

```
1: // $Id: debugf.h,v 1.1 2012-03-05 19:43:01-08 - - $
2:
3: #ifndef __DEBUGF_H__
4: #define __DEBUGF_H__
5:
6: //
7: // DESCRIPTION
8: //     Debugging library containing miscellaneous useful things.
9: //
10:
11: //
12: // Tell debugf what program is running.
13: //
14: void set_execname (char *name);
15:
16: //
17: // Support for stub messages.
18: //
19: #define STUBPRINTF(...) \
20:     __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
21: void __stubprintf (char *file, int line, const char *func,
22:                  char *format, ...);
23:
24: //
25: // Debugging utility.
26: //
27:
28: void set_debugflags (char *flags);
29:     //
30:     // Sets a string of debug flags to be used by DEBUGF statements.
31:     // Uses the address of the string, and does not copy it, so it
32:     // must not be dangling.  If a particular debug flag has been set,
33:     // messages are printed.  The format is identical to printf format.
34:     // The flag "@" turns on all flags.
35:     //
36:
37: #ifdef NDEBUG
38: #define DEBUGF(FLAG,...) // DEBUG (FLAG, __VA_ARGS__)
39: #else
40: #define DEBUGF(FLAG,...) \
41:     __debugprintf (FLAG, __FILE__, __LINE__, __VA_ARGS__)
42: void __debugprintf (char flag, char *file, int line,
43:                  char *format, ...);
44: #endif
45:
46: #endif
47:
```

```
1: // $Id: stack.h,v 1.1 2012-03-05 19:43:01-08 - - $
2:
3: #ifndef __STACK_H__
4: #define __STACK_H__
5:
6: #include <stdbool.h>
7:
8: typedef struct stack *stack_ref;
9: typedef struct bigint *stack_item;
10:
11: //
12: // Create a new empty stack.
13: //
14: stack_ref new_stack (void);
15:
16: //
17: // Free up the stack.
18: // Precondition: stack must be empty.
19: //
20: void free_stack (stack_ref);
21:
22: //
23: // Push a new stack_item onto the top of the stack.
24: //
25: void push_stack (stack_ref, stack_item);
26:
27: //
28: // Pop the top stack_item from the stack and return it.
29: //
30: stack_item pop_stack (stack_ref);
31:
32: //
33: // Peek into the stack and return a selected stack_item, with
34: // item 0 being the element at the top.
35: // and item length_stack - 1 being the element at the bottom.
36: // Precondition: 0 <= index && index < length_stack.
37: //
38: stack_item peek_stack (stack_ref, int index);
39:
40: //
41: // Indicate whether the stack is empty or not.
42: //
43: bool is_empty_stack (stack_ref);
44:
45: //
46: // Return a count of the number of items on the stack.
47: //
48: int length_stack (stack_ref);
49:
50: //
51: // Indicate whether or not a pointer points at a stack.
52: //
53: bool is_stack (stack_ref);
54:
55: #endif
56:
```

```
1: // $Id: bigint.h,v 1.2 2012-03-05 20:19:38-08 - - $
2:
3: #ifndef __BIGINT_H__
4: #define __BIGINT_H__
5:
6: #include <stdbool.h>
7:
8: typedef struct bigint *bigint_ref;
9:
10: typedef bigint_ref (*bigint_binop) (bigint_ref, bigint_ref);
11:
12: bigint_ref new_bigint (size_t length);
13:
14: bigint_ref from_string_bigint (char *string);
15:
16: void free_bigint (bigint_ref);
17:
18: void print_bigint (bigint_ref);
19:
20: bigint_ref add_bigint (bigint_ref left, bigint_ref right);
21:
22: bigint_ref sub_bigint (bigint_ref left, bigint_ref right);
23:
24: bigint_ref mul_bigint (bigint_ref left, bigint_ref right);
25:
26: bool is_bigint (bigint_ref);
27:
28: #endif
29:
```

```
1: // $Id: token.h,v 1.2 2012-03-05 20:19:38-08 - - $
2:
3: #ifndef __TOKEN_H__
4: #define __TOKEN_H__
5:
6: #define NUMBER 256
7:
8: typedef struct token *token_ref;
9:
10: token_ref new_token (FILE*);
11:
12: int scan_token (token_ref);
13:
14: char *peek_token (token_ref);
15:
16: #endif
17:
```

```
1: // $Id: debugf.c,v 1.1 2012-03-05 19:43:01-08 - - $
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: static char *debugflags = "";
14: static bool alldebugflags = false;
15: static char *execname = NULL;
16:
17: void set_execname (char *name) {
18:     execname = name;
19: }
20:
21: void print_execname (FILE *out) {
22:     if (execname != NULL) fprintf (out, "%s: ", execname);
23: }
24:
25: void __stubprintf (char *filename, int line, const char *func,
26:                   char *format, ...) {
27:     va_list args;
28:     fflush (NULL);
29:     print_execname (stdout);
30:     fprintf (stdout, "STUB: %s[%d] %s:\n", filename, line, func);
31:     va_start (args, format);
32:     vfprintf (stdout, format, args);
33:     va_end (args);
34:     fflush (NULL);
35: }
36:
37: void set_debugflags (char *flags) {
38:     debugflags = flags;
39:     if (strchr (debugflags, '@') != NULL) alldebugflags = true;
40:     DEBUGF ('a', "Debugflags = \"%s\"\n", debugflags);
41: }
42:
43: void __debugprintf (char flag, char *file, int line,
44:                    char *format, ...) {
45:     va_list args;
46:     if (alldebugflags || strchr (debugflags, flag) != NULL) {
47:         fflush (NULL);
48:         print_execname (stderr);
49:         fprintf (stderr, "DEBUGF(%c): %s[%d]:\n",
50:                 flag, file, line);
51:         va_start (args, format);
52:         vfprintf (stderr, format, args);
53:         va_end (args);
54:         fflush (NULL);
55:     }
56: }
57:
```

```
1: // $Id: stack.c,v 1.3 2012-03-06 16:27:19-08 - - $
2:
3: #include <stdio.h>
4: #include <assert.h>
5:
6: #include "stack.h"
7:
8: typedef struct stack_node *stack_node_ref;
9:
10: static char *stack_tag = "struct stack";
11: static char *stack_node_tag = "struct stack_node";
12:
13: struct stack {
14:     char *tag;
15:     stack_node_ref top;
16: };
17:
18: struct stack_node {
19:     char *tag;
20:     stack_item item;
21:     stack_node_ref link;
22: };
23:
24: stack_ref new_stack (void) {
25:     return NULL;
26: }
27:
28: void free_stack (stack_ref stack) {
29:     assert (is_stack (stack));
30:     assert (is_empty_stack (stack));
31: }
32:
33: void push_stack (stack_ref stack, stack_item item) {
34:     assert (is_stack (stack));
35: }
36:
37: stack_item pop_stack (stack_ref stack) {
38:     assert (is_stack (stack));
39:     assert (! is_empty_stack (stack));
40:     return NULL;
41: }
42:
43: stack_item peek_stack (stack_ref stack, int index) {
44:     assert (is_stack (stack));
45:     assert (index >= 0);
46:     assert (index < length_stack (stack));
47:     return NULL;
48: }
49:
50: bool is_empty_stack (stack_ref stack) {
51:     assert (is_stack (stack));
52:     return false;
53: }
54:
55: int length_stack (stack_ref stack) {
56:     assert (is_stack (stack));
57:     return 0;
58: }
59:
60: bool is_stack (stack_ref stack) {
61:     return false;
62: }
63:
```

```
1: // $Id: bigint.c,v 1.3 2012-03-06 16:27:19-08 - - $
2:
3: #include <assert.h>
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include <string.h>
7:
8: #include "bigint.h"
9:
10: static char *bigint_tag = "struct bigint";
11:
12: struct bigint {
13:     char *tag;
14:     bool is_negative;
15:     size_t length;
16:     size_t digits;
17:     char *buffer;
18: };
19:
20: static void trim_zeros (bigint_ref bigint) {
21:     while (bigint->digits > 0) {
22:         size_t digitpos = bigint->digits - 1;
23:         if (bigint->buffer[digitpos] != 0) break;
24:         --bigint->digits;
25:     }
26: }
27:
28: static bigint_ref do_add (bigint_ref left, bigint_ref right) {
29: }
30:
31: static bigint_ref do_sub (bigint_ref left, bigint_ref right) {
32: }
33:
```

```
34:
35: bigint_ref new_bigint (size_t length) {
36:     bigint_ref bigint = malloc (sizeof (struct bigint));
37:     assert (bigint != NULL);
38:     bigint->tag = bigint_tag;
39:     bigint->is_negative = false;
40:     bigint->length = length;
41:     bigint->digits = 0;
42:     bigint->buffer = calloc (length, sizeof (char));
43:     assert (bigint->buffer != NULL);
44:     return bigint;
45: }
46:
47: bigint_ref from_string_bigint (char *string) {
48:     assert (string != NULL);
49:     size_t length = strlen (string);
50:     bigint_ref bigint = new_bigint (length);
51:     if (*string == '_') {
52:         bigint->is_negative = true;
53:         ++string;
54:         --length;
55:     }
56:     int index = 0;
57:     while (--length > 0) {
58:         // LINTED (assignment of 32-bit integer to 8-bit integer)
59:         char digit = string[length] - '0';
60:         assert (0 <= digit && digit <= 9);
61:         bigint->buffer[index++] = digit;
62:     }
63:     trim_zeros (bigint);
64:     return NULL;
65: }
66:
```



```
67:
68: void free_bigint (bigint_ref bigint) {
69:     assert (is_bigint (bigint));
70: }
71:
72: void print_bigint (bigint_ref bigint) {
73:     assert (is_bigint (bigint));
74: }
75:
76: bigint_ref add_bigint (bigint_ref left, bigint_ref right) {
77:     assert (is_bigint (left));
78:     assert (is_bigint (right));
79:     return NULL;
80: }
81:
82: bigint_ref sub_bigint (bigint_ref left, bigint_ref right) {
83:     assert (is_bigint (left));
84:     assert (is_bigint (right));
85:     return NULL;
86: }
87:
88: bigint_ref mul_bigint (bigint_ref left, bigint_ref right) {
89:     assert (is_bigint (left));
90:     assert (is_bigint (right));
91:     return NULL;
92: }
93:
94: bool is_bigint (bigint_ref bigint) {
95:     return false;
96: }
97:
```

```
1: // $id$
2:
3: #include <ctype.h>
4: #include <assert.h>
5: #include <stdio.h>
6: #include <stdlib.h>
7:
8: #include "token.h"
9: #include "debugf.h"
10:
11: #define BUFFER_LENGTH 16
12:
13: struct token {
14:     FILE *file;
15:     int token;
16:     size_t length;
17:     size_t chars;
18:     char *buffer;
19: };
20:
21: token_ref new_token (FILE *file) {
22:     token_ref new = malloc (sizeof (struct token));
23:     assert (new != NULL);
24:     new->file = file;
25:     new->token = 0;
26:     new->length = BUFFER_LENGTH;
27:     new->buffer = malloc (new->length);
28:     assert (new->buffer != NULL);
29:     new->buffer[0] = '\0';
30:     new->chars = 0;
31:     DEBUGF ('t', "token_ref=%p\n", new);
32:     return new;
33: }
34:
35: char *peek_token (token_ref token) {
36:     DEBUGF ('t', "peek %p [%d] \"%s\"\n", token, token->chars,
37:             token->buffer);
38:     return token->buffer;
39: }
40:
```

```
41:
42: int scan_token (token_ref token) {
43:     token->chars = 0;
44:     token->buffer[token->chars] = '\0';
45:     int result = EOF;
46:     int nextchar = 0;
47:     do {
48:         nextchar = fgetc (token->file);
49:     } while (isspace (nextchar));
50:     if (nextchar == EOF) {
51:         result = EOF;
52:     }else if (nextchar == '_' || isdigit (nextchar)) {
53:         do {
54:             // LINTED (assignment of 32-bit integer to 8-bit integer)
55:             token->buffer[token->chars++] = nextchar;
56:             if (token->chars == token->length) {
57:                 token->length *= 2;
58:                 token->buffer = realloc (token->buffer, token->length);
59:                 assert (token->buffer);
60:             }
61:             nextchar = fgetc (token->file);
62:         } while (isdigit (nextchar));
63:         token->buffer[token->chars] = '\0';
64:         int ungetchar = ungetc (nextchar, token->file);
65:         assert (ungetchar == nextchar);
66:         result = NUMBER;
67:     }else {
68:         result = nextchar;
69:     }
70:     DEBUGF ('t', "scan %p [%d] \"%s\" %d\n", token, token->chars,
71:             token->buffer, result);
72:     return result;
73: }
74:
```

```
1: // $Id: main.c,v 1.3 2012-03-06 16:27:19-08 - - $
2:
3: #include <assert.h>
4: #include <ctype.h>
5: #include <libgen.h>
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <string.h>
9: #include <unistd.h>
10:
11: #include "bigint.h"
12: #include "debugf.h"
13: #include "stack.h"
14: #include "token.h"
15:
16: char *execname = NULL;
17:
18: #define DO_NOTHING(X) {DEBUGF ('s', ""); return X; }
19:
20: bool not_enough (stack_ref stack, int enough) {
21:     DO_NOTHING(false);
22:     if (length_stack (stack) >= enough) return true;
23:     printf ("%s: stack empty\n", execname);
24:     return false;
25: }
26:
27: void do_push (stack_ref stack, char *yytext) {
28:     DO_NOTHING();
29:     bigint_ref bigint = from_string_bigint (yytext);
30:     push_stack (stack, bigint);
31: }
32:
33: void do_binop (stack_ref stack, bigint_binop binop) {
34:     DO_NOTHING();
35:     if (not_enough (stack, 2)) return;
36:     bigint_ref right = pop_stack (stack);
37:     bigint_ref left = pop_stack (stack);
38:     bigint_ref answer = binop (left, right);
39:     push_stack (stack, answer);
40:     free_bigint (left);
41:     free_bigint (right);
42: }
43:
44: void do_clear (stack_ref stack) {
45:     DO_NOTHING();
46:     while (! is_empty_stack (stack)) {
47:         bigint_ref bigint = pop_stack (stack);
48:         free_bigint (bigint);
49:     }
50: }
51:
```

```
52:
53: void do_print (stack_ref stack) {
54:     DO_NOTHING();
55:     if (not_enough (stack, 1)) return;
56:     print_bigint (peek_stack (stack, 0));
57: }
58:
59: void do_print_all (stack_ref stack) {
60:     DO_NOTHING();
61:     int length = length_stack (stack);
62:     for (int index = 0; index < length; ++index) {
63:         print_bigint (peek_stack (stack, index));
64:     }
65: }
66:
67: void unimplemented (int oper) {
68:     printf ("%s: ", execname);
69:     if (isgraph (oper)) printf ("'%c' (0%o)", oper, oper);
70:     else printf ("0%o", oper);
71:     printf (" unimplemented\n");
72: }
73:
74: void scan_options (int argc, char **argv) {
75:     opterr = false;
76:     for (;;) {
77:         int option = getopt (argc, argv, "y@:");
78:         if (option == EOF) break;
79:         switch (option) {
80:             case '@': set_debugflags (optarg);
81:                     break;
82:             default : printf ("%s: -%c: invalid option\n",
83:                               execname, optopt);
84:                     break;
85:         }
86:     }
87: }
88:
89: int main (int argc, char **argv) {
90:     execname = basename (argv[0]);
91:     set_execname (execname);
92:     scan_options (argc, argv);
93:     stack_ref stack = new_stack ();
94:     token_ref scanner = new_token (stdin);
95:     for (;;) {
96:         int token = scan_token (scanner);
97:         if (token == EOF) break;
98:         switch (token) {
99:             case NUMBER: do_push (stack, peek_token (scanner)); break;
100:            case '+': do_binop (stack, add_bigint); break;
101:            case '-': do_binop (stack, sub_bigint); break;
102:            case '*': do_binop (stack, mul_bigint); break;
103:            case 'c': do_clear (stack); break;
104:            case 'f': do_print_all (stack); break;
105:            case 'p': do_print (stack); break;
106:            default: unimplemented (token); break;
107:        }
108:    }
109:    return EXIT_SUCCESS;
110: }
```

```
1: # $Id: Makefile,v 1.3 2012-03-06 18:41:37-08 - - $
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=gnu99
10: MKDEPS      = gcc -MM
11: LINT        = lint -Xa -fd -m -u -x -errchk=%all
12:
13: CSOURCE     = debugf.c stack.c bigint.c token.c main.c
14: CHEADER     = debugf.h stack.h bigint.h token.h
15: OBJECTS     = ${CSOURCE:.c=.o}
16: EXECBIN     = a5dc
17: SUBMITS     = ${CHEADER} ${CSOURCE} ${MKFILE}
18: SOURCES     = ${SUBMITS}
19: LISTING     = Listing.code.ps
20: PROJECT     = cmpps012b-wm.w12 asg5
21:
22: all : ${EXECBIN}
23:
24: ${EXECBIN} : ${OBJECTS}
25:             ${GCC} -o $@ ${OBJECTS}
26:
27: %.o : %.c
28:         ${GCC} -c $<
29:
30: lint : ${CSOURCE}
31:         ${LINT} ${CSOURCE}
32:         checksource ${SUBMITS}
33:
34: ci : ${SOURCES}
35:         cid + ${SOURCES}
36:         checksource ${SUBMITS}
37:
38: lis : ${SOURCES} ${DEPSFILE}
39:         mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}
40:
41: clean :
42:         - rm ${OBJECTS} ${DEPSFILE} core ${EXECBIN}.errs
43:
44: spotless : clean
45:         - rm ${EXECBIN}
46:
47: submit : ${SUBMITS}
48:         submit ${PROJECT} ${SUBMITS}
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:         ${GMAKE} spotless deps ci lint all lis
60:
61: ifeq "${NEEDINCL}" ""
62: include ${DEPSFILE}
63: endif
64:
```

```
1: # Makefile.deps created Tue Mar  6 18:41:37 PST 2012
2: debugf.o: debugf.c debugf.h
3: stack.o: stack.c stack.h
4: bigint.o: bigint.c bigint.h
5: token.o: token.c token.h debugf.h
6: main.o: main.c bigint.h debugf.h stack.h token.h
```