

```
1: // $Id: debug.h,v 1.5 2014-01-24 18:33:47-08 - - $
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: #define DEBUGF(FLAG,...) \
41:     if (get_debug_flag (FLAG)) { \
42:         __show_debug (FLAG, __FILE__, __LINE__, __func__); \
43:         fprintf (stderr, __VA_ARGS__); \
44:         fflush (NULL); \
45:     }
46: #define DEBUGS(FLAG,STMT) \
47:     if (get_debug_flag (FLAG)) { \
48:         __show_debug (FLAG, __FILE__, __LINE__, __func__); \
49:         STMT; \
50:         fflush (NULL); \
51:     }
52: void __show_debug (char flag, char *file, int line, const char *func);
53:
54: #endif
55:
```

```
1: // $Id: stack.h,v 1.6 2014-01-24 18:33:47-08 - - $
2:
3: #ifndef __STACK_H__
4: #define __STACK_H__
5:
6: #include <stdbool.h>
7: #include "bigint.h"
8:
9: typedef struct stack stack;
10: typedef bigint *stack_item;
11:
12: //
13: // Create a new empty stack.
14: //
15: stack *new_stack (void);
16:
17: //
18: // Free up the stack.
19: // Precondition: stack must be empty.
20: //
21: void free_stack (stack*);
22:
23: //
24: // Push a new stack_item onto the top of the stack.
25: //
26: void push_stack (stack *, stack_item);
27:
28: //
29: // Pop the top stack_item from the stack and return it.
30: //
31: stack_item pop_stack (stack*);
32:
33: //
34: // Peek into the stack and return a selected stack_item.
35: // Item 0 is the element at the top.
36: // Item size_stack - 1 is the element at the bottom.
37: // Precondition: 0 <= index && index < size_stack.
38: //
39: stack_item peek_stack (stack *, size_t index);
40:
41: //
42: // Indicate whether the stack is empty or not.
43: // Same as size_stack == 0.
44: //
45: bool empty_stack (stack*);
46:
47: //
48: // Return the current size of the stack (number of items on the stack).
49: //
50: size_t size_stack (stack*);
51:
52: //
53: // Print part of the stack in debug format.
54: //
55: void show_stack (stack*);
56:
57: #endif
58:
```

```
1: // $Id: bigint.h,v 1.9 2015-02-03 18:11:58-08 - - $
2:
3: #ifndef __BIGINT_H__
4: #define __BIGINT_H__
5:
6: typedef struct bigint bigint;
7:
8: typedef bigint *(*bigint_binop) (bigint*, bigint*);
9:
10: bigint *new_bigint (size_t capacity);
11:
12: bigint *new_string_bigint (const char *string);
13:
14: void free_bigint (bigint*);
15:
16: void print_bigint (bigint*);
17:
18: bigint *add_bigint (bigint*, bigint*);
19:
20: bigint *sub_bigint (bigint*, bigint*);
21:
22: bigint *mul_bigint (bigint*, bigint*);
23:
24: void show_bigint (bigint*);
25:
26: #endif
27:
```

```
1: // $Id: token.h,v 1.4 2014-01-24 18:33:47-08 - - $
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: yyextern.h,v 1.1 2015-02-03 18:11:58-08 - - $
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: #define YYEOF 0
12: #define YYNUMBER 256
13:
14: extern char *yytext;           // Pointer to the string that was found
15: extern int yy_flex_debug;     // yylex's verbose tracing flag
16: extern int yylex (void);      // Read next word from opened file yyin
17: extern void yycleanup (void); // Cleans up flex's buffers when done
18:
19: #endif
20:
```

```
1: // $Id: debug.c,v 1.5 2015-02-03 18:29:23-08 - - $
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: #include "yyextern.h"
12:
13: static char debug_flags[UCHAR_MAX + 1];
14: char *program_name = NULL;
15: int exit_status = EXIT_SUCCESS;
16:
17: void set_debug_flags (char *flags) {
18:     if (strchr (flags, '@') != NULL) {
19:         memset (debug_flags, true, sizeof debug_flags);
20:     } else {
21:         for (char *flag = flags; *flag != '\0'; ++flag) {
22:             if (*flag == 'y') yy_flex_debug = true;
23:             debug_flags[(unsigned char) *flag] = true;
24:         }
25:     }
26: }
27:
28: bool get_debug_flag (char flag) {
29:     return debug_flags[(unsigned char) flag];
30: }
31:
32: void __show_debug (char flag, char *file, int line, const char *func) {
33:     fflush (NULL);
34:     assert (program_name != NULL);
35:     fprintf (stderr, "%s: DEBUGF(%c): %s[%d]: %s()\n",
36:             program_name, flag, file, line, func);
37: }
38:
```

```
1: // $Id: stack.c,v 1.12 2014-05-14 18:03:26-07 - - $
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:     assert (this->data != NULL);
44:     memset (this->data + old_capacity, 0, old_capacity);
45: }
46:
```

```
47:
48: void push_stack (stack *this, stack_item item) {
49:     if (full_stack (this)) realloc_stack (this);
50:     DEBUGGS ('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:     DEBUGGS ('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:     DEBUGGS ('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 1.15 2015-02-03 18:11:58-08 - - $
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: 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:
42: bigint *new_string_bigint (const char *string) {
43:     assert (string != NULL);
44:     size_t length = strlen (string);
45:     bigint *this = new_bigint (length > MIN_CAPACITY
46:                               ? length : MIN_CAPACITY);
47:     const char *strdigit = &string[length - 1];
48:     if (*string == '_') {
49:         this->negative = true;
50:         ++string;
51:     }
52:     char *thisdigit = this->digits;
53:     while (strdigit >= string) {
54:         assert (isdigit (*strdigit));
55:         *thisdigit++ = *strdigit-- - '0';
56:     }
57:     this->size = thisdigit - this->digits;
58:     trim_zeros (this);
59:     DEBUGS ('b', show_bigint (this));
60:     return this;
61: }
62:
63: 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: 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) {
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:
98: bigint *mul_bigint (bigint *this, bigint *that) {
99:     DEBUGS ('b', show_bigint (this));
100:    DEBUGS ('b', show_bigint (that));
101:    STUB (return NULL);
102:    return NULL;
103: }
104:
105: void show_bigint (bigint *this) {
106:     fprintf (stderr, "bigint%p->{%lu,%lu,%c,%p->", this,
107:              this->capacity, this->size, this->negative ? '-' : '+',
108:              this->digits);
109:     for (char *byte = &this->digits[this->size - 1];
110:          byte >= this->digits; --byte) {
111:         fprintf (stderr, "%d", *byte);
112:     }
113:     fprintf (stderr, "}\n");
114: }
115:
```

```
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 1.12 2015-02-03 18:26:19-08 - - $
2:
3: #include <assert.h>
4: #include <ctype.h>
5: #include <libgen.h>
6: #include <stdbool.h>
7: #include <stdio.h>
8: #include <stdlib.h>
9: #include <string.h>
10: #include <unistd.h>
11:
12: #include "bigint.h"
13: #include "debug.h"
14: #include "stack.h"
15: #include "token.h"
16: #include "yyextern.h"
17:
18: void do_push (stack *stack, char *numstr) {
19:     DEBUGF ('m', "stack=%p, numstr=%p=\"%s\"\\n", stack, numstr, numstr);
20:     bigint *bigint = new_string_bigint (numstr);
21:     push_stack (stack, bigint);
22: }
23:
24: void do_binop (stack *stack, bigint_binop binop) {
25:     DEBUGS ('m', show_stack (stack));
26:     bigint *right = pop_stack (stack);
27:     bigint *left = pop_stack (stack);
28:     bigint *answer = binop (left, right);
29:     push_stack (stack, answer);
30:     free_bigint (left);
31:     free_bigint (right);
32: }
33:
34: void do_clear (stack *stack) {
35:     DEBUGF ('m', "stack=%p\\n", stack);
36:     while (! empty_stack (stack)) {
37:         bigint *bigint = pop_stack (stack);
38:         free_bigint (bigint);
39:     }
40: }
41:
```

```
42:
43: void do_print (stack *stack) {
44:     DEBUGS ('m', show_stack (stack));
45:     print_bigint (peek_stack (stack, 0));
46: }
47:
48: void do_print_all (stack *stack) {
49:     DEBUGS ('m', show_stack (stack));
50:     int size = size_stack (stack);
51:     for (int index = 0; index < size; ++index) {
52:         print_bigint (peek_stack (stack, index));
53:     }
54: }
55:
56: void unimplemented (int oper) {
57:     printf ("%s: ", program_name);
58:     if (isgraph (oper)) printf ("'%c' (0%o)", oper, oper);
59:     else printf ("0%o", oper);
60:     printf (" unimplemented\n");
61: }
62:
63: void scan_options (int argc, char **argv) {
64:     opterr = false;
65:     for (;;) {
66:         int option = getopt (argc, argv, "@:");
67:         if (option == EOF) break;
68:         switch (option) {
69:             case '@': set_debug_flags (optarg);
70:                     break;
71:             default : printf ("%s: -%c: invalid option\n",
72:                               program_name, optopt);
73:                     break;
74:         }
75:     }
76: }
77:
78: int main (int argc, char **argv) {
79:     program_name = basename (argv[0]);
80:     scan_options (argc, argv);
81:     stack *stack = new_stack ();
82:     bool quit = false;
83:     yy_flex_debug = false;
84:     while (! quit) {
85:         int token = yylex();
86:         if (token == YYEOF) break;
87:         switch (token) {
88:             case NUMBER: do_push (stack, yytext); break;
89:             case '+': do_binop (stack, add_bigint); break;
90:             case '-': do_binop (stack, sub_bigint); break;
91:             case '*': do_binop (stack, mul_bigint); break;
92:             case 'c': do_clear (stack); break;
93:             case 'f': do_print_all (stack); break;
94:             case 'p': do_print (stack); break;
95:             case 'q': quit = true; break;
96:             default: unimplemented (token); break;
97:         }
98:     }
99:     yycleanup();
```

02/09/16  
13:20:05

\$cmps012b-wm/Assignments/asg4c-mydc-stackbignum/code/  
main.c

3/3

```
100:  DEBUGF ('m', "EXIT %d\n", exit_status);  
101:  return EXIT_SUCCESS;  
102: }
```



```
1: %{
2: // $Id: scanner.l,v 1.2 2015-02-03 18:22:49-08 - - $
3:
4: #include "yyextern.h"
5: #define YY_NO_INPUT
6:
7: %}
8:
9: %option 8bit
10: %option debug
11: %option interactive
12: %option nodefault
13: %option nounput
14: %option noyywrap
15:
16: WHITESPACE ([ \t\n])
17: NUMBER      (_?[0-9]*)
18: OTHER       (.)
19:
20: %%
21:
22: [ \t\n]+ { }
23: _?[0-9]* { return YYNUMBER; }
24: .        { return *yytext; }
25:
26: %%
27:
28: void yycleanup (void) {
29:     yy_delete_buffer (YY_CURRENT_BUFFER);
30: }
31:
```

```
1: # $Id: Makefile,v 1.11 2015-02-03 18:27:45-08 - - $
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=gnull
10: MKDEPS      = gcc -MM
11:
12: CSOURCE     = debug.c stack.c bigint.c token.c main.c
13: LSOURCE     = scanner.l
14: CHEADER     = debug.h stack.h bigint.h token.h yyextern.h
15: OBJECTS     = ${CSOURCE:.c=.o} ${LSOURCE:.l=.o}
16: EXECBIN     = mydc
17: SOURCES     = ${CHEADER} ${CSOURCE} ${LSOURCE} ${MKFILE}
18: LISTING     = Listing.ps
19:
20: all : ${EXECBIN}
21:
22: ${EXECBIN} : ${OBJECTS}
23:             ${GCC} -o $@ ${OBJECTS}
24:
25: %.o : %.c
26:             ${GCC} -c $<
27:
28: scanner.c : scanner.l
29:             flex -oscanter.c scanner.l
30:
31: ci : ${SOURCES}
32:             cid + ${SOURCES}
33:             checksource ${SOURCES}
34:
35: lis : ${SOURCES} ${DEPSFILE}
36:             mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}
37:
38: clean :
39:             - rm ${LSOURCE:.l=.c} ${OBJECTS} ${DEPSFILE} core
40:
41: spotless : clean
42:             - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf}
43:
44: deps : ${CSOURCE} ${CHEADER}
45:             @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
46:             ${MKDEPS} ${CSOURCE} >>${DEPSFILE}
47:
48: ${DEPSFILE} :
49:             @ touch ${DEPSFILE}
50:             ${GMAKE} deps
51:
52: again :
53:             ${GMAKE} spotless deps ci all lis
54:
55: ifeq "${NEEDINCL}" ""
56: include ${DEPSFILE}
57: endif
58:
```

```
1: # Makefile.deps created Tue Feb  9 13:20:05 PST 2016
2: debug.o: debug.c debug.h yyextern.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 yyextern.h
```