```
 1: // $Id: debug.h,v 1.3 2014-05-15 21:07:47-07 - - $
 2:
 3: #ifndef __DEBUG_H__
 4: #define __DEBUG_H__
 5:
 6: #include <stdbool.h>
 7:
 8: //
 9: // DESCRIPTION
10: //    Debugging library containing miscellaneous useful things.
11: //
12:
13: //
14: // Program name and exit status.
15: //
16: extern char *program_name;
17: extern int exit_status;
18:
19: //
20: // Support for STUB statements.
21: //
22: #define STUBPRINTF(...) \
23:         __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
24: void __stubprintf (const char *file, int line, const char *func,
25:                    const char *format, ...);
26:
27: //
28: // Sets a string of debug flags to be used by DEBUGF and DEBUGS.
29: // If a particular debug flag has been set, messages are printed.
30: // The flag "@" turns on all flags.
31: //
32: void set_debug_flags (char *flags);
33:
34: //
35: // Check if a debug flag is set.
36: //
37: bool get_debug_flag (char flag);
38:
```

```
39:
40: //
41: // DEBUGF takes printf-like arguments.
42: // DEBUGS takes any fprintf(stderr...) statement as an argument.
43: //
44: #ifdef NDEBUG
45:
46: #define DEBUGF(FLAG,...) ;
47: #define DEBUGS(FLAG,...) ;
48:
49: #else
50:
51: #define DEBUGF(FLAG,...) \
52:         if (get_debug_flag (FLAG)) { \
53:            __show_debug (FLAG, __FILE__, __LINE__, __func__); \
54:            fprintf (stderr, __VA_ARGS__); \
55:            fflush (NULL); \
56:         }
57: #define DEBUGS(FLAG,STMT) \
58:         if (get_debug_flag (FLAG)) { \
59:            __show_debug (FLAG, __FILE__, __LINE__, __func__); \
60:            STMT; \
61:            fflush (NULL); \
62:         }
63: void __show_debug (char flag, char *file, int line, const char *func);
64:
65: #endif
66:
67: #endif
68:
```

```
 1: // $Id: hashset.h,v 1.3 2014-03-05 19:24:07-08 - - $
 2:
 3: #ifndef __HASHSET_H__
 4: #define __HASHSET_H__
 5:
 6: #include <stdbool.h>
 7:
 8: typedef struct hashset hashset;
 9:
10: //
11: // Create a new hashset with a default number of elements.
12: //
13: hashset *new_hashset (void);
14:
15: //
16: // Frees the hashset, and the words it points at.
17: //
18: void free_hashset (hashset*);
19:
20: //
21: // Inserts a new string into the hashset.
22: //
23: void put_hashset (hashset*, const char*);
24:
25: //
26: // Looks up the string in the hashset and returns true if found,
27: // false if not found.
28: //
29: bool has_hashset (hashset*, const char*);
30:
31: #endif
32:
```

```
 1: // $Id: strhash.h,v 1.3 2014-03-05 19:24:07-08 - - $
 2:
 3: //
 4: // NAME
 5: //    strhash - return an unsigned 32-bit hash code for a string
 6: //
 7: // SYNOPSIS
 8: //    size_t strhash (const char *string);
 9: //
10:
11: #ifndef __STRHASH_H__
12: #define __STRHASH_H__
13:
14: #include <inttypes.h>
15:
16: size_t strhash (const char *string);
17:
18: #endif
19:
```

```
 1: // $Id: yyextern.h,v 1.2 2013-05-21 19:58:24-07 - - $
 2:
 3: #ifndef __YYEXTERN_H__
 4: #define __YYEXTERN_H__
 5:
 6: //
 7: // DESCRIPTION
 8: //    Definitions of external names used by flex-generated code.
 9: //
10:
11: #include <stdio.h>
12:
13: extern FILE *yyin;         // File currently being read
14:
15: extern char *yytext;       // Pointer to the string that was found
16:
17: extern int yy_flex_debug; // yylex's verbose tracing flag
18:
19: extern int yylex (void);  // Read next word from opened file yyin
20:
21: extern int yylineno;       // Line number within the current file
22:
23: extern int yylex_destroy (void);
24:                            // Cleans up flex's buffers when done
25:                            // Avoids valgrind memory leak report.
26:
27: #endif
28:
```

```
 1: // $Id: debug.c,v 1.1 2014-05-15 20:01:08-07 - - $
 2:
 3: #include <assert.h>
 4: #include <limits.h>
 5: #include <stdarg.h>
 6: #include <stdio.h>
 7: #include <stdlib.h>
 8: #include <string.h>
 9:
10: #include "debug.h"
11:
12: static char debug_flags[UCHAR_MAX + 1];
13: char *program_name = NULL;
14: int exit_status = EXIT_SUCCESS;
15:
16: void __stubprintf (const char *filename, int line, const char *func,
17:                    const char *format, ...) {
18:    va_list args;
19:    fflush (NULL);
20:    fprintf (stdout, "%s: STUB (%s:%d) %s:\n",
21:             program_name, filename, line, func);
22:    va_start (args, format);
23:    vfprintf (stdout, format, args);
24:    va_end (args);
25:    fflush (NULL);
26: }
27:
28: void set_debug_flags (char *flags) {
29:    if (strchr (flags, '@') != NULL) {
30:       memset (debug_flags, true, sizeof debug_flags);
31:    }else {
32:       for (char *flag = flags; *flag != '\0'; ++flag) {
33:          debug_flags[(unsigned char) *flag] = true;
34:       }
35:    }
36: }
37:
38: bool get_debug_flag (char flag) {
39:    return debug_flags[(unsigned char) flag];
40: }
41:
42: void __show_debug (char flag, char *file, int line, const char *func) {
43:    fflush (NULL);
44:    assert (program_name != NULL);
45:    fprintf (stderr, "%s: DEBUGF(%c): %s[%d]: %s()\n",
46:             program_name, flag, file, line, func);
47: }
48:
```

```
 1: // $Id: hashset.c,v 1.9 2014-05-15 20:01:08-07 - - $
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5: #include <stdlib.h>
 6: #include <string.h>
 7:
 8: #include "debug.h"
 9: #include "hashset.h"
10: #include "strhash.h"
11:
12: #define HASH_NEW_SIZE 15
13:
14: typedef struct hashnode hashnode;
15: struct hashnode {
16:    char *word;
17:    hashnode *link;
18: };
19:
20: struct hashset {
21:    size_t size;
22:    size_t load;
23:    hashnode **chains;
24: };
25:
26: hashset *new_hashset (void) {
27:    hashset *this = malloc (sizeof (struct hashset));
28:    assert (this != NULL);
29:    this->size = HASH_NEW_SIZE;
30:    this->load = 0;
31:    size_t sizeof_chains = this->size * sizeof (hashnode *);
32:    this->chains = malloc (sizeof_chains);
33:    assert (this->chains != NULL);
34:    memset (this->chains, 0, sizeof_chains);
35:    DEBUGF ('h', "%p -> struct hashset {size = %zd, chains=%p}\n",
36:                 this, this->size, this->chains);
37:    return this;
38: }
39:
40: void free_hashset (hashset *this) {
41:    DEBUGF ('h', "free (%p)\n", this);
42: }
43:
44: void put_hashset (hashset *this, const char *item) {
45:    STUBPRINTF ("hashset=%p, item=%s\n", this, item);
46: }
47:
48: bool has_hashset (hashset *this, const char *item) {
49:    STUBPRINTF ("hashset=%p, item=%s\n", this, item);
50:    return true;
51: }
52:
```

```
 1: // $Id: strhash.c,v 1.6 2014-03-05 19:24:07-08 - - $
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5: #include <sys/types.h>
 6:
 7: #include "strhash.h"
 8:
 9: size_t strhash (const char *string) {
10:    assert (string != NULL);
11:    size_t hash = 0;
12:    for (; *string != '\0'; ++string) {
13:       hash = *string + (hash << 6) + (hash << 16) - hash;
14:    }
15:    return hash;
16: }
17:
```

```
 1: // $Id: spellchk.c,v 1.9 2014-05-15 21:07:47-07 - - $
 2:
 3: #include <errno.h>
 4: #include <libgen.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8: #include <unistd.h>
 9:
10: #include "debug.h"
11: #include "hashset.h"
12: #include "yyextern.h"
13:
14: #define STDIN_NAME       "-"
15: #define DEFAULT_DICTNAME \
16:         "/afs/cats.ucsc.edu/courses/cmps012b-wm/usr/dict/words"
17: #define DEFAULT_DICT_POS 0
18: #define EXTRA_DICT_POS   1
19: #define NUMBER_DICTS     2
20:
21: void print_error (const char *object, const char *message) {
22:    fflush (NULL);
23:    fprintf (stderr, "%s: %s: %s\n", program_name, object, message);
24:    fflush (NULL);
25:    exit_status = EXIT_FAILURE;
26: }
27:
28: FILE *open_infile (const char *filename) {
29:    FILE *file = fopen (filename, "r");
30:    if (file == NULL) print_error (filename, strerror (errno));
31:    DEBUGF ('m', "filename = \"%s\", file = 0x%p\n", filename, file);
32:    return file;
33: }
34:
35: void spellcheck (const char *filename, hashset *hashset) {
36:    yylineno = 1;
37:    DEBUGF ('m', "filename = \"%s\", hashset = 0x%p\n",
38:                 filename, hashset);
39:    for (;;) {
40:       int token = yylex ();
41:       if (token == 0) break;
42:       DEBUGF ('m', "line %d, yytext = \"%s\"\n", yylineno, yytext);
43:       STUBPRINTF ("%s: %d: %s\n", filename, yylineno, yytext);
44:    }
45: }
46:
47: void load_dictionary (const char *dictionary_name, hashset *hashset) {
48:    if (dictionary_name == NULL) return;
49:    DEBUGF ('m', "dictionary_name = \"%s\", hashset = %p\n",
50:            dictionary_name, hashset);
51:    STUBPRINTF ("Open dictionary, load it, close it\n");
52: }
53:
```

```
54:
55: void scan_options (int argc, char** argv,
56:                    char **default_dictionary,
57:                    char **user_dictionary) {
58:    // Scan the arguments and set flags.
59:    opterr = false;
60:    for (;;) {
61:       int option = getopt (argc, argv, "nxyd:@:");
62:       if (option == EOF) break;
63:       switch (option) {
64:          char optopt_string[16]; // used in default:
65:          case 'd': *user_dictionary = optarg;
66:                    break;
67:          case 'n': *default_dictionary = NULL;
68:                    break;
69:          case 'x': STUBPRINTF ("-x\n");
70:                    break;
71:          case 'y': yy_flex_debug = true;
72:                    break;
73:          case '@': set_debug_flags (optarg);
74:                    if (strpbrk (optarg, "@y")) yy_flex_debug = true;
75:                    break;
76:          default : sprintf (optopt_string, "-%c", optopt);
77:                    print_error (optopt_string, "invalid option");
78:                    break;
79:       }
80:    }
81: }
82:
```

```
 83:
 84: int main (int argc, char **argv) {
 85:    program_name = basename (argv[0]);
 86:    char *default_dictionary = DEFAULT_DICTNAME;
 87:    char *user_dictionary = NULL;
 88:    hashset *hashset = new_hashset ();
 89:    yy_flex_debug = false;
 90:    scan_options (argc, argv, &default_dictionary, &user_dictionary);
 91:
 92:    // Load the dictionaries into the hash table.
 93:    load_dictionary (default_dictionary, hashset);
 94:    load_dictionary (user_dictionary, hashset);
 95:
 96:    // Read and do spell checking on each of the files.
 97:    if (optind >= argc) {
 98:       yyin = stdin;
 99:       spellcheck (STDIN_NAME, hashset);
100:    }else {
101:       for (int fileix = optind; fileix < argc; ++fileix) {
102:          DEBUGF ('m', "argv[%d] = \"%s\"\n", fileix, argv[fileix]);
103:          char *filename = argv[fileix];
104:          if (strcmp (filename, STDIN_NAME) == 0) {
105:             yyin = stdin;
106:             spellcheck (STDIN_NAME, hashset);
107:          }else {
108:             yyin = open_infile (filename);
109:             if (yyin == NULL) continue;
110:             spellcheck (filename, hashset);
111:             fclose (yyin);
112:          }
113:       }
114:    }
115:
116:    yylex_destroy ();
117:    return exit_status;
118: }
119:
```

```
 1: %{
 2: // $Id: scanner.l,v 1.3 2013-05-21 19:58:24-07 - - $
 3:
 4: #include <stdlib.h>
 5:
 6: #include "yyextern.h"
 7:
 8: %}
 9:
10: %option 8bit
11: %option debug
12: %option ecs
13: %option interactive
14: %option nodefault
15: %option noyywrap
16: %option yylineno
17:
18: NUMBER  ([[:digit:]]+([-:.][[:digit:]]+)*)
19: WORD    ([[:alnum:]]+([-&'.][[:alnum:]]+)*)
20: OTHER   (.|\n)
21:
22: %%
23:
24: {NUMBER}        { }
25: {WORD}          { return 1;  }
26: {OTHER}         { }
27:
28: %%
29:
```

```
 1: # $Id: Makefile,v 1.9 2015-02-20 18:18:19-08 - - $
 2:
 3: MKFILE    = Makefile
 4: DEPSFILE  = ${MKFILE}.deps
 5: NOINCLUDE = ci clean spotless
 6: NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
 7: GMAKE     = gmake --no-print-directory
 8:
 9: GCC       = gcc -g -O0 -Wall -Wextra -std=gnu11
10: MKDEPS    = gcc -MM
11:
12: CSOURCE   = debug.c hashset.c strhash.c spellchk.c
13: CHEADER   = debug.h hashset.h strhash.h yyextern.h
14: OBJECTS   = ${CSOURCE:.c=.o} scanner.o
15: EXECBIN   = spellchk
16: SUBMITS   = ${CHEADER} ${CSOURCE} scanner.l ${MKFILE}
17: SOURCES   = ${SUBMITS}
18: LISTING   = Listing.ps
19: PROJECT   = cmps012b-wm.w15 asg5
20:
21: all : ${EXECBIN}
22:
23: ${EXECBIN} : ${OBJECTS}
24:         ${GCC} -o $@ ${OBJECTS}
25:
26: scanner.o : scanner.l
27:         flex -oscanner.c scanner.l
28:         gcc -g -O0 -std=gnu11 -c scanner.c
29:
30: %.o : %.c
31:         ${GCC} -c $<
32:
33: ci : ${SOURCES}
34:         cid + ${SOURCES}
35:         checksource ${SUBMITS}
36:
37: lis : ${SOURCES} ${DEPSFILE}
38:         mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}
39:
40: clean :
41:         - rm ${OBJECTS} ${DEPSFILE} core scanner.c ${EXECBIN}.errs
42:
43: spotless : clean
44:         - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf}
45:
46: submit : ${SUBMITS}
47:         submit ${PROJECT} ${SUBMITS}
48:
```

```
49:
50: deps : ${CSOURCE} ${CHEADER}
51:         @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
52:         ${MKDEPS} ${CSOURCE} >>${DEPSFILE}
53:
54: ${DEPSFILE} :
55:         @ touch ${DEPSFILE}
56:         ${GMAKE} deps
57:
58: again :
59:         ${GMAKE} spotless deps ci all lis
60:
61: ifeq "${NEEDINCL}" ""
62: include ${DEPSFILE}
63: endif
64:
```

```
1: # Makefile.deps created Thu Feb 26 18:06:47 PST 2015
2: debug.o: debug.c debug.h
3: hashset.o: hashset.c debug.h hashset.h strhash.h
4: strhash.o: strhash.c strhash.h
5: spellchk.o: spellchk.c debug.h hashset.h yyextern.h
```