```
 1: // $Id: bigint.h,v 1.9 2014-04-11 11:58:33-07 - - $
 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 pair<bigint,bigint> quotient_remainder;
21:       quotient_remainder divide (const bigint&) const;
22:    public:
23:       //
24:       // Override implicit members.
25:       //
26:       bigint();
27:       bigint (const bigint&);
28:       bigint& operator= (const bigint&);
29:       ~bigint();
30:       //
31:       // Extra ctors to make bigints.
32:       //
33:       bigint (const long);
34:       bigint (const string&);
35:       //
36:       // Basic add/sub operators.
37:       //
38:       bigint operator+ (const bigint&) const;
39:       bigint operator- (const bigint&) const;
40:       bigint operator-() const;
41:       long to_long() const;
42:       //
43:       // Extended operators implemented with add/sub.
44:       //
45:       bigint operator* (const bigint&) const;
46:       bigint operator/ (const bigint&) const;
47:       bigint operator% (const bigint&) const;
48:       //
49:       // Comparison operators.
50:       //
51:       bool operator== (const bigint&) const;
52:       bool operator<  (const bigint&) const;
53: };
54:
```

```
55:
56: bigint pow (const bigint& base, const bigint& exponent);
57:
58: //
59: // Rest of the comparisons don't need to be members.
60: //
61: #define BOOLOPER(OPER,EXPRESSION) \
62: inline bool operator OPER (const bigint &left, const bigint &right) { \
63:    return EXPRESSION; \
64: }
65: BOOLOPER(!=, not (left == right))
66: BOOLOPER(> , right < left       )
67: BOOLOPER(<=, not (right < left) )
68: BOOLOPER(>=, not (left < right) )
69:
70: //
71: // Operators with a left operand long and right operand bigint.
72: //
73: #define LONGLEFT(RTYPE,OPER) \
74: inline RTYPE operator OPER (long left, const bigint& right) { \
75:    return bigint (left) OPER right; \
76: }
77: LONGLEFT(bigint,+)
78: LONGLEFT(bigint,-)
79: LONGLEFT(bigint,*)
80: LONGLEFT(bigint,/)
81: LONGLEFT(bigint,%)
82: LONGLEFT(bool,==)
83: LONGLEFT(bool,<)
84: LONGLEFT(bool,!=)
85: LONGLEFT(bool,>)
86: LONGLEFT(bool,<=)
87: LONGLEFT(bool,>=)
88:
89: #endif
90:
```

```
 1: // $Id: scanner.h,v 1.2 2014-04-08 19:04:03-07 - - $
 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 2014-04-08 19:04:03-07 - - $
 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 1.5 2014-04-09 17:03:58-07 - - $
 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 1.10 2014-04-11 12:40:16-07 - - $
 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 vector<value_type>::crbegin;
31:       using vector<value_type>::crend;
32:       using vector<value_type>::push_back;
33:       using vector<value_type>::pop_back;
34:       using vector<value_type>::back;
35:       typedef typename vector<value_type>::const_reverse_iterator
36:               const_iterator;
37:    public:
38:       using vector<value_type>::clear;
39:       using vector<value_type>::empty;
40:       using vector<value_type>::size;
41:       const_iterator begin() { return crbegin(); }
42:       const_iterator end() { return crend(); }
43:       void push (const value_type& value) { push_back (value); }
44:       void pop() { pop_back(); }
45:       const value_type& top() const { return back(); }
46: };
47:
48: #endif
49:
```

```
 1: // $Id: bigint.cpp,v 1.55 2014-04-09 17:03:58-07 - - $
 2:
 3: #include <cassert>
 4: #include <cstdlib>
 5: #include <exception>
 6: #include <limits>
 7: #include <stack>
 8: #include <stdexcept>
 9: using namespace std;
10:
11: #include "bigint.h"
12: #include "debug.h"
13:
14: #define CDTOR_TRACE DEBUGF ('~', this << " -> " << long_value)
15:
16: bigint::bigint(): long_value (0) {
17:    CDTOR_TRACE;
18: }
19:
20: bigint::bigint (const bigint& that): long_value (that.long_value) {
21:    *this = that;
22:    CDTOR_TRACE;
23: }
24:
25: bigint& bigint::operator= (const bigint& that) {
26:    if (this == &that) return *this;
27:    this->long_value = that.long_value;
28:    return *this;
29: }
30:
31: bigint::~bigint() {
32:    CDTOR_TRACE;
33: }
34:
35: bigint::bigint (long that): long_value (that) {
36:    CDTOR_TRACE;
37: }
38:
39: bigint::bigint (const string& that) {
40:    assert (that.size() > 0);
41:    auto itor = that.cbegin();
42:    bool isnegative = false;
43:    if (*itor == '_') {isnegative = true; ++itor; }
44:    int newval = 0;
45:    while (itor != that.end()) newval = newval * 10 + *itor++ - '0';
46:    long_value = isnegative ? - newval : + newval;
47:    CDTOR_TRACE;
48: }
49:
```

```
50:
51: bigint bigint::operator+ (const bigint& that) const {
52:    return this->long_value + that.long_value;
53: }
54:
55: bigint bigint::operator- (const bigint& that) const {
56:    return this->long_value - that.long_value;
57: }
58:
59: bigint bigint::operator-() const {
60:    return -long_value;
61: }
62:
63: long bigint::to_long() const {
64:    if (*this <= bigint (numeric_limits<long>::min())
65:     or *this > bigint (numeric_limits<long>::max()))
66:                throw range_error ("to_long: out of range");
67:    return long_value;
68: }
69:
70: bool abs_less (const long& left, const long& right) {
71:    return left < right;
72: }
73:
74: //
75: // Multiplication algorithm.
76: //
77: bigint bigint::operator* (const bigint& that) const {
78:    return long_value * that.long_value;
79: }
80:
81: //
82: // Division algorithm.
83: //
84:
85: void mul_by_2 (long& long_value) {
86:    long_value *= 2;
87: }
88:
89: void div_by_2 (long& long_value) {
90:    long_value /= 2;
91: }
92:
```

```
 93:
 94: bigint::quotient_remainder bigint::divide (const bigint& that) const {
 95:    if (that == 0) throw domain_error ("divide by 0");
 96:    typedef long unumber;
 97:    static unumber zero = 0;
 98:    if (that == 0) throw domain_error ("bigint::divide");
 99:    unumber divisor = that.long_value;
100:    unumber quotient = 0;
101:    unumber remainder = this->long_value;
102:    unumber power_of_2 = 1;
103:    while (abs_less (divisor, remainder)) {
104:       mul_by_2 (divisor);
105:       mul_by_2 (power_of_2);
106:    }
107:    while (abs_less (zero, power_of_2)) {
108:       if (not abs_less (remainder, divisor)) {
109:          remainder = remainder - divisor;
110:          quotient = quotient + power_of_2;
111:       }
112:       div_by_2 (divisor);
113:       div_by_2 (power_of_2);
114:    }
115:    return {quotient, remainder};
116: }
117:
118: bigint bigint::operator/ (const bigint& that) const {
119:    return divide (that).first;
120: }
121:
122: bigint bigint::operator% (const bigint& that) const {
123:    return divide (that).second;
124: }
125:
126: bool bigint::operator== (const bigint& that) const {
127:    return this->long_value == that.long_value;
128: }
129:
130: bool bigint::operator< (const bigint& that) const {
131:    return this->long_value < that.long_value;
132: }
133:
134: ostream& operator<< (ostream& out, const bigint& that) {
135:    out << that.long_value;
136:    return out;
137: }
138:
```

```
139:
140: bigint pow (const bigint& base, const bigint& exponent) {
141:    DEBUGF ('^', "base = " << base << ", exponent = " << exponent);
142:    if (base == 0) return 0;
143:    bigint base_copy = base;
144:    long expt = exponent.to_long();
145:    bigint result = 1;
146:    if (expt < 0) {
147:       base_copy = 1 / base_copy;
148:       expt = - expt;
149:    }
150:    while (expt > 0) {
151:       if (expt & 1) { //odd
152:          result = result * base_copy;
153:          --expt;
154:       }else { //even
155:          base_copy = base_copy * base_copy;
156:          expt /= 2;
157:       }
158:    }
159:    DEBUGF ('^', "result = " << result);
160:    return result;
161: }
```

```
 1: // $Id: scanner.cpp,v 1.7 2014-04-08 18:43:33-07 - - $
 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 1.2 2014-04-08 18:41:29-07 - - $
 2:
 3: #include <cassert>
 4: #include <climits>
 5: #include <iostream>
 6: #include <vector>
 7: using namespace std;
 8:
 9: #include "debug.h"
10: #include "util.h"
11:
12: vector<bool> debugflags::flags (UCHAR_MAX + 1, false);
13:
14: void debugflags::setflags (const string& initflags) {
15:    for (const char flag: initflags) {
16:       if (flag == '@') flags.assign (flags.size(), true);
17:                   else flags[flag] = true;
18:    }
19:    // Note that DEBUGF can trace setflags.
20:    if (getflag ('x')) {
21:       string flag_chars;
22:       for (size_t index = 0; index < flags.size(); ++index) {
23:          if (getflag (index)) flag_chars += (char) index;
24:       }
25:       DEBUGF ('x', "debugflags::flags = " << flag_chars);
26:    }
27: }
28:
29: //
30: // getflag -
31: //    Check to see if a certain flag is on.
32: //
33:
34: bool debugflags::getflag (char flag) {
35:    // WARNING: Don't TRACE this function or the stack will blow up.
36:    unsigned uflag = (unsigned char) flag;
37:    assert (uflag < flags.size());
38:    return flags[uflag];
39: }
40:
41: void debugflags::where (char flag, const char* file, int line,
42:                         const char* func) {
43:    cout << sys_info::execname() << ": DEBUG(" << flag << ") "
44:         << file << "[" << line << "] " << func << "()" << endl;
45: }
46:
```

```
 1: // $Id: util.cpp,v 1.10 2014-04-09 16:45:33-07 - - $
 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 1.38 2014-04-08 18:57:45-07 - - $
 2:
 3: #include <cassert>
 4: #include <deque>
 5: #include <iostream>
 6: #include <map>
 7: #include <stdexcept>
 8: #include <utility>
 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: typedef iterstack<bigint> bigint_stack;
20:
21: void do_arith (bigint_stack& stack, const char oper) {
22:    bigint right = stack.top(); \
23:    stack.pop(); \
24:    DEBUGF ('d', "right = " << right); \
25:    bigint left = stack.top(); \
26:    stack.pop(); \
27:    DEBUGF ('d', "left = " << left); \
28:    bigint result;
29:    switch (oper) {
30:       case '+': result = left + right; break;
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 = pow (left, right); break;
36:       default: throw invalid_argument (
37:                      string ("do_arith operator is ") + oper);
38:    }
39:    DEBUGF ('d', "result = " << result); \
40:    stack.push (result); \
41: }
42:
43: void do_clear (bigint_stack& stack, const char) {
44:    DEBUGF ('d', "");
45:    stack.clear();
46: }
47:
48: void do_dup (bigint_stack& stack, const char) {
49:    bigint top = stack.top();
50:    DEBUGF ('d', top);
51:    stack.push (top);
52: }
53:
```

```
54:
55: void do_printall (bigint_stack& stack, const char) {
56:    for (const auto &elem: stack) cout << elem << endl;
57: }
58:
59: void do_print (bigint_stack& stack, const char) {
60:    cout << stack.top() << endl;
61: }
62:
63: void do_debug (bigint_stack& stack, const char) {
64:    (void) stack; // SUPPRESS: warning: unused parameter 'stack'
65:    cout << "Y not implemented" << endl;
66: }
67:
68: class ydc_quit: public exception {};
69: void do_quit (bigint_stack&, const char) {
70:    throw ydc_quit();
71: }
72:
73: typedef void (*function_t) (bigint_stack&, const char);
74: typedef map <string, function_t> fn_map;
75: fn_map do_functions = {
76:    {"+", do_arith},
77:    {"-", do_arith},
78:    {"*", do_arith},
79:    {"/", do_arith},
80:    {"%", do_arith},
81:    {"^", do_arith},
82:    {"Y", do_debug},
83:    {"c", do_clear},
84:    {"d", do_dup},
85:    {"f", do_printall},
86:    {"p", do_print},
87:    {"q", do_quit},
88: };
89:
```

```
 90:
 91: //
 92: // scan_options
 93: //    Options analysis:  The only option is -Dflags.
 94: //
 95:
 96: void scan_options (int argc, char** argv) {
 97:    assert (sys_info::execname().size() > 0);
 98:    opterr = 0;
 99:    for (;;) {
100:       int option = getopt (argc, argv, "@:");
101:       if (option == EOF) break;
102:       switch (option) {
103:          case '@':
104:             debugflags::setflags (optarg);
105:             break;
106:          default:
107:             complain() << "-" << (char) optopt << ": invalid option"
108:                        << endl;
109:             break;
110:       }
111:    }
112:    if (optind < argc) {
113:       complain() << "operand not permitted" << endl;
114:    }
115: }
116:
117: int main (int argc, char** argv) {
118:    sys_info::execname (argv[0]);
119:    scan_options (argc, argv);
120:    bigint_stack operand_stack;
121:    scanner input;
122:    try {
123:       for (;;) {
124:          try {
125:             token_t token = input.scan();
126:             if (token.symbol == SCANEOF) break;
127:             switch (token.symbol) {
128:                case NUMBER:
129:                   operand_stack.push (token.lexinfo);
130:                   break;
131:                case OPERATOR: {
132:                   fn_map::const_iterator fn
133:                            = do_functions.find (token.lexinfo);
134:                   if (fn == do_functions.end()) {
135:                      throw ydc_exn (octal (token.lexinfo[0])
136:                                     + " is unimplemented");
137:                   }
138:                   fn->second (operand_stack, token.lexinfo.at(0));
139:                   break;
140:                   }
141:                default:
142:                   break;
143:             }
144:          }catch (ydc_exn& exn) {
145:             cout << exn.what() << endl;
146:          }
147:       }
```

```
148:    }catch (ydc_quit&) {
149:       // Intentionally left empty.
150:    }
151:    return sys_info::status();
152: }
153:
```

```
# $Id: Makefile,v 1.9 2014-04-09 17:05:13-07 - - $

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.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
```

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

```
1: $Id: README,v 1.2 2011-01-18 22:18:39-08 - - $
2:
```

```
1: # Makefile.dep created Fri Apr 11 12:40:18 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
```