ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Programování v jazyce C

Programovací techniky

K03-21: Převáděč kódování české diakritiky

Jméno a příjmení: *Martin Sloup*
Osobní číslo: *A04372*
Obor: *INIB – INF*
E-mail: *msloup@students.zcu.cz*

Zadání

Připravte program, který bude převádět kódy české diakritiky v textových souborech (Kameničtí , Latin-2, ISO-8859-2, Windows-1250, atd.). Prostudujte materiály o kódování české diakritiky a zajistěte podporu pokud možno všech použitelných kódů. Naprogramujte automatickou detekci kódování vstupního souboru a kontrolu, zdali je vstupní soubor opravdu prostý text. Zároveň zajistěte automatické ověření, zdali vstupní soubor nepochází z UNIXu, tj. zda neobsahuje na koncích řádků jen znak CR (pozn. Unix má LF, CR má MacOS), a pokud je tomu tak, ať program doplní na koncích řádků za znak CR i znak LF. Umožněte také přenos textových souborů zpět na UNIX odstraněním znaku LF. Program může být ovládán jen parametry zadávanými z příkazové řádky nebo může mít interaktivní uživatelské rozhraní. Snažte se však, aby bylo ovládání co nejjednodušší a nejrychlejší.

Analýza úlohy a popis řešení

Hlavním účelem toho programu je překódování češtiny v souboru. Podle zadání jsem napsal algoritmus na převedení mezi osmi bitovými kódováními. Při tom bylo potřeba použít konverzní tabulky. Nejjednodušší způsob by byl je přímo včlenit do zdrojového kódu, ale rozhodl jsem se použít jinou metodu – tabulky v textových soborech. Tím je možné v budoucnu rozšířit konverzní tabulky i na jiné kódování, než které jsem použil v programu. Dále jsem se rozhodl včlenit do programu i podporu pro UTF-8 a UTF-16 kódování, které se v poslední době začíná hojně vyskytovat na internetu a to i u českých stránek.

Dále, podle zadání, jsem udělal detekci, zda je soubor binární a detekci kódování vstupního souboru. Pro detekci binárního souboru jsem použil jednoduchou metodu, kdy procházím celým souborem a porovnávám ascii hodnotu každého znaku, zda neobsahuje ascii znaky velmi netypické pro textové soubory. Pro detekci vstupního kódování jsem použil ideu ze zdrojového kódu jiného konverzního programu napsaného Petrem Laštovičkou[1]. Algoritmus vychází ze skutečnosti, že veškerá česká kódování nemají ascii hodnoty pro diakritické znaky na stejném místě. Například kódování WINDOWS-1250 se liší celkově v osmi ascii hodnotách od kódování ISO-8859-2. Takže nakonec stačí si vytvořit tabulku která určuje, zda příslušná ascii hodnota je použita v české diakritice. Pak se jen projede celý soubor a porovná se s touto tabulkou a pokud je ascii hodnota pro tento znak definovaná, tak se do pomocné proměnné přičte jeden bod shody. Toto se provede pro kódování WINDOWS-1250, LATIN-2, KEYBCS2 (Bratři Kameničtí) a ISO-8859-2. Z těchto kódování se vybere kódování, které dostane nejvíc bodů. U kódování UTF-8 jsem vytvořil test validity podle definice, jak mají vypadat vyšší znaky než 128. Tzn., že pokud se vyskytne znak v binárním kódu `110x xxxx`, musí za ním následovat `10xx xxxx`, atd. U kódování UTF-16 provádí detekce na základě prvních dvou znaků, tzv. BOM (Byte Order Mark).

Načtení tabulky a konverze

Když už víme vstupní a výstupní kódování, provedeme načtení tabulky. Pro tabulky jsem zvolil formát A, který je nyní hojně používaný organizací Unicode.org. Formát tabulky je následující:

```
…
# Komentar
0x00   0x0000       #      NULL
0x01   0x0001       #      START OF HEADING
0x02   0x0002       #      START OF TEXT
0x03   0x0003       #      END OF TEXT
…
```

Jedná se o tři sloupce oddělené tabulátorem. V prvním sloupci je ascii hodnota v hexadecimálním tvaru definovaného znaku. Ve druhém sloupci je hexadecimální číslo příslušného znaku v unicode a ve třetím sloupci je název znaku v Unicode (před ním je ko-

[1] Zdrojový kód v jazyce Turbo Pascal: http://sweb.cz/petr.lastovicka/cs.zip

mentářový znak).

Nejznámější kódování jsem našel na stránkách organizace Unicode a přiložil je k tomuto programu. Bohužel pro některá česká kódování tabulky nebyly k dispozici. Jedná se například o kódování Bratrů Kamenických (není registrován u IANA, proto mu nelze dát přesný zkrácený název – já jsem použil keybcs2, které se nejvíce používá) či KOI-8 ČS2 definované směrnicí ČNS 369103[2], které jsem si musel napsat sám.

Nejprve tedy načtu zdrojové kódování, které rozřežu na index (ascii hodnota) a na tzv. nový znak (hodnota Unicode znaku) a ty uložím do konverzní tabulky programu. Pro cílové kódování se provede, dá se říct, to samé, jen s tím rozdílem, že hodnoty nebude přímo přidávat do tabulky, ale bude se snažit v tabulce najít Unicode kód znaku totožný s Unicode hodnotou znaku právě načteného z tabulkového souboru. Pokud ho najde do nového znaku zapíše ascii hodnotu kódování (hodnotu v prvním sloupci tabulkového souboru) a nastaví si atribut, že pro tuto hodnotu našel cílový ascii kód. Při překódování pak postupně nalézá ascii kód v konverzní tabulce a nahrazuje ho novým ascii kódem.

Pro UTF-8 se načítá jen zdrojové případně cílové kódování, protože vychází z Unicode kódu a jen se liší příslušným zakódováním ve speciálním tvaru. Pokud bude tedy UTF-8 jako zdrojové kódování do tabulky se zapíše ascii hodnota jako nový znak a Unicode hodnota jako index. Při překódování souboru z UTF-8 na ascii nejprve dekóduje UTF-8 znak na Unicode variantu, najde ho v tabulce a následně zapíše ascii hodnotu do nového souboru. Viz tabulka níže:

UTF-8	Binárně v UTF-8	Binárně v Unicode	Unicode
0x74	0111 0100 0xxx xxxx	111 0100	0x0074
0xC599	1100 0101 1001 1001 110x xxxx 10xx xxxx	0 0101 01 1001	0x0159
0xEF91B8	1110 1111 1001 0001 1011 1000 1110 xxxx 10xx xxxx 10xx xxxx	1111 01 0001 11 1000	0xF478

Poznámka: Znak „x“ označuje bit znaku, který se použije.

Při konverzi z ascii na UTF-8 se postupuje obráceně. Nejdříve se najde v tabulce ascii hodnota právě načteného znaku a z tabulky se načte Unicode kód, který se následně převede na UTF-8. Veškeré převody mezi Unicode a UTF-8 se provádějí patřičným posunutím bitů a přičtením hodnoty tak, aby výsledný kód byl podle definice.

U UTF-16 se načítají nebo ukládají vždy dva znaky, které jsou seřazeny podle režimu BOM (Byte Order Mark), jinak hodnoty těchto dvou znaků odpovídají hodnotám Unicode. Při režimu Big Endian se nejprve uloží vyšší bajt znaku, po něm nižší bajt. U Little Endian to probíhá obráceně. Například Unicode znak 0x0159 se v Big Endian uloží jako dva znaky 0x01 a 0x59, v Little Endian jako 0x59 a 0x01.

Během konverzního cyklu algoritmus rozeznává nové řádky a podle nastavení provádí jejich záměnu.

Popis souborů a stručný popis funkcí

Program je rozložen v následujících souborech:

`conv.c`	provádí konverzi mezi znakovýma sadama
`file_test.c`	zajišťuje detekci vstupního kódování
`function.c`	pomocné funkce

[2] http://www.cestina.cz/kodovani/csn.html

`load_table.c`	funkce pro načítání konverzní tabulky ze souboru
`main.c`	hlavní program

Funkce obsažené v jednotlivých souborech:

conv.c

`void conv_ascii2ascii();`
Převádí soubor podle kódovací tabulky z jednoho kódování na druhé.

`void conv_ascii2utf8();`
Převádí soubor podle kódovací tabulky z ascii na UTF-8.

`void conv_utf82ascii();`
Převádí soubor podle kódovací tabulky z UTF-8 na ascii.

`void conv_ascii2utf16();`
Převádí soubor podle kódovací tabulky z ascii na UTF-16.

`void conv_utf162ascii();`
Převádí soubor podle kódovací tabulky z UTF-16 na ascii.

`void conv_utf82utf16();`
Převádí soubor z UTF-8 na UTF-16.

`void conv_utf162utf8();`
Převádí soubor z UTF-16 na UTF-8.

file_test.c

`int is_file_binary(char *filename);`
Vrací 1, pokud je obsah souboru filename binární.

`char *detect_encoding(char *filename);`
Vrací označení detekovaného kódování v souboru filename

function.c

`char *my_substr(char *text, int from, int length);`
Vrací výřez textu. Funkce se chová stejně jako v jazyce PHP.

`void my_strtr(char *text, char *oldchar, char *newchar);`
Funkce nahrazuje v textu každý znak uvedený v oldchar znakem z newchar.

`int my_strrchr(char *text, char s_char);`
Vrací pozici posledního výskytu znaku s_char v řetězci text.

`char **my_strtok(char *text, int item_count);`
Rozřezává vstupní text podle mezer či tabulátorů a vrací pole částí o velikosti item_count.

`char *trim(char *text);`
Vrací text oříznutý o znaky \n, \r, \t, a mezeru.

`char *lcase(char *text);`
Konverguje vstupní text na malá písmena.

```
char *ucase(char *text);
```
Konverguje vstupní text na velká písmena.

```
void gg_sort(void *base, size_t nel, size_t width, int
(*cmp)(const void *, const void *));
```
Setřídí pole combsort algoritmem. Je to náhrada za, u mě padající, qsort pod Linuxem.

load_table.c

```
void load_table(char *filename, char mode);
```
Načte tabulku ze souboru filename, mode udává jak se má načíst(A = ascii index, U = Unicode index, S=Přiřazení k ascii hodnotě ascii hodnotu)

```
void sorttable();
```
Setřídí kódovací tabulku podle indexů od nejmenšího k největšímu.

```
void compresstable();
```
Vymaže z tabulky hodnoty, které nemůžou být použity pro konverzi.

```
unsigned int search_char(unsigned int index, unsigned int
c_default);
```
Provádí hledání znaku v tabulce.

main.c

void init();
Inicializuje potřebné proměnné.

void showhelp();
Zobrazí nápovědu příkazové řádky.

void parse_param(char *params);
Ošetří přepínače z příkazové řádky.

void showheader();
Zobrazí informační hlavičku programu.

int main(int argc, char *argv[]);
Hlavní program.

structs.h

```
typedef struct {
      unsigned int index;
      unsigned int new_char;
      int is_used;
} CHARSTRUCT;
```
Struktura pro uložení dat načtených z tabulkových souborů.

Uživatelská dokumentace

Přeložení programu

V případě použití kompilátoru `gcc`, je přeložení jednoduché – příkazem `make`, který vyhledá `Makefile` a podle něj program přeloží. Tento `Makefile` funguje s `gcc` určeným pro Linux i Windows:

```
eryx3> make
gcc -ansi -Wall    -c -o main.o main.c
gcc -ansi -Wall    -c -o load_table.o load_table.c
gcc -ansi -Wall    -c -o function.o function.c
gcc -ansi -Wall    -c -o file_test.o file_test.c
gcc -ansi -Wall    -c -o conv.o conv.c
gcc -ansi -Wall -o build/konv main.o load_table.o function.o file_test.o
conv.o
rm -f *.o
eryx3>
```

Pokud by se kompilovalo překladačem Open Watcom, je možné použít mnou vytvořený `makefile.watcom`, který je možno vykonat příkazem `nmake -f makefile.watcom` nebo spuštěním `make-watcom.bat`. Program se přeloží do adresáře build, kde se vyskytuje i testovací soubor test.bat (pro Windows), případně test.sh (pro Linux). V tomto adresáři se dále nachází adresář s potřebnými tabulkami pro konverzi.

Spuštění programu

Program se používá pomocí příkazové řádky pomocí parametrů. Pokud se program spustí bez parametrů nebo s parametrem –h či –help, zobrazí se nápověda:

```
konv verze 0.9.9g

Autor: Martin Arcao Sloup
E-mail: msloup@students.zcu.cz, arcao@arcao.com
Tato aplikace vznikla v ramci semestralni prace z predmetu
KIV/PT & KIV/PC na Zapadoceske Univerzite v Plzni.

Pouziti:

  konv [prepinace] [<vstupni_kodovani>] <vystupni_kodovani>
       <vstupni_soubor> <vystupni_soubor> [prepinace]

  <vstupni_kodovani> / <vystupni_kodovani> - kodovani definovane v adresari
                                             tables

  Prepinace:
    -b        vypnuti detekce binarniho souboru
    -nw       Windowsovske konce radku (<CR><LF>) ve vystupnim souboru
    -n[l|u]   Linuxovske konce radku (<LF>) ve vystupnim souboru
    -nm       Macovske konce radku (<CR>) ve vystupnim souboru
    -u        Vytvori na zacatku souboru v kodovani UTF-8 identifikacni bajty
              BOM (Byte Order Mark) - lze pouzit jen na kodovani UTF-8
    -obe      Pouzije se Big Endian poradi bytu u UTF-16 (pokud neni
              detekovano)
    -ole      Pouzije se Little Endian poradi bytu u UTF-16 (pokud neni
              detekovano)
```

```
Pokud neni uvedeno vstupni kodovani, program dokaze rozeznat nasledujici
kodovani vstupniho souboru:
  iso-8859-2, windows-1250, keybcs2 (kamenicti), koi-8_cs2, latin-2 (DOS
  Latin 2), mac-ce (apple-ce), utf-8, utf-16

Priklady:
  konv utf-8 vstupni.txt vystupni.txt
    Provede automatickou detekci vstupniho kodovani a overi, zda je soubor
    binarni. Vystupni kodovani bude UTF-8.

  konv -b utf-8 vstupni.txt vystupni.txt
    Provede automatickou detekci vstupniho kodovani, ale neoveri, zda je soubor
    binarni. Vystupni kodovani bude UTF-8.

  konv windows-1250 utf-8 vstupni.txt vystupni.txt
    Overi, zda je vstupni soubor binarni a provede konverzi z kodovani
    WINDOWS-1250 do UTF-8. Toto je vyhodne, pokud chceme delat konverzi na
    neceskych kodovanich.

  konv -nl utf-8 vstupni.txt vystupni.txt
    Overi, zda je vstupni soubor binarni, provede automatickou detekci
    vstupniho kodovani. Vystupni bude UTF-8. Pri konverzi pouzije Linuxove
    konce radku <LF>.

  konv -nm -u utf-8 vstupni.txt vystupni.txt
    Overi, zda je vstupni soubor binarni, provede automatickou detekci
    vstupniho kodovani. Vystupni bude UTF-8. Pri konverzi pouzije Macovske
    konce radku <CR> a na zacatek vystupniho souboru vlozi identifikacni bajty
    BOM (Byte Order Mark).
```

Pro spuštění je nutné mít v cestě, kde spouštíme program i složku s tabulkami, jinak se nepovede načíst příslušné tabulky. Bohužel v dokumentaci k ANSI C není nikde uvedeno jak zjistit adresář, kde se nachází spustitelný soubor.

Závěr

Při vytváření semestrální práce jsem prostudoval všechny dostupné materiály, které jsem našel na internetu. Nejvíce však server cestina.cz[3], sekce faq na unicode.org[4] a tabulky dvojznaků ze stránek Lukáše Petrlíka[5], které mi pomohli při tvorbě kódování bratrů Kamenických.

Program jsem otestoval na překládačích Microsoft Visual C++ 6, Open Watcom 1.3, přeportovaném gcc 3.4.2 (mingw) na platformu WIN32, gcc 3.3.5 na Linuxu (eryx3.zcu.cz). Během vývoje nastalo pár problému s uvolňováním paměti a hlavně s pointrama. Uvolňování paměti jsem snad zdárně vyřešil.

Přemýšlel jsem, jak by šel program zrychlit. Jediná možnost by bylo použít tabulku o velikosti 65536 položek a provést vyhledávání pouhým příslušným vybíráním indexu. Jednalo by se o zrychleni na úkor použité paměti.

[3] http://www.cestina.cz/
[4] http://www.unicode.org/faq/
[5] http://www.kiv.zcu.cz/~luki/cs/cs-encodings-faq.html