```
 1: // $Id: list.h,v 1.5 2012-11-14 21:35:53-08 - - $
 2:
 3: #ifndef __LIST_H__
 4: #define __LIST_H__
 5:
 6: #include <stdbool.h>
 7:
 8: //
 9: // NAME
10: //    list - maintain a doubly linked list of elements.
11: //
12: // DESCRIPTION
13: //    A doubly linked list of elements is maintained, with the
14: //    ability to insert, delete, and view elements of the list
15: //    and to move up and down the list one step at a time.
16: //
17:
18: typedef enum {MOVE_HEAD, MOVE_PREV, MOVE_NEXT, MOVE_LAST} list_move;
19:    //
20:    // Type:  enumeration code for moving up and down the list.
21:    //        Used as second argument to 'setmove_list':
22:    //
23:    // MOVE_HEAD - set the cursor position to the null node at the
24:    //             beginning of the list.
25:    //
26:    // MOVE_LAST - set the cursor position to the last node in the
27:    //             list.
28:    //
29:    // MOVE_PREV - set the cursor position to the node immediately
30:    //             before it in the list.
31:    //
32:    // MOVE_NEXT - set the cursor position to the node immediately
33:    //             after the current position.
34:    //
35:
36: typedef struct list *list_ref;
37:    //
38:    // Type:  the handle returned by the constructor and passed to
39:    //        the other functions.
40:    //
41:
42: void debugdump_list (list_ref list);
43:    //
44:    // Accessor:       Prints out a debug dump of the list to stderr.
45:    // Precondition:   valid list.
46:    // Postcondition:  none.
47:    //
48:
```

```
49:
50: list_ref new_list (void);
51:    //
52:    // Constructor:    return a new valid empty list.
53:    // Precondition:   none;
54:    // Postcondition:  returns a properly constituted empty list.
55:    //
56:
57: void free_list (list_ref);
58:    //
59:    // Destructor:     frees up the list and its internal headers.
60:    // Preconditions:  list must be valid and empty.
61:    // Postcondition:  memory is freed and its argument is dangling.
62:    // Asserts:        that the precondition is met.
63:    //
64:
65: bool setmove_list (list_ref, list_move);
66:    //
67:    // Mutator:        moves the current position to another position
68:    //                 in the list (see 'typedef list_move' above).
69:    // Precondition:   list must be valid.
70:    //                 MOVE_PREV may not be used in the head position.
71:    //                 MOVE_NEXT may not be used in the last position.
72:    // Postcondition:  returns TRUE if successful and FALSE if failed.
73:    //
74:
75: char *viewcurr_list (list_ref);
76:    //
77:    // Accessor:       returns the data item in the current node in the
78:    //                 list.  Does not release space, this pointer's
79:    //                 target may not be changed or freed.
80:    // Precondition:   valid list and current not in the head position.
81:    // Postcondition:  returns NULL if called from the head positin.
82:    //
83:
```

```
 84:
 85: void insert_list (list_ref, char *);
 86:    //
 87:    // Mutator:        inserts new char* into the list immediately
 88:    //                 after the current position.
 89:    // Precondition:   valid list.
 90:    // Postcondition:  char* passed in is now property of list and
 91:    //                 not the client.
 92:    //
 93:
 94: void delete_list (list_ref);
 95:    //
 96:    // Mutator:        deletes the current line from the list.
 97:    // Precondition:   valid list and not empty.
 98:    // Postcondition:  same list with one line removed.
 99:    //
100:
101: bool empty_list (list_ref);
102:    //
103:    // Accessor:       checks to see if the list is empty.
104:    // Precondition:   list is valid.
105:    // Postcondition:  returns TRUE iff the only node is the head node.
106:    //
107:
108: bool is_list (list_ref);
109:    //
110:    // Accessor:       checks to see if its argument is a list.
111:    // Precondition:   none.
112:    // Postcondition:  returns TRUE if is not null and is a list.
113:    //
114:
115: #endif
116:
```

```
 1: // $Id: debugf.h,v 1.1 2012-11-14 21:32:49-08 - - $
 2:
 3: #ifndef __DEBUGF_H__
 4: #define __DEBUGF_H__
 5:
 6: //
 7: // DESCRIPTION
 8: //    Debugging library containing miscellaneous useful things.
 9: //
10:
11: //
12: // Keep track of Exec_Name and Exit_Status.
13: //
14: extern char *Exec_Name;
15: extern int Exit_Status;
16:
17: //
18: // Tell debugf what program is running.
19: //
20: void set_Exec_Name (char *name);
21:
22: //
23: // Support for stub messages.
24: //
25: #define STUBPRINTF(...) \
26:         __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
27: void __stubprintf (char *file, int line, const char *func,
28:                    char *format, ...);
29:
30: //
31: // Debugging utility.
32: //
33:
34: void set_debugflags (char *flags);
35:    //
36:    // Sets a string of debug flags to be used by DEBUGF statements.
37:    // Uses the address of the string, and does not copy it, so it
38:    // must not be dangling.  If a particular debug flag has been set,
39:    // messages are printed.  The format is identical to printf format.
40:    // The flag "@" turns on all flags.
41:    //
42:
43: #ifdef NDEBUG
44: #define DEBUGF(FLAG,...) // DEBUG (FLAG, __VA_ARGS__)
45: #else
46: #define DEBUGF(FLAG,...) \
47:         __debugprintf (FLAG, __FILE__, __LINE__, __VA_ARGS__)
48: void __debugprintf (char flag, char *file, int line,
49:                     char *format, ...);
50: #endif
51:
52: #endif
53:
```

```
 1: // $Id: edfile.c,v 1.14 2012-11-14 21:35:53-08 - - $
 2:
 3: #include <assert.h>
 4: #include <libgen.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8: #include <unistd.h>
 9:
10: #include "debugf.h"
11: #include "list.h"
12:
13: bool want_echo = false;
14:
15: void badline (int stdincount, char *stdinline) {
16:    fflush (NULL);
17:    fprintf (stderr, "Bad input line %d: %s\n", stdincount, stdinline);
18:    fflush (NULL);
19: }
20:
21: void editfile (list_ref list) {
22:    char stdinline[1024];
23:    int stdincount = 0;
24:    for(;; ++stdincount) {
25:       printf ("%s: ", Exec_Name);
26:       char *linepos = fgets (stdinline, sizeof stdinline, stdin);
27:       if (linepos == NULL) break;
28:       if (want_echo) printf ("%s", stdinline);
29:       linepos = strchr (stdinline, '\n');
30:       if (linepos == NULL || stdinline[0] == '\0') {
31:          badline (stdincount, stdinline);
32:       }else {
33:          *linepos = '\0';
34:          switch (stdinline[0]) {
35:             case '$': STUBPRINTF ("stdin[%d]: $\n", stdincount); break;
36:             case '*': STUBPRINTF ("stdin[%d]: *\n", stdincount); break;
37:             case '.': STUBPRINTF ("stdin[%d]: .\n", stdincount); break;
38:             case '0': STUBPRINTF ("stdin[%d]: 0\n", stdincount); break;
39:             case '<': STUBPRINTF ("stdin[%d]: <\n", stdincount); break;
40:             case '>': STUBPRINTF ("stdin[%d]: >\n", stdincount); break;
41:             case '@': debugdump_list (list); break;
42:             case 'a': STUBPRINTF ("stdin[%d]: a\n", stdincount); break;
43:             case 'd': STUBPRINTF ("stdin[%d]: d\n", stdincount); break;
44:             case 'i': STUBPRINTF ("stdin[%d]: i\n", stdincount); break;
45:             case 'r': STUBPRINTF ("stdin[%d]: r\n", stdincount); break;
46:             case 'w': STUBPRINTF ("stdin[%d]: w\n", stdincount); break;
47:             default : badline (stdincount, stdinline);
48:          }
49:       }
50:    }
51: }
52:
```

```
53:
54: void usage_exit() {
55:    fflush (NULL);
56:    fprintf (stderr, "Usage: %s filename\n", Exec_Name);
57:    fflush (NULL);
58:    exit (EXIT_FAILURE);
59: }
60:
61: int main (int argc, char **argv) {
62:    Exec_Name = basename (argv[0]);
63:    if (argc != 2) usage_exit();
64:    want_echo = ! (isatty (fileno (stdin)) && isatty (fileno (stdout)));
65:    list_ref list = new_list();
66:    editfile (list);
67:    free_list (list); list = NULL;
68:    return Exit_Status;
69: }
```

```
 1: // $Id: list.c,v 1.13 2012-11-14 21:36:52-08 - - $
 2:
 3: #include <assert.h>
 4: #include <stdbool.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7:
 8: #include "debugf.h"
 9: #include "list.h"
10:
11: static char *list_tag = "struct list";
12: static char *listnode_tag = "struct listnode";
13:
14: typedef struct listnode *listnode_ref;
15:
16: struct list {
17:    //
18:    // INVARIANT:  Both head and last are NULL or neither are NULL.
19:    // If neither are null, then following listnode next links from
20:    // head will get to last, and prev links from last gets to head.
21:    //
22:    char *tag;
23:    listnode_ref head;
24:    listnode_ref curr;
25:    listnode_ref last;
26: };
27:
28: struct listnode {
29:    char *tag;
30:    char *line;
31:    listnode_ref prev;
32:    listnode_ref next;
33: };
34:
35: void debugdump_list (list_ref list) {
36:    listnode_ref itor = NULL;
37:    assert (is_list (list));
38:    fflush (NULL);
39:    fprintf (stderr,
40:             "\n[%p]->struct list {tag=[%p]->[%s];"
41:             " head=[%p]; curr=[%p]; last=[%p]; }\n",
42:             list, list->tag, list->tag,
43:             list->head, list->curr, list->last);
44:    for (itor = list->head; itor != NULL; itor = itor->next) {
45:       fprintf (stderr,
46:                "[%p]->struct listnode {tag=[%p]->[%s];"
47:                " line=[%p]=[%s]; prev=[%p]; next=[%p]; }\n",
48:                itor, itor->tag, itor->tag, itor->line, itor->line,
49:                itor->prev, itor->next);
50:    }
51:    fflush (NULL);
52: }
53:
```

```
54:
55: list_ref new_list (void) {
56:    //
57:    // Creates a new struct list.
58:    //
59:    list_ref list = malloc (sizeof (struct list));
60:    assert (list != NULL);
61:    list->tag = list_tag;
62:    list->head = NULL;
63:    list->curr = NULL;
64:    list->last = NULL;
65:    return list;
66: }
67:
68: void free_list (list_ref list) {
69:    assert (is_list (list));
70:    assert (empty_list (list));
71:    STUBPRINTF ("list=[%p]\n", list);
72: }
73:
74: bool setmove_list (list_ref list, list_move move) {
75:    assert (is_list (list));
76:    switch (move) {
77:       case MOVE_HEAD:
78:            STUBPRINTF ("MOVE_HEAD: list=[%p]\n", list);
79:            break;
80:       case MOVE_PREV:
81:            STUBPRINTF ("MOVE_PREV: list=[%p]\n", list);
82:            break;
83:       case MOVE_NEXT:
84:            STUBPRINTF ("MOVE_NEXT: list=[%p]\n", list);
85:            break;
86:       case MOVE_LAST:
87:            STUBPRINTF ("MOVE_LAST: list=[%p]\n", list);
88:            break;
89:       default: assert (false);
90:    }
91:    return false;
92: }
93:
94: char *viewcurr_list (list_ref list) {
95:    assert (is_list (list));
96:    STUBPRINTF ("list=[%p]\n", list);
97:    return NULL;
98: }
99:
```

```
100:
101: void insert_list (list_ref list, char *line) {
102:    assert (is_list (list));
103:    STUBPRINTF ("list=[%p], line=[%p]=%s\n", list, line, line);
104: }
105:
106: void delete_list (list_ref list) {
107:    assert (is_list (list));
108:    assert (! empty_list (list));
109:    STUBPRINTF ("list=[%p]\n", list);
110: }
111:
112: bool empty_list (list_ref list) {
113:    assert (is_list (list));
114:    return list->head == NULL;
115: }
116:
117: bool is_list (list_ref list) {
118:    return list != NULL && list->tag == list_tag;
119: }
120:
```

```
 1: // $Id: debugf.c,v 1.6 2012-11-14 19:11:05-08 - - $
 2:
 3: #include <errno.h>
 4: #include <stdarg.h>
 5: #include <stdbool.h>
 6: #include <stdio.h>
 7: #include <stdlib.h>
 8: #include <string.h>
 9: #include <unistd.h>
10:
11: #include "debugf.h"
12:
13: char *Exec_Name = NULL;
14: int Exit_Status = EXIT_SUCCESS;
15:
16: static char *debugflags = "";
17: static bool alldebugflags = false;
18:
19: void __stubprintf (char *filename, int line, const char *func,
20:                    char *format, ...) {
21:    va_list args;
22:    fflush (NULL);
23:    fprintf (stdout, "%s: STUB: %s[%d] %s:\n",
24:             Exec_Name, filename, line, func);
25:    va_start (args, format);
26:    vfprintf (stdout, format, args);
27:    va_end (args);
28:    fflush (NULL);
29: }
30:
31: void set_debugflags (char *flags) {
32:    debugflags = flags;
33:    if (strchr (debugflags, '@') != NULL) alldebugflags = true;
34:    DEBUGF ('a', "Debugflags = \"%s\"\n", debugflags);
35: }
36:
37: void __debugprintf (char flag, char *file, int line,
38:                     char *format, ...) {
39:    va_list args;
40:    if (alldebugflags || strchr (debugflags, flag) != NULL) {
41:       fflush (NULL);
42:       fprintf (stderr, "%s: DEBUGF(%c): %s[%d]:\n",
43:                Exec_Name, flag, file, line);
44:       va_start (args, format);
45:       vfprintf (stderr, format, args);
46:       va_end (args);
47:       fflush (NULL);
48:    }
49: }
50:
```

```
 1: # $Id: Makefile,v 1.7 2012-11-14 21:39:16-08 - - $
 2: MKFILE    = Makefile
 3: DEPSFILE  = ${MKFILE}.deps
 4: NOINCLUDE = ci clean spotless
 5: NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
 6: SUBMAKE   = ${MAKE} --no-print-directory
 7:
 8: GCC       = gcc -g -O0 -Wall -Wextra -std=gnu99
 9: MAKEDEPS  = cc -MM
10:
11: CHEADER   = list.h debugf.h
12: CSOURCE   = edfile.c ${CHEADER:.h=.c}
13: OBJECTS   = ${CSOURCE:.c=.o}
14: EXECBIN   = edfile
15: SOURCES   = ${CHEADER} ${CSOURCE} ${MKFILE}
16: LISTING   = asg4c-edfile.code.ps
17:
18: all : ${EXECBIN}
19:
20: ${EXECBIN} : ${OBJECTS}
21:         ${GCC} -o $@ ${OBJECTS}
22:
23: %.o : %.c
24:         cid + $<
25:         ${GCC} -c $<
26:
27: ci : ${SOURCES}
28:         cid + ${SOURCES}
29:
30: lis : ${SOURCES}
31:         mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}
32:
33: clean :
34:         - rm ${OBJECTS} ${DEPSFILE} core
35:
36: spotless : clean
37:         - rm ${EXECBIN}
38:
39: deps : ${CSOURCE} ${CHEADER}
40:         @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
41:         ${MAKEDEPS} ${CSOURCE} | sort | uniq >>${DEPSFILE}
42:
43: ${DEPSFILE} :
44:         @ touch ${DEPSFILE}
45:         ${SUBMAKE} deps
46:
47: again :
48:         ${MAKE} --no-print-directory spotless deps ci all lis
49:
50: ifeq "${NEEDINCL}" ""
51: include ${DEPSFILE}
52: endif
53:
```

```
1: # Makefile.deps created Wed Nov 14 21:39:16 PST 2012
2: debugf.o: debugf.c debugf.h
3: edfile.o: edfile.c debugf.h list.h
4: list.o: list.c debugf.h list.h
```