9.2.4 SymStrip

Application purpose

SymStrip takes a transcribed language corpus with the words tagged with their word type (vowel, noun, etc) and separates the tags and words into separate text files. Symstrip can mark superfluous tags and insert reset markers at sentence boundaries.

Usage

2 input files are needed:

- A tags file, with whitespace-separated tokens that are the tags of interest. All other tokens will be "stripped" into the "words" file.
- A marked-up corpus, in a text file. SymStrip does not know about tag definitions like enclosing braces. The entire tag, verbatim, must be placed into the "tags" file. Tags and words must be whitespace-separated in the corpus. The code could be modified to incorporate this type of tagging, but it would probably be easier to run the corpus through a script to insert spaces.

SymStrip will process the corpus to extract the defined tags into one output file, and the other text within the corpus into another output file.

Modules used

TokenLst, TokScan, StdDefs, HTable

Source code

Makefile

```
symstrip: ui.o symsyms.o symstrip.o symcmds.o htable.o symopts.o tokenlst.o

symsyms.o: symsyms.c

symcmds.o: symcmds.c

htable.o: htable.c

ui.o: ui.c

symopts.o: symopts.c

tokenlst.o: tokenlst.c

symstrip.o: symstrip.c
```

ui.h

```
/* ui.h -- User interface for symbolstripper
 */

#ifndef  __UI_H
#define  __UI_H

#include "tokenlst.h"
#include "stddefs.h"


/* --- Lexical enumeration --- */

enum CommandsTag {exec_symb,
                      exec_symb_init,
                      exec_symb_read,
                     exec_symb_add,
                      exec_symb_del,
                      exec_symb_clear,
                      exec_symb_linfo,
                     exec_symb_help,

                  exec_opt,
                     exec_opt_disp,
```

```
                    exec_opt_load,
                    exec_opt_save,
                    exec_opt_def,
                    exec_opt_sfile,
                    exec_opt_hsize,
                    exec_opt_extra,
                    exec_opt_extratag,
                    exec_opt_eos,
                    exec_opt_eostag,
                    exec_opt_help,

                exec_end, exec_strip, exec_core, exec_help, exec_info, exec_exit, INVALID};

typedef  enum CommandsTag  command;

enum MenusTag {menu_main, menu_symb, menu_opt};

typedef  enum MenusTag     menus;


/* -- Function prototypes -- */

void Help(void);
void Header(void);
void Footer(void);
void Info(void);
void Init(void);
void Cleanup(void);
void Prompt(void);
void CoreCheck(unsigned begin, unsigned end);
void SyntaxError(tokenList *comm);
void ExecuteMain(command comm);
bool StringCompI(char *string1, char *string2);
tokenList *GetInput(string buffer, int length);

#endif   /* __UI_H */

/* --- END of ui.h --- */
```

ui.c

```
/* UI -- User Interface for Symbol Stripper
 *
 * Author: Dylan Muir
 * Date: 11th Feburary 1999
 * Project: Language Processing Group -- QUT MLRC -- Sem 1 1999
 */

#include <stdlib.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "ui.h"
#include "stddefs.h"
#include "tokenlst.h"
#include "htable.h"
#include "symcmds.h"


/* --- Function prototypes --- */

command SyntaxCheck(tokenList *comm);
command SyntaxCheckMain(tokenList *comm);
void ExecuteCommand(command comm);


/* --- Globals --- */

time_t   tm;                 /* timestamp of program execution */
symOpts  options;            /* variables (defined in "symcmds.h") */
menus    menu;               /* Current execution menu */


/* --- Main --- */

int main(void)
{
   char        line[MAXSTRING];
```

```
   tokenList   *commands,
               *index;
   bool        contExec;           /* Should we continue execution? */
   command     comm;               /* Currently read command */

   unsigned    begin, end;         /* variables for checking memory usage */

   Header();                       /* Display program header */

   Init();                         /* Initiallise system */

   Help();                         /* Show help info */

#ifdef   DEBUG
/*   begin = coreleft(); */
#endif /* DEBUG */

   contExec = TRUE;

   while (contExec) {
      Prompt();              /* Display prompt */
      index = commands = GetInput(line, MAXSTRING);

      while (index != NULL) {
         if ((comm = SyntaxCheck(index)) == INVALID)
            SyntaxError(index);

         else if (comm == exec_exit) {
            contExec = FALSE;
            break;

         } else
            ExecuteCommand(comm);

         index = index -> NEXT;
      }

      commands = DestroyTokenList(commands);

#ifdef   DEBUG
/*      end = coreleft(); */

      CoreCheck(begin, end);

/*       begin = coreleft(); */
#endif /* DEBUG */
   }

   Footer();

   return 0;
}


/* --- Function bodies --- */

/* -- Function Help
 * Pre: TRUE
 * Post: Help info was printed
 */
void Help(void)
{
   printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
          "Commands:",
          "| strip - strip symbols",
          "| symbols - symbols commands",
          "| options - options commands",
          "| core - display the amout of free core memory",
          "| help - this screen",
          "| info - display info about the test driver",
          "| exit - leave program",
          "|",
          "| * -- Not implemented!");
}


/* -- Function Header
 * Pre: TRUE
 * Post: Test driver header info was printed
 */
void Header(void)
{
```

```
   tm = time(NULL);

   printf("\n%s\n%s%s%s\n%s\n%s\n%s %s%s\n\n",
          "+------------------------------+",
          "| SymbolStripper ver ", VERSION, "     |",
          "| QUT MLRC LPG (Sem 1 1999)    |",
          "+------------------------------+",
          "Executed on", ctime(&tm),
          "------------------------------------------------------------");
}


/* -- Function Footer
 * Pre: TRUE
 * Post: Test driver footer info was printed
 */
void Footer(void)
{
   tm = time(NULL);

   printf("%s\n%s%s\n%s %s\n",
          "------------------------------------------------------------",
          "QUT MLRC LPG -- SymbolStripper ver ", VERSION,
          "Terminated on", ctime(&tm));
}


/* -- Function Info
 * Pre: TRUE
 * Post: Test driver info was printed
 */
void Info(void)
{
   printf("\n%s %s\n%s\n%s\n%s %s %s\n%s %s%s %d\n%s %d\n\n",
          "SymbolStripper ver", VERSION,
          " Author: Dylan Muir [dr.muir@student.qut.edu.au]",
          " Project: QUT MLRC LPG (Sem1 1999)",
          " Compile time:", __TIME__, __DATE__,
          " Execution time:", ctime(&tm),
          " Max string:", MAXSTRING,
          " Max lookup key size:", MAX_KEY_SIZE);
}


/* -- Function Init
 * Pre: TRUE
 * Post: Performs any initiallisation required
 */
void Init(void)
{
   menu = menu_main;          /* Set main menu */

   InitOptionsCommand();      /* \__ Defined in SymCmds.h */
   LoadOptionsCommand();      /* /                        */
}

/* -- Function Cleanup
 * Pre: TRUE
 * Post: Cleans up anything necessary
 */
void Cleanup(void)
{
   if (options.symbolsRead)
      ClearSymbolsCommand();     /* Clear symbols if necessary */

   if (options.hashInit)
      hashDestroyTable();        /* Destroy lookup table if necessary */
}


/* -- Function Prompt
 * Pre: TRUE
 * Post: the prompt for 'menu' was output
 */
void Prompt(void)
{
   switch (menu) {
      case menu_main:
         printf("] ");
         break;

      case menu_symb:
```

```
         printf("symbols> ");
         break;

      case menu_opt:
         printf("options> ");
         break;

      default:
         printf("> ");
   }
}


/* -- Function CoreCheck
 * Pre:
 * Post:
 */
void CoreCheck(unsigned begin, unsigned end)
{
   if (begin != end)
      printf("Core discrepancy: %+d\n", ((int) end) - ((int) begin));
}


/* -- Function Getinput
 * Pre: 'buffer' has been allocated
 * Post: a line of text was read from stdin, converted to a list of tokens and returned
 */
tokenList *GetInput(string buffer, int length)
{
   char        read,             /* Character read from stdin */
               *index;           /* index into buffer */

   index = buffer;         /* Reset index */

   while (length > 0) {
      length--;
      if ((read = (char) getchar()) == EOL)
         length = 0;
      else
         *index++ = read;
   }
   *index = EOS;        /* Nul-terminate */

   printf("\n");

   return GetTokens(buffer);
}


/* -- Function SyntaxCheck
 * Pre:
 * Post
 */
command SyntaxCheck(tokenList *comm)
{
   switch (menu) {
      case menu_main:
         return SyntaxCheckMain(comm);

      case menu_symb:
         return SyntaxCheckSymbol(comm);

      case menu_opt:
         return SyntaxCheckOption(comm);

      default:
         return exec_end;
   }
}


/* -- Function SyntaxCheckMain
 * Pre:
 * Post
 */
command SyntaxCheckMain(tokenList *comm)
{

   /* Un-implemented commands */

   /*--- beginning of unimplemented commands  ---*-
```

```
   -*--- end of unimplemented commands        ---*/

   if (StringCompI(comm -> token, "strip"))
      return exec_strip;

   if (StringCompI(comm -> token, "symbols"))
      return exec_symb;

   if (StringCompI(comm -> token, "options"))
      return exec_opt;

   if (StringCompI(comm -> token, "core"))
      return exec_core;

   if (StringCompI(comm -> token, "help"))
      return exec_help;

   if (StringCompI(comm -> token, "info"))
      return exec_info;

   if (StringCompI(comm -> token, "exit"))
      return exec_exit;

   return INVALID;
}



/* -- Function SyntaxError
 * Pre:
 * Post:
 */
void SyntaxError(tokenList *comm)
{
   printf("Command not recognised: %s.\n\n", comm -> token);
}


/* -- Function ExecuteCommand
 * Pre:
 * Post:
 */
void ExecuteCommand(command comm)
{
   if (comm == exec_end) {
      menu = menu_main;
      return;
   }

   switch (menu) {
      case menu_symb:
         ExecuteSymbol(comm);
         return;

      case menu_opt:
         ExecuteOption(comm);
         return;

      case menu_main:
      default:
         ExecuteMain(comm);
         return;
   }
}


/* -- Function ExecuteMain
 * Pre:
 * Post:
 */
void ExecuteMain(command comm)
{
   switch (comm) {
      case exec_strip:
         StripCommand();
         break;

      case exec_symb:
         menu = menu_symb;
         SymbolHelpCommand();
         break;
```

```
      case exec_opt:
         menu = menu_opt;
         OptionHelpCommand();
         break;

      case exec_core:
        printf("Not implemented on Unix\n\n");
/*       printf("Free memory: %u\n\n", coreleft()); */
         break;

      case exec_info:
         Info();
         break;

      case exec_help:
         Help();
         break;

      default:
         ;/* Should never be reached */
   }
}


/* -- Function StringCompI
 * Pre: 'string1' and 'string2' are valid nul-terminated strings
 * Post: ('string' == 'string2' && TRUE was returned) ||
 *       ('string1' != 'string2' && FALSE was returned)
 */
bool StringCompI(char *string1, char *string2)
{
   char  *copy1 = string1,
         *copy2 = string2;

   if (strlen(string1) != strlen(string2))
      return FALSE;

   {  while (*copy1 != '\0') {
         if (toupper(*copy1++) != toupper(*copy2++))
            return FALSE;
      }
   }
   return TRUE;
}
```

symcmds.h

```
/* symcmds.h -- include file for symbolstripper ui
 *
 * Author: Dylan Muir
 * Date: 12th February, 1999
 * Modified: 15th Fenruary, 1999
 * Version: 1.04
 *
 * Purpose: Holds definitions for non-ui commands and options struct definition
 */

#ifndef  __SYMCMDS_H
#define  __SYMCMDS_H

#include "stddefs.h"
#include "ui.h"
#include "tokenlst.h"


/* -- SymbolStripper definitions -- */

#define  OPTIONS_FILE   "symopts.opt"
#define  VERSION        "1.05"

/* -- Options structure -- */

struct symOptsTag {
   char  symbolFile[MAXSTRING];             /* default file for symbols */
   bool  symbolsRead;                       /* have symbols been read in? */

   unsigned hashSize;                       /* size of hash table */
   bool     hashInit;                       /* has the htable been initiallised? */

   bool  markExtra;                         /* mark extra tags per word? */
   char  markExtraTag[MAXSTRING];           /* symbol to mark extra tags with */
```

```
   bool  markEOS;                           /* mark end of sentances? */
   char  markEOSTag[MAXSTRING];             /* symbol to mark eos with */
};

typedef struct symOptsTag symOpts;


/* -- Command functions -- */

void     StripCommand(void);

void     SymbolHelpCommand(void);
command  SyntaxCheckSymbol(tokenList *comm);
void     ExecuteSymbol(command comm);

void     InitSymbolsCommand(void);
void     ReadSymbolsCommand(void);
void     AddSymbolsCommand(void);
void     DeleteSymbolsCommand(void);
void     ClearSymbolsCommand(void);
void     LookupInfoSymbolsCommand(void);


void     OptionHelpCommand(void);
command  SyntaxCheckOption(tokenList *comm);
void     ExecuteOption(command comm);

void     InitOptionsCommand(void);
void     LoadOptionsCommand(void);
void     SaveOptionsCommand(void);
void     LoadDefaultOptionsCommand(void);
void     DisplayOptionsCommand(void);
void     HashSizeOptionsCommand(void);
void     SymbolFileOptionsCommand(void);
void     ExtraOptionsCommand(void);
void     ExtraTagOptionsCommand(void);
void     EOSOptionsCommand(void);
void     EOSTagOptionsCommand(void);


/* -- Helper functions -- */

/* -- Function GetInputFileName
 * Pre: 'buffer' has been allocated and prompt has been printed
 * Post: 'buffer' contains a string that could be opened for reading
 */
void GetInputFileName(char *fName);

/* -- Function GetNewFileName
 * Pre: 'buffer' has been allocated and prompt has been printed
 * Post: 'buffer' contains a string that could be created as a file
 */
void GetNewFileName(char *fName);

/* -- Function ReadInput
 * Pre: 'buffer' has been allocated, and 'length' <= len(buffer)
 * Post: Up to 'length' characters have been read from stdin, and are in 'buffer'
 */
void ReadInput(char *buffer, unsigned length);

#endif   /* __SYMCMDS_H */

/* --- END of symcmds.h --- */
```

SymCmds.c

```
/* SymCmds.c -- functions called by UI
 *
 * See SymCmds.h for details
 */

#include <stdio.h>
#include "symcmds.h"
#include "symstrip.h"


void StripCommand(void)
{
   char  inputFile[MAXSTRING],        /* file to process */
         tagsFile[MAXSTRING],         /* file to strip tags to */
```

```
         wordsFile[MAXSTRING];           /* file to strip words to */

   printf("Enter an input file\n");
   GetInputFileName(inputFile);

   printf("\nEnter a file to strip tags to\n");
   GetNewFileName(tagsFile);

   printf("\nEnter a file to strip words to\n");
   GetNewFileName(wordsFile);

   printf("\nStripping file [%s]\n", inputFile);
   printf(" |\n");
   printf(" +--(tags)--> [%s]\n", tagsFile);
   printf(" |\n");
   printf(" +--(words)-> [%s]\n", wordsFile);
   printf(" \n");

   SymStrip(inputFile, tagsFile, wordsFile);
}


void GetInputFileName(char *fName)
{
   FILE  *fTest;              /* Used for testing validity of filename */
   bool  validFile;
   char  buffer[MAXSTRING];

   validFile = FALSE;

   while (!validFile) {
      printf("file> ");
      ReadInput(buffer, MAXSTRING);
      sscanf(buffer, "%s", fName);

      if ((fTest = fopen(fName, "rb")) != NULL) {
         fclose(fTest);
         validFile = TRUE;
         break;
      }

      printf("Unable to open [%s]\n", fName);
   }
}

void GetNewFileName(char *fName)
{
   FILE  *fTest;              /* Used for testing validity of filename */
   bool  validFile;
   char  buffer[MAXSTRING];

   validFile = FALSE;

   while (!validFile) {
      printf("file> ");
      ReadInput(buffer, MAXSTRING);
      sscanf(buffer, "%s", fName);

      if ((fTest = fopen(fName, "wb+")) != NULL) {
         fclose(fTest);
         validFile = TRUE;
         break;
      }

      printf("Unable to create [%s]\n", fName);
   }
}

void ReadInput(char *buffer, unsigned length)
{
   char  read;

   while (length > 0) {
      length--;
      if ((read = (char) getchar()) == EOL)
         length = 0;
      else
         *buffer++ = read;
   }
   *buffer = EOS;       /* Nul-terminate */
}
```

```
/* --- END or symcmds.c --- */
```

SymSyms.c

```
/* Symsyms.c -- commands file for symbolstripper UI
 *
 * See SymCmds.h for details
 */

#include <stdio.h>
#include "htable.h"
#include "symcmds.h"
#include "symstrip.h"

extern symOpts options;


void SymbolHelpCommand(void)
{
   printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
          "Symbols commands:",
          "| init - (re)initiallise lookup system",
          "| read - read in symbols from file",
          "| add - add symbols",
          "| delete - delete symbols",
          "| clear - clear all symbols and destroy lookup table",
          "| linfo - display info about lookup table",
          "| help - this page",
          "| end - return to main command input");
}


command SyntaxCheckSymbol(tokenList *comm)
{
   /* Un-implemented commands */

   /*--- beginning of unimplemented commands  ---*-
   -*--- end of unimplemented commands        ---*/

   if (StringCompI(comm -> token, "init"))
      return exec_symb_init;

   if (StringCompI(comm -> token, "read"))
      return exec_symb_read;

   if (StringCompI(comm -> token, "add"))
      return exec_symb_add;

   if (StringCompI(comm -> token, "delete"))
      return exec_symb_del;

   if (StringCompI(comm -> token, "clear"))
      return exec_symb_clear;

   if (StringCompI(comm -> token, "linfo"))
      return exec_symb_linfo;

   if (StringCompI(comm -> token, "help"))
      return exec_symb_help;

   if (StringCompI(comm -> token, "end"))
      return exec_end;


   return INVALID;
}


void  ExecuteSymbol(command comm)
{
   switch (comm) {
      case exec_symb_init:
         InitSymbolsCommand();
         break;

      case exec_symb_read:
         ReadSymbolsCommand();
         break;

      case exec_symb_add:
         AddSymbolsCommand();
```

```
         break;

      case exec_symb_del:
         DeleteSymbolsCommand();
         break;

      case exec_symb_clear:
         ClearSymbolsCommand();
         break;

      case exec_symb_linfo:
         LookupInfoSymbolsCommand();
         break;

      case exec_symb_help:
         SymbolHelpCommand();
         break;

      default:
         ;/* Should never be reached */
   }
}


void InitSymbolsCommand(void)
{
   if (options.hashInit)
      hashDestroyTable();

   options.hashInit = hashInitTable(options.hashSize);
}


void ReadSymbolsCommand(void)
{
   FILE           *symbolFile;                /* file pointer */
   bool           inFile;                     /* are we in the file? */
   tokenList      *tokens,                    /* list of tokens extracted from file */
                  *index;                     /* index into token list */
   char           *text;                      /* temp token text holder */
   long unsigned  symCount;                   /* number of symbols loaded */

   if (options.symbolsRead)
      ClearSymbolsCommand();               /* Clear tokens if already read */

   if (!options.hashInit)                  /* ensure table is initiallised */
      options.hashInit = hashInitTable(options.hashSize);

   printf("Reading symbols from [%s]...\n", options.symbolFile);

   symbolFile = fopen(options.symbolFile, "rt");  /* open file */

   if (symbolFile == NULL) {               /* error opening file */
      printf("Could not open [%s]\n", options.symbolFile);
      printf("*** SYMBOLS NOT READ\n");
      return;
   }

   inFile = TRUE;                          /* we are in the file */
   symCount = 0;

   while (inFile) {
      tokens = ReadLineTokens(symbolFile, &inFile);

      index = tokens;

      while (index != NULL) {
         text = index -> token;            /* Avoid unnecessary indexing */

         if (!hashIsValidKey(text))
            printf("*** Symbol [%s] not a valid key\n", text);

         else if (hashIsIn(text))
            printf("*** Symbol [%s] already in table\n", text);

         else if (hashIsFull())
            printf("*** Symbol [%s] not added -- table full\n", text);

         else {
            if (!hashInsert(index -> token))
               printf("*** Symbol [%s] not added -- undefined error\n", text);
```

```
            else {
               symCount++;
               if (hashLoadFactor() > 80)
                  printf("--- Warning: lookup table load factor is %u\%\n", hashLoadFactor());
            }

         }

         index = index -> NEXT;
      }

      DestroyTokenList(tokens);
   }

   fclose(symbolFile);

   options.symbolsRead = TRUE;

   printf("...loaded %u symbols\n\n", symCount);
}


void AddSymbolsCommand(void)
{
   tokenList   *symbols,            /* list of symbols to add */
               *index;              /* index into otoken list */
   char        buffer[MAXSTRING],   /* buffer for input */
               *text;               /* temp to prevent excessive indexing */

   if (!options.hashInit) {
      hashInitTable(options.hashSize);
      options.hashInit = TRUE;
   }

   printf("Enter symbols to add, separated by a space\n");

   printf("add> ");
   index = symbols = GetInput(buffer, MAXSTRING);

   while (index != NULL) {
      text = index -> token;             /* Avoid unnecessary indexing */

      if (!hashIsValidKey(text))
         printf("*** Symbol [%s] not a valid key\n", text);

      else if (hashIsIn(text))
         printf("*** Symbol [%s] already in table\n", text);

      else if (hashIsFull())
         printf("*** Symbol [%s] not added -- table full\n", text);

      else
         if (!hashInsert(index -> token))
            printf("*** Symbol [%s] not added -- undefined error\n", text);

      if (hashLoadFactor() > 80)
         printf("--- Warning: lookup table load factor is %d\%\n");

      index = index -> NEXT;
   }

   DestroyTokenList(symbols);

   printf("...done\n\n");
}


void DeleteSymbolsCommand(void)
{
   tokenList   *symbols,            /* list of symbols to delete */
               *index;              /* index into otoken list */
   char        buffer[MAXSTRING],   /* buffer for input */
               *text;               /* temp to prevent excessive indexing */

   if (!options.hashInit) {
      printf("No symbols to delete\n");
      return;
   }

   printf("Enter symbols to delete, separated by a space\n");

   printf("del> ");
```

```
   index = symbols = GetInput(buffer, MAXSTRING);

   while (index != NULL) {
      text = index -> token;             /* Avoid unnecessary indexing */

      if (!hashIsValidKey(text))
         printf("*** Symbol [%s] not a valid key\n", text);

      else if (hashIsEmpty())
         printf("*** Symbol [%s] not deleted -- table empty\n", text);

      else if (!hashIsIn(text))
         printf("*** Symbol [%s] is not in table\n", text);

      else
         if (!hashDelete(index -> token))
            printf("*** Symbol [%s] not deletd -- undefined error\n", text);

      index = index -> NEXT;
   }

   DestroyTokenList(symbols);

   printf("...done\n\n");
}

void ClearSymbolsCommand(void)
{
   if (!options.hashInit)                /* No symbols to clear */
      return;

   printf("Clearing all symbols...");

   if (options.hashInit) hashDestroyTable(); /* Destroy table */
   options.hashInit = FALSE;                 /* Set flags */
   options.symbolsRead = FALSE;

   printf(" done\n\n");
}

void LookupInfoSymbolsCommand(void)
{
   printf("Lookup table info:\n");
   if (!options.hashInit) {
      printf(" Table not initiallised\n\n");
      return;
   }

   printf(" Maximum entries: %u\n", options.hashSize);
   printf(" Loaded symbols: %u\n", hashNumEntries());
   printf(" Load factor: %u\%\n", hashLoadFactor());

   printf(" Table full? ");
   if (hashIsFull())
      printf("YES\n");
   else
      printf("NO\n");

   printf(" Table empty? ");
   if (hashIsEmpty())
      printf("YES\n");
   else
      printf("NO\n");

   printf(" Max key length: %d\n", MAX_KEY_SIZE);

   printf("\n");
}

/* --- END of symsyms.c --- */
```

SymOpts.c

```
/* Symopts.c -- manipulates system options
 *
 * See SymBmds.h for details
 */

#include <stdio.h>
#include <string.h>
#include "symcmds.h"
```

```
#include "htable.h"

extern symOpts options;


void OptionHelpCommand(void)
{
   printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
          "Options commands:",
          "| display - show all options",
          "| load - load options",
          "| save - save options",
          "| default - load default options",
          "| symfile - specify symbol file",
          "| hashsize - specify hash table size",
          "| extra - toggle marking extra symbols per word",
          "| extratag - specify extra symbol mark tag",
          "| eos - toggle use of 'end of sentance' tag",
          "| eostag - specify 'end of sentance' tag",
          "| help - this page",
          "| end - return to main command input");
}


command SyntaxCheckOption(tokenList *comm)
{
   /* Un-implemented commands */

   /*--- beginning of unimplemented commands  ---*-
   -*--- end of unimplemented commands        ---*/

   if (StringCompI(comm -> token, "display"))
      return exec_opt_disp;

   if (StringCompI(comm -> token, "load"))
      return exec_opt_load;

   if (StringCompI(comm -> token, "save"))
      return exec_opt_save;

   if (StringCompI(comm -> token, "default"))
      return exec_opt_def;

   if (StringCompI(comm -> token, "symfile"))
      return exec_opt_sfile;

   if (StringCompI(comm -> token, "hashsize"))
      return exec_opt_hsize;

   if (StringCompI(comm -> token, "extra"))
      return exec_opt_extra;

   if (StringCompI(comm -> token, "extratag"))
      return exec_opt_extratag;

   if (StringCompI(comm -> token, "eos"))
      return exec_opt_eos;

   if (StringCompI(comm -> token, "eostag"))
      return exec_opt_eostag;

   if (StringCompI(comm -> token, "help"))
      return exec_opt_help;

   if (StringCompI(comm -> token, "end"))
      return exec_end;


   return INVALID;
}

void ExecuteOption(command comm)
{
   switch (comm) {
      case exec_opt_disp:
         DisplayOptionsCommand();
         break;

      case exec_opt_load:
         LoadOptionsCommand();
         break;
```

```
      case exec_opt_save:
         SaveOptionsCommand();
         break;

      case exec_opt_def:
         LoadDefaultOptionsCommand();
         break;

      case exec_opt_sfile:
         SymbolFileOptionsCommand();
         break;

      case exec_opt_hsize:
         HashSizeOptionsCommand();
         break;

      case exec_opt_extra:
         ExtraOptionsCommand();
         break;

      case exec_opt_extratag:
         ExtraTagOptionsCommand();
         break;

      case exec_opt_eos:
         EOSOptionsCommand();
         break;

      case exec_opt_eostag:
         EOSTagOptionsCommand();
         break;

      case exec_opt_help:
         OptionHelpCommand();
         break;

      default:
         ;/* Should never be reached */
   }
}


void InitOptionsCommand(void)
{
   strcpy(options.symbolFile, "");
   options.symbolsRead = FALSE;

   options.hashSize = 0;
   options.hashInit = FALSE;

   options.markExtra = FALSE;
   strcpy(options.markExtraTag, "");

   options.markEOS = FALSE;
   strcpy(options.markEOSTag, "");
}

void LoadOptionsCommand(void)
{
   FILE  *optionsFile;

   char  version[5];

   printf("Loading options from [%s]...\n", OPTIONS_FILE);

   optionsFile = fopen(OPTIONS_FILE, "rb");

   if (optionsFile == NULL) {
      printf("Unable to open [%s]\n", OPTIONS_FILE);
      printf("Loading default options...\n");

      LoadDefaultOptionsCommand();          /* Load defaults */
      DisplayOptionsCommand();              /* Show the defaults */
      SaveOptionsCommand();

   } else {
      fread(version, 5, 1, optionsFile);    /* first 5 bytes == version */

      if (strcmp(version, VERSION) != 0) {  /* compare SymbolStripper versions */
         printf("This options file was created with version [%s]\n", version);
         printf("You are running version [%s]\n", VERSION);
         printf("Loading default options...\n");
```

```
         LoadDefaultOptionsCommand();          /* Load defaults */
         DisplayOptionsCommand();              /* Show the defaults */

      } else {
         fread(&options, sizeof(options), 1, optionsFile); /* Read options */
         fclose(optionsFile);                      /* Close file */
      }
   }

   options.symbolsRead = FALSE;                /* reset symbolsRead flag */
   options.hashInit = FALSE;                   /* reset hashInit flag */

   printf("...done.\n\n");
}


void SaveOptionsCommand(void)
{
   FILE  *optionsFile;

   char  version[5];

   printf("Saving options to [%s]...\n", OPTIONS_FILE);

   optionsFile = fopen(OPTIONS_FILE, "wb");

   if (optionsFile == NULL) {
      printf("Unable to open [%s]\n", OPTIONS_FILE);
      printf("*** OPTIONS NOT SAVED\n");

   } else {
      strcpy(version, VERSION);                /* assign version string */

      fwrite(version, 5, 1, optionsFile);      /* version -> first five bytes */

      fwrite(&options, sizeof(options), 1, optionsFile);/* Write options */
      fclose(optionsFile);                     /* Close file */
   }

   printf("...done.\n\n");
}


void LoadDefaultOptionsCommand(void)
{
   if (options.hashInit)
      ClearSymbolsCommand();

   strcpy(options.symbolFile, "symbols.txt");
   options.symbolsRead = FALSE;

   options.hashSize = 201;
   options.hashInit = FALSE;

   options.markExtra = TRUE;
   strcpy(options.markExtraTag, "*");

   options.markEOS = TRUE;
   strcpy(options.markEOSTag, "/S");
}


void DisplayOptionsCommand(void)
{
   printf("\n--- SymbolStripper settings ---\n");

   printf("Current symbol file -- [%s]\n", options.symbolFile);

   printf("Symbols read? ");
   if (options.symbolsRead)
      printf("YES\n");
   else
      printf("NO\n");

   printf("Maximum lookup table size: %u\n", options.hashSize);

   printf("Lookup table initiallised? ");
   if (options.hashInit)
      printf("YES\n");
   else
      printf("NO\n");
```

```
   printf("Mark extra symbols per word with [%s]? ", options.markExtraTag);
   if (options.markExtra)
      printf("YES\n");
   else
      printf("NO\n");

   printf("Mark end of sentance with [%s]? ", options.markEOSTag);
   if (options.markEOS)
      printf("YES\n");
   else
      printf("NO\n");


   printf("\n");
}


void HashSizeOptionsCommand(void)
{
   unsigned hashSize;

   printf("Current maximum hash size: %u\n", options.hashSize);
   printf("Enter new maximum hash size: ");
   scanf("%u", &hashSize);

   if (hashSize == options.hashSize)        /* If it's the same, */
      return;                               /* don't do anything */

   else {
      if (options.hashInit)
         ClearSymbolsCommand();

      options.hashSize = hashSize;

      if (options.hashInit) {
         hashDestroyTable();
         options.hashInit = FALSE;
      }

      options.symbolsRead = FALSE;
   }

   printf("...done.\n");
}

void ExtraOptionsCommand(void)
{
   printf("Marking of extra symbols per word ");

   if (options.markExtra) {
      options.markExtra = FALSE;
      printf("OFF\n");

   } else {
      options.markExtra = TRUE;
      printf("ON\n");
   }
}

void ExtraTagOptionsCommand(void)
{
   printf("Current extra symbols tag: [%s]\n", options.markExtraTag);
   printf("Enter new extra symbols tag: ");
   scanf("%s", &options.markExtraTag);

   printf("...done\n");
}

void EOSOptionsCommand(void)
{
   printf("Marking of end of sentance is ");

   if (options.markEOS) {
      options.markEOS = FALSE;
      printf("OFF\n");

   } else {
      options.markEOS = TRUE;
      printf("ON\n");
   }
}
```

```
void EOSTagOptionsCommand(void)
{
   printf("Current eos tag: [%s]\n", options.markEOSTag);
   printf("Enter new eos tag: ");
   scanf("%s", &options.markEOSTag);

   printf("...done\n");
}


void SymbolFileOptionsCommand(void)
{
   char  symbolFile[MAXSTRING];       /* New symbol file name */

   printf("Current symbol file: [%s]\n", options.symbolFile);
   printf("Enter new symbol file\n");

   GetInputFileName(symbolFile);

   if (StringCompI(symbolFile, options.symbolFile))
      return;                              /* no change, so exit */

   strcpy(options.symbolFile, symbolFile);  /* assign new file */

   if (options.symbolsRead) {
      hashDestroyTable();                  /* Remove prior symbols */
      options.symbolsRead = FALSE;         /* Set flags */
      options.hashInit = FALSE;
   }

   printf("... done\n");
}
```

SymStrip.h

```
/* SymStrip.h -- Symbol stripping engine
 *
 * Author: Dylan Muir [dr.muir@student.qut.edu.au
 * Date: 14th February, 1999
 * Version: 1.00
 */

#ifndef  __SYMSTRIP_H
#define  __SYMSTRIP_H

#include <stdio.h>
#include "tokenlst.h"
#include "stddefs.h"


/* -- Function SymStrip
 * Pre: 'inputFile' is an existing file
 *      'tagsFile' and 'wordsFile' can be created
 * Post: 'inputFile' has been processed &&
 *       tags were stripped to 'tagsfile' &&
 *       words were strripped to 'wordsFile'
 */
void  SymStrip(string inputFile, string tagsFile, string wordsFile);


/* -- Function ReadLineTokens
 * Pre: 'file' is a text file opened for reading
 *      'inFile' has been allocated, and indicates wether EOF has been reached
 * Post: a line of text up to an EOL or EOF has been scanned and
 *       converted into a token list, which was returned
 */
tokenList   *ReadLineTokens(FILE *ifile, bool *inFile);

/* -- Function ReadLineText
 * Pre: 'file' is a text file opened for reading
 *      'buffer' has been allocated, and has max length 'length'
 *      'inFile' has been allocated, and indicated wether EOF has been reached
 * Post: 'buffer' contains a line of text up to EOL or EOF (defined in stddefs.h) &&
 *       'inFile' indicates wether EOF has been reached
 */
void  ReadLineText(FILE *ifile, char *buffer, unsigned length, bool *inFile);

#endif   /* __SYMSTRIP_H */

/* --- END of SymStrip.h --- */
```

SymStrip.c

```
/* symstrip.c -- Symbol stripping engine
 *
 * See symstrip.h for details
 */

#include "stddefs.h"
#include "symcmds.h"
#include "symstrip.h"
#include "tokenlst.h"
#include "htable.h"

extern symOpts options;

/* -- Local functions -- */

void NewLine(FILE *FP);
void MarkEOS(FILE *FP);
void MarkExtra(FILE *FP);
void WriteToken(FILE *FP, tokenList *token);
bool CheckSequence(FILE *tagsFP, FILE *wordsFP, string tag, string word, tokenList *index);

/* -- Exported functions -- */

void SymStrip(string inputFile, string tagsFile, string wordsFile)
{
   FILE  *inputFP,
         *tagsFP,
         *wordsFP;

   tokenList   *list,                  /* line of tokens from file */
               *index;                 /* index into token list */

   bool        inFile;                 /* Are we within the file? */
   bool        readTag;                /* Did we just read a tag? */
   unsigned    lineCount,              /* Number of lines */
               wordCount;              /* Number of words (non-tags) read */

   if (!options.symbolsRead) {                 /* Ensure we have some tags to work with! */
      printf("Reading symbols...\n");
      ReadSymbolsCommand();

      if (!options.symbolsRead) {              /* Unsuccessful symbol read */
         printf("Unable to open specified symbols file.\n");
         printf("Aborting...\n");
         return;                    /* So die */
      }
   }

   if ((inputFP = fopen(inputFile, "rt")) == NULL) {
      printf("Unable to open [%s] for input\n", inputFile);
      printf("Aborting...\n");
      return;
   }

   if ((tagsFP = fopen(tagsFile, "wt")) == NULL) {
      printf("Unable to open [%s] for tags output\n", tagsFile);
      printf("Aborting...\n");

      fclose(inputFP);
      return;
   }

   if ((wordsFP = fopen(wordsFile, "wt")) == NULL) {
      printf("Unable to open [%s] for words output\n", wordsFile);
      printf("Aborting...\n");

      fclose(inputFP);
      fclose(tagsFP);
      return;
   }


   inFile = TRUE;                              /* We are currently in the file */
   readTag = FALSE;                            /* No tags read yet */

   lineCount = 0;                              /* \___ Reset counts */
   wordCount = 0;                              /* /                 */
```

```
   printf("Stripping file...\n");

   while (inFile) {
      index = list = ReadLineTokens(inputFP, &inFile);  /* Get a line of tokens */

      while (index != NULL) {
         if (hashIsIn(index -> token)) {       /* Is it a tag? */
         if (readTag)                          /* Is it an extra tag? */
                  MarkExtra(tagsFP);                /* Mark it */

      WriteToken(tagsFP, index);            /* Strip to tags file */
      readTag = TRUE;                    /* We just read a tag */

         } else {
            WriteToken(wordsFP, index);        /* Strip to words file */
            readTag = FALSE;                   /* We didn't read a tag */

            wordCount++;                       /* read another word */
         }

         index = index -> NEXT;                /* Next tokoen */
      }

      DestroyTokenList(list);

      MarkEOS(tagsFP);                         /* End of sentance */
      NewLine(tagsFP);                         /* \___ New lines */
      NewLine(wordsFP);                        /* /              */

      lineCount++;                             /* read another line */
   }

   fclose(inputFP);                            /* \                 */
   fclose(tagsFP);                             /*  >--- Close files */
   fclose(wordsFP);                             /* /                 */

   printf("Stripped [%s]: %u words, %u lines\n\n",
          inputFile, wordCount, lineCount);
}


tokenList *ReadLineTokens(FILE *ifile, bool *inFile)
{
   char  buffer[MAXSTRING * 4];

   ReadLineText(ifile, buffer, MAXSTRING * 4, inFile);

   return GetTokens(buffer);
}

void ReadLineText(FILE *ifile, char *buffer, unsigned length, bool *inFile)
{
   char        read;                  /* Character read from 'file' */
   unsigned    count;                 /* number of characters in 'buffer' */

   count = 0;

   fread(&read, 1, 1, ifile);

   while (read != EOL && *inFile) {
      if (feof(ifile))
         *inFile = FALSE;

      else {
         *buffer++ = read;
         count++;

         if (count == length - 1)     /* Filled the buffer */
            break;
      }

      fread(&read, 1, 1, ifile);
   }

   *buffer = EOS;
}


/* -- Local functions -- */

/* -- Function NewLine
 * Pre: 'FP' is a file opened for writing
```

```
 * Post: a new line was written to 'FP'
 */
void NewLine(FILE *FP)
{
   fprintf(FP, "\n");
}


/* -- Function MarkEOS
 * Pre: 'FP' is a file opened for writing
 * Post: (options.markEOS == TRUE && EOS marker was written to file) ||
 *       (options.markEOS == FALSE)
 */
void MarkEOS(FILE *FP)
{
   if (options.markEOS)
      fprintf(FP, "%s", options.markEOSTag);
}


/* -- Function MarkExtra
 * Pre: 'FP' is a file opened for writing
 * Post: (options.markExtra == TRUE && Extra tag marker was written to file) ||
 *       (options.markExtra == FALSE)
 */
void MarkExtra(FILE *FP)
{
   if (options.markExtra)
      fprintf(FP, "%s", options.markExtraTag);
}

/* -- Function WriteToken
 * Pre: 'FP' is a file opened for writing
 *      'token' is a valid token
 * Post: 'token' + " " (space) was written to 'FP'
 */
void WriteToken(FILE *FP, tokenList *token)
{
   fprintf(FP, "%s ", token -> token);
}

/* -- Function CheckSequence
 * Pre: 'tagsFP' is a file opened for writing; destination for tags
 *      'wordsFP' is a file opened for writing; destination for words
 *      'tag' is a string to check if it matches a tag
 *      'word' is a string so that if it comes after 'tag' then it must be a word
 *      'index' is an index into a token list
 * Post: If index == 'tag' and index -> NEXT == 'word' then
 *        'tag' --> tags
 *        'word --> words && TRUE is returned
 *       Otherwise 'tag' --> tags && FALSE is returned
 *
 * Note: Due to the stupidity of the WALES corpus designers, many tags were used with the same
 *       names as common words, with no way to identify wether one was a tag or a word.
 */
int CheckSequence(   FILE *tagsFP, FILE *wordsFP,
         string tag, string word,
         tokenList *index)
{
   if ((strcmp(index -> token, tag) == 0) && (strcmp(index -> NEXT -> token, word) == 0)) {
      WriteToken(tagsFP, index);
      WriteToken(tagsFP, index -> NEXT);
      return TRUE;
   } else
      return FALSE;
}


/* --- END of SymStrip.c --- */
```