```
// $Id: debugf.h,v 1.2 2012-11-20 18:22:12-08 - - $

#ifndef __DEBUGF_H__
#define __DEBUGF_H__

//
// DESCRIPTION
//    Debugging library containing miscellaneous useful things.
//

//
// Keep track of Exec_Name and Exit_Status.
//
extern char *Exec_Name;
extern int Exit_Status;

//
// Support for stub messages.
//
#define STUBPRINTF(...) \
        __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
void __stubprintf (char *file, int line, const char *func,
                   char *format, ...);

//
// Debugging utility.
//

void set_debugflags (char *flags);
   //
   // Sets a string of debug flags to be used by DEBUGF statements.
   // Uses the address of the string, and does not copy it, so it
   // must not be dangling.  If a particular debug flag has been set,
   // messages are printed.  The format is identical to printf format.
   // The flag "@" turns on all flags.
   //

#ifdef NDEBUG
#define DEBUGF(FLAG,...) // DEBUG (FLAG, __VA_ARGS__)
#else
#define DEBUGF(FLAG,...) \
        __debugprintf (FLAG, __FILE__, __LINE__, __VA_ARGS__)
void __debugprintf (char flag, char *file, int line,
                    char *format, ...);
#endif

#endif
```

```c
// $Id: hashset.h,v 1.1 2012-11-16 18:05:22-08 - - $

#ifndef __HASHSET_H__
#define __HASHSET_H__

#include <stdbool.h>

typedef struct hashset *hashset_ref;

//
// Create a new hashset with a default number of elements.
//
hashset_ref new_hashset (void);

//
// Frees the hashset, and the words it points at.
//
void free_hashset (hashset_ref);

//
// Inserts a new string into the hashset.
//
void put_hashset (hashset_ref, char*);

//
// Looks up the string in the hashset and returns true if found,
// false if not found.
//
bool has_hashset (hashset_ref, char*);

#endif

```

```
// $Id: strhash.h,v 1.1 2012-11-16 18:05:22-08 - - $

//
// NAME
//    strhash - return an unsigned 32-bit hash code for a string
//
// SYNOPSIS
//    hashcode_t strhash (char *string);
//
// DESCRIPTION
//    Uses Horner’s method to compute the hash code of a string
//    as is done by java.lang.String.hashCode:
//    .  s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
//    Using strength reduction, the multiplication is replaced by
//    a shift.  However, instead of returning a signed number,
//    this function returns an unsigned number.
//
// REFERENCE
//    http://java.sun.com/j2se/1.4.1/docs/api/java/lang/
//    String.html#hashCode()
//
//

#ifndef __STRHASH_H__
#define __STRHASH_H__

#include <inttypes.h>

typedef uint32_t hashcode_t;

hashcode_t strhash (char *string);

#endif

```

```
// $Id: yyextern.h,v 1.1 2012-11-16 18:05:22-08 - - $

#ifndef __YYEXTERN_H__
#define __YYEXTERN_H__

//
// DESCRIPTION
//    Definitions of external names used by flex-generated code.
//

#include <stdio.h>

extern FILE *yyin;             // File currently being read

extern char *yytext;           // Pointer to the string that was found

extern int yy_flex_debug;      // yylex's verbose tracing flag

extern int yylex (void);       // Read next word from opened file yyin

extern int yylineno;           // Line number within the current file

extern void yycleanup (void); // Cleans up flex's buffers when done

#endif

```

```c
// $Id: debugf.c,v 1.3 2012-11-20 18:25:15-08 - - $

#include <errno.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "debugf.h"

char *Exec_Name = NULL;
int Exit_Status = EXIT_SUCCESS;

static char *debugflags = "";
static bool alldebugflags = false;

void __stubprintf (char *filename, int line, const char *func,
                   char *format, ...) {
   va_list args;
   fflush (NULL);
   fprintf (stdout, "%s: STUB (%s:%d) %s:\n",
            Exec_Name, filename, line, func);
   va_start (args, format);
   vfprintf (stdout, format, args);
   va_end (args);
   fflush (NULL);
}

void set_debugflags (char *flags) {
   debugflags = flags;
   if (strchr (debugflags, '@') != NULL) alldebugflags = true;
   DEBUGF ('a', "Debugflags = \"%s\"\n", debugflags);
}

void __debugprintf (char flag, char *file, int line,
                    char *format, ...) {
   va_list args;
   if (alldebugflags || strchr (debugflags, flag) != NULL) {
      fflush (NULL);
      fprintf (stderr, "%s: DEBUGF(%c) (%s:%d):\n",
               Exec_Name, flag, file, line);
      va_start (args, format);
      vfprintf (stderr, format, args);
      va_end (args);
      fflush (NULL);
   }
}

```

```c
// $Id: hashset.c,v 1.1 2012-11-16 18:05:22-08 - - $

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "debugf.h"
#include "hashset.h"
#include "strhash.h"

#define HASH_NEW_SIZE 15

struct hashset {
   size_t length;
   int load;
   char **array;
};

hashset_ref new_hashset (void) {
   hashset_ref new = malloc (sizeof (struct hashset));
   assert (new != NULL);
   new->length = HASH_NEW_SIZE;
   new->load = 0;
   new->array = malloc (new->length * sizeof (char*));
   for (size_t index = 0; index < new->length; ++index) {
      new->array[index] = NULL;
   }
   assert (new->array != NULL);
   DEBUGF ('h', "%p -> struct hashset {length = %d, array=%p}\n",
                 new, new->length, new->array);
   return new;
}

void free_hashset (hashset_ref hashset) {
   DEBUGF ('h', "free (%p), free (%p)\n", hashset->array, hashset);
   memset (hashset->array, 0, hashset->length * sizeof (char*));
   free (hashset->array);
   memset (hashset, 0, sizeof (struct hashset));
   free (hashset);
}

void put_hashset (hashset_ref hashset, char *item) {
   STUBPRINTF ("hashset=%p, item=%s\n", hashset, item);
}

bool has_hashset (hashset_ref hashset, char *item) {
   STUBPRINTF ("hashset=%p, item=%s\n", hashset, item);
   return true;
}

```

```c
// $Id: strhash.c,v 1.1 2012-11-16 18:05:22-08 - - $

#include <assert.h>
#include <stdio.h>
#include <sys/types.h>

#include "strhash.h"

hashcode_t strhash (char *string) {
   assert (string != NULL);
   hashcode_t hashcode = 0;
   for (int index = 0; string[index] != '\0'; ++index) {
      hashcode = hashcode * 31 + (unsigned char) string[index];
   }
   return hashcode;
}

```

```c
// $Id: spellchk.c,v 1.2 2012-11-20 18:22:12-08 - - $

#include <errno.h>
#include <libgen.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "debugf.h"
#include "hashset.h"
#include "yyextern.h"

#define STDIN_NAME       "-"
#define DEFAULT_DICTNAME "/usr/share/dict/words"
#define DEFAULT_DICT_POS 0
#define EXTRA_DICT_POS   1
#define NUMBER_DICTS     2

void print_error (char *object, char *message) {
   fflush (NULL);
   fprintf (stderr, "%s: %s: %s\n", Exec_Name, object, message);
   fflush (NULL);
   Exit_Status = EXIT_FAILURE;
}

FILE *open_infile (char *filename) {
   FILE *file = fopen (filename, "r");
   if (file == NULL) print_error (filename, strerror (errno));
   DEBUGF ('m', "filename = \"%s\", file = 0x%p\n", filename, file);
   return file;
}

void spellcheck (char *filename, hashset_ref hashset) {
   yylineno = 1;
   DEBUGF ('m', "filename = \"%s\", hashset = 0x%p\n",
                filename, hashset);
   for (;;) {
      int token = yylex ();
      if (token == 0) break;
      DEBUGF ('m', "line %d, yytext = \"%s\"\n", yylineno, yytext);
      STUBPRINTF ("%s: %d: %s\n", filename, yylineno, yytext);
   }
}

void load_dictionary (char *dictionary_name, hashset_ref hashset) {
   if (dictionary_name == NULL) return;
   DEBUGF ('m', "dictionary_name = \"%s\", hashset = %p\n",
           dictionary_name, hashset);
   STUBPRINTF ("Open dictionary, load it, close it\n");
}

int main (int argc, char **argv) {
   Exec_Name = basename (argv[0]);
   char *default_dictionary = DEFAULT_DICTNAME;
   char *user_dictionary = NULL;
   hashset_ref hashset = new_hashset ();
   yy_flex_debug = false;

   // Scan the arguments and set flags.
   opterr = false;
   for (;;) {
      int option = getopt (argc, argv, "nxyd:@:");
      if (option == EOF) break;
      switch (option) {
         char optopt_string[16]; // used in default:
         case 'd': user_dictionary = optarg;
                   break;
         case 'n': default_dictionary = NULL;
                   break;
         case 'x': STUBPRINTF ("-x\n");
                   break;
         case 'y': yy_flex_debug = true;
                   break;
         case '@': set_debugflags (optarg);
                   if (strpbrk (optarg, "@y")) yy_flex_debug = true;
                   break;
         default : sprintf (optopt_string, "-%c", optopt);
                   print_error (optopt_string, "invalid option");
                   break;
      }
   }

   // Load the dictionaries into the hash table.
   load_dictionary (default_dictionary, hashset);
   load_dictionary (user_dictionary, hashset);

   // Read and do spell checking on each of the files.
   if (optind >= argc) {
      yyin = stdin;
      spellcheck (STDIN_NAME, hashset);
   }else {
      int fileix = optind;
      for (; fileix < argc; ++fileix) {
         DEBUGF ('m', "argv[%d] = \"%s\"\n", fileix, argv[fileix]);
         char *filename = argv[fileix];
         if (strcmp (filename, STDIN_NAME) == 0) {
            yyin = stdin;
            spellcheck (STDIN_NAME, hashset);
         }else {
            yyin = open_infile (filename);
            if (yyin == NULL) continue;
            spellcheck (filename, hashset);
            fclose (yyin);
         }
      }
   }

   yycleanup ();
   return Exit_Status;
}

```

```
%{
// $Id: scanner.l,v 1.1 2012-11-16 18:05:22-08 - - $

#include <stdlib.h>

#include "yyextern.h"

%}

%option 8bit
%option debug
%option ecs
%option interactive
%option nodefault
%option noyywrap
%option yylineno

NUMBER  ([[:digit:]]+([-:.][[:digit:]]+)*)
WORD    ([[:alnum:]]+([-&'.][[:alnum:]]+)*)
OTHER   (.|\n)

%%

{NUMBER}         { }
{WORD}           { return 1;  }
{OTHER}          { }

%%

void yycleanup (void) {
   yy_delete_buffer (YY_CURRENT_BUFFER);
}

```

```
# $Id: Makefile,v 1.2 2012-11-20 18:25:15-08 - - $

MKFILE    = Makefile
DEPSFILE  = ${MKFILE}.deps
NOINCLUDE = ci clean spotless
NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
GMAKE     = gmake --no-print-directory

GCC       = gcc -g -O0 -Wall -Wextra -std=gnu99
MKDEPS    = gcc -MM

CSOURCE   = debugf.c hashset.c strhash.c spellchk.c
CHEADER   = debugf.h hashset.h strhash.h yyextern.h
OBJECTS   = ${CSOURCE:.c=.o} scanner.o
EXECBIN   = spellchk
SUBMITS   = ${CHEADER} ${CSOURCE} scanner.l ${MKFILE}
SOURCES   = ${SUBMITS}
LISTING   = Listing.code.ps
PROJECT   = cmps012b-wm.f11 asg4

all : ${EXECBIN}

${EXECBIN} : ${OBJECTS}
        ${GCC} -o $@ ${OBJECTS}

scanner.o : scanner.l
        flex -oscanner.c scanner.l
        gcc -g -O0 -std=gnu99 -c scanner.c

%.o : %.c
        cid + $<
        ${GCC} -c $<

ci : ${SOURCES}
        cid + ${SOURCES}
        checksource ${SUBMITS}

lis : ${SOURCES} ${DEPSFILE}
        mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}

clean :
        - rm ${OBJECTS} ${DEPSFILE} core scanner.c ${EXECBIN}.errs

spotless : clean
        - rm ${EXECBIN}

submit : ${SUBMITS}
        submit ${PROJECT} ${SUBMITS}

deps : ${CSOURCE} ${CHEADER}
        @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
        ${MKDEPS} ${CSOURCE} >>${DEPSFILE}

${DEPSFILE} :
        @ touch ${DEPSFILE}
        ${GMAKE} deps

again :
        ${GMAKE} spotless deps ci all lis

ifeq "${NEEDINCL}" ""
include ${DEPSFILE}
endif

```

```
# Makefile.deps created Tue Nov 20 18:41:08 PST 2012
debugf.o: debugf.c debugf.h
hashset.o: hashset.c debugf.h hashset.h strhash.h
strhash.o: strhash.c strhash.h
spellchk.o: spellchk.c debugf.h hashset.h yyextern.h
```