `\pdflastcolork`, resp. `\pdflastcolorK`, naposledy použitou barvu. Takže na příští straně, až narazíme při čtení REF souboru na další `\Xpage`, budeme vědět, zda je tato naposledy použitá barva černá nebo jiná. Pokud jiná, je potřeba ji nastavit jako výchozí i pro tuto (tedy příští) stranu. Makro `\Xpage` v takovém případě uloží do makra `\pgc:`⟨*číslo-strany*⟩ povel `\setpgcolor`␣⟨*k-nebo-K*⟩␣{⟨*CMYK-barvy*⟩}.

opmac.tex

```
\def\Xpdfcolork#1{\def\pdflastcolork{#1}}
\def\XpdfcolorK#1{\def\pdflastcolorK{#1}}
\let\pdflastcolork=\pdfblackcolor \let\pdflastcolorK=\pdfblackcolor

\def\Xpage#1{\lastpage=#1 \fnotenumlocal=0
   \ifx\pdflastcolork\pdfblackcolor\else
      \isdefined{pgc:#1}\iftrue \else \sxdef{pgc:#1}{}\fi
      {\let\setpgcolor=\relax \sxdef{pgc:#1}%
            {\csname pgc:#1\endcsname\setpgcolor k{\pdflastcolork}}}\fi
   \ifx\pdflastcolorK\pdfblackcolor\else
      \isdefined{pgc:#1}\iftrue \else \sxdef{pgc:#1}{}\fi
      {\let\setpgcolor=\relax \sxdef{pgc:#1}%
            {\csname pgc:#1\endcsname\setpgcolor K{\pdflastcolorK}}}\fi
}
\def\setpgcolor#1#2{\special{PDF:#2 #1}}
```

\begoutput: 31, 50 \endoutput: 31, 50 \Xpdfcolork: 29, 31 \XpdfcolorK: 29, 31
\pdflastcolork: 31–32 \pdflastcolorK: 31–32 \Xpage: 31, 44–45

Makro \preboxcclv jednoduše spustí \pgc:⟨číslo-strany⟩. Není-li \pgc:⟨číslo-strany⟩ definováno, je to známka, že barva na začátku je černá a díky dvojici \csname, \endcsname se provede \relax, tedy nic. Jinak se nastaví správná barva pomocí \setpgcolor.

opmac.tex
```
\def\preboxcclv{\csname pgc:\the\pageno\endcsname}
```

Makro \postboxcclv nastaví zpět černou barvu. Dělá to inteligentně. Nejprve nastaví správné hodnoty makrům \currcolork a \currcolorK po vložení boxu 255. K tomu účelu předefinuje \setpgcolor tak, aby pouze ukládal tyto hodnoty a spustí \pgc:⟨číslo-strany+1⟩. Poté příkazy \Black a \linecolor\Black budou vědět, jaká je aktuální barva. A je-li černá, neudělají nic, jinak vloží příslušný \special.

opmac.tex
```
\def\postboxcclv{%
   \def\setpgcolor##1##2{\expandafter\xdef\csname currcolor##1\endcsname{##2}}%
   \ifnum\pageno<\lastpage {\advance\pageno by1 \csname pgc:\the\pageno\endcsname}%
   \else \xdef\currcolork{\pdflastcolork}\xdef\currcolorK{\pdflastcolorK}\fi
   \Black \linecolor\Black
}
```

Není-li použit pdfTEX, některá makra pro barvu deaktivujeme:

opmac.tex
```
\ifpdftex\else
  \def\setcmykcolor#1{}
  \let\localcolor=\relax
\fi
```

Makro \draft vloží do \headline box nulové výšky a šířky \draftbox, který vystrčí svou šedou sazbu ven ze svého rozměru a je tištěn dřív, než jakýkoli jiný materiál na stránce.

opmac.tex
```
\def\draft{\edef\tmp{\headline={\noexpand\draftbox{\tenbf DRAFT}\the\headline}}
   \ifpdftex \tmp \else
      \opwarning{\string\draft: Grey color is possible in pdfTeX only}%
   \fi}
```

V makru \draftbox {⟨text⟩} je ⟨text⟩ otočen o 55 stupňů, zvětšen desetkrát a vytištěn v barvě \LightGrey. K tomu jsou využity PDF transformace souřadnic.

opmac.tex
```
\def\draftbox#1{\vbox to0pt{\setbox0=\hbox{\typosize[10/]#1}%
   \kern.5\vsize \kern4\wd0 \hbox to0pt{\kern.5\hsize \kern-2.5\wd0
   \pdfsave \pdfrotate{55}\pdfscale{10}{10}%
   \hbox to0pt{\localcolor\LightGrey \box0\hss}%
   \pdfrestore
\hss}\vss}\hss}
```

3.16 Klikací odkazy

Makro \destactive [⟨typ⟩:⟨lejblík⟩] založí cíl odkazu jen tehdy, když je ⟨lejblík⟩ neprázdný. Ve vertikálním módu se nalepí na předchozí box díky \prevdepth=-1000pt a po vložení boxu s cílem vrátí hodnotu \prevdepth do původního stavu, aby následující box byl správně řádkován. V horizontálním módu prostě vloží \destbox. Makro \destbox [⟨typ⟩:⟨lejblík⟩] vytvoří box nulové výšky a z něj vystrčí nahoru cíl klikacího odkazu vzdálený od účaří o \destheight. Interně použije pdfTEXový primitiv \pdfdest s parametrem xyz, což charakterizuje obvyklou možnost chování PDF prohlížeče při odskoku na cíl. Podrobněji viz manuál k pdfTEXu. PDF prohlížeče většinou lícují horní hranu okna přesně s místem cíle, je tedy potřeba cíl umístit poněkud výše, abychom viděli i odkazovaný text. K tomu právě slouží registr \destheight.

opmac.tex
```
\newdimen\destheight \destheight=1.3em
\def\destactive[#1:#2]{\if$#2$\else\ifvmode
      \tmpdim=\prevdepth \prevdepth=-1000pt
      \destbox[#1:#2]\prevdepth=\tmpdim
   \else \destbox[#1:#2]%
   \fi\fi
}
```

\preboxcclv: 31–32, 50–51 \setpgcolor: 31–32 \postboxcclv: 32, 50–51 \draft: 32
\draftbox: 32 \destactive: 32–33 \destbox: 32–33 \destheight: 14, 16, 32–33, 50

```
\def\destbox[#1]{\vbox to0pt{\kern-\destheight \pdfdest name{#1} xyz\vss}}
\def\dest[#1]{}
```

V uživatelské dokumentaci je zmíněno místo \destactive makro \dest se stejnými parametry. Toto makro je implicitně prázdné a tedy nečiné. Teprve \hyperlinks je přinutí k činnosti.

Někdy je účelné v režimu „draft“ dokumentu tisknout v místě cílů odkazů jména lejblíků, aby autor viděl, jaké lejblíky použil a lépe se mu dílo modifikovalo. Stačí předefinovat pro tento režim makro \destbox třeba takto:

```
\def\destbox[#1#2:#3]{\vbox to0pt{\kern-\destheight
   \pdfdest name{#1#2:#3} xyz\relax
   \if#1r\llap{\localcolor\Green\ttt[#3]}\vss
   \else \if#1c\vss\llap{\localcolor\Green\ttt[\tmpb] }\kern-\prevdepth
   \else \vss \fi\fi}}
```

Při tomto řešení budou lejblíky z \label tištěny nahoru v místě cíle zatímco lejblíky z \bib a \bibitem budou tištěny vedle položky se seznamem literatury. V obou případech budou lejblíky zelené a díky \llap neovlivní polohu ostatní sazby.

Klikací text vytvoří makro \link [⟨*typ*⟩:⟨*lejblík*⟩]{⟨*barva*⟩}{⟨*text*⟩}. Makro používá pdfTEXový primitiv \pdfstartlink, ve kterém je vymezena výška a hloubka aktivní plochy. Nakonec přepne na požadovanou ⟨*barvu*⟩ (pokud není černá), vytiskne aktivní ⟨*text*⟩ a přepne zpět na černou barvu. PdfTEXový primitiv \pdfendlink ukončí sazbu aktivního textu.

opmac.tex

```
\def\link[#1:#2]#3#4{\leavevmode\pdfstartlink height.9em depth.3em
      \pdfborder{#1} goto name{#1:#2}\relax {#3#4}\pdfendlink
}
```

Makro \urllink [⟨*typ*⟩:⟨*lejblík*⟩]{⟨*text*⟩} pracuje analogicky jako \link. Jen navíc přidává některé atributy do PDF výstupu a pracuje s barvou \urlcolor. Toto makro vytvoří externí odkaz. Je použito v makru \url prostřednictvím makra \ulink.

opmac.tex

```
\def\urllink[#1:#2]#3{{\let~=\relax \let\\=\relax \let\{=\relax \let\}=\relax
   \leavevmode\pdfstartlink height.9em depth.3em
   \pdfborder{#1}user{/Subtype/Link/A <</Type/Action/S/URI/URI(#2)>>}\relax
   {\def~{\nobreak\space}\urlcolor#3}\pdfendlink}%
}
```

Makra \toclink, \pglink, \citelink, \reflink, \ulink, která se specializují na určitý typ linku, implicitně nedělají nic:

opmac.tex

```
\def\toclink#1{\toclinkA{#1}}
\def\pglink#1{#1}
\def\citelink#1{#1}
\def\reflink[#1]#2{#2}
\def\ulink[#1]#2{{#2}}
\def\urlcolor{}
```

Ovšem po použití makra \hyperlinks {⟨*barva-lok*⟩}{⟨*barva-url*⟩} se uvedená makra \toclink, \pglink, \citelink a \reflink probouzejí k životu. Zde je také definováno makro \urlcolor.

opmac.tex

```
\def\hyperlinks#1#2{%
   \let\dest=\destactive
   \def\toclink##1{\link[toc:##1]{\localcolor#1}{\toclinkA{##1}}}%
   \def\pglink##1{\link[pg:##1]{\localcolor#1}{##1}}%
   \def\citelink##1{\link[cite:##1]{\localcolor#1}{##1}}%
   \def\reflink[##1]##2{\link[ref:##1]{\localcolor#1}{##2}}%
   \def\ulink[##1]##2{\urllink[##1]{##2}}%
   \def\urlcolor{\longlocalcolor#2}%
}
```

PdfTEXové primitivy pro klikací odkazy dovolují dopravit do PDF další atributy odkazu za slovem `attr`. Tam je možné dát najevo, že chceme vidět aktivní plochy ve formě rámečků. To zařídí makro

\dest: 11, 15, 33, 47, 49–50 \link: 33–34 \urllink: 33–34 \toclink: 18, 33
\pglink: 12, 18, 33 \citelink: 33, 46 \reflink: 12, 33 \ulink: 33–34 \hyperlinks: 33–34
\urlcolor: 33

\pdfborder {⟨typ⟩}, které expanduje na nic, pokud není kontrolní sekvence \⟨typ⟩border definována. Jinak expandují na arrt /C s obsahem podle \⟨typ⟩border.

opmac.tex
```
\def\pdfborder#1{\ifˆˆ#1ˆˆ\else \isdefined{#1border}\iftrue
   \ifˆˆ\csname#1border\endcsnameˆˆ\else
      attr{/C[\csname#1border\endcsname] /Border[0 0 1]}%
   \fi\fi\fi
}
```

Pokud je dokument zpracován do DVI výstupu, je vhodné výše zmíněná makra deaktivovat:

opmac.tex
```
\ifpdftex \else
  \def\link[#1]#2#3{#3}
  \def\urllink[#1]#2{#2}
  \def\hyperlinks#1#2{}
\fi
```

Makro \url {⟨text⟩} se používá k tisku URL. Vytiskne ⟨text⟩ fontem \urlfont, přitom kolem znaků lomítko, tečka a dalších přidává nulovou mezeru s dodatečnou mírnou roztažitelností \urlskip. Mezera vpravo od těchto znaků je navíc zlomitelná s penaltou definovanou v makru \urlbskip. Dvojité lomítko \urlslashslash má zlomitelnou mezeru jen na konci.

opmac.tex
```
\def\url#1{{\def\tmpb{#1}%
   \replacestrings{//}{\urlskip\urlslashslash\urlbskip}%
   \replacestrings{/}{\urlskip/\urlbskip}%
   \replacestrings{.}{\urlskip.\urlbskip}%
   \replacestrings{?}{\urlskip?\urlbskip}%
   \replacestrings{=}{\urlskip=\urlbskip}%
   \replacestrings{~}{\char‘\~ }%
   \replacestrings{_}{\char‘\_ }%
   \replacestrings{^}{\char‘\^ }%
   \replacestrings{\\}{\bslash}%
   \replacestrings{\{}{\char‘\{ }%
   \replacestrings{\}}{\char‘\} }%
   \replacestrings{&}{\urlbskip\char‘\& \urlskip}%
   \ulink[url:#1]{\urlfont\tmpb}%
}}
\def\urlfont{\tt}
\def\urlskip{\null\nobreak\hskip0pt plus0.05em\relax}
\def\urlbskip{\penalty100 \hskip0pt plus0.05em\relax}
\def\urlslashslash{/\urlskip/}
\addprotect\url
```

Je třeba vysvětlit, proč je v makru \urlskip použito \null, neboli \hbox{}. Tento box se přilepí na předchozí slovo a tím zakážeme toto slovo dělit podle vzorů dělení slov. Spojovník při rozděleném slovu je totiž pro čtenáře matoucí: nemůže vědět, zda je nebo není součástí URL.

Makro \url{⟨text⟩} pracuje tak, že uloží ⟨text⟩ do \tmpb a nechá vyměnit příslušné znaky uvnitř \tmpb pomocí \replacestrings. Nakonec vytiskne ⟨text⟩ prostřednictvím \ulink.

Aktivní vlnku lze v ⟨textu⟩ vyměnit za \char‘\~. Podobně lze řešit některé další znaky, ale ne všechny: procento, backlash. U těchto znaků bychom nejprve museli vyměnit jejich kategorie. Pak by ale makro \url nefungovalo uvnitř parametrů jiných maker. V zájmu jednoduchosti makra \url to neděláme. Takže pokud uživatel má v URL znak procento, nahradí ho sekvencí \percent manuálně a pokud tam má další speciální znak, vyřeší to podobným makrem jako \percent.

Makro \replacestrings {⟨string1⟩}{⟨string2⟩} vymění v makru \tmpb veškeré výskyty ⟨string1⟩ za ⟨string2⟩. Pro tento účel definuje pracovní makro \tmp, které pracuje podobně, jako makro \iiscanch (doporučujeme se podívat na výklad makra \iiscanch). Makro \tmp ale není na rozdíl od \iiscanch expandující. Místo toho postupně kumuluje výsledek do nového \tmpb pomocí \addto. Před spuštěním \tmp expandujeme \tmpb pomocí pěti \expandafter a poté ho pronulujeme, takže to pracuje jako \def\tmbb{}\tmp⟨původní-obsah-tmpb⟩\/⟨string1⟩\relax.

\pdfborder: 33–34 \url: 33–34, 48 \urlfont: 34 \urlskip: 34 \urlbskip: 34
\urlslashslash: 34 \replacestrings: 34–35

opmac.tex
```
\def\replacestrings#1#2{%
   \def\tmp##1#1##2\relax{\if$##2$\addto\tmpb{##1}\else\addto\tmpb{##1#2}\tmp##2\relax\fi}%
   \expandafter\def\expandafter\tmpb\expandafter{\expandafter}%
   \expandafter\tmp\tmpb\/#1\relax
   \def\tmp##1\/{\def\tmpb{##1}}\expandafter\tmp\tmpb
}
```

Na konci ⟨*textu*⟩ je vložena sekvence `\/`, která jej odděluje od přidaného ⟨*string1*⟩. Kdybychom ji tam nedali, pak při ⟨*string1*⟩=// a při lomítku na konci ⟨*textu*⟩ bychom měli `...///` a to způsobí potíže. V závěru makra `\replacestring` je přidaná sekvence `\/` zase odstraněna.

3.17 Outlines – obsah v záložce PDF dokumentu

Hlavní problém implementace strukturovaného obsahu do záložky PDF dokumentu spočívá v tom, že při vkládání jednotlivých položek obsahu je nutno znát počet přímých potomků každé položky (v rámci stromové struktury položek), ovšem tito přímí potomci budou zařazeni později. OPmac tento problém řeší dvěma průchody nad daty, které jsou vytvořeny pro tisk obsahu, tj. v makru `\toclist`. V prvním průchodu spočítá potřebné potomky a ve druhém průchodu zařadí všechny položky postupně jako „outlines“ do záložky. Připomeneme si, že v `\toclist` se nachází seznam maker tvaru `\tocline{`⟨*odsazení*⟩`}{`⟨*font*⟩`}{`⟨*číslo*⟩`}{`⟨*text*⟩`}{`⟨*strana*⟩`}`. Makro `\outlines {`⟨*úroveň*⟩`}` nejprve nastaví `\tocline` na hodnotu `\outlinesA` a projde `\toclist`. Pak je nastaví na hodnotu `\outlinesB` a znovu projde `\toclist`.

opmac.tex
```
\def\outlines#1{\pdfcatalog{/PageMode/UseOutlines}\openref\ifx\toclist\empty
     \opwarning{\noexpand\outlines -- data unavailable. TeX me again}%
   \else
     {\let\tocline=\outlinesA
      \count0=0 \count1=0 \toclist % calculate numbers o childs
      \def\outlinelevel{#1}\let\tocline=\outlinesB
      \count0=0 \count1=0 \toclist}% create outlines
   \fi
}
```

V makru `\outlinesA {`⟨*odsazení*⟩`}{`⟨*font*⟩`}{`⟨*číslo*⟩`}{`⟨*text*⟩`}{`⟨*strana*⟩`}` počítáme potomky. Makro je navrženo tak, aby bylo snadno rozšiřitelné na libovolnou úroveň hloubky stromu, nicméně pro potřeby OPmac stačí hloubka tři (kapitoly, sekce, podsekce). Úroveň uzlu přečteme v parametru ⟨*odsazení*⟩. Pro kapitolu je ⟨*odsazení*⟩=0, pro sekci je ⟨*odsazení*⟩=1 a pro podsekci je ⟨*odsazení*⟩=2. Představme si vedle sebe řadu counterů `\count0:\count1:\count2`. Při sekvenčním čtení jednotlivých uzlů stromu si každý uzel zvětší v této pomyslné řadě hodnotu svého counteru o jedničku. Kapitoly zvětšují `\count0`, sekce `\count1`, podsekce `\count2`. Stačí tedy zvětšit `\count`⟨*odsazení*⟩. Řada counterů pak jednoznačně určuje zpracovávaný uzel. Uzly pro kapitoly mají přidělenu kontrolní sekvenci `ol:\the\count0` a uzly pro sekce mají přidělenu kontrolní sekvenci `ol:\the\count0:\the\count1`. Jsou to makra, jejichž obsahem je počet potomků daného uzlu. Makrem `\addoneol` ⟨*csname*⟩ zvětšíme obsah dané kontrolní sekvence o jedničku. Příkazem `\ifcase`⟨*odsazení*⟩ řešíme, kterému rodiči je třeba zvednout tuto hodnotu. Při nule (kapitola) nikomu, neboť daný uzel nemá rodiče. Při ⟨*odsazeni*⟩=1 zvětšíme o jedničku počet potomků nadřazené kapitole a při ⟨*odsazeni*⟩=2 nadřazené sekci. Asi by bylo přehlednější na začátku definovat všechny potřebné sekvence `ol:`⟨*něco*⟩ a nastavit jim hodnotu 0. Ovšem šetříme pamětí i časem, takže zakládáme sekvenci `ol:`⟨*něco*⟩ teprve v makru `\addoneol` a to tehdy, když je ji poprvé potřeba zvětšit o jedničku.

opmac.tex
```
\def\outlinesA#1#2#3#4#5{%
   \advance\count#1 by1
   \ifcase#1\or
     \addoneol{ol:\the\count0}\or
     \addoneol{ol:\the\count0:\the\count1}\fi
}
\def\addoneol#1{\isdefined{#1}%
   \iftrue \tmpnum=\csname#1\endcsname\relax
           \advance\tmpnum by1 \sxdef{#1}{\the\tmpnum}%
   \else \sxdef{#1}{1}%
   \fi
```

\outlines: 35–37 \outlinesA: 35 \addoneol: 35

```
1011: }
```

V makru \outlinesB {⟨odsazení⟩}{⟨font⟩}{⟨číslo⟩}{⟨text⟩}{⟨strana⟩} vkládáme jednotlivou položku obsahu do záložek pomocí pdfTEXového primitivu \pdfoutline␣goto␣name{⟨lejblík⟩}␣count⟨potomci⟩␣{⟨text⟩}. Číslo ⟨potomci⟩ je opatřeno znaménkem mínus právě tehdy, když chceme, aby položka ve výchozím stavu nezobrazovala své potomky, ale jen trojúhelníček. Potomci se zobrazí až po kliknutí na trojúhelníček. V makru \outlinelevel máme makrem \outlines připravenu úroveň rozevření, kterou si uživatel přeje. Nejprve přičtením \count⟨odsazení⟩ dostaneme řadu \count0:\count1:\count2 do stejného stavu, jako v předchozím prvním průchodu a máme tím jednoznačně přidělen uzel stromu. Do \tmpnum vložíme údaj o počtu potomků daného uzlu. K tomu je potřeba rozvětvit výpočet příkazem \ifcase, protože pro různou úroveň uzlu máme údaj v různě definovaném makru. Příkazem \protectlist zastavíme expanze případných maker registrovaných pomocí \addprotect a definujeme vlnku jako mezeru (v záložce vypadá líp než vlnka). Dále pomocí \setcnvcodesA expandujeme \toasciidata. Pomocí \setlccodes\toasciidata připravíme \lccode znaků tak, aby \lowercase odstranil háčky a čárky. To vzápětí provedeme, ale nejprve ještě do toho může promluvit uživatel v makru \cnvhook, které je implicitně nastaveno na makro prázdné.

opmac.tex

```
1012: \def\outlinesB#1#2#3#4#5{%
1013:    \advance\count#1 by1
1014:    \ifcase#1\tmpnum=\isdefined{ol:\the\count0}%
1015:                \iftrue\csname ol:\the\count0\endcsname\else0\fi \or
1016:             \tmpnum=\isdefined{ol:\the\count0:\the\count1}%
1017:                \iftrue\csname ol:\the\count0:\the\count1\endcsname\else0\fi \or
1018:             \tmpnum = 0 \fi
1019:   \protectlist \def~{ }\setcnvcodesA
1020:   \expandafter \setlccodes \toasciidata{}{}%
1021:   \cnvhook \lowercase{\gdef\tmp{#4}}%
1022:   \pdfoutline goto name{toc:#3} count
1023:      \ifnum#1<\outlinelevel\space\else-\fi\tmpnum {\tmp}\relax
1024: }
```

Makro \setcnvcodesA zkontroluje podle definovanosti \r, zda je zapnutý \csaccents a pokud je, expanduje \toasciidata. Makro \toasciidata potřebujeme expandovat, protože neobsahuje přímý zápis znaků. Důvod je zřejmý, nechceme, aby se soubor opmac.tex stal závislý na použitém kódování.

opmac.tex

```
1025: \def\setcnvcodesA{\global\let\setcnvcodesA=\relax % I am working only once
1026:   \ifx\r\undefined
1027:      \gdef\toasciidata{}
1028:      \opwarning{\noexpand\csaccents unused, CZ/SK outline-conversion is off}%
1029:   \else
1030:      \xdef\toasciidata{\toasciidata}%
1031:   \fi
1032: }
1033: \def\toasciidata{%  Removes Czech+Slovak accents
1034:        AA\'AA\"AA\'aa\"aaBBCC\v CC\v ccDD\v DD\v ddEE\'EE\v EE\'ee\v ee%
1035:        FFGGHHII\'II\'iiJJKKLL\'LL\v LL\'ll\v llMMNN\v NN\v nnOO\'OO\"OO\^OO%
1036:        \'oo\"oo\^ooPPQQRR\v RR\v rrSS\v SS\v ssTT\v TT\v ttUU\'UU\"UU\r UU%
1037:        \'uu\"uu\r  uuVVWWXXYY\'YY\'yyZZ\v ZZ\v zz%
1038: }
```

Na řádku 1020 se makro \setlccodes spustí jako \setlccodes␣AAÄAÄÄáa...{}{}. Toto makro si odloupne dva parametry xy, provede \lccode‘x=‘y a v rekurzivním cyklu pokračuje v činnosti, dokud nenarazí na {}{}.

opmac.tex

```
1039: \def\setlccodes#1#2{\if\relax#2\relax \else \lccode‘#1=‘#2 \expandafter \setlccodes \fi}
```

Makro \insertoutline {⟨text⟩} vloží jedinou položku do záložky. Pro tuto položku se předpokládá nulový počet potomků. Využití: uživatel může takto odkázat na začátek nebo konec dokumentu.

opmac.tex

```
1041: \def\insertoutline#1{\pdfdest name{oul:#1} xyz\relax
1042:    \pdfoutline goto name{oul:#1} count0 {#1}\relax
1043: }
```

\outlinesB: 35–36 \outlinelevel: 35–36 \setcnvcodesA: 36 \toasciidata: 36
\setlccodes: 25, 36 \insertoutline: 36–37

Pokud je dokument zpracován do DVI výstupu, je vhodné výše zmíněná makra deaktivovat:

opmac.tex

```
\ifpdftex \else
  \def\outlines#1{\opwarning{DVI output has no outlines}\gdef\outlines##1{}}
  \let\insertoutline=\outlines
\fi
```

3.18 Verbatim

Verbatim výpisy budou odsazeny o \ttindent. Je nastaven na hodnotu \parindent v době čtení souboru a společně s \parindent by měl uživatel změnit i \ttindent. Čítač \ttline čísluje řádky běžného verbatim výstupu, čítač \viline čísluje řádky souboru čteného pomoci \verbinput. Souborový deskriptor \vifile bude přiřazen souboru v makru \verbinput.

opmac.tex

```
\newcount\ttline    \ttline=-1
\newcount\viline
\newread\vifile
```

Makra \setverb, \begtt ...\endtt jsou dokumentována v TBN, str. 29.

opmac.tex

```
\def\setverb{\frenchspacing\def\do##1{\catcode`##1=12}\dospecials \catcode`\*=12 }
\def\begtt{\par\ttskip\bgroup \wipeepar
   \setverb \adef{ }{ }%
   \ifx\savedttchar\undefined \else \catcode\savedttchar=12 \fi
   \parindent=\ttindent
   \tthook\relax
   \ifnum\ttline<0 \else
     \tenrm \thefontscale[700]\let\sevenrm=\thefont
     \everypar={\global\advance\ttline by1
                \llap{\sevenrm\the\ttline\kern.9em}}\fi
   \def\par##1{\endgraf\ifx##1\egroup\else\penalty\ttpenalty\leavevmode\fi ##1}
   \obeylines \startverb}
{\catcode`\|=0 \catcode`\\=12
|gdef|startverb#1\endtt{|tt#1|egroup|par|ttskip|testparA}}
```

Makro \begtt očichá na konci své činnosti, zda se nachází pod \endtt prázdný řádek (alias \par). K tomu slouží makra \testparA (přeskočí mezeru, která za \endtt vždy je), \testparB (přečte následující znak pomocí \futurelet) a \testparC (ošetří, zda tento následující znak je \par).

opmac.tex

```
\def\testparA{\afterassignment\testparB\let\tmpa= }
\def\testparB{\futurelet\tmpa\testparC}
\def\testparC{\ifx\tmpa\par\else\afternoindent\fi}
```

Makro \activettchar pracuje podobně, jako makro \adef. Navíc potřebuje použít nově načtený znak ve své aktivní kategorii jako separátor vymezující konec parametru.

opmac.tex

```
\def\activettchar#1{%
   \ifx\savedttchar\undefined\else \catcode\savedttchar=\savedttcharc \fi
   \chardef\savedttchar=`#1%
   \chardef\savedttcharc=\catcode`#1%
   \lccode`\~=`#1
   \lowercase {\def~}{\leavevmode\hbox\bgroup\setverb\adef{ }{ }%
                      \intthook\tt\readverb}%
   \lowercase{\def\readverb ##1~}{##1\egroup}%
   \lccode`\~=0 \catcode\savedttchar=13
}
```

Makro \verbinput si pomocí \tmpa ověří, zda minule byl čten stejný soubor. Pokud ne, otevře soubor #2 ke čtení pomocí \openin a uloží do \vifilename jméno naposledy otevřeného souboru. Dále zkontroluje pomocí \ifeof, zda je možné ze souboru číst. Pokud ne, vypíše se varování a pomocí \skiptorelax se přeskočí zbytek obsahu makra až po \relax, takže se neprovede nic dalšího. Je-li soubor úspěšně otevřen nebo byl-li otevřen již minule, pustí se makro \verbinput do prozkoumání parametru #1 zapsaného v závorce před jménem souboru.

\ttline: 37, 39 \viline: 37–39 \vifile: 37–39 \setverb: 37–39 \begtt: 5, 37
\testparA: 37 \testparB: 37, 39 \testparC: 37 \activettchar: 37–38 \verbinput: 5, 37–38
\vifilename: 38–39 \skiptorelax: 38, 46

opmac.tex
```
\def\verbinput (#1) #2 {\par \def\tmpa{#2}%
   \ifx\vifilename\tmpa \else
      \openin\vifile=#2
      \global\viline=0 \global\let\vifilename=\tmpa
      \ifeof\vifile
         \opwarning{\noexpand\verbinput - file "#2" is unable to reading}
         \expandafter\expandafter\expandafter\skiptorelax
      \fi
   \fi
   \viscanparameter #1+\relax
}
\def\skiptorelax#1\relax{}
```

Cílem vyhodnocení parametru v závorce makra \verbinput jsou dva údaje: \vinolines bude obsahovat počet řádků, které je od začátku souboru nutno přeskočit, než se má zahájit přepisování řádků a \vidolines bude obsahovat počet řádků, které se mají přepsat ze souboru do dokumentu. Písmena vi na začátku těchto názvů představují zkratku pro verbinput. Vyšetření parametru ukončeného textem +\relax se v makru \viscanparameter větví na případ, kdy parametr obsahuje symbol + a použije se pak \viscanplus. Druhý případ, kdy uživatel nenapsal symbol plus (takže parametr #2 makra \viscanparameter je prázdný) je dále vyšetřen v makru \viscanminus. Obě makra si oddělí do svých parametrů první a druhou číslici (každá z nich může být prázdná) a nastaví podle zdokumentovaných pravidel pro zápis parametru odpovídající interní údaje \vinolines a \vidolines. Vychází přitom z předpokladu, že registr \viline obsahuje číslo naposledy přečteného řádku (nebo nulu, jsme-li na začátku souboru).

opmac.tex
```
\def \viscanparameter #1+#2\relax{%
   \if$#2$\viscanminus(#1)\else \viscanplus(#1+#2)\fi
}
\def\viscanplus(#1+#2+){%
   \if$#1$\tmpnum=\viline
   \else \ifnum#1<0 \tmpnum=\viline \advance\tmpnum by-#1
       \else \tmpnum=#1
             \advance\tmpnum by-1
             \ifnum\tmpnum<0 \tmpnum=0 \fi % (0+13) = (1+13)
   \fi \fi
   \edef\vinolines{\the\tmpnum}%
   \if$#2$\def\vidolines{0}\else\edef\vidolines{#2}\fi
   \doverbinput
}
\def\viscanminus(#1-#2){%
   \if$#1$\tmpnum=0
      \else \tmpnum=#1 \advance\tmpnum by-1 \fi
   \ifnum\tmpnum<0 \tmpnum=0 \fi  % (0-13) = (1-13)
   \edef\vinolines{\the\tmpnum}%
   \if$#2$\tmpnum=0
      \else \tmpnum=#2 \advance\tmpnum by-\vinolines \fi
   \edef\vidolines{\the\tmpnum}%
   \doverbinput
}
```

Makro \doverbinput provede samotnou práci: přeskočí \vinolines řádků a přepíše \vidolines řádků. To provede v prvním a druhém cyklu \loop. Než se k těmto cyklům dostane, musí udělat jisté přípravné práce. Nejprve odečte od \vinolines počet už přečtených řádků, protože při opakovaném čtení stejného souboru jej neotevíráme znova, jen přeskočíme příslušný menší počet řádků. Pokud ale se ukáže, že rozdíl je záporný (je potřeba se v souboru vracet dozadu), makro znovuotevře soubor ke čtení pomocí \openin a upraví podle toho příslušné údaje o řádcích. Pak zahájí skupinu, dále pomocí \setverb nastaví speciálním znakům kategorii 12 a pomocí \adef{␣}{␣} nastaví mezeře aktivní kategorii (bude expandovat na neaktivní mezeru jako \space) a také nastaví kategorii 12 znaku, který byl deklarován pomocí \activettchar. Připraví odsazení podle \ttindent a spustí uživatelský \tthook. Je-li potřeba tisknout čísla řádků, připraví si na to font \sevenrm, který má velikost rovnu 0,7 násobku základní velikosti. A pustí se do zmíněných dvou cyklů \loop. V obou cyklech se může stát, že narazíme nečekaně na

\vinolines: 38–39 \vidolines: 38–39 \viscanparameter: 38 \viscanplus: 38
\viscanminus: 38 \doverbinput: 38–39

konec souboru. To je ošetřeno testem \ifeof\vifile a následnou úpravou čítače \tmpnum tak, abychom okamžitě vyskočili z cyklu. Druhý cyklus obsahuje ještě jeden speciální rys: přeje-li si uživatel číst až do konce souboru, je nastaveno \vidolines na nulu a před zahájením cyklu je čítač \tmpnum nastaven na -1. Uvnitř cyklu je pak zajištěno, že v tomto případě není čítač zvětšován o jedničku. Po ukončení práce v těchto dvou cyklech je ukončena skupina, vložena mezera \ttskip a makrem \testparB se ověří, zda následuje prázdný řádek.

opmac.tex

```
\def\doverbinput{%
   \tmpnum=\vinolines
   \advance\tmpnum by-\viline
   \ifnum\tmpnum<0
      \openin\vifile=\vifilename\space
      \global\viline=0
   \else
      \edef\vinolines{\the\tmpnum}%
   \fi
   \par\ttskip\bgroup \wipeepar
   \setverb \adef{ }{ }%
   \ifx\savedttchar\undefined \else \catcode\savedttchar=12 \fi
   \parindent=\ttindent
   \tthook\relax
   \ifnum\ttline<-1 \else
      \tenrm \thefontscale[700]\let\sevenrm=\thefont \fi
   \tmpnum=0 \tt
   \loop \ifeof\vifile \tmpnum=\vinolines\space \fi
         \ifnum\tmpnum<\vinolines\space
         \vireadline \advance\tmpnum by1 \repeat      %% skip line
   \tmpnum=0 \ifnum\vidolines=0 \tmpnum=-1 \fi
   \loop \ifeof\vifile \tmpnum=\vidolines\space \fi
         \ifnum\tmpnum<\vidolines\space
         \vireadline \penalty\ttpenalty \viprintline  %% print line
             \ifnum\vidolines=0 \else\advance\tmpnum by1 \fi \repeat
   \egroup\par\ttskip\testparB
}
```

V prvním cyklu \loop v těle makra \doverbinput se opakovaně volá \vireadline, což je makro, které přečte další řádek ze souboru. V druhém cyklu se opakovaně volá \vireadline následované \viprintline. Toto makro vytiskne přečtený řádek do dokumentu. Před řádkem může být v \llap vytištěno číslo řádku. Záleží na hodnotě \ttline. Je to naprogramováno v souladu s uživatelskou dokumentací.

opmac.tex

```
\def\vireadline{\read\vifile to \tmp \global\advance\viline by1 }
\def\viprintline{\indent
   \ifnum \ttline<-1 \else
      \llap{\sevenrm\ifnum\ttline<0 \the\viline \else
               \global\advance\ttline by1 \the\ttline \fi \kern.9em}%
   \fi
   \tmp\par % print the line from \tmp
}
```

3.19 Jednoduchá tabulka

Tabulku makrem \table vytvoříme jako \vbox, ve kterém je \halign. Je tedy potřeba načíst deklaraci typu {llc|rr} a převést ji na deklaraci pro \halign. Tato deklarace obsahuje znak # a tento znak se obtížně přidává do těla maker. Nashromáždíme tedy postupně deklaraci pro \halign do registru typu \toks, který je nazvaný \tabdata. Dále definujeme interní \tabstrutA, který bude obsahovat uživatelův \tabstrut, ovšem přechodně budeme toto makro měnit. Také deklarujeme čítač \colnum, ve kterém budeme mít po přečtení deklarace uložen počet sloupců tabulky. Dále během skenování ⟨*deklarace*⟩ vytvoříme makro \ddlinedata, které bude obsahovat &\dditem␣&\dditem␣... (počet těchto dvojic bude roven $n-1$, kde n je počet sloupců). Pokud je v deklaraci dvojitá svislá čára, bude v makru \ddlinedata na příslušném místě ještě \vvitem. Makro \ddlinedata pak použijeme v \crli a v \tskip, Strýček

\vireadline: 39 \viprintline: 39 \tabdata: 40 \tabstrutA: 40–41 \colnum: 40
\ddlinedata: 39–41

Příhoda to může použít jinde a jinak. Konečně makro \vvleft je neprázdné, pokud úplně vlevo tabulky je dvojitá čára.

opmac.tex
```
\newtoks\tabdata
\def\tabstrutA{\tabstrut}
\newcount\colnum  \colnum=0
\def\ddlinedata{}
\def\vvleft{}
```

Makro \table {⟨deklarace⟩}{⟨data⟩} vypadá takto:

opmac.tex
```
\def\table{\vbox\bgroup \catcode`\|=12 \tableA}
\def\tableA#1#2{\offinterlineskip \def\tmpa{}\tabdata={}\scantabdata#1\relax
   \halign\expandafter{\the\tabdata\tabstrutA\cr#2\crcr}\egroup}
```

Makro \scantabdata postupně čte znak po znaku z deklarace \table a podle přečteného znaku ukládá do \tabdata odpovídající úsek skutečné deklarace pro \halign. Volá přitom \addtabvrule nebo \addtabitem{\tabdeclare⟨znak⟩}.

opmac.tex
```
\def\scantabdata#1{\let\next=\scantabdata
   \ifx#1\relax\let\next=\relax
   \else\if#1|\addtabvrule
   \else\expandafter\ifx\csname tabdeclare#1\endcsname \relax
     \opwarning{tab-declare letter #1 unknown, ignored}%
        \else\expandafter \addtabitem\expandafter{\csname tabdeclare#1\endcsname}%
   \fi\fi\fi \next
}
```

OPmac předdefinuje tři ⟨znaky⟩ pro ⟨deklaraci⟩, sice ⟨znaky⟩ c, l, r v makrech \tabdeclarec, \tabdeclarel, \tabdeclarer.

opmac.tex
```
\def\tabdeclarec{\tabiteml\hfil##\unsskip\hfil\tabitemr}
\def\tabdeclarel{\tabiteml##\unsskip\hfil\tabitemr}
\def\tabdeclarer{\tabiteml\hfil##\unsskip\tabitemr}
```

Makro \unsskip vkládané na konec každé datové položky odebere mezeru, pokud má nenulovou základní velikost. Uživatelé totiž někdy dávají kolem datových položek mezery a někdy ne, přitom chtějí, aby se jim to chovalo stejně. Je náročné si pamatovat, že mezery před položkou jsou ignorovány primitivem \halign, ale mezery za položkou jsou podstatné. Tak raději i mezery za položkou uděláme nepodstatné.

opmac.tex
```
\def\unsskip{\ifdim\lastskip>0pt \unskip\fi}
```

Příklad: po deklaraci: {|cr|l|} makro \scantabdata vytvoří:

```
tabdata: \vrule\tabiteml\hfil#\unsskip\hfil\tabitemr
        &\tabiteml\hfil#\unsskip\tabitemr \vrule\kern\vvkern\vrule
        &\tabiteml\hfil#\unsskip\hfil\tabitemr
        &\tabiteml#\unsskip\hfil\tabitemr\vrule
ddlinedata: &\dditem &\dditem\vvitem &\dditem &\dditem
```

Makra \addtabitem, \addtabdata a \addtabvrule vloží do \tabdata a \ddlinedata požadovaný údaj. Makro \addtabitem pozná podle \colnum=0, zda vkládá data pro první sloupec (nepřidává &) nebo pro další sloupce (přidává &). Makro \addtabvrule pozná podle \tmpa, zda před ním předchází další \vrule. Pokud ano, vloží dodatečnou mezeru \kern\vvkern a přidá \vvitem do \ddlinedata.

opmac.tex
```
\def\addtabitem#1{\ifnum\colnum>0 \addtabdata{&}\addto\ddlinedata{&\dditem}\fi
    \advance\colnum by1 \let\tmpa=\relax \expandafter\addtabdata\expandafter{#1}}
\def\addtabdata#1{\expandafter\tabdata\expandafter{\the\tabdata#1}}
\def\addtabvrule{\ifx\tmpa\vrule \addtabdata{\kern\vvkern}%
    \ifnum\colnum=0\def\vvleft{\vvitem}\else\addto\ddlinedata{\vvitem}\fi\fi
    \let\tmpa=\vrule \addtabdata{\vrule}}
```

\vvleft: 40–41 \table: 6, 39–40 \scantabdata: 40 \tabdeclarec: 40 \tabdeclarel: 40
\tabdeclarer: 40 \unsskip: 40 \addtabitem: 40 \addtabdata: 40 \addtabvrule: 40

Než se pustíme do výkladu dalších maker, předvedeme příklad, ve kterém je definováno další písmeno `P` pro ⟨*deklaraci*⟩. Písmeno `P` vymezí tabulkovou položku, jež má stanovenou šířku a delší text se láme do více řádků. Je možné si vyzkoušet třeba tento kód:

```
\newdimen\Pwidth
\def\tabdeclareP {\enskip\vtop{\hsize=\Pwidth \rightskip=0pt plus1fil
    \baselineskip=1.2em \lineskiplimit=0pt \noindent ##\tabstrut}\enskip}

\Pwidth=3cm \table{|c|P|}{\crl
   aaa & Tady je delší textík, který se nevejde na jeden řádek. \crl
   bb  & A tady je taky je něco delšího. \crl}
```

Pusťme se nyní do rozboru maker na ukončení řádků. Makro `\crl` přidá čáru pomocí `\noalign`. Makro `\crll` přidá dvojitou čáru pomocí `\noalign`.

opmac.tex

```
1192: \def\crl{\cr\noalign{\hrule}}
1193: \def\crll{\cr\noalign{\hrule\kern\hhkern\hrule}}
```

Makro `\crli` provede `\cr` a dále se vnoří do řádku tabulky, ve kterém klade postupně následující `\omit\tablinefil␣&\omit\tablinefil␣&...` Přitom v místě dvojité vertikální čáry naklade navíc `\tabvvline`. Makro `\tablinefil` vloží natahovací čáru na šířku celé položky a makro `\tabvvline` vloží dvě `\vrule` vzdáleny od sebe o `\vvkern`. Tím vzniká přetrzené místo v postupně tvořené lince. Ke správnému nakladení uvedených povelů použije makro `\crli` obsah makra `\ddlinedata` a vlevo přidává `\vvleft`. Před spuštěním makra `\ddlinedata` definuje odpovídajícím způsobem `\dditem` a `\vvitem`.

opmac.tex

```
1195: \def\crli{\cr \omit \gdef\dditem{\omit\tablinefil}\gdef\vvitem{\tabvvline}%
1196:    \vvleft\tablinefil\ddlinedata\cr}
1197: \def\crlli{\crli\noalign{\kern\hhkern}\crli}
1198: \def\tablinefil{\leaders\hrule\hfil}
1199: \def\tabvvline{\vrule\kern\vvkern\vrule}
```

Makro `\tskip` prostřednictvím `\tskipA` přechodně vyprázdní `\tabstrut` předefinováním `\tabstrutA` a také vyprázdní `\dditem` a `\vvitem`, aby po použití `\ddlinedata` vznikl řádek tabulky s prázdnými položkami. Řádek je vypodložený strutem stanovené výšky `\tmpdim`. Nakonec je potřeba vrátit `\tabstrutA` do původního stavu.

opmac.tex

```
1201: \def\tskip{\afterassignment\tskipA \tmpdim}
1202: \def\tskipA{\gdef\dditem{}\gdef\vvitem{}\gdef\tabstrutA{}%
1203:     \vrule height\tmpdim width0pt \ddlinedata\cr
1204:     \gdef\tabstrutA{\tabstrut}}
```

Globální změna šířek všech linek tvořených pomocí `\vrule` a `\hrule` je provedena makry `\rulewidth` a `\rulewidthA`. Myšlenka je dokumentována v TBN na str. 328.

opmac.tex

```
1206: \let\orihrule=\hrule  \let\orivrule=\vrule
1207: \def\rulewidth{\afterassignment\rulewidthA \tmpdim}
1208: \def\rulewidthA{\edef\hrule{\orihrule height\the\tmpdim}%
1209:                \edef\vrule{\orivrule width\the\tmpdim}}
```

Makro `\frame` {⟨*text*⟩} vloží vnější `\vbox{\hrule..\hrule}`. V něm se nachází další box `\hbox{\vrule\kern\vvkern..\kern\vvkern\vrule}` a v něm `\vbox{\kern\hhkern..\kern\hhkern}`. Nejvíce uvnitř je pak `\hbox{`⟨*text*⟩`}`. To by pro sazbu rámovaného textu stačilo, nicméně my ještě řešíme úpravu výsledného boxu tak, aby měl účaří ve stejném místě jako je účaří textu. Proto uložíme `\hbox{`⟨*text*⟩`}` do boxu0 a změříme mu hloubku. V proměnné `\tmpdim` spočítáme celkovou hloubku výsledného boxu. Ve výpočtu přičítáme výšku `\hrule`, která nemusí být 0.4pt. Proto si její výšku změříme v boxu1. Ve vnějším `\vbox`u po nakreslení spodní `\hrule` se pak vracíme pomocí `\kern-\tmpdim` na úroveň účaří a zde umístíme strut hloubky `\tmpdim`, aby sazba směrem dolů nepřečnívala, ale byla obsažena v hloubce výsledného boxu.

\crl: 41 \crll: 41 \crli: 39, 41 \tablinefil: 41 \tabvvline: 41 \dditem: 39–41
\vvitem: 39–41 \tskip: 39, 41 \tskipA: 41 \rulewidth: 41 \rulewidthA: 41 \orihrule: 41
\orivrule: 41 \frame: 42

opmac.tex
```
\def\frame#1{\setbox0=\hbox{#1}\setbox1=\vbox{\hrule}%
   \tmpdim=\dp0 \advance\tmpdim by\ht1 \advance\tmpdim by\hhkern
   \vbox{\hrule\hbox{\vrule\kern\vvkern
         \vbox{\kern\hhkern\box0\kern\hhkern}\kern\vvkern\vrule}%
         \hrule\kern-\tmpdim\hbox{\vrule depth\tmpdim width0pt}}}
```

3.20 Vložení obrázku

Nejprve deklarujeme \picwidth a \picheight. Z důvodu zpětné kompatibility je dále ztotožněn \picwidth se sekvencí \picw.

opmac.tex
```
\newdimen\picwidth    \picwidth=0pt   \let\picw=\picwidth
\newdimen\picheight   \picheight=0pt
```

Makro \inspic je zkratka za použití primitivů \pdfximage, \pdfrefximage a \pdflastximage. Kdo si to má pořád pamatovat. Není-li aktivován PDF výstup, napíšeme jen varování a neprovedeme nic.

opmac.tex
```
\ifpdftex
  \def\inspic #1 {\hbox{%
      \pdfximage \ifdim\picwidth=0pt \else width\picwidth\fi
                 \ifdim\picheight=0pt \else height\picheight\fi {\picdir#1}%
      \pdfrefximage\pdflastximage}}
\else
  \def\inspic #1 {\opwarning
     {The \noexpand\inspic is supported for PDF output only}}
\fi
```

3.21 PDF transformace

Makro \pdfscale {⟨*vodorovně*⟩}{⟨*svisle*⟩} pracuje jednoduše:

opmac.tex
```
\def\pdfscale#1#2{\pdfsetmatrix{#1 0 0 #2}}
```

Na druhé straně makro \pdfrotate ⟨*úhel*⟩ vytvoří \pdfsetmatrix{$\cos\varphi\ \ \sin\varphi\ \ -\sin\varphi\ \ \cos\varphi$}, což není jednoduché, protože funkce cos, sin nejsou v TEXu implementovány. Balíček `trig.sty` nabízí vyhodnocování těchto funkcí pomocí Taylorových polynomů, nicméně OPmac nechce být závislý na balíčcích a také chce ukázat alternativní způsob implementace. Makro \pdfrotate pracuje zhruba takto: je-li argument 0, neprovede nic, je-li argument 90, provede otočení o 90 stupňů. V ostatních případech zavolá makro \pdfrotateA, které rozloží argument na celou #1 a zlomkovou #2 část. V další části na řádcích 1247 až 1256 se zabývá jen celými stupni. Nejprve pomocí prvního a druhého \loop posune argument o celé násobky 360 stupňů tak, že poté je argument mezi 0 až 360 stupni, a přitom se hodnoty funkcí sin a cos nezměnily. Ve třetím \loop postupně snižuje argument o 90 stupňů a přitom dělá rotaci o 90 stupňů tak dlouho, až máme argument mezi nulou a devadesáti. Je-li dále argument větší než 44 stupňů, otočíme se o 45 a snížíme argument o 45. Je-li dále argument větší než 22, otočíme se o 22 a snížíme argument o 22. Nyní máme argument v množině $\{0, 1, 2, 3, \ldots, 22\}$. Pro každý prvek z této množiny argumentů máme předpřipraveny hodnoty funkcí cos a sin v makrech \smallcos a \smallsin. Použijeme je pro závěrečnou rotaci. Tím máme sazbu otočenou o celé stupně. Další část makra na řádcích 1259 až 1263 řeší jemné dotočení podle zlomkové části argumentu. V intervalu nula až jeden stupeň aproximujeme funkci cos konstantní jedničkou a funkci sin lineární funkcí $x \cdot \pi/180$. V daném rozmezí je to velmi dobrá aproximace.

opmac.tex
```
\def\pdfrotate#1{\tmpdim=#1pt
   \ifdim\tmpdim=0pt
   \else \ifdim\tmpdim=90pt \pdfsetmatrix{0 1 -1 0}%
         \else \edef\tmp{#1}\expandafter\pdfrotateA\tmp..\relax
   \fi   \fi
}
\def\pdfrotateA #1.#2.#3\relax{%
   \def\tmp##1.##2\relax {##1}%
   \tmpnum=\expandafter \tmp \the\tmpdim \relax % round
```

\picwidth: 42 \picheight: 42 \picw: 42 \inspic: 42 \pdfscale: 32, 42
\pdfrotate: 32, 42–43 \pdfrotateA: 42 \smallcos: 43 \smallsin: 43

```
1246:    \ifdim\tmpdim>0pt \def\tmpa{}\else\def\tmpa{-}\fi % save -
1247:    \loop \ifnum\tmpnum<0 \advance\tmpnum by360 \repeat
1248:    \loop \ifnum\tmpnum>360 \advance\tmpnum by-360 \repeat
1249:    \loop \ifnum\tmpnum>90 \pdfrotate{90}\advance\tmpnum by-90 \repeat
1250:    \ifnum\tmpnum=90 \pdfrotate{90}\else
1251:       \ifnum\tmpnum>44 \pdfsetmatrix{.7071 .7071 -.7071 .7071}%
1252:                        \advance\tmpnum by-45 \fi
1253:       \ifnum\tmpnum>22 \pdfsetmatrix{.9272 .3746 -.3746 .9272}%
1254:                        \advance\tmpnum by-22 \fi
1255:       \ifnum\tmpnum>0
1256:          \pdfsetmatrix{\smallcos \smallsin -\smallsin \smallcos}%
1257:    \fi\fi
1258:    \if$#2$\else % fraction part
1259:       \tmpdim=.01745329pt % \pi/180
1260:       \tmpdim=.#2\tmpdim  %
1261:       \edef\tmp{\expandafter\ignorept\the\tmpdim\space}%
1262:       \ifx\tmpa\empty \pdfsetmatrix{1 \tmp -\tmp 1}%
1263:       \else           \pdfsetmatrix{1 -\tmp \tmp 1}%
1264:    \fi\fi
1265: }
1266: \def\smallcos{.\ifcase\tmpnum \or9998\or9994\or9986\or9976\or9962\or9945\or
1267:   9925\or9903\or9877\or9848\or9816\or9781\or9744\or9703\or9659\or9613\or
1268:   9563\or9511\or9455\or9397\or9336\or9272\fi\space}
1269: \def\smallsin{.\ifcase\tmpnum 0\or0175\or0359\or0523\or0698\or0872\or1045\or
1270:   1219\or1391\or1564\or1736\or1908\or2079\or2250\or2419\or2588\or2756\or
1271:   2924\or309\or3256\or342\or3584\or3746\fi\space}
```

Pro případ, že nepracujeme s PDF výstupem, definujeme klíčové primitivy pdfTEXu jako makra, která nedělají nic.

opmac.tex

```
1273: \ifpdftex \else
1274:   \def\pdfsetmatrix#1{} \def\pdfsave{} \def\pdfrestore{}
1275: \fi
```

3.22 Poznámky pod čarou a na okraji stránek

Makro \fnote předpokládá, že správné číslo poznámky na dané stránce je připraveno v makru \fn:⟨*číslo*⟩, kde ⟨*číslo*⟩ je celkové číslo poznámky napříč celým dokumentem sledované globálním čítačem \fnotenum. Makro ohlásí svou existenci do REF souboru záznamem \Xfnote (bez parametru). Dále vytiskne značku pomocí \fnmarkx a ve skupině přejde na menší sazbu a zavolá plainTEXové makro \vfootnote, které vloží sazbu pomocí tzv. insertu (TBN, kapitola 6.7). PlainTEXové nastavení této třídy insertu není makrem OPmac nijak měněno.

opmac.tex

```
1280: \newcount\fnotenum \fnotenum=0
1281: \newcount\fnotenumlocal
1282: \newif\iflocfnum \locfnumtrue
1283:
1284: \def\fnote#1{\global\advance \fnotenum by1
1285:    \iflocfnum \leavevmode\openref\wref\Xfnote{}%
1286:       \isdefined{fn:\the\fnotenum}\iftrue
1287:       \else\opwarning{unknown \noexpand\fnote mark. TeX me again}\fi\fi
1288:    \fnmarkx{\typoscale[800/800]\vfootnote\fnmarkx{#1}}%
1289: }
```

Makro \fnotemark přičte lokálně k \fnotenum svůj parametr a vytiskne odpovídající značku. Celá práce makra probíhá ve skupině, takže po ukončení makra se \fnotenum vrátí do své původní hodnoty.

opmac.tex

```
1290: \def\fnotemark#1{{\advance\fnotenum by#1\relax
1291:    \isdefined{fn:\the\fnotenum}\iftrue\thefnote
1292:    \else$^?$\opwarning{unknown \string\fnotemark. TeX me again}\fi}%
1293: }
```

\fnote: 43 \fnotenum: 10, 43–44 \fnotemark: 43

Makro \fnotetext teprve zvedne čítač \fnotenum globálně a vytiskne poznámku pomocí plainTEXového \vfootnote.

opmac.tex
```
\def\fnotetext#1{\global\advance \fnotenum by1 \openref\wref\Xfnote{}%
   {\typoscale[800/800]\vfootnote\fnmarkx{#1}}%
}
```

Makro \fnmarkx vytiskne otazník nebo \thefnote. Předpokládá se, že si uživatel předefinuje \thefnote k obrazu svému. Lokální číslo poznámky na stránce má připraveno v makru \locfnum.

opmac.tex
```
\def\fnmarkx{\isdefined{fn:\the\fnotenum}\iftrue\thefnote\else$^?$\fi}
\def\thefnote{$^{\locfnum}$)}
\def\locfnum{\csname fn:\the\fnotenum\endcsname}
```

Při čtení REF souboru se pro každou stranu přečte nejprve \Xpage, což je makro, které pronuluje \fnotenumlocal. Makru \Xfnote tedy stačí pozvednout \fnotenumlocal o jedničku a pomocí \sxdef si tuto hodnotu zapamatovat v makru \fn:⟨*číslo*⟩.

opmac.tex
```
\def\Xfnote{\advance\fnotenumlocal by1 \advance\fnotenum by1
   \sxdef{fn:\the\fnotenum}{\the\fnotenumlocal}}
```

Makro \runningfnotes vypne lokální číslování poznámek na každé stránce. Místo toho se budou poznámky číslovat podle registru \fnotenum. Ten se zvětšuje o jedničku v celém dokumentu. Chcete-li mít poznámky číslované zvlášť například v každé kapitole, je nutno navíc resetovat tento čítač například pomocí \addto\chaphook{\global\fnotenum=0}.

opmac.tex
```
\def\runningfnotes{\locfnumfalse\def\locfnum{\the\fnotenum}\def\fnmarkx{\thefnote}}
```

Registr \mnotenum. globálně čísluje okrajové poznámky a plní podobnou funkci, jako registr \fnotenum pro podčárové poznámky. Registr \mnoteskip udává hodnotu vertikálního posunu poznámky.

opmac.tex
```
\newcount\mnotenum    \mnotenum=0       % global counter of mnotes
\newdimen\mnoteskip   \mnoteskip=0pt
```

Makro \mnote ve vertikálním módu založí box nulové výšky pomocí \mnoteA a vycouvá na původní místo sazby pomocí \vskip-\baselineskip. V odstavcovém módu toto makro nalepí box nulové výšky pod právě vytvořený řádek v odstavci. Víme, že \vadjust nalepí svůj materiál bez mezery pod tento řádek. My ovšem potřebujeme vycouvat nahoru na účaří řádku. To nejde snadno provést, protože hloubka řádku je proměnlivá. Proto do je řádku vložen \strut a předpokládá se, že nyní má řádek hloubku \dp\strutbox a o tento rozměr makro vycouvá nahoru. Vloží požadovaný box výšky nula na úrovni účaří a pak se vrátí na původní místo.

opmac.tex
```
\def\mnote#1{\ifvmode \mnoteA{#1}\nobreak\vskip-\baselineskip
   \else \strut\vadjust{\kern-\dp\strutbox \mnoteA{#1}\kern\dp\strutbox}%
   \fi
}
```

Makro \mnoteA si zjistí, zda je v makru \mn:⟨*číslo*⟩ uložen primitivní příkaz \left nebo \right. Podle toho pozná, zda má umístit poznámku doleva nebo doprava. Rovněž dá o sobě vědět do REF souboru vložením sekvence \Xmnote (bez parametru). Sazba musí v obou případech vyprodukovat box nulové výšky i hloubky. Proto je \vtop, uvnitř kterého je text poznámky zpracován, vložen přechodně do boxu0 a je mu pronulována hloubka. Nulová výška je zařízena pomocí \vbox␣to0pt{\vss\box0}. Vlastní sazbu poznámky zahajujeme pomocí \noindent s tím, že je připraven pružný \leftskip nebo \rightskip podle toho, zda poznámku klademe vlevo nebo vpravo. Při kladení vlevo musíme použít fill, abychom přeprali natahovací mezeru z \parfillskip.

opmac.tex
```
\def\mnoteA#1{\global\advance \mnotenum by1
   \isdefined{mn:\the\mnotenum}\iftrue
   \else\opwarning{unknown \noexpand\mnote side. TeX me again}\fi
   \edef\tmp{\csname mn:\the\mnotenum\endcsname}%
   \openref\wref\Xmnote{}%
   \expandafter\ifx\tmp \left
```

\fnotetext: 44 \fnmarkx: 43–44 \thefnote: 43–44 \locfnum: 43–44 \fnotenumlocal: 31, 43–44 \Xfnote: 43–44 \runningfnotes: 44 \mnotenum: 10, 44–45 \mnoteskip: 44–45 \mnote: 6, 44 \mnoteA: 44

```
1319:       \hbox to0pt{\kern-\mnotesize \kern-\mnoteindent
1320:          \vbox to0pt{\vss \setbox0=\vtop{\hsize=\mnotesize
1321:             \leftskip=0pt plus 1fill \rightskip=0pt \relax \mnotehook \noindent#1}%
1322:          \dp0=0pt \box0 \kern\mnoteskip \global\mnoteskip=0pt}\hss}%
1323:    \else
1324:       \hbox to0pt{\kern\hsize \kern\mnoteindent
1325:          \vbox to0pt{\vss \setbox0=\vtop{\hsize=\mnotesize
1326:              \rightskip=0pt plus 1fil \leftskip=0pt \relax \mnotehook \noindent#1}%
1327:           \dp0=0pt \box0 \kern\mnoteskip \global\mnoteskip=0pt}\hss}%
1328:    \fi
1329: }
```

Makro \Xmnote pracuje během čtení REF souboru a využívá toho, že makro \Xpage nastavuje číslo právě procesované strany do registru \lastpge. Takže stačí použít \sxdef následujícím způsobem:

opmac.tex

```
1330: \def\Xmnote{\advance\mnotenum by1
1331:    \sxdef{mn:\the\mnotenum}{\ifodd\lastpage \right \else \left \fi}}
```

Makro \fixmnotes ⟨*token*⟩ předefinuje v cyklu všechny definované kontrolní sekvence \mn:⟨*číslo*⟩ tak, že budou obsahovat ⟨*token*⟩. Tím toto makro zlikviduje práci makra \Xmnote, ale o to nám právě jde v případě, když chceme mít poznámky na jednoznačně dané straně.

opmac.tex

```
1333: \def\fixmnotes#1{\tmpnum=0
1334:    \loop \advance\tmpnum by1
1335:          \isdefined{mn:\the\tmpnum}\iftrue \sxdef{mn:\the\tmpnum}{#1}\repeat}
```

3.23 Bibliografické reference

Nejprve uvedeme deklarace deskriptoru \auxfile a čítačů \bibnum a \lastcitenum.

opmac.tex

```
1339: \newwrite\auxfile                    % AUX file for BibTeX
1340: \newcount\bibnum                     % the bibitem counter
1341: \newcount\lastcitenum  \lastcitenum=0  % for \shortcitations
```

Makro \cite [⟨*lejblík1*⟩,⟨*lejblík2*⟩,...] si prostřednictvím \citeA zavolá \rcite{⟨*lejblík*⟩} pro každou jednotlivou položku oddělenou čárkou ve svém parametru. Makro \citeA v sobě skrývá fintu: parametr nemá jediný, ale dva #1#2. Protože první z nich je neseparovaný, ignorují se případné mezery za čárkou a #1 obsahuje první písmeno ⟨*lejblíku*⟩. Uvnitř makra měníme \citesep, což je čárka a mezera, která se má mezi údaji vytisknout. Makro \nocite [⟨*lejblík1*⟩,⟨*lejblík2*⟩,...] je definováno stejně, jen připraví jiný význam makra \docite, krz které budeme tisknout v \rcite potřebný údaj. Veškerá činnost maker \cite a \nocite probíhá uvnitř skupiny.

opmac.tex

```
1343: \def\cite[#1]{{[\chardef\tmpb=0 \citeA #1,,,%
1344:    \ifnum\tmpb>0 \printdashcite{\the\tmpb}\fi]}}
1345: \def\nocite[#1]{{\def\docite##1{}\citeA #1,,,}}
1346: \def\citeA #1#2,{\if#1,\else \rcite{#1#2}\expandafter\citeA\fi}
1347: \def\citesep{}
```

Makro \rcite {⟨*lejblík*⟩} řeší zhruba řečeno následující věci:

- Zjistí, zda je definován \csname␣bib:⟨*lejblík*⟩\endcsname. Pokud ano, vytiskne jeho hodnotu, pokud ne, vytiskne do textu otazníky a na terminál varování. Tato kontrolní sekvence začne být známá po použití \bib[⟨*lejblík*⟩] nebo \bibitem{⟨*lejblík*⟩}. Tato makra uloží odpovídající informaci do REF souboru, odkud ji při opakovaném TEXování vyzvedneme. Je to klasická činnost, kterou provozujeme i u ostatních křížových referencí.
- Uloží o sobě zprávu do bufferu \citelist. To použijeme v makrech \usebibtex nebo \usebbl.

Makro \rcite je naprogramováno zhruba takto

```
function rcite(⟨lejblík⟩) {
  if (⟨lejblík⟩ == '*') { ⟨zapiš do⟩ \citelist '*'; return; }
  if (\bib:⟨lejblík⟩ == nedef) {
```

\Xmnote: 44–45 \fixmnotes: 45 \auxfile: 45, 47–49 \bibnum: 45, 47–49
\lastcitenum: 45, 47 \cite: 45–48, 50 \citeA: 45 \citesep: 45–47 \nocite: 45, 50
\rcite: 45–46

```
    ⟨zapiš do⟩ \citelist ⟨lejblík⟩;
    ⟨na terminál:⟩ "Warning, cite [label] unknown";
    ⟨do tiskového výstupu:⟩ "??";
    \bib:⟨lejblík⟩ = empty;
    return;
  if (\bib:⟨lejblík⟩ == empty) {
    ⟨do tiskového výstupu:⟩ "??";
    return;
  }
  if (\bib:⟨lejblík⟩ končí znakem '&') {
    ⟨zapiš do⟩ \citelist ⟨lejblík⟩;
    ⟨odstraň znak & z obsahu makra⟩ \bib:⟨lejblík⟩;
  }
  ⟨tiskni obsah makra⟩ \bib:⟨lejblík⟩;
}
```

Výklad kódu: Protože chceme šetřit pamětí bufferu `\citelist`, zapisujeme tam každý ⟨*lejblík*⟩ jen jednou. Zda se nedeklarovaný ⟨*lejblík*⟩ vyskytl poprvé poznáme podle nedefinované hodnoty `\bib:`⟨*lejblík*⟩. Zda se vyskytl později znovu poznáme podle toho, že má hodnotu `empty`. Zda se deklarovaný ⟨*lejblík*⟩ vyskytl poprvé poznáme podle znaku `&` v jeho obsahu.

Návrh kódu v C-like notaci nyní převedeme do maker v TEXu:

opmac.tex

```
\def\rcite#1{%
   \if *#1\addcitelist{*}\expandafter \skiptorelax \fi
   \expandafter \ifx \csname bib:#1\endcsname \relax
      \addcitelist{#1}%
      \opwarning{The cite [#1] unknown. Try to TeX me again}%
      \docite{}\openref
      \expandafter\gdef\csname bib:#1\endcsname {}%
      \expandafter \skiptorelax \fi
   \expandafter \ifx \csname bib:#1\endcsname \empty
      \docite{}%
      \expandafter \skiptorelax \fi
   \def\bibnn##1{}%
   \if &\csname bib:#1\endcsname
      \addcitelist{#1}%
      \def\bibnn##1##2{##1}%
      \sxdef{bib:#1}{\csname bib:#1\endcsname}%
   \fi
   \docite{\csname bib:#1\endcsname}%
   \relax
}
```

Asi nejzajímavější vychytávka v tomto makru se týká testu na znak `&`. Implicitně při čtení REF souboru se do makra `bib:`⟨*lejblík*⟩ uloží `\bibnn{`⟨*hodnota*⟩`}&`. Příkaz `\if` za sebou totálně expanduje vše následující, takže nejprve narazí na `&`, pak se obsah `bib:`⟨*lejblík*⟩ expanduje prostřednictvím `\bibnn {`⟨*hodnota*⟩`}` na nic a za tímto „nic“ se zjeví druhý znak `&`, který se tedy přilepí na ten první. Ano, je pravda, že tyto dva znaky jsou stejné. Odstranění tohoto znaku probíhá znovu totální expanzí, tentokrát `\bibnn` první parametr ⟨*hodnota*⟩ zopakuje a druhý parametr se znakem `&` zahodí.

Než se pustíme do výkladu makra `\docite`, připravíme si makra `\printcite` ⟨*položka*⟩ a `\printdashcite` ⟨*položka*⟩. První z nich tiskne jednu položku oddělenou od případné další čárkou, druhé tiskne položku, před kterou předchází pomlčka vyznačující interval položek. Pointa makra `\printcite` je v tom, že si samo po prvním zavolání připraví separátor `\citesep` (který je na začátku činnosti `\cite` prázdný), takže při opakovaném volání `\printcite` se vytiskne i požadovaný separátor. Pomlčka v `\printdashcite` je schována do `\hbox`, aby nedocházelo těsně za ní ke zlomu řádku.

opmac.tex

```
\def\printcite#1{\citesep\citelink{#1}\def\citesep{,\hskip.2em\relax}}
\def\printdashcite#1{\hbox{--}\citelink{#1}}
```

\bibnn: 46–47 \printcite: 46–47 \printdashcite: 45–47

Makro \docite ⟨položka⟩ vytiskne otazníky při prázdném parametru a jinak vytiskne prostřednictvím \printcite jednu ⟨položku⟩. Kromě toho řeší při nenulovém \lastcitenum slučování po sobě následujících čísel položek do intervalů. Naposledy vytištěnou položku uchovává v registru \lastcitenum. Při příštím zavolání zvětší \lastcitenum o jedničku a srovná ji s ⟨položkou⟩. Jsou-li si rovny, jde o následující položku v řadě a takovou položku netiskneme, nicméně si její hodnotu uchováme v \tmpb. Pokud je mezi souvislou řadou položek díra, tj. \lastcitenum se nerovná ⟨položce⟩, pak dovytiskneme předchozí interval pomocí \printdashcite{\the\tmpb} a následně vytiskneme i ⟨položku⟩. Makro \shortcitations jednoduše nastavuje \lastcitenum na nenulovou hodnotu a tím probudí k životu hlavní část makra \docite.

opmac.tex

```
\def\docite#1{\if$#1$??%
   \else
      \ifnum\lastcitenum=0   % only comma separated list
         \printcite{#1}%
      \else
         \ifx\citesep\empty  % first cite item
            \lastcitenum=#1\relax
            \printcite{#1}%
         \else               % next cite item
            \advance\lastcitenum by1
            \ifnum\lastcitenum=#1\relax % cosecutive cite item
               \mathchardef\tmpb=\lastcitenum
            \else  % there is a gap between cite items
               \lastcitenum=#1\relax
               \ifnum\tmpb=0 % previous items were printed
                  \printcite{#1}%
               \else
                  \printdashcite{\the\tmpb}\printcite{#1}\chardef\tmpb=0
   \fi\fi\fi\fi\fi
}
\def\shortcitations{\lastcitenum=1 }
```

Následuje kód makra \bib [⟨lejblík⟩], které prostřednictvím \wbib {⟨lejblík⟩}{⟨hodnota⟩} vloží do REF souboru propojené údaje o tom, jaké má ⟨lejblík⟩ přiřazeno číslo v seznamu literatury. Makro \wbib připojí před \wref příkaz \immediate právě tehdy, když \wref je ve stavu, kdy skutečně zapisuje do souboru REF. Makro \Xbib pracuje při čtení souboru REF a dělá to, co jsme si řekli už dříve: nastaví hodnotu makra \bib:⟨lejblík⟩ na \bibnn{⟨hodnota⟩}&.

opmac.tex

```
\def\bib[#1]{\par \ifnum\bibnum>0 \bibskip \fi
   \advance\bibnum by1
   \wbib{#1}{\the\bibnum}%
   \hangindent=\iindent
   \noindent \def\tmpb{#1}\dest[cite:\the\bibnum]%
   \indent \llap{[\the\bibnum] }\ignorespaces
}
\def\wbib#1#2{\edef\tmp{\wref\Xbib{{#1}{#2}}}%
  \ifx\tmp\empty\else \immediate\tmp \fi
}
\def\Xbib#1#2{\sdef{bib:#1}{\bibnn{#2}&}}
```

Makro \addcitelist {⟨lejblík⟩} přidá do \citelist údaj ve tvaru \lcite[⟨lejblík⟩]. Hranaté závorky jsou použity proto, aby fungoval test \isinlist\citelist{[⟨lejblík⟩]}. Jak uvidíme za chvíli, makro \addcitelist změní během činnosti makra \usebibtex svůj význam na \writeaux, aby případné použití \cite až za \usebitex rovnou zapisovalo do AUX souboru. Podobně makro \addcitelist změní v makru \usebbl svůj význam \writeXcite {⟨lejblík⟩}, aby v příštím průchodu TEXem mělo makro \usebbl přehled i o výskytech \cite, které jsou napsány později, než \usebbl.

opmac.tex

```
\def\addcitelist#1{\global\addto\citelist{\lcite[#1]}}
\def\writeaux#1{\immediate\write\auxfile{\string\citation{#1}}}
\def\writeXcite#1{\openref\immediate\wref\Xcite{{#1}}}
\def\citelist{} \def\citelistB{}
```

\docite: 45–47 \shortcitations: 45, 47 \bib: 33, 45–47 \wbib: 47, 49 \Xbib: 47
\addcitelist: 46–50 \citelist: 45–50 \writeaux: 47–48 \writeXcite: 47, 49–50

Než se pustíme do výkladu maker \usebibtex, \genbbl a \usebbl, uvedeme stručně popis činnosti BibTEXu. Příkaz bibtex␣⟨dokument⟩ způsobí, že program bibtex se podívá do souboru ⟨dokument⟩.aux a tam si všímá sekvencí \bibdata {⟨bib-báze⟩}, \bibstyle {⟨bib-style⟩} a \citation {⟨lejblík⟩}. Na základě toho následně přečte soubor ⟨bib-báze⟩.bib se zdrojovými zápisy bibliografických údajů. Pro konverzi těchto zdrojových zápisů do výstupního souboru ⟨dokument⟩.bbl použije stylový soubor ⟨bib-style⟩.bst. Není-li mezi sekvencemi \citation uvedeno \citation{*}, program bibtex zahrne do výstupu jen ty bibliografické údaje, které mají ⟨lejblík⟩ shodný s některým z ⟨lejblíků⟩ uvedených v parametrech sekvencí \citation. Každá sekvence \citation{⟨lejblík⟩} v souboru ⟨dokument⟩.aux typicky odpovídá jednomu použití příkazu \cite[⟨lejblík⟩].

Makro \usebibtex {⟨bib-báze⟩}{⟨bst-styl⟩} otevře soubor AUX prostřednictvím \openauxfile {⟨bib-báze⟩}{⟨bst-styl⟩}. Napíše tam tedy požadovaná data pro BibTEX. Dále z \citelist přepíše do AUX souboru lejbílky ve formátu \citation{⟨lejblík⟩}. Nakonec se uvnitř skupiny pustí do čtení souboru BBL prostřednictvím makra \readbblfile.

opmac.tex

```
\def\usebibtex#1#2{%
   \openref \openauxfile{#1}{#2}%
   \def\lcite[##1]{\writeaux{##1}}\citelist
   \global\let\addcitelist=\writeaux
   \bgroup \readbblfile{\jobname}\egroup
}
\def\openauxfile#1#2{%
   \immediate\openout\auxfile=\jobname.aux
   \immediate\write\auxfile
      {\percent\percent\space Opmac: AUX file reserved for bibtex only}%
   \immediate\write\auxfile{\string\bibdata{#1}}%
   \immediate\write\auxfile{\string\bibstyle{#2}}%
}
```

Makro \readbblfile {⟨soubor⟩} vyzkouší, zda je ⟨soubor⟩.bbl připraven ke čtení. Pokud ne, podá o tom odpovídající zprávu na terminál. Jinak nastaví čítač \bibnum na nulu a (vědomo si toho, že je spuštěno ve skupině) pustí se do lokálních re-definic LaTEXových konstrukcí, které se typicky v BBL souborech používají. Nastaví \leftskip na \iindent a spustí \bibtexhook. Konečně načte soubor BBL.

opmac.tex

```
\def\readbblfile #1{%
  \openin\testin=#1.bbl
  \ifeof\testin
    \opwarning{.bbl file doesn't exist.
               Use the ``bibtex #1'' command}%
  \else
    \closein\testin
    \bibnum=0
    \def\begin##1##2{}\def\end##1{}% LaTeX environment
    \def\newcommand##1 {}%
    \def\httpAddr##1{\url{http:##1}}\def\\{\hfill\break}%
    \def\newblock{\hskip .11em plus.33em minus.07em}%
    \def\mbox{\leavevmode\hbox}\let\em=\it
    \parindent=\iindent \bibtexhook\relax
    \input #1.bbl
    \par
  \fi
}
```

V BBL souboru se vyskytují povely \bibitem. Za každým z nich se možná objeví parametr v hranaté závorce [⟨značka⟩] a následně je uveden {⟨lejblík⟩}. Pak na dalších řádcích jsou bibliografická data jednoho záznamu ukončená prázdným řádkem. Objeví-li se [⟨značka⟩], dává tím BibTEX najevo, že se má tato ⟨značka⟩ použít místo běžného číslování záznamů. Následuje kód, který takové údaje přečte, vytiskne a vloží do REF souboru o tom zprávu prostřednictvím \wref{⟨lejblík⟩}{⟨hodnota⟩}. Rozlišují se dva režimy tisku: není-li přítomna [⟨značka⟩] (makro \tmpa je prázdné), pak pomocí \llap vytiskneme [⟨číslo⟩]. Jinak se posuneme o -\iindent a tiskneme [⟨značku⟩] následovanou mezerou. Pak se vytisknou další údaje bibliografického záznamu.

\bibdata: 48 \bibstyle: 48 \citation: 47–49 \usebibtex: 6, 45, 47–48 \openauxfile: 48–49
\readbblfile: 48–50 \bibitem: 33, 45, 49

opmac.tex
```
\def\bibitem{\isnextchar[{\bibitemB}{\def\tmpa{}\bibitemC}}
\def\bibitemB[#1]{\def\tmpa{#1}\bibitemC}
\def\bibitemC#1{\bibitemD{#1}}
\def\bibitemD#1{\par\ifnum\bibnum>0 \bibskip \fi
   \advance\bibnum by1
   \noindent \def\tmpb{#1}\dest[cite:\if$\tmpa$\the\bibnum\else\tmpa\fi]%
   \hangindent=\parindent
   \if$\tmpa$\indent\llap{[\the\bibnum] }\wbib{#1}{\the\bibnum}%
   \else [\tmpa]\wbib{#1}{\tmpa}\enskip
   \fi
   \ignorespaces
}
```

Makro \genbbl {⟨*bib-báze*⟩}{⟨*bst-style*⟩} otevře AUX soubor a zapíše do něj údaje potřebné pro BibTEX včetně \citation{*}. Poté se makro pokusí přečíst výstup z BibTEXu pomocí \readbblfile. V tomto případě pracuje \bibitem ve zvláštním režimu, kdy netiskne ⟨*hodnoty*⟩, ale ⟨*lejblíky*⟩. Z toho důvodu je předefinováno makro \bibitemC.

opmac.tex
```
\def\genbbl#1#2{\openauxfile{#1}{#2}%
   \immediate\write\auxfile{\string\citation{*}}%
   \bgroup
     \iindent=4em
     \def\bibitemC##1{\par\ifnum\bibnum>0 \bibskip \fi
        \advance\bibnum by1
        \noindent \hangindent=\parindent
        \indent \llap{[##1]\enspace}\ignorespaces
     }%
     \readbblfile{\jobname}%
   \egroup
}
```

Makro \usebbl /⟨*typ*⟩␣⟨*bbl-file*⟩ spustí jiné makro s názvem \bbl:⟨*typ*⟩. Tři taková makra jsou definována pomocí \sdef. První \bbl:a je jednoduché: prostě projde BBL soubor a vytiskne údaje z něj. Druhé makro \bbl:b projde BBL soubor v režimu, při kterém jsou bibliografická data každého záznamu (až po prázdný řádek alias \par) přečtena do parametru #2 makra \bibitemC. Celý údaj je pak vytištěn jen za předpokladu, že [⟨*lejblík*⟩] je přítomen v seznamu \citelist. Třetí makro \bbl:c pracuje jako druhé až na to, že údaj netiskne, ale zapamatuje si ho do makra \bb:⟨*lejblík*⟩. Po takovém projití BBL souboru ještě projde \citelist, kde se \lcite[⟨*lejblík*⟩] promění v \bb:⟨*lejblík*⟩, takže se záznam vytiskne. Nyní ale v pořadí, v jakém jsou ⟨*lejblíky*⟩ zařazeny do \citelist.

opmac.tex
```
\def\usebbl/#1 #2 {\isdefined{bbl:#1}%
   \iftrue \csname bbl:#1\endcsname {#2}\else
      \opwarning{\string\usebbl/#1 #2 ... the ‘#1’ type undefined}%
   \fi
}
\sdef{bbl:a}#1{\bgroup \readbblfile{#1}\egroup}

\sdef{bbl:b}#1{\bgroup
     \let\lcite=\relax \xdef\citelist{\citelist\citelistB}%
     \def\bibitemC##1 ##2\par{%
        \isinlist\citelist{[##1]}\iftrue \bibitemD{##1}##2\par\fi}%
     \readbblfile{#1}%
     \global\let\addcitelist=\writeXcite
  \egroup
}
\sdef{bbl:c}#1{\bgroup
     \let\lcite=\relax \xdef\citelist{\citelist\citelistB}%
     \def\bibitemC##1 ##2\par{%
        \isinlist\citelist{[##1]}\iftrue
           \ifx\tmpa\empty \sdef{bb:##1}{\bibitemD{##1}##2\par}%
           \else \toks0={##2\par}%
                 \edef\tmpa{\noexpand\sdef{bb:##1}{% \tmpa have to expand
                      \noexpand\bibitemB[\tmpa]{##1}\the\toks0}}\tmpa
```

\bibitemB: 49 \bibitemC: 49–50 \bibitemD: 49–50 \genbbl: 48–49 \usebbl: 6, 45, 47–50

```
1489:         \fi\fi}%
1490:       \readbblfile{#1}%
1491:       \def\bibitemC##1{\bibitemD{##1}}%
1492:       \def\lcite[##1]{\csname bb:##1\endcsname}\citelist
1493:       \global\let\addcitelist=\writeXcite
1494:     \egroup
1495: }
```

Za zmínku stojí ještě práce uvedených maker s \citelist. Před výskytem makra \usebbl se lejblíky z \cite a \nocite hromadí v \citelist. Ovšem další \cite a \nocite se mohou vyskytovat za příkazem \usebbl. Pokud se tak stane, pracuje \addcitelist nyní ve významu \writeXcite a uloží potřebnou informaci do REF souboru. Při dalším TEXování se tato informace přečte makrem \Xcite {⟨*lejblík*⟩} z REF souboru takto:

opmac.tex

```
1496: \def\Xcite#1{\addto\citelistB{\lcite[#1]}}
```

To tedy znamená, že se uloží do seznamu \citelistB. Konečně makra \bbl:b a \bbl:c si dva seznamy \citelist a \citelistB před svou činností spojí do seznamu jediného nazvaného \citelist.

3.24 Úprava output rutiny

Místo původního makra \plainoutput používá OPmac makro \opmacoutput, která je obklopeno makry \begoutput a \endoutput kvůli barvám, jak bylo vysvětleno v sekci 3.15.

opmac.tex

```
1501: \output={\begoutput \opmacoutput \endoutput}
```

OPmac mění output rutinu proti originální \plainoutput jen v nejnutnějších věcech. Řeší tyto tři problémy:

- Místo přímého \shipout nechá nejprve box sestavit jako \box0, pak provede \protectlist a pak provede \shipout\box0. Tím jsou zabezpečeny tzv. protektované příkazy při \write.
- Je vložen \pghook po sestavení boxů, ale před \shipout. Implicitně je \pghook prázdný. Mění jej makro \margins pro účely pravolevého střídání okrajů.
- Do \pagecontents vkládá \prepage (kvůli odkazům na stránku) a \preboxcclv, \postboxcclv (kvůli barvám).

První úprava zabrání expanzi maker zabezpečených pomocí \addprotect v době práce příkazu \shipout, tedy v době, kdy expandují parametry \write. Těmto makrům je přidělen prostřednictvím \doprotect ⟨*makro*⟩ pro tento okamžik význam \relax. Je potřeba ještě vysvětlit, proč bylo nutné sestavit nejprve \box0 a teprve poté jej poslat ven pomocí \shipout. Je to z toho důvodu, že v době sestavování \box0 jsou expandována \headline a \footline a pro ten případ ještě chceme, aby všechna makra správně expandovala.

opmac.tex

```
1503: \def\opmacoutput{%
1504:   \setbox0=\vbox{\makeheadline\pagebody\makefootline}%
1505:   \pghook \protectlist
1506:   \shipout\box0 \advancepageno
1507:   \ifnum\outputpenalty>-20000 \else\dosupereject\fi
1508: }
1509: \def\doprotect#1{\let#1=\relax}
```

K makru \begoutput přidáme lokální změnu makra \nl v mezeru. Makro \nl implicitně zalamuje řádky a může se vyskytovat v titulcích, takže může být dopraveno do plovoucího záhlaví, kde je zalamování řádků neřádoucí. Původní obsah makra \begoutput je definován v sekci 3.15 a řeší zejména správné nastavení barev. Makro \prepage vkládá klikatelný cíl pro stránku, je-li známo její číslo.

opmac.tex

```
1510: \addto\begoutput{\def\nl{ }}
1511: \def\prepage{\destheight=25pt \dest[pg:\the\pageno]}
```

Poslední úprava mění plainovské \pagecontents tak, že vkládá \prepage, \preboxcclv a \postboxcclv. Jinak nechává obsah makra stejný, jako v plainTEXu.

\Xcite: 47, 50 \opmacoutput: 50 \doprotect: 4, 50 \prepage: 50–51 \pagecontents: 50–51

opmac.tex
```
{\catcode`\@=11
\gdef\pagecontents{\prepage % dest of pageno
  \ifvoid\topins\else\unvbox\topins\fi
  \preboxcclv  % colors setting
  \dimen@=\dp\@cclv \unvbox\@cclv % open up \box255
  \postboxcclv % colors restoring
  \ifvoid\footins\else % footnote info is present
    \vskip\skip\footins
    \footnoterule
    \unvbox\footins\fi
  \ifr@ggedbottom \kern-\dimen@ \vfil \fi
}}
```

Když bude uživatel měnit velikost fontů v dokumentu, jistě nechce mít stránkovou číslici pokaždé jinak velkou. Proto je do `\footline` vloženo `\thefontsize`. Je nastaveno pevně na 10pt. Předpokládáme, že pokud bude někdo chtít jinak velkou stránkovou číslici, jednoduše si `\footline` nastaví podle svého. Jinak je `\footline` shodná s původním nastavením v plainTEXu.

opmac.tex
```
\footline={\hss\tenrm\thefontsize[10]\the\pageno\hss}
```

3.25 Okraje

V registrech `\pgwidth` a `\pgheight` budeme mít po zavolání `\setpagedimens` šířku a výšku strany. V registru `\shiftoffset` budeme mít případný rozdíl okrajů mezi levou a pravou stránkou.

opmac.tex
```
\newdimen\pgwidth  \newdimen\pgheight  \pgwidth=0pt
\newdimen\shiftoffset
\newif\ifmarginshook  \marginshookfalse
```

Makro `\margins` `/`⟨*typ*⟩␣⟨*formát*⟩␣`(`⟨*levý*⟩`,`⟨*pravý*⟩`,`⟨*horní*⟩`,`⟨*dolní*⟩`)`⟨*jednotka*⟩␣ si nastaví registry `\pgwidth` a `\pgheight` prostřednictvím `\setpagedimens` a dále v souladu s uživatelskou dokumentací nastaví potřebné okraje. V makru `\tmp` je schována jednotka, kterou uživatel taky může zapomenout napsat. V takovém případě vypíšeme varování a doplníme jednotku `mm`. Jakmile měníme `\hoffset` nebo `\voffset`, nastavíme je nejprve na `-1in` (tím se dostaneme na okraj papíru) a pak budeme požadovanou velikost okraje k těmto registrům přidávat. Nemohu za to, že Knutha napadla taková ne příliš podařená myšlenka dát výchozí bod sazby kamsi doprostřed papíru umístěný pomocí ujetých jednotek. Za zmínku stojí ještě dvě myšlenky. Makro `\rbmargin` `\h(v)offset\h(v)size{`⟨*okraj*⟩`}` provede výpočet hodnoty `\hoffset` nebo `\voffset` v případě, že je dána protější hodnota okraje než je okraj přímo nastavitelný pomocí `\h(v)offset`. A konečně posun okraje při přechodu z pravé na levou stránku `\shiftoffset` počítáme jako `\pgwidth - \hsize - 2*`⟨*levý*⟩ což dá stejnou hodnotu jako ⟨*pravý*⟩`-`⟨*levý*⟩. Změna `\hoffset` o tuto hodnotu je provedena v makru `\pghook`, tedy v `\output` rutině, schována do skupiny, takže po ukončení `\output` rutiny se vrátí `\hoffset` na původní hodnotu.

opmac.tex
```
\def\margins/#1 #2 (#3,#4,#5,#6)#7 {\def\tmp{#7}%
   \ifx\tmp\empty
      \opwarning{\string\margins: missing unit, mm inserted}\def\tmp{mm}\fi
   \addto\tmp{\relax}%
   \setpagedimens #2 % setting \pgwidth, \pgheight
   \ifdim\pgwidth=0pt \else
      \hoffset=-1\trueunit in \voffset=-1\trueunit in
      \if$#3$\if$#4$\tmpdim=\pgwidth \advance\tmpdim -\hsize
                    \divide\tmpdim by2 \advance\hoffset \tmpdim % left=right
             \else  \rbmargin\hoffset\hsize{#4\tmp}% only right margin
             \fi
      \else  \if$#4$\advance\hoffset #3\tmp   % only left margin
             \else  \hsize=\pgwidth           % left+right margin
                    \advance\hsize -#3\tmp \advance\hsize -#4\tmp
                    \advance\hoffset #3\tmp
      \fi\fi
      \if$#5$\if$#6$\tmpdim=\pgheight \advance\tmpdim -\vsize
                    \divide\tmpdim by2 \advance\voffset \tmpdim % top=bottom
             \else  \rbmargin\voffset\vsize{#6\tmp}% only bottom margin
```

\pgwidth: 51–52 \pgheight: 51–52 \shiftoffset: 51–52 \margins: 50–52 \rbmargin: 51–52

```
1554:              \fi
1555:       \else  \if$#6$\advance\voffset #5\tmp   % only top margin
1556:              \else  \vsize=\pgheight          % top+bottom margin
1557:                     \advance\vsize -#5\tmp \advance\vsize -#6\tmp
1558:                     \advance\voffset #5\tmp
1559:       \fi\fi
1560:       \if 1#1\shiftoffset=0pt \else \if 2#1% double-page layout
1561:          \shiftoffset=\pgwidth \advance\shiftoffset -\hsize
1562:          \advance\shiftoffset -2\hoffset \advance\shiftoffset -2in
1563:          \ifmarginshook \else \marginshooktrue
1564:             \addto\pghook{\ifodd\pageno \else \advance\hoffset \shiftoffset \fi}\fi
1565:       \else \opwarning{use \string\margins/1 or \string\margins/2}%
1566:    \fi\fi\fi
1567: }
1568: \def\rbmargin#1#2#3{\advance#1\pgwidth \advance#1-#2 \advance#1-#3}
```

Makro \setpagedimens ⟨*formát*⟩␣ spustí \setpagedimensA (⟨*šířka*⟩,⟨*výška*⟩)⟨*jednotka*⟩&, k tomu musí dopředu vyexpandovat obsah makra \pgs:⟨*formát*⟩. To provedeme pomocí tří \expandafter.

opmac.tex

```
1570: \def\setpagedimens#1 {\isdefined{pgs:#1}\iftrue
1571:    \expandafter\expandafter\expandafter \setpagedimensA \csname pgs:#1\endcsname&%
1572:    \else \opwarning{page specification "#1" is undefined}\fi}
1573: \def\setpagedimensA (#1,#2)#3&{\pgwidth=#1\trueunit#3 \pgheight=#2\trueunit#3\relax
1574:     \ifx\pdfpagewidth\undefined \else
1575:         \pdfpagewidth=\pgwidth \pdfpageheight=\pgheight \fi}
```

Jednotlivé ⟨*formáty*⟩ papíru je potřeba deklarovat.

opmac.tex

```
1577: \sdef{pgs:a3}{(297,420)mm}  \sdef{pgs:a4}{(210,297)mm}  \sdef{pgs:a5}{(148,210)mm}
1578: \sdef{pgs:a3l}{(420,297)mm} \sdef{pgs:a4l}{(297,210)mm} \sdef{pgs:a5l}{(210,148)mm}
1579: \sdef{pgs:b5}{(176,250)mm}  \sdef{pgs:letter}{(8.5,11)in}
```

Makro \magscale [⟨*factor*⟩] zvětší/zmenší sazbu nastavením registru \mag a definuje dosud prázdné makro \trueunit hodnotou true, aby později při činnosti makra \setpagedimensA zůstaly zachovány rozměry stránek. Pokud ale je makro \magscale spuštěno až po nastavení velikosti stránek, jsou tyto velikosti dodatečně korigovány na „true“ jednotky pomocí makra \truedimen.

opmac.tex

```
1581: \def\trueunit{}
1582: \def\magscale[#1]{\mag=#1\def\trueunit{true}%
1583:    \ifdim\pgwidth=0pt \else \truedimen\pgwidth \truedimen\pgheight \fi
1584:    \ifx\pdfpagewidth\undefined \else
1585:        \truedimen\pdfpagewidth \truedimen\pdfpageheight
1586:        \pdfhorigin=1truein \pdfvorigin=1truein % Origin is independent off \mag
1587:    \fi}
1588: \def\truedimen#1{#1=\expandafter\ignorept\the#1truept }
```

3.26 Závěr

Nakonec pomocí \inputref přečteme REF soubor (pokud existuje) a vrhneme se na zpracování dokumentu, který nám připravil uživatel. Přeji dobré pořízení.

opmac.tex

```
1592: \inputref
1593: \endinput
```

4 Rejstřík

Tučně je označena strana, kde je slovo dokumentováno, pak následuje seznam všech stran, na kterých se slovo vyskytuje.