```
// $Id: auxlib.h,v 1.1 2010-11-04 19:15:48-07 - - $

#ifndef __AUXLIB_H__
#define __AUXLIB_H__

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

//
// Miscellaneous useful typedefs.
//

typedef enum {FALSE = 0, TRUE = 1} bool;

//
// Error message and exit status utility.
//

void set_execname (char *argv0);
   //
   // Sets the program name for use by auxlib messages.
   // Must called from main before anything else is done,
   // passing in argv[0].
   //

char *get_execname (void);
   //
   // Returns a read-only value previously stored by set_progname.
   //

void dprintf (char *format, ...);
   //
   // Print a message to stderr according to the printf format
   // specified.  Usually called for debug output.
   // Precedes the message by the program name if the format
   // begins with the characters '%:'.
   //

void eprintf (char *format, ...);
   //
   // Print an error message according to the printf format
   // specified.  Precedes the message by the program name if
   // the format begins with the characters "%:".
   //

void syseprintf (char *object);
   //
   // Print a message resulting from a bad system call.  The
   // object is the name of the object causing the problem and
   // the reason is taken from the external variable errno.
   //

int get_exitstatus (void);
   //
   // Returns the exit status.  Default is EXIT_SUCCESS unless
   // set_exitstatus (int) is called.  The last statement in main
   // should be:  ''return get_exitstatus();''.
   //

void set_exitstatus (int);
   //
   // Sets the exit status.  Remebers only the largest value passed in.
```

```
 65:    //
 66:
 67: //
 68: // Redefinition of a few functions to keep lint from whining about
 69: // ''function returns value which is always ignored''.  This is not
 70: // generally recommended, but illustrates a very hackish way of
 71: // keeping lint quiet.  Generally, it is recommended just to ignore
 72: // that particular message.
 73: //
 74:
 75: #define xfclose   (void) fclose
 76: #define xfflush   (void) fflush
 77: #define xfprintf  (void) fprintf
 78: #define xmemset   (void) memset
 79: #define xprintf   (void) printf
 80: #define xsprintf  (void) sprintf
 81: #define xvfprintf (void) vfprintf
 82: #define xvprintf  (void) vprintf
 83:
 84: //
 85: // Support for stub messages.
 86: //
 87: #define STUBPRINTF(...) \
 88:         __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
 89: void __stubprintf (char *file, int line, const char *func,
 90:                    char *format, ...);
 91:
 92: //
 93: // Debugging utility.
 94: //
 95:
 96: void set_debugflags (char *flags);
 97:    //
 98:    // Sets a string of debug flags to be used by DEBUGF statements.
 99:    // Uses the address of the string, and does not copy it, so it
100:    // must not be dangling.  If a particular debug flag has been set,
101:    // messages are printed.  The format is identical to printf format.
102:    // The flag "@" turns on all flags.
103:    //
104:
105: #ifdef NDEBUG
106: #define DEBUGF(FLAG,...) // DEBUG (FLAG, __VA_ARGS__)
107: #else
108: #define DEBUGF(FLAG,...) \
109:         __debugprintf (FLAG, __FILE__, __LINE__, __VA_ARGS__)
110: void __debugprintf (char flag, char *file, int line,
111:                     char *format, ...);
112: #endif
113:
114: #endif
115:
```

```
 1: // $Id: hashset.h,v 1.1 2010-11-04 19:15:48-07 - - $
 2:
 3: #ifndef __HASHSET_H__
 4: #define __HASHSET_H__
 5:
 6: #include "auxlib.h"
 7:
 8: typedef struct hashset *hashset_ref;
 9:
10: //
11: // Create a new hashset with a default number of elements.
12: //
13: hashset_ref new_hashset (void);
14:
15: //
16: // Frees the hashset, and the words it points at.
17: //
18: void free_hashset (hashset_ref);
19:
20: //
21: // Inserts a new string into the hashset.
22: //
23: void put_hashset (hashset_ref, 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_ref, char*);
30:
31: #endif
32:
```

```
 1: // $Id: strhash.h,v 1.1 2010-11-04 19:15:48-07 - - $
 2:
 3: //
 4: // NAME
 5: //    strhash - return an unsigned 32-bit hash code for a string
 6: //
 7: // SYNOPSIS
 8: //    hashcode_t strhash (char *string);
 9: //
10: // DESCRIPTION
11: //    Uses Horner's method to compute the hash code of a string
12: //    as is done by java.lang.String.hashCode:
13: //    .  s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
14: //    Using strength reduction, the multiplication is replaced by
15: //    a shift.  However, instead of returning a signed number,
16: //    this function returns an unsigned number.
17: //
18: // REFERENCE
19: //    http://java.sun.com/j2se/1.4.1/docs/api/java/lang/
20: //    String.html#hashCode()
21: //
22: //
23:
24: #ifndef __STRHASH_H__
25: #define __STRHASH_H__
26:
27: #include <inttypes.h>
28:
29: #include "auxlib.h"
30:
31: typedef uint32_t hashcode_t;
32:
33: hashcode_t strhash (char *string);
34:
35: #endif
36:
```

```
 1: // $Id: yyextern.h,v 1.1 2010-11-04 19:15:48-07 - - $
 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 void yycleanup (void); // Cleans up flex's buffers when done
24:
25: #endif
26:
```

```
 1: // $Id: auxlib.c,v 1.1 2010-11-04 19:15:48-07 - - $
 2:
 3: #include <assert.h>
 4: #include <errno.h>
 5: #include <libgen.h>
 6: #include <limits.h>
 7: #include <stdarg.h>
 8: #include <stdio.h>
 9: #include <stdlib.h>
10: #include <string.h>
11:
12: #include "auxlib.h"
13:
14: static int exitstatus = EXIT_SUCCESS;
15: static char *execname = NULL;
16: static char *debugflags = "";
17: static bool alldebugflags = FALSE;
18:
19: void set_execname (char *argv0) {
20:    execname = basename (argv0);
21: }
22:
23: char *get_execname (void) {
24:    assert (execname != NULL);
25:    return execname;
26: }
27:
28: static char *init_dprintf (char *format) {
29:    assert (format != NULL);
30:    xfflush (NULL);
31:    if (strstr (format, "%:") == format) {
32:       xfprintf (stderr, "%s: ", get_execname ());
33:       format += 2;
34:    };
35:    return format;
36: }
37:
38: void dprintf (char *format, ...) {
39:    va_list args;
40:    format = init_dprintf (format);
41:    va_start (args, format);
42:    xvfprintf (stderr, format, args);
43:    va_end (args);
44:    xfflush (NULL);
45: }
46:
47: void eprintf (char *format, ...) {
48:    va_list args;
49:    assert (execname != NULL);
50:    assert (format != NULL);
51:    format = init_dprintf (format);
52:    va_start (args, format);
53:    xvfprintf (stderr, format, args);
54:    va_end (args);
55:    xfflush (NULL);
56:    exitstatus = EXIT_FAILURE;
57: }
58:
59: void syseprintf (char *object) {
60:    eprintf ("%:%s: %s\n", object, strerror (errno));
61: }
62:
63: int get_exitstatus (void) {
64:    return exitstatus;
```

```
 65: }
 66:
 67: void set_exitstatus (int newexitstatus) {
 68:    newexitstatus &= 0xFF;
 69:    if (exitstatus < newexitstatus) exitstatus = newexitstatus;
 70:    DEBUGF ('a', "exitstatus = %d\n", exitstatus);
 71: }
 72:
 73: void __stubprintf (char *file, int line, const char *func,
 74:                    char *format, ...) {
 75:    va_list args;
 76:    xfflush (NULL);
 77:    xprintf ("%s: %s[%d] %s: ", execname, file, line, func);
 78:    va_start (args, format);
 79:    xvprintf (format, args);
 80:    va_end (args);
 81:    xfflush (NULL);
 82: }
 83:
 84: void set_debugflags (char *flags) {
 85:    debugflags = flags;
 86:    if (strchr (debugflags, '@') != NULL) alldebugflags = TRUE;
 87:    DEBUGF ('a', "Debugflags = \"%s\"\n", debugflags);
 88: }
 89:
 90: void __debugprintf (char flag, char *file, int line,
 91:                     char *format, ...) {
 92:    va_list args;
 93:    if (alldebugflags || strchr (debugflags, flag) != NULL) {
 94:       xfflush (NULL);
 95:       va_start (args, format);
 96:       xfprintf (stderr, "DEBUGF(%c): %s[%d]:\n",
 97:                 flag, file, line);
 98:       xvfprintf (stderr, format, args);
 99:       va_end (args);
100:       xfflush (NULL);
101:    }
102: }
103:
```

```
 1: // $Id: hashset.c,v 1.4 2010-11-08 19:56:46-08 - - $
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5: #include <stdlib.h>
 6: #include <string.h>
 7:
 8: #include "hashset.h"
 9: #include "strhash.h"
10:
11: #define HASH_NEW_SIZE 15
12:
13: struct hashset {
14:    size_t length;
15:    int load;
16:    char **array;
17: };
18:
19: hashset_ref new_hashset (void) {
20:    hashset_ref new = malloc (sizeof (struct hashset));
21:    assert (new != NULL);
22:    new->length = HASH_NEW_SIZE;
23:    new->load = 0;
24:    new->array = malloc (new->length * sizeof (char*));
25:    for (size_t index = 0; index < new->length; ++index) {
26:       new->array[index] = NULL;
27:    }
28:    assert (new->array != NULL);
29:    DEBUGF ('h', "%p -> struct hashset {length = %d, array=%p}\n",
30:                 new, new->length, new->array);
31:    return new;
32: }
33:
34: void free_hashset (hashset_ref hashset) {
35:    DEBUGF ('h', "free (%p), free (%p)\n", hashset->array, hashset);
36:    xmemset (hashset->array, 0, hashset->length * sizeof (char*));
37:    free (hashset->array);
38:    xmemset (hashset, 0, sizeof (struct hashset));
39:    free (hashset);
40: }
41:
42: void put_hashset (hashset_ref hashset, char *item) {
43:    STUBPRINTF ("hashset=%p, item=%s\n", hashset, item);
44: }
45:
46: bool has_hashset (hashset_ref hashset, char *item) {
47:    STUBPRINTF ("hashset=%p, item=%s\n", hashset, item);
48:    return TRUE;
49: }
50:
```

```
 1: // $Id: strhash.c,v 1.1 2010-11-04 19:15:48-07 - - $
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5: #include <sys/types.h>
 6:
 7: #include "strhash.h"
 8:
 9: hashcode_t strhash (char *string) {
10:    assert (string != NULL);
11:    hashcode_t hashcode = 0;
12:    while (*string) hashcode = hashcode * 31 + (unsigned char) *string++;
13:    return hashcode;
14: }
15:
```

```
 1: // $Id: spellchk.c,v 1.1 2010-11-04 19:15:48-07 - - $
 2:
 3: #include <stdio.h>
 4: #include <stdlib.h>
 5: #include <string.h>
 6: #include <unistd.h>
 7:
 8: #include "auxlib.h"
 9: #include "hashset.h"
10: #include "yyextern.h"
11:
12: #define STDIN_NAME       "-"
13: #define DEFAULT_DICTNAME "/usr/share/dict/words"
14: #define DEFAULT_DICT_POS 0
15: #define EXTRA_DICT_POS   1
16: #define NUMBER_DICTS     2
17:
18: FILE *open_infile (char *filename) {
19:    FILE *file = fopen (filename, "r");
20:    if (file == NULL) {
21:       syseprintf (filename);
22:       set_exitstatus (EXIT_FAILURE);
23:    };
24:    DEBUGF (’m’, "filename = \"%s\", file = 0x%p\n", filename, file);
25:    return file;
26: }
27:
28: void spellcheck (char *filename, hashset_ref hashset) {
29:    yylineno = 1;
30:    DEBUGF (’m’, "filename = \"%s\", hashset = 0x%p\n",
31:                 filename, hashset);
32:    for (;;) {
33:       int token = yylex ();
34:       if (token == 0) break;
35:       DEBUGF (’m’, "line %d, yytext = \"%s\"\n", yylineno, yytext);
36:       STUBPRINTF ("%s: %d: %s\n", filename, yylineno, yytext);
37:    };
38: }
39:
40: void load_dictionary (char *dictionary_name, hashset_ref hashset) {
41:    if (dictionary_name == NULL) return;
42:    DEBUGF (’m’, "dictionary_name = \"%s\", hashset = %p\n",
43:            dictionary_name, hashset);
44:    STUBPRINTF ("Open dictionary, load it, close it\n");
45: }
46:
47: int main (int argc, char **argv) {
48:    char *default_dictionary = DEFAULT_DICTNAME;
49:    char *user_dictionary = NULL;
50:    hashset_ref hashset = new_hashset ();
51:    yy_flex_debug = FALSE;
52:    set_execname (argv[0]);
53:
54:    // Scan the arguments and set flags.
55:    opterr = FALSE;
56:    for (;;) {
57:       int option = getopt (argc, argv, "nxyd:@:");
58:       if (option == EOF) break;
59:       switch (option) {
60:          case ’d’: user_dictionary = optarg;
61:                    break;
62:          case ’n’: default_dictionary = NULL;
63:                    break;
64:          case ’x’: STUBPRINTF ("-x\n");
```

```
 65:                      break;
 66:          case 'y': yy_flex_debug = TRUE;
 67:                      break;
 68:          case '@': set_debugflags (optarg);
 69:                      if (strpbrk (optarg, "@y")) yy_flex_debug = TRUE;
 70:                      break;
 71:          default : eprintf ("%: -%c: invalid option\n", optopt);
 72:                      set_exitstatus (EXIT_FAILURE);
 73:       };
 74:    };
 75:
 76:    // Load the dictionaries into the hash table.
 77:    load_dictionary (default_dictionary, hashset);
 78:    load_dictionary (user_dictionary, hashset);
 79:
 80:    // Read and do spell checking on each of the files.
 81:    if (optind >= argc) {
 82:       yyin = stdin;
 83:       spellcheck (STDIN_NAME, hashset);
 84:    }else {
 85:       int fileix = optind;
 86:       for (; fileix < argc; ++fileix) {
 87:          DEBUGF ('m', "argv[%d] = \"%s\"\n", fileix, argv[fileix]);
 88:          char *filename = argv[fileix];
 89:          if (strcmp (filename, STDIN_NAME) == 0) {
 90:             yyin = stdin;
 91:             spellcheck (STDIN_NAME, hashset);
 92:          }else {
 93:             yyin = open_infile (filename);
 94:             if (yyin == NULL) continue;
 95:             spellcheck (filename, hashset);
 96:             xfclose (yyin);
 97:          }
 98:       };
 99:    }
100:
101:    yycleanup ();
102:    return get_exitstatus ();
103: }
104:
```

```
 1: %{
 2: // $Id: scanner.l,v 1.1 2010-11-04 19:15:48-07 - - $
 3:
 4: #include <stdlib.h>
 5:
 6: #include "auxlib.h"
 7: #include "yyextern.h"
 8:
 9: %}
10:
11: %option 8bit
12: %option debug
13: %option ecs
14: %option interactive
15: %option nodefault
16: %option noyywrap
17: %option yylineno
18:
19: NUMBER  ([[:digit:]]+([-:.][[:digit:]]+)*)
20: WORD    ([[:alnum:]]+([-&'.][[:alnum:]]+)*)
21: OTHER   (.|\n)
22:
23: %%
24:
25: {NUMBER}        { }
26: {WORD}          { return 1;  }
27: {OTHER}         { }
28:
29: %%
30:
31: void yycleanup (void) {
32:    yy_delete_buffer (YY_CURRENT_BUFFER);
33: }
34:
```

```
# $Id: Makefile,v 1.3 2010-11-04 19:36:19-07 - - $
MKFILE    = Makefile
DEPSFILE  = ${MKFILE}.deps
NOINCLUDE = ci clean spotless
NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
GMAKE     = gmake --no-print-directory

CCOPT     = -Xa -v -g -xO0
LINTOPT   = -Xa -fd -m -u -x -errchk=%all

CSOURCE   = auxlib.c hashset.c strhash.c spellchk.c
CHEADER   = auxlib.h hashset.h strhash.h yyextern.h
OBJECTS   = ${CSOURCE:.c=.o} scanner.o
EXECBIN   = spellchk
SUBMITS   = ${CHEADER} ${CSOURCE} scanner.l ${MKFILE}
SOURCES   = ${SUBMITS}
LISTING   = ../asg4c-spellchk.code.ps
PROJECT   = cmps012b-wm.f10 asg4

all : ${EXECBIN}

${EXECBIN} : ${OBJECTS}
        cc ${CCOPT} -o $@ ${OBJECTS}

scanner.o : scanner.l
        flex -oscanner.c scanner.l
        cc -g -xO0 -c scanner.c

%.o : %.c
        cc ${CCOPT} -c $<

lint : ${CSOURCE}
        lint ${LINTOPT} ${CSOURCE}
        checksource ${SUBMITS}

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}
        testsubmit ${PROJECT} ${SUBMITS}

deps : ${CSOURCE} ${CHEADER}
        @ echo "# ${DEPSFILE} created ‘date‘" >${DEPSFILE}
        cc -xM1 ${CSOURCE} \
        | grep -v /usr/local/sunstudio/sunstudio12.1/prod/include/ \
        | sort | uniq >>${DEPSFILE}

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

again :
        ${GMAKE} spotless deps ci lint all lis
```

```
65:
66: ifeq "${NEEDINCL}" ""
67: include ${DEPSFILE}
68: endif
69:
```

```
 1: # Makefile.deps created Mon Nov  8 19:56:46 PST 2010
 2: auxlib.o: auxlib.c
 3: auxlib.o: auxlib.h
 4: hashset.o: auxlib.h
 5: hashset.o: hashset.c
 6: hashset.o: hashset.h
 7: hashset.o: strhash.h
 8: spellchk.o: auxlib.h
 9: spellchk.o: hashset.h
10: spellchk.o: spellchk.c
11: spellchk.o: yyextern.h
12: strhash.o: auxlib.h
13: strhash.o: strhash.c
14: strhash.o: strhash.h
```