

```
1: // $Id: bigint.h,v 1.16 2014-07-02 20:01:17-07 - - $
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:     using quot_rem = pair<bigint, bigint>;
21:     using unumber = unsigned long;
22:     friend quot_rem divide (const bigint&, const bigint&);
23:     friend void multiply_by_2 (unumber&);
24:     friend void divide_by_2 (unumber&);
25: public:
26:
27:     //
28:     // Ensure synthesized members are genrated.
29:     //
30:     bigint() = default;
31:     bigint (const bigint&) = default;
32:     bigint (bigint&&) = default;
33:     bigint& operator= (const bigint&) = default;
34:     bigint& operator= (bigint&&) = default;
35:     ~bigint() = default;
36:
37:     //
38:     // Extra ctors to make bigints.
39:     //
40:     bigint (const long);
41:     bigint (const string&);
42:
43:     //
44:     // Basic add/sub operators.
45:     //
46:     friend bigint operator+ (const bigint&, const bigint&);
47:     friend bigint operator- (const bigint&, const bigint&);
48:     friend bigint operator+ (const bigint&);
49:     friend bigint operator- (const bigint&);
50:     long to_long() const;
```

```
51:
52:     //
53:     // Extended operators implemented with add/sub.
54:     //
55:     friend bigint operator* (const bigint&, const bigint&);
56:     friend bigint operator/ (const bigint&, const bigint&);
57:     friend bigint operator% (const bigint&, const bigint&);
58:
59:     //
60:     // Comparison operators.
61:     //
62:     friend bool operator== (const bigint&, const bigint&);
63:     friend bool operator< (const bigint&, const bigint&);
64: };
65:
66: //
67: // The rest of the operators do not need to be friends.
68: // Make the comparisons inline for efficiency.
69: //
70:
71: bigint pow (const bigint& base, const bigint& exponent);
72:
73: inline bool operator!= (const bigint &left, const bigint &right) {
74:     return not (left == right);
75: }
76: inline bool operator> (const bigint &left, const bigint &right) {
77:     return right < left;
78: }
79: inline bool operator<= (const bigint &left, const bigint &right) {
80:     return not (right < left);
81: }
82: inline bool operator>= (const bigint &left, const bigint &right) {
83:     return not (left < right);
84: }
85:
86: #endif
87:
```

```
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:
```



```
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: #ifndef 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 1.13 2014-06-26 17:21:55-07 - - $
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: #endif
49:
```



```
1: // $Id: bigint.cpp,v 1.61 2014-06-26 17:06:06-07 - - $
2:
3: #include <cstdlib>
4: #include <exception>
5: #include <limits>
6: #include <stack>
7: #include <stdexcept>
8: using namespace std;
9:
10: #include "bigint.h"
11: #include "debug.h"
12:
13: bigint::bigint (long that): long_value (that) {
14:     DEBUGF ('~', this << " -> " << long_value)
15: }
16:
17: bigint::bigint (const string& that) {
18:     auto itor = that.cbegin();
19:     bool isnegative = false;
20:     if (itor != that.cend() and *itor == '_') {
21:         isnegative = true;
22:         ++itor;
23:     }
24:     int newval = 0;
25:     while (itor != that.end()) newval = newval * 10 + *itor++ - '0';
26:     long_value = isnegative ? - newval : + newval;
27:     DEBUGF ('~', this << " -> " << long_value)
28: }
29:
```

```
30:
31: bigint operator+ (const bigint& left, const bigint& right) {
32:     return left.long_value + right.long_value;
33: }
34:
35: bigint operator- (const bigint& left, const bigint& right) {
36:     return left.long_value - right.long_value;
37: }
38:
39: bigint operator+ (const bigint& right) {
40:     return +right.long_value;
41: }
42:
43: bigint operator- (const bigint& right) {
44:     return -right.long_value;
45: }
46:
47: long bigint::to_long() const {
48:     if (*this <= bigint (numeric_limits<long>::min()))
49:         or *this > bigint (numeric_limits<long>::max()))
50:         throw range_error ("bigint__to_long: out of range");
51:     return long_value;
52: }
53:
54: bool abs_less (const long& left, const long& right) {
55:     return left < right;
56: }
57:
58: //
59: // Multiplication algorithm.
60: //
61: bigint operator* (const bigint& left, const bigint& right) {
62:     return left.long_value * right.long_value;
63: }
64:
65: //
66: // Division algorithm.
67: //
68:
69: void multiply_by_2 (bigint::unumber& unumber_value) {
70:     unumber_value *= 2;
71: }
72:
73: void divide_by_2 (bigint::unumber& unumber_value) {
74:     unumber_value /= 2;
75: }
76:
```

```
77:
78: bigint::quot_rem divide (const bigint& left, const bigint& right) {
79:     if (right == 0) throw domain_error ("divide by 0");
80:     using unnumber = unsigned long;
81:     static unnumber zero = 0;
82:     if (right == 0) throw domain_error ("bigint::divide");
83:     unnumber divisor = right.long_value;
84:     unnumber quotient = 0;
85:     unnumber remainder = left.long_value;
86:     unnumber power_of_2 = 1;
87:     while (abs_less (divisor, remainder)) {
88:         multiply_by_2 (divisor);
89:         multiply_by_2 (power_of_2);
90:     }
91:     while (abs_less (zero, power_of_2)) {
92:         if (not abs_less (remainder, divisor)) {
93:             remainder = remainder - divisor;
94:             quotient = quotient + power_of_2;
95:         }
96:         divide_by_2 (divisor);
97:         divide_by_2 (power_of_2);
98:     }
99:     return {quotient, remainder};
100: }
101:
102: bigint operator/ (const bigint& left, const bigint& right) {
103:     return divide (left, right).first;
104: }
105:
106: bigint operator% (const bigint& left, const bigint& right) {
107:     return divide (left, right).second;
108: }
109:
110: bool operator== (const bigint& left, const bigint& right) {
111:     return left.long_value == right.long_value;
112: }
113:
114: bool operator< (const bigint& left, const bigint& right) {
115:     return left.long_value < right.long_value;
116: }
117:
118: ostream& operator<< (ostream& out, const bigint& that) {
119:     out << that.long_value;
120:     return out;
121: }
122:
```

```
123:
124: bigint pow (const bigint& base, const bigint& exponent) {
125:     DEBUGF ('^', "base = " << base << ", exponent = " << exponent);
126:     if (base == 0) return 0;
127:     bigint base_copy = base;
128:     long expt = exponent.to_long();
129:     bigint result = 1;
130:     if (expt < 0) {
131:         base_copy = 1 / base_copy;
132:         expt = - expt;
133:     }
134:     while (expt > 0) {
135:         if (expt & 1) { //odd
136:             result = result * base_copy;
137:             --expt;
138:         }else { //even
139:             base_copy = base_copy * base_copy;
140:             expt /= 2;
141:         }
142:     }
143:     DEBUGF ('^', "result = " << result);
144:     return result;
145: }
```

```
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 1.3 2014-06-26 16:51:09-07 - - $
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 1.10 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 1.41 2014-07-02 20:01:17-07 - - $
2:
3: #include <deque>
4: #include <iostream>
5: #include <map>
6: #include <stdexcept>
7: #include <utility>
8: using namespace std;
9:
10: #include <unistd.h>
11:
12: #include "bigint.h"
13: #include "debug.h"
14: #include "iterstack.h"
15: #include "scanner.h"
16: #include "util.h"
17:
18: using bigint_stack = iterstack<bigint>;
19:
20: void do_arith (bigint_stack& stack, const char oper) {
21:     if (stack.size() < 2) throw ydc_exn ("stack empty");
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: using function_t = void (*)(bigint_stack&, const char);
74: using fn_map = map<string, function_t>;
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:     if (sys_info::execname().size() == 0) sys_info::execname (argv[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: //
118: // Main function.
119: //
120:
121: int main (int argc, char** argv) {
122:     sys_info::execname (argv[0]);
123:     scan_options (argc, argv);
124:     bigint_stack operand_stack;
125:     scanner input;
126:     try {
127:         for (;;) {
128:             try {
129:                 token_t token = input.scan();
130:                 if (token.symbol == SCANEOF) break;
131:                 switch (token.symbol) {
132:                     case NUMBER:
133:                         operand_stack.push (token.lexinfo);
134:                         break;
135:                     case OPERATOR: {
136:                         fn_map::const_iterator fn
137:                             = do_functions.find (token.lexinfo);
138:                         if (fn == do_functions.end()) {
139:                             throw ydc_exn (octal (token.lexinfo[0])
140:                                             + " is unimplemented");
141:                         }
142:                         fn->second (operand_stack, token.lexinfo.at(0));
143:                         break;
144:                     }
145:                     default:
146:                         break;
147:                 }
148:             } catch (ydc_exn& exn) {
149:                 cout << exn.what() << endl;
150:             }
151:         }
152:     } catch (ydc_quit&) {
153:         // Intentionally left empty.
154:     }
155:     return sys_info::status();
156: }
157:
```

```
1: # $Id: Makefile,v 1.11 2014-07-02 20:01:17-07 - - $
2:
3: MKFILE      = Makefile
4: DEPFILE     = ${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 -std=gnu++11
10: MAKEDEPCPP  = g++ -MM
11:
12: CPPHEADER    = bigint.h scanner.h debug.h util.h iterstack.h
13: CPPSOURCE    = bigint.cpp scanner.cpp debug.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.ps
19:
20: all : ${EXECBIN}
21:     - checksource ${ALLSOURCES}
22:
23: ${EXECBIN} : ${OBJECTS}
24:     ${COMPILECPP} -o $@ ${OBJECTS}
25:
26: %.o : %.cpp
27:     ${COMPILECPP} -c $<
28:
29: ci : ${ALLSOURCES}
30:     - checksource ${ALLSOURCES}
31:     cid + ${ALLSOURCES}
32:
33: lis : ${ALLSOURCES}
34:     mkpspdf ${LISTING} ${ALLSOURCES} ${DEPFILE}
35:
36: clean :
37:     - rm ${OBJECTS} ${DEPFILE} core ${EXECBIN}.errs
38:
39: spotless : clean
40:     - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf}
41:
42: dep : ${CPPSOURCE} ${CPPHEADER}
43:     @ echo "# ${DEPFILE} created `LC_TIME=C date`" >${DEPFILE}
44:     ${MAKEDEPCPP} ${CPPSOURCE} >>${DEPFILE}
45:
46: ${DEPFILE} :
47:     @ touch ${DEPFILE}
48:     ${GMAKE} dep
49:
50: again :
51:     ${GMAKE} spotless dep ci all lis
52:
53: ifeq (${NEEDINCL}, )
54: include ${DEPFILE}
55: endif
56:
```

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

```
1: # Makefile.dep created Tue Apr 14 12:28:42 PDT 2015
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
```