

```
1: (* $Id: bigint.mli,v 1.1 2011-04-26 13:39:18-07 - - $ *)
2:
3: module Bigint : sig
4:   type bigint
5:   val bigint_of_string : string -> bigint
6:   val string_of_bigint : bigint -> string
7:   val add : bigint -> bigint -> bigint
8:   val sub : bigint -> bigint -> bigint
9:   val mul : bigint -> bigint -> bigint
10:  val div : bigint -> bigint -> bigint
11:  val rem : bigint -> bigint -> bigint
12:  val pow : bigint -> bigint -> bigint
13:  val zero : bigint
14: end
15:
```

```
1: (* $Id: bigint.ml,v 1.5 2014-11-11 15:06:24-08 - - $ *)
2:
3: open Printf
4:
5: module Bigint = struct
6:
7:     type sign      = Pos | Neg
8:     type bigint    = Bigint of sign * int list
9:     let radix      = 10
10:    let radixlen    = 1
11:
12:    let car         = List.hd
13:    let cdr         = List.tl
14:    let map         = List.map
15:    let reverse     = List.rev
16:    let strcat      = String.concat
17:    let strlen      = String.length
18:    let strsub      = String.sub
19:    let zero        = Bigint (Pos, [])
20:
21:    let charlist_of_string str =
22:        let last = strlen str - 1
23:        in let rec charlist pos result =
24:            if pos < 0
25:            then result
26:            else charlist (pos - 1) (str.[pos] :: result)
27:        in charlist last []
28:
29:    let bigint_of_string str =
30:        let len = strlen str
31:        in let to_intlist first =
32:            let substr = strsub str first (len - first) in
33:            let digit char = int_of_char char - int_of_char '0' in
34:            map digit (reverse (charlist_of_string substr))
35:        in if len = 0
36:           then zero
37:           else if str.[0] = '_'
38:              then Bigint (Neg, to_intlist 1)
39:              else Bigint (Pos, to_intlist 0)
40:
41:    let string_of_bigint (Bigint (sign, value)) =
42:        match value with
43:        | []      -> "0"
44:        | value  -> let reversed = reverse value
45:                    in strcat ""
46:                        ((if sign = Pos then "" else "-") ::
47:                         (map string_of_int reversed))
```

```
48:
49:   let rec add' list1 list2 carry = match (list1, list2, carry) with
50:     | list1, [], 0      -> list1
51:     | [], list2, 0      -> list2
52:     | list1, [], carry  -> add' list1 [carry] 0
53:     | [], list2, carry  -> add' [carry] list2 0
54:     | car1::cdr1, car2::cdr2, carry ->
55:       let sum = car1 + car2 + carry
56:       in  sum mod radix :: add' cdr1 cdr2 (sum / radix)
57:
58:   let add (Bigint (neg1, value1)) (Bigint (neg2, value2)) =
59:     if neg1 = neg2
60:     then Bigint (neg1, add' value1 value2 0)
61:     else zero
62:
63:   let sub = add
64:
65:   let mul = add
66:
67:   let div = add
68:
69:   let rem = add
70:
71:   let pow = add
72:
73: end
74:
```

```
1: (* $Id: maindc.ml,v 1.5 2017-04-07 13:24:41-07 - - $ *)
2:
3: include Scanner
4: include Bigint
5:
6: open Bigint
7: open Printf
8: open Scanner
9:
10: type stack_t = Bigint.bigint Stack.t
11: let push = Stack.push
12: let pop = Stack.pop
13:
14: let ord thechar = int_of_char thechar
15: type binop_t = bigint -> bigint -> bigint
16:
17: let print_number number = printf "%s\n%!" (string_of_bigint number)
18:
19: let print_stackempty () = printf "stack empty\n%!"
20:
21: let executereg (thetack: stack_t) (oper: char) (reg: int) =
22:   try match oper with
23:     | 'l' -> printf "operator l reg 0%o is unimplemented\n%!" reg
24:     | 's' -> printf "operator s reg 0%o is unimplemented\n%!" reg
25:     | _ -> printf "0%o 0%o is unimplemented\n%!" (ord oper) reg
26:   with Stack.Empty -> print_stackempty()
27:
28: let executebinop (thetack: stack_t) (oper: binop_t) =
29:   try let right = pop thetack
30:     in try let left = pop thetack
31:       in push (oper left right) thetack
32:     with Stack.Empty -> (print_stackempty ();
33:       push right thetack)
34:   with Stack.Empty -> print_stackempty ()
35:
36: let execute (thetack: stack_t) (oper: char) =
37:   try match oper with
38:     | '+' -> executebinop thetack Bigint.add
39:     | '-' -> executebinop thetack Bigint.sub
40:     | '*' -> executebinop thetack Bigint.mul
41:     | '/' -> executebinop thetack Bigint.div
42:     | '%' -> executebinop thetack Bigint.rem
43:     | '^' -> executebinop thetack Bigint.pow
44:     | 'c' -> Stack.clear thetack
45:     | 'd' -> push (Stack.top thetack) thetack
46:     | 'f' -> Stack.iter print_number thetack
47:     | 'l' -> failwith "operator l scanned with no register"
48:     | 'p' -> print_number (Stack.top thetack)
49:     | 'q' -> raise End_of_file
50:     | 's' -> failwith "operator s scanned with no register"
51:     | '\n' -> ()
52:     | ' ' -> ()
53:     | _ -> printf "0%o is unimplemented\n%!" (ord oper)
54:   with Stack.Empty -> print_stackempty()
```

```
55:
56: let toploop (thestack: stack_t) inputchannel =
57:   let scanbuf = Lexing.from_channel inputchannel in
58:   let rec toploop () =
59:     try let nexttoken = Scanner.scanner scanbuf
60:        in (match nexttoken with
61:           | Number number      -> push number thestack
62:           | Regoper (oper, reg) -> executereg thestack oper reg
63:           | Operator oper      -> execute thestack oper
64:           );
65:     toploop ()
66:   with End_of_file -> printf "End_of_file\n%!";
67:   in toploop ()
68:
69: let readfiles () =
70:   let thestack : bigint Stack.t = Stack.create ()
71:   in ((if Array.length Sys.argv > 1
72:       then try let thefile = open_in Sys.argv.(1)
73:          in toploop thestack thefile
74:       with Sys_error message -> (
75:         printf "%s: %s\n%!" Sys.argv.(0) message;
76:         exit 1));
77:   toploop thestack stdin)
78:
79: let interact () =
80:   let thestack : bigint Stack.t = Stack.create ()
81:   in toploop thestack stdin
82:
83: let _ = if not !Sys.interactive then readfiles ()
84:
```

```
1: (* $Id: scanner.mll,v 1.1 2011-04-26 13:39:18-07 - - $ *)
2:
3: {
4:
5: module Scanner = struct
6:   include Bigint
7:
8:   type token = Number    of Bigint.bigint
9:               | Regoper  of char * int
10:              | Operator of char
11:
12:   let bigstr = Bigint.bigint_of_string
13:   let lexeme = Lexing.lexeme
14:   let ord   = int_of_char
15:   let strlen = String.length
16:
17:   let regoper lexbuf =
18:     let token = lexeme lexbuf
19:     in Regoper (token.[0], ord token.[1])
20:
21: }
22:
23: let number = '_'? ['0' - '9']*
24: let regoper = ['s' 'l']
25:
26: rule scanner = parse
27:   | number    { Number (bigstr (lexeme lexbuf)) }
28:   | regoper _ { regoper lexbuf }
29:   | _         { Operator (lexeme lexbuf).[0] }
30:   | eof       { raise End_of_file }
31:
32: {
33:
34: end
35:
36: }
```

```
1: (* $Id: dc.ml,v 1.1 2011-04-26 13:39:18-07 - - $ *)
2:
3: (*
4: * This file is useless for compilation.  However, for interactive
5: * testing it make loading all three files easier.  Normally for
6: * interactive use, type
7: *
8: *   #use "dc.ml";;
9: *
10: * at the toplevel.  Alternately, to run it directly without
11: * interacting with the toplevel, just use:
12: *
13: *   ocaml dc.ml
14: *
15: * which will run the program without need for compilation.
16: *)
17:
18: #use "bigint.ml";;
19: #use "scanner.ml";;
20: #use "maindc.ml";;
21:
```

```
1: # $Id: Makefile,v 1.19 2018-04-19 14:58:55-07 - - $
2:
3: DEPSFILE = Makefile.deps
4: NOINCLUDE = ci clean spotless
5: NEEDINCL = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
6: SOURCE = bigint.mli bigint.ml maindc.ml scanner.mll
7: ALLSRC = ${SOURCE} dc.ml Makefile
8: OBJCMI = bigint.cmi scanner.cmi
9: MAINCMI = maindc.cmi
10: OBJCMO = ${OBJCMI:.cmi=.cmo} maindc.cmo
11: CAMLRUN = ocamlc
12: LISTING = Listing.ps
13:
14: all : ${CAMLRUN}
15:
16: ${CAMLRUN} : ${OBJCMI} ${OBJCMO}
17:     ocamlc ${OBJCMO} -o ${CAMLRUN}
18:
19: %.cmi : %.mli
20:     - checksource $<
21:     ocamlc -c $<
22:
23: %.cmo : %.ml
24:     - checksource $<
25:     ocamlc -c $<
26:
27: scanner.cmi scanner.cmo scanner.ml : scanner.mll
28:     - checksource $<
29:     ocamllex $<
30:     ocamlc -c scanner.ml
31:
32: clean :
33:     - rm ${OBJCMO} ${OBJCMI} ${MAINCMI} ${DEPSFILE} scanner.ml
34:
35: spotless : clean
36:     - rm ${CAMLRUN} ${LISTING} ${LISTING:.ps=.pdf}
37:
38: ci : ${RCSFILES}
39:     - checksource ${ALLSRC}
40:     cid + ${ALLSRC}
41:
42: deps : ${SOURCE}
43:     ocamldep ${SOURCE} >${DEPSFILE}
44:
45: ${DEPSFILE} :
46:     @ touch ${DEPSFILE}
47:     make --no-print-directory deps
48:
49: lis : ${ALLSRC}
50:     mkpspdf ${LISTING} ${ALLSRC} ${DEPSFILE}
51:
52: again :
53:     make --no-print-directory spotless ci deps all lis
54:
55: ifeq (${NEEDINCL}, )
56: include ${DEPSFILE}
57: endif
58:
```



```
1: bigint.cmi :  
2: bigint.cmo : bigint.cmi  
3: bigint.cmx : bigint.cmi  
4: maindc.cmo : bigint.cmi  
5: maindc.cmx : bigint.cmx
```