C*s*Index v.2.11
czech/slovak implementation of —
— česká slovenská implementace programu
MakeIndex

27–Aug–1992

This is a czech/slovak implementation of the ***MakeIndex*** program. It was written by Zdeněk Wagner (wagner@csearn.bitnet) as a modification of an original ***MakeIndex*** program whose key contributors include

Toto je česká/slovenská implementace programu ***MakeIndex***. Napsal ji Zdeněk Wagner (wagner@csearn.bitnet) jako modifikaci programu ***MakeIndex***, jehož hlavními tvůrci jsou

Pehong Chen	chen@renoir.berkeley.edu
Leslie Lamport	lamport@src.dec.com

and, in reverse alphabetical order by last name,

a, v obráceném abecedním pořadí,

Klaus Zuenkler	kl%falke@ztivax.siemens.com
Martha Wershofen–Mersbach	grztex%dbngmd21.bitnet@net.bio.net
Art Werschulz	werschulz@cs.columbia.edu
Ed Sznyter	ews@pescadero.stanford.edu
Joachim Schrod	schrod@iti.informatik.th–darmstadt.de
Rainer Schoepf	schoepf@sc.zib–berlin.de
Rick Rodgers	rodgers@maxwell.mmwb.ucsf.edu
Sebastian Rahtz	spqr@ecs.southampton.ac.uk
Eberhard Mattes	mattes@azu.informatik.uni–stuttgart.de
Charles Karney	karney%ppc.mfenet@nmfecc.arpa
Mark James	mark@bdblues.altair.fr
Anita Hoover	anita@brahms.udel.edu
Michael Harrison	harrison@berkeley.edu
Baron Grey	baron@cs.ucla.edu
Andreas Brosig	brosig@gmdzi.gmd.de
Johannes Braams	braams@hlsdnl5.bitnet
Nelson Beebe	beebe@math.utah.edu

Copyright

The copyright statement of ***MakeIndex*** v.2.11 is:

Autorská práva na šíření programu ***MakeIndex*** v.2.11 jsou vymezena takto[1]:

Author:
Pehong Chen (phc@renoir.berkeley.edu)
Chen & Harrison International Systems, Inc.
Palo Alto, California
USA

[1] Český překlad: Zdeněk Wagner.

Obsah

1 Introduction
Úvod

As stated earlier, ***CsIndex*** is a czech/slovak[2] implementation of the ***MakeIndex*** program. It is named ***CsIndex*** to denote that a new functionality was added to the program. The version number remained 2.11 to denote that it comes from ***MakeIndex*** v.2.11.

Jak již bylo řečeno, ***CsIndex*** je česká/slovenská implementace programu ***MakeIndex***. Je pojmenována ***CsIndex***, čímž se má zdůraznit přidání nové funkce. Bylo však zachováno číslo verze 2.11, aby bylo zřejmé, že implementace pochází z programu ***MakeIndex*** v.2.11.

This document deals with description of the specific features of ***CsIndex*** and their implementation only. It does not explain the general problems of indexing. It does not describe the usage of the program either. These topics are covered in other files in this distribution. Hopefully they will also be translated into czech language.

Tento dokument popisuje pouze specifické znaky programu ***CsIndex*** a jejich implementaci. Nevysvětluje obecné problémy indexace ani nepopisuje použití programu. Tato problematika se vyskytuje v jiných souborech této distribuce. Lze jen doufat, že tyto soubory budou brzy přeloženy do češtiny.

This document is written in two languages: english for ***MakeIndex*** maintainers and czech for czech and slovak users. The contents of both parts may not necessarily be identical because czech and slovak people are quite familiar with some facts which may look strange to foreigners. Formatting of this document requires many style files and the czech hyphenation tables which might not be easily available for all potential users. Therefore only the formatted dvi–file is provided in this distribution.

Tento dokument je psán dvojjazyčně: anglicky pro programátory udržující ***MakeIndex*** a česky pro české a slovenské uživatele. Obsah obou částí nemusí být nutně identický, protože některé skutečnosti, jež jsou pro Čechy a Slováky samozřejmostí, mohou vypadat podivně pro cizince. Formátování tohoto dokumentu vyžaduje řadu stylů, které nemusí být snadno dostupné všem potenciálním uživatelům. Proto je zde poskytován pouze dvi–file.

CsIndex is a by–product of my commercial activities and therefore its design may be far from optimum. When writing it, I had the following in mind:

- To develop a usable program with as least effort as possible.
- To utilize routines already made.
- To build it in such a way that the original ***MakeIndex*** might easily be reconstructed.
- To interfere with the heart of ***MakeIndex*** as least as possible.

CsIndex je vedlejším produktem mých komerčních aktivit, a proto není zdaleka navržen optimálně. Při jeho tvorbě jsem měl na zřeteli následující:

- Vytvořit použitelný program s vynaložením co nejmenšího úsilí.
- Využít již hotové procedury.
- Vytvořit program tak, aby se dal snadno obnovit původní ***MakeIndex***.
- Zasahovat do srdce programu ***MakeIndex*** co nejméně.

[2] These are two distinct languages. However, they are very similar one another, so that one implementation of the index program will suit well both languages.

2 The first problem of czech and slovak languages
První problém češtiny a slovenštiny

The first problem of czech and slovak languages is their encoding in PC. As everybody knows, the PC was originally designed for US english. Only later National Language Support was added to MS–DOS. By that time, however, Czechoslovakia was behind the Iron Curtain. Therefore, neither czech nor slovak languages were supported. But some of us managed to get a PC. It was quite expensive so we usually had Hercules graphics without the possibility of downloading national characters. Therefore brothers Marian and Jan Kamenický invented encoding where the national letters were approximated by those letters of IBM extended character set which were most alike, e.g. č was approximated by ç, ý with ÿ, etc. The disadvantage of this code was the fact that it could not be downloaded into LaserJet Plus. This was probably one of the reasons why IBM and Hewlett–Packard developed another code, namely Latin 2. This is the code which is now part of MS–DOS 5.0. But to have it more complex, there is one more code coming from the ancient times where everything was centrally planned with close cooperation with Russians. So we have also KOI–8–ČS which was stated as standard but is rarely used. Moreover, *MicroSoft* has already announced Unicode to appear probably with Windows 4.0 and then everything will be different but I rather stop at the moment.

Prvním problémem češtiny a slovenštiny je jejich kódování. Jak všichni víme, osobní počítače byly původně vytvořeny pro americkou angličtinu. Teprve později byl MS–DOS doplněn o podporu národních jazyků. V té době však Československo leželo za Železnou oponou. Proto nebyla podporována čeština ani slovenština. Protože počítače, které tehdy byly nejčastěji importovány, neumožňovaly zavedení národních znaků, navrhli bratři Marian a Jan Kameničtí takové kódování, v němž byly národní znaky aproximovány nejpodobnějším znakem z rozšířené znakové sady IBM. Nevýhodou tohoto kódu však byla skutečnost, že nemohl být zaveden do tiskárny LaserJet Plus. To byl zřejmě jeden z důvodů, proč IBM a Hewlett–Packard vyvinuly jiný kód, Latin 2. Tento kód je nyní součástí operačního systému MS–DOS 5.0. Abychom to neměli tak jednoduché, pochází z dávných dob centrálního plánování v rámci RVHP kód KOI–8–ČS, který je sice normalizován, ale používá se jen zřídka. Navíc již *MicroSoft* ohlásil Unicode, který by se měl pravděpodobně objevit současně s Windows 4.0 a pak bude všechno úplně jinak. Ale zde raději úvahu o kódech přeruším.

So there is no code which could be stated as a standard. We might support one of them and ask users to use conversion programs but I would consider this "User Hostile". I prefer to support all codes and the conversion routines being built-in into ***CsIndex***.

Z výše uvedeného vyplývá, že neexistuje obecně užívaný standard. Samozřejmě bychom zde mohli použít jeden z kódů a požádat uživatele, aby používali konverzní programy. Takové řešení však považuji za „uživatelsky nepřátelské". Dávám přednost podpoře všech kódů, přičemž konverzní procedury jsou vestavěny v ***CsIndex****u*.

3 Czech/Slovak sorting Abecední řazení

3.1 Basic requirements Základní požadavky

Sorting requirements are standardized in ČSN 01 0181. This is not devoted to computerized sorting but nothing else exists so we must follow it.

Požadavky na abecední řazení jsou stanoveny normou ČSN 01 0181. Ta sice není určena pro abecední řazení na počítači, ale protože neexistuje nic jiného, musíme ji dodržet.

The sorting rules must take into account the order of national characters in the alphabet. We should stress that both czech and slovak alphabets have many accented letters which have their places in the alphabet. Most of them have only secondary ordering force, the primary force have only "č", "ř", "š", and "ž". All other accented characters are ordered as the corresponding unaccented ones and the difference is taken into account only in case of otherwise identical words:

Pravidla abecedního řazení vycházejí z národního pořadí písmen v abecedě. Většina písmen má přitom až sekundární řadicí platnost, primární platnost mají pouze „č", „ř", „š" a „ž". Ostatní písmena se řadí stejně jako příslušná písmena bez diakritických znamének a k rozdílu mezi nimi se přihlíží pouze u jinak shodného slova:

$$\text{plast} \prec \text{plást} \prec \text{plastický}$$

Capital letters have tertial sorting force. They are taken into consideration in case of totally identical words (including accented characters). Unlike the original US ***MakeIndex***, the small letters precede capitals:

Velká písmena mají terciární řadicí platnost. Berou se v úvahu až u zcela identických slov (včetně diakritických znamének):

$$\text{sokol} \prec \text{Sokol} \prec \text{SOKOL} \prec \text{sokolík}$$

3.2 Compound character CH Dvojhláska CH

There is a group CH which should be considered one character. As a matter of fact, this is not only specific to czech and slovak. We have it in other languages too, e.g. in spanish CH has a primary sorting force and belongs in between C and D. In czech and slovak languages it belongs in between H and I.

Dvojhláska CH je považována za jeden znak. Není to specifikum pouze češtiny a slovenštiny. Máme ji i v jiných jazycích, např. ve španělštině má CH primární řadicí platnost a patří mezi C a D.

The small ch has two capital equivalents: Ch, which is usually used in a normal text, and CH, which is used if the text is in all capitals. Both variants may appear in abbreviations. However cH should not be considered compound character but two distinct letters c and H.

Malé ch má dva „velké“ ekvivalenty: Ch, které se obvykle používá v normálním textu, a CH, jež se používá v případě, že celý text je psán velkými písmeny. Ve zkratkách se mohou vyskytovat obě varianty. Avšak cH nelze považovat za dvojhlásku, ale za dvě samostatná písmena c a H.

As mentioned above, CH is one character and thence it has primary sorting force. Capitals have tertiary sorting force and here we assign quaternary sorting force to the difference between Ch and CH, so that it holds:

Jak již bylo uvedeno, CH má primární řadicí platnost. Velká písmena mají terciární řadicí platnost a zde přiřadíme kvaternární řadicí platnost rozdílu mezi Ch a CH, takže platí:

$$\text{ch} \prec \text{Ch} \prec \text{CH}$$

3.3 When C and H is not CH
Když C a H není CH

There are some rare cases where C and H are two distinct characters. It happens with compound words where the first one ends with C and the second one begins with H. However, we can easily demonstrate that there is no way how to algorithmize the recognition. The word “mochnatý” is derived from a flower “mochna” ≡ cinquefoil, while “mochnátý” is a compound of two words “moc|hnátý[3]” which means “having many limbs”.

Existují vzácné případy, kdy C a H jsou dvě samostatná písmena. Stává se to u složených slov, kdy první část končí písmenem C a druhá začíná písmenem H. Lze však snadno ukázat, že neexistuje způsob, jak toto rozlišování algoritmizovat. Slovo „mochnatý“ je odvozeno od květiny „mochna“, zatímco „mochnátý“ je složenina znamenající mající mnoho končetin.

Such cases could be handled by a table of exceptions. But such a table could grow quite large and and it will not be an easy thing to build it. And there is more difficult task — abbreviations. Someone may invent his own abbreviation for “Cvičná Horská CHata” which would be CHCH. So we can conclude that *there exists **no** close algorithm for distinguishing whether C and H **is** or **is not** CH.*

Tyto případy lze řešit tabulkou výjimek. Ale taková tabulka by se mohla značně rozrůst a vytvoření takové tabulky by nebylo jednoduché. A navíc tu máme obtížnější úlohu — zkratky. Někdo může pro výraz „Cvičná Horská CHata“ navrhnout vlastní zkratku CHCH. Uzavřeme tedy tvrzením, že *neexistuje **žádný** uzavřený algoritmus pro rozlišení, zda C a H **je** či **není** CH.*

So we must help ourself with a little trick. If TEX finds an empty group {}, it just ignores it. Therefore we can use it to separate C and H. We will write `\index`{moc{}hnátý} and `\index`{C{}HCH}. Such transcription may sometimes interfere with the hyphenation algorithm but it can always be repaired by other means.

Musíme si tedy pomoci malým trikem. Když TEX narazí na prázdnou skupinu {}, prostě ji ignoruje. Můžeme ji proto použít k oddělení C a H. Budeme tedy psát `\index`{moc{}hnátý} a `\index`{C{}HCH}. Takový zápis může někdy kolidovat s algoritmem dělení slov, ale to lze vždycky napravit jinými prostředky.

[3] moc ≡ many, hnát ≡ limb, shank, pin, bone

3.4 Sorting numbers
Třídění čísel

The rules for sorting numbers and numerals are not easy to implement in usual sorting programs. However, both ***MakeIndex*** and ***CsIndex*** provide a solution.

Pravidla pro řazení čísel a číslovek se v obecných třídicích programech implementují nesnadno. Avšak ***MakeIndex*** i ***CsIndex*** poskytují řešení.

Cardinal numerals should be ordered according to their verbatim expression not depending whether written as words or as ciphers. Here we can achieve it with `\index{sto@100}` (sto ≡ one houndred).

Základní číslovky se řadí podle jejich slovního vyjádření bez ohledu na to, zda jsou psány slovy nebo číslicemi. To lze ovšem dosáhnout pomocí `\index{sto@100}`.

On the other hand, ordinal numerals should be ordered according to the value not depending whether they are written verbatim or with arabic or roman ciphers. Again we can use the transcription with "@" but we must supply sufficient number of leading zeros[4]:

Naproti tomu řadové číslovky se řadí podle své hodnoty bez ohledu na to, zda jsou psány slovně nebo arabskými či římskými číslicemi. to lze opět dosáhnout zápisem se znakem „@“, ale nyní musíme doplnit dostatečný počet úvodních nul:

```
\index{007.VII.@VII.}
\index{014.xiv.@xiv.}
\index{014.14.@14.}[5]
\index{100.stý@stý[6]}
```

3.5 Hierarchical expressions
Hierarchická hesla

This approach should be applied to personal names. First we should sort them according to the surname and only afterwards according to the first names. Therefore the correct order should be:

Tento přístup je třeba použít k řazení vlastních jmen. Nejprve je třeba třídit podle příjmení a teprve potom podle křestního jména. Správné pořadí by tedy mělo být:

sokol ≺ Sokol Jan ≺ Sokol Ján ≺ SOKOL ≺ sokolík

However, this is not incorporated into this version of ***CsIndex***. It will produce:

To však není zahrnuto do této verze programu ***CsIndex***. V tomto případě dostaneme:

sokol ≺ SOKOL ≺ Sokol Jan ≺ Sokol Ján ≺ sokolík

[4] In czech and slovak the ordinal numerals are written with dot.
[5] Note the difference between the usage of 14. and xiv. — Všimněte si rozdílu v zápisu 14. a xiv.
[6] stý ≡ houndredth

This case usually does not occur in index of books and articles. But even if it occured, it would not cause problems since we can always make use of ***MakeIndex***'s `\index`{Sokol!Jan}.

Tyto případy se obvykle nevyskytují v rejstřících knih a článků. Ale i kdyby se vyskytly, nebyl by to problém, protože vždy můžeme využít vlastnost programu ***MakeIndex***, tj. zápis `\index`{Sokol!Jan}.

4 How to use *CsIndex* Jak používat *CsIndex*

As mentioned earlier, we do **not** provide here the full description of ***CsIndex***. Instead we refer you to the original documentation of ***MakeIndex***. We just describe here the one parameter which is added for ***CsIndex***.

Jak již bylo uvedeno, v této verzi dokumentace **není** úplný popis programu ***CsIndex***. Místo toho odkazujeme na původní dokumentaci programu ***MakeIndex***. Zde pouze vysvětlíme jeden parametr, který je přidán pro ***CsIndex***.

CsIndex is invoked by a command:

CsIndex se odstartuje příkazem:

$$\mathsf{CSINDEX}\left[-\mathsf{z}\left\{\begin{array}{c} keybcs2 \\ latin2 \\ koi8cs \end{array}\right\}\right]\ldots$$

Parameter –z denotes the code used: keybcs2 (code of brothers Kamenický), Latin2, or KOI–8–ČS, respectively. The dots at the end mean that there are other parameters described in the documentation of ***MakeIndex***.

Parametr –z určuje použitý kód: keybcs2 (kód Kamenických), Latin2, KOI–8–ČS. Tečky na konci znamenají, že další parametry jsou popsány v původní dokumentaci programu ***MakeIndex***.

It is important to say that idx–file, ind–file and the optional style file MUST use the same code otherwise *CsIndex* will produce garbage.

Je nutno zdůraznit, že vstupní idx–soubor, výstupní ind–soubor i nepovinný „stylový soubor“ MUSÍ používat stejný kód, jinak *CsIndex* vyprodukuje smetí.

It is evident that one cannot sort using two languages. You cannot specify both czech/slovak sort and german sort (option –g). If you do specify two or more language options, only the last one is used. Also the –c option is not supported with the czech/slovak sort.

Je zřejmé, že nelze třídit podle dvou jazyků současně. Nemůžete současně požadovat české/slovenské a německé třídění (parametr –g). Pokud tak přesto učiníte, pouze poslední požadavek bude programem uspokojen. Kromě toho není při českém/slovenském třídění podporována volba –c.

5 Implementation Implementace

If you are just a user, then only the subsection 5.1 may be devoted for you. But if you happen to be a “local wizard”, you should read the text to the very end.

Pokud jste pouhý uživatel, je pro vás určena jen podkapitola 5.1. Pokud ale patříte k osobám zvaným „místní čaroděj“, pak musíte přečíst tento text až do konce.

5.1 How to compile *CsIndex* Jak přeložit *CsIndex*

CsIndex is written for IBM–PC compatible computers and it was compiled with TURBO C++ 1.01. Compilation with Borland C++ 2.0, 3.0, or 3.1 should not cause problems while usage of a different C or C++ compiler might be painful. ***CsIndex*** has not been ported to other systems than MS–DOS. Possibly the national language support is solved for UNIX but I am not aware of czech and slovak support on other systems.

CsIndex je napsán pro počítače kompatibilní s IBM–PC a byl přeložen kompilátorem TURBO C++ 1.01. Překlad kompilátorem Borland C++ 2.0, 3.0 nebo 3.1 by neměl činit potíže, zatímco přechod k jinému kompilátoru může být bolestný. ***CsIndex*** byl vytvořen a testován pouze pro MS–DOS. Pravděpodobně je národní podpora vyřešena pro UNIX, ale nejsem si vědom existence českého a slovenského prostředí v jiných operačních systémech.

The program was written in such a way which enables easy restoration of the original ***MakeIndex***. The czech/slovak implementation appears in two files: **csindex.h** and **csindex.c**. The header file **csindex.h** is ***#includ'ed*** at the very end of **mkind.h** and the changes in other files are made conditional. You can find them by searching for the string **CS_INDEX**.

Program byl napsán způsobem, který umožňuje snadné obnovení původního ***MakeIndex****u*. Implementace češtiny a slovenštiny se nachází ve dvou souborech: **csindex.h** a **csindex.cpp**. Soubor **csindex.h** je ***#includ'ován*** na samém konci souboru **mkind.h** a změny v ostatních souborech jsou podmíněné. Můžete je snadno nalézt hledáním řetězce **CS_INDEX**.

In this distribution you will find a makefile **csindex.mak**[7]. When writing the makefile for a different environment, you should bear in mind that:

V této distribuci naleznete makefile **csindex.mak**[8]. Budete-li vytvářet makefile pro jiné prostředí, musíte mít na zřeteli, že:

- **csindex.c** depends only on **csindex.h**,
- all files which depend on **mkind.h** must also depend on **csindex.h**.

- **csindex.c** závisí pouze na **csindex.h**,
- všechny soubory, které závisí na **mkind.h**, musí také záviset na **csindex.h**.

If you want to produce the original ***MakeIndex***, you just omit the #include "csindex.h" command at the end of **mkind.h** and exclude **csindex.c** from the makefile.

Pokud chcete vytvořit původní ***MakeIndex***, pak odstraníte příkaz #include "csindex.h" na konci souboru **mkind.h** a vyloučíte **csindex.c** z makefile.

[7] It was created from the project file by the PRJ2MAK utility.
[8] Byl vytvořen z projektového souboru programem PRJ2MAK.

5.2 The alphabet used
Použitá abeceda

As you remember from the beginning of the document, we have three codes for czech and slovak. For the computer, there is no difference. The sorting must always be implemented by tables. But KOI–8–ČS is systematic, so uppercase and lowercase may be easily programmed. This is the reason why KOI–8–ČS is used in ***CsIndex***. To make both sorting and conversion easier, we modify the original code table and use some unnecessary characters for storing CH.

Jak si pamatujete ze začátku tohoto dokumentu, existují tři kódování češtiny a slovenštiny. Pro počítač není mezi nimi žádný rozdíl. Třídění musí být vždy implementováno pomocí tabulek. Ale kód KOI–8–ČS je systematický, takže lze snadno naprogramovat konverzi malých a velkých písmen. Proto ***CsIndex*** pracuje v kódu KOI–8–ČS. Abychom si usnadnili třídění i konverzi, upravíme si kód tak, že využijeme neužitečné znaky pro zakódování CH.

5.3 csindex.h

This is a header file which defines the variables and funcions prototypes for the czech/slovak sort. It starts with defining **CS_INDEX** as its signature and loading **stdio.h** which provides some definitions essential for the functions defined further.

Tento soubor definuje proměnné a prototypy funkcí pro české/slovenské třídění. Nejprve definujeme **CS_INDEX** a pak načteme **stdio.h**, kde jsou definice potřebné ve funkcích definovaných později.

```
#define CS_INDEX

#include <stdio.h>
```

We will need some global variables throughout the whole program. Thence it is a good place to put their extern declarations here. To maintain the possibility af restoring the original ***MakeIndex***, we define them in **csindex.c**. But we will include this file to **csindex.c** too. So we must make the extern declaration conditional. For this purpose we use **CS_MAIN**.

V celém programu budeme potřebovat jisté globální proměnné. Zde je vhodné místo pro jejich deklaraci jako „extern“. Abychom zajistili možnost obnovení původního programu ***MakeIndex***, budeme tyto proměnné deklarovat v souboru **csindex.c**. Ale tento soubor budeme načítat i do **csindex.c**, proto musíme externí proměnné deklarovat podmíněně. K tomuto účelu použijeme **CS_MAIN**.

```
#ifndef CS_MAIN
extern char cscode;
extern char koi8cs[];
#endif
```

Now we redefine some macros from the standard library and from **mkind.h**. First we make sure that **ctype.h** is loaded. Then we supply our table **koi8cs** with specifications of character types and redefine TOLOWER and TOUPPER to use our functions cslower and csupper, respectively. Depending on the code used, they may contain a large code. Therefore they are made functions rather than macros.

Nyní předefinujeme některá makra ze standardní knihovny a ze souboru **mkind.h**. Nejprve se ujistíme, že jsme načetli **ctype.h**. Potom dodáme svoji tabulku **koi8cs** se specifikacemi typů znaků a předefinujeme TOLOWER a TOUPPER, aby používaly naše funkce cslower a csupper. tyto funkce generují poměrně dlouhý kód, a proto nejsou implementovány jako makra.

```
#ifndef __CTYPE_H
#include <ctype.h>
#endif

int csupper (int ch);
int cslower (int ch);

#define _ctype koi8cs

#undef isascii
#undef TOLOWER
#undef TOUPPER
#define isascii(c)  ((unsigned(c) < 256)
#define TOLOWER(c)  cslower((unsigned char)c)
#define TOUPPER(c)  csupper((unsigned char)c)
```

Here come the prototypes of the conversion, comparison, and input functions.

Zde jsou prototypy konverzních funkcí, porovnávání a funkce vstupu.

```
char *cs2ind (char *stg);
char  ccs2ind (char c);
int   cstrcmp (char *c1, char *c2);
int   csgetc (FILE*);
```

At last we redefine some more macros from **mkind.h**.

Nakonec předefinujeme několik dalších maker ze souboru **mkind.h**.

```
#ifndef CS_MAIN
#undef  GET_CHAR
#undef  STREQ
#undef  STRNEQ
#undef  VERSION
#undef  USAGE
#endif

#define GET_CHAR csgetc
#define VERSION   ">>Czech<< (hopefully) portable version 2.11 [21-Aug-1992]"
#define USAGE \
  "Usage:"n %s [-ilqrcg] [-s sty] [-o ind] [-t log] [-p num] [-z code] [idx0 idx1 ...]"ncode=keybcs2 — latin2 — koi8cs"
#define STREQ(a,b)  (!cstrcmp((a),(b)))
#define STRNEQ(a,b)  (cstrcmp((a),(b)))
```

5.4 genind.h

In this file only the macros for output are redefined. If czech/slovak option is used, the output string must first be converted to the appropriate code. One problem might only arise in PUTC, which is used to output a single letter. It is called if the first letter of the sorted words changes. As mentioned earlier, there is a coumpound character CH. If this is the case, we must actually output two characters.

V tomto souboru jsou předefinována pouze makra pro výstup. Pokud je aktivní české/slovenské třídění, musí být výstupní řetězec překonvertován do správného kódu. S problémem se můžeme setkat pouze v makru PUTC, které zapisuje jeden znak. Makro se volá tehdy, když se změní počáteční písmeno setříděných slov. Pokud je počátečním písmenem dvojhláska CH, musíme ve skutečnosti zapsat do výstupního souboru dva znaky.

```
#ifdef CS_INDEX
#define PUTC(C) { \
  if (!cscode || (unsigned char)C ≠ 221 || (unsigned char)C ≠ 253) { \
    fputc(cscode ? ccs2ind(C) : C, ind_fp); \
    if (cscode && ((unsigned char)C == 220 || (unsigned char)C == 252)) \
      fputc('h', ind_fp); \
  } \
}

#define PUT(S) { \
    fputs(cscode ? cs2ind(S) : S, ind_fp); \
}

#define PUTLN(S) { \
    fputs(cscode ? cs2ind(S) : S, ind_fp); \
    fputc('\n', ind_fp); \
    ind_lc++; \
}

#else

#define PUTC(C) { \
    fputc(C, ind_fp); \
}

#define PUT(S) { \
    fputs(S, ind_fp); \
}

#define PUTLN(S) { \
    fputs(S, ind_fp); \
    fputc('\n', ind_fp); \
    ind_lc++; \
}
#endif
```

5.5 csindex.c

This file comtains the implementation of input/output routines, conversion routines and the comparison routine.

Tento soubor obsahuje implementace vstupních/výstupních podprogramů, konverzních podprogramů a provnávacího podprogramu.

First we define CS_MAIN which is used in **csindex.h**. This file is included on the next line. As the third step we define global variable cscode. Its meaning is explained in the comment. We will also need prototypes from **string.h** and a useful type **uchar**.

Nejprve definujeme makroproměnnou CS_MAIN, která se používá v souboru **csindex.h**. Tento soubor je načten příkazem na následujícím řádku. Ve třetím kroku definujeme globální proměnnou cscode, jejíž význam je vysvětlen v komentáři. Kromě toho budeme potřebovat prototypy ze souboru **string.h** a užitečný typ **uchar**.

```
#define CS_MAIN
#include "csindex.h"

char cscode;

/*
 * 0 = original sort
 * 1 = keybcs2
 * 2 = latin2
 * 3 = koi8cs
 */

#include <string.h>

typedef unsigned char uchar;
```

We have already mentioned that inside ***CsIndex*** we use KOI–8–ČS. Thence we need conversion tables between this and the two remaining codes.

Již jsme uvedli, že v programu ***CsIndex*** používáme kód KOI–8–ČS. Potřebujeme tedy konverzní tabulky mezi tímto kódem a dvěma zbývajícími.

The first parts of all codes are identical. Differences are encountered within codes 128–255. Therefore only the second part of the tables must actually be defined. If you want to find an equivalent of a letter of KOI–8–ČS, find its code and then look at the corresponding place of _latin2 or _kamen. This way you obtain the equivalent in Latin2 or Keybcs2, respectively.

První části všech kódů jsou identické. Rozdíly se vyskytují v oblasti kódů 128–255. Proto musí být definována pouze druhá část tabulek. Chcete-li nalézt ekvivalent písmena zadaného v KOI–8–ČS, zjistěte si jeho kód a pak se podívejte na odpovídající místo tabulky _latin2 nebo _kamen. Tím zjistíte ekvivalent v kódu Latin2 resp. Keybcs2.

To make life easier, the tables are modified. We use following encodings:

Abychom si usnadnili život, tabulky jsme mírně modifikovali. Používáme následující kódování:

ch = (220, 221), Ch = (252, 221), CH = (252, 253)

The definition of the tables follows.

Definice konverzních tabulek následuje.

```
static uchar _latin2 [] =
  {128,131,134,245,209,135,136,137,138,139,140,143,151,141,207,152,
   157,158,164,165,168,169,219,220,223,177,178,182,184,185,186,187,
   188,189,190,198,199,200,246,201,202,203,204,205,206,173,174,175,
   208,225,211,215,170,171,221,176,227,228,235,238,239,240,242,244,
   218,160,192,159,212,216,234,196,129,161,133,146,150,148,229,162,
   147,132,253,231,156,163,195,130,247,236,167,194,'c','h',249,197,
   191,181,217,172,210,183,232,179,154,214,222,145,149,153,213,224,
   226,142,252,230,155,233,180,144,250,237,166,193,'C','H',254,255,0};

static uchar _kamen [] =
  {177,178,206,173,181,182,183,184,185,186,187,188,189,190,198,199,
   200,201,202,203,204,205,219,220,223,209,210,211,212,213,214,215,
   240,241,242,243,244,245,246,247,248,251,253,208,172,176,174,175,
   224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,
   218,160,192,135,131,136,170,196,129,161,150,139,140,148,164,162,
   147,132,169,168,159,163,195,130,216,152,145,194,'c','h',222,197,
   191,143,217,128,133,137,171,179,154,139,166,138,156,153,165,149,
   167,142,158,155,134,151,180,144,149,157,146,193,'C','H',254,255,0};
```

We also need to redefine the table of character types. The meaning of the numeric constants is explained in the comment of **ctype.h**.

Dále musíme předefinovat tabulku typů znaků. Význam numerických konstant je vysvětlen v komentářích souboru **ctype.h**.

```
char koi8cs[] = { 0,
0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x21,0x21,0x21,0x21,0x21,0x20,0x20,
0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,
0x01,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x40,0x40,0x40,0x40,0x40,0x40,
0x40,0x14,0x14,0x14,0x14,0x14,0x14,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,
0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x40,0x40,0x40,0x40,0x40,
0x40,0x18,0x18,0x18,0x18,0x18,0x18,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,
0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x40,0x40,0x40,0x40,0x20,
0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
0x40,0x08,0x40,0x08,0x08,0x08,0x08,0x40,0x08,0x08,0x08,0x08,0x08,0x08,0x08,0x08,
0x08,0x08,0x08,0x08,0x08,0x08,0x40,0x08,0x08,0x08,0x08,0x40,0x08,0x08,0x40,0x40,
0x40,0x04,0x40,0x04,0x04,0x04,0x04,0x40,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,
0x04,0x04,0x04,0x04,0x04,0x04,0x40,0x04,0x04,0x04,0x04,0x40,0x04,0x04,0x40,0x20};
```

Now we can write the conversion primitives. Function tocode performs in–place conversion of the KOI–8–ČS string **to** another code while fromcode makes the conversion of an int **from** another code. The second argument is the conversion table. Function itocode is an integer variant of tocode.

Nyní napíšeme primitivní konverzní funkce. Funkce tocode provádí konverzi na místě **do** jiného kódu, zatímco fromcode konvertuje proměnnou typu int **z** jiného kódu. Druhým parametrem je konverzní tabulka. Funkce itocode je celočíselnou variantou tocode.

With aid of these functions we develop conversions between KOI–8–ČS and Latin2 or Keybcs2, respectively.

S pomocí těchto funkcí vytvoříme konverze mezi KOI–8–ČS a kódy Latin2 resp. Keybcs2.

```
static void tocode (uchar *convstr, const uchar *mask) {
while (*convstr) {
  if (*convstr > 127)
    *convstr = mask[*convstr-128];
  convstr++;
}}

static int itocode (int conv, const uchar *mask) {
return (conv > 127) ? mask[conv-128] : conv;
}

static int fromcode (int conv, const uchar *mask) {
int i;
if (conv > 127)
  for (i = 0;  i < 128;  i++)
    if (conv == mask[i]) { conv = i + 128;   break; }
return conv;
}

static uchar *tolatin (uchar *stg) {
tocode(stg, _latin2);       return stg;
}

int itolatin (int stg) {
return stg = itocode (stg, _latin2);
}

static uchar *tokamen (uchar *stg) {
tocode(stg, _kamen);        return stg;
}

int itokamen (int stg) {
return stg = itocode (stg, _kamen);
}

int fromlatin (int stg) {
return stg = fromcode (stg, _latin2);
}

int fromkamen (int stg) {
return stg = fromcode (stg, _kamen);
}
```

Following functions are called from macros which make output to the ind–file.

Následující funkce jsou volány z maker provádějících výstup do ind–souboru.

```
char *cs2ind (char *stg) {
switch (cscode) {
case 1: tokamen ((uchar*)stg);  break;
case 2: tolatin ((uchar*)stg);  break;
}
return stg;
}

char ccs2ind (char c) {
int ch = (unsigned char)c;
switch (cscode) {
case 1: ch = itokamen (ch);  break;
case 2: ch = itolatin (ch);  break;
}
return ch;
}
```

Function idx2cs is called from csgetc for converting the input. Notice that we scan for ch, Ch, and CH.

Funkce idx2cs je volána z csgetc pro konverzi vstupních dat. Všimněte si, že přitom hledáme dvojhlásky ch, Ch CH.

```
char idx2cs (char ch) {
int c = (unsigned char)ch;
if (cscode && c ≠ 220 && c ≠ 221 && c ≠ 252 && c ≠ 253)
switch (cscode) {
case 1: c = fromkamen (c);  break;
case 2: c = fromlatin (c);  break;
}
return c;
}

int csgetc (FILE *fptr) {
int ch, preview;
ch = getc (fptr);
if (!cscode) return ch;
while (ch ≠ EOF && (ch == '{' || ch == 'c' || ch == 'C')) {
  preview = getc (fptr);
  if (ch == '{') {
    if (preview == '}') ch = getc (fptr);
    else {
      ungetc (preview, fptr);  break;
    }
  }
  else {
    if (preview == 'h') {
      ch = ch == 'C' ? 252 : 220;  preview = 221;
    }
    else
    if (preview == 'H' && ch == 'C') {
      ch = 252;  preview = 253;
    }
    ungetc (preview, fptr);  break;
  }
}
return idx2cs (ch);
}
```

Before we explain the following code, we must make one comment. In english sort, the capital letters take precedence over the small ones. This is the ordering of the original ASCII. In czech/slovak sort, however, the ordering should be opposite. This corresponds to the order of accented characters in KOI–8–ČS. To make life easier for sort implementation, we exchange small letters and capitals in the original ASCII, i.e. we look at **a** as if it were **A** and vice versa.

Než vysvětlíme následující část programu, musíme vložit jeden komentář. Při anglickém třídění jsou velká písmena před malými, tj. podle pořadí v ASCII. V českém/slovenském třídění však tomu má být naopak, což odpovídá pořadí písmen s diakritickými znaménky v KOI–8–ČS. Abychom si usnadnili život při implementaci třídění, prohodíme v původním ASCII malá a velká písmena, tj. pohlížíme na **a**, jako by to bylo **A** a naopak.

The next tables define the collating sequence. The characters which appear in Same belong to the same place as the corresponding characters from SameAs, while the characters from After belong just after the corresponding characters from Before.

Následující tabulky definují třídicí posloupnost. Znaky, které se vyskytují v Same, patří na stejné místo jako odpovídající znaky ze SameAs, zatímco znaky z After patří bezprostředně za odpovídající znaky z Before.

```
static uchar Same [] = {225,241,248,247,229,233,239,237,240,245,234,232,249,
      228,230,235,236,238,244,193,209,216,215,197,201,207,205,208,213,202,
      200,217,196,198,203,204,206,212,0};
static uchar SameAs [] = "aaaeeiooouuuydrllntAAAEEIOOOUUUYDRLLNT";
static uchar After [] = {227,242,243,250,195,210,211,218,252,220,0};
static uchar Before [] = "crszCRSZhH";
```

Functions cslower and csupper are called to test whether the first letter of the sorted words has changed. For this purpose the accented characters which have secondary sorting force should be converted to the corresponding unaccented characters.

Funkce cslower a csupper se volají pro zjištění, zda se změnilo počáteční písmeno setříděných slov. Pro tento účel je nutno odstranit diakritické znaménko z těch písmen, která mají sekundární řadicí platnost.

```
#if __TURBOC__
#pragma warn -pia
#pragma warn -sig
#endif

int cslower (int ch) {
char *p;
if (!cscode) return tolower(ch);
if (p = strchr((char*)Same, ch)) ch = SameAs[p-(char*)Same];
return isupper(ch) ? ch ^ 32 : ch;
}

int csupper (int ch) {
char *p;
if (!cscode) return toupper(ch);
if (p = strchr((char*)Same, ch)) ch = SameAs[p-(char*)Same];
return islower(ch) ? ch ^ 32 : ch;
}

#if __TURBOC__
#pragma warn .sig
#pragma warn .pia
#endif
```

Finally we define four pass comparison. The first two passes are case insensitive. In the first pass, the characters from Same are replaced by corresponding characters from SameAs while in the next passes they are expanded reflecting their order in Same. In the first three passes, the codes 221 and 253 (the second part of CH) are skipped. They are examined only in the fourth pass in order to distinguish Ch and CH.

Nakonec definujeme čtyřprůchodové porovnávání. V prvních dvou průchodech se nerozlišují malá a velká písmena. Při prvním průchodu jsou písmena ze Same nahrazena odpovídajícími znaky ze SameAs, zatímco při dalších průchodech jsou nahrazena kódem odvozeným z jejich umístění v Same. V prvních třech průchodech se přeskakují kódy 221 a 253 (druhá část CH). Porovnávají se až ve čtvrtém průchodu, abychom rozlišili Ch a CH.

The progress of sorting may be demostrated by a following example.

Postup třídění můž být demonstrován následujícím příkladem.

0: car = Car = CAR = cár = Cár = CÁR[9]
1: car = Car = CAR ≺ cár = Cár = CÁR
2: car ≺ Car ≺ CAR ≺ cár ≺ Cár ≺ CÁR

```
int cstrcmp (char *s1, char *s2) {
int x1, x2;     char *p, *c1, *c2;
int st1, st2, result = 0;
uchar d1, d2;
int pass;
if (!cscode) return strcmp (s1, s2);
for (pass = 0;  !result && pass < 4;  pass++) {
  c1 = s1;  c2 = s2;
  do {
    st1 = st2 = 0;
    if (pass < 3) {
      if (*(uchar*)c1 == 221 || *(uchar*)c1 == 253) c1++;
      if (*(uchar*)c2 == 221 || *(uchar*)c2 == 253) c2++;
    }
    d1 = *(uchar*)c1;  d2 = *(uchar*)c2;
    if (pass > 1) {
      if (isalpha (d1) && d1 < 128) d1 ^= 0x20;
      if (isalpha (d2) && d2 < 128) d2 ^= 0x20;
    }
    else {
      if (isalpha (d1)) d1 |= 0x20;
      if (isalpha (d2)) d2 |= 0x20;
    }
    #if __TURBOC__
    #pragma warn -pia
    #pragma warn -sig
    #endif
    if (d1 < 193) x1 = d1;
    else
    if (p = strchr((char*)Same, d1))
      x1 = pass ? p - (char*)Same + 128 : SameAs[p-(char*)Same];
```

[9] car ≡ tsar, cár ≡ rag.

```
    else
    if (p = strchr((char*)After, d1)) {
      x1 = Before[p-(char*)After];  st1 = 1;
    }
    else x1 = d1;
    if (d2 < 193) x2 = d2;
    else
    if (p = strchr((char*)Same, d2))
      x2 = pass ? p - (char*)Same + 128 : SameAs[p-(char*)Same];
    else
    if (p = strchr((char*)After, d2)) {
      x2 = Before[p-(char*)After];  st2 = 1;
    }
    else x2 = d2;
    #if __TURBOC__
    #pragma warn .sig
    #pragma warn .pia
    #endif
    if (!(result = x1 - x2)) result = st1 - st2;
  } while (!result && *c1++ && *c2++);
}
return result;
}
```

5.6 sortid.h

The czech/slovak comparison function is used within this file. It appears in two functions only: in check_mixsym and compare_string. All other functions remained unchanged.

V tomto souboru se používá česká/slovenská porovnávací funkce. Vyskytuje se poze ve dvou podprogramech: check_mixsym a compare_string. Všechny ostatní funkce zůstaly beze změny.

```
static int
check_mixsym(x, y)
char   *x;
char   *y;
{
    int     m;
    int     n;

    m = ISDIGIT(x[0]);
    n = ISDIGIT(y[0]);

    if (m && !n)
      return (1);

    if (!m && n)
      return (-1);

    #ifdef CS_INDEX
    return (cstrcmp(x, y));
```

```
    #else
    return (strcmp(x, y));
    #endif
}


static int
compare_string(a, b)
unsigned char   *a;
unsigned char   *b;
{
    int     i = 0;
    int     j = 0;
    int     al;
    int     bl;

    #ifdef CS_INDEX
    if (cscode) {
      return cstrcmp ((signed char*)a, (signed char*)b);
    }
    #endif

    while ((a[i] ≠ NUL) || (b[j] ≠ NUL)) {
      if (a[i] == NUL)
          return (−1);
      if (b[j] == NUL)
          return (1);
      if (letter_ordering) {
          if (a[i] == SPC)
            i++;
          if (b[j] == SPC)
            j++;
      }
      al = TOLOWER(a[i]);
      bl = TOLOWER(b[j]);

      if (al ≠ bl)
          return (al − bl);
      i++;
      j++;
    }
    if (german_sort)
      return (new_strcmp(a, b, GERMAN));
#if (OS_BS2000 | OS_MVSXA)          /* in EBCDIC 'a' is lower than 'A' */
    else
      return (new_strcmp(a, b, ASCII));
#else
    else
      return (strcmp((char*)a, (char*)b));
#endif                              /* (OS_BS2000 | OS_MVSXA) */
}
```

5.7 mkind.c

This is the main file which also needed some changes. First, the help file BCHELP10.ZIP (released by Borland), which I have recently downloaded from **wsmr–simtel20.army-mil**, says that you must include **dos.h** if you modify the stack length. Also it is a good practice to mark the constant as unsigned.

Toto je hlavní soubor. I ten vyžadoval modifikace. Především, soubor BCHELP10.ZIP (zveřejněný firmou Borland), který jsem nedávno získal z archivu **wsmr–simtel20.army.mil**, tvrdí, že misíme načíst **dos.h**, jestliže měníme délku zásobníku. Dále je dobrým zvykem označení konstanty jako unsigned.

```
#if   IBM_PC_TURBO
#include <dos.h>
unsigned      _stklen = 0xf000u;      /* Turbo C ignores size set in
                                      .exe file by LINK or EXEMOD,
                                      sigh... */
#endif                                /* IBM_PC_TURBO */
```

In the main program we just need to recognize the new switch.

V hlavním programu pouze musíme rozeznat nový parametr.

```
int
main(argc, argv)
int     argc;
char   *argv[];
{
   char   *fns[ARRAY_MAX];
   char   *ap;
   int     use_stdin = FALSE;
   int     sty_given = FALSE;
   int     ind_given = FALSE;
   int     ilg_given = FALSE;
   int     log_given = FALSE;

   /* determine program name */
#if (OS_ATARI | OS_VAXVMS | OS_BS2000 | OS_MVSXA | OS_VMCMS)
   pgm_fn = "MakeIndex";              /* Use symbolic name on some systems */
#else
   pgm_fn = strrchr(*argv, DIR_DELIM);
   if (pgm_fn == NULL)
     pgm_fn = *argv;
   else
     pgm_fn++;

#if OS_PCDOS
   {
     register char *ext = pgm_fn + strlen(pgm_fn)-4;
     if ( *ext == '.' )
         *ext = NUL;                  /* cut off ".EXE" */
   }
   (void)strlwr(pgm_fn);              /* lower program name    */
#endif                                /* OS_PCDOS */
```

```
#endif                          /* OS_VAXVMS | OS_BS2000 | OS_MVSXA */

   /* process command line options */
   while (--argc > 0) {
     if (**++argv == SW_PREFIX) {
         if (*(*argv + 1) == NUL)
           break;
         for (ap = ++*argv; *ap ≠ NUL; ap++)
#if  (OS_BS2000 | OS_MVSXA | OS_VAXVMS)
           switch (tolower(*ap)) {
#else
           switch (*ap) {
#endif /* (OS_BS2000 | OS_MVSXA | OS_VAXVMS) */

              /* use standard input */
           case 'i':
              use_stdin = TRUE;
              break;

              /* enable letter ordering */
           case 'l':
              letter_ordering = TRUE;
              break;

              /* disable range merge */
           case 'r':
              merge_page = FALSE;
              break;

              /* supress progress message – quiet mode */
           case 'q':
              verbose = FALSE;
              break;

              /* compress blanks */
           case 'c':
              compress_blanks = TRUE;
              break;

              /* style file */
           case 's':
              argc--;
              if (argc ≤ 0)
                FATAL("Expected -s <stylefile>\n","");
              open_sty(*++argv);
              sty_given = TRUE;
              break;

              /* output index file name */
           case 'o':
              argc--;
```

```
        if (argc ≤ 0)
          FATAL("Expected -o <ind>\n","");
        ind_fn = *++argv;
        ind_given = TRUE;
        break;

        /* transcript file name */
      case 't':
        argc--;
        if (argc ≤ 0)
          FATAL("Expected -t <logfile>\n","");
        ilg_fn = *++argv;
        ilg_given = TRUE;
        break;

        /* initial page */
      case 'p':
        argc--;
        if (argc ≤ 0)
          FATAL("Expected -p <num>\n","");
        strcpy(pageno, *++argv);
        init_page = TRUE;
        if (STREQ(pageno, EVEN)) {
          log_given = TRUE;
          even_odd = 2;
        } else if (STREQ(pageno, ODD)) {
          log_given = TRUE;
          even_odd = 1;
        } else if (STREQ(pageno, ANY)) {
          log_given = TRUE;
          even_odd = 0;
        }
        break;

        /* enable german sort */
      case 'g':
        german_sort = TRUE;
  #ifdef CS_INDEX
  cscode = 0;  /* cannot use both German and Czech sort */
  #endif
        break;

#ifdef CS_INDEX
  /* enable Czech sort */
case 'z':
  argc--;  argv++;
  if (STREQ(strlwr(*argv), "keybcs2")) cscode = 1;
  else
  if (STREQ(strlwr(*argv), "latin2")) cscode = 2;
  else
  if (STREQ(strlwr(*argv), "koi8cs")) cscode = 3;
```

```
      german_sort = FALSE;
      break;
  #endif

            /* bad option */
          default:
            FATAL("Unknown option -%c.\n", *ap);
            break;
          }
    } else {
        if (fn_no < ARRAY_MAX) {
          check_idx(*argv, FALSE);
          fns[++fn_no] = *argv;
        } else {
          FATAL("Too many input files (max %d).\n", ARRAY_MAX);
        }
    }
  }
  process_idx(fns, use_stdin, sty_given, ind_given, ilg_given, log_given);
  idx_gt = idx_tt - idx_et;
  ALL_DONE;
  if (idx_gt > 0) {
    prepare_idx();
    sort_idx();
    gen_ind();
    MESSAGE("Output written in %s.\n", ind_fn);
  } else
    MESSAGE("Nothing written in %s.\n", ind_fn);

  MESSAGE("Transcript written in %s.\n", ilg_fn);
  CLOSE(ind_fp);
  CLOSE(ilg_fp);
  EXIT(0);

  return (0);                    /* never executed–avoids complaints */
  /* about no return value */
}
```

A Codes of czech/slovak characters Kódování češtiny/slovenštiny

In the following table only the codes of czech and slovak characters are given.

V následující tabulce jsou pouze kódy písmen používaných v češtině a slovenštině.

CHAR	KOI–8–ČS			LATIN 2			KEYBCS 2		
	dec	hex	octal	dec	hex	octal	dec	hex	octal
á	193	C1	301	160	A0	240	160	A0	240
ä	209	D1	321	132	84	204	132	84	204
à	216	D8	330	247	F7	367	216	D8	330
č	195	C3	303	159	9F	237	135	87	207
ď	196	C4	304	212	D4	324	131	83	203
é	215	D7	327	130	82	202	130	82	202
ě	197	C5	305	216	D8	330	136	88	210
í	201	C9	311	161	A1	241	161	A1	241
Í	203	CB	313	146	92	222	139	8B	213
ľ	204	CC	314	150	96	226	140	8C	214
ň	206	CE	316	229	E5	345	164	A4	244
ó	207	CF	317	162	A2	242	162	A2	242
ô	208	D0	320	147	93	223	147	93	223
ö	205	CD	315	148	94	224	148	94	224
ŕ	198	C6	306	234	EA	352	170	AA	252
ř	210	D2	322	253	FD	375	169	A9	251
š	211	D3	323	231	E7	347	168	A8	250
ť	212	D4	324	156	9C	234	159	9F	237
ú	213	D5	325	163	A3	243	163	A3	243
ů	202	CA	312	133	85	205	150	96	226
ü	200	C8	310	129	81	201	129	81	201
ý	217	D9	331	236	EC	354	152	98	230
ž	218	DA	332	167	A7	247	145	91	221
Á	225	E1	341	181	B5	265	143	8F	217
Ä	241	F1	361	142	8E	216	142	8E	216
À	248	F8	370	250	FA	372	149	95	225
Č	227	E3	343	172	AC	254	128	80	200
Ď	228	E4	344	210	D2	322	133	85	205
É	247	F7	367	144	90	220	144	90	220
Ě	229	E5	345	183	B7	267	137	89	211
Í	233	E9	351	214	D6	326	139	8B	213
Ĺ	235	EB	353	145	91	221	138	8A	212
Ľ	236	EC	354	149	95	225	156	9C	234
Ň	238	EE	356	213	D5	325	165	A5	245
Ó	239	EF	357	224	E0	340	149	95	225
Ô	240	F0	360	226	E2	342	167	A7	247
Ö	237	ED	355	153	99	231	153	99	231
Ŕ	230	E6	346	232	E8	350	171	AB	253
Ř	242	F2	362	252	FC	374	158	9E	236

CHAR	KOI–8–ČS			LATIN 2			KEYBCS 2		
	dec	hex	octal	dec	hex	octal	dec	hex	octal
Š	243	F3	363	230	E6	346	155	9B	233
Ť	244	F4	364	155	9B	233	134	86	206
Ú	245	F5	365	233	E9	351	151	97	227
Ů	234	EA	352	222	DE	336	166	A6	246
Ü	232	E8	350	154	9A	232	154	9A	232
Ý	249	F9	371	237	ED	355	157	9D	235
Ž	250	FA	372	166	A6	246	146	92	222

There are currently no codes recognized as "à" and "À" in czech TEX. They must be entered as `\‘a` and `\‘A`, respectively.

V současné verzi českého TEXu neexistují kódy pro „à“ a „À“. Tyto znaky je nutno zadávat jako `\‘a` a `\‘A`.