```
 1: // $Id: bigint.h,v 1.1 2011-01-18 22:17:09-08 - - $
 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: class bigint {
18:       friend ostream &operator<< (ostream &, const bigint &);
19:    private:
20:       int small_value;
21:       bigpair div_rem (const bigint &that) const;
22:       int compare (const bigint &that) const;
23:       int abscompare (const bigint &that) const;
24:       bigint mul_by_2 ();
25:    public:
26:       //
27:       // Override implicit members.
28:       //
29:       bigint ();
30:       bigint (const bigint &that);
31:       bigint &operator= (const bigint &that);
32:       ~bigint ();
33:       //
34:       // Extra ctors to make bigints.
35:       //
36:       bigint (const int that);
37:       bigint (const string &that);
38:       //
39:       // Basic add/sub operators.
40:       //
41:       bigint operator+ (const bigint &that) const;
42:       bigint operator- (const bigint &that) const;
43:       bigint operator- () const;
44:       int smallint () const;
45:       //
46:       // Extended operators implemented with add/sub.
47:       //
48:       bigint operator* (const bigint &that) const;
49:       bigint operator/ (const bigint &that) const;
50:       bigint operator% (const bigint &that) const;
51:       bigint pow (const bigint &that) const;
52:       //
53:       // Comparison operators.
54:       //
55:       bool operator== (const bigint &that) const;
56:       bool operator!= (const bigint &that) const;
57:       bool operator<  (const bigint &that) const;
58:       bool operator<= (const bigint &that) const;
59:       bool operator>  (const bigint &that) const;
60:       bool operator>= (const bigint &that) const;
61: };
62:
63: //
64: // Operators with a left side of int.
```

```
65: //
66: bigint operator+ (int left, const bigint &that);
67: bigint operator- (int left, const bigint &that);
68: bigint operator* (int left, const bigint &that);
69: bigint operator/ (int left, const bigint &that);
70: bigint operator% (int left, const bigint &that);
71: bool operator== (int left, const bigint &that);
72: bool operator!= (int left, const bigint &that);
73: bool operator<  (int left, const bigint &that);
74: bool operator<= (int left, const bigint &that);
75: bool operator>  (int left, const bigint &that);
76: bool operator>= (int left, const bigint &that);
77:
78: #endif
79:
```

```
 1: // $Id: iterstack.h,v 1.1 2011-01-18 22:17:09-08 - - $
 2:
 3: //
 4: // The class std::stack does not provide an iterator, which is needed
 5: // for this class.  So, like std::stack, class iterstack is implemented
 6: // on top of a std::deque.  We don't use a deque directly because we
 7: // want to restrict operations.
 8: //
 9: // All functions are merely forwarded to the deque as inline functions
10: // for efficiency.  For detailed documentation of the functions see
11: // std::deque.
12: //
13: // No implementation file is needed because all functions are inline.
14: // Inline functions are only a good idea for trivial forwarding
15: // functions.
16: //
17:
18: #ifndef __ITERSTACK_H__
19: #define __ITERSTACK_H__
20:
21: #include <deque>
22:
23: using namespace std;
24:
25: template <typename item_t>
26: class iterstack {
27:    private:
28:       deque<item_t> data;
29:    public:
30:       typedef typename deque<item_t>::const_reference const_reference;
31:       typedef typename deque<item_t>::const_iterator const_iterator;
32:       void push_front (const item_t &item) { data.push_front (item); }
33:       void pop_front ()                    { data.pop_front (); }
34:       void clear ()                        { data.clear (); }
35:       const_reference front () const       { return data.front (); }
36:       size_t size () const                 { return data.size (); }
37:       bool empty () const                  { return data.empty (); }
38:       const_iterator begin () const        { return data.begin (); }
39:       const_iterator end () const          { return data.end (); }
40: };
41:
42: #endif
43:
```

```
 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:
```

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

//
// 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:
```

```
65: //       complain() << filename << ": some problem" << endl;
66: //
67:
68: ostream &complain();
69:
70: //
71: // operator<< (vector) -
72: //    An overloaded template operator which allows vectors to be
73: //    printed out as a single operator, each element separated from
74: //    the next with spaces.  The item_t must have an output operator
75: //    defined for it.
76: //
77:
78: template <typename item_t>
79: ostream &operator<< (ostream &out, const vector<item_t> &vec);
80:
81: #endif
82:
```

```
 1: // $Id: main.cc,v 1.1 2011-01-18 22:17:09-08 - - $
 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.front(); \
22:       stack.pop_front(); \
23:       TRACE (TFLAG, "right = " << right); \
24:       bigint left = stack.front(); \
25:       stack.pop_front(); \
26:       TRACE (TFLAG, "left = " << left); \
27:       bigint result = left OPER (right); \
28:       TRACE (TFLAG, "result = " << result); \
29:       stack.push_front (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.front();
45:    TRACE ('d', top);
46:    stack.push_front (top);
47: }
48:
49: void do_printall (bigint_stack &stack) {
50:    bigint_stack::const_iterator itor = stack.begin();
51:    bigint_stack::const_iterator end = stack.end();
52:    for (; itor != end; ++itor) cout << *itor << endl;
53: }
54:
55: void do_print (bigint_stack &stack) {
56:    cout << stack.front() << 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: class ydc_quit: public exception {};
```

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

```
129:                case OPERATOR: {
130:                   function fn = functions[token.lexinfo];
131:                   if (fn == NULL) {
132:                      throw ydc_exn (octal (token.lexinfo[0])
133:                                     + " is unimplemented");
134:                   }
135:                   fn (operand_stack);
136:                   break;
137:                   }
138:                default:
139:                   break;
140:             }
141:          }catch (ydc_exn exn) {
142:             cout << exn.what() << endl;
143:          }
144:       }
145:    }catch (ydc_quit) {
146:    }
147:    return sys_info::get_status ();
148: }
149:
```

```
// $Id: bigint.cc,v 1.2 2011-01-18 22:18:35-08 - - $

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

using namespace std;

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

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

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

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

bigint::~bigint() {
   TRACE ('~', cout << *this);
}

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

bigint::bigint (const string &that) {
   TRACE ('b', 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;
}

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;
```

```
 65: }
 66:
 67: int bigint::smallint () const {
 68:    if (*this < numeric_limits<int>::min()
 69:     || *this > numeric_limits<int>::max())
 70:                throw range_error ("smallint: out of range");
 71:    return small_value;
 72: }
 73:
 74: bigint bigint::mul_by_2 () {
 75:    return this->small_value *= 2;
 76: }
 77:
 78: static bigpair popstack (stack <bigpair> &egyptstack) {
 79:    bigpair result = egyptstack.top ();
 80:    egyptstack.pop();
 81:    return result;
 82: }
 83:
 84: //
 85: // Ancient Egyptian multiplication algorithm.
 86: //
 87: bigint bigint::operator* (const bigint &that) const {
 88:    bigint top = that;
 89:    bigint count = 1;
 90:    TRACE ('*', *this << " * " << that);
 91:    stack <bigpair> egyptstack;
 92:    popstack (egyptstack); // junk to suppress a warning
 93:    bigint result = 0;
 94:    if ((*this < 0) != (that < 0)) result = - result;
 95:    return result;
 96: }
 97:
 98: //
 99: // Ancient Egyptian division algorithm.
100: //
101: bigpair bigint::div_rem (const bigint &that) const {
102:    if (that == 0) throw range_error ("divide by 0");
103:    bigint count = 1;
104:    bigint top = abs (that.small_value);
105:    TRACE ('/', *this << " /% " << that);
106:    stack <bigpair> egyptstack;
107:    bigint quotient = 0;
108:    bigint remainder = abs (this->small_value);
109:    return bigpair (quotient, remainder);
110: }
111:
112: bigint bigint::operator/ (const bigint &that) const {
113:    return div_rem (that).first;
114: }
115:
116: bigint bigint::operator% (const bigint &that) const {
117:    return div_rem (that).second;
118: }
119:
120: #define TRACE_POW \
121:    TRACE ('^', "result: " << result << ", base: " << base \
122:               << ", expt: " << expt);
123: bigint bigint::pow (const bigint &that) const {
124:    bigint base = *this;
125:    if (that > 999) throw range_error ("exp too big");
126:    int expt = that.smallint();
127:    bigint result = 1;
128:    TRACE_POW;
```

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

```
 1: // $Id: scanner.cc,v 1.1 2011-01-18 22:17:09-08 - - $
 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:       #define CASE_SYMBOL(SYMBOL) case SYMBOL: out << #SYMBOL; break;
46:       CASE_SYMBOL (NUMBER);
47:       CASE_SYMBOL (OPERATOR);
48:       CASE_SYMBOL (SCANEOF);
49:    }
50:    return out;
51: }
52:
53: ostream &operator<< (ostream &out, const token_t &token) {
54:    out << token.symbol << ": \"" << token.lexinfo << "\"";
55:    return out;
56: }
57:
```

```
 1: // $Id: trace.cc,v 1.1 2011-01-18 22:17:09-08 - - $
 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.cc,v 1.1 2011-01-18 22:17:09-08 - - $
 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: Makefile,v 1.2 2011-01-18 22:19:23-08 - - $
 2:
 3: MKFILE      = Makefile
 4: DEPSFILE    = ${MKFILE}.deps
 5: NOINCL      = ci clean spotless
 6: NEEDINCL    = ${filter ${NOINCL}, ${MAKECMDGOALS}}
 7: GMAKE       = ${MAKE} --no-print-directory
 8: UNAME      ?= ${shell uname -s}
 9:
10: ifeq (${UNAME},SunOS)
11: COMPILECCC  = CC -g -features=extensions
12: MAKEDEPSCCC = CC -xM1
13: endif
14: ifeq (${UNAME},Linux)
15: COMPILECCC  = g++ -g -Wall -Wextra -Werror
16: MAKEDEPSCCC = g++ -MM
17: endif
18:
19: CCHEADER    =         bigint.h iterstack.h scanner.h  trace.h  util.h
20: CCSOURCE    = main.cc bigint.cc            scanner.cc trace.cc util.cc
21: EXECBIN     = ydc
22: OBJECTS     = ${CCSOURCE:.cc=.o}
23: OTHERS      = ${MKFILE} ${DEPSFILE} README
24: ALLSOURCES  = ${CCHEADER} ${CCSOURCE} ${OTHERS}
25: LISTING     = ../asg2-ydc.code.ps
26: CLASS       = cmps109-wm.w11
27: PROJECT     = asg2
28:
29: all : ${EXECBIN}
30:         - checksource ${ALLSOURCES}
31:
32: ${EXECBIN} : ${OBJECTS}
33:         ${COMPILECCC} -o $@ ${OBJECTS}
34:
35: %.o : %.cc
36:         cid + $<
37:         ${COMPILECCC} -c $<
38:
39: ci : ${ALLSOURCES}
40:         - checksource ${ALLSOURCES}
41:         cid + ${ALLSOURCES}
42:
43: lis : ${ALLSOURCES}
44:         mkpspdf ${LISTING} ${ALLSOURCES} ${DEPSFILE}
45:
46: clean :
47:         - rm ${OBJECTS} ${DEPSFILE} core ${EXECBIN}.errs
48:
49: spotless : clean
50:         - rm ${EXECBIN}
51:
52: submit : ${ALLSOURCES}
53:         - checksource ${ALLSOURCES}
54:         submit ${CLASS} ${PROJECT} ${ALLSOURCES}
55:         testsubmit ${CLASS} ${PROJECT} ${ALLSOURCES}
56:
57: deps : ${CCSOURCE} ${CCHEADER}
58:         @ echo "# ${DEPSFILE} created ‘LC_TIME=C date‘" >${DEPSFILE}
59:         ${MAKEDEPSCCC} ${CCSOURCE} | sort | uniq >>${DEPSFILE}
60:
61: ${DEPSFILE} :
62:         @ touch ${DEPSFILE}
63:         ${GMAKE} deps
64:
```

```
65: again :
66:         ${GMAKE} spotless deps ci all lis
67:
68: ifeq (${NEEDINCL}, )
69: include ${DEPSFILE}
70: endif
71:
```

```
1: # Makefile.deps created Tue Jan 18 22:19:22 PST 2011
2: bigint.o: bigint.cc bigint.h trace.h
3: main.o: main.cc bigint.h trace.h iterstack.h util.h scanner.h
4: scanner.o: scanner.cc scanner.h trace.h
5: trace.o: trace.cc trace.h
6: util.o: util.cc util.h trace.h
```

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

```
1: # Makefile.deps created Tue Jan 18 22:19:22 PST 2011
2: bigint.o: bigint.cc bigint.h trace.h
3: main.o: main.cc bigint.h trace.h iterstack.h util.h scanner.h
4: scanner.o: scanner.cc scanner.h trace.h
5: trace.o: trace.cc trace.h
6: util.o: util.cc util.h trace.h
```