

```
1: // $Id: bigint.h,v 1.2 2012-04-10 21:02:20-07 - - $
2:
3: #ifndef __BIGINT_H__
4: #define __BIGINT_H__
5:
6: #include <exception>
7: #include <iostream>
8: #include <utility>
9:
10: using namespace std;
11:
12: #include "trace.h"
13:
14: class bigint;
15: typedef pair <bigint, bigint> bigpair;
16:
17: //
18: // Operators with a left side of int.
19: //
20: bigint operator+ (int left, const bigint &that);
21: bigint operator- (int left, const bigint &that);
22: bigint operator* (int left, const bigint &that);
23: bigint operator/ (int left, const bigint &that);
24: bigint operator% (int left, const bigint &that);
25: bool operator== (int left, const bigint &that);
26: bool operator!= (int left, const bigint &that);
27: bool operator< (int left, const bigint &that);
28: bool operator<= (int left, const bigint &that);
29: bool operator> (int left, const bigint &that);
30: bool operator>= (int left, const bigint &that);
31:
```

```
32:
33: //
34: // Define class bigint
35: //
36: class bigint {
37:     friend ostream &operator<< (ostream &, const bigint &);
38:     private:
39:         int small_value;
40:         bigpair div_rem (const bigint &that) const;
41:         int compare (const bigint &that) const;
42:         int abscompare (const bigint &that) const;
43:         bigint mul_by_2 ();
44:     public:
45:         //
46:         // Override implicit members.
47:         //
48:         bigint ();
49:         bigint (const bigint &that);
50:         bigint &operator= (const bigint &that);
51:         ~bigint ();
52:         //
53:         // Extra ctors to make bigints.
54:         //
55:         bigint (const int that);
56:         bigint (const string &that);
57:         //
58:         // Basic add/sub operators.
59:         //
60:         bigint operator+ (const bigint &that) const;
61:         bigint operator- (const bigint &that) const;
62:         bigint operator- () const;
63:         int smallint () const;
64:         //
65:         // Extended operators implemented with add/sub.
66:         //
67:         bigint operator* (const bigint &that) const;
68:         bigint operator/ (const bigint &that) const;
69:         bigint operator% (const bigint &that) const;
70:         bigint pow (const bigint &that) const;
71:         //
72:         // Comparison operators.
73:         //
74:         bool operator== (const bigint &that) const;
75:         bool operator!= (const bigint &that) const;
76:         bool operator< (const bigint &that) const;
77:         bool operator<= (const bigint &that) const;
78:         bool operator> (const bigint &that) const;
79:         bool operator>= (const bigint &that) const;
80: };
81:
82: #endif
83:
```

```
1: // $Id: scanner.h,v 1.1 2011-01-18 22:17:09-08 - - $
2:
3: #ifndef __SCANNER_H__
4: #define __SCANNER_H__
5:
6: #include <iostream>
7: #include <utility>
8:
9: using namespace std;
10:
11: #include "trace.h"
12:
13: enum terminal_symbol {NUMBER, OPERATOR, SCANEOF};
14: struct token_t {
15:     terminal_symbol symbol;
16:     string lexinfo;
17: };
18:
19: class scanner {
20:     private:
21:         bool seen_eof;
22:         char lookahead;
23:         void advance();
24:     public:
25:         scanner();
26:         token_t scan ();
27: };
28:
29: ostream &operator<< (ostream &, const terminal_symbol &);
30: ostream &operator<< (ostream &, const token_t &);
31:
32: #endif
33:
```

```
1: // $Id: trace.h,v 1.1 2011-01-18 22:17:09-08 - - $
2:
3: #ifndef __TRACE_H__
4: #define __TRACE_H__
5:
6: #include <iostream>
7: #include <string>
8: #include <vector>
9:
10: using namespace std;
11:
12: //
13: // traceflags -
14: //     static class for maintaining global trace flags, each indicated
15: //     by a single character.
16: // setflags -
17: //     Takes a string argument, and sets a flag for each char in the
18: //     string. As a special case, '@', sets all flags.
19: // getflag -
20: //     Used by the TRACE macro to check to see if a flag has been set.
21: //     Not to be called by user code.
22: //
23:
24: class traceflags {
25:     private:
26:         static vector<char> flags;
27:     public:
28:         static void setflags (const string &optflags);
29:         static bool getflag (char flag);
30: };
31:
32: //
33: // TRACE -
34: //     Macro which expands into trace code. First argument is a
35: //     trace flag char, second argument is output code that can
36: //     be sandwiched between <<. Beware of operator precedence.
37: //     Example:
38: //         TRACE ('u', "foo = " << foo);
39: //     will print two words and a newline if flag 'u' is on.
40: //     Traces are preceded by filename, line number, and function.
41: //
42:
43: #define TRACE(FLAG, CODE) { \
44:     if (traceflags::getflag (FLAG)) { \
45:         cerr << __FILE__ << ":" << __LINE__ << ":" \
46:             << __func__ << ":" << endl; \
47:         cerr << CODE << endl; \
48:     } \
49: }
50:
51: #endif
52:
```

```
1: // $Id: util.h,v 1.2 2012-04-10 21:02:20-07 - - $
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 <vector>
14:
15: #ifdef __GNUC__
16: #include <stdexcept>
17: #endif
18:
19: using namespace std;
20:
21: #include "trace.h"
22:
23: //
24: // ydc_exn -
25: //      Indicate a problem where processing should be abandoned and
26: //      the main function should take control.
27: //
28:
29: class ydc_exn: public runtime_error {
30:     public:
31:         explicit ydc_exn (const string &what);
32: };
33:
34: //
35: // octal -
36: //      Convert integer to octal string.
37: //
38:
39: const string octal (int decimal);
40:
```

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

```
1: // $Id: iterstack.h,v 1.6 2012-04-12 19:32:54-07 - - $
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:
18: #ifndef __ITERSTACK_H__
19: #define __ITERSTACK_H__
20:
21: #include <list>
22:
23: using namespace std;
24:
25: template <typename value_type>
26: class iterstack: private list<value_type> {
27:     public:
28:         using list<value_type>::const_reference;
29:         using list<value_type>::const_reverse_iterator;
30:         using list<value_type>::push_back;
31:         using list<value_type>::pop_back;
32:         using list<value_type>::clear;
33:         using list<value_type>::back;
34:         using list<value_type>::size;
35:         using list<value_type>::empty;
36:         using list<value_type>::rbegin;
37:         using list<value_type>::rend;
38:         inline void push (const value_type &value) { push_back (value); }
39:         inline void pop() { pop_back(); }
40:         inline value_type& top() { return back(); }
41:         inline const value_type& top() const { return back(); }
42: };
43:
44: #endif
45:
```

```
1: // $Id: bigint.cpp,v 1.3 2012-04-10 21:02:20-07 - - $
2:
3: #include <cstdlib>
4: #include <exception>
5: #include <limits>
6: #include <stack>
7: #include <stdexcept>
8:
9: using namespace std;
10:
11: #include "bigint.h"
12: #include "trace.h"
13:
14: #define CDTOR_TRACE TRACE ('~', this << " -> " << small_value)
15:
16: bigint::bigint (): small_value (0) {
17:     CDTOR_TRACE;
18: }
19:
20: bigint::bigint (const bigint &that): small_value (that.small_value) {
21:     *this = that;
22:     CDTOR_TRACE;
23: }
24:
25: bigint &bigint::operator= (const bigint &that) {
26:     if (this == &that) return *this;
27:     this->small_value = that.small_value;
28:     return *this;
29: }
30:
31: bigint::~~bigint() {
32:     CDTOR_TRACE;
33: }
34:
35: bigint::bigint (int that): small_value (that) {
36:     CDTOR_TRACE;
37: }
38:
39: bigint::bigint (const string &that) {
40:     string::const_iterator itor = that.begin();
41:     string::const_iterator end = that.end();
42:     bool isnegative = false;
43:     if (*itor == '_') {isnegative = true; ++itor; }
44:     int newval = 0;
45:     for (; itor != end; ++itor) newval = newval * 10 + *itor - '0';
46:     small_value = isnegative ? - newval : + newval;
47:     CDTOR_TRACE;
48: }
49:
```



```
50:
51: bigint bigint::operator+ (const bigint &that) const {
52:     return this->small_value + that.small_value;
53: }
54:
55: bigint bigint::operator- (const bigint &that) const {
56:     return this->small_value - that.small_value;
57: }
58:
59: bigint bigint::operator- () const {
60:     return -small_value;
61: }
62:
63: int bigint::compare (const bigint &that) const {
64:     return this->small_value < that.small_value ? -1
65:         : this->small_value > that.small_value ? +1 : 0;
66: }
67:
68: int bigint::abscompare (const bigint &that) const {
69:     return abs (this->small_value) < abs (that.small_value) ? -1
70:         : abs (this->small_value) > abs (that.small_value) ? +1 : 0;
71: }
72:
73: int bigint::smallint () const {
74:     if (*this < numeric_limits<int>::min()
75:         || *this > numeric_limits<int>::max())
76:         throw range_error ("smallint: out of range");
77:     return small_value;
78: }
79:
80: bigint bigint::mul_by_2 () {
81:     return this->small_value *= 2;
82: }
83:
84: static bigpair popstack (stack <bigpair> &egyptstack) {
85:     bigpair result = egyptstack.top ();
86:     egyptstack.pop();
87:     return result;
88: }
89:
```

```
90:
91: //
92: // Ancient Egyptian multiplication algorithm.
93: //
94: bigint bigint::operator* (const bigint &that) const {
95:     bigint top = that;
96:     bigint count = 1;
97:     TRACE ('*', *this << " * " << that);
98:     stack <bigpair> egyptstack;
99:     bigint a; // junk code -- delete
100:    bigint b; // junk code -- delete
101:    egyptstack.push (pair<bigint, bigint> (a, b)); // junk code -- delete
102:    popstack (egyptstack); // junk to suppress a warning
103:    bigint result = 0;
104:    if ((*this < 0) != (that < 0)) result = - result;
105:    return result;
106: }
107:
108: //
109: // Ancient Egyptian division algorithm.
110: //
111: bigpair bigint::div_rem (const bigint &that) const {
112:     if (that == 0) throw range_error ("divide by 0");
113:     bigint count = 1;
114:     bigint top = abs (that.small_value);
115:     TRACE ('/', *this << " /% " << that);
116:     stack <bigpair> egyptstack;
117:     bigint quotient = 0;
118:     bigint remainder = abs (this->small_value);
119:     return bigpair (quotient, remainder);
120: }
121:
122: bigint bigint::operator/ (const bigint &that) const {
123:     return div_rem (that).first;
124: }
125:
126: bigint bigint::operator% (const bigint &that) const {
127:     return div_rem (that).second;
128: }
129:
```

```
130:
131: #define TRACE_POW TRACE ('^', "result: " << result \
132:     << ", base: " << base << ", expt: " << expt);
133: bigint bigint::pow (const bigint &that) const {
134:     bigint base = *this;
135:     if (that > 999) throw range_error ("exp too big");
136:     int expt = that.smallint();
137:     bigint result = 1;
138:     TRACE_POW;
139:     if (expt < 0) {
140:         base = 1 / base;
141:         expt = - expt;
142:     }
143:     while (expt > 0) {
144:         TRACE_POW;
145:         if (expt & 1) { //odd
146:             result = result * base;
147:             --expt;
148:         }else { //even
149:             base = base * base;
150:             expt /= 2;
151:         }
152:     }
153:     TRACE_POW;
154:     return result;
155: }
156:
157: //
158: // Macros can make repetitive code easier.
159: //
160:
161: #define COMPARE(OPER) \
162:     bool bigint::operator OPER (const bigint &that) const { \
163:         return compare (that) OPER 0; \
164:     }
165: COMPARE (==)
166: COMPARE (!=)
167: COMPARE (< )
168: COMPARE (<=)
169: COMPARE (> )
170: COMPARE (>=)
171:
172: #define INT_LEFT(RESULT,OPER) \
173:     RESULT operator OPER (int left, const bigint &that) { \
174:         return bigint (left) OPER that; \
175:     }
176: INT_LEFT (bigint, +)
177: INT_LEFT (bigint, -)
178: INT_LEFT (bigint, *)
179: INT_LEFT (bigint, /)
180: INT_LEFT (bigint, %)
181: INT_LEFT (bool, ==)
182: INT_LEFT (bool, !=)
183: INT_LEFT (bool, < )
184: INT_LEFT (bool, <=)
185: INT_LEFT (bool, > )
186: INT_LEFT (bool, >=)
187:
188: ostream &operator<< (ostream &out, const bigint &that) {
189:     out << that.small_value;
190:     return out;
191: }
192:
```

```
1: // $Id: scanner.cpp,v 1.2 2012-04-10 20:46:21-07 - - $
2:
3: #include <iostream>
4: #include <locale>
5:
6: using namespace std;
7:
8: #include "scanner.h"
9: #include "trace.h"
10:
11: scanner::scanner () {
12:     seen_eof = false;
13:     advance();
14: }
15:
16: void scanner::advance () {
17:     if (! seen_eof) {
18:         cin.get (lookahead);
19:         if (cin.eof()) seen_eof = true;
20:     }
21: }
22:
23: token_t scanner::scan() {
24:     token_t result;
25:     while (!seen_eof && isspace (lookahead)) advance();
26:     if (seen_eof) {
27:         result.symbol = SCANEOF;
28:     }else if (lookahead == '_' || isdigit (lookahead)) {
29:         result.symbol = NUMBER;
30:         do {
31:             result.lexinfo += lookahead;
32:             advance();
33:         }while (!seen_eof && isdigit (lookahead));
34:     }else {
35:         result.symbol = OPERATOR;
36:         result.lexinfo += lookahead;
37:         advance();
38:     }
39:     TRACE ('S', result);
40:     return result;
41: }
42:
43: ostream &operator<< (ostream &out, const terminal_symbol &symbol) {
44:     switch (symbol) {
45:         case NUMBER : out << "NUMBER" ; break;
46:         case OPERATOR: out << "OPERATOR"; break;
47:         case SCANEOF : out << "SCANEOF" ; break;
48:     }
49:     return out;
50: }
51:
52: ostream &operator<< (ostream &out, const token_t &token) {
53:     out << token.symbol << ": \" " << token.lexinfo << "\" ";
54:     return out;
55: }
56:
```

```
1: // $Id: trace.cpp,v 1.1 2012-04-10 20:43:21-07 - - $
2:
3: #include <climits>
4: #include <vector>
5:
6: using namespace std;
7:
8: #include "trace.h"
9:
10: //
11: // ** BUG IN STL ** BUG IN STL **
12: // We should use vector<bool> instead of vector<char>,
13: // but vector<bool> has a bug:
14: // http://forums.sun.com/thread.jspa?threadID=5277939
15: // Static linking works, but doubles the size of the executable
16: // image.
17: // ** BUG IN STL ** BUG IN STL **
18: //
19:
20: typedef vector<char> boolvec;
21: boolvec traceflags::flags (UCHAR_MAX + 1, false);
22: const boolvec trueflags (UCHAR_MAX + 1, true);
23:
24: void traceflags::setflags (const string &optflags) {
25:     string::const_iterator itor = optflags.begin();
26:     string::const_iterator end = optflags.end();
27:     for (; itor != end; ++itor) {
28:         if (*itor == '@') {
29:             flags = trueflags;
30:         } else {
31:             flags[*itor] = true;
32:         }
33:     }
34:     // Note that TRACE can trace setflags.
35:     TRACE ('t', "optflags = " << optflags);
36: }
37:
38: //
39: // getflag -
40: //     Check to see if a certain flag is on.
41: //
42:
43: bool traceflags::getflag (char flag) {
44:     // WARNING: Don't TRACE this function or the stack will blow up.
45:     bool result = flags[flag];
46:     return result;
47: }
48:
```

```
1: // $Id: util.cpp,v 1.1 2012-04-10 20:43:21-07 - - $
2:
3: #include <cstdlib>
4: #include <sstream>
5:
6: using namespace std;
7:
8: #include "util.h"
9:
10: ydc_exn::ydc_exn (const string &what): runtime_error (what) {
11: }
12:
13: const string octal (int decimal) {
14:     ostringstream ostring;
15:     ostring.setf (ios::oct);
16:     ostring << decimal;
17:     return ostring.str ();
18: }
19:
20: int sys_info::exit_status = EXIT_SUCCESS;
21: string sys_info::execname; // Must be initialized from main().
22:
23: void sys_info::set_execname (const string &argv0) {
24:     execname = argv0;
25:     cout << boolalpha;
26:     cerr << boolalpha;
27:     TRACE ('Y', "execname = " << execname);
28: }
29:
30: ostream &complain() {
31:     sys_info::set_status (EXIT_FAILURE);
32:     cerr << sys_info::get_execname () << ": ";
33:     return cerr;
34: }
35:
36: template <typename item_t>
37: ostream &operator<< (ostream &out, const vector<item_t> &vec) {
38:     typename vector<item_t>::const_iterator itor = vec.begin();
39:     typename vector<item_t>::const_iterator end = vec.end();
40:
41:     // If the vector is empty, do nothing.
42:     if (itor != end) {
43:         // Print out the first element without a space.
44:         out << *itor++;
45:         // Print out the rest of the elements each preceded by a space.
46:         while (itor != end) out << " " << *itor++;
47:     }
48:     return out;
49: }
50:
```

```
1: // $Id: main.cpp,v 1.8 2012-04-12 19:31:01-07 - - $
2:
3: #include <deque>
4: #include <exception>
5: #include <map>
6: #include <iostream>
7: #include <utility>
8:
9: using namespace std;
10:
11: #include "bigint.h"
12: #include "iterstack.h"
13: #include "util.h"
14: #include "scanner.h"
15: #include "trace.h"
16:
17: typedef iterstack<bigint> bigint_stack;
18:
19: #define DO_BINOP(FN_NAME,TFLAG,OPER) \
20:     void FN_NAME (bigint_stack &stack) { \
21:         bigint right = stack.top(); \
22:         stack.pop(); \
23:         TRACE (TFLAG, "right = " << right); \
24:         bigint left = stack.top(); \
25:         stack.pop(); \
26:         TRACE (TFLAG, "left = " << left); \
27:         bigint result = left OPER (right); \
28:         TRACE (TFLAG, "result = " << result); \
29:         stack.push (result); \
30:     }
31: DO_BINOP(do_add, '+', + )
32: DO_BINOP(do_sub, '-', - )
33: DO_BINOP(do_mul, '*', * )
34: DO_BINOP(do_div, '/', / )
35: DO_BINOP(do_rem, '%', % )
36: DO_BINOP(do_pow, '^', .pow)
37:
38: void do_clear (bigint_stack &stack) {
39:     TRACE ('c', "");
40:     stack.clear();
41: }
42:
43: void do_dup (bigint_stack &stack) {
44:     bigint top = stack.top();
45:     TRACE ('d', top);
46:     stack.push (top);
47: }
48:
49: void do_printall (bigint_stack &stack) {
50:     bigint_stack::const_reverse_iterator itor = stack.rbegin();
51:     bigint_stack::const_reverse_iterator end = stack.rend();
52:     for (; itor != end; ++itor) cout << *itor << endl;
53: }
54:
55: void do_print (bigint_stack &stack) {
56:     cout << stack.top() << endl;
57: }
58:
59: void do_debug (bigint_stack &stack) {
60:     (void) stack; // SUPPRESS: warning: unused parameter 'stack'
61:     cout << "Y not implemented" << endl;
62: }
63:
```

```
64:
65: class ydc_quit: public exception {};
66: void do_quit (bigint_stack &stack) {
67:     (void) stack; // SUPPRESS: warning: unused parameter 'stack'
68:     throw ydc_quit ();
69: }
70:
71: typedef void (*function) (bigint_stack&);
72: typedef map <string, function> fnmap;
73: fnmap load_fn () {
74:     fnmap functions;
75:     functions["+"] = do_add;
76:     functions["-"] = do_sub;
77:     functions["*"] = do_mul;
78:     functions["/"] = do_div;
79:     functions["%"] = do_rem;
80:     functions["^"] = do_pow;
81:     functions["Y"] = do_debug;
82:     functions["c"] = do_clear;
83:     functions["d"] = do_dup;
84:     functions["f"] = do_printall;
85:     functions["p"] = do_print;
86:     functions["q"] = do_quit;;
87:     return functions;
88: }
89:
90: //
91: // scan_options
92: //   Options analysis:  The only option is -Dflags.
93: //
94:
95: void scan_options (int argc, char **argv) {
96:     opterr = 0;
97:     for (;;) {
98:         int option = getopt (argc, argv, "@:");
99:         if (option == EOF) break;
100:         switch (option) {
101:             case '@':
102:                 traceflags::setflags (optarg);
103:                 break;
104:             default:
105:                 complain() << "-" << (char) optopt << ": invalid option"
106:                     << endl;
107:                 break;
108:         }
109:     }
110:     if (optind < argc) {
111:         complain() << "operand not permitted" << endl;
112:     }
113: }
114:
```



```
115:
116: int main (int argc, char **argv) {
117:     sys_info::set_execname (argv[0]);
118:     scan_options (argc, argv);
119:     fnmap functions = load_fn();
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:                         function fn = functions[token.lexinfo];
133:                         if (fn == NULL) {
134:                             throw ydc_exn (octal (token.lexinfo[0])
135:                                             + " is unimplemented");
136:                         }
137:                         fn (operand_stack);
138:                         break;
139:                     }
140:                     default:
141:                         break;
142:                 }
143:             } catch (ydc_exn &exn) {
144:                 cout << exn.what() << endl;
145:             }
146:         }
147:     } catch (ydc_quit &) {
148:         // Intentionally left empty.
149:     }
150:     return sys_info::get_status ();
151: }
152:
```

```
1: # $Id: Makefile,v 1.4 2012-04-10 20:43:21-07 - - $
2:
3: MKFILE      = Makefile
4: DEPPFILE    = ${MKFILE}.dep
5: NOINCL      = ci clean spotless
6: NEEDINCL    = ${filter ${NOINCL}, ${MAKECMDGOALS}}
7: GMAKE       = ${MAKE} --no-print-directory
8:
9: COMPILECPP  = g++ -g -O0 -Wall -Wextra -Werror
10: MAKEDEPCPP = g++ -MM
11:
12: CPPHEADER   = bigint.h scanner.h trace.h util.h iterstack.h
13: CPPSOURCE   = bigint.cpp scanner.cpp trace.cpp util.cpp main.cpp
14: EXECBIN     = ydc
15: OBJECTS     = ${CPPSOURCE:.cpp=.o}
16: OTHERS      = ${MKFILE} README
17: ALLSOURCES  = ${CPPHEADER} ${CPPSOURCE} ${OTHERS}
18: LISTING     = Listing.code.ps
19: CLASS       = cmpls109-wm.sl2
20: PROJECT     = asg2
21:
22: all : ${EXECBIN}
23:     - checksource ${ALLSOURCES}
24:
25: ${EXECBIN} : ${OBJECTS}
26:     ${COMPILECPP} -o $@ ${OBJECTS}
27:
28: %.o : %.cpp
29:     cid + $<
30:     ${COMPILECPP} -c $<
31:
32: ci : ${ALLSOURCES}
33:     - checksource ${ALLSOURCES}
34:     cid + ${ALLSOURCES}
35:
36: lis : ${ALLSOURCES}
37:     mkpspdf ${LISTING} ${ALLSOURCES} ${DEPPFILE}
38:
39: clean :
40:     - rm ${OBJECTS} ${DEPPFILE} core ${EXECBIN}.errs
41:
42: spotless : clean
43:     - rm ${EXECBIN} ${LISTING}
44:
45: submit : ${ALLSOURCES}
46:     - checksource ${ALLSOURCES}
47:     submit ${CLASS} ${PROJECT} ${ALLSOURCES}
48:
49: dep : ${CPPSOURCE} ${CPPHEADER}
50:     @ echo "# ${DEPPFILE} created `LC_TIME=C date`" >${DEPPFILE}
51:     ${MAKEDEPCPP} ${CPPSOURCE} >>${DEPPFILE}
52:
53: ${DEPPFILE} :
54:     @ touch ${DEPPFILE}
55:     ${GMAKE} dep
56:
57: again :
58:     ${GMAKE} spotless dep ci all lis
59:
60: ifeq (${NEEDINCL}, )
61: include ${DEPPFILE}
62: endif
63:
```

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

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