```
 1: // $Id: debug.h,v 1.4 2013-05-16 15:07:42-07 - - $
 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 STUB(STMT) STMT
23:
24: //
25: // Sets a string of debug flags to be used by DEBUGF and DEBUGS.
26: // If a particular debug flag has been set, messages are printed.
27: // The flag "@" turns on all flags.
28: //
29: void set_debug_flags (char *flags);
30:
31: //
32: // Check if a debug flag is set.
33: //
34: bool get_debug_flag (char flag);
35:
36: //
37: // DEBUGF takes printf-like arguments.
38: // DEBUGS takes any fprintf(stderr...) statement as an argument.
39: //
40: #ifdef NDEBUG
41: #define DEBUGF(FLAG,...)
42: #define DEBUGS(FLAG,STMT)
43: #else
44: #define DEBUGF(FLAG,...) \
45:         if (get_debug_flag (FLAG)) { \
46:            __show_debug (FLAG, __FILE__, __LINE__, __func__); \
47:            fprintf (stderr, __VA_ARGS__); \
48:            fflush (NULL); \
49:         }
50: #define DEBUGS(FLAG,STMT) \
51:         if (get_debug_flag (FLAG)) { \
52:            __show_debug (FLAG, __FILE__, __LINE__, __func__); \
53:            STMT; \
54:            fflush (NULL); \
55:         }
56: void __show_debug (char flag, char *file, int line, const char *func);
57: #endif
58:
59: #endif
60:
```

```
 1: // $Id: stack.h,v 1.5 2013-05-08 18:53:30-07 - - $
 2:
 3: #ifndef __STACK_H__
 4: #define __STACK_H__
 5:
 6: #include <stdbool.h>
 7:
 8: #include "bigint.h"
 9:
10: typedef struct stack stack;
11: typedef bigint *stack_item;
12:
13: //
14: // Create a new empty stack.
15: //
16: stack *new_stack (void);
17:
18: //
19: // Free up the stack.
20: // Precondition: stack must be empty.
21: //
22: void free_stack (stack *);
23:
24: //
25: // Push a new stack_item onto the top of the stack.
26: //
27: void push_stack (stack *, stack_item);
28:
29: //
30: // Pop the top stack_item from the stack and return it.
31: //
32: stack_item pop_stack (stack *);
33:
34: //
35: // Peek into the stack and return a selected stack_item.
36: // Item 0 is the element at the top.
37: // Item size_stack - 1 is the element at the bottom.
38: // Precondition: 0 <= index && index < size_stack.
39: //
40: stack_item peek_stack (stack *, size_t index);
41:
42: //
43: // Indicate whether the stack is empty or not.
44: // Same as size_stack == 0.
45: //
46: bool empty_stack (stack *);
47:
48: //
49: // Return the current size of the stack (number of items on the stack).
50: //
51: size_t size_stack (stack *);
52:
53: //
54: // Print part of the stack in debug format.
55: //
56: void show_stack (stack *);
57:
58: #endif
59:
```

```
 1: // $Id: bigint.h,v 1.6 2013-05-07 21:14:09-07 - - $
 2:
 3: #ifndef __BIGINT_H__
 4: #define __BIGINT_H__
 5:
 6: #include <stdbool.h>
 7:
 8: typedef struct bigint bigint;
 9:
10: typedef bigint *(*bigint_binop) (bigint *, bigint *);
11:
12: bigint *new_bigint (size_t capacity);
13:
14: bigint *new_string_bigint (char *string);
15:
16: void free_bigint (bigint *);
17:
18: void print_bigint (bigint *, FILE *);
19:
20: bigint *add_bigint (bigint *, bigint *);
21:
22: bigint *sub_bigint (bigint *, bigint *);
23:
24: bigint *mul_bigint (bigint *, bigint *);
25:
26: void show_bigint (bigint *);
27:
28: #endif
29:
```

```
 1: // $Id: token.h,v 1.3 2013-05-08 22:09:41-07 - - $
 2:
 3: #ifndef __TOKEN_H__
 4: #define __TOKEN_H__
 5:
 6: #include <stdbool.h>
 7:
 8: #define NUMBER 256
 9:
10: typedef struct token token;
11:
12: token *new_token (FILE*);
13:
14: void free_token (token *);
15:
16: int scan_token (token *);
17:
18: char *peek_token (token *);
19:
20: void show_token (token *);
21:
22: #endif
23:
```

```
 1: // $Id: debug.c,v 1.3 2013-05-08 22:09:41-07 - - $
 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 set_debug_flags (char *flags) {
17:    if (strchr (flags, '@') != NULL) {
18:       memset (debug_flags, true, sizeof debug_flags);
19:    }else {
20:       for (char *flag = flags; *flag != '\0'; ++flag) {
21:          debug_flags[(unsigned char) *flag] = true;
22:       }
23:    }
24: }
25:
26: bool get_debug_flag (char flag) {
27:    return debug_flags[(unsigned char) flag];
28: }
29:
30: void __show_debug (char flag, char *file, int line, const char *func) {
31:    fflush (NULL);
32:    assert (program_name != NULL);
33:    fprintf (stderr, "%s: DEBUGF(%c): %s[%d]: %s()\n",
34:             program_name, flag, file, line, func);
35: }
36:
```

```
 1: // $Id: stack.c,v 1.11 2013-05-16 15:07:42-07 - - $
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5: #include <stdlib.h>
 6: #include <string.h>
 7:
 8: #include "stack.h"
 9: #include "debug.h"
10:
11: #define DEFAULT_CAPACITY 16
12:
13: struct stack {
14:    size_t capacity;
15:    size_t size;
16:    stack_item *data;
17: };
18:
19: stack *new_stack (void) {
20:    stack *this = malloc (sizeof (stack));
21:    assert (this != NULL);
22:    this->capacity = DEFAULT_CAPACITY;
23:    this->size = 0;
24:    this->data = calloc (this->capacity, sizeof (stack_item));
25:    assert (this->data != NULL);
26:    return this;
27: }
28:
29: void free_stack (stack *this) {
30:    assert (empty_stack (this));
31:    free (this->data);
32:    free (this);
33: }
34:
35: static bool full_stack (stack *this) {
36:    return this->size == this->capacity;
37: }
38:
39: static void realloc_stack (stack *this) {
40:    size_t old_capacity = this->capacity;
41:    this->capacity *= 2;
42:    this->data = realloc (this->data, this->capacity);
43:    memset (this->data + old_capacity, 0, old_capacity);
44:    assert (this->data != NULL);
45: }
46:
```

```
47:
48: void push_stack (stack *this, stack_item item) {
49:    if (full_stack (this)) realloc_stack (this);
50:    DEBUGS ('s', show_stack (this));
51:    DEBUGF ('s', "item=%p\n", item);
52: }
53:
54: stack_item pop_stack (stack *this) {
55:    assert (! empty_stack (this));
56:    DEBUGS ('s', show_stack (this));
57:    STUB (return NULL;)
58: }
59:
60: stack_item peek_stack (stack *this, size_t index) {
61:    assert (index < size_stack (this));
62:    DEBUGS ('s', show_stack (this));
63:    STUB (return NULL;)
64: }
65:
66: bool empty_stack (stack *this) {
67:    return size_stack (this) == 0;
68: }
69:
70: size_t size_stack (stack *this) {
71:    return this->size;
72: }
73:
74: void show_stack (stack *this) {
75:    fprintf (stderr, "stack@%p->{%lu,%lu,%p}\n",
76:             this, this->capacity, this->size, this->data);
77: }
78:
```

```
 1: // $Id: bigint.c,v 1.12 2013-05-16 15:14:31-07 - - $
 2:
 3: #include <assert.h>
 4: #include <ctype.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8:
 9: #include "bigint.h"
10: #include "debug.h"
11:
12: #define MIN_CAPACITY 16
13:
14: struct bigint {
15:    size_t capacity;
16:    size_t size;
17:    bool negative;
18:    char *digits;
19: };
20:
21: static void trim_zeros (bigint *this) {
22:    while (this->size > 0) {
23:       size_t digitpos = this->size - 1;
24:       if (this->digits[digitpos] != 0) break;
25:       --this->size;
26:    }
27: }
28:
29: bigint *new_bigint (size_t capacity) {
30:    bigint *this = malloc (sizeof (bigint));
31:    assert (this != NULL);
32:    this->capacity = capacity;
33:    this->size = 0;
34:    this->negative = false;
35:    this->digits = calloc (this->capacity, sizeof (char));
36:    assert (this->digits != NULL);
37:    DEBUGS ('b', show_bigint (this));
38:    return this;
39: }
40:
41: bigint *new_string_bigint (char *string) {
42:    assert (string != NULL);
43:    size_t length = strlen (string);
44:    bigint *this = new_bigint (length > MIN_CAPACITY
45:                               ? length : MIN_CAPACITY);
46:    char *strdigit = &string[length - 1];
47:    if (*string == '_') {
48:       this->negative = true;
49:       ++string;
50:    }
51:    char *thisdigit = this->digits;
52:    while (strdigit >= string) {
53:       assert (isdigit (*strdigit));
54:       *thisdigit++ = *strdigit-- - '0';
55:    }
56:    this->size = thisdigit - this->digits;
57:    trim_zeros (this);
58:    DEBUGS ('b', show_bigint (this));
59:    return this;;
60: }
61:
```

```
 62:
 63: static bigint *do_add (bigint *this, bigint *that) {
 64:    DEBUGS ('b', show_bigint (this));
 65:    DEBUGS ('b', show_bigint (that));
 66:    STUB (return NULL);
 67: }
 68:
 69: static bigint *do_sub (bigint *this, bigint *that) {
 70:    DEBUGS ('b', show_bigint (this));
 71:    DEBUGS ('b', show_bigint (that));
 72:    STUB (return NULL);
 73: }
 74: void free_bigint (bigint *this) {
 75:    free (this->digits);
 76:    free (this);
 77: }
 78:
 79: void print_bigint (bigint *this, FILE *file) {
 80:    DEBUGS ('b', show_bigint (this));
 81: }
 82:
 83: bigint *add_bigint (bigint *this, bigint *that) {
 84:    DEBUGS ('b', show_bigint (this));
 85:    DEBUGS ('b', show_bigint (that));
 86:    STUB (return NULL);
 87:    return NULL;
 88: }
 89:
 90: bigint *sub_bigint (bigint *this, bigint *that) {
 91:    DEBUGS ('b', show_bigint (this));
 92:    DEBUGS ('b', show_bigint (that));
 93:    STUB (return NULL);
 94:    return NULL;
 95: }
 96:
 97: bigint *mul_bigint (bigint *this, bigint *that) {
 98:    DEBUGS ('b', show_bigint (this));
 99:    DEBUGS ('b', show_bigint (that));
100:    STUB (return NULL);
101:    return NULL;
102: }
103:
104: void show_bigint (bigint *this) {
105:    fprintf (stderr, "bigint@%p->{%lu,%lu,%c,%p->", this,
106:             this->capacity, this->size, this->negative ? '-' : '+',
107:             this->digits);
108:    for (char *byte = &this->digits[this->size - 1];
109:         byte >= this->digits; --byte) {
110:       fprintf (stderr, "%d", *byte);
111:    }
112:    fprintf (stderr, "}\n");
113: }
114:
```

```
 1: // $Id: token.c,v 1.8 2013-05-16 15:14:31-07 - - $
 2:
 3: #include <assert.h>
 4: #include <ctype.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8:
 9: #include "token.h"
10: #include "debug.h"
11:
12: #define INIT_CAPACITY 16
13:
14: struct token {
15:    FILE *file;
16:    size_t capacity;
17:    size_t size;
18:    int token;
19:    char *buffer;
20: };
21:
22: token *new_token (FILE *file) {
23:    token *this = malloc (sizeof (token));
24:    assert (this != NULL);
25:    this->file = file;
26:    this->capacity = INIT_CAPACITY;
27:    this->buffer = malloc (this->capacity);
28:    assert (this->buffer != NULL);
29:    this->buffer[0] = '\0';
30:    this->size = 0;
31:    this->token = 0;
32:    DEBUGS ('t', show_token (this));
33:    return this;
34: }
35:
36: void free_token (token *this) {
37:    free (this->buffer);
38:    free (this);
39: }
40:
41: char *peek_token (token *this) {
42:    DEBUGS ('t', show_token (this));
43:    return this->buffer;
44: }
45:
```

```
46:
47: void ensure_capacity (token *this, size_t capacity) {
48:    if (capacity > this->capacity) {
49:       size_t double_capacity = this->capacity * 2;
50:       this->capacity = capacity > double_capacity
51:                      ? capacity : double_capacity;
52:       this->buffer = realloc (this->buffer, this->capacity);
53:       assert (this->buffer);
54:    }
55: }
56:
57: int scan_token (token *this) {
58:    this->size = 0;
59:    this->buffer[this->size] = '\0';
60:    int result = EOF;
61:    int nextchar = 0;
62:    do {
63:       nextchar = fgetc (this->file);
64:    } while (isspace (nextchar));
65:    if (nextchar == EOF) {
66:       result = EOF;
67:    }else if (nextchar == '_' || isdigit (nextchar)) {
68:       do {
69:          this->buffer[this->size++] = nextchar;
70:          ensure_capacity (this, this->size + 1);
71:          nextchar = fgetc (this->file);
72:       } while (isdigit (nextchar));
73:       this->buffer[this->size] = '\0';
74:       int ungetchar = ungetc (nextchar, this->file);
75:       assert (ungetchar == nextchar);
76:       result = NUMBER;
77:    }else {
78:       result = nextchar;
79:    }
80:    DEBUGS ('t', show_token (this));
81:    return result;
82: }
83:
84: void show_token (token *this) {
85:    fprintf (stderr, "token@%p->{%lu,%lu,%d,%p->\"%s\"}\n",
86:             this, this->capacity, this->size, this->token,
87:             this->buffer, this->buffer);
88: }
89:
```

```
 1: // $Id: main.c,v 1.8 2013-05-08 22:09:41-07 - - $
 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 "debug.h"
13: #include "stack.h"
14: #include "token.h"
15:
16: void do_push (stack *stack, char *numstr) {
17:    DEBUGF ('m', "stack=%p, numstr=%p=\"%s\"\n", stack, numstr, numstr);
18:    bigint *bigint = new_string_bigint (numstr);
19:    push_stack (stack, bigint);
20: }
21:
22: void do_binop (stack *stack, bigint_binop binop) {
23:    DEBUGS ('m', show_stack (stack));
24:    bigint *right = pop_stack (stack);
25:    bigint *left = pop_stack (stack);
26:    bigint *answer = binop (left, right);
27:    push_stack (stack, answer);
28:    free_bigint (left);
29:    free_bigint (right);
30: }
31:
32: void do_clear (stack *stack) {
33:    DEBUGF ('m', "stack=%p\n", stack);
34:    while (! empty_stack (stack)) {
35:       bigint *bigint = pop_stack (stack);
36:       free_bigint (bigint);
37:    }
38: }
39:
```

```
40:
41: void do_print (stack *stack) {
42:    DEBUGS ('m', show_stack (stack));
43:    print_bigint (peek_stack (stack, 0), stdout);
44: }
45:
46: void do_print_all (stack *stack) {
47:    DEBUGS ('m', show_stack (stack));
48:    int size = size_stack (stack);
49:    for (int index = 0; index < size; ++index) {
50:       print_bigint (peek_stack (stack, index), stdout);
51:    }
52: }
53:
54: void unimplemented (int oper) {
55:    printf ("%s: ", program_name);
56:    if (isgraph (oper)) printf ("'%c' (0%o)", oper, oper);
57:                   else printf ("0%o", oper);
58:    printf (" unimplemented\n");
59: }
60:
61: void scan_options (int argc, char **argv) {
62:    opterr = false;
63:    for (;;) {
64:       int option = getopt (argc, argv, "@:");
65:       if (option == EOF) break;
66:       switch (option) {
67:          case '@': set_debug_flags (optarg);
68:                    break;
69:          default : printf ("%s: -%c: invalid option\n",
70:                            program_name, optopt);
71:                    break;
72:       }
73:    }
74: }
75:
76: int main (int argc, char **argv) {
77:    program_name = basename (argv[0]);
78:    scan_options (argc, argv);
79:    stack *stack = new_stack ();
80:    token *scanner = new_token (stdin);
81:    for (;;) {
82:       int token = scan_token (scanner);
83:       if (token == EOF) break;
84:       switch (token) {
85:          case NUMBER: do_push (stack, peek_token (scanner)); break;
86:          case '+': do_binop (stack, add_bigint); break;
87:          case '-': do_binop (stack, sub_bigint); break;
88:          case '*': do_binop (stack, mul_bigint); break;
89:          case 'c': do_clear (stack); break;
90:          case 'f': do_print_all (stack); break;
91:          case 'p': do_print (stack); break;
92:          default: unimplemented (token); break;
93:       }
94:    }
95:    DEBUGF ('m', "EXIT %d\n", exit_status);
96:    return EXIT_SUCCESS;
97: }
```

```
 1: # $Id: Makefile,v 1.5 2013-05-07 21:14:09-07 - - $
 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:
12: CSOURCE   = debug.c stack.c bigint.c token.c main.c
13: CHEADER   = debug.h stack.h bigint.h token.h
14: OBJECTS   = ${CSOURCE:.c=.o}
15: EXECBIN   = mydc
16: SUBMITS   = ${CHEADER} ${CSOURCE} ${MKFILE}
17: SOURCES   = ${SUBMITS}
18: LISTING   = Listing.ps
19: PROJECT   = cmps012b-wm.w13 asg4
20:
21: all : ${EXECBIN}
22:
23: ${EXECBIN} : ${OBJECTS}
24:         ${GCC} -o $@ ${OBJECTS}
25:
26: %.o : %.c
27:         ${GCC} -c $<
28:
29: ci : ${SOURCES}
30:         cid + ${SOURCES}
31:         checksource ${SUBMITS}
32:
33: lis : ${SOURCES} ${DEPSFILE}
34:         mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}
35:
36: clean :
37:         - rm ${OBJECTS} ${DEPSFILE} core ${EXECBIN}.errs
38:
39: spotless : clean
40:         - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf}
41:
42: submit : ${SUBMITS}
43:         submit ${PROJECT} ${SUBMITS}
44:
45: deps : ${CSOURCE} ${CHEADER}
46:         @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
47:         ${MKDEPS} ${CSOURCE} >>${DEPSFILE}
48:
49: ${DEPSFILE} :
50:         @ touch ${DEPSFILE}
51:         ${GMAKE} deps
52:
53: again :
54:         ${GMAKE} spotless deps ci all lis
55:
56: ifeq "${NEEDINCL}" ""
57: include ${DEPSFILE}
58: endif
59:
```

```
1: # Makefile.deps created Thu May 16 15:28:18 PDT 2013
2: debug.o: debug.c debug.h
3: stack.o: stack.c stack.h bigint.h debug.h
4: bigint.o: bigint.c bigint.h debug.h
5: token.o: token.c token.h debug.h
6: main.o: main.c bigint.h debug.h stack.h token.h
```