```
// $Id: debug.h,v 1.5 2014-01-24 18:33:47-08 - - $

#ifndef __DEBUG_H__
#define __DEBUG_H__

#include <stdbool.h>

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

//
// Program name and exit status.
//
extern char *program_name;
extern int exit_status;

//
// Support for STUB statements.
//
#define STUB(STMT) STMT

//
// Sets a string of debug flags to be used by DEBUGF and DEBUGS.
// If a particular debug flag has been set, messages are printed.
// The flag "@" turns on all flags.
//
void set_debug_flags (char *flags);

//
// Check if a debug flag is set.
//
bool get_debug_flag (char flag);

//
// DEBUGF takes printf-like arguments.
// DEBUGS takes any fprintf(stderr...) statement as an argument.
//
#define DEBUGF(FLAG,...) \
        if (get_debug_flag (FLAG)) { \
           __show_debug (FLAG, __FILE__, __LINE__, __func__); \
           fprintf (stderr, __VA_ARGS__); \
           fflush (NULL); \
        }
#define DEBUGS(FLAG,STMT) \
        if (get_debug_flag (FLAG)) { \
           __show_debug (FLAG, __FILE__, __LINE__, __func__); \
           STMT; \
           fflush (NULL); \
        }
void __show_debug (char flag, char *file, int line, const char *func);

#endif

```

```c
// $Id: stack.h,v 1.6 2014-01-24 18:33:47-08 - - $

#ifndef __STACK_H__
#define __STACK_H__

#include <stdbool.h>
#include "bigint.h"

typedef struct stack stack;
typedef bigint *stack_item;

//
// Create a new empty stack.
//
stack *new_stack (void);

//
// Free up the stack.
// Precondition: stack must be empty.
//
void free_stack (stack*);

//
// Push a new stack_item onto the top of the stack.
//
void push_stack (stack *, stack_item);

//
// Pop the top stack_item from the stack and return it.
//
stack_item pop_stack (stack*);

//
// Peek into the stack and return a selected stack_item.
// Item 0 is the element at the top.
// Item size_stack - 1 is the element at the bottom.
// Precondition: 0 <= index && index < size_stack.
//
stack_item peek_stack (stack *, size_t index);

//
// Indicate whether the stack is empty or not.
// Same as size_stack == 0.
//
bool empty_stack (stack*);

//
// Return the current size of the stack (number of items on the stack).
//
size_t size_stack (stack*);

//
// Print part of the stack in debug format.
//
void show_stack (stack*);

#endif

```

```
// $Id: bigint.h,v 1.9 2015-02-03 18:11:58-08 - - $

#ifndef __BIGINT_H__
#define __BIGINT_H__

typedef struct bigint bigint;

typedef bigint *(*bigint_binop) (bigint*, bigint*);

bigint *new_bigint (size_t capacity);

bigint *new_string_bigint (const char *string);

void free_bigint (bigint*);

void print_bigint (bigint*);

bigint *add_bigint (bigint*, bigint*);

bigint *sub_bigint (bigint*, bigint*);

bigint *mul_bigint (bigint*, bigint*);

void show_bigint (bigint*);

#endif

```

```c
// $Id: token.h,v 1.4 2014-01-24 18:33:47-08 - - $

#ifndef __TOKEN_H__
#define __TOKEN_H__

#include <stdbool.h>

#define NUMBER 256

typedef struct token token;

token *new_token (FILE*);

void free_token (token*);

int scan_token (token*);

char *peek_token (token*);

void show_token (token*);

#endif

```

```
// $Id: yyextern.h,v 1.1 2015-02-03 18:11:58-08 - - $

#ifndef __YYEXTERN_H__
#define __YYEXTERN_H__

//
// DESCRIPTION
//    Definitions of external names used by flex-generated code.
//

#define YYEOF 0
#define YYNUMBER 256

extern char *yytext;           // Pointer to the string that was found
extern int yy_flex_debug;      // yylex's verbose tracing flag
extern int yylex (void);       // Read next word from opened file yyin
extern void yycleanup (void); // Cleans up flex's buffers when done

#endif

```

```c
// $Id: debug.c,v 1.5 2015-02-03 18:29:23-08 - - $

#include <assert.h>
#include <limits.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "debug.h"
#include "yyextern.h"

static char debug_flags[UCHAR_MAX + 1];
char *program_name = NULL;
int exit_status = EXIT_SUCCESS;

void set_debug_flags (char *flags) {
   if (strchr (flags, '@') != NULL) {
      memset (debug_flags, true, sizeof debug_flags);
   }else {
      for (char *flag = flags; *flag != '\0'; ++flag) {
         if (*flag == 'y') yy_flex_debug = true;
         debug_flags[(unsigned char) *flag] = true;
      }
   }
}

bool get_debug_flag (char flag) {
   return debug_flags[(unsigned char) flag];
}

void __show_debug (char flag, char *file, int line, const char *func) {
   fflush (NULL);
   assert (program_name != NULL);
   fprintf (stderr, "%s: DEBUGF(%c): %s[%d]: %s()\n",
            program_name, flag, file, line, func);
}

```

```c
// $Id: stack.c,v 1.12 2014-05-14 18:03:26-07 - - $

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "stack.h"
#include "debug.h"

#define DEFAULT_CAPACITY 16

struct stack {
   size_t capacity;
   size_t size;
   stack_item *data;
};

stack *new_stack (void) {
   stack *this = malloc (sizeof (stack));
   assert (this != NULL);
   this->capacity = DEFAULT_CAPACITY;
   this->size = 0;
   this->data = calloc (this->capacity, sizeof (stack_item));
   assert (this->data != NULL);
   return this;
}

void free_stack (stack *this) {
   assert (empty_stack (this));
   free (this->data);
   free (this);
}

static bool full_stack (stack *this) {
   return this->size == this->capacity;
}

static void realloc_stack (stack *this) {
   size_t old_capacity = this->capacity;
   this->capacity *= 2;
   this->data = realloc (this->data, this->capacity);
   assert (this->data != NULL);
   memset (this->data + old_capacity, 0, old_capacity);
}

```

```c

void push_stack (stack *this, stack_item item) {
   if (full_stack (this)) realloc_stack (this);
   DEBUGS ('s', show_stack (this));
   DEBUGF ('s', "item=%p\n", item);
}

stack_item pop_stack (stack *this) {
   assert (! empty_stack (this));
   DEBUGS ('s', show_stack (this));
   STUB (return NULL;)
}

stack_item peek_stack (stack *this, size_t index) {
   assert (index < size_stack (this));
   DEBUGS ('s', show_stack (this));
   STUB (return NULL;)
}

bool empty_stack (stack *this) {
   return size_stack (this) == 0;
}

size_t size_stack (stack *this) {
   return this->size;
}

void show_stack (stack *this) {
   fprintf (stderr, "stack@%p->{%lu,%lu,%p}\n",
            this, this->capacity, this->size, this->data);
}

```

```
// $Id: bigint.c,v 1.15 2015-02-03 18:11:58-08 - - $

#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "bigint.h"
#include "debug.h"

#define MIN_CAPACITY 16

struct bigint {
   size_t capacity;
   size_t size;
   bool negative;
   char *digits;
};

void trim_zeros (bigint *this) {
   while (this->size > 0) {
      size_t digitpos = this->size - 1;
      if (this->digits[digitpos] != 0) break;
      --this->size;
   }
}

bigint *new_bigint (size_t capacity) {
   bigint *this = malloc (sizeof (bigint));
   assert (this != NULL);
   this->capacity = capacity;
   this->size = 0;
   this->negative = false;
   this->digits = calloc (this->capacity, sizeof (char));
   assert (this->digits != NULL);
   DEBUGS ('b', show_bigint (this));
   return this;
}

```

```c

bigint *new_string_bigint (const char *string) {
   assert (string != NULL);
   size_t length = strlen (string);
   bigint *this = new_bigint (length > MIN_CAPACITY
                              ? length : MIN_CAPACITY);
   const char *strdigit = &string[length - 1];
   if (*string == '_') {
      this->negative = true;
      ++string;
   }
   char *thisdigit = this->digits;
   while (strdigit >= string) {
      assert (isdigit (*strdigit));
      *thisdigit++ = *strdigit-- - '0';
   }
   this->size = thisdigit - this->digits;
   trim_zeros (this);
   DEBUGS ('b', show_bigint (this));
   return this;
}

bigint *do_add (bigint *this, bigint *that) {
   DEBUGS ('b', show_bigint (this));
   DEBUGS ('b', show_bigint (that));
   STUB (return NULL);
}

bigint *do_sub (bigint *this, bigint *that) {
   DEBUGS ('b', show_bigint (this));
   DEBUGS ('b', show_bigint (that));
   STUB (return NULL);
}
void free_bigint (bigint *this) {
   free (this->digits);
   free (this);
}

void print_bigint (bigint *this) {
   DEBUGS ('b', show_bigint (this));
}

bigint *add_bigint (bigint *this, bigint *that) {
   DEBUGS ('b', show_bigint (this));
   DEBUGS ('b', show_bigint (that));
   STUB (return NULL);
   return NULL;
}

bigint *sub_bigint (bigint *this, bigint *that) {
   DEBUGS ('b', show_bigint (this));
   DEBUGS ('b', show_bigint (that));
   STUB (return NULL);
   return NULL;
}

```

```

bigint *mul_bigint (bigint *this, bigint *that) {
   DEBUGS ('b', show_bigint (this));
   DEBUGS ('b', show_bigint (that));
   STUB (return NULL);
   return NULL;
}

void show_bigint (bigint *this) {
   fprintf (stderr, "bigint@%p->{%lu,%lu,%c,%p->", this,
            this->capacity, this->size, this->negative ? '-' : '+',
            this->digits);
   for (char *byte = &this->digits[this->size - 1];
        byte >= this->digits; --byte) {
      fprintf (stderr, "%d", *byte);
   }
   fprintf (stderr, "}\n");
}

```

```c
// $Id: token.c,v 1.8 2013-05-16 15:14:31-07 - - $

#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "token.h"
#include "debug.h"

#define INIT_CAPACITY 16

struct token {
   FILE *file;
   size_t capacity;
   size_t size;
   int token;
   char *buffer;
};

token *new_token (FILE *file) {
   token *this = malloc (sizeof (token));
   assert (this != NULL);
   this->file = file;
   this->capacity = INIT_CAPACITY;
   this->buffer = malloc (this->capacity);
   assert (this->buffer != NULL);
   this->buffer[0] = '\0';
   this->size = 0;
   this->token = 0;
   DEBUGS ('t', show_token (this));
   return this;
}

void free_token (token *this) {
   free (this->buffer);
   free (this);
}

char *peek_token (token *this) {
   DEBUGS ('t', show_token (this));
   return this->buffer;
}

```

```c

void ensure_capacity (token *this, size_t capacity) {
   if (capacity > this->capacity) {
      size_t double_capacity = this->capacity * 2;
      this->capacity = capacity > double_capacity
                     ? capacity : double_capacity;
      this->buffer = realloc (this->buffer, this->capacity);
      assert (this->buffer);
   }
}

int scan_token (token *this) {
   this->size = 0;
   this->buffer[this->size] = '\0';
   int result = EOF;
   int nextchar = 0;
   do {
      nextchar = fgetc (this->file);
   } while (isspace (nextchar));
   if (nextchar == EOF) {
      result = EOF;
   }else if (nextchar == '_' || isdigit (nextchar)) {
      do {
         this->buffer[this->size++] = nextchar;
         ensure_capacity (this, this->size + 1);
         nextchar = fgetc (this->file);
      } while (isdigit (nextchar));
      this->buffer[this->size] = '\0';
      int ungetchar = ungetc (nextchar, this->file);
      assert (ungetchar == nextchar);
      result = NUMBER;
   }else {
      result = nextchar;
   }
   DEBUGS ('t', show_token (this));
   return result;
}

void show_token (token *this) {
   fprintf (stderr, "token@%p->{%lu,%lu,%d,%p->\"%s\"}\n",
            this, this->capacity, this->size, this->token,
            this->buffer, this->buffer);
}

```

```
// $Id: main.c,v 1.12 2015-02-03 18:26:19-08 - - $

#include <assert.h>
#include <ctype.h>
#include <libgen.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "bigint.h"
#include "debug.h"
#include "stack.h"
#include "token.h"
#include "yyextern.h"

void do_push (stack *stack, char *numstr) {
   DEBUGF ('m', "stack=%p, numstr=%p=\"%s\"\n", stack, numstr, numstr);
   bigint *bigint = new_string_bigint (numstr);
   push_stack (stack, bigint);
}

void do_binop (stack *stack, bigint_binop binop) {
   DEBUGS ('m', show_stack (stack));
   bigint *right = pop_stack (stack);
   bigint *left = pop_stack (stack);
   bigint *answer = binop (left, right);
   push_stack (stack, answer);
   free_bigint (left);
   free_bigint (right);
}

void do_clear (stack *stack) {
   DEBUGF ('m', "stack=%p\n", stack);
   while (! empty_stack (stack)) {
      bigint *bigint = pop_stack (stack);
      free_bigint (bigint);
   }
}

```

```

void do_print (stack *stack) {
   DEBUGS ('m', show_stack (stack));
   print_bigint (peek_stack (stack, 0));
}

void do_print_all (stack *stack) {
   DEBUGS ('m', show_stack (stack));
   int size = size_stack (stack);
   for (int index = 0; index < size; ++index) {
      print_bigint (peek_stack (stack, index));
   }
}

void unimplemented (int oper) {
   printf ("%s: ", program_name);
   if (isgraph (oper)) printf ("'%c' (0%o)", oper, oper);
                  else printf ("0%o", oper);
   printf (" unimplemented\n");
}

void scan_options (int argc, char **argv) {
   opterr = false;
   for (;;) {
      int option = getopt (argc, argv, "@:");
      if (option == EOF) break;
      switch (option) {
         case '@': set_debug_flags (optarg);
                   break;
         default : printf ("%s: -%c: invalid option\n",
                           program_name, optopt);
                   break;
      }
   }
}

int main (int argc, char **argv) {
   program_name = basename (argv[0]);
   scan_options (argc, argv);
   stack *stack = new_stack ();
   bool quit = false;
   yy_flex_debug = false;
   while (! quit) {
      int token = yylex();
      if (token == YYEOF) break;
      switch (token) {
         case NUMBER: do_push (stack, yytext); break;
         case '+': do_binop (stack, add_bigint); break;
         case '-': do_binop (stack, sub_bigint); break;
         case '*': do_binop (stack, mul_bigint); break;
         case 'c': do_clear (stack); break;
         case 'f': do_print_all (stack); break;
         case 'p': do_print (stack); break;
         case 'q': quit = true; break;
         default: unimplemented (token); break;
      }
   }
   yycleanup();
   DEBUGF ('m', "EXIT %d\n", exit_status);
   return EXIT_SUCCESS;
}
```

```
%{
// $Id: scanner.l,v 1.2 2015-02-03 18:22:49-08 - - $

#include "yyextern.h"
#define YY_NO_INPUT

%}

%option 8bit
%option debug
%option interactive
%option nodefault
%option nounput
%option noyywrap

WHITESPACE  ([ \t\n])
NUMBER      (_?[0-9]*)
OTHER       (.)

%%

[ \t\n]+  { }
_?[0-9]*  { return YYNUMBER; }
.         { return *yytext; }

%%

void yycleanup (void) {
   yy_delete_buffer (YY_CURRENT_BUFFER);
}

```

```
# $Id: Makefile,v 1.11 2015-02-03 18:27:45-08 - - $

MKFILE    = Makefile
DEPSFILE  = ${MKFILE}.deps
NOINCLUDE = ci clean spotless
NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
GMAKE     = gmake --no-print-directory

GCC       = gcc -g -O0 -Wall -Wextra -std=gnu11
MKDEPS    = gcc -MM

CSOURCE   = debug.c stack.c bigint.c token.c main.c
LSOURCE   = scanner.l
CHEADER   = debug.h stack.h bigint.h token.h yyextern.h
OBJECTS   = ${CSOURCE:.c=.o} ${LSOURCE:.l=.o}
EXECBIN   = mydc
SOURCES   = ${CHEADER} ${CSOURCE} ${LSOURCE} ${MKFILE}
LISTING   = Listing.ps

all : ${EXECBIN}

${EXECBIN} : ${OBJECTS}
        ${GCC} -o $@ ${OBJECTS}

%.o : %.c
        ${GCC} -c $<

scanner.c : scanner.l
        flex -oscanner.c scanner.l

ci : ${SOURCES}
        cid + ${SOURCES}
        checksource ${SOURCES}

lis : ${SOURCES} ${DEPSFILE}
        mkpspdf ${LISTING} ${SOURCES} ${DEPSFILE}

clean :
        - rm ${LSOURCE:.l=.c} ${OBJECTS} ${DEPSFILE} core

spotless : clean
        - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf}

deps : ${CSOURCE} ${CHEADER}
        @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
        ${MKDEPS} ${CSOURCE} >>${DEPSFILE}

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

again :
        ${GMAKE} spotless deps ci all lis

ifeq "${NEEDINCL}" ""
include ${DEPSFILE}
endif

```

```
# Makefile.deps created Tue Feb  9 13:20:05 PST 2016
debug.o: debug.c debug.h yyextern.h
stack.o: stack.c stack.h bigint.h debug.h
bigint.o: bigint.c bigint.h debug.h
token.o: token.c token.h debug.h
main.o: main.c bigint.h debug.h stack.h token.h yyextern.h
```