```
// $Id: debugf.h,v 1.1 2012-03-05 19:43:01-08 - - $

#ifndef __DEBUGF_H__
#define __DEBUGF_H__

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

//
// Tell debugf what program is running.
//
void set_execname (char *name);

//
// Support for stub messages.
//
#define STUBPRINTF(...) \
        __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
void __stubprintf (char *file, int line, const char *func,
                   char *format, ...);

//
// Debugging utility.
//

void set_debugflags (char *flags);
   //
   // Sets a string of debug flags to be used by DEBUGF statements.
   // Uses the address of the string, and does not copy it, so it
   // must not be dangling.  If a particular debug flag has been set,
   // messages are printed.  The format is identical to printf format.
   // The flag "@" turns on all flags.
   //

#ifdef NDEBUG
#define DEBUGF(FLAG,...) // DEBUG (FLAG, __VA_ARGS__)
#else
#define DEBUGF(FLAG,...) \
        __debugprintf (FLAG, __FILE__, __LINE__, __VA_ARGS__)
void __debugprintf (char flag, char *file, int line,
                    char *format, ...);
#endif

#endif

```

```
// $Id: stack.h,v 1.1 2012-03-05 19:43:01-08 - - $

#ifndef __STACK_H__
#define __STACK_H__

#include <stdbool.h>

typedef struct stack *stack_ref;
typedef struct bigint *stack_item;

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

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

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

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

//
// Peek into the stack and return a selected stack_item, with
// item 0 being the element at the top.
// and item length_stack - 1 being the element at the bottom.
// Precondition: 0 <= index && index < length_stack.
//
stack_item peek_stack (stack_ref, int index);

//
// Indicate whether the stack is empty or not.
//
bool is_empty_stack (stack_ref);

//
// Return a count of the number of items on the stack.
//
int length_stack (stack_ref);

//
// Indixate whether or not a pointer points at a stack.
//
bool is_stack (stack_ref);

#endif

```

```c
// $Id: bigint.h,v 1.2 2012-03-05 20:19:38-08 - - $

#ifndef __BIGINT_H__
#define __BIGINT_H__

#include <stdbool.h>

typedef struct bigint *bigint_ref;

typedef bigint_ref (*bigint_binop) (bigint_ref, bigint_ref);

bigint_ref new_bigint (size_t length);

bigint_ref from_string_bigint (char *string);

void free_bigint (bigint_ref);

void print_bigint (bigint_ref);

bigint_ref add_bigint (bigint_ref left, bigint_ref right);

bigint_ref sub_bigint (bigint_ref left, bigint_ref right);

bigint_ref mul_bigint (bigint_ref left, bigint_ref right);

bool is_bigint (bigint_ref);

#endif

```

```
// $Id: token.h,v 1.2 2012-03-05 20:19:38-08 - - $

#ifndef __TOKEN_H__
#define __TOKEN_H__

#define NUMBER 256

typedef struct token *token_ref;

token_ref new_token (FILE*);

int scan_token (token_ref);

char *peek_token (token_ref);

#endif

```

```c
// $Id: debugf.c,v 1.1 2012-03-05 19:43:01-08 - - $

#include <errno.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "debugf.h"

static char *debugflags = "";
static bool alldebugflags = false;
static char *execname = NULL;

void set_execname (char *name) {
   execname = name;
}

void print_execname (FILE *out) {
   if (execname != NULL) fprintf (out, "%s: ", execname);
}

void __stubprintf (char *filename, int line, const char *func,
                   char *format, ...) {
   va_list args;
   fflush (NULL);
   print_execname (stdout);
   fprintf (stdout, "STUB: %s[%d] %s:\n", filename, line, func);
   va_start (args, format);
   vfprintf (stdout, format, args);
   va_end (args);
   fflush (NULL);
}

void set_debugflags (char *flags) {
   debugflags = flags;
   if (strchr (debugflags, '@') != NULL) alldebugflags = true;
   DEBUGF ('a', "Debugflags = \"%s\"\n", debugflags);
}

void __debugprintf (char flag, char *file, int line,
                    char *format, ...) {
   va_list args;
   if (alldebugflags || strchr (debugflags, flag) != NULL) {
      fflush (NULL);
      print_execname (stderr);
      fprintf (stderr, "DEBUGF(%c): %s[%d]:\n",
                flag, file, line);
      va_start (args, format);
      vfprintf (stderr, format, args);
      va_end (args);
      fflush (NULL);
   }
}

```

```c
// $Id: stack.c,v 1.3 2012-03-06 16:27:19-08 - - $

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

#include "stack.h"

typedef struct stack_node *stack_node_ref;

static char *stack_tag = "struct stack";
static char *stack_node_tag = "struct stack_node";

struct stack {
   char *tag;
   stack_node_ref top;
};

struct stack_node {
   char *tag;
   stack_item item;
   stack_node_ref link;
};

stack_ref new_stack (void) {
   return NULL;
}

void free_stack (stack_ref stack) {
   assert (is_stack (stack));
   assert (is_empty_stack (stack));
}

void push_stack (stack_ref stack, stack_item item) {
   assert (is_stack (stack));
}

stack_item pop_stack (stack_ref stack) {
   assert (is_stack (stack));
   assert (! is_empty_stack (stack));
   return NULL;
}

stack_item peek_stack (stack_ref stack, int index) {
   assert (is_stack (stack));
   assert (index >= 0);
   assert (index < length_stack (stack));
   return NULL;
}

bool is_empty_stack (stack_ref stack) {
   assert (is_stack (stack));
   return false;
}

int length_stack (stack_ref stack) {
   assert (is_stack (stack));
   return 0;
}

bool is_stack (stack_ref stack) {
   return false;
}

```

```c
// $Id: bigint.c,v 1.3 2012-03-06 16:27:19-08 - - $

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

#include "bigint.h"

static char *bigint_tag = "struct bigint";

struct bigint {
   char *tag;
   bool is_negative;
   size_t length;
   size_t digits;
   char *buffer;
};

static void trim_zeros (bigint_ref bigint) {
   while (bigint->digits > 0) {
      size_t digitpos = bigint->digits - 1;
      if (bigint->buffer[digitpos] != 0) break;
      --bigint->digits;
   }
}

static bigint_ref do_add (bigint_ref left, bigint_ref right) {
}

static bigint_ref do_sub (bigint_ref left, bigint_ref right) {
}

```

```c

bigint_ref new_bigint (size_t length) {
   bigint_ref bigint = malloc (sizeof (struct bigint));
   assert (bigint != NULL);
   bigint->tag = bigint_tag;
   bigint->is_negative = false;
   bigint->length = length;
   bigint->digits = 0;
   bigint->buffer = calloc (length, sizeof (char));
   assert (bigint->buffer != NULL);
   return bigint;
}

bigint_ref from_string_bigint (char *string) {
   assert (string != NULL);
   size_t length = strlen (string);
   bigint_ref bigint = new_bigint (length);
   if (*string == '_') {
      bigint->is_negative = true;
      ++string;
      --length;
   }
   int index = 0;
   while (--length > 0) {
      // LINTED (assignment of 32-bit integer to 8-bit integer)
      char digit = string[length] - '0';
      assert (0 <= digit && digit <= 9);
      bigint->buffer[index++] = digit;
   }
   trim_zeros (bigint);
   return NULL;
}

```

```

void free_bigint (bigint_ref bigint) {
   assert (is_bigint (bigint));
}

void print_bigint (bigint_ref bigint) {
   assert (is_bigint (bigint));
}

bigint_ref add_bigint (bigint_ref left, bigint_ref right) {
   assert (is_bigint (left));
   assert (is_bigint (right));
   return NULL;
}

bigint_ref sub_bigint (bigint_ref left, bigint_ref right) {
   assert (is_bigint (left));
   assert (is_bigint (right));
   return NULL;
}

bigint_ref mul_bigint (bigint_ref left, bigint_ref right) {
   assert (is_bigint (left));
   assert (is_bigint (right));
   return NULL;
}

bool is_bigint (bigint_ref bigint) {
   return false;
}

```

```c
// $id$

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

#include "token.h"
#include "debugf.h"

#define BUFFER_LENGTH 16

struct token {
   FILE *file;
   int token;
   size_t length;
   size_t chars;
   char *buffer;
};

token_ref new_token (FILE *file) {
   token_ref new = malloc (sizeof (struct token));
   assert (new != NULL);
   new->file = file;
   new->token = 0;
   new->length = BUFFER_LENGTH;
   new->buffer = malloc (new->length);
   assert (new->buffer != NULL);
   new->buffer[0] = ’\0’;
   new->chars = 0;
   DEBUGF (’t’, "token_ref=%p\n", new);
   return new;
}

char *peek_token (token_ref token) {
   DEBUGF (’t’, "peek %p [%d] \"%s\"\n", token, token->chars,
           token->buffer);
   return token->buffer;
}

```

```c

int scan_token (token_ref token) {
   token->chars = 0;
   token->buffer[token->chars] = ’\0’;
   int result = EOF;
   int nextchar = 0;
   do {
      nextchar = fgetc (token->file);
   } while (isspace (nextchar));
   if (nextchar == EOF) {
      result = EOF;
   }else if (nextchar == ’_’ || isdigit (nextchar)) {
      do {
         // LINTED (assignment of 32-bit integer to 8-bit integer)
         token->buffer[token->chars++] = nextchar;
         if (token->chars == token->length) {
            token->length *= 2;
            token->buffer = realloc (token->buffer, token->length);
            assert (token->buffer);
         }
         nextchar = fgetc (token->file);
      } while (isdigit (nextchar));
      token->buffer[token->chars] = ’\0’;
      int ungetchar = ungetc (nextchar, token->file);
      assert (ungetchar == nextchar);
      result = NUMBER;
   }else {
      result = nextchar;
   }
   DEBUGF (’t’, "scan %p [%d] \"%s\" %d\n", token, token->chars,
           token->buffer, result);
   return result;
}

```

```
// $Id: main.c,v 1.3 2012-03-06 16:27:19-08 - - $

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

#include "bigint.h"
#include "debugf.h"
#include "stack.h"
#include "token.h"

char *execname = NULL;

#define DO_NOTHING(X) {DEBUGF ('s', ""); return X; }

bool not_enough (stack_ref stack, int enough) {
   DO_NOTHING(false);
   if (length_stack (stack) >= enough) return true;
   printf ("%s: stack empty\n", execname);
   return false;
}

void do_push (stack_ref stack, char *yytext) {
   DO_NOTHING();
   bigint_ref bigint = from_string_bigint (yytext);
   push_stack (stack, bigint);
}

void do_binop (stack_ref stack, bigint_binop binop) {
   DO_NOTHING();
   if (not_enough (stack, 2)) return;
   bigint_ref right = pop_stack (stack);
   bigint_ref left = pop_stack (stack);
   bigint_ref answer = binop (left, right);
   push_stack (stack, answer);
   free_bigint (left);
   free_bigint (right);
}

void do_clear (stack_ref stack) {
   DO_NOTHING();
   while (! is_empty_stack (stack)) {
      bigint_ref bigint = pop_stack (stack);
      free_bigint (bigint);
   }
}
```

```

void do_print (stack_ref stack) {
   DO_NOTHING();
   if (not_enough (stack, 1)) return;
   print_bigint (peek_stack (stack, 0));
}

void do_print_all (stack_ref stack) {
   DO_NOTHING();
   int length = length_stack (stack);
   for (int index = 0; index < length; ++index) {
      print_bigint (peek_stack (stack, index));
   }
}

void unimplemented (int oper) {
   printf ("%s: ", execname);
   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, "y@:");
      if (option == EOF) break;
      switch (option) {
         case '@': set_debugflags (optarg);
                   break;
         default : printf ("%s: -%c: invalid option\n",
                           execname, optopt);
                   break;
      }
   }
}

int main (int argc, char **argv) {
   execname = basename (argv[0]);
   set_execname (execname);
   scan_options (argc, argv);
   stack_ref stack = new_stack ();
   token_ref scanner = new_token (stdin);
   for (;;) {
      int token = scan_token (scanner);
      if (token == EOF) break;
      switch (token) {
         case NUMBER: do_push (stack, peek_token (scanner)); 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;
         default: unimplemented (token); break;
      }
   }
   return EXIT_SUCCESS;
}
```

```
# $Id: Makefile,v 1.3 2012-03-06 18:41:37-08 - - $

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=gnu99
MKDEPS    = gcc -MM
LINT      = lint -Xa -fd -m -u -x -errchk=%all

CSOURCE   = debugf.c stack.c bigint.c token.c main.c
CHEADER   = debugf.h stack.h bigint.h token.h
OBJECTS   = ${CSOURCE:.c=.o}
EXECBIN   = a5dc
SUBMITS   = ${CHEADER} ${CSOURCE} ${MKFILE}
SOURCES   = ${SUBMITS}
LISTING   = Listing.code.ps
PROJECT   = cmps012b-wm.w12 asg5

all : ${EXECBIN}

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

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

lint : ${CSOURCE}
        ${LINT} ${CSOURCE}
        checksource ${SUBMITS}

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

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

clean :
        - rm ${OBJECTS} ${DEPSFILE} core ${EXECBIN}.errs

spotless : clean
        - rm ${EXECBIN}

submit : ${SUBMITS}
        submit ${PROJECT} ${SUBMITS}

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

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

again :
        ${GMAKE} spotless deps ci lint all lis

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

```

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