verze Mar. 2011

DocBy.TEX – nástroj na dokumentování zdrojových kódů

Petr Olšák

www.olsak.net/docbytex.html

Obsah

1 Úvod

DocBy.TEX umožňuje jednoduše dokumentovat pomocí TEXu zdrojové kódy programu napsaném v jazyce C případně v jakémkoli jiném jazyce.

Na rozdíl od Knuthova literárního programování tento nástroj nepoužívá žádné preprocesory nebo filtry pro oddělení informace pro člověka a pro počítač. Vycházím z toho, že programátor je zvyklý psát tyto informace odděleně a chce mít věci pod vlastní kontrolou. Rovněž mnozí programátoři uvítají, že mohou psát dokumentaci dodatečně, a přitom skoro nezasahovat do už napsaného (a možná odladěného) zdrojového kódu. Doba, kdy Knuth navrhoval literární programování, pokročila a tvůrce dokumentace dnes může mít zároveň ve více oknech otevřeno více textů. Některé jsou určeny pro člověka a jiné pro počítač. Nevnímám tedy tak hlasitou potřebu tyto informace slučovat do jednoho souboru, jako tomu bylo kdysi.

V první části (sekce 2) dokumentu seznamujeme čtenáře s použitím DocBy.TEXu na uživatelské úrovni. V další sekci jsou dokumentovaná výchozí makra DocBy.TEXu, u nichž se předpokládá, že je bude chtít náročný uživatel měnit, aby přizpůsobil chování DocBy.TEXu obrazu svému. Dále následuje sekce 4 s dokumentací maker, která rovněž budou měněna, pokud uživatel bude chtít jiný vzhled dokumentu. V poslední sekci 5 je dokumentován kompletní DocBy.TEX na implementační úrovni. Takže se tam můžete dočíst, jak makra fungují.

Tento dokument je zpracován DocBy.TEXem, takže slouží mimo jiné jako ukázka, co je možné tímto nástrojem vytvořit.

2 Pro uživatele

2.1 Členění souborů

DocBy.TEX je implicitně navržen pro dokumentování zdrojových kódů v jazyce C. Proto i následující ukázka dokumentuje hypotetický program napsaný v tomto jazyce. Chcete-li dokumentovat jiný jazyk, můžete implicitní chování DocBy.TEXu pozměnit. Tomu je věnována sekce 3.

Předpokládá se, že zdrojové kódy programu jsou členěny na moduly. Každý modul je myšlenkově samostatná záležitost. Alespoň pro programátora. Každý modul má své jméno (například `cosi`) a je napsán v souborech `cosi.h` a `cosi.c`, případně v dalších. Tyto soubory se kompilují, aby vznikl `cosi.o` a v závěru kompilace se linkují všechny kompilované moduly do výsledného programu.

Chceme-li takové zdrojové kódy dokumentovat, připíšeme ke každému modulu soubor s příponou `.d`, například `cosi.d`, který obsahuje dokumentaci k danému modulu. Dále založíme třeba soubor `program.tex`, ze kterého postupně načítáme dokumentace jednotlivých modulů pomocí příkazu `\module`. V „hlavním souboru“ `program.tex` můžeme též použít příkazy `\title` pro vyznačení názvu programu, `\author` se jménem autora programu a třeba `\dotoc` pro vytvoření obsahu a `\doindex` pro vygenerování rejstříku. Samozřejmě zde můžeme napsat třeba úvodní poznámky ke zdrojovým kódům programu a použít plno dalších vymezovacích příkazů (viz dále). Obsah souboru `program.tex` může vypadat třeba takto:

```
\input docby.tex
\title   Program lup -- dokumentace ke zdrojovým textům

\author  Progr a Mátor

\dotoc  % tady bude obsah

\sec Členění zdrojových textů

Zdrojové texty programu "lup" jsou rozděleny do tří modulů.
V "base.c" jsou definovány pomocné funkce a v "base.h" jsou jejich
prototypy. Podobně ve "win.c" jsou funkce pro okenní záležitosti a
"win.h" obsahuje jejich prototypy. Konečně "main.c" obsahuje hlavní
funkci programu.
\module base
\module win
\module main
\doindex  % v tomto místě bude sestaven rejstřík
\bye
```

V tomto příkladě jsme se rozhodli čtenáře dokumentace seznamovat s programem „zdola nahoru“, tedy od elementárních funkcí až k hotovému programu. Někdo možná preferuje cestu „shora dolů“ a může mít v dokumentaci napsáno:

```
\module main
\module win
\module base
\doindex
\bye
```

Oba přístupy jsou možné, protože dokumentace je automaticky provázána hyperlinky. Čtenář se kdykoli může podívat na dokumentaci té funkce, jejíž použití zrovna čte, a obráceně může projít výskyty veškerého použití funkce, když čte její dokumentaci.

2.2 Příklad dokumentace modulu

Soubor s dokumentací jednotlivého modulu budu pro tento případ značit `cosi.d`. Ten je načten příkazem `\module cosi␣`. V souboru `cosi.d` je možno se literárně vyřádit a kdykoli vložit část existujícího zdrojového kódu programu se stejným jménem modulu. To provedeme příkazem `\ins c␣keyword␣`, který vloží do dokumentace část zdrojového kódu ze souboru `cosi.c`, která je vymezena pomocí slova `keyword`. Místo písmene `c` je možno použít `h` nebo jakoukoli jinou příponu souboru, ze kterého chceme vložit část do dokumentace. K vymezení částí, které se mají vložit, je nutno mít ve zdrojovém souboru text `//:␣keyword`. Vše vysvětlí následující příklad.

Předpokládejme, že v souboru `cosi.d` máme napsánu tuto dokumentaci:

```
Struktura \dg dvojice se používá jako návratová hodnota funkce
"uzasna_funkce" a sdružuje dvě hodnoty typu "float".
\ins c dvojice

Funkce \dg [struct dvojice] uzasna_funkce() si vezme jeden parametr "p"
a vrátí ve struktuře "dvojice" dvojnásobek a trojnásobek tohoto parametru.
\ins c uzasna_funkce
```

V tomto případě je nutné, aby v souboru `cosi.c` existoval vymezující text `//:␣dvojice` a text `//:␣uzasna_funkce`. Tyto texty vymezují úseky, které se mají do dokumentace vložit. Soubor `cosi.c` může vypadat třeba takto:

```
#include <stdio.h>

//: dvojice

struct dvojice {
  float x, y;
};

//: uzasna_funkce

struct dvojice uzasna_funkce (float p)
{
  struct dvojice navrat;
  navrat.x = 2*p; // tady nasobim p dvema
  navrat.y = 3*p; // tady nasobim p tremi
  return navrat;
}
```

Výsledek po zpracování části dokumentace z `cosi.d` pak vypadá takto:

Struktura `dvojice` se používá jako návratová hodnota funkce `uzasna_funkce` a sdružuje dvě hodnoty typu `float`.

cosi.c

```
5: struct dvojice {
6:   float x, y;
7: };
```

Funkce `uzasna_funkce` si vezme jeden parametr `p` a vrátí ve struktuře `dvojice` dvojnásobek a trojnásobek tohoto parametru.

cosi.c

```
11: struct dvojice uzasna_funkce (float p)
12: {
13:   struct dvojice navrat;
14:   navrat.x = 2*p; // tady nasobim p dvema
15:   navrat.y = 3*p; // tady nasobim p tremi
16:   return navrat;
17: }
```

---

struct dvojice: 5–6 struct dvojice uzasna_funkce(): 5

V ukázkovém zdrojovém kódu je první vložený úsek vymezen na začátku textem `//:␣dvojice` a na konci textem `//:`. Druhý úsek je vymezen textem `//:␣uzasna_funkce` a končí na konci souboru.

Na pořadí úseků, které zahrnujeme ze zdrojového textu do dokumentace, nezáleží. Klidně jsme mohli dokumentaci začít od povídání o úžasné funkci (včetně vložení jejího kódu) a potom ještě dopsat, co to je ta struktura `dvojice` a následně vložit deklaraci této struktury.

Kdybychom před řádek `#include␣<stdio.h>` vložili třeba text `//:␣start`, bylo by možné příkazem `\ins c␣start␣` vložit do dokumentace začátek souboru `cosi.c`, který v ukázce vložen není.

Všimněme si, že TEX zapsal čísla řádků přesně podle toho, jak jsou ve zdrojovém kódu. Tj. počítal i přeskakovaný řádek `#include␣<stdio.h>` i přeskakované prázdné a vymezující řádky.

Vymezení `//:␣keyword` se může v řádku nacházet kdekoli, není nutné, aby se vyskytovalo na začátku řádku. Řádek s tímto vymezením není do dokumentace zahrnut a pokud následuje za řádkem s vymezením prázdný řádek, ani ten není do dokumentace zahrnut.

Stejně tak koncové vymezení `//:` se může v řádku nacházet kdekoli a celý řádek s tímto vymezením není do dokumentace zahrnut. Pokud před tímto koncovým řádkem je prázdný řádek, ani ten není do dokumentace zahrnut.

Konečně za povšimnutí stojí použití příkazu `\dg` v dokumentaci. Za ním následuje slovo (separované mezerou), které dokumentujeme. Toto slovo se v dokumentaci výrazně označí (v PDF verzi červenou barvou navíc v barevném rámečku) a jakýkoli jiný výskyt takového slova ve zdrojovém textu nebo mezi uvozovkami `"..."` bude automaticky označen modrou barvou a bude klikací. Kliknutí na modrý výskyt slova kdekoli v dokumentaci vrátí čtenáře na červený výskyt, kde je slovo dokumentováno.

Dokumentované slovo může mít před sebou v hranatých závorkách text, který např. označuje typ funkce a za sebou může mít kulaté závorky `()`. Tím můžeme dát najevo, že dokumentujeme funkci. V místě dokumentace se neobjeví ani tento nepovinný text ani závorky, ale v poznámce pod čarou a v rejstříku se tyto informace vytisknou.

„Palcové uvozovky“ `"..."` vymezují kusy kódu uvnitř odstavce. Text takto uvozený je psán strojopisem a pokud se v něm vyskutují deklarovaná slova, tato slova automaticky modrají a stávají se klikatelnými odkazy. Text mezi těmito uvozovkami je navíc přepisován ve „verbatim“ módu TEXu, tj. žádné znaky nemají speciální vlastnosti (s výjimkou koncové palcové uvozovky).

Na stránce, kde je slovo dokumentováno (pomocí `\dg`), je v poznámkách pod čarou slovo znovu zmíněno a vedle této zmínky je seznam všech stránek, na kterých se kdekoli v textu vyskytuje použití tohoto slova. Dále jsou všechna dokumentovaná slova zahrnuta do závěrečného abecedního rejstříku, který odkazuje jednak na stránku, kde je slovo dokumentováno, i na stránky se všemi výskyty slova.

2.3 Jaký TEX pro DocBy.TEX?

Aby fungovaly všechny výše uvedené vlastnosti, je potřeba použít pdfTEX rozšířený o encTEX. Formát může být `plain` nebo `csplain`. V takovém případě se DocBy.TEX ohlásí na terminálu těmito slovy:

```
This is DocBy.TeX, version Mar. 2011, modes: enc+PDF+plain
nebo:
This is DocBy.TeX, version Mar. 2011, modes: enc+PDF+csplain
```

DocBy.TEX rozlišuje tři módy, každý může nabývat dvou stavů: mód `enc/NOenc`, dále mód `PDF/DVI` a konečně mód `plain/csplain`.

Mód `enc` se zapne, je-li detekována přítomnost encTEXu. Pokud encTEX není dostupný, vypíše o tom DocBy.TEX varování a přejde do `NOenc` módu. V tomto módu nefunguje automatická detekce slov, která jsou dokumentována, takže tato slova nemodrají a nestávají se klikacími odkazy. V rejstříku pak také není seznam stránek se všemi výskyty slova, ale jen místo, kde je slovo dokumentováno. V tomto případě tedy je deaktivována nejdůležitější vlastnost DocBy.TEXu, takže je žádoucí vynaložit jisté úsilí a encTEX zprovoznit. V současných distribucích TEXu bývá encTEX zahrnut. Poznáte to podle možnosti použít přepínač `-enc` v případě generování formátu. Zajistěte ve své TEXové distribuci, aby byl v době generování formátu tento přepínač použit. Jak to udělat závisí na distribuci. Univerzální, ale ne příliš elegantní postup, může vypadat takto:

```
generování formátu:
pdfetex -enc -ini -jobname pdfcsplain csplain.ini
zpracování dokumentu:
pdfetex -fmt pdfcsplain dokument.tex
```

Formát samozřejmě můžete generovat jen jednou a příkaz na zpracování dokumentu pak můžete opakovat. Pokud postupujete podle výše uvedené ukázky, uloží se formát `pdfcsplain.fmt` s podporou encTEXu do aktuálního adresáře, odkud jej příkaz `pdfetex␣-fmt␣pdfcsplain` čte. Elegantnější asi je uložit formát někam do TEXové distribuce, ovšem to je závislé na použité distribuci.

Mód `PDF` je detekován, pokud je použit pdfTEX, jinak DocBy.TEX přejde do módu `DVI` a napíše o tom varování na terminál. V módu `DVI` nefungují barvy ani klikací odkazy. Ovšem seznam stránek s použitím dokumentovaného slova se generuje, je-li přítomen encTEX.

DocBy.TEX detekuje mód `csplain`, je-li použit tento formát. V takovém případě se zapne české dělení slov a některé názvy (např. „Rejstřík“) jsou vypisovány v češtině. Slováci mohou po načtení `docby.tex` přepnout do slovenštiny pomocí `\shyph` a mohou si tyto názvy předefinovat (viz sekci 3.1). Pokud je použit formát `plain`, DocBy.TEX zůstane u vzorů dělení pro angličtinu a názvy (např. „Index“) zůstávají anglické.

2.4 Vyhledávání slov encTEXem

Slova, která se stávají klikatelnými odkazy vyhledává encTEX. Ten má zabudován tzv. „hladový algoritmus“. To znamená, že jsou-li dokumentována např. slova `abc` a `abcde`, pak text `abcdefgh` zmodrá až po písmeno `e` a odkazuje na `abcde`, zatímco `abcdx` zmodrá až po písmeno `c` a odkazuje na `abc`. To bývá obvykle žádoucí. V encTEXu není možno programovat vyhledávání podle regulárních výrazů, takže není možné jednoduše říci, aby encTEX hledal jen slova, která jsou ohraničena mezerou, tečkou, závorkou, středníkem, atd. Místo toho encTEX tupě vyhledá slovo třeba uvnitř jiného slova.

Může se tedy stát, že máme dokumentováno kratší slovo, které se objevuje jako část jiných nedokumentovaných slov. Například je dokumentována struktura `turn`, ale ve výpisech programu nechceme, aby v každém výskytu klíčového slova `return` zmodrala jeho část. V takovém případě je potřeba explicitně definovat `return` jako „normální“ nedokumentované slovo. K tomu slouží příkaz `\noactive{`⟨*slovo*⟩`}`, tedy například `\noactive{return}`. Tento příkaz globálně deklaruje ⟨*slovo*⟩ jako vyhledávané slovo (pro encTEX), ale specifikuje jej jako neaktivní.

Může se také stát, že máme dokumentováno slovo, které se objevuje ve zdrojových textech i v jiném (nedokumentovaném) významu. Přitom dokumentované slovo poznáme podle toho, jak vypadá text před slovem a za slovem. Pak lze použít deklaraci `\onlyactive{`⟨*před*⟩`}{`⟨*slovo*⟩`}{`⟨*za*⟩`}`, která sama o sobě nedělá nic. Pokud ale vyznačíme ⟨*slovo*⟩ pomocí `\dg` (nebo podobného makra na dokumentování slov, viz sekce 2.9), pak bude ⟨*slovo*⟩ automaticky modrat jen tehdy, předchází-li mu text ⟨*před*⟩ a následuje text ⟨*za*⟩. Texty ⟨*před*⟩ nebo ⟨*za*⟩ mohou být prázdné (ne oba současně) a k jednomu ⟨*slovu*⟩ můžeme napsat více různých deklarací `\onlyactive`.

DocBy.TEX aktivuje encTEX (pomocí `\mubytein=1`) jen uvnitř skupiny, když zpracovává text mezi palcovými uvozovkami (`"..."`) nebo při načítání zdrojového textu programu. Předpokládá se, že nepoužíváte encTEX k dekódování UTF-8 kódu. Pokud používáte, zkuste si zapnout `\mubytein=1` pro celý dokument, ale na *vlastní riziko*. V takovém případě vám budou modrat slova nebo jejich části i v běžném textu a pokud je dokumentované slovo podmnožinou nějaké TEXové sekvence, kterou používáte, pak se dočkáte nepříjemných chyb.

2.5 Generování rejstříku, obsahu, poznámek pod čarou a záložek

Generování rejstříku i obsahu probíhá v DocBy.TEXu zcela automaticky. Pro vytvoření rejstříku není nutné používat externí program (DocBy.TEX si slova abecedně zatřídí sám). Stačí tedy vložit na požadovaná místa příkazy `\dotoc` a `\doindex`. Upozorňuji, že rejstřík ani obsah nejsou správně vygenerovány po prvním průchodu TEXu. Je potřeba TEXovat dvakrát. Po druhém průchodu dojde zřejmě k přestránkování textu (protože je například vložen obsah). Je tedy nutné TEXovat ještě jednou. Tři průchody TEXem jsou (snad) dostačující. Slovo „snad“ vychází z problému s poznámkami pod čarou podrobně popsaném v sekci 5.7. Poznámky pod čarou se totiž průběžně v průchodech mění a ovlivňují zpětně vertikální sazbu. DocBy.TEX proto provádí na konci zpracování v příkaze `\bye` kontrolu, zda nedošlo ke změnám v referencích. Je proto užitečné používat `\bye` místo `\end`. V závěru zpracování pak DocBy.TEX vypíše zprávu `OK,␣all␣references␣are␣consistent` nebo vypíše varování, že některé reference jsou nekonzistentní a že je tedy potřeba TEXovat znovu.

Další test konzistence můžeme provést například následujícím skriptem:

```
#!/bin/bash
cp dokument.ref dokument.r0
```

```
pdfcsplain dokument.d
diff dokument.r0 dokument.ref
```

DocBy.TEX se snaží (z důvodu záruky konvergence dokumentu) fixovat zpracování poznámek pod čarou po druhém průchodu. Pokud poté měníte rozsáhle dokument, takže seznamy stránek vedle poznámek pod čarou jsou výrazně jiné délky, DocBy.TEX to nepozná a může docházet k přeplnění nebo nenaplnění stránek. V takovém případě je rozumné vymazat soubor `.ref` a znovu spustit tři průchody.

Pro vytvoření záložek se strukturovaným obsahem v PDF výstupu slouží příkaz `\bookmarks`. Je zcela jedno, v které části dokumentu je tento příkaz napsaný, neboť sestaví stukturovaný seznam záložek prolinkovaný s dokumentem na základě údajů ze souboru `.ref`. Může se stát, že některé texty v záložkách nejsou optimálně čitelné. O možnostech, jak toto řešit, pojednává sekce 3.2.

2.6 Vkládání zdrojových textů podrobněji

Kromě jednoduchého příkazu `\ins` na vkládání zdrojových textů jsou k dispozici příkazy `\ifirst` a `\inext`, které nabízejí uživateli daleko více možností.

Příkaz `\ifirst{`⟨*soubor*⟩`}{`⟨*odkud*⟩`}{`⟨*kam*⟩`}{`⟨*jak*⟩`}` vloží do dokumentu část souboru ⟨*soubor*⟩ (plný název souboru včetně přípony) od prvního řádku, na kterém se vyskytuje text ⟨*odkud*⟩ po řádek, na kterém se vyskytuje text ⟨*kam*⟩, nebo (pokud text ⟨*kam*⟩ nelze nalézt) po konec souboru. Neexistuje-li ani řádek s textem ⟨*odkud*⟩, DocBy.TEX vypíše pouze varování na terminál.

Příkaz `\ifirst` si své parametry nejprve expanduje a pak teprve použije. Aktivní vlnka v parametru expanduje na mezeru.

Parametr ⟨*jak*⟩ udává, zda se bude tisknout výchozí řádek (s textem ⟨*odkud*⟩) a koncový řádek (s textem ⟨*kam*⟩). Tento parametr obsahuje právě dva znaky (plus nebo mínus) s následujícím významem:

```
jak:  --   netiskne se výchozí ani koncový řádek
jak:  +-   tiskne se výchozí řádek a netiskne se koncový řádek
jak:  -+   netiskne se výchozí řádek, tiskne se koncový řádek
jak:  ++   tisknou se oba řádky
```

Je-li parametr ⟨*odkud*⟩ prázdný (zapíšeme pomocí `{}`), tiskne se od začátku souboru. Je-li parametr ⟨*kam*⟩ prázdný, tiskne se jediný řádek. Je-li parametr ⟨*kam*⟩`=\end`, tiskne se až do konce souboru. Koncový řádek v tomto případě neexistuje.

Má-li parametr ⟨*odkud*⟩ (nebo ⟨*kam*⟩) hodnotu `\empty` (zapíšeme pomocí `{\empty}`), tiskne se od (nebo do) prvního prázdného řádku. Parametr ⟨*jak*⟩ ovlivní jeho tisk.

Parametry ⟨*odkud*⟩ nebo ⟨*kam*⟩ mohou mít na svém začátku znak `^^B` (tím dáváme najevo, že text musí na řádku začínat) nebo na svém konci znak `^^E` (tím dáváme najevo, že text musí na řádku končit). Takže třeba `^^Btext^^E` znamená, že se vyhledává řádek, ve kterém je pouze `text` a nic jiného.

V parametrech ⟨*odkud*⟩ a ⟨*kam*⟩ se nesmějí vyskytovat speciální TEXové znaky (speciální kategorie). Pro použití znaků `\`, `{`, `}`, `%` a `"` v těchto parametrech jsou v DocBy.TEXu připraveny zástupné kontrolní sekvence `\nb`, `\obrace`, `\cbrace`, `\percent` a `\inchquote`. Sekvence pro další speciální znaky `#`, `$`, atd. si musíte vytvořit např. pomocí:

```
{\catcode`\#=12 \gdef\vezeni{#}}
```

Jsou-li parametry ⟨*odkud*⟩ a ⟨*kam*⟩ stejné, nebo oba texty jsou na stejném řádku, pak se při ⟨*jak*⟩`=++` nebo ⟨*jak*⟩`=+-` vytiskne tento jeden řádek. Při ⟨*jak*⟩`=-+` nebo ⟨*jak*⟩`=--` se tiskne až do konce souboru nebo do dalšího výskytu textu ⟨*kam*⟩.

Příkaz `\ifirst` si zapamatuje název čteného souboru a pozici posledního přečteného řádku v daném souboru. Pak je možné použít příkaz `\inext{`⟨*odkud*⟩`}{`⟨*kam*⟩`}{`⟨*jak*⟩`}`, který začíná hledat výchozí řádek s textem ⟨*odkud*⟩ od místa v souboru, kde naposledy skončilo čtení příkazem `\ifirst` nebo `\inext`. Parametry ⟨*odkud*⟩, ⟨*kam*⟩ a ⟨*jak*⟩ mají stejný význam, jako u příkazu `\ifirst`.

V registru `\lineno` je po ukončení příkazu `\ifirst` nebo `\inext` číslo řádku, které bylo naposledy přečteno (třebaže tento řádek nebyl vytištěn). Pokud bylo dosaženo konce souboru, obsahuje `\lineno` počet řádků souboru. Pomocí `\ifeof\infile` je možné se zeptat, zda bylo dosaženo konce souboru.

Příklady

```
\ifirst {soubor.txt}{textik}{textik}{++}    % vytiskne první výskyt řádku
                                            % obsahující slovo textik
\inext {textik}{textik}{++}                 % vytiskne následující výskyt
```

```
                                        % řádku obsahující slovo textík
\ifirst {soubor.c}{//: odkud}{//:}{--}  % analogie příkazu \ins
\ifirst {soubor.c}{funkce(}{)}{++}      % tisk prototypu funkce
\ifirst {soubor.c}{funkce(}{^^B\cbrace}{++} % tisk celého kódu funkce
\ifirst {soubor.txt}{}{\end}{++}        % tisk celého souboru
\ifirst {soubor.txt}{}{\empty}{+-}      % tisk po prázdný řádek
```

Je-li první řádek, který se má tisknout, prázdný, netiskne se. Je-li poslední řádek, který se má tisknout, prázdný, také se netiskne. Toto je implicitní chování. Pokud napíšete `\skippingfalse`, uvedená inteligence je zrušena a přepisují se i prázdné řádky vpředu a vzadu. Příkazem `\skippingtrue` se vrátíte k původnímu nastavení.

Parametrům ⟨*odkud*⟩ a ⟨*kam*⟩ může předcházet text `\count=`⟨*číslo*⟩␣. Hodnota ⟨*číslo*⟩ označuje, kolikátý výskyt textu ⟨*odkud*⟩ nebo ⟨*kam*⟩ se má použít. Například `{\count=3␣`⟨*odkud*⟩`}` znamená, že se má při vyhledávání ⟨*odkud*⟩ přeskočit dva jeho výskyty a začít přepisovat soubor až od výskytu třetího. Podobně `{\count=5␣`⟨*kam*⟩`}` značí, že se při přepisování souboru ignorují čtyři výskyty ⟨*kam*⟩ a přepisování se zastaví až u výskytu pátého.

Implicitně, není-li `\count=`⟨*číslo*⟩␣ uvedeno, předpokládá se `\count=1␣`.

Pokud je text ⟨*odkud*⟩ prázdný, pak `\count` označuje číslo řádku, na kterém se má zahájit výpis. Je-li prázdný parametr ⟨*kam*⟩, pak `\count` označuje počet přepisovaných řádků. Toto platí pro ⟨*jak*⟩`=++` a pro `\skippingfalse`. Při jiných hodnotách ⟨*jak*⟩ se uvedená čísla logicky posunou o jedničku. Při prázdném ⟨*odkud*⟩ nebo ⟨*kam*⟩ není mezera za `\count=`⟨*číslo*⟩ povinná. Příklady:

```
\skippingfalse
\ifirst {soubor.txt}{\count=20}{\count=10}{++} % tisk řádků 20 až 29
\ifirst {soubor.txt}{}{\count=2 \empty}{+-} % tisk po druhý prázdný řádek
\ifirst {soubor.txt}{\count=50}{\end}{++}  % tisk od 50. řádku do konce
\ifirst {soubor.tex}{\count=5 \nb section}{\count=2 \nb section}{+-}
                                    % tisk páté sekce z TeXového souboru
```

2.7 Odkazy na čísla řádků

Pomocí `\cite[`⟨*lejblík*⟩`]` je možné odkazovat na číslo řádku ve výpisu zdrojového kódu. Tento příkaz se promění na skutečné číslo řádku. Před použitím příkazu `\ifirst` nebo `\inext` je nutné ⟨*lejblík*⟩ deklarovat příkazem `\ilabel␣[`⟨*lejblík*⟩`]{`⟨*text*⟩`}`. Těchto příkazů může být před použitím `\ifirst` resp. `\inext` více. Na pořadí příkazů `\ilabel` před jedním `\ifirst` nebo `\inext` nezáleží.

Existují-li deklarované ⟨*lejblík*⟩y a ⟨*text*⟩y, pak příkaz `\ifirst` nebo `\inext` si všímá výskytu ⟨*text*⟩u ve vkládaných řádcích. Pokud takový ⟨*text*⟩ najde, přiřadí číslo řádku odpovídajícímu ⟨*lejblík*⟩u, takže příkaz `\cite` bude fungovat, jak má.

Parametr ⟨*lejblík*⟩ musí být jednoznačný v celém dokumentu. Příkaz `\cite` funguje dopředně i zpětně.

Příkazy `\ilabel` mají lokální působnost a spolupracují jen s nejbližším následujícím `\ifirst` a `\inext`. Takže před použitím dalšího `\ifirst` resp. `\inext` je potřeba deklarovat další vyhledávané texty pomocí `\ilabel` znovu.

DocBy.TEX nevypíše žádné varování, pokud nějaký ⟨*text*⟩ deklarovaný v `\ilabel` nenajde. Ovšem při použití `\cite` se objeví varování, že není známý ⟨*lejblík*⟩ a toto varování nezmizí ani při opakovaném TEXování.

Pokud se ⟨*text*⟩ vyskytuje ve více řádcích ukázky, je odkazován řádek s prvním výskytem.

V následující ukázce je čten již známý soubor `cosi.c` (viz kapitolu 2.2).

```
Na řádku~\cite[ufunkce] je deklarovaná úžasná funkce.

\ilabel [ufunkce] {funkce (float}
\ilabel [navratx] {navrat.x}
\ifirst {cosi.c}{}{}{++}

Zvláště upozorňuji na geniální myšlenku na řádku~\cite[navratx],
kde je vstupní parametr vynásoben dvěma.
```

2.8 Verbatim ukázky pomocí \begtt/\endtt a palcových uvozovek

Verbatim ukázky můžete do dokumentace vkládat pomocí \begtt a \endtt. Ty jsou (na rozdíl od vkládaných souborů) napsány přímo ve zdrojovém textu TEXu. Všechny řádky za \begtt jsou vloženy beze změn až po ukončovací \endtt. Řádky nejsou číslovány a texty v nich nemodrají a nestávají se klikatelnými odkazy.

Následující sekce 3.2 a 4.8 obsahují informace, jak je možné toto implicitní chování změnit.

Verbatim ukázky uvnitř odstavce lze vymezit palcovými uvozovkami "...". V tomto prostředí probíhá tisk strojopisem a je aktivní encTEX, takže dokumentovaná slova se stávají automaticky odkazy na místo, kde je \dg. Doporučuje se toto prostředí používat na výpisy veškerých částí kódů dokumentovaného programu, které jsou vloženy uvnitř textu v odstavci (analogie matematického prostředí $...$).

2.9 Deklarace dokumentovaného slova

Na deklaraci slova, které dokumentujeme, lze použít příkaz \dg, \dgn, \dgh, \dl, \dln nebo \dlh. Významy jednotlivých příkazů vysvětlíme později. Nejprve se věnujme syntaxi parametrů. Všechny příkazy mají stejnou syntaxi, takže nebude vadit, když bude vyložena jen v souvislosti s příkazem \dg. Syntaxe je poněkud zvláštní. Účelem totiž bylo minimalizovat práci písaře, takže jsem se vyhnul kučeravým závorkám, parametr separuji podle mezery nebo něčeho jiného, atd.

Existují tyto možnosti syntaxe parametrů:

```
\dg ⟨slovo⟩              % ⟨slovo⟩ separované mezerou
\dg [⟨text⟩] ⟨slovo⟩     % navíc nepovinný "přední" ⟨text⟩
\dg [⟨text⟩]⟨slovo⟩      % ⟨slovo⟩ může na [⟨text⟩] navazovat bez mezery
\dg ⟨slovo⟩()            % ⟨slovo⟩ s dvojicí "()" separované mezerou
\dg [⟨text⟩]⟨slovo⟩()    % kombinace předchozího
\dg ⟨slovo⟩,             % ⟨slovo⟩ separované čárkou
\dg [⟨text⟩] ⟨slovo⟩,    % kombinace předchozího
\dg ⟨slovo⟩(),           % ⟨slovo⟩ s dvojicí "()" separované čárkou
\dg [⟨text⟩]⟨slovo⟩(),   % kombinace předchozího
\dg ⟨slovo⟩.             % slovo separované tečkou
atd...
```

Obecně: za příkazem \dg může následovat nepovinná [. Pokud následuje, pak se přečte ⟨*text*⟩ až po ukončovací]. Parametr ⟨*text*⟩ může obsahovat mezery. Za ukončovací] může a nemusí být mezera. Pokud tam je, pak ji makro přesune před koncovou závorku], takže \dg [aha]␣slovo je totéž jako \dg [aha␣]slovo. Dále následuje čtení parametru ⟨*slovo*⟩. Tento parametr nesmí obsahovat mezeru, čárku, tečku, středník a dvojtečku. Čtení parametru je ukončeno, jakmile se objeví mezera nebo čárka nebo tečka nebo středník nebo dvojtečka. Uvedená interpunkce není součástí parametru ⟨*slovo*⟩ a po zpracování parametru se vrátí do vstupní fronty, takže se běžně vytiskne. Nakonec se zjistí, zda přečtený parametr až po separátor není ve tvaru ⟨*slovo*⟩(). Pokud je, pak symbol () se nepovažuje za součást parametru ⟨*slovo*⟩, ale mluvíme o ⟨*slovu*⟩ následovaném dvojicí ().

Pozor, za separátorem typu čárka, tečka, středník a dvojtečka se musí vyskytnout mezera. Ne nutně ihned, ale dříve, než se objeví úsek textu, který má být přečten s jinými kategoriemi (např. "..."). Není tedy možné psát \dg text,"...". Pokud za separátorem mezera následuje znak ‘ (obrácený apostrof), mezera ani tento znak se netiskne. To je možné využít například pro vložení nezlomitelné mezery nebo pro jiné účely: \dg ⟨*slovo*⟩␣‘~⟨*přilepený text*⟩ nebo \dg ⟨*slovo*⟩␣‘"...".

Příkazy \dgh, \dgn, \dln, \dlh separující mezeru netisknou nikdy, protože tyto příkazy většinou netisknou nic (viz níže).

Parametr ⟨*slovo*⟩ je dokumentované slovo. Pokud se takové ⟨*slovo*⟩ vyskytne někde jinde v dokumentu mezi "..." nebo ve vloženém zdrojovém kódu, automaticky zmodrá a stává se klikatelným odkazem na místo, kde je použito \dg. V místě použití \dg je slovo zvýrazněno červenou barvou. Je vytištěno samotné bez parametru ⟨*text*⟩ a bez případných závorek (). V poznámce pod čarou se vypíše ⟨*slovo*⟩ (červeně). Tam je i případný ⟨*text*⟩ (před slovem) a za ním je případná dvojice (). Vedle tohoto výpisu je seznam stránek s výskyty ⟨*slova*⟩. V rejstříku se objeví něco podobného, jako v poznámce pod čarou. Rejstřík je řazen abecedně podle ⟨*slovo*⟩, nikoli podle ⟨*text*⟩.

Příkaz \dg deklaruje ⟨*slovo*⟩ globálně. Bude na něj odkazováno v celém dokumentu.

Příkaz \dgh pracuje jako \dg, ale slovo nebude v místě \dgh vypsáno (\dg hidden). Bude tam jen cíl odkazů a ⟨*slovo*⟩ se objeví v poznámce a v rejstříku.

Příkaz \dgn způsobí, že první následující výskyt ⟨*slova*⟩ ve vypisovaném zdrojovém kódu se stane cílem všech ostatních odkazů, zčervená (tedy nezmodrá) a v místě tohoto výskytu se objeví příslušná poznámka pod čarou. Příkaz \dgn čteme jako \dg next, nebo \dg následující.

Příkaz \dl deklaruje ⟨*slovo*⟩ lokálně. Bude na něj odkazováno svým krátkým jménem ⟨*slovo*⟩ jen v místě stejného jmenného prostoru, typicky při dokumentaci jednoho modulu. Každý modul zahájený příkazem \module zavádí jmenný prostor tvaru ⟨*slovo*⟩`./`⟨*název*⟩, kde ⟨*název*⟩ je jméno modulu. Slovo deklarované pomocí \dl žije ve dvou variantách. V krátké variantě jako ⟨*slovo*⟩ jen v rozsahu jednoho jmenného prostoru a v dlouhé variantě ⟨*slovo*⟩`./`⟨*název*⟩ žije globálně v celém dokumentu. Případný výskyt dlouhého názvu odkáže na místo deklarace napříč celým dokumentem.

Podrobněji o jmenných prostorech a možnosti jejich změny najdete v sekci 2.10.

Každé ⟨*slovo*⟩ musí být v dokumentu deklarováno nejvýše jednou, jinak DocBy.TEX ohlásí chybu. V případě \dl musí existovat jednoznačný dlouhý název.

Příkaz \dlh je skrytý \dl. Příkaz \dln znamená \dl next. Analogicky, jako příkazy \dgh a \dgn.

Pokud někoho irituje vysoká inteligence těchto příkazů při čtení parametrů, může použít interní verzi příkazů s povinnými třemi parametry obalenými do kučeravých závorek: \iidg, \iidgh, \iidgn, \iidl, \iidlh, \iidln. Parametry vypadají takto: \iidg`{`⟨*před*⟩`}{`⟨*slovo*⟩`}{`⟨*za*⟩`}`. Pravda, tyto příkazy umožňují více než jejich krátké verze: umožňují do parametru ⟨*slovo*⟩ propašovat čárku, mezeru, středník atd. a do parametru ⟨*za*⟩ napsat cokoli, nejen kulaté závorky.

2.10 Jmenné prostory

Jmenný prostor je pravidlo, podle kterého se krátký název dokumentovaného ⟨*slova*⟩ transformuje při použití \dl na název dlouhý. Je možné jej nastavit nebo změnit pomocí příkazu \namespace, který se použije takto: \namespace `{`⟨*text1*⟩`#1`⟨*text2*⟩`}...`\endnamespace. Pokud je uvnitř tohoto prostředí použit příkaz \dl⟨*slovo*⟩, je slovu přidělen krátký název ⟨*slovo*⟩ a dlouhý název ⟨*text1*⟩⟨*slovo*⟩⟨*text2*⟩. Uvnitř takto deklarovaného prostředí se všechny výskyty krátkého názvu ⟨*slovo*⟩ transformují na dlouhý název a jsou prolinkovány s odpovídajícím místem \dl. Jmenný prostor je lokální uvnitř svého prostředí, takže vně prostředí se ⟨*slovo*⟩ chová, jakoby nebyl žádný příkaz \dl použit. Například uvnitř prostředí \namespace `{#1//uff}...`\endnamespace je ke každému slovu deklarovanému pomocí \dl⟨*slovo*⟩ přidělen dlouhý název ⟨*slovo*⟩`//uff` a výskyty ⟨*slovo*⟩ odkazují na místo \dl⟨*slovo*⟩.

Vně všech prostředí \namespace`...`\endnamespace není jmenný prostor definován, takže tam není možné použít příkaz \dl. Ovšem příkaz \module ⟨*název*⟩␣ nastaví jmenný prostor na `{#1./`⟨*název*⟩`}`, takže uvnitř dokumentace modulu je možné používat příkaz \dl.

V rejstříku a v poznámce pod čarou se tisknou dlouhé názvy. Rejstřík abecedně řadí podle dlouhých názvů. V obsahu se tisknou názvy krátké.

Příklad práce se jmennými prostory:

```
\namespace {ju::#1}   %% nastavuji namespace ju
Tady deklaruji slovo \dl aha.
Tady slovo "aha" automaticky odkazuje na místo deklarace.
Slovo "ju::aha" také odkazuje na místo deklarace.
\endnamespace
\namespace {hele::#1} %% nastavuji namespace hele
Tady znovu deklaruji slovo \dl aha.
Zde slovo "aha" odkazuje na lokální deklaraci uvnitř "hele"
\endnamespace         %% ruším namespace
Zde slovo "aha" neodkazuje nikam, ale slova "ju::aha"
a "hele::aha"  stále odkazují na místa, kde byla deklarována.
```

Prostředí \namespace`...`\endnamespace je možné vnořovat, ovšem vnořená prostředí musejí mít jiný jmenný prostor než prostředí vnější. Prostředí jmenných prostorů pracují globálně nezávisle na \bgroup, \egroup. Příkaz \endnamespace použitý vně všech prostředí \namespace`...`\endnamespace neudělá nic. Prostředí není nutné před příkazem \bye ukončovat.

2.11 Místo pro dokumentaci aplikačního rozhraní

Může se stát, že píšeme dokumentaci jednak pro uživatele, které zajímá způsob použití dokumentovaných funkcí a co zhruba dělají (tzv. API), ale nezajímá je, jak je funkce naprogramovaná. Druhak

chceme mít dokumentován i způsob, jak funkce funguje uvnitř. V takovém případě musí dokumentované ⟨*slovo*⟩ odkazovat na dvě místa v dokumentu.

Místo, kde je podrobně ⟨*slovo*⟩ popsáno, je vymezeno příkazem `\dg` nebo podobným. Místo, kde slovo dokumentujeme pro uživatele (je-li toto místo odlišné od prvního místa), lze vyznačit příkazem `\api{`⟨*slovo*⟩`}`. V místě použití `\api{`⟨*slovo*⟩`}` se nestane nic, jen se tam umístí neviditelný cíl odkazů. V obsahu se pak ⟨*slovo*⟩ objeví s odkazem na toto místo. V rejstříku se v seznamu stránek objeví jedna stránka podtržená: to je stránka, kde byl použit příkaz `\api{`⟨*slovo*⟩`}`. Ovšem, aby se v rejstříku ⟨*slovo*⟩ vůbec objevilo, musí se někde v dokumentu vyskytovat i jeho plná deklarace pomocí `\dg` nebo podobných příkazů. Na stránce, kde je použito `\dg`, je pod čarou vedle slova seznam stránek a rovněž je tam jedna stránka podtržená. Když čtete implementační popis pro ⟨*slovo*⟩, snadno se tedy dostanete na stránku, kde je API k tomuto ⟨*slovu*⟩. V rejstříku a obsahu jsou také slova, která byla deklarovaná pomocí `\api`, zleva vyznačena textem `\apitext`. Ten je implicitně nastaven na šipku. Můžete se podívat do rejstříku a do obsahu tohoto dokumentu. V tomto místě bylo použito `\api{\nb␣api}`, zatímco skutečná definice příkazu `\api` je v sekci 5.9.

Je-li použito `\api{`⟨*slovo*⟩`}`, pak je možné se na místo odkazovat také pomocí `\cite[+`⟨*slovo*⟩`]`. Tato konstrukce se promění v číslo stránky, kde je dokumentováno API daného slova. Například v tomto dokumentu se `\cite[+\nb␣api]` promění na: 12.

Pokud toto slovo má také svůj API cíl (vytvořený pomocí `\api`), pak se červený text (tištěný v místě `\dg`) stává aktivním odkazem na API cíl. Tam typicky čtenář najde výskyt slova, který je zase klikatelným odkazem na `\dg` cíl. Takže tyto dva cíle jsou prolinkovány křížem.

2.12 Sekce, sekcičky, část, titul

Sekce se uvozují příkazem `\sec` ⟨*název-sekce*⟩`\par`. Každá sekce může mít několik podsekcí (sekciček), které lze vyznačit příkazem `\subsec` ⟨*název-podsekce*⟩`\par`. Symbol `\par` zde znamená, že název sekce či podsekce je oddělen od dalšího textu prázdným řádkem (viz ukázku v 2.1).

Několik sekcí může tvořit část. Část je uvozena příkazem `\part` ⟨*název-části*⟩`\par`. Části jsou automaticky označeny písmeny A,B,C,... a jsou vyznačeny výrazněji než sekce v místě začátku části i v obsahu. Části ale nenarušují číslování sekcí. Tj. sekce jsou číslovány od jedné napříč celým dokumentem bez ohledu na to, zda jsou nebo nejsou rozděleny na části.

Příkaz `\module␣`⟨*soubor*⟩`␣` automaticky založí sekci s názvem `Modul␣`⟨*soubor*⟩ a deklaruje svůj jmenný prostor. Toto chování lze změnit, viz 3.1, 3.3.

Příkaz `\title`⟨*název*⟩`\par` vytiskne název dokumentu větším písmem a v rámečku. Je-li definováno makro `\projectversion`, bude jeho obsah vytištěn drobně vpravo nahoře doplněný zepředu textem `verze`. Pokud váš projekt nemá verzi, může se hodit třeba:

```
\def\projectversion{\the\day. \the\month. \the\year}
```

Příkaz `\author`⟨*text*⟩`\par` napíše do středu řádku tučně ⟨*text*⟩, což bývá obvykle jméno autora (jména autorů).

Do záhlaví každé stránky se začne přepisovat zleva název aktuální sekce a zprava název dokumentu. Uživatel může text pro pravé záhlaví změnit změnou makra `\headtitle`.

Příkazy `\sec` a `\subsec` mohou mít v hranaté závorce nepovinný parametr ⟨*lejblík*⟩. V takovém případě vypadají parametry takto: `\sec [`⟨*lejblík*⟩`]␣`⟨*název-sekce*⟩`\par`. Po takovém použití je možné se na sekci (podsekci) odkazovat příkazem `\cite[`⟨*lejblík*⟩`]`. Tento příkaz se promění v číslo odkazované sekce (podsekce) a navíc se stane aktivním odkazem.

Pomocí příkazu `\savetocfalse` lze před použitím příkazu `\sec` nebo `\subsec` zajistit, že název sekce se nedostane do obsahu a nebude mít své číslo. Místo čísla se vytiskne obsah makra `\emptynumber`, které je implicitně prázdné. Příkaz `\savetocfalse` ovlivní jen první následující `\sec` nebo `\subsec`.

2.13 Křížové odkazy

Cíl, kam směřuje odkaz, je potřeba vyznačit pomocí ⟨*lejblíku*⟩. To je možné udělat v příkaze `\sec`, `\subsec` (viz předchozí sekci 2.12) nebo kdekoli v textu samostatným příkazem `\label[`⟨*lejblík*⟩`]`. Také je možné odkazovat na číslo řádku (viz sekci 2.7). Všechny lejblíky musejí být jednoznačné (bez ohledu na jejich typ) napříč celým dokumentem.

Příkaz `\pgref[`⟨*lejblík*⟩`]` expanduje na číslo strany, na které se vyskytuje cíl odkazu. Příkaz `\numref[`⟨*lejblík*⟩`]` expanduje v závislosti na typu cíle na:

- číslo sekce, je-li cílem sekce,
- dvojčíslí ⟨*sekce*⟩.⟨*podsekce*⟩, je-li cílem podsekce,
- číslo řádku, je-li cílem řádek zdrojového kódu,
- prázdné makro, je-li ⟨*lejblík*⟩ deklarovaný pomocí `\label`.

Oba příkazy `\pgref` a `\numref` expandují na uvedené texty bez další inteligence. Tj. výstupní text se nestává klikatelným odkazem.

K aktivaci odkazu v PDF módu slouží makro `\ilink [`⟨*lejblík*⟩`]{`⟨*text*⟩`}`. Toto makro vytiskne modře ⟨*text*⟩, který se stává klikatelným odkazem na cíl, deklarovaný pomocí ⟨*lejblíku*⟩. Takže již známý příkaz `\cite[`⟨*lejblík*⟩`]` udělá zhruba to samé, jako `\ilink[`⟨*lejblík*⟩`]{\numref[`⟨*lejblík*⟩`]}`. Skutečný příkaz `\cite` navíc ověří, zda není `\numref[`⟨*lejblík*⟩`]` prázdné makro. Pokud je, obarví namísto výstupu `\numref` výstup makra `\pgref`.

Pokud ⟨*lejblík*⟩ jako argument příkazu `\pgref`, `\numref` nemá svůj cíl, příkaz `\pgref` expanduje na hodnotu −1000 a `\numref` expanduje na prázdný výstup. Jsou to expanzní makra, takže v nich není implementován například tisk varování. Podívejte se na definici příkazu `\cite` (na straně 36), jak se dá tisk varování implementovat.

Makro `\module` ⟨*jméno*⟩␣ založí sekci s lejblíkem `m:`⟨*jméno*⟩, takže lze na ní pak odkazovat. Například si můžete vytvořit makro

```
\def\refmodul[#1]{\ilink[m:#1]{\tt#1}}
```

které aktivizuje svůj parametr, pokud tento je názvem nějakého modulu. Třeba `\refmodul[base]` vytiskne slovo `base` strojopisem a modře a stává se klikatelným odkazem na začátek sekce „Modul base“, pokud je tato sekce založena příkazem `\module`.

Makra `\dg`, `\dgn`, `\dgh` interně provedou příkaz `\label[@`⟨*slovo*⟩`]` a makra `\dl`, `\dln`, `\dlh` provedou příkaz `\label[@`⟨*dlouhé-slovo*⟩`]`, kde ⟨*dlouhé-slovo*⟩ je ⟨*slovo*⟩ po transformaci podle aktuálního jmenného prostoru. Na místa, kde jsou slova dokumentovaná, je tedy možné odkazovat například pomocí `\link[@`⟨*slovo*⟩`]{`⟨*slovo*⟩`␣dokumentované␣na␣straně~\pgref[@`⟨*slovo*⟩`]}`.

Makro `\api{`⟨*slovo*⟩`}` interně provede `\label[+`⟨*slovo*⟩`]`, takže je možné na toto místo odkazovat třeba pomocí `\ilink[+`⟨*slovo*⟩`]{API:␣`⟨*slovo*⟩`}`.

DocBy.TEX nenabízí kromě čísel sekcí, podsekcí a čísel řádků žádné další automatické číslování. Pokud tedy chcete implementovat např. číslování obrázků, čísla publikací atd., musíte si napsat makra vlastní. K tomu můžete využít makro `\labeltext[`⟨*lejblík*⟩`]{`⟨*text*⟩`}`, které uloží v horizontálním módu do sazby neviditelný cíl odkazu, a při dalším průchodu TEXem expanduje makro `\numref` na ⟨*text*⟩. Použití makra ukážeme na příkladě, ve kterém definujeme makro `\bib[`⟨*lejblík*⟩`]`. Toto makro zahájí sazbu další položky v seznamu literatury. Odkazovat na knihu pak lze pomocí `\cite[b:`⟨*lejblík*⟩`]`.

```
\newcount\bibnum
\def\bib [#1]{\par\advance\bibnum by1 \indent
    \llap{[\the\bibnum] }\labeltext[b:#1]{[\the\bibnum]}\ignorespaces}
```

2.14 Vkládání obrázků

Příkazem `\ifig` ⟨*šířka*⟩␣⟨*jméno-obrázku*⟩␣ je možné vložit obrázek. Obrázek musí být připraven v souboru `fig/`⟨*jméno-obrázku*⟩`.eps` (v případě DVI módu) a v souboru `fig/`⟨*jméno-obrázku*⟩`.pdf` (v případě PDF módu). Adresář, kde DocBy.TEX vyhledává obrázky (`fig/`), lze změnit předefinováním sekvence `\figdir` . Rozměr ⟨*šířka*⟩ je bez jednotky a udává poměr požadované šířky obrázku ku šířce sazby. Obrázek je umístěn zarovnán doleva na odstavcovou zarážku.

Máte-li připraven obrázek ve formátu `eps`, pak jej do `pdf` převedete příkazem

```
ps2pdf -dEPSCrop ⟨jméno obrázku⟩.eps
```

2.15 Výčty

Seznam položek obklopíte `\begitems` a `\enditems`. V tomto prostředí je text odsazen zleva o odstavcovou zarážku. Prostředí lze vnořovat. Jednotlivou položku zahájíte pomocí `\item` ⟨*značka*⟩␣⟨*text*⟩, přitom ⟨*značka*⟩ se vystrčí vlevo od ⟨*textu*⟩. Je-li ⟨*značka*⟩ hvězdička, promění se v puntík. Další možnost: `\item \the\itemno)`␣⟨*text*⟩, což vytvoří číslované výčty, v každém prostředí číslovány od jedné.

Makro plainu `\item` není předefinováno globálně, ale jen uvnitř `\begitems...\enditems`. Můžete tedy použít i makro plainu, pokud se vám koncept položek nabízený DocBy.TEXem nelíbí.

3 Pro náročné

V této sekci jsou uvedeny a vysvětleny definice základních příkazů DocBy.TEXu. Uživatel si může tyto definice změnit, pokud chce změnit chování DocBy.TEXu. Pokud například pracuje s jiným programovacím jazykem, může si změnit makro \docsuffix nebo kompletně předefinovat makra \module a \ins.

3.1 Interní názvy

Příkazem \doindex vytvoří DocBy.TEX automaticky novou sekci s názvem „Rejstřík“. Podobně při tvorbě obsahu nebo natažení modulu vzniká název „Obsah“ nebo „Modul“. Před názvem verze v titulu při použití \projectversion se objeví slůvko „verze“. Část (vytvořená pomocí \part) má v záložkách uvozující text >>␣CAST. Tyto texty jsou definovány v makrech \titindex, \tittoc, \titmodule, \titversion a \opartname.

docby.tex

```
19: \ifx\chyph\undefined      % format: plain, anglicke nazvy
20:    \def\titmodule{Module}
21:    \def\tittoc{Table Of Contents}
22:    \def\titindex{Index}
23:    \def\titversion{version }
24:    \def\opartname{>> PART}
25: \else                     % format: csplain, ceske nazvy
26:    \def\titmodule{Modul}
27:    \def\tittoc{Obsah}
28:    \def\titindex{Rejstřík}
29:    \def\titversion{verze }
30:    \def\opartname{>> CAST}
31: \fi
```

Za povšimnutí stojí, že jsou jinak tato makra definována při použití klasického plainu a jinak při použití csplainu. To ovšem neznamená, že uživatel si tyto názvy nemůže předefinovat ještě jinak, nezávisle na použitém formátu.

3.2 Vložené skupiny příkazů (hooks)

Některá složitější makra (\begtt, palcové uvozovky, \ifirst, \inext, \doindex, \dotoc) dovolují vkládat uživateli na začátku zpracování různé příkazy (tzv. hooks). Implicitně jsou tyto vložky prázdné:

docby.tex

```
35: \def\begtthook{}
36: \def\quotehook{}
37: \def\indexhook{}
38: \def\tochook{}
39: \def\bookmarkshook{}
40: \def\outputhook{}
```

Makro \begtthook je vloženo po založení skupiny a nastavení všech kategorií těsně před začátkem zpracování prostředí \begtt...\endtt. Makro \quotehook je vloženo po založení skupiny a nastavení všech kategorií těsně před začátkem zpracování prostředí "...". Makro \indexhook je vloženo makrem \doindex po založení sekce a před přechodem do sazby ve dvou sloupcích. V tomto dokumentu je v něm úvodní povídání k rejstříku. Makro \tochook je vloženo makrem \dotoc po založení sekce před sazbou prvního řádku obsahu. Makro \bookmarkshook je vloženo uvnitř skupiny na začátku zpracování záložek. Je možné v něm nastavit expanze maker vyskytujících se v nadpisech na rozumnou hodnotu pro záložky. Pokud navíc nastavíte \let\cnvbookmark=\lowercase, budou všechny znaky pro záložky procházet filtrem \lowercase. Uvnitř \bookmarkshook je pak možné nastavit \lccode vybraným znakům (například pro odstranění háčků a čárek). Makro \outputhook je vloženo na začátek výstupní rutiny. Je vhodné v něm nastavit vybrané příkazy na hodnotu \relax, aby se neexpandovaly do souboru .ref.

Příklady použití

\titindex: 14, 39 \tittoc: 14, 38 \titmodule: 14–15 \titversion: 14, 18–19
\opartname: 14, 40 \begtthook: 14–15, 28 \quotehook: 14–15, 44 \indexhook: 14–15, 39
\tochook: 14, 38 \bookmarkshook: 14, 40 \outputhook: 14–15, 33

```
\def\quotehook{\obeyspaces}   % ve výpisech "..." budou normální mezery
\def\quotehook{\langleactive} % <text> se promění na ⟨text⟩
\def\begtthook{\mubytein=1}   % mezi \begtt...\endtt bude aktivní encTeX
\def\begtthook{\setsmallprinting} % ukázky \begtt...\endtt budou malé
\def\begtthook{\catcode`\!=0} % mezi \begtt...\endtt fungují !prikazy
\def\indexhook{To čubrníte, jaký tu mám rejstřík.}
\def\outputhook{\let\mylogo=\relax} % \mylogo nebude expandovat
```

3.3 Příkaz \module a \ins

Uživatelská dokumentace k těmto příkazům je v sekci 2.1. Příkaz \module ⟨*soubor*⟩␣ načte soubor s názvem ⟨*soubor*⟩\docsuffix, kde makro \docsuffix obsahuje příponu souboru včetně tečky.

docby.tex

```
44: \def\docsuffix {.d}  % implicit filename extension (for \module command)
45: \def\module #1 {\endnamespace\namespace{##1./#1}\sec [m:#1] \titmodule\space #1 \par
46:    \def\modulename{#1} \input #1\docsuffix\relax
47: }
```

Příkaz \module vloží název čteného souboru (bez přípony) do pomocného makra \modulename. Toto makro pak využívá příkaz \ins ⟨*pripona*⟩␣⟨*text*⟩␣.

docby.tex

```
48: \def\ins #1 #2 {\ifirst {\modulename.#1}{//: #2}{//:}{--}}
```

3.4 Zelenající komentáře

Příkazy \ifirst a \inext si také všímají (implicitně) C komentářů tvaru //..⟨*eol*⟩ a /*␣..␣*/. Tyto komentáře barví ve výpisu programu zeleně. Zrušit tuto vlastnost lze příkazem \noactive⟨*string*⟩. Pomocí \setlinecomment {⟨*string*⟩} lze nastavit nový typ komentářů, které budou barveny zeleně od ⟨*string*⟩ do konce řádku. Příkazy mají globální platnost. Například

```
\noactive{/*}\noactive{*/}\noactive{//}
\setlinecomment{\percent} \noactive{\nb\percent}}
```

nastaví komentáře podle zvyklostí v TEXu a PostScriptu.

Příkazem \setlrcomment {⟨*levý*⟩}{⟨*pravý*⟩} lze nastavit komentáře typu /*...*/.

Pro změnu vlastností obarvování komentářů stačí uvedená makra použít. Kdo chce vědět, jak jsou implementovaná, nechť čte dále.

docby.tex

```
52: \ifx\mubyte\undefined
53:     \def\setlinecomment#1{}
54:     \def\setlrcomment#1#2{}
55: \else
56:    \def\setlinecomment#1{\mubyte \linecomment ##0 #1\endmubyte}
57:    \def\setlrcomment#1#2{\mubyte \leftcomment ##0 #1\endmubyte
58:         \mubyte \rightcomment #2\endmubyte \gdef\rightcomment{#2\returntoBlack}}
59: \fi
```

Uvedené příkazy jsou prázdné v módu bez encTEXu a při detekci encTEXu zapíší informace do encTEXové tabulky prostřednictvím primitivů \mubyte...\endmubyte.

Příkazy \linecomment a \leftcomment se díky encTEXu automaticky vloží před detekovanou sekvenci znaků. Tyto příkazy nastaví barvu textu na zelenou:

docby.tex

```
61: \def\linecomment {\let\Black=\Green \Green}
62: \def\leftcomment {\global\let\Black=\Green \Green}
```

Na druhé straně příkaz \rightcomment potřebuje vypnout zelenou barvu až po přeskočení detekované sekvence. Proto encTEX v tomto případě detekovanou sekvenci zruší a příkaz \rightcomment má za úkol ji vrátit do sazby zpět a teprve poté pomocí \returntoBlack se vrátit k černé barvě.

docby.tex

```
64: \def\returntoBlack {\global\let\Black=\oriBlack \Black}
```

\module: 4, 5, 11–15 \docsuffix: 14–15 \modulename: 15 \ins: 5, 6, 8, 14–15 \setlinecomment: 15–16 \setlrcomment: 15–16 \linecomment: 15, 25, 44 \leftcomment: 15, 25, 44 \rightcomment: 15 \returntoBlack: 15, 25, 44

Je potřeba vysvětlit, proč přepínače barev jsou tak komplikovaně zapsány. Přepínač totiž v PDF zapíná barvu nezávisle na skupině a barva textu se drží tak dlouho, dokud není použit jiný přepínač barvy. Každý tisk řádku kódu je uveden přepínačem \Black, takže při poznámce „do konce řádku“ stačí jen přepnout na \Green. Ovšem uvnitř komentáře se může objevit link obalený příkazy \Blue...\Black (viz např. řádek 44 v předchozí sekci). Pak ale chceme, aby \Black vrátil barvu \Green. Proto je provedeno předefinování pomocí \let. Toto předefinování je lokální. Protože řádek je tištěn uvnitř skupiny, je další řádek už černý.

Při tisku komentáře, který má úvodní a koncový znak a může přesáhnout jeden řádek, musíme globálně předefinovat \Black na \Green, aby i další řádky (uvozené příkazem \Black) byly zelené. Koncový znak komentáře pak musí uvést barvy do původního stavu.

DocBy.TEX inicializuje poznámky podle pravidel jazyka C:

docby.tex
```
66: \setlinecomment{//}  \setlrcomment{/*}{*/}
```

4 Pro designéry

Následuje dokumentace definic maker ovlivňující vzhled dokumentu. Jejich předefinování může způsobit změnu vzhledu podle požadavku uživatele. Místo komplikovaných maker s množstvím parametrů pro řízení vzhledu jsou zde jednoduchá dobře dokumentovaná makra pro jedno použití. Předpokládá se, že při potřebě jiného vzhledu dokumentu je uživatel předefinuje.

Makra zabývající se vzhledem dokumentu jsou pokud možno oddělena od složitosti ostatních maker, ve kterých probíhá hlavní zpracování DocBy.TEXu. To umožňuje designérovi zaměřit se jen na programování vzhledu a neutopit se v různých cyklech a rekurzích interních maker DocBy.TEXu.

Typicky jsou makra pro vzhled ve dvou verzích: pro pdfTEX a bez pdfTEXu. To je důvod, proč ve výpisech se často vyskytuje test \ifx\pdfoutput\undefined.

4.1 Parametry a pomocná makra pro nastavení vzhledu

Velikost \hsize ani \vsize neměníme. Buď si ji nastaví uživatel, nebo se převezme velikost z plainu (vhodné pro papír letter) či csplainu (vhodné pro papír A4). Nastavujeme ale větší \parindent, neboť chceme do proužku vymezeného \parindent dát podbarvené čtverečky u názvů sekcí.

docby.tex
```
70: \parindent=30pt
```

Připravíme si „zúženou šířku“ \nwidth využitou např. jako šířka záhlaví:

docby.tex
```
72: \newdimen\nwidth \nwidth=\hsize \advance\nwidth by-2\parindent
```

Příkazem plainu \raggedbottom nastavíme pružnost stránky dole, a ne mezi jednotlivými řádky. Nastavením \exhyphenpenalty=10000 zakážeme zlom za pomlčkou (v tisku rozsahu stránek, např. 11–13, takový zlom působí rušivě).

docby.tex
```
74: \raggedbottom
75: \exhyphenpenalty=10000
```

Zavedeme potřebné fonty \bbf, \bbbf, \btt, \ttsmall, \rmsmall, \itsmall a \partfont.

docby.tex
```
77: \font\bbf=csb10 at12pt
78: \font\bbbf=csb10 at14.4pt
79: \font\btt=cstt12
80: \font\ttsmall=cstt8
81: \font\rmsmall=csr8
82: \font\itsmall=csti8
83: \font\partfont=csb10 at80pt
```

Makro \setsmallprinting přepne do malého strojopisu, připraví \ttstrut vhodné velikosti a pomocí \offinterlineskip připraví tisk řádků v režimu, kdy se o sebe opírají. Hodnota \parskip je nastavena na -1pt, aby docházelo k mírnému překrývání a nevznikaly v tisku nebo na obrazovce pruhy. Analogicky pracuje makro \setnormalprinting.

\hsize: 16, 21–22, 33, 38, 43 \vsize: 32, 43–44 \parindent: 16, 21–23, 33, 38, 43
\nwidth: 16, 19, 33 \bbf: 16, 18 \bbbf: 16, 18–20 \btt: 16, 18 \ttsmall: 16–17, 20–21, 32
\rmsmall: 16–20, 32 \itsmall: 16–17 \partfont: 16, 18 \setsmallprinting: 15, 17, 21–22
\ttstrut: 17, 21–22 \setnormalprinting: 17, 22

docby.tex

```
\def\setsmallprinting{\ttsmall \let\it=\itsmall \let\rm=\rmsmall
   \def\ttstrut{\vrule height8pt depth3pt width0pt}%
   \offinterlineskip \parskip=-1pt\relax
}
\def\setnormalprinting{\tt \baselineskip=0pt \hfuzz=4em
   \def\ttstrut{\vrule height10pt depth3pt width0pt}%
   \offinterlineskip \parskip=-1pt\relax
}
```

V návrhu vzhledu pracuji jen s barvami \Blue, \Red, \Brown, \Green, \Yellow a \Black. Pokud budete chtít další barvy, definujte si je.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\setcmykcolor#1{}
\else
   \def\setcmykcolor#1{\special{PDF:#1 k}}
\fi
\def\Blue{\setcmykcolor{0.9 0.9 0.1 0}}
\def\Red{\setcmykcolor{0.1 0.9 0.9 0}}
\def\Brown{\setcmykcolor{0 0.85 0.87 0.5}}
\def\Green{\setcmykcolor{0.9 0.1 0.9 0.2}}
\def\Yellow{\setcmykcolor{0.0 0.0 0.3 0.03}}
\def\Black{\setcmykcolor{0 0 0 1}}
\let\oriBlack=\Black
```

Barvy jsou definovány pomocí makra \setcmykcolor, které je v případě DVI výstupu nastaveno na prázdné makro a v případě PDF výstupu je použit PDF \special. Takže příkazy \Brown atd. je možné použít i ve verzi maker pro DVI, ovšem v této verzi neudělají nic. Barva \oriBlack je konstantně černá barva. Některá makra totiž normální \Black předefinovávají a pak se potřebují vrátit pomocí \oriBlack ke skutečné černé barvě.

Makro \rectangle {⟨*výška*⟩}{⟨*hloubka*⟩}{⟨*šířka*⟩}{⟨*obsah*⟩} vytvoří rámeček o stanovených rozměrech se stanoveným obsahem. V PDF verzi je rámeček ve tvaru plného žlutého obdélníku na kterém se nachází ⟨*obsah*⟩ zatímco v DVI verzi se vytvoří obrysový rámeček. Pozor: parametr ⟨*obsah*⟩ musí obsahovat přepínač barvy, jinak nebude v PDF verzi viditelný. Na druhé straně makro \rectangle se postará o návrat do „normální“ černé barvy.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\rectangle#1#2#3#4{\vbox to0pt{\vss\hrule\kern-.3pt
      \hbox to#3{\vrule height#1 depth#2\hss#4\hss\vrule}%
      \kern-.3pt\hrule\kern-#2\kern-.1pt}}
\else
   \def\rectangle#1#2#3#4{\vbox to0pt{\vss\hbox to#3{%
       \rlap{\Yellow \vrule height#1 depth#2 width#3}%
       \hss#4\Black\hss}\kern-#2}}
\fi
```

Nakonec připravíme makro \docbytex jako zkratku pro logo DocBy.TEXu.

docby.tex

```
\def\docbytex {\leavevmode\hbox{DocBy}.\TeX}
```

4.2 Vzhled sekcí a podsekcí

Makra \printsec {⟨*nadpis*⟩} a \printsecbelow, jsou volána z makra pro vytvoření sekce \sec a mají za úkol vytisknout nadpis. Ostatní problematika, kterou musí řešit makro \sec (reference do obsahu, cílové reference, čísla sekcí, plovoucí záhlaví atd.) je zde odstíněna a nemusíme se jí v tuto chvíli zabývat.

Musíme ale dodržet následující úmluvu: Na začátku makra \printsec přejdeme pro jistotu do vertikálního módu, pak vložíme potřebné mezery, pak vložíme text nadpisu. V okamžiku, kdy přejdeme do horizontálního módu, vložíme makro \makelinks, které zajistí umístění cílů odkazů. Nakonec přejdeme do vertikálního módu příkazem \par a *nevkládáme žádné další vertikální mezery*. Makro \sec vloží pod

\Blue: 16–17, 20, 35 \Red: 17, 20, 36 \Brown: 17–19, 21 \Green: 15–17, 35 \Yellow: 17, 19–22
\Black: 15–22, 26, 33, 35–36 \setcmykcolor: 17 \oriBlack: 15, 17, 20, 26 \rectangle: 17–20
\docbytex: 17, 33, 40 \printsec: 17–18, 34, 44 \printsecbelow: 18, 34

vytištěný nadpis do horizontálního seznamu další prvky a posléze zavolá \printsecbelow. Tam teprve vložíme mezery obvykle blokované proti zlomu pomocí \nobreak. Základní řazení vertikálního seznamu v TEXu totiž vypadá takto: box, (whatsit, mark, atd.), penalty, glue. O objekty uvedené v závorce se postará \sec, my zde řešíme jen box (v makru \printsec), a dále penaltu a glue (v makru \printsecbelow).

K dispozici máme hodnotu \secnum a \subsecnum a dále můžeme použít test \ifsavetoc, kterým se ptáme, zda daný nadpis bude v obsahu. Nebude-li, měli bychom místo \the\secnum tisknout \emptynumber. V makru \seclabel je obsah lejblíku sekce, nebo je makro prázdné. To můžeme využít při tisku v režimu „nahrubo“, například tisknout tyto lejblíky do okrajů. DocBy.TEX tuto vlastnost implicitně neimplementuje.

docby.tex

```
\def\printsec #1{\par
    \removelastskip\bigskip\medskip
    \noindent \makelinks
    \rectangle{16pt}{9pt}{25pt}{\Brown\bbbf\ifsavetoc\the\secnum\else\emptynumber\fi}%
    \kern5pt{\bbbf\let\_=\subori #1}\par
}
\def\printsecbelow {\nobreak\medskip}
```

Makra \printsubsec a \printsubsecbelow fungují analogicky jako právě zmíněná, ale spolupracují s makrem \subsec.

docby.tex

```
\def\printsubsec #1{\par
    \removelastskip\bigskip
    \noindent \makelinks
    \vbox to0pt{\vss
       \rectangle{16pt}{9pt}{25pt}{\Brown\bf
          \ifsavetoc\the\secnum.\the\subsecnum\else\emptynumber\fi}\kern-5pt}%
    \kern5pt{\bbf\let\_=\subori \let\tt=\btt #1}\par
}
\def\printsubsecbelow {\nobreak\smallskip}
```

Makro \printpart vytiskne nadpis části a dopředu dá veliké písmeno. Makro \printpartbelow tiskne mezeru pod nadpisem části.

docby.tex

```
\def\printpart #1{\par
    \removelastskip\bigskip\medskip
    \noindent {\linkskip=60pt\makelinks}%
    \rectangle{16pt}{9pt}{25pt}{}%
    \kern-20pt{\Brown\partfont\thepart\Black}\kern10pt{\bbbf #1}\par
}
\def\printpartbelow {\nobreak\bigskip}
```

Makro \emptynumber, které se použije při \savetocfalse, je implicitně nastaveno na prázdnou hodnotu.

docby.tex

```
\def\emptynumber{}
```

4.3 Titul, autor

Makro \title ⟨*titul*⟩\par čte parametr ⟨*titul*⟩ pomocí makra \secparam, které se postará o případné ignorování mezery na konci parametru (viz sekci 5.8). Makro \secparam uloží parametr ⟨*titul*⟩ do tokenlistu \sectitle a spustí interní \iititle. Toto makro pracuje ve dvou módech (DVI a PDF). V obou módech \iititle uloží ⟨*titul*⟩ do makra \headtitle (pokud je toto makro prázdné, tedy neinicializované uživatelem) a pomocí příkazu \noheadline potlačí na aktuální stránce tisk záhlaví.

docby.tex

```
\def\title{\def\tmpA{title}\futurelet\nextchar\secparam}
\ifx\pdfoutput\undefined
   \def\iititle {\par
      \ifx\headtitle\empty\edef\headtitle{\the\sectitle}\fi
      \noheadline
      \ifx\projectversion\empty \else
         \line{\hss\rmsmall\titversion\projectversion}\nobreak\medskip\fi
      \centerline{\bbbf \let\_=\subori\the\sectitle}\nobreak\medskip}
```

\printsubsec: 18, 34, 44 \printsubsecbelow: 18, 34 \printpart: 18, 35 \printpartbelow: 18, 35
\emptynumber: 12, 18, 34 \title: 12, 4, 18–19, 44 \iititle: 18–19

```
\else
  \def\iititle {\par
      \ifx\headtitle\empty\edef\headtitle{\the\sectitle}\fi
      \noheadline
      \indent\rlap{\rectangle{25pt}{15pt}{\nwidth}{\Black\let\_=\subori\bbbf\the\sectitle}}%
          \ifx\projectversion\empty \else
          \hbox to\nwidth{\hss
             \raise26pt\hbox{\Brown\rmsmall\titversion\projectversion\Black}}\fi
      \par\nobreak\vskip15pt}
\fi
```

Makro \title v DVI verzi je prosté \centerline, zatímco v PDF verzi tiskne podkladový obdélník šířky \nwidth.

Pokud není makro \projectversion definováno, nastavíme mu výchozí hodnotu jako prázdné makro:

docby.tex

```
\ifx\projectversion\undefined \def\projectversion{}\fi
```

Makro \author ⟨*autor*⟩\par je společné v obou módech. Umístí jméno autora tučně a na střed.

docby.tex

```
\def\author #1\par{\centerline{\bf #1\unskip}\smallskip}
```

4.4 Hlavičky a patičky

DocBy.TEX nemění výstupní rutinu plainu. Využívá tedy klasické nástroje na modifikaci vzhledu, tj. text \footline a \headline.

Návrh vzhledu stránky nepočítá s pravou a levou stranou, protože dokumentaci většinou čteme na monitoru a když ji tiskneme, tak kdo ví, na čem…

Text \footline je nastaven tak, aby byla stránková číslice uprostřed podbarvena případně orámována pomocí \rectangle.

docby.tex

```
\footline={\hss\rectangle{8pt}{2pt}{25pt}{\tenrm\Black\folio}\hss}
```

Text \headline se mění. Implicitně obsahuje jen makro \normalhead, ale při použití příkazu \noheadline na chvíli změní svůj obsah.

docby.tex

```
\headline={\normalhead}
\def\normalhead {\savepglink \let\_=\subori
    \vbox to0pt{\vss \baselineskip=7pt \lineskiplimit=0pt
     \line{\indent\Black\tenit \firstmark \hss \headtitle\indent}
     \line{\indent\Yellow\xleaders\headlinebox\hfil\indent\Black}}}
```

Makro \normalhead uloží stránkový link pomocí \savepglink a \vbox/\hbox gymnastikou vytvoří potřebné záhlaví. Zleva je tištěn název sekce (\firstmark) a zprava konstantní text \headtitle.

Makro \noheadline nastaví \headline přechodně na text, podle kterého se vloží jen stránkový odkaz a provede změna obsahu \headline na standardní hodnotu. Operace musíme provádět globálně, protože jsme uvnitř výstupní rutiny.

docby.tex

```
\def\noheadline {\global\headline={\savepglink\hfil\global\headline={\normalhead}}}
```

Makro \headtitle obsahuje text shodný v celém dokumentu tištěný vpravo v záhlaví. Implicitně je makro prázdné, po použití příkazu \title obsahuje název dokumentu, pokud si uživatel makro nedefinoval sám.

docby.tex

```
\ifx\headtitle\undefined \def\headtitle {}\fi
```

Pomocné makro \headlinebox udělá v DVI módu prázdný čtvereček a v PDF módu plný (žlutý) čtvereček. Je použito na řádku 182 pro vytvoření čtverečkované čáry v záhlaví,

docby.tex

```
\ifx\pdfoutput\undefined
   \def\headlinebox{\hbox{\kern2pt\rectangle{4pt}{0pt}{4pt}{}\kern2pt}}
\else
   \def\headlinebox{\hbox{\kern2pt\vrule height4pt depth0pt width4pt\kern2pt}}
\fi
```

\projectversion: 12, 14, 18–19 \author: 12, 4, 19 \footline: 19 \headline: 19, 35
\normalhead: 19, 44 \noheadline: 18–19 \headtitle: 12, 18–19, 40 \headlinebox: 19

4.5 Tisk cíle odkazu a odkazů pod čarou

Cíl odkazu vytvořený makry \dg nebo \dl je potřeba vytisknout výrazně, aby jej čtenář pokud možno rychle našel. Tisk probíhá v makru \printdg {⟨před⟩}{⟨slovo⟩}{⟨za⟩}, kde ⟨před⟩ je text před slovem a ⟨za⟩ je prázdný parametr nebo obsahuje (), pokud tyto závorky uživatel v příkaze \dg, \dl použil.

Současný návrh DocBy.TEXu tiskne z těchto tří parametrů jen jeden, sice ⟨slovo⟩. V DVI módu tiskne ⟨slovo⟩ v rámečku a v PDF módu tiskne ⟨slovo⟩ červeně a na pozadí je žlutý obdélník.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\printdg#1#2#3{\leavevmode\hbox{\kern-.6pt
      \vbox{\hrule\hbox{\vrule height8.5pt depth2.5pt \kern.2pt
            \tt#2\kern.2pt\vrule}\hrule\kern-2.9pt}\kern-.6pt}}
\else
   \def\printdg#1#2#3{\leavevmode \setbox0=\hbox{\tt#2}%
           \Yellow\rlap{\vrule height8.7pt depth2.7pt width\wd0}%
           \printdginside{#2}{\box0}}
\fi
```

Červený text se tiskne pomocným makrem \printdginside, které tiskne jednoduše červeně, pokud ke slovu neexistuje \api cíl a tiskne červeně pomocí \ilink, jestliže existuje \api cíl.

docby.tex

```
\def\printdginside#1#2{\ifnum\pgref[+#1]>-1 {\let\Blue=\Red \ilink[+#1]{#2}}%
    \else \Red#2\relax\Black\fi}
```

Údaj pod čáru tiskneme makrem \printfnote {⟨před⟩}{⟨d-slovo⟩}{⟨za⟩}{⟨k-slovo⟩}, kde parametry ⟨před⟩ a ⟨za⟩ mají stejný význam, jako u makra \printdg. Parametr ⟨k-slovo⟩ (krátká verze slova) tiskneme červeně, ostatní parametry černě. Parametr ⟨d-slovo⟩ (dlouhá verze slova) není použit.

K naprogramování tohoto makra využiji makro \specfootnote {⟨text⟩}, které pošle text do speciální poznámky pod čarou. Dále je potřeba vědět, že \pgref[+⟨slovo⟩] vrátí číslo strany, kde je \api deklarace ⟨slova⟩ nebo vrátí −1000. Toto číslo vložíme do \apinum a je-li nezáporné, tak jej uvedeme jako první v seznamu stránek a podtržené. Seznam stránek vytiskneme pomocí \listofpages{⟨slovo⟩}. V seznamu bude chybět stránka \apinum, protože makro \listofpages ji vynechává. Prázdný seznam stránek (při kterém netiskneme dvojtečku ani čárku) poznáme podle toho, že \box0 má nulovou šířku.

docby.tex

```
\def\printfnote #1#2#3#4{%
  \specfootnote{{\let\Black=\oriBlack \ttsmall #1\Red #4\Black#3\rmsmall
       \apinum=\pgref[+#2]\relax
       \ifnum\apinum>-1 :~\lower1.4pt\vbox{\hbox{\pglink\apinum}\kern1pt\hrule}\fi
       \undef{w:#2}\iftrue \setbox0=\hbox{}\else \dgnum=-1 \setbox0=\hbox{\listofpages{#2}}\fi
       \ifdim\wd0=0pt \else
           \ifnum\apinum>-1 , \else :~\fi
           \unhbox0
       \fi}}%
}
```

4.6 Tisk údaje v obsahu a v rejstříku

Příkaz \ptocline {⟨číslo⟩}{⟨text⟩}{⟨strana⟩} se postará o tisk údaje o sekci nebo části do obsahu. Dále příkaz \ptocsubline {⟨číslo⟩}{⟨text⟩}{⟨strana⟩} vytiskne údaj o subsekci. Jak je patrné, tyto dva příkazy se liší jen o jeden \indent:

docby.tex

```
\def\ptocline #1#2#3{%
   \if^^X#1^^X\advance\partnum by1 \medskip \fi
   \line{\rectangle{8pt}{1pt}{25pt}{%
     \if^^X#1^^X\ilink[sec:\thepart]{\bbbf \thepart}\else\ilink[sec:#1]{#1}\fi}\kern5pt
     {\bf\let\_=\subori #2}\mydotfill\pglink#3}}
\def\ptocsubline #1#2#3{%
   \line{\indent\rectangle{8pt}{1pt}{25pt}{\ilink[sec:#1]{#1}}\kern5pt
         \let\_=\subori #2\mydotfill\pglink#3}}
\def\mydotfill{\leaders\hbox to5pt{\hss.\hss}\hfil}
```

Příkaz \mydotfill vytiskne tečky do obsahu tak, aby byly pod sebou zarovnány.

\printdg: 20, 31 \printdginside: 20, 31–32 \printfnote: 20, 31 \ptocline: 20, 38
\ptocsubline: 20, 38 \mydotfill: 20

Příkaz \ptocentry ⟨typ⟩{⟨slovo⟩}{⟨k-slovo⟩} vytiskne jednu položku o dokumentovaném slově do obsahu. Parametr ⟨typ⟩=+, pokud je v daném místě \api dokumentace, a ⟨typ⟩=@, je-li v daném místě \dg dokumentace. ⟨k-slovo⟩ je prázdné, ale při použití \dl je v něm krátká verze slova, zatímco ve ⟨slovo⟩ je dlouhá verze slova. Dlouhou verzí odkazujeme, krátkou verzi tiskneme.

docby.tex

```
\def\ptocentry#1#2#3{\ifhmode,\hskip 7pt plus 20pt minus 3pt \fi
    \noindent \hbox{\ttsmall \if+#1\apitext\fi \ilink[#1#2]{\ifx^^X#3^^X#2\else#3\fi}}%
    \nobreak\myldots\pglink\pgref[#1#2]\relax
}
\def\myldots{\leaders\hbox to5pt{\hss.\hss}\hskip20pt\relax}
```

Kdyby bylo potřeba tisknout text před slovem nebo závorky za slovem, je možné využít kontrolní sekvenci \csname-⟨slovo⟩\endcsname jako v následujícím makru \printindexentry.

Makro \myldots vytvoří tři tečky, které jsou zarovnány s ostatními tečkami v obsahu.

Makro \printindexentry {⟨slovo⟩} tiskne údaj o slově do rejstříku. Začíná ve vertikálním módu uvnitř sloupce, vytiskne údaj a pomocí \par se musí vrátit do vertikálního módu.

docby.tex

```
\def\printindexentry #1{%
     \expandafter \expandafter\expandafter \separeright \csname-#1\endcsname\end
     \apinum=\pgref[+#1]\relax
     \leavevmode\llap{\ttsmall \ifnum\apinum>-1 \apitext\fi\tmpa}%
         {\tt \ilink[@#1]{#1}\tmpb}: {\bf\pglink\pgref[@#1]}%
         \ifnum\apinum>-1 , $\underline{\pglink\apinum}$\fi
         \dgnum=\pgref[@#1]\relax
         \undef{w:#1}\iftrue \setbox0=\hbox{}\else \setbox0=\hbox{\it\listofpages{#1}}\fi
         \ifdim\wd0=0pt \else, \unhbox0 \fi
     \hangindent=2\parindent \hangafter=1 \par
}
\def\separeright #1\right#2\end{\def\tmpa{#1}\def\tmpb{#2}}
```

Pomocí \separeright uložím do \tmpa text vlevo od slova a do \tmpb text vpravo od slova. Makro \refdg tyto údaje uložilo do makra \csname-⟨slovo⟩\endcsname oddělené od sebe značkou \right. Pomocí makra \pgref[@⟨slovo⟩] získám stránku s \dg deklarací slova. Pomocí \pgref[+⟨slovo⟩] získám stránku s \api deklarací slova. Tuto stránku (pokud existuje) tisknu podtrženě.

4.7 Tisk zdrojového textu

Makra \ifirst a \inext přetisknou požadovanou část zdrojového textu. Při řešení návrhu vzhledu tisku nás nyní pouze zajímá, že tato makra založí skupinu, pak zavolají příkaz \printiabove, pak pro tisk každého řádku zavolají \printiline {⟨číslo⟩}{⟨text-řádku⟩} a nakonec před ukončením skupiny se spustí \printibelow. Právě tato tři makra si nyní naprogramujeme. Budeme rozlišovat mezi DVI a PDF módem.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\printiabove{\line{\leaders\specrule\hfill \kern2pt
         {\ttsmall \Brown\inputfilename}\kern2pt \specrule width\parindent}\nobreak
          \setsmallprinting}
   \def\printibelow{\vskip2pt\hrule\medskip}
   \def\specrule{\vrule height 2pt depth-1.6pt }
   \def\printiline#1#2{\noindent\ttstrut
      \hbox to\parindent{\hss#1:\kern.5em}{#2\par}\penalty11 }
\else
   \def\printiabove{\smallskip \setsmallprinting}
   \def\printibelow{\medskip}
   \def\printiline #1#2{\noindent
      \rlap{\Yellow \ttstrut width\hsize}%
      \ifx\isnameprinted\undefined
          \rlap{\line{\hss \raise8.5pt
             \hbox{\ttsmall \Brown \vrule height5pt width0pt \inputfilename}}}%
          \let\isnameprinted=\relax
      \fi
      \hbox to\parindent{\hss\Brown#1:\Black\kern.5em}{#2\par}\penalty11 }
\fi
```

\ptocentry: 21, 37–38, 40 \myldots: 21 \printindexentry: 21, 39 \separeright: 21
\printiabove: 21, 26 \printiline: 21–22, 26–27 \printibelow: 21, 26–27

V DVI módu tiskneme nahoře čáru se jménem souboru pomocí \leaders a makra \specrule. Dole pak tiskneme jen jednoduchou čáru. V PDF módu nahoře pouze nastavíme \setsmallprinting a vložíme malou mezeru. Dole vložíme střední mezeru.

Makro \printiline přejde nejprve do horizontálního módu, tam vloží v DVI módu podpěru a dále box s číslem a box s řádkem. Mezi řádky vkládám penaltu 11. V PDF módu se místo podpěry tiskne celý žlutý proužek v \rlap. Protože přes první řádek je potřeba vpravo nahoru vytisknout jméno souboru (později než žlutý proužek), je potřeba zjistit, zda tisknu první řádek nebo další řádky. K tomu slouží kontrolní sekvence \isnameprinted, která je typicky \undefined. Po vytištění jména souboru (řádky 268 a 269) nastavím \isnameprinted na \relax a tím poznám, že už je práce provedena. Až makro \ifirst nebo \inext ukončí skupinu, bude zase mít \isnameprinted hodnotu \undefined.

4.8 Tisk z prostředí \begtt/\endtt

Makro \begtt založí skupinu a zavolá \printvabove. Dále pro každý tištěný řádek volá makro \printvline {⟨číslo⟩}{⟨text-řádku⟩} a nakonec zavolá \printvbelow. Číslo řádku jsme se rozhodli nevyužít. V DVI verzi kreslíme jen čáry nahoře a dole. V PDF verzi kreslíme žluté čáry nahoře a dole a v každém řádku pomocí \rlap kreslíme žluté obdélníky vpravo a vlevo.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\printvabove{\smallskip\hrule\nobreak\smallskip\setnormalprinting}
   \def\printvbelow{\nobreak\smallskip\hrule\smallskip}
   \def\printvline#1#2{\hbox{\ttstrut\indent#2}\penalty12 }
\else
   \def\printvabove{\medskip\Yellow\hrule height2pt \setnormalprinting\nobreak}
   \def\printvbelow{\Yellow\hrule height2pt \Black\medskip}
   \def\printvline#1#2{\noindent
     \rlap{\hbox to\hsize{\Yellow\ttstrut width25pt\hfil
        \vrule width25pt\Black}}\hbox{\indent#2}\par\penalty12 }
\fi
```

4.9 Vkládání obrázků

Obrázky jsou vkládány nalevo podle odstavcové zarážky. Tato zarážka je dostatečně velká, takže to působí docela dobře. Celkovou šířku prostoru pro obrázek \figwidth spočítám jako \hsize mínus \parindent

docby.tex

```
\newdimen\figwidth  \figwidth=\hsize \advance\figwidth by-\parindent
```

Makro \ifig ⟨poměr-šířky⟩␣⟨název⟩␣ v DVI módu vloží ⟨název⟩.eps a využije k tomu makrobalík epsf.tex. V PDF módu vloží ⟨název⟩.pdf a využije k tomu pdfTEXové primitivy \pdfximage, \pdfrefximage, \pdflastximage.

docby.tex

```
\ifx\pdfoutput\undefined
   \input epsf
   \def\ifig #1 #2 {\bigskip\indent
      \hbox{\epsfxsize=#1\figwidth\epsfbox{\figdir#2.eps}}\bigskip}
\else
   \def\ifig #1 #2 {\bigskip\indent
       \hbox{\pdfximage width#1\figwidth {\figdir#2.pdf}%
            \pdfrefximage\pdflastximage}\bigskip}
\fi
\def\figdir{fig/}
```

Makro \figdir obsahuje adresář, ze kterého se obrázky loví.

4.10 Výčty

Makra pro výčty jsou natolik jednoduchá, že asi nepotřebují dalšího konimentáře. \begitems zahájí prostředí s výčty, \enditems ukončí toto prostředí, \itemno čísluje a \dbtitem ⟨značka⟩␣ zahajuje položku, přičemž se uvnitř prostředí převtělí na \item.

\specrule: 21 \isnameprinted: 21–22 \printvabove: 22, 28 \printvline: 22, 28
\printvbelow: 22, 28 \figwidth: 22 \ifig: 13, 22 \figdir: 13, 22 \begitems: 13, 23
\enditems: 13, 23 \itemno: 13, 23 \dbtitem: 23 \item: 13, 23

docby.tex

```
\newcount\itemno
\def\begitems{\medskip\begingroup\advance\leftskip by\parindent \let\item=\dbtitem}
\def\dbtitem #1 {\par\advance\itemno by1 \noindent\llap{\ifx*#1$\bullet$\else#1\fi\kern3pt}}
\def\enditems{\medskip\endgroup}
```

5 Pro otrlé

Zde je dokumentována implementace DocBy.TEXu. Je zde výpis všech jeho interních maker včetně podrobného komentáře, jak fungují. Asi není rozumné tato makra měnit, ledaže by si chtěl čtenář naprogramovat DocBy.TEX vlastní.

5.1 Pomocná makra

Makro \dbtwarning zprostředkuje tisk varovných hlášek:

docby.tex

```
\def\dbtwarning#1{\immediate\write16{DocBy.TeX WARNING: #1.}}
```

Makra \defsec {⟨*text*⟩}, \edefsec {⟨*text*⟩} a \undef {⟨*text*⟩} jsou zkratky za časté operace s \csname⟨*text*⟩\endcsname.

docby.tex

```
\def\defsec#1{\expandafter\def\csname#1\endcsname}
\def\edefsec#1{\expandafter\edef\csname#1\endcsname}
\def\undef#1\iftrue{\expandafter\ifx\csname#1\endcsname\relax}
```

Makro \undef je potřeba použít takto:

\undef{⟨*text*⟩}\iftrue ⟨*sekvence nedefinovaná*⟩ \else ⟨*sekvence definovaná*⟩ \fi

Nutnost použití \iftrue se bohatě vyplatí, až budeme \undef přeskakovat vnějšími podmínkami typu \if.

Definuji makro \nb (normální backslash). Toto makro je pak možné používat při vyhledávání textu s tímto znakem. Rovněž definuji aktivní tabelátor a zástupné sekvence \obrace, \cbrace, \percent a \inchquote.

docby.tex

```
{\catcode`\^^I=\active \gdef^^I{\space\space\space\space\space\space\space\space}
 \catcode`\|=0 \catcode`\\=12 |gdef|nb{\}}
\bgroup
   \catcode`\[=1 \catcode`]=2 \catcode`\{=12 \catcode`\}=12 \catcode`\%=12
   \gdef\obrace[{] \gdef\cbrace[}] \gdef\percent[%]
\egroup
\def\inchquote{"}
```

Makro \softinput je vysvětleno v TEXbooku naruby na straně 288, takže bez komentáře.

docby.tex

```
\def\softinput #1 {\let\next=\relax \openin\infile=#1
   \ifeof\infile \dbtwarning{The file #1 does not exist, run me again}
   \else \closein\infile \def\next{\input #1 }\fi
   \next}
```

Makro \setverb nastaví kategorie všech speciálních znaků na normální. Viz TEXbook naruby, stranu 28.

docby.tex

```
\def\setverb{\def\do##1{\catcode`##1=12}\dospecials}
```

5.2 Inicializace

Ohlásíme se na terminál:

docby.tex

```
\immediate\write16{This is DocBy.TeX, version \dbtversion, modes:
    \ifx\mubyte\undefined NO\fi enc+%
    \ifx\pdfoutput\undefined DVI\else PDF\fi+%
    \ifx\chyph\undefined \else cs\fi plain}
```

\dbtwarning: 23–25, 27, 31, 36–37, 39 \defsec: 23, 28, 30, 36–37, 39, 42 \edefsec: 23, 30, 37–39, 42
\undef: 20–21, 23–24, 28–30, 32, 36, 39–40 \nb: 8, 12, 15, 29–30, 33, 35, 39–40 \obrace: 8, 23
\cbrace: 8, 23 \percent: 8, 15, 23 \inchquote: 8, 23 \softinput: 23 \setverb: 23, 26, 28, 44

Makro \dbtversion obsahuje verzi DocBy.TEXu a je definováno na začátku souboru docby.tex. Tam je autor DocBy.TEXu pozmění, pokud přejde na novou verzi.

docby.tex
```
4: \def\dbtversion {Mar. 2011} % version of DocBy.TeX
```

Inicializujeme csplain mód:

docby.tex
```
342: \ifx\chyph\undefined \else \chyph \fi
```

Inicializujeme encTEXový mód:

docby.tex
```
344: \ifx\mubyte\undefined
345:    \dbtwarning{encTeX is not detected}
346:    \message{ \space The documented words will be not recognized in source code.}
347:    \message{ \space Use pdfetex -ini -enc format.ini to make
348:                     your format with encTeX support.}
349:    \csname newcount\endcsname \mubytein
350:    \def\enctextable#1#2{}
351:    \def\noactive#1{}
352: \else
353:    \def\enctextable#1#2{%
354:       \def\tmp ##1,#1,##2\end{\ifx^^X##2^^X}%
355:       \expandafter \tmp \owordbuffer ,#1,\end
356:          \expandafter \mubyte \csname.#1\endcsname #1\endmubyte \fi
357:       \expandafter \gdef \csname.#1\endcsname {#2}%
358:    }
359:    \def\noactive#1{\mubyte \emptysec ##0 #1\endmubyte}
360:    \def\emptysec{}
361: \fi
```

Makro \enctextable {⟨slovo⟩}{⟨tělo-makra⟩} vloží do encTEXové tabulky vzor ⟨slovo⟩. Jakmile takový vzor encTEX objeví, zruší jej ze vstupního proudu a promění jej v kontrolní sekvenci \.⟨slovo⟩, která expanduje na ⟨tělo-makra⟩. Například makro \dg ⟨slovo⟩ aktivuje pro encTEX ⟨slovo⟩, takže provede (mimo jiné) \enctextable{⟨slovo⟩}{\sword{⟨slovo⟩}}, což způsobí, že se ⟨slovo⟩ v načítaném zdrojovém kódu promění na \sword{⟨slovo⟩}.

Makro \enctextable odmítá uložit do encTEXové tabulky slova, která jsou v seznamu „zakázaných“ slov \owordbuffer. Tam jsou slova (oddělená z obou stran čárkou), která se nesmějí aktivovat kvůli \onlyactive. Pro taková slova provede \enctextable jen definici sekvence \.⟨slovo⟩.

Makro \noactive {⟨text⟩} vloží do encTEXové tabulky vyhledávaný text, který ve vstupu zůstane a před něj bude vložena sekvence \emptysec. Protože encTEX neumí ze své tabulky zrušit údaj (umí jen přepsat informaci, na co se má vyhledávaný text proměnit), je potřeba texty, které už v encTeXové tabulce nepotřebujeme, deaktivovat alespoň pomocí \noactive.

Na \sword {⟨text⟩} se díky encTEXu proměňují texty, které se mají automaticky stát klikatelnými linky.

docby.tex
```
363: \def\sword#1{\ilink[@#1]{#1}\write\reffile{\string\refuseword{#1}{\the\pageno}}}
```

Makro \onlyactive {⟨před⟩}{⟨slovo⟩}{⟨za⟩} zakáže vkládat ⟨slovo⟩ do encTEXové tabulky (vloží je do \owordbuffer, ovšem jen za předpokladu, že už tam není), a nechá celý text ⟨před⟩⟨slovo⟩⟨za⟩ proměnit v \oword{#1}{#2}{#3}. Dále pomocí \noactive dekativuje ⟨slovo⟩ (při čtení \reffile totiž pravděpodobně bylo aktivováno). Makro \oword {⟨před⟩}{⟨slovo⟩}{⟨za⟩} tiskne normálně ⟨před⟩, dále, pokud je definováno \.⟨slovo⟩, tak je spustí, jinak tiskne normálně ⟨slovo⟩. Konečně tiskne vždy normálně text ⟨za⟩.

docby.tex
```
365: \def\onlyactive #1#2#3{\enctextable{#1#2#3}{\oword{#1}{#2}{#3}}%
366:    \def\tmp ##1,#2,##2\end{\ifx^^X##2^^X}%
367:    \expandafter \tmp \owordbuffer ,#2,\end
368:       \addtext #2,\to\owordbuffer \noactive{#2}\fi}
369: \def\owordbuffer{,}
370: \def\oword#1#2#3{#1\undef{.#2}\iftrue #2\else\csname.#2\endcsname\fi #3}
```

Nakonec inicializujeme DVI/PDF mód:

\dbtversion: 23–24 \enctextable: 24, 29, 31, 38 \owordbuffer: 24 \noactive: 7, 15, 24, 29, 32
\emptysec: 24 \sword: 24–25, 31, 38 \onlyactive: 7, 24 \oword: 24

docby.tex

```
372: \ifx\pdfoutput\undefined
373:    \dbtwarning{pdfTeX is not detected}
374:    \message{ \space The document will be without colors and hyperlinks.}
375:    \message{ \space Use pdfTeX engine, it means: pdfetex command, for example. }
376: \else
377:    \pdfoutput=1
378: \fi
```

5.3 Makra \ifirst, \inext, \ilabel

Deklarujeme \lineno jako číslo řádku, \ttlineno jako číslo řádku pro \begtt...\endtt výpisy, \ifcontinue pro řízení cyklu a \infile je deskriptor souboru otevřeného ke čtení. \ifskipping implementuje uživatelské \skippingfalse a \skippingtrue.

docby.tex

```
382: \newcount\lineno
383: \newcount\ttlineno
384: \newif\ifcontinue
385: \newif\ifskipping  \skippingtrue
386: \newread\infile
```

Příkaz \ifirst {⟨soubor⟩}{⟨odkud⟩}{⟨kam⟩}{⟨jak⟩} nejprve pomocí \readiparamwhy analyzuje parametr ⟨jak⟩, pak otevře soubor ke čtení primitivem \openin. Je-li otevření neúspěšné, vypíše varování, jinak si uloží název souboru do makra \inputfilename a analyzuje parametry pomocí \scaniparam: ⟨odkud⟩ je uloženo do \tmpa a ⟨kam⟩ do \tmpb. Do \tmpA a \tmbB se uloží počet opakování (z konstruktoru \count=⟨num⟩). Nakonec se spustí makro \insinternal s expandovanými parametry ⟨odkud⟩, ⟨kam⟩. K tomu je použit známý trik s makrem \act.

docby.tex

```
388: \def\ifirst #1#2#3#4{\par\readiparamwhy#4..\end
389:    \openin\infile=#1 \global\lineno=0
390:    \ifeof\infile
391:       \dbtwarning {I am not able to open the file "#1" to reading}
392:    \else
393:       \xdef\inputfilename{#1}
394:       \scaniparam #2^^X\tmpa\tmpA \scaniparam #3^^X\tmpb\tmpB
395:       {\let~=\space \def\empty{^^B^^E}\let\end=\relax \uccode`\~=`\"\uppercase{\let~}"%
396:       \noswords \xdef\act{\noexpand\insinternal {\tmpa}{\tmpb}}}\act
397:    \fi
398: }
```

Příkaz \inext {⟨odkud⟩}{⟨kam⟩}{⟨jak⟩} pracuje analogicky, jako \ifirst, pouze neotevírá soubor, ale pomocí testu na definovanost makra \inputfilename kontroluje, zda náhodou nebyl spuštěn příkaz \inext bez předchozího \ifirst.

docby.tex

```
399: \def\inext #1#2#3{\par\readiparamwhy#3..\end
400:    \ifx\inputfilename\undefined
401:       \dbtwarning {use \string\ifirst\space before using of \string\inext}
402:    \else
403:       \ifeof\infile
404:          \dbtwarning {the file "\inputfilename" is completely read}
405:       \else
406:          \scaniparam #1^^X\tmpa\tmpA \scaniparam #2^^X\tmpb\tmpB
407:          {\let~=\space \def\empty{^^B^^E}\let\end=\relax \uccode`\~=`\"\uppercase{\let~}"%
408:          \noswords \xdef\act{\noexpand\insinternal{\tmpa}{\tmpb}}}\act
409:    \fi\fi
410: }
```

V rámci expanze parametrů chceme, aby zmizely všechny kontrolní sekvence, které nám do textu vložil automaticky encTEX. To provede makro \noswords.

docby.tex

```
411: \def\noswords{\def\sword##1{##1}\def\lword##1{##1}\def\fword##1##2##3{##2}%
412:    \let\flword=\fword \def\leftcomment{}\def\returntoBlack{}\def\linecomment{}}
```

Makro \readiparamwhy načte znaky + nebo - z parametru ⟨jak⟩ a uloží je do sekvencí \startline a \stopline.

docby.tex
```
414: \def\readiparamwhy#1#2#3\end{\let\startline=#1\relax\let\stopline=#2\relax}
```

Makro \scaniparam ⟨param⟩^^X⟨out⟩⟨outnum⟩ čte ⟨param⟩ ve tvaru \count=⟨num⟩␣⟨text⟩. Do sekvence ⟨out⟩ uloží ⟨text⟩ a do sekvence ⟨outnum⟩ uloží ⟨num⟩. Protože konstruktor \count=⟨num⟩ je nepovinný, dá trochu více práce parametr analyzovat. K tomu slouží i pomocná makra \scaniparamA, \scaniparamB, \scaniparamC. V případě nepřítomnosti \count=⟨num⟩ je v ⟨outnum⟩ jednička.

docby.tex
```
416: \def\scaniparam{\futurelet\nextchar\scaniparamA}
417: \def\scaniparamA{\ifx\nextchar\count \expandafter\scaniparamB
418:    \else \def\tmp{\scaniparamB \count=1 }\expandafter\tmp
419:    \fi}
420: \def\scaniparamB \count{\afterassignment\scaniparamC\tempnum}
421: \def\scaniparamC #1^^X#2#3{\def#2{#1}\edef#3{\the\tempnum}}
```

Hlavní práci při vkládání zdrojového textu do dokumentace dělá makro \insinternal s parametry {⟨odkud⟩}{⟨kam⟩}.

docby.tex
```
423: \def\insinternal #1#2{%
424:    \bgroup
425:       \printiabove  % graficke zpracovani zacatku
426:       \setverb \catcode`\"=12 \catcode`\^^M=9 \catcode`\^^I=\active
427:       \mubytein=1 \obeyspaces \continuetrue \tempnum=\tmpA\relax
428:       \def\testline##1#1##2\end{\ifx^^Y##2^^Y\else \nocontinue \fi}%
429:       \ifx^^X#1^^X\def\testline##1\end{\nocontinue}\fi
430:       \loop % preskakovani radku
431:          \ifeof\infile \returninsinternal{Text "#1" not found (\string\count=\the\tempnum)}{}\fi
432:          \readnewline
433:          \expandafter \testline \expandafter^^B\etext ^^E#1\end
434:          \ifcontinue \repeat
435:       \let\lastline=\empty
436:       \continuetrue  \tempnum=\tmpB\relax
437:       \def\testline##1#2##2\end{\ifx^^Y##2^^Y\else \nocontinue \fi}%
438:       \ifx^^X#2^^X\def\testline##1\end{\nocontinue}\fi
439:       \ifx+\startline \printilineA
440:          \expandafter \testline \expandafter ^^B\etext ^^E#2\end
441:          \ifcontinue\else\returninsinternal{}\fi
442:          \readnewline
443:       \else
444:          \readnewline
445:          \ifskipping\ifx\text\empty \readnewline \fi\fi
446:       \fi
447:       \loop  % pretisk radku
448:          \expandafter \testline \expandafter ^^B\etext ^^E#2\end
449:          \ifcontinue
450:             \printilineA
451:             \ifeof\infile \returninsinternal{}\fi
452:             \readnewline \repeat
453:       \ifx+\stopline \printilineA
454:          \ifx\lastline\relax \else \printiline{\lastline}{}\relax\fi
455:       \fi
456:       \global\let\Black=\oriBlack % pokud jsme skoncili vypis uvnitr komentare
457:       \printibelow  % graficke zpracovani konce
458:    \egroup\gdef\ilabellist{}\Black
459: }
```

Makro \insinternal se skládá ze dvou hlavních cyklů. První (na řádcích 430 až 434) čte postupně řádky ze vstupního souboru (makrem \readnewline) a uloží je do makra \etext. V tomto cyklu hledá výskyt textu ⟨odkud⟩ a nic netiskne.

Druhý cyklus na řádku 447 až 452 čte postupně řádky ze vstupního souboru a hledá výskyt textu ⟨kam⟩. V této chvíli tiskne pomocí makra \printilineA.

\readiparamwhy: 25–26 \startline: 26–27 \stopline: 26–27 \scaniparam: 25–26
\scaniparamA: 26 \scaniparamB: 26 \scaniparamC: 26 \insinternal: 25–27

Před prvním cyklem jsou provedeny přípravné práce: nastavení kategorií, fontů, \mubytein. Dále je v přípravné fázi definováno makro \testline se separátorem ⟨*odkud*⟩, pomocí něhož budeme testovat přítomnost textu ⟨*odkud*⟩. Variantní definice makra \testline následují pro speciální případ parametru ⟨*odkud*⟩ (viz uživatelská dokumentace v sekci 2.6). Ukončení cyklu je řízeno podmínkou \ifcontinue. Příkaz \nocontinue provede \continuefalse, ovšem ne vždy. Pokud je zadáno \count>1, tj. \tempnum>1, pak příkaz pouze zaznamená výskyt hledaného textu a sníží \tempnum o jedničku.

docby.tex

```
460: \def\nocontinue{\advance\tempnum by-1 \ifnum\tempnum<1 \continuefalse \fi}
```

Před druhým cyklem v makru \insinternal jsou provedeny podobné přípravné práce jako před prvním, znovu je definováno makro \testline, tentokrát se separátorem ⟨*kam*⟩. Vyhledávání probíhá podobně, jako když jsme hledali ⟨*odkud*⟩.

Pomocí \ifx+\startline testujeme, zda tisknout výchozí řádek. Pomocí \ifx+\stopline testujeme, zda tisknout ukončovací řádek.

Makro \ilabellist obsahuje testování přítomnosti lejblíků deklarovaných příkazem \ilabel.

Trikoidní je makro \returninsinternal {⟨*text*⟩}{⟨*možná-fi*⟩}{⟨*ignoruj*⟩}, které se spustí při dosažení konce čteného souboru. Marko opustí svůj cyklus pomocí parametru ⟨*ignoruj*⟩, který je separován textem \printibelow, takže to přeskočí větší část obsahu makra \insinternal až po řádek 457. Abychom správně opustili vnořené podmínky, jsou přečtena v druhém parametru případná \fi a v makru použita. První parametr obsahuje varovací hlášku, chceme-li vypsat varování. Chceme-li být zticha, je parametr prázdný.

docby.tex

```
462: \def\returninsinternal #1#2#3\printibelow{%
463:    \ifx^^X#1^^X\else
464:      \dbtwarning{#1 in file \inputfilename}\fi
465:    #2\fi\printibelow
466: }
```

Makro \readnewline je naproti tomu jednoduché:

docby.tex

```
467: \def\readnewline {\read\infile to\text \global\advance\lineno by1\relax
468:    {\noswords \xdef\etext{\text}}}
```

Pracujeme s řádkem čteného souboru ve dvou verzích: neexpandovaným \text a expandovaným \etext při \noswords. Tím máme zaručeno, že v \etext nejsou kontrolní sekvence vytvořené encTEXem (pro test přítomnosti ⟨*odkud*⟩ nebo ⟨*kam*⟩ by tam ty sekvence překážely). Verze s encTEXovými sekvencemi \text se použije při tisku.

Makro \printilineA musí mít svou inteligenci: nesmí bezhlavě tisknout prázdné řádky, ale ty tiskne až se zpožděním, následuje-li tisk neprázdného řádku. Tím je zaručeno, že se při \skippingtrue nevytiskne poslední prázdný řádek. Makro \lastline má tři stavy: \empty (na začátku), \relax (po vytištění řádku), ⟨*číslo-řádku*⟩ (je-li předchozí řádek prázdný).

docby.tex

```
470: \def\printilineA {%
471:    \ifskipping\else \ifx\text\empty \def\text{ }\fi\fi % trik pro pripad \skippingfalse
472:    \ifx\text\empty
473:       \ifx\lastline\empty % nacten prvni prazdny radek
474:          \let\lastline=\relax
475:       \else               % nacten pozdejsi prazdny radek
476:          \ifx\lastline\relax \else \printiline{\lastline}{}\relax\fi
477:          \edef\lastline{\the\lineno}%
478:       \fi
479:    \else                  % nacten plny radek
480:       \ifx\lastline\empty \let\lastline=\relax \fi
481:       \ifx\lastline\relax \else \printiline{\lastline}{}\relax\fi
482:       \printiline{\the\lineno}{\text}\relax
483:       \let\lastline=\relax
484:    \fi \ilabellist
485: }
```

Pro uložení deklarací pomocí \ilabel [⟨*lejblík*⟩]{⟨*text*⟩} slouží makro \ilabellist, které musíme nastavit nejprve na prázdnou hodnotu.

\testline: 26–27 \nocontinue: 26–27 \returninsinternal: 26–27 \readnewline: 26–27
\text: 26–27, 36–37 \etext: 26–28 \printilineA: 26–27 \lastline: 26–27 \ilabel: 9, 27–28
\ilabellist: 26–28

docby.tex
```
\def\ilabellist {}
\def\ilabel [#1]#2{{\noswords\edef\act{\noexpand\ilabelee{#1}{#2}}\expandafter}\act}
\def\ilabelee #1#2{\expandafter\def\expandafter\ilabellist\expandafter{%
    \ilabellist \expandafter\testilabel\etext\end{#1}{#2}}
}
```

Makro \ilabel nejprve expanduje své parametry (pomocí \act) a zavolá interní \ilabelee. Toto makro přidá do \ilabellist toto:

```
\expandafter\testilabel\etext\end{⟨lejblík⟩}{⟨text⟩}
```

Makro \testilabel ⟨řádek⟩\end{⟨lejblík⟩}{⟨text⟩} si definuje pomocné makro \tmp se separátorem ⟨text⟩, aby zjistilo, zda je ⟨text⟩ uvnitř ⟨řádek⟩. Pokud se to povede, registruje cíl odkazu pomocí \labeltext.

docby.tex
```
\def\testilabel#1\end#2#3{%
    \def\tmp ##1#3##2\end{\ifx^^Y##2^^Y\else
       \undef{d:#2}\iftrue \defsec{d:#2}{}\labeltext[#2]{\the\lineno}\fi\fi}
    \tmp^^B#1^^E#3\end
}
```

5.4 Příkazy \begtt, \endtt

Makro \begtt a \endtt je podrobně popsáno v TEXbooku naruby na stranách 27 až 30. Makru \startverb dodáme kompletní verbatim text separovaný \endtt. Tento text je dělený znakem ^^M (kategorie 12) na řádky a koncový řádek obsahuje token \end. Makro spustí ve spolupráci s makrem \runttloop cyklus a řádky rozebere, každý řádek zvlášť předá makru \printvline. Na konci cyklu se provede makro \endttloop. To udělá závěrečné činnosti (zavolá \printvbelow, ukončí skupinu) a pomocí makra \scannexttoken otestuje první následující token. Pokud to není \par, není pod \endtt prázdný řádek, takže se provede \noindent.

docby.tex
```
\def\begtt {\bgroup\printvabove
   \setverb \catcode`\"=12 \catcode`\^^M=12 \obeyspaces
   \begtthook\relax \startverb}
{\catcode`\|=0 \catcode`\^^M=12 \catcode`\\=12 %
  |gdef|startverb^^M#1\endtt{|runttloop#1|end^^M}%
  |gdef|runttloop#1^^M{|ifx|end#1 |expandafter|endttloop%
     |else|global|advance|ttlineno by1 %
       |printvline{|the|ttlineno}{#1}|relax|expandafter|runttloop|fi}} %
\def\endttloop#1{\printvbelow\egroup\futurelet\nextchar\scannexttoken}
\long\def\scannexttoken{\ifx\nextchar\par\else\noindent\fi}
```

V numerickém registru \ttlineno je číslo řádku průběžně zvětšované v celém dokumentu. Pokud by někdo chtěl toto číslo využít, může jej nulovat například na začátku každé sekce.

5.5 Jmenné prostory

Každý jmenný prostor si udržuje své \namespacemacro, což je makro s jedním parametrem, které příkazem \namespace{⟨tělo-makra⟩} mimoděk definuje uživatel. Na počátku je \namespacemacro prázdné:

docby.tex
```
\def\namespacemacro#1{}
```

Ke každému jmennému prostoru budeme chtít přiřadit lejblík. Rozhodl jsem se za lejblík považovat výsledek expanze \namespacemacro{@!}. Budu jej nadále značit ⟨nslejblík⟩. Existuje sice určité riziko nejednoznačnosti ⟨nslejblíku⟩, ale předpokládám, že v praxi nenastane.

Každý jmenný prostor už na počátku musí vědět, jaká všechna lokální slova obsahuje, aby jejich výskyt mohl směřovat na místo, kde je deklarace \dl, která může být třeba později než výskyt. Jmenný prostor na svém startu musí tedy do encTEXových tabulek uložit všechna lokální slova a na svém konci vrátit vše pokud možno do původního stavu. Je tedy zřejmé, že není vhodné čekat až na příkaz \dl, ale že je třeba využít soubor \reffile. V prvním průchodu tedy jmenné prostory nemohou být aktivní.

\ilabelee: 28 \testilabel: 28 \begtt: 10, 14, 22, 25, 28 \startverb: 28 \runttloop \endttloop: 28 \scannexttoken: 28 \namespacemacro: 28–29

Po přečtení \reffile má každý jmenný prostor k dispozici makro \ns:⟨nslejblík⟩, které obsahuje seznam všech svých lokálně deklarovaných slov ve formátu

```
\locword{⟨slovo1⟩}\locword{⟨slovo2⟩}\locword{⟨slovo3⟩}...
```

Protože ukládání do encTEXové tabulky je globální, definujeme v rámci duševní hygieny všechna makra s tím spojená globálně. Proto je prostředí \namespace...\endnamespace nezávislé na skupinách TEXu.

Při startu \namespace je třeba definovat \namespacemacro. Původní hodnotu \namespacemacro uložíme do \no:⟨nslejblík⟩, abychom se k němu mohli na konci prostředí \namespace...\endnamespace vrátit. Dále definujeme makro \locword tak, aby uložilo potřebné údaje do encTEXové tabulky a před tím ještě si uložilo stávající významy předefinovávaných kontrolních sekvencí. Pak se prostě spustí \ns:⟨nslejblík⟩.

docby.tex

```
\def\namespace #1{%
   \let\tmp=\namespacemacro
   \gdef\namespacemacro##1{#1}%
   \global\expandafter\let\csname no:\namespacemacro{@!}\endcsname\tmp
   \ewrite{\string\refns{\namespacemacro{@!}}}%
   \def\locword##1{%
      \global\expandafter\let
         \csname\namespacemacro{@!},##1\expandafter\endcsname\csname.##1\endcsname
      \enctextable{##1}{\lword{##1}}}%
   \csname ns:\namespacemacro{@!}\endcsname
}
```

Na konci \endnamespace znovu definujeme makro \locword tentokrát tak, aby vrátilo pozměněným sekvencím původní význam. Pokud původní význam byl „nedefinovaná sekvence“, je potřeba do encTEXové tabulky vložit aspoň \nword, protože zcela odstranit údaj z tabulky nelze. Dále se vrátíme k původní hodnotě \namespacemacro, kterou máme uloženu v \no:⟨nslejblík⟩.

docby.tex

```
\def\endnamespace{\if^^X\namespacemacro{@!}^^X\else
   \def\locword##1{%
      \global\expandafter\let
         \csname.##1\expandafter\endcsname\csname\namespacemacro{@!},##1\endcsname
      \undef{.##1}\iftrue \noactive{##1}\fi}%
   \csname ns:\namespacemacro{@!}\endcsname
   \ewrite{\string\refnsend{\namespacemacro{@!}}}%
   \global\expandafter\let\expandafter\namespacemacro\csname no:\namespacemacro{@!}\endcsname
   \fi
}
```

Uvedená makra pracují s užitečnou zkratkou \ewrite, která zapíše text do \reffile se zpožděním (primitivem \write), ale expanzi udělá hned. Přitom neexpanduje \nb.

docby.tex

```
\def\ewrite#1{{\let\nb=\relax \edef\act{\write\reffile{#1}}\act}}
```

EncTEX od startu jmenného prostoru vkládá tedy místo každého lokálního ⟨slova⟩ kontrolní sekvenci \.⟨slovo⟩, která expanduje na \lword{⟨slovo⟩}. Jakmile se tedy objeví výskyt lokálního slova, pracuje \lword takto:

docby.tex

```
\def\lword#1{\genlongword\tmp{#1}\ilink[@\tmp]{#1}%
    \ewrite{\string\refuseword{\tmp}{\noexpand\the\pageno}}}
\def\genlongword#1#2{\expandafter\def\expandafter#1\expandafter{\namespacemacro{#2}}}
```

Makro \genlongword ⟨tmp⟩{⟨slovo⟩} vytvoří z krátké verze slova dlouhou verzi slova a uloží ji do ⟨tmp⟩. Výskyt ⟨slova⟩ dává o sobě vědět v parametru \ilink i při zápisu do souboru svým dlouhým (jednoznačným) jménem, zatímco krátké jméno se tiskne.

Zbývá zařídit čtení ze souboru \reffile. Makro \refns {⟨nslejblík⟩} se objeví v souboru v místě začátku jmenného prostoru a \refnsend {⟨nslejblík⟩} na konci jmenného prostoru. Mezi nimi se vyskytují \refdg{⟨před⟩}{⟨slovo⟩}{⟨za⟩}{⟨k-slovo⟩}, přičemž si nyní všímáme jen takových výskytů, které mají neprázdné ⟨k-slovo⟩. Právě tyto výskyty zanesl do \reffile příkaz \dl.

\namespace: 11, 15, 28–29 \locword: 29, 38 \endnamespace: 11, 15, 29 \ewrite: 29, 31, 34–35
\lword: 25, 29 \genlongword: 29, 31 \refns: 29–30, 38 \refnsend: 29–30, 38

docby.tex

```
541: \def\refns#1{\edefsec{o:#1}{\currns}
542:    \edef\currns{#1}\undef{ns:\currns}\iftrue \defsec{ns:\currns}{}\fi}
543: \def\refnsend#1{\edef\currns{\csname o:#1\endcsname}}
544: \def\currns{}
```

Makro \refns si zapamatuje předchozí ⟨*nslejblík*⟩, který je uložen v makru \currns, do sekvence \o:⟨*nový-nslejblík*⟩ a definuje pak \currns jako ⟨*nový-nslejblík*⟩. Připraví také výchozí stav makra \ns:⟨*nslejblík*⟩ na prázdnou hodnotu. Makro \refdg pak postupně plní buffer \ns:⟨*nslejblík*⟩ (viz řádky 843 až 846 v definici makra \refdg v sekci 5.9). Konečně makro \refnsend vrátí \currns do stavu, v jakém bylo před vstupem do stávajícího jmenného prostoru.

5.6 \dg a přátelé

Makra \dg, \dl, \dgn, \dgh, \dln, \dlh uloží do \tmpA svůj název, spustí sken parametrů pomocí \dgpar a nakonec se promění ve svou interní verzi pomocí \csname␣ii\tmpA\endcsname.

docby.tex

```
548: \def\dg{\def\tmpA{dg}\dgpar}  \def\dgn{\def\tmpA{dgn}\dgpar}  \def\dgh{\def\tmpA{dgh}\dgpar}
549: \def\dl{\def\tmpA{dl}\dgpar}  \def\dln{\def\tmpA{dln}\dgpar}  \def\dlh{\def\tmpA{dlh}\dgpar}
550:
551: \def\dgpar {\futurelet\nextchar\dgparA}
552: \def\dgparA {\ifx\nextchar[\def\tmp{\dparam}\else\def\tmp{\dparam[]}\fi\tmp}
```

Předchozí makra připraví čtení nepovinného parametru. Hlavní práci provede makro \dparam.

docby.tex

```
554: \def\dparam [#1]#2 {%
555:     \def\printbrackets{}%
556:     \ifx^^X#2^^X\nextdparam{#1}\fi
557:     \def\tmpa{#2}\def\tmpb{}%
558:     \varparam,\tmpa, \varparam.\tmpa. \varparam;\tmpa; \varparam:\tmpa:
559:     \expandafter\managebrackets\tmpa()\end
560:     {\let\nb=\relax
561:     \edef\act{\expandafter\noexpand \csname ii\tmpA\endcsname{#1}{\tmpa}{\printbrackets}}%
562:     \expandafter}\act
563:     \tmpb \if|\expandafter\ignoretwo\tmpA|\expandafter\maybespace\fi
564: }
565: \def\nextdparam#1#2\maybespace\fi{\fi\dparam[#1 ]}
```

Je-li za ukončovací závorkou] mezera, pak je parametr #2 prázdný (je separovaný mezerou). V této situaci se makro \dparam protočí ještě jednou prostřednictvím makra \nextdparam, které sežere obsah zbytku makra \dparam, vloží mezeru dovnitř závorky a spustí \dparam ještě jednou. Nyní už je možné začít parametr #2, tj. ⟨*slovo*⟩ rozdělit na část před první čárkou, tečkou, středníkem nebo dvojtečkou a za za tímto znakem. Část před bude v \tmpa a část za (včetně separátoru) bude v \tmpb. Tuto práci vykoná postupné volání makra \varparam:

docby.tex

```
567: \def\varparam#1{\def\tmp ##1#1##2 {\def\tmpa{##1}\if^^X##2^^X\else
568:         \expandafter\gobblelast\tmpb\end#1##2\fi}%
569:    \expandafter\tmp}
570: \def\gobblelast#1\end#2{\def\tmp##1#2{\def\tmpb{#2##1#1}}\tmp}
```

Makro \varparam⟨*separ*⟩ definuje pomocné makro \tmp#1⟨*separ*⟩#2␣, kterému je předloženo ⟨*slovo*⟩⟨*separ*⟩␣. Je-li #2 prázdné, pak zabral až ⟨*separ*⟩ na konci, takže uvnitř ⟨*slova*⟩ není ⟨*separ*⟩. Pak v \tmpa zůstává ⟨*slovo*⟩. Je-li uvnitř ⟨*slova*⟩ separátor, pak je potřeba doplnit k \tmpb zbytek za separátorem včetně tohoto separátoru. V #2 máme ⟨*zbytek*⟩⟨*separ*⟩ a my potřebujeme do \tmpb uložit stávající obsah \tmpb před kterým předchází ⟨*separ*⟩⟨*zbytek*⟩. Tuto práci udělá \gobblelast, kterému je předložen ⟨*obsah-tmpb*⟩\end⟨*separ*⟩⟨*zbytek*⟩⟨*separ*⟩. Makro definuje \tmp#1⟨*separ*⟩ a předloží mu ⟨*zbytek*⟩⟨*separ*⟩. Je tedy v #1 holý ⟨*zbytek*⟩ a do \tmpb se dostává ⟨*separ*⟩⟨*zbytek*⟩⟨*starý-obsah-tmpb*⟩.

Po rozdělení vyhledání separátoru máme n \tmpa ¡slovo¿, ovšem může obsahovat na konci (). Proto spustíme na řádku 559 makro \managebrackets, které se postará o případné oddělení těchto závorek. Pokud se závorky skutečně oddělily od \tmpa, zůstávají v \printbrackets.

docby.tex
```
572: \def\managebrackets #1()#2\end{\def\tmpa{#1}%
573:    \if|#2|\else\def\printbrackets{()}\fi}
```

Makro \maybespace v závěru činnosti makra \dparam vytiskne za obsahem \tmpb mezeru, ale jen tehdy, když je jméno makra dvoupísmenkové (\dg, \dl) a nenásleduje znak ‘.

docby.tex
```
575: \def\maybespace{\futurelet\tmp\domaybespace}
576: \def\domaybespace{\let\next=\space
577:    \ifx\tmp‘\def\next##1{}\fi
578:    \next}
```

Na řádku 561 vytvoří makro \dparam z původního příkazu \dg* resp. \dl* jeho interní verzi \iidg* resp. \iidl*. Parametry předá expandovány, aby s nimi bylo méně práce. Stačí tedy naprogramovat uvedená interní makra.

Makro \iidg vloží do encTEXové tabulky \sword (je to mírně nadbytečné, totéž se provede na začátku zpracování při čtení \reffile příkazem \refdg). Dále makro vytvoří cíl odkazu tvaru @⟨*slovo*⟩, uloží informaci do \reffile ve formátu \refdg{⟨*před*⟩}{⟨*slovo*⟩}{⟨*za*⟩}{}, vytiskne ⟨*slovo*⟩ zvýrazněné pomocí \printdg a vloží poznámku pod čáru pomocí \printfnote.

docby.tex
```
580: \def\iidg #1#2#3{%
581:     \enctextable{#2}{\sword{#2}}%
582:     \label [@#2]%
583:     \write\reffile{\string\refdg{#1}{#2}{#3}{}}%
584:     \printdg{#1}{#2}{#3}%
585:     \printfnote{#1}{#2}{#3}{#2}%
586: }
```

Makro \iidl nevkládá nic do encTEXové tabulky, vytvoří cíl pomocí \label␣[@⟨*dlouhé-slovo*⟩], zapíše info do \reffile ve formátu \refdg{⟨*před*⟩}{⟨*d-slovo*⟩}{⟨*za*⟩}{⟨*k-slovo*⟩}, vytiskne ⟨*k-slovo*⟩ zvýrazněné pomocí \printdg a vloží poznámku pomocí \printfnote{⟨*před*⟩}{⟨*d-slovo*⟩}{⟨*za*⟩}.

docby.tex
```
587: \def\iidl #1#2#3{%
588:     \genlongword\tmpB{#2}%
589:     \ifx\tmpB\empty \dbtwarning{\string\dl\space#2 outside namespace, ignored}%
590:     \else
591:        \expandafter\label\expandafter [\expandafter @\tmpB]%
592:        \ewrite{\string\refdg{#1}{\tmpB}{#3}{#2}}%
593:        \printdg{#1}{#2}{#3}%
594:        \printfnote{#1}{\tmpB}{#3}{#2}%
595:     \fi
596: }
```

Makra \iidgh a \iidlh dělají to samé jako jejich non-h protějšky, jen netisknou slovo v místě výskytu. Lokálně tedy předefinujeme, aby \printdg nedělalo nic.

docby.tex
```
597: \def\iidgh#1#2#3{{\def\printdg##1##2##3{}\iidg{#1}{#2}{#3}}}
598: \def\iidlh#1#2#3{{\def\printdg##1##2##3{}\iidl{#1}{#2}{#3}}}
```

Makro \iidgn předefinuje makro \.⟨*slovo*⟩, které vyrábí encTEX, tak, že výsledkem expanze je \fword{⟨*před*⟩}{⟨*slovo*⟩}{⟨*za*⟩} (namísto obvyklého \sword{⟨*slovo*⟩}).

docby.tex
```
600: \def\iidgn#1#2#3{\enctextable{#2}{\fword{#1}{#2}{#3}}}
```

Až se \fword spustí (při prvním následujícím výskytu ⟨*slova*⟩), má za úkol provést \iidgh, vytisknout ⟨*slovo*⟩ červeně a vrátit \.⟨*slovo*⟩ do původního stavu.

docby.tex
```
602: \def\fword#1#2#3{\iidgh{#1}{#2}{#3}\printdginside{#2}{#2}}
```

Makro \iidln si uloží stávající význam \.⟨*slovo*⟩ do sekvence \;⟨*slovo*⟩ a předefinuje makro \.⟨*slovo*⟩, které vyrábí encTEX, tak, že výsledkem je \flword{⟨*před*⟩}{⟨*slovo*⟩}{⟨*za*⟩}.

docby.tex
```
604: \def\iidln#1#2#3{%
605:     \global\expandafter\let\csname;#2\expandafter\endcsname\csname.#2\endcsname
606:     \enctextable{#2}{\flword{#1}{#2}{#3}}}
```

\maybespace: 30–31 \iidg: 11, 31 \iidl: 11, 31 \iidgh: 11, 31 \iidlh: 11, 31–32
\iidgn: 11, 31 \fword: 25, 31 \iidln: 11, 31

Makro \flword má za úkol provést \iidlh, vytisknout ⟨slovo⟩ červeně a vrátit význam makra \.⟨slovo⟩ do původního stavu (který je uložen v sekvenci \;⟨slovo⟩. Byl-li tento původní význam nedefinován, je potřeba potlačit další činnost makra \.⟨slovo⟩ registrováním jako \nword{⟨slovo⟩}, protože z encTEXové tabulky už záznam nelze odebrat.

docby.tex

```
608: \def\flword#1#2#3{\iidlh{#1}{#2}{#3}\printdginside{#2}{#2}%
609:     \global\expandafter\let\csname.#2\expandafter\endcsname\csname;#2\endcsname
610:     \undef{.#2}\iftrue \noactive{#2}\fi}
```

5.7 Speciální poznámky pod čarou

Poznámky pod čarou jsou řazeny vedle sebe a obsahují jen slova, která mají na stránce své \dg. Protože toto řešení je vizuálně nekompatibilní s uživatelskými poznámkami pod čarou, jednoduše je zakážeme:

docby.tex

```
614: \let\footnote=\undefined
```

Pro speciální poznámky pod čarou využiji už deklarovaný insert \footins. Problém je, jak odhadnout, kolik zabere vertikálního místa v poznámkách jedno slovo, když jich může být vedle sebe více. Dirty trick z TEXbooku (vkládat inserty ve výšce rovné jistému procentu své šířky) se neujal, neboť zlom často nekonvergoval, ale osciloval. V druhém průchodu poznámky teprve dostávají své seznamy stránek a tyto seznamy se pak mohou dále upřesňovat, což zpětně ovlivní vertikální sazbu. Po její změně se mění seznamy stránek a tak pořád dokola.

Rozhodl jsem se tedy pracovat pouze s průměrným koeficientem poznámek, který budou mít všechny poznámky společný. Tento koeficient získám jako celkový počet řádků poznámek v celém dokumentu dělený počtem poznámek. Každá poznámka pak „překáží“ v hlavním vertikálním seznamu výškou řádku poznámek (10pt) násobenou tímto koeficientem. Stačí tedy nastavit \count\footins.

Aby problém určitě konvergoval, bylo nutné fixovat výše uvedený koeficient po druhém průchodu. Kdybych jej každý následující průchod měnil, zase se nedočkáme konvergence. Získat uvedený koeficient hned po prvním průchodu není rozumné, protože v té době poznámky ještě nemají vedle sebe seznamy stránek. Výchozí koeficient pro první a druhý průchod je tedy nastaven na \count\footins=200 (předpokládám zhruba pět poznámek na řádku).

Pracovat s průměrem místo s každou jednotlivou poznámkou může samozřejmě způsobit, že některé stránky jsou plnější a některé prázdnější. Proto je potřeba mít rezervu ve \skip\footins a vertikálně pružit kolem poznámkové čáry.

docby.tex

```
616: \skip\footins=18pt
617: \dimen\footins=\vsize
618: \count\footins=200
```

V registru \totalfoocount se bude postupně přičítat jednička za každou poznámku a na konci zpracování tam tedy je celkový počet poznámek. V registru \totalfoodim bude na konci zpracování celková výška všech řádků s poznámkami.

docby.tex

```
620: \newcount\totalfoocount
621: \newdimen\totalfoodim
```

Makro \specfootnote {⟨text⟩} vloží do insertu \footins jediný \hbox{⟨text⟩} a připočte jedničku do \totalfoocount.

docby.tex

```
623: \def\specfootnote#1{\insert\footins\bgroup
624:     \let\tt=\ttsmall \rmsmall
625:     \floatingpenalty=20000 \setbox0=\hbox{#1}%
626:     \ht0=10pt \dp0=0pt \box0 \egroup
627:     \global\advance\totalfoocount by1
628: }
```

Protože jsem se rozhodl neměnit výstupní rutinu plainu, musel jsem se „nabourat“ aspoň do její části na tisk poznámek pod čarou. Je to provedeno předefinováním makra \footnoterule výstupní rutiny plainu. Separátor \unvbox\footins způsobí odstranění stejného textu z output rutiny plainu.

\flword: 25, 31–32 \totalfoocount: 32, 36 \totalfoodim: 32–33, 36 \specfootnote: 20, 32

docby.tex

```
630: \def\footnoterule \unvbox\footins {
631:    \vskip-12pt \vfil
632:    \moveright\parindent\vbox{\hsize=\nwidth \hrule
633:    \setbox2=\vbox{\unvbox\footins \unskip
634:        \setbox2=\lastbox
635:        \global\setbox4=\hbox{\unhbox2}
636:        \loop \unskip\unskip\unpenalty
637:            \setbox2=\lastbox
638:            \ifhbox2 \global\setbox4=
639:               \hbox{\unhbox2 \penalty-300\hskip15pt plus5pt \unhbox4}
640:            \repeat}
641:        \setbox2=\vbox{\hbox{} \parskip=0pt
642:        \lineskiplimit=0pt \baselineskip=10pt \raggedright \rightskip=0pt plus7em
643:        \leftskip=0pt \hyphenpenalty=10000 \noindent \Black \unhbox4 }
644:        \global\advance\totalfoodim by\ht2 \unvbox2}
645: }
```

Makro rozebere vertikální seznam insertů `\footins` a poskládá je vedle sebe do boxu 4. Pak nastaví parametry sazby na praporek a vypustí box 4 do horizontálního seznamu (`\noindent`) ukončeném `\endgraf`. Tím jsou ve výstupní rutině poznámky pod čarou vysázeny. Nakonec připočteme `\totalfoodim`.

V závěru zpracování v makru `\bye` (viz řádek 783) zapíšeme do souboru `\reffile` informaci o počtu poznámek ⟨*počet*⟩, o celkové výšce řádků poznámek v dokumentu ⟨*výška*⟩ a přidáme aktuální koeficient příspěvku poznámek do vertikálního seznamu ⟨*koeficient*⟩. Informaci zapisujeme jen tehdy, když je `\indexbuffer` neprázdný, tj. když probíhá aspoň druhý průchod. Kdybychom zapisovali i první průchod, dostali bychom velmi zkreslené informace (poznámky v tu chvíli nemají vedle sebe seznamy stránek). Uvedenou informaci zapsanou v předchozím průchodu přečteme na začátku zpracování makrem `\refcoef` {⟨*koeficient*⟩}{⟨*počet*⟩}{⟨*výška*⟩} a nastavíme podle toho společný koeficient všech poznámek `\count\footins`. Makro změní koeficient z výchozí hodnoty 200 na vypočtenou jen jednou. Při dalších průchodech už zůstává u vypočtené hodnoty. Pomocné makro `\gobblerest` odstraní cifry za desetinou tečkou včetně nápisu `pt`.

docby.tex

```
647: \def\refcoef#1#2#3{%
648:    \ifnum#1=200 % jsme na zacatku tretiho pruchodu
649:       \dimen0=#3 \divide\dimen0 by #2
650:       \multiply \dimen0 by100
651:       \afterassignment\gobblerest \count\footins=\the\dimen0 \end
652:    \else \count\footins=#1
653:    \fi
654:    \message{foot-coef: \the\count\footins}
655: }
656: \def\gobblerest #1\end{}
```

Výstupní rutina `\plainoutput` není změněna. Potřebuji ale uvnitř `\output` potlačit expanzi některých maker, které se objeví v argumentu `\write`. Tato makra jsou tedy uvnitř `\output` nastavena na `\relax`. Aby toto nastavení nezměnilo sazbu záhlaví, je potřeba `\makeheadline` provést před změnou maker a uložit si výsledek do boxu.

docby.tex

```
658: \output={\setbox0=\makeheadline \def\makeheadline{\box0\nointerlineskip}
659:   \let~=\relax \let\nb=\relax \let\TeX=\relax \let\docbytex=\relax \let\_=\relax \let\tt=\relax
660:   \outputhook \plainoutput }
```

5.8 Sekce, podsekce

Nejprve zapíšeme deklarace `\secnum`, `\subsecnum`, `\sectitle`, `\ifsavetoc`. Poslední deklarace připraví uživatelské `\savetocfalse`.

docby.tex

```
665: \newcount\secnum
666: \newcount\subsecnum
667: \newtoks\sectitle
668: \newif\ifsavetoc \savetoctrue
```

\refcoef: 33, 36–38 \gobblerest: 33, 41 \secnum: 18, 33–34, 36 \subsecnum: 18, 33–34
\sectitle: 18–19, 33–35 \ifsavetoc: 18, 33–35 \savetocfalse: 12, 18, 38

Makra \sec a \subsec mají možnost nepovinného parametru [⟨lejblík⟩], za ním může a nemusí být mezera, kterou musíme ignorovat. Na konci parametru ⟨titul⟩ před \par rovněž může a nemusí být mezera, kterou musíme ignorovat. Dá tedy práci parametry správně načíst. Makra si uloží svůj název do \tmpA a spustí proces načítání parametrů pomocí \secparam.

docby.tex

```
\def\sec{\def\tmpA{sec}\futurelet\nextchar\secparam}
\def\subsec{\def\tmpA{subsec}\futurelet\nextchar\secparam}
```

Makro \secparam se vypořádá s případným nepovinným parametrem [⟨lejblík⟩]. Pokud je přítomen, uloží ⟨lejblík⟩ do pomocného makra \seclabel, jinak tam je prázdno. Makro \secparamA se vypořádává s případnou mezerou za hranatou závorkou] a odstraní ji. Makro \secparamB ⟨titul⟩\par načte ⟨titul⟩, ale ten může mít nežádoucí mezeru zcela na konci. S tím se vypořádá makro \nolastspace ve spolupráci s makrem \setparamC. Posledně jmenované makro uloží už od nežádoucí mezery ošetřený ⟨titul⟩ do \sectitle a spustí \iisec resp. \iisubsec.

docby.tex

```
\def\secparam{\ifx\nextchar[%
      \def\tmp[##1]{\def\seclabel{##1}\futurelet\nextchar\secparamA}%
      \expandafter\tmp
    \else \def\seclabel{}\expandafter\secparamB\fi
}
\def\secparamA{\expandafter\ifx\space\nextchar
       \def\tmp{\afterassignment\secparamB\let\next= }\expandafter\tmp
    \else \expandafter\secparamB \fi
}
\def\secparamB #1\par{\nolastspace #1^^X ^^X\end}
\def\nolastspace #1 ^^X#2\end{\ifx^^X#2^^X\secparamC #1\else \secparamC #1^^X\fi}
\def\secparamC #1^^X{\sectitle={#1}\csname ii\tmpA\endcsname}
```

Makro \iisec nejprve nastaví hodnoty \secnum a \subsecnum, dále definuje \makelinks, kde je připravena tvorba odkazů (to použije makro \printsec). Dále zavolá \printsec na vytištění názvu sekce. Poté uloží informace do \reffile ve tvaru \reftocline {⟨secnum⟩}{⟨titul⟩}{⟨strana⟩} Nakonec se provede \mark{⟨secnum⟩␣⟨titul⟩} a vloží se závěrečná mezera pomocí \printsecbelow.

docby.tex

```
\def\iisec{%
    \ifsavetoc \global\advance\secnum by1 \global\subsecnum=0 \fi
    \edef\makelinks{%
        \ifsavetoc \noexpand\savelink[sec:\the\secnum]\fi
        \if^^X\seclabel^^X\else \noexpand\labeltext[\seclabel]{\the\secnum}\fi}
    \expandafter \printsec \expandafter{\the\sectitle}% vlozi horni mezeru, text, nakonec \par
    \ifsavetoc
       \ewrite {\string\reftocline{\the\secnum}{\the\sectitle}{\noexpand\the\pageno}}\fi
    \mark{\ifsavetoc \the\secnum\space \else
          \ifx\emptynumber\empty\else\emptynumber\space\fi\fi \the\sectitle}
    \savetoctrue \printsecbelow
}
```

Makro \iisubsec, které vytváří podsekci, pracuje analogicky, jako makro \iisec.

docby.tex

```
\def\iisubsec {%
    \ifsavetoc \global\advance\subsecnum by1 \fi
    \edef\makelinks{%
        \ifsavetoc \noexpand\savelink[sec:\the\secnum.\the\subsecnum]\fi
        \if^^X\seclabel^^X\else \noexpand\labeltext[\seclabel]{\the\secnum.\the\subsecnum}\fi}
    \expandafter \printsubsec \expandafter{\the\sectitle}% vlozi horni mezeru, text, nakonec \par
    \ifsavetoc  \ewrite
         {\string\reftocline{\the\secnum.\the\subsecnum}{\the\sectitle}{\noexpand\the\pageno}}\fi
    \savetoctrue \printsubsecbelow
}
```

Makro \part bylo zapracováno dodatečně ve verzi Jan. 2009. Registr \partnum uchovává číslo části a makro \thepart toto číslo konvertuje na písmeno.

docby.tex
```
\newcount\partnum
\def\thepart{\ifcase\partnum --\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 +\the\partnum\fi}
```

Makro \part má svou implementaci v makru \iipart podobně jako například makro \sec.

docby.tex
```
\def\part{\def\tmpA{part}\futurelet\nextchar\secparam}
\def\iipart{%
    \ifsavetoc \global\advance\partnum by1 \fi
    \edef\makelinks{%
        \ifsavetoc \noexpand\savelink[sec:\thepart]\fi
        \if^^X\seclabel^^X\else \noexpand\labeltext[\seclabel]{\thepart}\fi}
    \expandafter \printpart \expandafter{\the\sectitle}% vlozi horni mezeru, text, nakonec \par
    \ifsavetoc
       \ewrite {\string\reftocline{}{\the\sectitle}{\noexpand\the\pageno}}\fi
    \savetoctrue \printpartbelow
}
```

5.9 Odkazy, reference

Klikací odkazy řeší makra \savelink [⟨*lejblík*⟩] a a \ilink [⟨*lejblík*⟩]{⟨*text*⟩}. Makro \savelink uloží do sazby cíl odkazu. Cíl odkazu vystrčí do výšky \linkskip nad účaří. Makro \ilink (čti interní link) je dokumentováno v sekci 2.13. Konečně makro \savepglink uloží cíl numerického typu (číslo stránky), který bude využit makrem \pglink při odkazech na stránky.

docby.tex
```
\ifx\pdfoutput\undefined
   \def\savelink[#1]{}
   \def\ilink [#1]#2{#2}
   \def\savepglink{}
   \def\pglink{\afterassignment\dopglink\tempnum=}
   \def\dopglink{\the\tempnum}
   \def\ulink[#1]#2{#2}
\else
   \def\savelink[#1]{\ifvmode\nointerlineskip\fi
      \vbox to0pt{\def\nb{/_}\vss\pdfdest name{#1} xyz\vskip\linkskip}}
   \def\ilink [#1]#2{{\def\nb{/_}\pdfstartlink height 9pt depth 3pt
      attr{/Border[0 0 0]} goto name{#1}}\Blue#2\Black\pdfendlink}
   \def\savepglink{\ifnum\pageno=1 \pdfdest name{sec::start} xyz\relax\fi % viz \bookmarks
       \pdfdest num\pageno fitv\relax}
   \def\pglink{\afterassignment\dopglink\tempnum=}
   \def\dopglink{\pdfstartlink height 9pt depth 3pt
       attr{/Border[0 0 0]} goto num\tempnum\relax\Blue\the\tempnum\Black\pdfendlink}
   \def\ulink[#1]#2{\pdfstartlink height 9pt depth 3pt
   user{/Border[0 0 0]/Subtype/Link/A << /Type/Action/S/URI/URI(#1)>>}\relax
   \Green{\tt #2}\Black\pdfendlink}
\fi
\newdimen\linkskip \linkskip=12pt
```

Uvedená makra jsem definoval zvlášť pro DVI výstup (jako prázdná makra) a zvlášť pro PDF výstup. Až zase tvůrci pdfTEXu změní syntaxi nebo názvy primitivů, bude stačit pozměnit uvedená makra. V makru \ilink je přímo řečeno, že se má použít modrá barva pro vytvoření odkazů a že odkaz má být bez rámečku. Pokud to někomu nevyhovuje, může si makro předefinovat.

Trik s předefinováním \nb (normální backslash) při tvorbě PDF linků vychází ze zkušenosti, že pokud se v názvu linku objeví backslash, některé PDF prohlížeče si s tím neporadí a chovají se podivně. Je tedy nutné, aby argument příkazů \savelink a \ilink byl neexpandovaný.

Makro \savepglink (definice je v předchozím výpisu) je použito v \headline každé stránky, takže vytvoří cíl „nahoře“ na každé stránce. Makro \pglink ⟨*number*⟩ přečte ⟨*number*⟩ (může být ve tvaru numerického registru i přímo jako číslo) a vytvoří link na stránku s tímto číslem. Číslo samotné je vytištěno modře a dá se na ně kliknout. Ke čtení numerického registru je použit primitiv \afterassignment a pomocné makro \dopglink.

\part: 12, 14, 34–35 \iipart: 35 \savelink: 34–36 \ilink: 13, 20–21, 24, 29, 35–36
\linkskip: 18, 35 \savepglink: 19, 35, 40 \pglink: 20–21, 35, 43 \dopglink: 35

V souboru `\jobname.ref` se prostřednictvím makra `\labeltext[`⟨*lejblík*⟩`]{`⟨*text*⟩`}` uloží řádek, který obsahuje `\reflabel {`⟨*lejblík*⟩`}{`⟨*text*⟩`}{`⟨*strana*⟩`}`. Makrem `\reflabel` tyto údaje přečtu a zapíšu do kontrolních sekvencí `^^X`⟨*lejblík*⟩ a `^^Y`⟨*lejblík*⟩. Tyto kontrolní sekvence jsou následně využity v makrech `\numref` a `\pgref`. Za povšimnutí stojí, že pokud je ⟨*text*⟩ prázdný (to jsou například všechny případy dokumentovaných slov), pak kontrolní sekvenci `^^X`⟨*lejblík*⟩ vůbec nedefinuji, abych šetřil pamětí, kterou má TEX rezervovánu na kontrolní sekvence.

docby.tex

```
\def\reflabel #1#2#3{%
   \undef{^^Y#1}\iftrue
      \ifx^^X#2^^X\else\defsec{^^X#1}{#2}\fi
      \defsec{^^Y#1}{#3}%
   \else
      \dbtwarning{The label [#1] is declared twice}%
   \fi
}
\def\numref [#1]{\undef{^^X#1}\iftrue \else \csname^^X#1\endcsname\fi}
\def\pgref [#1]{\undef{^^Y#1}\iftrue-1000\else \csname^^Y#1\endcsname\fi}
```

Makro `\labeltext` `[`⟨*lejblík*⟩`]{`⟨*text*⟩`}`, jak bylo před chvílí řečeno, uloží do souboru potřebné údaje. Jednak zapíše PDF link pomocí makra `\savelink` a dále uloží do souboru `\reffile` potřebné údaje. K tomu je makro `\writelabel` `[`⟨*lejblík*⟩`]{`⟨*text*⟩`}`, které pracuje se zpožděným `\write` (aby číslo strany bylo správně). V okamžiku načtení parametru ⟨*text*⟩ jej potřebuji expandovat, protože tam obvykle bývá něco jako `\the\secnum`. Pro vyřešení tohoto problému jsem na chvíli prohodil parametry (⟨*lejblík*⟩ totiž nechci expandovat) a zavedl pomocné makro `\writelabelinternal` `{`⟨*text*⟩`}{`⟨*lejblík*⟩`}`. První část, tj. `\writelabel{`⟨*text*⟩`}` expanduji pomocí `\edef`.

docby.tex

```
\def\labeltext[#1]#2{\savelink[#1]\writelabel[#1]{#2}}
\def\writelabel[#1]#2{\edef\tmp{\noexpand\writelabelinternal{#2}}\tmp{#1}}
\def\writelabelinternal#1#2{\write\reffile{\string\reflabel{#2}{#1}{\the\pageno}}}
```

Makro `\label` je už definováno jednoduše jako „prázdný“ `\labeltext`.

docby.tex

```
\def\label[#1]{\labeltext[#1]{}}
```

Makro `\cite` `[`⟨*lejblík*⟩`]` vytiskne klikatelný text. Při chybném ⟨*lejblíku*⟩ vytiskne varování na terminál. Makro je dokumentováno v sekci 2.13.

docby.tex

```
\def\cite[#1]{\ifnum \pgref[#1]=-1000
      \dbtwarning{label [#1] is undeclared}\Red??\Black
   \else \edef\tmp{\numref[#1]}%
      \ifx\tmp\empty \edef\tmp{\pgref[#1]}\fi
      \ilink[#1]{\tmp}%
   \fi
}
```

S odkazy souvisí makro `\api` `{`⟨*slovo*⟩`}`, které vloží `\label[+`⟨*slovo*⟩`]` dá o sobě vědět ještě jednou do `\reffile`.

docby.tex

```
\def\api #1{\label[+#1]\write\reffile{\string\refapiword{#1}}}
\def\apitext{$\succ$}
```

Makro `\apitext` obsahuje text tištěný vedle ⟨*slova*⟩ do obsahu a rejstříku.

Při činnosti makra `\bye` zapíšeme do souboru `\reffile` údaje pro `\refcoef` (řádek 783) a dále se zabýváme testem konzistence referencí.

docby.tex

```
\def\bye{\par\vfill\supereject
   \ifx\indexbuffer\empty \else % jsme ve druhem a dalsim pruchodu
      \immediate\write\reffile{\string\refcoef
         {\the\count\footins}{\the\totalfoocount}{\the\totalfoodim}}
      \immediate\closeout\reffile
      \setrefchecking \continuetrue \input \jobname.ref
      \ifcontinue  \indexbuffer \relax \fi
      \ifcontinue \ifx\text\tocbuffer \else
```

\reflabel: 36–38 \numref: 12, 13, 36 \pgref: 12, 13, 20–21, 36 \labeltext: 13, 28, 34–36
\writelabel: 36 \writelabelinternal: 36 \label: 12, 13, 31, 36 \cite: 13, 9, 12, 36
\api: 12, 13, 20–21, 36, 38 \apitext: 12, 21, 36 \bye: 7, 11, 33, 36–37

```
789:            \continuefalse \dbtwarning{toc-references are inconsistent, run me again}\fi
790:        \fi
791:        \ifcontinue \immediate\write16{DocBy.TeX: OK, all references are consistent.}\fi
792:    \fi
793:    \end
794: }
```

Test konzistence vypadá následovně: nejprve uzavřeme zápis do souboru `\reffile`, pak pomocí `\setrefchecking` předefinujeme kontrolní sekvence vyskytující se v `\reffile` a soubor znovu načteme. Nyní makra v něm napsaná dělají test a pokud narazí na problém, provedou `\continuefalse`. Můžeme tedy pomocí `\ifcontinue` zjistit, jak test dopadl. Po přečtení souboru je potřeba udělat ještě důkladnou kontrolu všech automatických odkazů. Proč je tato kontrola vyřešena vypuštěním `\indexbuffer` do vstupní fronty bude jasné po prostudování makra `\setrefchecking`.

docby.tex

```
795: \def\setrefchecking{\catcode`\"=12
796:    \def\refcoef##1##2##3{}
797:    \def\reflabel##1##2##3{\def\tmp{##3}\let\next=\relax
798:        \expandafter\ifx\csname^^Y##1\endcsname \tmp
799:           \ifx^^X##2^^X\else
800:              \def\tmp{##2} \expandafter \ifx \csname^^X##1\endcsname \tmp \else
801:                 \continuefalse
802:                 \dbtwarning{text references are inconsistent, run me again}
803:                 \let\next=\endinput
804:           \fi\fi
805:        \else
806:           \continuefalse
807:           \dbtwarning{page references are inconsistent, run me again}
808:           \let\next=\endinput
809:        \fi\next}
810:    \def\refuseword##1##2{\expandafter \ifx\csname -##1\endcsname \relax
811:       \defsec{-##1}{##2}\else \edefsec{-##1}{\csname -##1\endcsname,##2}\fi}
812:    \def\refdg##1##2##3##4{\addtext\ptocentry @{##2}{##4}\to\tocbuffer}
813:    \let\text=\tocbuffer \def\tocbuffer{}
814:    \def\,##1{\let##1=\relax}\indexbuffer
815:    \def\,##1{\edef\tmp{\expandafter\ignoretwo \string ##1}%
816:        \expandafter\ifx \csname w:\tmp\endcsname ##1\else
817:           \continuefalse
818:           \dbtwarning{auto-references are inconsistent, run me again}
819:           \expandafter\ignoretorelax \fi}
820: }
821: \def\ignoretorelax #1\relax{}
```

Zde předefinujeme makro `\refcoef`, aby nedělalo nic. Dále nová verze `\reflabel` kontroluje, zda odkaz je na stejné stránce, jako byl a má stejný text. Nové makro `\refuseword` pracuje jako jeho originální protějšek, jen místo sekvencí `w:`⟨*slovo*⟩ plní sekvence `-`⟨*slovo*⟩. Tyto sekvence už známe, nyní je využijeme jinak. Šetříme pamětí TEXu, proto nezakládáme sekvence nové. Nejprve je nutné těmto sekvencím nastavit výchozí hodnotu `\relax`, což je provedeno na řádku 797. Pak znovu předefinuji sekvenci `\,`, aby provedla test shodnosti sekvence `w:`⟨*slovo*⟩ se sekvencí `-`⟨*slovo*⟩ a v makru `\bye` na řádku 782 spustím tento test expandováním makra `\indexbuffer\relax`. Když makro najde nekonzistenci, ohlásí chybu a uteče pomocí `\ignoretorelax`. Dále je předefinováno makro `\refdg`, aby pouze zapisovalo do `\tocbuffer`. Ostatní makra z `\reffile` také zapisují do `\tocbuffer`. Stávající verzi `\tocbuffer` uložíme do `\text` a `\tocbuffer` se při načtení `\reffile` vytvoří znovu. Na řádku 788, zda se nezměnil obsah.

5.10 Tvorba obsahu, rejstříku a záložek

Obsah i rejstřík se mohou pomocí `\dotoc` a `\doindex` objevit kdekoli v dokumentu (třeba na začátku, na konci, uprostřed…). Musíme být připraveni je kdykoli vytisknout. Soubor `\reffile` z minulého běhu můžeme otevřít ke čtení jen na začátku, pak jej mažeme a začínáme znova zapisovat. Při čtení ze souboru `\reffile` tedy musíme uložit všechny potřebné informace k sazbě obsahu i rejstříku. Používáme na to makro `\tocbuffer` a `\indexbuffer`. Na začátku tyto „buffery“ vyprázdníme. Makro

`\setrefchecking`: 36–37 `\ignoretorelax`: 37

\addtext ⟨text⟩\to⟨buffer⟩ budeme používat na vkládání ⟨textu⟩ do ⟨bufferu⟩, čímž buffery postupně naplníme.

docby.tex
```
825: \def\tocbuffer{}
826: \def\indexbuffer{}
827: \def\addtext #1\to#2{\expandafter\gdef\expandafter#2\expandafter{#2#1}}
```

V souboru \reffile se vyskytují tyto příkazy:

```
\reftocline{⟨číslo⟩}{⟨název⟩}{⟨strana⟩}  % údaje o sekci a subsekci pro obsah
\refdg{⟨před⟩}{⟨slovo⟩}{⟨za⟩}{⟨k-slovo⟩}  % údaj o použití \dg, \dl
\refapiword{⟨slovo⟩}                     % údaj o výskytu \api{⟨slovo⟩}
\refuseword{⟨slovo⟩}{⟨strana⟩}           % údaj o přímém výskytu ⟨slova⟩
\reflabel{⟨lejblík⟩}{⟨text⟩}{⟨strana⟩}   % viz sekci 5.9, odkazy, reference
\refcoef{⟨koeficient⟩}{⟨počet⟩}{⟨výška⟩} % viz sekci 5.7, spec. poznámky
\refns{⟨nslejblík⟩}                      % viz sekci 5.5, jmenné prostory
\refnsend{⟨nslejblík⟩}                   % viz sekci 5.5, jmenné prostory
```

Při čtení souboru \reffile ukládáme potřebné údaje do bufferů. Nejprve se zaměříme na **obsah** a definujeme \reftocline {⟨číslo⟩}{⟨název⟩}{⟨strana⟩}.

docby.tex
```
829: \def\reftocline#1#2#3{\def\currb{#1}%
830:    \istocsec#1.\iftrue \def\currsecb{#1}\else \addbookmark\currsecb \fi
831:    \addtext\dotocline{#1}{#2}{#3}\to\tocbuffer}
```

V \tocbuffer tedy máme postupně údaje o všech sekcích a podsekcích v za sebou jdoucích sekvencích \dotocline {⟨číslo⟩}{⟨název⟩}{⟨strana⟩}. Mezi sekcí a subsekcí rozlišíme jen podle toho, zda parametr ⟨číslo⟩ obsahuje tečku. K tomu slouží pomocné makro \istocsec.

docby.tex
```
833: \def\dotocline#1#2#3{\par
834:    \istocsec#1.\iftrue \ptocline{#1}{#2}{#3}\else \ptocsubline{#1}{#2}{#3}\fi}
835: \def\istocsec#1.#2\iftrue{\if^^X#2^^X}
```

Kdybychom spustili makro \tocbuffer, dostaneme obsah. Ale ten se neskládá jen z údajů o sekcích a podsekcích. Ještě je potřeba přečíst \refdg a \refapiword, abychom mohli vkládat do obsahu i údaje o dokumentovaných slovech.

docby.tex
```
837: \def\refdg#1#2#3#4{%
838:     \edefsec{-#2}{#1\noexpand\right\if!#4!#3\fi}
839:     \expandafter\addtext\csname-#2\endcsname,\to\indexbuffer
840:     \addbookmark\currb
841:     \addtext\ptocentry @{#2}{#4}\to\tocbuffer
842:     \ifx^^X#4^^X\enctextable{#2}{\sword{#2}}    % slovo je z \dg
843:     \else \expandafter\def\csname ns:\currns    % slovo je z \dl
844:           \expandafter\expandafter\expandafter\endcsname
845:           \expandafter\expandafter\expandafter
846:            {\csname ns:\currns\endcsname \locword{#4}}
847:     \fi
848: }
849: \def\refapiword#1{\addbookmark\currb \addtext\ptocentry +{#1}{}\to\tocbuffer}
```

Makro \refdg pracuje s parametry {⟨před⟩}{⟨slovo⟩}{⟨za⟩}{⟨k-slovo⟩}, kde ⟨před⟩ je text před slovem, ⟨slovo⟩ je dlouhé slovo, ⟨za⟩ obsahuje případné závorky (). Je-li dlouhé slovo rozdílné od krátkého slova (při použití \dl), obsahuje ⟨k-slovo⟩ krátké slovo, jinak je tento parametr prázdný. Makro \refdg ukládá informace nejen do \tocbuffer, ale také do \indexbuffer. Rovněž při prázdném ⟨k-slovo⟩ makro ukládá \sword do encTEXové tabulky a při neprázdném ⟨k-slovo⟩ makro cosi kutí se jmennými prostory. Nyní je ale naše pozornost věnována tvorbě obsahu. Ten vytvoří makro \dotoc.

docby.tex
```
851: \def\dotoc{\bgroup \savetocfalse \sec \tittoc \par \smallskip
852:    \leftskip=\parindent \rightskip=\parindent plus .5\hsize
853:    \tochook \tocbuffer \par\egroup}
```

\addtext: 24, 37–38, 41 \reffile: 24, 28–29, 31, 33–34, 36–39, 42 \reftocline: 34–35, 38–39
\tocbuffer: 36–38, 40 \dotocline: 38, 40 \istocsec: 38 \refdg: 21, 29–31, 37–39
\refapiword: 36, 38 \dotoc: 7, 4, 14, 37–38

Rejstřík je vybudován z bufferu \indexbuffer, ve kterém je seznam deklarovaných slov v dokumentu. Každé slovo je v bufferu zapsáno jako kontrolní sekvence (to zabere v paměti TEXu nejmíň místa) a je odděleno od další sekvence oddělovačem. Před zatříděním podle abecedy jsou položky v \indexbuffer odděleny čárkami za položkami, po zatřídění jsou položky odděleny \, před položkami. Takže obsah \indexbuffer vypadá zhruba takto:

```
před zatříděním:  \-⟨slovo1⟩ , \-⟨slovo2⟩ , \-⟨slovo3⟩ , \-⟨slovo4⟩ , ...
po zatřídění:     \, \-⟨slovoA⟩ \, \-⟨slovoB⟩ \, \-⟨slovoC⟩ \, \-⟨slovoD⟩ ...
```

Zde zápis \-⟨*slovo*⟩ znamená jednu kontrolní sekvenci. Každá taková kontrolní sekvence je makrem tvaru ⟨*před*⟩\right⟨*za*⟩. To zařídí řádek 838. Rejstřík vytiskneme makrem \doindex.

docby.tex

```
855: \def\doindex {\par\penalty0
856:    \calculatedimone
857:    \ifdim\dimen1<7\baselineskip \vfil\break \fi
858:    \sec \titindex \par
859:    \ifx\indexbuffer\empty
860:        \dbtwarning {index is empty, run me again}
861:    \else
862:        \message{DocBy.TeX: sorting index...}
863:        \sortindex
864:        \indexhook
865:        \vskip-\baselineskip
866:        \begmulti 2 \rightskip=0pt plus5em \parfillskip=0pt plus2em
867:          \widowpenalty=0 \clubpenalty=0
868:          \let\,=\doindexentry \indexbuffer \endmulti
869:    \fi
870: }
```

Příkaz \calculatedimone s následujícím testem \dimen1 souvisí se sazbou do dvou sloupců, což necháme na sekci 5.13. Makro tedy založí příkazem \sec sekci nazvanou \titindex a pokud je \indexbuffer neprázdný, spustí sazbu rejstříku. Nejprve se příkazem \sortindex setřídí \indexbuffer podle abecedy (viz sekci 5.11). Pak makro \doindex založí dvousloupcovou sazbu (\begmulti␣2) a oddělovači \, přidělí význam \doindexentry. Nakonec vypustí \indexbuffer do vstupní fronty, takže další práci opakovaně provede makro \doindexentry \-⟨*slovo*⟩, které se postará o tisk slova v rejstříku.

docby.tex

```
871: \def\doindexentry #1{%
872:       \edef\tmp{\expandafter\ignoretwo \string #1}%
873:       \expandafter \remakebackslash \tmp\end
874:       \expandafter \printindexentry \expandafter {\tmp}%
875: }
876: \def\remakebackslash#1#2\end{\if#1\nb \def\tmp{\nb#2}\fi}
877: \def\ignoretwo #1#2{}
```

Makro \doindexentry pomocí \ignoretwo odstraní z kontrolní sekvence \-⟨*slovo*⟩ úvodní dva znaky \-, takže v \tmp zůstane ⟨*slovo*⟩. Pokud ⟨*slovo*⟩ začíná backslashem, uděláme z něj makrem \remakebackslash sekvenci \nb, neboť přímý backslash není uložen v PDF odkazech (zlobí některé PDF prohlížeče, viz sekci 5.9. Nakonec se vytiskne položka v rejstříku už známým makrem \printindexentry.

Při tvorbě **strukturovaných záložek** je potřeba vědět, kolik má každý uzel potomků. Tento údaj je počítaný při čtení \reffile voláním makra \addbookmark ⟨*uzel*⟩ (viz makra \reftocline a \refdg). Parametr ⟨*uzel*⟩ může být číslo sekce, nebo dvojčíslí ⟨*sekce*⟩.⟨*podsekce*⟩. V makru \currb je ⟨*uzel*⟩, ke kterému je potřeba přičítat potomka a \currsecb je případný nadřazený ⟨*uzel*⟩ sekce. Makro \addbookmark připočte jedničku k hodnotě makra \bk:⟨*uzel*⟩.

docby.tex

```
879: \def\addbookmark#1{\undef{bk:#1}\iftrue\defsec{bk:#1}{1}%
880:    \else \tempnum=\csname bk:#1\endcsname\relax
881:          \advance\tempnum by1
882:          \edefsec{bk:#1}{\the\tempnum}
883:    \fi}
884: \def\currb{} % vychozi hodnota <uzel> pro jistotu
```

\indexbuffer: 33, 36–42 \doindex: 7, 4, 14, 37, 39, 44 \doindexentry: 39 \ignoretwo: 30, 37, 39, 41 \remakebackslash: 39 \addbookmark: 38–39 \currb: 38–39 \currsecb: 38

Makro \bookmarks založí skupinu, předefinuje \dotocline a \ptocentry (tj. makra obsažená v \tocbuffer) vloží první záložku s názvem dokumentu a spustí \tocbuffer.

docby.tex

```
\def\bookmarks{\ifx\pdfoutput\undefined \else
   \bgroup
     \def\dotocline##1##2##3{%
         \undef{bk:##1}\iftrue \tempnum=0 \else \tempnum=\csname bk:##1\endcsname\relax\fi
         \if^^X##1^^X\advance\partnum by1
            \setoutline[sec:\thepart]{##2}{\opartname\space\thepart: }%
         \else \setoutline[sec:##1]{##2}{}\fi}
     \def\ptocentry##1##2##3{\edef\tmpb{\ifx^^X##3^^X##2\else##3\fi}%
         \tempnum=0 \setoutline[##1##2]{\tmpb}{}}%
     \def\nb{\string\\}\def\TeX{TeX}\def\docbytex{DocBy.TeX}\def\_{_}\def\tt{}\def~{ }%
     \def\unskip{}\bookmarkshook
     \ifx\headtile\empty \else
        \tempnum=0 \setoutline[sec::start]{...\headtitle\empty...}{}\fi % viz \savepglink
     \tocbuffer
   \egroup \fi
}
```

Makro \setoutline [⟨lejblík⟩]{⟨text⟩}{⟨prefix⟩} vytvoří záložku ⟨prefix⟩⟨text⟩ a prolinkuje ji s cílem označeným ⟨lejblík⟩. V \tempnum musí být uložen počet potomků záložky.

docby.tex

```
\def\setoutline[#1]#2#3{{\def\nb{/_}\xdef\tmp{#1}}%
   \def\tmpa{\pdfoutline goto name{\tmp} count -\tempnum}%
   \cnvbookmark{\tmpa{#3\nobraces#2{\end}}}%
}
\def\cnvbookmark#1{#1}  % zadna konverze
\def\nobraces#1#{#1\nobrA}
\def\nobrA#1{\ifx\end#1\empty\else\nobraces#1\fi}
```

V tomto makru je použito konverzní makro \cnvbookmark, které je implicitně neaktivní. Uživatel může například nastavit \let\cnvbookmark=\lowercase a nechat konvertovat pomocí \lccode znak č na c, znak ž na z, atd. Nastavení \lccode musí mít v \bookmarkshook.

Dále je text před vložením do záložky podroben konverzi \nobraces, která ve spolupráci s makrem \nobrA sundá případné závorky {}. Takže, pokud máme třeba {\tt␣text}␣v␣\TeX{}u, po konverzi dostáváme text␣v␣TeXu.

5.11 Abecední řazení rejstříku

Tuto práci provede makro \sortindex. Původně bylo v DocBy.TEXu implementováno algoritmem bubblesort, což vyšlo na šest řádků makrokódu (prezentováno na tutoriálu TEXperience 2008), ale pro větší rejstříky to bylo pomalé. Např. pro rejstřík tohoto dokumentu to vygenerovalo 52 tisíc dotazů na porovnání a trvalo to asi dvě vteřiny. Můj syn Mirek byl pozorný posluchač tutoriálu, takže nabyté znalosti okamžitě využil a přepsal třídicí makro na mergesort. Ten na stejně velkém rejstříku generuje 1600 dotazů na porovnání, tedy třicetkrát méně. Cena za to je skutečnost, že makro už nemá jen šest řádků, ale je mírně komplikovanější. Od možnosti použít quicksort jsme upustili, protože implementace by vyžadovala vyšší paměťové nároky na inputstack TEXu.

Nejprve deklarujeme podmínku pro výsledek srovnání dvou položek \ifAleB a vytvoříme pomocná makra \nullbuf, \return a \fif. Pomocné makro \return ve spolupráci se zakrytým \fi uvnitř \fif budeme používat pro únik z košatých hluboce vnořených podmínek typu \if..\else..\fi. Jak uvidíte, makro pracuje na úrovni expandprocesoru a nebude potřeba psát žádné \expandafter.

docby.tex

```
\newif\ifAleB
\def\nullbuf{\def\indexbuffer{}}
\def\return#1#2\fi\relax{#1} \def\fif{\fi}
```

Makro \sortindex vypustí do vstupní fronty celý \indexbuffer, přimaluje k němu \end,\end, pronuluje \indexbuffer a spustí \mergesort.

docby.tex

```
916: \def\sortindex{\expandafter\nullbuf
917:   \expandafter\mergesort\indexbuffer\end,\end
918: }
```

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 (\indexbuffer odděluje každou položku čárkou). Makro \mergesort v tomto případě projde seznam a vytvoří seznam zatříděných dvoupoložkových skupin, uložený zpětně v \indexbuffer. V dalším průchodu znovu vyvrhne \indexbuffer do vstupní fronty, vyprázdní ho a startuje znovu. Nyní vznikají čtyřpoložkové zatříděné skupiny. Pak osmipoložkové atd. V závěru (na řádku 929) je první skupina celá setříděná a druhá obsahuje \end, tj. všechny položky jsou už setříděné v první skupině, takže stačí ji uložit do \indexbuffer a ukončit činnost. Pomocí \gobblerest odstraníme druhé \end ze vstupního proudu.

docby.tex

```
919: \def\mergesort #1#2,#3{%
920:    \ifx,#1                       % prazdna-skupina,neco,  (#2=neco #3=pokracovani)
921:       \addtext#2,\to\indexbuffer    % dvojice skupin vyresena
922:       \return{\fif\mergesort#3}%    % \mergesort pokracovani
923:    \fi
924:    \ifx,#3                       % neco,prazna-skupina,  (#1#2=neco #3=,)
925:       \addtext#1#2,\to\indexbuffer  % dvojice skupin vyresena
926:       \return{\fif\mergesort}%      % \mergesort dalsi
927:    \fi
928:    \ifx\end#3                    % neco,konec (#1#2=neco)
929:       \ifx\empty\indexbuffer                    % neco=kompletni setrideny seznam
930:          \edef\indexbuffer{\napercarky#1#2\end}%  % vlozim \, mezi polozky
931:          \return{\fif\fif\gobblerest}%            % koncim
932:       \else                      % neco=posledni skupina nebo \end
933:          \return{\fif\fif \expandafter\nullbuf    % spojim \indexbuffer+neco a cele znova
934:                \expandafter\mergesort\indexbuffer#1#2,#3}%
935:    \fi\fi                       % zatriduji: p1+neco1,p2+neco2, (#1#2=p1+neco1 #3=p2)
936:    \isAleB #1#3\ifAleB          % p1<p2
937:       \addtext#1\to\indexbuffer   % p1 do bufferu
938:       \return{\fif\mergesort#2,#3}%         % \mergesort neco1,p2+neco2,
939:    \else                        % p1>p2
940:       \addtext#3\to\indexbuffer   % p2 do bufferu
941:       \return{\fif\mergesort#1#2,}%         % \mergesort p1+neco1,neco2,
942:    \fi
943:    \relax % zarazka, na ktere se zastavi \return
944: }
```

Jádro \mergesort vidíme na řádcích 936 až 941. 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 920 až 926 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.

Vlastní srovnání dvou položek dělá makro \isAleB ⟨*polA*⟩⟨*polB*⟩. Položky jsou tvaru kontrolní sekvence \-⟨*slovoA*⟩ a \-⟨*slovoB*⟩. Makro konvertuje své parametry pomocí \string na řadu znaků a expanduje na \testAleB ⟨*slovoA*⟩\relax⟨*slovoB*⟩\relax. Navíc je na tento test aplikováno \lowercase, neboť nerozlišujeme při řazení mezi velkými a malými písmeny.

docby.tex

```
945: \def\isAleB #1#2{%
946:    \edef\tmp{\expandafter\ignoretwo\string#1&0\relax\expandafter\ignoretwo\string#2&1\relax}%
947:    \lowercase \expandafter {\expandafter \testAleB \tmp}%
948: }
```

\mergesort: 40–41 \isAleB: 41

Makro \testAleB ⟨slovoA⟩\relax⟨slovoB⟩\relax zjistí, zda je ⟨slovoA⟩ menší než ⟨slovoB⟩. Makro volá samo sebe, pokud jsou první porovnávané znaky stejné. Rekurze určitě skončí, neboť na řádku 946 jsou k porovnávaným slovům připojeny různé ocasy.

docby.tex

```
\def\testAleB #1#2\relax #3#4\relax {%
   \ifx #1#3\testAleB #2\relax #4\relax \else
       \ifnum `#1<`#3 \AleBtrue \else \AleBfalse \fi
   \fi
}
```

Makro \napercarky vloží mezi položky do \indexbuffer separátory \,. To se provede uvnitř \edef\indexbuffer na řádku 930.

docby.tex

```
\def\napercarky#1{\ifx#1\end \else
    \noexpand\,\noexpand#1\expandafter\napercarky
  \fi
}
```

5.12 Transformace seznamu stránek

Každý výskyt ⟨slova⟩ uloží do \reffile údaj \refuseword {⟨slovo⟩}{⟨strana⟩}, který přečteme na začátku zpracování v dalším průchodu:

docby.tex

```
\def\refuseword#1#2{%
   \expandafter \ifx\csname w:#1\endcsname \relax
      \defsec{w:#1}{#2}
   \else
      \edefsec{w:#1}{\csname w:#1\endcsname,#2}
   \fi
}
```

V sekvenci \w:⟨slovo⟩ tedy máme seznam stránek s výskyty ⟨slova⟩, stránky jsou odděleny čárkami. Seznam může vypadat třeba takto:

```
2,5,5,10,11,12,12,13,13,13,27
```

Cílem je takovýto seznam stránek vytisknout ve formátu 2,␣5,␣10--13,␣27, tj. odstranit duplicity a nahradit souvisle jdoucí řadu stránek zápisem ve tvaru ⟨od⟩--⟨do⟩. Tuto práci dělá makro \listofpages {⟨slovo⟩}, které předhodí makru \transf expandovaný seznam stránek ukončený ,0,.

docby.tex

```
\def\listofpages#1{%
    \expandafter\expandafter\expandafter \transf\csname w:#1\endcsname,0,%
}
```

Makro \transf vyloučí ze seznamu stránek ty, které jsou rovny \dgnum a \apinum. Nechceme totiž, aby se v seznamu opakovala hlavní stránka ⟨slova⟩ a podtržená stránka. Tyto stránky jsou vytištěny už dříve. Deklarujeme uvedené registry:

docby.tex

```
\newcount\apinum
\newcount\dgnum
\newcount\tempnum
\newif\ifdash
\newif\iffirst
```

Kromě toho jsme deklarovali pomocné \tempnum (aktuálně zpracovávaná stránka), \ifdash (zda zpracováváme souvislou skupinu stránek a vytiskli jsme --) a \iffirst (zda vkládáme první číslo, tj. není nutné vložit čárku).

Makro \transf ⟨seznam-stránek⟩,0, spustí cyklus pomocí \cykltransf.

docby.tex

```
\def\transf{\dashfalse \firsttrue \tempnum=-100 \bgroup \cykltransf}

\def\cykltransf #1,{\ifnum #1=\apinum \else \ifnum #1=\dgnum \else
   \ifnum #1=0 \let\cykltransf=\egroup
```

\testAleB: 41–42 \napercarky: 41–42 \refuseword: 24, 29, 37–38, 42 \listofpages: 20–21, 42
\dgnum: 20–21, 42 \apinum: 20–21, 42 \tempnum: 26–27, 35, 39–40, 42–43 \ifdash: 42–43
\iffirst: 42–43 \transf: 42 \cykltransf: 42–43

```
983:       \ifdash \pglink\the\tempnum\relax \fi
984:    \else
985:       \ifnum #1=\tempnum % cislo se opakuje, nedelam nic
986:       \else
987:          \advance\tempnum by 1
988:          \ifnum #1=\tempnum  % cislo navazuje
989:             \ifdash \else
990:                --\dashtrue
991:             \fi
992:          \else               % cislo nenavazuje
993:             \ifdash
994:                \advance\tempnum by-1
995:                \pglink\the\tempnum \relax\dashfalse, \pglink#1\relax
996:             \else
997:                \carka \pglink#1\relax
998:    \fi\fi\fi\fi
999:    \tempnum=#1 \fi\fi \cykltransf
1000: }
1001: \def\carka{\iffirst \firstfalse \else, \fi}
```

Makro \cykltransf je takový malý stavový automat. Věřím, že mu čtenář porozumí bez dalšího komentáře.

5.13 Více sloupců

Sazba do více sloupců je kompletně převzata z TEXbooku naruby, strany 244–246.

docby.tex

```
1005: \newdimen\colsep  \colsep=\parindent % horiz. mezera mezi sloupci
1006: \def\roundtolines #1{%% zaokrouhlí na celé násobky vel. rádku
1007:    \divide #1 by\baselineskip \multiply #1 by\baselineskip}
1008: \def\corrsize #1{%% #1 := #1 + \splittopskip - \topskip
1009:    \advance #1 by \splittopskip \advance #1 by-\topskip}
1010:
1011: \def\begmulti #1 {\par\bigskip\penalty0 \def\Ncols{#1}
1012:    \splittopskip=\baselineskip
1013:    \setbox0=\vbox\bgroup\penalty0
1014:    \advance\hsize by\colsep
1015:    \divide\hsize by\Ncols  \advance\hsize by-\colsep}
1016: \def\endmulti{\vfil\egroup \setbox1=\vsplit0 to0pt
1017:    \calculatedimone
1018:    \ifdim \dimen1<2\baselineskip
1019:      \vfil\break \dimen1=\vsize \corrsize{\dimen1} \fi
1020:    \dimen0=\Ncols\baselineskip \advance\dimen0 by-\baselineskip
1021:    \advance\dimen0 by \ht0 \divide\dimen0 by\Ncols
1022:    \roundtolines{\dimen0}%
1023:    \ifdim \dimen0>\dimen1 \splitpart
1024:    \else \makecolumns{\dimen0} \fi
1025:    \ifvoid0 \else \errmessage{ztracený text ve sloupcích?} \fi
1026:    \bigskip}
1027: \def\makecolumns#1{\setbox1=\hbox{}\tempnum=0
1028:    \loop \ifnum\Ncols>\tempnum
1029:       \setbox1=\hbox{\unhbox1 \vsplit0 to#1 \hss}
1030:       \advance\tempnum by1
1031:    \repeat
1032:    \hbox{}\nobreak\vskip-\splittopskip \nointerlineskip
1033:    \line{\unhbox1\unskip}}
1034: \def\splitpart{\roundtolines{\dimen1}
1035:    \makecolumns{\dimen1} \advance\dimen0 by-\dimen1
1036:    \vskip 0pt plus 1fil minus\baselineskip \break
1037:    \dimen1=\vsize \corrsize{\dimen1}
1038:    \ifvoid0 \else
1039:       \ifdim \dimen0>\dimen1 \splitpart
1040:       \else \makecolumns{\dimen0} \fi \fi}% TBN
```

Zde navíc řešíme problém, že na začátku přepnutí do dvou sloupců pomocí \begmulti 2 si makro na řádku 1018 zkontroluje, zda není blízko dna stránky a v takovém případě zahájí dva sloupce až na

\begmulti: 39, 43

nové stránce. Ovšem vypadá hloupě, pokud se kvůli tomu ulomí nadpis „Rejstřík“ od jeho obsahu. Je tedy potřeba provést podobné měření stránky už před tiskem nadpisu „Rejstřík“. K tomu slouží makro \calculatedimone spuštěné na řádku 856 v makru \doindex.

docby.tex
```
1041: \def\calculatedimone{%
1042:    \ifdim\pagegoal=\maxdimen \dimen1=\vsize \corrsize{\dimen1}
1043:    \else \dimen1=\pagegoal \advance\dimen1 by-\pagetotal \fi}
```

5.14 Závěrečná nastavení, kategorie

Kategorie je rozumné nastavit až na konci souboru docby.tex. Na rozdíl od plainu přidáváme aktivní kategorii pro znak palce a nastavujeme podtržítko na obyčejnou kategorii, protože se ve zdrojových kódech programů často používá a vůbec ne ve významu matematického indexu. Kdyby mu tento význam zůstal, byly by jen potíže.

Podtržítko je další problém. Skoro vždy chceme, aby se ve vnitřních makrech chovalo jako normální podtržítko, ale když tiskneme text s podtržítkem fontem, který na dané pozici podtržítko nemá (to je Knuthův odkaz), pak by se to mělo udělat plainovským makrem _, Toto makro tedy schováme do \subori a pak ho probudíme k životu jen v okamžiku tisku v makrech \printsec, \printsubsec, \title a \normalhead. Uživatel tedy může napsat \sec␣moje_funkce a v makrech se bude _ jako obyčejné podtržítko, zatímco při tisku v nadpise se použije \subori.

docby.tex
```
1048: \catcode`\_=12
1049: \let\subori=\_ \def\_{_}
1050: \everymath={\catcode`\_=8 } \everydisplay={\catcode`\_=8 }
```

Nastavením \everymath a \everydisplay zaručíme matematikům stále možnost používat podtržítko ve významu indexu.

Aktivní palec spustí lokální verbatim prostředí uvnitř odstavce:

docby.tex
```
1056: \catcode`\"=\active
1057: \let\activeqq="
1058: \def"{\leavevmode\hbox\bgroup\mubytein=1\let\leftcomment=\empty
1059:       \let\returntoBlack=\empty \let\linecomment=\empty \let"=\egroup
1060:       \def\par{\errmessage{\string\par\space inside \string"...\string"}}%
1061:       \setverb\tt\quotehook
1062: }
```

Makro \langleactive nastaví aktivní kategorii pro znak je menší (<), takže bude možné zapisovat <text> a vytiskne se ⟨*text*⟩.

docby.tex
```
1064: \def\langleactive{\uccode`\~=`\<\catcode`\<=13
1065:       \uppercase{\def~}##1>{{$\langle$\it##1\/$\rangle$}}}
```

V tomto dokumentu jsem \langleactive použil v makru \quotehook, protože nechat znak < aktivní všude nedělalo dobrotu.

6 Rejstřík

Kontrolní sekvence označené šipkou (≻) jsou uživatelskými příkazy. Ostatní kontrolní sekvence jsou v DocBy.TEXu interní. Tučně je označena strana, kde je slovo dokumentováno, pak následuje seznam stran s výskyty slova. Uživatelské příkazy mají v seznamu stránek podtržené číslo, což je stránka, kde je příkaz vyložen na uživatelské úrovni.

\calculatedimone: 39, 43–44 \subori: 18–20, 44 \langleactive: 15, 44