```
 1: // $Id: bigint.h,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: #ifndef __BIGINT_H__
 4: #define __BIGINT_H__
 5:
 6: #include <exception>
 7: #include <iostream>
 8: #include <utility>
 9: using namespace std;
10:
11: #include "debug.h"
12:
13: //
14: // Define class bigint
15: //
16: class bigint {
17:       friend ostream& operator<< (ostream&, const bigint&);
18:    private:
19:       //long long_value {};
20:       typedef unsigned char digit_t;
21:       typedef vector<digit_t> bigvalue_t;
22:       bool negative;
23:       bigvalue_t big_value;
24:       typedef pair<bigint,bigint> quotient_remainder;
25:       quotient_remainder divide (const bigint&) const;
26:
27:       bigvalue_t do_bigadd (const bigvalue_t& left,
28:                             const bigvalue_t& right) const;
29:       bigvalue_t do_bigsub (const bigvalue_t& left,
30:                             const bigvalue_t& right) const;
31:       bigvalue_t do_bigmul (const bigvalue_t& left,
32:                             const bigvalue_t& right) const;
33:       bool do_bigless      (const bigvalue_t& left,
34:                             const bigvalue_t& right) const;
35:       bigvalue_t clear_zeros(bigvalue_t bignum) const;
36:       bigvalue_t mul_by_2 (bigvalue_t& big_value) const;
37:       bigvalue_t div_by_2 (bigvalue_t& big_value) const;
38:
39:    public:
40: bigint zero_clear(bigint big) const;
41:       bigvalue_t get_value() { return big_value; }
42:       void set_value(bigvalue_t val) { big_value = val; }
43:       //
44:       // Override implicit members.
45:       //
46:       bigint();
47:       bigint (const bigint&);
48:       bigint& operator= (const bigint&);
49:       ~bigint();
50:       //
51:       // Extra ctors to make bigints.
52:       //
53:       bigint (const long);
54:       bigint (const string&);
55:       //
56:       // Basic add/sub operators.
57:       //
58:       bigint operator+ (const bigint&) const;
```

```
  59:       bigint operator- (const bigint&) const;
  60:       bigint operator-() const;
  61:       long to_long() const;
  62:       //
  63:       // Extended operators implemented with add/sub.
  64:       //
  65:       bigint operator* (const bigint&) const;
  66:       bigint operator/ (const bigint&) const;
  67:       bigint operator% (const bigint&) const;
  68:       //
  69:       // Comparison operators.
  70:       //
  71:       bool operator== (const bigint&) const;
  72:       bool operator<  (const bigint&) const;
  73:       //
  74:       // Mutators (added)
  75:       //
  76:       void set_sign(bool neg) { this->negative = neg;}
  77: };
  78:
  79: //
  80: // The rest of the operators do not need to be friends.
  81: // Make the comparisons inline for efficiency.
  82: //
  83:
  84: bigint pow (const bigint& base, const bigint& exponent);
  85:
  86: inline bool operator!= (const bigint &left, const bigint &right) {
  87:    return not (left == right);
  88: }
  89: inline bool operator>  (const bigint &left, const bigint &right) {
  90:    return right < left;
  91: }
  92: inline bool operator<= (const bigint &left, const bigint &right) {
  93:    return not (right < left);
  94: }
  95: inline bool operator>= (const bigint &left, const bigint &right) {
  96:    return not (left < right);
  97: }
  98:
  99: #endif
 100:
```

```
 1: // $Id: scanner.h,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: #ifndef __SCANNER_H__
 4: #define __SCANNER_H__
 5:
 6: #include <iostream>
 7: #include <utility>
 8: using namespace std;
 9:
10: #include "debug.h"
11:
12: enum terminal_symbol {NUMBER, OPERATOR, SCANEOF};
13: struct token_t {
14:    terminal_symbol symbol;
15:    string lexinfo;
16: };
17:
18: class scanner {
19:    private:
20:       bool seen_eof;
21:       char lookahead;
22:       void advance();
23:    public:
24:       scanner();
25:       token_t scan();
26: };
27:
28: ostream& operator<< (ostream&, const terminal_symbol&);
29: ostream& operator<< (ostream&, const token_t&);
30:
31: #endif
32:
```

```
 1: // $Id: debug.h,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: #ifndef __DEBUG_H__
 4: #define __DEBUG_H__
 5:
 6: #include <string>
 7: #include <vector>
 8: using namespace std;
 9:
10: //
11: // debug -
12: //    static class for maintaining global debug flags, each indicated
13: //    by a single character.
14: // setflags -
15: //    Takes a string argument, and sets a flag for each char in the
16: //    string.  As a special case, '@', sets all flags.
17: // getflag -
18: //    Used by the DEBUGF macro to check to see if a flag has been set.
19: //    Not to be called by user code.
20: //
21:
22: class debugflags {
23:    private:
24:       static vector<bool> flags;
25:    public:
26:       static void setflags (const string& optflags);
27:       static bool getflag (char flag);
28:       static void where (char flag, const char* file, int line,
29:                          const char* func);
30: };
31:
```

```
32:
33: //
34: // DEBUGF -
35: //    Macro which expands into trace code.  First argument is a
36: //    trace flag char, second argument is output code that can
37: //    be sandwiched between <<.  Beware of operator precedence.
38: //    Example:
39: //       DEBUGF ('u', "foo = " << foo);
40: //    will print two words and a newline if flag 'u' is  on.
41: //    Traces are preceded by filename, line number, and function.
42: //
43:
44: #ifdef NDEBUG
45: #define DEBUGF(FLAG,CODE) ;
46: #define DEBUGS(FLAG,STMT) ;
47: #else
48: #define DEBUGF(FLAG,CODE) { \
49:            if (debugflags::getflag (FLAG)) { \
50:               debugflags::where (FLAG, __FILE__, __LINE__, __func__); \
51:               cerr << CODE << endl; \
52:            } \
53:         }
54: #define DEBUGS(FLAG,STMT) { \
55:            if (debugflags::getflag (FLAG)) { \
56:               debugflags::where (FLAG, __FILE__, __LINE__, __func__); \
57:               STMT; \
58:            } \
59:         }
60: #endif
61:
62: #endif
63:
```

```
 1: // $Id: util.h,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: //
 4: // util -
 5: //    A utility class to provide various services not conveniently
 6: //    included in other modules.
 7: //
 8:
 9: #ifndef __UTIL_H__
10: #define __UTIL_H__
11:
12: #include <iostream>
13: #include <stdexcept>
14: #include <vector>
15: using namespace std;
16:
17: #include "debug.h"
18:
19: //
20: // ydc_exn -
21: //    Indicate a problem where processing should be abandoned and
22: //    the main function should take control.
23: //
24:
25: class ydc_exn: public runtime_error {
26:    public:
27:       explicit ydc_exn (const string& what);
28: };
29:
30: //
31: // octal -
32: //    Convert integer to octal string.
33: //
34:
35: const string octal (long decimal);
36:
```

```
37:
38: //
39: // sys_info -
40: //    Keep track of execname and exit status.  Must be initialized
41: //    as the first thing done inside main.  Main should call:
42: //       sys_info::execname (argv[0]);
43: //    before anything else.
44: //
45:
46: class sys_info {
47:    private:
48:       static string execname_;
49:       static int status_;
50:    public:
51:       static void execname (const string& argv0);
52:       static const string& execname() {return execname_; }
53:       static void status (int status) {status_ = status; }
54:       static int status() {return status_; }
55: };
56:
57: //
58: // complain -
59: //    Used for starting error messages.  Sets the exit status to
60: //    EXIT_FAILURE, writes the program name to cerr, and then
61: //    returns the cerr ostream.  Example:
62: //       complain() << filename << ": some problem" << endl;
63: //
64:
65: ostream& complain();
66:
67: //
68: // operator<< (vector) -
69: //    An overloaded template operator which allows vectors to be
70: //    printed out as a single operator, each element separated from
71: //    the next with spaces.  The item_t must have an output operator
72: //    defined for it.
73: //
74:
75: template <typename item_t>
76: ostream& operator<< (ostream& out, const vector<item_t>& vec){
77:    string space = "";
78:    for (const auto& elem: vec) {
79:       out << space << elem;
80:       space = " ";
81:    }
82:    return out;
83: }
84:
85: #endif
86:
```

```
 1: // $Id: iterstack.h,v 1.5 2014-07-18 02:27:12-07 - - $
 2:
 3: //
 4: // The class std::stack does not provide an iterator, which is
 5: // needed for this class.  So, like std::stack, class iterstack
 6: // is implemented on top of a container.
 7: //
 8: // We use private inheritance because we want to restrict
 9: // operations only to those few that are approved.  All functions
10: // are merely inherited from the container, with only ones needed
11: // being exported as public.
12: //
13: // No implementation file is needed because all functions are
14: // inherited, and the convenience functions that are added are
15: // trivial, and so can be inline.
16: //
17: // Any underlying container which supports the necessary operations
18: // could be used, such as vector, list, or deque.
19: //
20:
21: #ifndef __ITERSTACK_H__
22: #define __ITERSTACK_H__
23:
24: #include <vector>
25: using namespace std;
26:
27: template <typename value_type>
28: class iterstack: private vector<value_type> {
29:    private:
30:       using stack_t = vector<value_type>;
31:       using stack_t::crbegin;
32:       using stack_t::crend;
33:       using stack_t::push_back;
34:       using stack_t::pop_back;
35:       using stack_t::back;
36:       using const_iterator = typename stack_t::const_reverse_iterator;
37:    public:
38:       using stack_t::clear;
39:       using stack_t::empty;
40:       using stack_t::size;
41:       inline const_iterator begin() {return crbegin();}
42:       inline const_iterator end() {return crend();}
43:       inline void push (const value_type& value) {push_back (value);}
44:       inline void pop() {pop_back();}
45:       inline const value_type& top() const {return back();}
46:
47: };
48:
49: #endif
50:
```

```
 1: // $Id: bigint.cpp,v 1.3 2014-07-18 02:26:41-07 - - $
 2:
 3: #include <cassert>
 4: #include <cstdlib>
 5: #include <exception>
 6: #include <limits>
 7: #include <stack>
 8: #include <stdexcept>
 9: #include <string>
10: #include <cmath>
11: #include "bigint.h"
12: #include "debug.h"
13:
14: using namespace std;
15:
16: typedef unsigned char digit_t;
17: typedef vector<digit_t> bigvalue_t;
18:
19: //
20: // constructor: creates an empty bigint
21: //
22: bigint::bigint() {
23:
24:    negative = false;
25: }
26:
27: //
28: // copy constructor
29: //
30: bigint::bigint (const bigint& that) {
31:   big_value = that.big_value;
32:   negative = that.negative;
33:   // CDTOR_TRACE;
34: }
35:
36:
37: bigint& bigint::operator= (const bigint& that) {
38:    if (this == &that) return *this;
39:    negative = that.negative;
40:    big_value = that.big_value;
41:    return *this;
42: }
43:
44: bigint::~bigint() {
45:    //CDTOR_TRACE;
46: }
47:
48: //construct from a long
49: bigint::bigint (long that) {
50:    bigint new_b_i = bigint(to_string(that));
51:    //CDTOR_TRACE;
52: }
53:
54: //Construct from a string
55: bigint::bigint (const string& that) {
56:    assert (that.size() > 0);
57:    negative = false; // set to positive by default
58:    int itor = that.size() - 1;
```

```
   59:    while (itor >= 0) {
   60:       if (that[itor] == '_'|| that[itor] == '-') {
   61:          this->negative = true; break;
   62:       }
   63:
   64:       this->big_value.push_back(that[itor]);
   65:       --itor;
   66:    }
   67:    //long_value = isnegative ? - newval : + newval;
   68:    //CDTOR_TRACE;
   69: }
   70:
   71: //simple print for debugging
   72: void print(const bigvalue_t& num) {
   73:    for(int i = num.size() - 1; i >= 0; --i) {
   74:       cout << num[i];
   75:    }
   76: }
   77:
   78: bigint bigint::operator+ (const bigint& that) const {
   79:    bigint result;
   80:    if(this->negative == that.negative) {
   81:       result.big_value = do_bigadd(this->big_value, that.big_value);
   82:       result.set_sign(this->negative);
   83:    } else {
   84:       if(do_bigless(this->big_value, that.big_value)) {
   85:          result.big_value = do_bigsub(that.big_value, this->big_value);
   86:          result.negative = that.negative;
   87:       } else {
   88:          result.big_value = do_bigsub(this->big_value, that.big_value);
   89:          result.set_sign(this->negative);
   90:       }
   91:    }
   92:    return result;
   93: }
   94:
   95: bigint bigint::operator- (const bigint& that) const {
   96:    bigint result;
   97:    if(this->negative != that.negative) {
   98:       result.big_value = do_bigadd(this->big_value, that.big_value);
   99:       result.negative = this->negative;
  100:    } else {
  101:       if(do_bigless(this->big_value, that.big_value)) {
  102:          result.big_value = do_bigsub(that.big_value, this->big_value);
  103:          result.negative = not(that.negative);
  104:       } else {
  105:         result.big_value = do_bigsub(this->big_value, that.big_value);
  106:         result.negative = (this->negative);
  107:       }
  108:    }
  109:    return result;
  110: }
  111:
  112: //
  113: // WHAT IS THIS SUPPOSED TO DO???
  114: //
  115: bigint bigint::operator-() const {
  116:    return *this;
```

```
117: }
118:
119: //Converts a bigint to type long
120: long bigint::to_long() const {
121:    //max value of a long
122:    bigint min = bigint("9223372036854775807");
123:    if(!do_bigless(big_value, min.big_value)) {
124:       throw range_error("to_long: out of range");
125:    }
126:    long long_val = 0;
127:    for(size_t i = 0; i < big_value.size(); ++i) {
128:       long_val = long_val + ((big_value[i] - '0') * (pow(10,i)));
129:    }
130:    if(negative) long_val -= (long_val * 2);
131:    return long_val;
132: }
133:
134:
135:
136: //
137: // Multiplication algorithm.
138: //
139: bigint bigint::operator* (const bigint& that) const {
140:    bigint result;
141:    if(this->negative == that.negative){
142:       result.negative = false;
143:    } else {
144:       result.negative = true;
145:    }
146:    result.big_value = do_bigmul(this->big_value, that.big_value);
147:    return result;
148: }
149:
150: // following algorithm in project description
151: bigvalue_t bigint::do_bigmul(const bigvalue_t& left,
152:                              const bigvalue_t& right) const {
153:
154:    bigvalue_t product;
155:    for(size_t i = 0; i < (left.size() + right.size()); i++){
156:       product.push_back('0');
157:    }
158:    unsigned int c, d;
159:    for(size_t i = 0; i < left.size(); ++i) {
160:       c = 0;
161:       for(size_t j = 0; j < right.size(); ++j) {
162:          d = (product[i+j]-'0') + ((left[i]-'0') * (right[j]-'0')) + c;
163:          product[i+j] = (d % 10) + '0';
164:          c = d/10; // takes floor(d/10) by truncation like we want
165:       }
166:       product[i + right.size()] = (c + '0');
167:    }
168:    product = clear_zeros(product);
169:    return product;
170: }
171:
172: //Multiply by 2 algorithm for long division
173: bigvalue_t bigint::mul_by_2 (bigvalue_t& big_value) const {
174:    bigint two;
```

```
175:    two.big_value.push_back('2');
176:    bigvalue_t result = do_bigmul(big_value, two.big_value);
177:    return result;
178: }
179:
180: //Division by 2 algorithm to aid with long division
181: bigvalue_t bigint::div_by_2 (bigvalue_t& big_value) const {
182:    bigvalue_t tmp = big_value;
183:    bigvalue_t size_one;
184:    size_one.push_back('1');
185:    bigvalue_t size_two;
186:    size_two.push_back('2');
187:    bigvalue_t pow_ten = bigint("1024").big_value;
188:    bigvalue_t pow_nine = bigint("512").big_value;
189:    bigvalue_t pow_eight = bigint("256").big_value;
190:    bigvalue_t quotient;
191:    quotient.push_back('0');
192:
193:    for(;;) {
194:       bigvalue_t pow_big = pow_ten;
195:       bigvalue_t pow_mid = pow_nine;
196:       bigvalue_t pow_less = pow_eight;
197:       while(do_bigless(pow_big, tmp)) {
198:          pow_big = clear_zeros(pow_big);
199:          pow_mid = clear_zeros(pow_mid);
200:          pow_less = clear_zeros(pow_less);
201:          pow_big = mul_by_2(pow_big);
202:          pow_mid = mul_by_2(pow_mid);
203:          pow_less = mul_by_2(pow_less);
204:       }
205:       if(do_bigless(pow_mid, tmp)) {
206:          tmp = do_bigsub(tmp, pow_mid);
207:          quotient = do_bigadd(quotient, pow_less);
208:       } else if(do_bigless(size_one, tmp)) {
209:          tmp = do_bigsub(tmp, size_two);
210:          quotient = do_bigadd(quotient, size_one);
211:       }
212:       if(tmp.empty()){ break; }
213:       else if(tmp == size_one) { break; }
214:    }
215:    return quotient;
216: }
217:
218: //Long division algorithm
219: bigint::quotient_remainder bigint::divide (const bigint& that) const {
220:    bigvalue_t div = that.big_value;
221:    div = clear_zeros(div);
222:    if (div.empty()){ throw domain_error ("divide by 0"); }
223:    bigvalue_t zero;
224:    zero.push_back('0');
225:    bigvalue_t divisor = that.big_value;
226:    bigint quotient;
227:    quotient.big_value.push_back('0');
228:    bigint remainder;
229:    remainder.big_value = this->big_value;
230:    bigvalue_t size_one;
231:    size_one.push_back('1');
232:    bigvalue_t power_of_2;
```

```
233:    power_of_2.push_back('1');
234:    while (do_bigless (divisor, remainder.big_value)) {
235:       divisor = mul_by_2 (divisor);
236:       power_of_2 = mul_by_2 (power_of_2);
237:    }
238:    while (do_bigless (zero, power_of_2)) {
239:       if (do_bigless (divisor, remainder.big_value)) {
240:          remainder.big_value = do_bigsub(remainder.big_value, divisor);
241:          quotient.big_value = do_bigadd(quotient.big_value, power_of_2);
242:       }
243:       divisor = div_by_2 (divisor);
244:       power_of_2 = div_by_2 (power_of_2);
245:    }
246:    if(remainder.big_value == div) {
247:       quotient.big_value = do_bigadd(quotient.big_value, size_one);
248:       remainder.big_value = do_bigsub(remainder.big_value, div);
249:    }
250:    return {quotient, remainder};
251: }
252:
253:
254: bigint bigint::operator/ (const bigint& that) const {
255:    return divide (that).first;
256: }
257:
258: bigint bigint::operator% (const bigint& that) const {
259:    return divide (that).second;
260: }
261:
262:
263: bool bigint::operator== (const bigint& that) const {
264:    // check signs
265:    if(this->negative!=that.negative) return false;
266:    // check lengths
267:    if(this->big_value.size()!=that.big_value.size()) return false;
268:    // check digits
269:    for(size_t i = this->big_value.size(); i>0; --i) {
270:       if(this->big_value.at(i-1)!=that.big_value.at(i-1)){
271:          return false;
272:       }
273:    }
274:    return true;
275: }
276:
277: bool bigint::operator< (const bigint& that) const {
278:    // check signs
279:    if(this->negative==that.negative) {
280:       if(!(this->negative)) {
281:          return do_bigless(this->big_value, that.big_value);
282:       } else {
283:          return !(do_bigless(this->big_value, that.big_value));
284:       }
285:    } else {
286:         // if signs are not the same:
287:         if(this->negative)
288:            return true;
289:         else
290:            return false;
```

```
291:    }
292:    return false;
293: }
294:
295: ostream& operator<< (ostream& out, const bigint& that) {
296:    if(that.big_value.empty()){
297:       out << "0";
298:    } else {
299:       if(that.negative) cout << "-";
300:       for(size_t i = that.big_value.size(); i > 0; --i) {
301:          out << that.big_value.at(i - 1);
302:       }
303:    }
304:    return out;
305: }
306:
307: //Exponent algorithm
308: bigint pow (const bigint& base, const bigint& exponent) {
309:    DEBUGF ('^', "base = " << base << ", exponent = " << exponent);
310:    bigint zero = bigint("0");
311:    bigint one = bigint("1");
312:    zero = zero.zero_clear(zero);
313:    if (base == zero){
314:       return zero;
315:    }
316:    bigint base_copy = base;
317:    long expt = exponent.to_long();
318:    bigint result = bigint("1");
319:    if (expt < 0) {
320:       base_copy = one / base_copy;
321:       expt = - expt;
322:    }
323:    while (expt > 0) {
324:       if (expt & 1) { //odd
325:          result = result * base_copy;
326:          --expt;
327:       } else { //even
328:          base_copy = base_copy * base_copy;
329:          expt /= 2;
330:       }
331:    }
332:    DEBUGF ('^', "result = " << result);
333:    return result;
334: }
335:
336: // do_bigadd: adds left and right
337: bigvalue_t bigint::do_bigadd (const bigvalue_t& left,
338:                               const bigvalue_t& right) const {
339:
340:    bigvalue_t result;
341:    unsigned int l_dig, r_dig, sum, carry;
342:    carry = 0;
343:    // note: iterate 1 more time then size to account for carry and
344:    //       assign 0's if necessary
345:    for(size_t i = 0;
346:        i <= max(left.size(), right.size());
347:        ++i) {
348:       // reset sum
```

```
349:       sum = 0;
350:       // get digits
351:       if(i < left.size())
352:          l_dig = left[i] - '0';
353:       else
354:          l_dig = 0;
355:       if(i < right.size())
356:          r_dig = right[i] - '0';
357:       else
358:          r_dig = 0;
359:       // add digits (including carry from last iteration)
360:       sum = l_dig + r_dig + carry;
361:       carry = 0;
362:       if(sum > 9) {
363:          sum -= 10; carry = 1;
364:       }
365:       result.push_back(sum + '0'); // make sure this is a char somehow?
366:    }
367:    result = clear_zeros(result);
368:    return result;
369: }
370:
371: // do_bigsub: subtracts right from left
372: bigvalue_t bigint::do_bigsub (const bigvalue_t& left,
373:                               const bigvalue_t& right) const {
374:    bigvalue_t result;
375:    int l_dig, r_dig, diff, borrow;
376:    borrow = 0;
377:    for(size_t i = 0;
378:        i < max(left.size(), right.size());
379:        ++i) {
380:       // reset diff
381:       diff = 0;
382:       // get digits
383:       if(i < left.size())
384:          l_dig = left[i] - '0';
385:       else
386:          l_dig = 0;
387:       if(i < right.size())
388:          r_dig = right[i] - '0';
389:       else
390:          r_dig = 0;
391:
392:       // subtract digits (including borrow from last iteration)
393:       diff = l_dig - r_dig + borrow;
394:       borrow = 0;
395:       if(diff < 0) { diff += 10; borrow = -1; }
396:          result.push_back(diff + '0');
397:    }
398:    result = clear_zeros(result);
399:    return result;
400: }
401:
402: // do_bigless: compares abs value of the 2 elements
403: // returns true if left < right
404: bool bigint::do_bigless(const bigvalue_t& left,
405:                         const bigvalue_t& right) const {
406:
```

```
407:    if(left.size() < right.size()) return true;
408:    if(left.size() > right.size()) return false;
409:      for(size_t i = left.size(); i > 0; --i) {
410:        if(left.at(i-1) < right.at(i-1)) return true;
411:        if(left.at(i-1) > right.at(i-1)) return false;
412:      }
413:    return false;
414: }
415:
416: // clear_zeros(): clears leading 0's
417: bigvalue_t bigint::clear_zeros (bigvalue_t big_num) const {
418:    for(size_t i = big_num.size(); i > 0; --i) {
419:       if(big_num[i-1] != '0') break;
420:       else big_num.pop_back();
421:    }
422:
423:    return big_num;
424: }
425:
426: bigint bigint::zero_clear(bigint big) const{
427:    for(size_t i = big.big_value.size(); i > 0; --i) {
428:       if(big.big_value[i-1] != '0')
429:          break;
430:       else
431:          big.big_value.pop_back();
432:    }
433:    return big;
434: }
```

```
 1: // $Id: scanner.cpp,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: #include <iostream>
 4: #include <locale>
 5: using namespace std;
 6:
 7: #include "scanner.h"
 8: #include "debug.h"
 9:
10: scanner::scanner() {
11:    seen_eof = false;
12:    advance();
13: }
14:
15: void scanner::advance() {
16:    if (not seen_eof) {
17:       cin.get (lookahead);
18:       if (cin.eof()) seen_eof = true;
19:    }
20: }
21:
22: token_t scanner::scan() {
23:    token_t result;
24:    while (not seen_eof and isspace (lookahead)) advance();
25:    if (seen_eof) {
26:       result.symbol = SCANEOF;
27:    }else if (lookahead == '_' or isdigit (lookahead)) {
28:       result.symbol = NUMBER;
29:       do {
30:          result.lexinfo += lookahead;
31:          advance();
32:       }while (not seen_eof and isdigit (lookahead));
33:    }else {
34:       result.symbol = OPERATOR;
35:       result.lexinfo += lookahead;
36:       advance();
37:    }
38:    DEBUGF ('S', result);
39:    return result;
40: }
41:
42: ostream& operator<< (ostream& out, const terminal_symbol& symbol) {
43:    switch (symbol) {
44:       case NUMBER  : out << "NUMBER"  ; break;
45:       case OPERATOR: out << "OPERATOR"; break;
46:       case SCANEOF : out << "SCANEOF" ; break;
47:    }
48:    return out;
49: }
50:
51: ostream& operator<< (ostream& out, const token_t& token) {
52:    out << token.symbol << ": \"" << token.lexinfo << "\"";
53:    return out;
54: }
55:
```

```
 1: // $Id: debug.cpp,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: #include <climits>
 4: #include <iostream>
 5: #include <vector>
 6: using namespace std;
 7:
 8: #include "debug.h"
 9: #include "util.h"
10:
11: vector<bool> debugflags::flags (UCHAR_MAX + 1, false);
12:
13: void debugflags::setflags (const string& initflags) {
14:    for (const unsigned char flag: initflags) {
15:       if (flag == '@') flags.assign (flags.size(), true);
16:                   else flags[flag] = true;
17:    }
18:    // Note that DEBUGF can trace setflags.
19:    if (getflag ('x')) {
20:       string flag_chars;
21:       for (size_t index = 0; index < flags.size(); ++index) {
22:          if (getflag (index)) flag_chars += (char) index;
23:       }
24:       DEBUGF ('x', "debugflags::flags = " << flag_chars);
25:    }
26: }
27:
28: //
29: // getflag -
30: //    Check to see if a certain flag is on.
31: //
32:
33: bool debugflags::getflag (char flag) {
34:    return flags[static_cast<unsigned char> (flag)];
35: }
36:
37: void debugflags::where (char flag, const char* file, int line,
38:                         const char* func) {
39:    cout << sys_info::execname() << ": DEBUG(" << flag << ") "
40:         << file << "[" << line << "] " << func << "()" << endl;
41: }
42:
```

```
 1: // $Id: util.cpp,v 1.1 2014-07-18 02:03:07-07 - - $
 2:
 3: #include <cstdlib>
 4: #include <sstream>
 5: using namespace std;
 6:
 7: #include "util.h"
 8:
 9: ydc_exn::ydc_exn (const string& what): runtime_error (what) {
10: }
11:
12: const string octal (long decimal) {
13:    ostringstream ostring;
14:    ostring.setf (ios::oct);
15:    ostring << decimal;
16:    return ostring.str();
17: }
18:
19: string sys_info::execname_; // Must be initialized from main().
20: int sys_info::status_ = EXIT_SUCCESS;
21:
22: void sys_info::execname (const string& argv0) {
23:    execname_ = argv0;
24:    cout << boolalpha;
25:    cerr << boolalpha;
26:    DEBUGF ('Y', "execname = " << execname_);
27: }
28:
29: ostream& complain() {
30:    sys_info::status (EXIT_FAILURE);
31:    cerr << sys_info::execname() << ": ";
32:    return cerr;
33: }
34:
```

```
 1: // $Id: main.cpp,v 1.2 2014-07-18 02:17:42-07 - - $
 2:
 3: #include <deque>
 4: #include <iostream>
 5: #include <map>
 6: #include <stdexcept>
 7: #include <utility>
 8: #include <cassert>
 9: using namespace std;
10:
11: #include <unistd.h>
12:
13: #include "bigint.h"
14: #include "debug.h"
15: #include "iterstack.h"
16: #include "scanner.h"
17: #include "util.h"
18:
19: using bigint_stack = iterstack<bigint>;
20:
21: void do_arith (bigint_stack& stack, const char oper) {
22:    if (stack.size() < 2) throw ydc_exn ("stack empty");
23:    bigint right = stack.top();
24:    stack.pop();
25:    DEBUGF ('d', "right = " << right);
26:    bigint left = stack.top();
27:    stack.pop();
28:    DEBUGF ('d', "left = " << left);
29:    bigint result;
30:    switch (oper) {
31:       case '+': result = left + right; break;
32:       case '-': result = left - right; break;
33:       case '*': result = left * right; break;
34:       case '/': result = left / right; break;
35:       case '%': result = left % right; break;
36:       case '^': result = pow (left, right); break;
37:       default: throw invalid_argument (
38:                      string ("do_arith operator is ") + oper);
39:    }
40:    DEBUGF ('d', "result = " << result);
41:    stack.push (result);
42: }
43:
44: void do_clear (bigint_stack& stack, const char) {
45:    DEBUGF ('d', "");
46:    stack.clear();
47: }
48:
49: void do_dup (bigint_stack& stack, const char) {
50:    bigint top = stack.top();
51:    DEBUGF ('d', top);
52:    stack.push (top);
53: }
54:
```

```
55:
56: void do_printall (bigint_stack& stack, const char) {
57:    for (const auto &elem: stack) cout << elem << endl;
58: }
59:
60: void do_print (bigint_stack& stack, const char) {
61:    cout << stack.top() << endl;
62: }
63:
64: void do_debug (bigint_stack& stack, const char) {
65:    (void) stack; // SUPPRESS: warning: unused parameter 'stack'
66:    cout << "Y not implemented" << endl;
67: }
68:
69: class ydc_quit: public exception {};
70: void do_quit (bigint_stack&, const char) {
71:    throw ydc_quit();
72: }
73:
74: using function_t = void (*)(bigint_stack&, const char);
75: using fn_map = map<string,function_t>;
76: fn_map do_functions = {
77:    {"+", do_arith},
78:    {"-", do_arith},
79:    {"*", do_arith},
80:    {"/", do_arith},
81:    {"%", do_arith},
82:    {"^", do_arith},
83:    {"Y", do_debug},
84:    {"c", do_clear},
85:    {"d", do_dup},
86:    {"f", do_printall},
87:    {"p", do_print},
88:    {"q", do_quit},
89: };
90:
```

```
 91:
 92: //
 93: // scan_options
 94: //    Options analysis:  The only option is -Dflags.
 95: //
 96:
 97: void scan_options (int argc, char** argv) {
 98:    assert (sys_info::execname().size() > 0);
 99:    //if (sys_info::execname().size() == 0) sys_info::execname (argv[0]);
100:    opterr = 0;
101:    for (;;) {
102:       int option = getopt (argc, argv, "@:");
103:       if (option == EOF) break;
104:       switch (option) {
105:          case '@':
106:             debugflags::setflags (optarg);
107:             break;
108:          default:
109:             complain() << "-" << (char) optopt << ": invalid option"
110:                        << endl;
111:             break;
112:       }
113:    }
114:    if (optind < argc) {
115:       complain() << "operand not permitted" << endl;
116:    }
117: }
```

```
118:
119: //
120: // Main function.
121: //
122:
123: int main (int argc, char** argv) {
124:    sys_info::execname (argv[0]);
125:    scan_options (argc, argv);
126:    bigint_stack operand_stack;
127:    scanner input;
128:    try {
129:       for (;;) {
130:          try {
131:             token_t token = input.scan();
132:             if (token.symbol == SCANEOF) break;
133:             switch (token.symbol) {
134:                case NUMBER:
135:                   operand_stack.push (token.lexinfo);
136:                   break;
137:                case OPERATOR: {
138:                   fn_map::const_iterator fn
139:                            = do_functions.find (token.lexinfo);
140:                   if (fn == do_functions.end()) {
141:                      throw ydc_exn (octal (token.lexinfo[0])
142:                                     + " is unimplemented");
143:                   }
144:                   fn->second (operand_stack, token.lexinfo.at(0));
145:                   break;
146:                   }
147:                default:
148:                   break;
149:             }
150:          }catch (ydc_exn& exn) {
151:             cout << exn.what() << endl;
152:          }
153:       }
154:    }catch (ydc_quit&) {
155:       // Intentionally left empty.
156:    }
157:    return sys_info::status();
158: }
159:
```

```
# $Id: Makefile,v 1.2 2014-07-18 02:33:47-07 - - $

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

COMPILECPP  = g++ -g -O0 -Wall -Wextra -std=gnu++11
MAKEDEPCPP  = g++ -MM

CPPHEADER   = bigint.h   scanner.h   debug.h   util.h   iterstack.h
CPPSOURCE   = bigint.cpp scanner.cpp debug.cpp util.cpp main.cpp
EXECBIN     = ydc
OBJECTS     = ${CPPSOURCE:.cpp=.o}
OTHERS      = ${MKFILE} README
ALLSOURCES  = ${CPPHEADER} ${CPPSOURCE} ${OTHERS}
LISTING     = Listing.ps
CLASS           = cmps109-wm.u14
PORJECT         = asg2

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

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

%.o : %.cpp
        ${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} ${LISTING:.ps=.pdf}

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

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

```
59:
60: ifeq (${NEEDINCL}, )
61: include ${DEPFILE}
62: endif
63:
```

```
1: $Id: README,v 1.1 2014-07-18 02:03:07-07 - - $
2:
```

```
1: # Makefile.dep created Fri Jul 18 02:33:47 PDT 2014
2: bigint.o: bigint.cpp bigint.h debug.h
3: scanner.o: scanner.cpp scanner.h debug.h
4: debug.o: debug.cpp debug.h util.h
5: util.o: util.cpp util.h debug.h
6: main.o: main.cpp bigint.h debug.h iterstack.h scanner.h util.h
```