```
// $Id: bigint.h,v 1.2 2012-04-10 21:02:20-07 - - $

#ifndef __BIGINT_H__
#define __BIGINT_H__

#include <exception>
#include <iostream>
#include <utility>

using namespace std;

#include "trace.h"

class bigint;
typedef pair <bigint, bigint> bigpair;

//
// Operators with a left side of int.
//
bigint operator+ (int left, const bigint &that);
bigint operator- (int left, const bigint &that);
bigint operator* (int left, const bigint &that);
bigint operator/ (int left, const bigint &that);
bigint operator% (int left, const bigint &that);
bool operator== (int left, const bigint &that);
bool operator!= (int left, const bigint &that);
bool operator<  (int left, const bigint &that);
bool operator<= (int left, const bigint &that);
bool operator>  (int left, const bigint &that);
bool operator>= (int left, const bigint &that);

```

```

//
// Define class bigint
//
class bigint {
      friend ostream &operator<< (ostream &, const bigint &);
   private:
      int small_value;
      bigpair div_rem (const bigint &that) const;
      int compare (const bigint &that) const;
      int abscompare (const bigint &that) const;
      bigint mul_by_2 ();
   public:
      //
      // Override implicit members.
      //
      bigint ();
      bigint (const bigint &that);
      bigint &operator= (const bigint &that);
      ~bigint ();
      //
      // Extra ctors to make bigints.
      //
      bigint (const int that);
      bigint (const string &that);
      //
      // Basic add/sub operators.
      //
      bigint operator+ (const bigint &that) const;
      bigint operator- (const bigint &that) const;
      bigint operator- () const;
      int smallint () const;
      //
      // Extended operators implemented with add/sub.
      //
      bigint operator* (const bigint &that) const;
      bigint operator/ (const bigint &that) const;
      bigint operator% (const bigint &that) const;
      bigint pow (const bigint &that) const;
      //
      // Comparison operators.
      //
      bool operator== (const bigint &that) const;
      bool operator!= (const bigint &that) const;
      bool operator<  (const bigint &that) const;
      bool operator<= (const bigint &that) const;
      bool operator>  (const bigint &that) const;
      bool operator>= (const bigint &that) const;
};

#endif

```

```
// $Id: scanner.h,v 1.1 2011-01-18 22:17:09-08 - - $

#ifndef __SCANNER_H__
#define __SCANNER_H__

#include <iostream>
#include <utility>

using namespace std;

#include "trace.h"

enum terminal_symbol {NUMBER, OPERATOR, SCANEOF};
struct token_t {
   terminal_symbol symbol;
   string lexinfo;
};

class scanner {
   private:
      bool seen_eof;
      char lookahead;
      void advance();
   public:
      scanner();
      token_t scan ();
};

ostream &operator<< (ostream &, const terminal_symbol &);
ostream &operator<< (ostream &, const token_t &);

#endif

```

```
// $Id: trace.h,v 1.1 2011-01-18 22:17:09-08 - - $

#ifndef __TRACE_H__
#define __TRACE_H__

#include <iostream>
#include <string>
#include <vector>

using namespace std;

//
// traceflags -
//    static class for maintaining global trace flags, each indicated
//    by a single character.
// setflags -
//    Takes a string argument, and sets a flag for each char in the
//    string.  As a special case, '@', sets all flags.
// getflag -
//    Used by the TRACE macro to check to see if a flag has been set.
//    Not to be called by user code.
//

class traceflags {
   private:
      static vector<char> flags;
   public:
      static void setflags (const string &optflags);
      static bool getflag (char flag);
};

//
// TRACE -
//    Macro which expands into trace code.  First argument is a
//    trace flag char, second argument is output code that can
//    be sandwiched between <<.  Beware of operator precedence.
//    Example:
//       TRACE ('u', "foo = " << foo);
//    will print two words and a newline if flag 'u' is  on.
//    Traces are preceded by filename, line number, and function.
//

#define TRACE(FLAG,CODE) { \
           if (traceflags::getflag (FLAG)) { \
              cerr << __FILE__ << ":" << __LINE__ << ":" \
                   << __func__ << ":" << endl; \
              cerr << CODE << endl; \
           } \
        }

#endif

```

```
// $Id: util.h,v 1.2 2012-04-10 21:02:20-07 - - $

//
// util -
//    A utility class to provide various services not conveniently
//    included in other modules.
//

#ifndef __UTIL_H__
#define __UTIL_H__

#include <iostream>
#include <vector>

#ifdef __GNUC__
#include <stdexcept>
#endif

using namespace std;

#include "trace.h"

//
// ydc_exn -
//    Indicate a problem where processing should be abandoned and
//    the main function should take control.
//

class ydc_exn: public runtime_error {
   public:
      explicit ydc_exn (const string &what);
};

//
// octal -
//    Convert integer to octal string.
//

const string octal (int decimal);

```

```

//
// sys_info -
//    Keep track of execname and exit status.  Must be initialized
//    as the first thing done inside main.  Main should call:
//       sys_info::set_execname (argv[0]);
//    before anything else.
//

class sys_info {
   private:
      static string execname;
      static int exit_status;
   public:
      static void set_execname (const string &argv0);
      static const string &get_execname () {return execname; }
      static void set_status (int status) {exit_status = status; }
      static int get_status () {return exit_status; }
};

//
// complain -
//    Used for starting error messages.  Sets the exit status to
//    EXIT_FAILURE, writes the program name to cerr, and then
//    returns the cerr ostream.  Example:
//       complain() << filename << ": some problem" << endl;
//

ostream &complain();

//
// operator<< (vector) -
//    An overloaded template operator which allows vectors to be
//    printed out as a single operator, each element separated from
//    the next with spaces.  The item_t must have an output operator
//    defined for it.
//

template <typename item_t>
ostream &operator<< (ostream &out, const vector<item_t> &vec);

#endif

```

```
// $Id: iterstack.h,v 1.6 2012-04-12 19:32:54-07 - - $

//
// The class std::stack does not provide an iterator, which is
// needed for this class.  So, like std::stack, class iterstack
// is implemented on top of a container.
//
// We use private inheritance because we want to restrict
// operations only to those few that are approved.  All functions
// are merely inherited from the container, with only ones needed
// being exported as public.
//
// No implementation file is needed because all functions are
// inherited, and the convenience functions that are added are
// trivial, and so can be inline.
//

#ifndef __ITERSTACK_H__
#define __ITERSTACK_H__

#include <list>

using namespace std;

template <typename value_type>
class iterstack: private list<value_type> {
   public:
      using list<value_type>::const_reference;
      using list<value_type>::const_reverse_iterator;
      using list<value_type>::push_back;
      using list<value_type>::pop_back;
      using list<value_type>::clear;
      using list<value_type>::back;
      using list<value_type>::size;
      using list<value_type>::empty;
      using list<value_type>::rbegin;
      using list<value_type>::rend;
      inline void push (const value_type &value) { push_back (value); }
      inline void pop() { pop_back(); }
      inline value_type& top() { return back(); }
      inline const value_type& top() const { return back(); }
};

#endif

```

```
// $Id: bigint.cpp,v 1.3 2012-04-10 21:02:20-07 - - $

#include <cstdlib>
#include <exception>
#include <limits>
#include <stack>
#include <stdexcept>

using namespace std;

#include "bigint.h"
#include "trace.h"

#define CDTOR_TRACE TRACE ('~', this << " -> " << small_value)

bigint::bigint (): small_value (0) {
   CDTOR_TRACE;
}

bigint::bigint (const bigint &that): small_value (that.small_value) {
   *this = that;
   CDTOR_TRACE;
}

bigint &bigint::operator= (const bigint &that) {
   if (this == &that) return *this;
   this->small_value = that.small_value;
   return *this;
}

bigint::~bigint() {
   CDTOR_TRACE;
}

bigint::bigint (int that): small_value (that) {
   CDTOR_TRACE;
}

bigint::bigint (const string &that) {
   string::const_iterator itor = that.begin();
   string::const_iterator end = that.end();
   bool isnegative = false;
   if (*itor == '_') {isnegative = true; ++itor; }
   int newval = 0;
   for (; itor != end; ++itor) newval = newval * 10 + *itor - '0';
   small_value = isnegative ? - newval : + newval;
   CDTOR_TRACE;
}

```

```

bigint bigint::operator+ (const bigint &that) const {
   return this->small_value + that.small_value;
}

bigint bigint::operator- (const bigint &that) const {
   return this->small_value - that.small_value;
}

bigint bigint::operator- () const {
   return -small_value;
}

int bigint::compare (const bigint &that) const {
   return this->small_value < that.small_value ? -1
        : this->small_value > that.small_value ? +1 : 0;
}

int bigint::abscompare (const bigint &that) const {
   return abs (this->small_value) < abs (that.small_value) ? -1
        : abs (this->small_value) > abs (that.small_value) ? +1 : 0;
}

int bigint::smallint () const {
   if (*this < numeric_limits<int>::min()
    || *this > numeric_limits<int>::max())
               throw range_error ("smallint: out of range");
   return small_value;
}

bigint bigint::mul_by_2 () {
   return this->small_value *= 2;
}

static bigpair popstack (stack <bigpair> &egyptstack) {
   bigpair result = egyptstack.top ();
   egyptstack.pop();
   return result;
}

```

```

//
// Ancient Egyptian multiplication algorithm.
//
bigint bigint::operator* (const bigint &that) const {
   bigint top = that;
   bigint count = 1;
   TRACE ('*', *this << " * " << that);
   stack <bigpair> egyptstack;
   bigint a; // junk code -- delete
   bigint b; // junk code -- delete
   egyptstack.push (pair<bigint, bigint> (a, b)); // junk code -- delete
   popstack (egyptstack); // junk to suppress a warning
   bigint result = 0;
   if ((*this < 0) != (that < 0)) result = - result;
   return result;
}

//
// Ancient Egyptian division algorithm.
//
bigpair bigint::div_rem (const bigint &that) const {
   if (that == 0) throw range_error ("divide by 0");
   bigint count = 1;
   bigint top = abs (that.small_value);
   TRACE ('/', *this << " /% " << that);
   stack <bigpair> egyptstack;
   bigint quotient = 0;
   bigint remainder = abs (this->small_value);
   return bigpair (quotient, remainder);
}

bigint bigint::operator/ (const bigint &that) const {
   return div_rem (that).first;
}

bigint bigint::operator% (const bigint &that) const {
   return div_rem (that).second;
}

```

```

#define TRACE_POW TRACE ('^', "result: " << result \
                  << ", base: " << base << ", expt: " << expt);
bigint bigint::pow (const bigint &that) const {
   bigint base = *this;
   if (that > 999) throw range_error ("exp too big");
   int expt = that.smallint();
   bigint result = 1;
   TRACE_POW;
   if (expt < 0) {
      base = 1 / base;
      expt = - expt;
   }
   while (expt > 0) {
      TRACE_POW;
      if (expt & 1) { //odd
         result = result * base;
         --expt;
      }else { //even
         base = base * base;
         expt /= 2;
      }
   }
   TRACE_POW;
   return result;
}

//
// Macros can make repetitive code easier.
//

#define COMPARE(OPER) \
   bool bigint::operator OPER (const bigint &that) const { \
      return compare (that) OPER 0; \
   }
COMPARE (==)
COMPARE (!=)
COMPARE (< )
COMPARE (<=)
COMPARE (> )
COMPARE (>=)

#define INT_LEFT(RESULT,OPER) \
   RESULT operator OPER (int left, const bigint &that) { \
      return bigint (left) OPER that; \
   }
INT_LEFT (bigint, +)
INT_LEFT (bigint, -)
INT_LEFT (bigint, *)
INT_LEFT (bigint, /)
INT_LEFT (bigint, %)
INT_LEFT (bool, ==)
INT_LEFT (bool, !=)
INT_LEFT (bool, < )
INT_LEFT (bool, <=)
INT_LEFT (bool, > )
INT_LEFT (bool, >=)

ostream &operator<< (ostream &out, const bigint &that) {
   out << that.small_value;
   return out;
}

```

```cpp
// $Id: scanner.cpp,v 1.2 2012-04-10 20:46:21-07 - - $

#include <iostream>
#include <locale>

using namespace std;

#include "scanner.h"
#include "trace.h"

scanner::scanner () {
   seen_eof = false;
   advance();
}

void scanner::advance () {
   if (! seen_eof) {
      cin.get (lookahead);
      if (cin.eof()) seen_eof = true;
   }
}

token_t scanner::scan() {
   token_t result;
   while (!seen_eof && isspace (lookahead)) advance();
   if (seen_eof) {
      result.symbol = SCANEOF;
   }else if (lookahead == '_' || isdigit (lookahead)) {
      result.symbol = NUMBER;
      do {
         result.lexinfo += lookahead;
         advance();
      }while (!seen_eof && isdigit (lookahead));
   }else {
      result.symbol = OPERATOR;
      result.lexinfo += lookahead;
      advance();
   }
   TRACE ('S', result);
   return result;
}

ostream &operator<< (ostream &out, const terminal_symbol &symbol) {
   switch (symbol) {
      case NUMBER  : out << "NUMBER"  ; break;
      case OPERATOR: out << "OPERATOR"; break;
      case SCANEOF : out << "SCANEOF" ; break;
   }
   return out;
}

ostream &operator<< (ostream &out, const token_t &token) {
   out << token.symbol << ": \"" << token.lexinfo << "\"";
   return out;
}

```

```
// $Id: trace.cpp,v 1.1 2012-04-10 20:43:21-07 - - $

#include <climits>
#include <vector>

using namespace std;

#include "trace.h"

//
// ** BUG IN STL ** BUG IN STL **
// We should use vector<bool> instead of vector<char>,
// but vector<bool> has a bug:
// http://forums.sun.com/thread.jspa?threadID=5277939
// Static linking works, but doubles the size of the executable
// image.
// ** BUG IN STL ** BUG IN STL **
//

typedef vector<char> boolvec;
boolvec traceflags::flags (UCHAR_MAX + 1, false);
const boolvec trueflags (UCHAR_MAX + 1, true);

void traceflags::setflags (const string &optflags) {
   string::const_iterator itor = optflags.begin();
   string::const_iterator end = optflags.end();
   for (; itor != end; ++itor) {
      if (*itor == '@') {
         flags = trueflags;
      }else {
         flags[*itor] = true;
      }
   }
   // Note that TRACE can trace setflags.
   TRACE ('t',  "optflags = " << optflags);
}

//
// getflag -
//    Check to see if a certain flag is on.
//

bool traceflags::getflag (char flag) {
   // WARNING: Don't TRACE this function or the stack will blow up.
   bool result = flags[flag];
   return result;
}

```

```
// $Id: util.cpp,v 1.1 2012-04-10 20:43:21-07 - - $

#include <cstdlib>
#include <sstream>

using namespace std;

#include "util.h"

ydc_exn::ydc_exn (const string &what): runtime_error (what) {
}

const string octal (int decimal) {
   ostringstream ostring;
   ostring.setf (ios::oct);
   ostring << decimal;
   return ostring.str ();
}

int sys_info::exit_status = EXIT_SUCCESS;
string sys_info::execname; // Must be initialized from main().

void sys_info::set_execname (const string &argv0) {
   execname = argv0;
   cout << boolalpha;
   cerr << boolalpha;
   TRACE ('Y', "execname = " << execname);
}

ostream &complain() {
   sys_info::set_status (EXIT_FAILURE);
   cerr << sys_info::get_execname () << ": ";
   return cerr;
}

template <typename item_t>
ostream &operator<< (ostream &out, const vector<item_t> &vec) {
   typename vector<item_t>::const_iterator itor = vec.begin();
   typename vector<item_t>::const_iterator end = vec.end();

   // If the vector is empty, do nothing.
   if (itor != end) {
      // Print out the first element without a space.
      out << *itor++;
      // Print out the rest of the elements each preceded by a space.
      while (itor != end) out << " " << *itor++;
   }
   return out;
}

```

```
// $Id: main.cpp,v 1.8 2012-04-12 19:31:01-07 - - $

#include <deque>
#include <exception>
#include <map>
#include <iostream>
#include <utility>

using namespace std;

#include "bigint.h"
#include "iterstack.h"
#include "util.h"
#include "scanner.h"
#include "trace.h"

typedef iterstack<bigint> bigint_stack;

#define DO_BINOP(FN_NAME,TFLAG,OPER) \
   void FN_NAME (bigint_stack &stack) { \
      bigint right = stack.top(); \
      stack.pop(); \
      TRACE (TFLAG, "right = " << right); \
      bigint left = stack.top(); \
      stack.pop(); \
      TRACE (TFLAG, "left = " << left); \
      bigint result = left OPER (right); \
      TRACE (TFLAG, "result = " << result); \
      stack.push (result); \
   }
DO_BINOP(do_add, '+', +   )
DO_BINOP(do_sub, '-', -   )
DO_BINOP(do_mul, '*', *   )
DO_BINOP(do_div, '/', /   )
DO_BINOP(do_rem, '%', %   )
DO_BINOP(do_pow, '^', .pow)

void do_clear (bigint_stack &stack) {
   TRACE ('c', "");
   stack.clear();
}

void do_dup (bigint_stack &stack) {
   bigint top = stack.top();
   TRACE ('d', top);
   stack.push (top);
}

void do_printall (bigint_stack &stack) {
   bigint_stack::const_reverse_iterator itor = stack.rbegin();
   bigint_stack::const_reverse_iterator end = stack.rend();
   for (; itor != end; ++itor) cout << *itor << endl;
}

void do_print (bigint_stack &stack) {
   cout << stack.top() << endl;
}

void do_debug (bigint_stack &stack) {
   (void) stack; // SUPPRESS: warning: unused parameter 'stack'
   cout << "Y not implemented" << endl;
}

```

```

class ydc_quit: public exception {};
void do_quit (bigint_stack &stack) {
   (void) stack; // SUPPRESS: warning: unused parameter ’stack’
   throw ydc_quit ();
}

typedef void (*function) (bigint_stack&);
typedef map <string, function> fnmap;
fnmap load_fn () {
   fnmap functions;
   functions["+"] = do_add;
   functions["-"] = do_sub;
   functions["*"] = do_mul;
   functions["/"] = do_div;
   functions["%"] = do_rem;
   functions["^"] = do_pow;
   functions["Y"] = do_debug;
   functions["c"] = do_clear;
   functions["d"] = do_dup;
   functions["f"] = do_printall;
   functions["p"] = do_print;
   functions["q"] = do_quit;;
   return functions;
}

//
// scan_options
//    Options analysis:  The only option is -Dflags.
//

void scan_options (int argc, char **argv) {
   opterr = 0;
   for (;;) {
      int option = getopt (argc, argv, "@:");
      if (option == EOF) break;
      switch (option) {
         case ’@’:
            traceflags::setflags (optarg);
            break;
         default:
            complain() << "-" << (char) optopt << ": invalid option"
                       << endl;
            break;
      }
   }
   if (optind < argc) {
      complain() << "operand not permitted" << endl;
   }
}

```

```

int main (int argc, char **argv) {
   sys_info::set_execname (argv[0]);
   scan_options (argc, argv);
   fnmap functions = load_fn();
   bigint_stack operand_stack;
   scanner input;
   try {
      for (;;) {
         try {
            token_t token = input.scan();
            if (token.symbol == SCANEOF) break;
            switch (token.symbol) {
               case NUMBER:
                  operand_stack.push (token.lexinfo);
                  break;
               case OPERATOR: {
                  function fn = functions[token.lexinfo];
                  if (fn == NULL) {
                     throw ydc_exn (octal (token.lexinfo[0])
                                    + " is unimplemented");
                  }
                  fn (operand_stack);
                  break;
                  }
               default:
                  break;
            }
         }catch (ydc_exn &exn) {
            cout << exn.what() << endl;
         }
      }
   }catch (ydc_quit &) {
      // Intentionally left empty.
   }
   return sys_info::get_status ();
}

```

```
# $Id: Makefile,v 1.4 2012-04-10 20:43:21-07 - - $

MKFILE      = Makefile
DEPFILE     = ${MKFILE}.dep
NOINCL      = ci clean spotless
NEEDINCL    = ${filter ${NOINCL}, ${MAKECMDGOALS}}
GMAKE       = ${MAKE} --no-print-directory

COMPILECPP  = g++ -g -O0 -Wall -Wextra -Werror
MAKEDEPCPP  = g++ -MM

CPPHEADER   = bigint.h   scanner.h   trace.h   util.h   iterstack.h
CPPSOURCE   = bigint.cpp scanner.cpp trace.cpp util.cpp main.cpp
EXECBIN     = ydc
OBJECTS     = ${CPPSOURCE:.cpp=.o}
OTHERS      = ${MKFILE} README
ALLSOURCES  = ${CPPHEADER} ${CPPSOURCE} ${OTHERS}
LISTING     = Listing.code.ps
CLASS       = cmps109-wm.s12
PROJECT     = asg2

all : ${EXECBIN}
        - checksource ${ALLSOURCES}

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

%.o : %.cpp
        cid + $<
        ${COMPILECPP} -c $<

ci : ${ALLSOURCES}
        - checksource ${ALLSOURCES}
        cid + ${ALLSOURCES}

lis : ${ALLSOURCES}
        mkpspdf ${LISTING} ${ALLSOURCES} ${DEPFILE}

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

spotless : clean
        - rm ${EXECBIN} ${LISTING}

submit : ${ALLSOURCES}
        - checksource ${ALLSOURCES}
        submit ${CLASS} ${PROJECT} ${ALLSOURCES}

dep : ${CPPSOURCE} ${CPPHEADER}
        @ echo "# ${DEPFILE} created ‘LC_TIME=C date‘" >${DEPFILE}
        ${MAKEDEPCPP} ${CPPSOURCE} >>${DEPFILE}

${DEPFILE} :
        @ touch ${DEPFILE}
        ${GMAKE} dep

again :
        ${GMAKE} spotless dep ci all lis

ifeq (${NEEDINCL}, )
include ${DEPFILE}
endif

```

```
$Id: README,v 1.2 2011-01-18 22:18:39-08 - - $

```

```
# Makefile.dep created Thu Apr 12 19:35:31 PDT 2012
bigint.o: bigint.cpp bigint.h trace.h
scanner.o: scanner.cpp scanner.h trace.h
trace.o: trace.cpp trace.h
util.o: util.cpp util.h trace.h
main.o: main.cpp bigint.h trace.h iterstack.h util.h scanner.h
```