

```
1: (* $Id: absyn.mli,v 1.4 2020-01-28 16:24:35-08 - - $ *)
2:
3: (*
4:  * Abstract syntax definitions for SB.
5:  *)
6:
7: type linenr      = int
8: type ident       = string
9: type label       = string
10: type number      = float
11: type oper        = string
12:
13: type memref      = Arrayref of ident * expr
14:                  | Variable of ident
15:
16: and  expr        = Number of number
17:                  | Memref of memref
18:                  | Unary of oper * expr
19:                  | Binary of oper * expr * expr
20:
21: type printable   = Printexpr of expr
22:                  | String of string
23:
24: type stmt        = Dim of ident * expr
25:                  | Let of memref * expr
26:                  | Goto of label
27:                  | If of expr * label
28:                  | Print of printable list
29:                  | Input of memref list
30:
31: type propline    = linenr * label option * stmt option
32:
33: type program     = propline list
34:
```

```
1: (* $Id: etc.mli,v 1.1 2019-01-24 15:47:38-08 - - $ *)
2:
3: (*
4:  * Main program and system access.
5:  *)
6:
7: val warn : string list -> unit
8:
9: val die : string list -> unit
10:
11: val syntax_error : Lexing.position -> string list -> unit
12:
13: val usage_exit : string list -> unit
14:
15: val read_number : unit -> float
16:
```

```
1: (* $Id: etc.ml,v 1.3 2020-01-22 16:07:24-08 - - $ *)
2:
3: let execname = Filename.basename Sys.argv.(0)
4:
5: let exit_status_ref : int ref = ref 0
6:
7: let quit () =
8:   if !Sys.interactive
9:   then Printf.printf "quit (): exit %d\n%!" !exit_status_ref
10:    else exit !exit_status_ref
11:
12: let eprint_list message =
13:   (exit_status_ref := 1;
14:    flush_all ();
15:    List.iter prerr_string message;
16:    prerr_newline ();
17:    flush_all ())
18:
19: let warn message = eprint_list (execname :: ": " :: message)
20:
21: let die message = (warn message; quit ())
22:
23: let syntax_error position message =
24:   warn (position.Lexing.pos_fname :: ": "
25:        :: string_of_int position.Lexing.pos_lnum :: ": "
26:        :: message)
27:
28: let usage_exit message =
29:   (eprint_list ("Usage: " :: execname :: " " :: message); quit ())
30:
31: let buffer : string list ref = ref []
32:
33: let rec read_number () = match !buffer with
34: | head::tail -> (buffer := tail;
35:                  try float_of_string head
36:                  with Failure _ -> nan)
37: | [] -> let line = input_line stdin
38:         in (buffer := Str.split (Str.regexp "[ \\t]+") line;
39:            read_number ())
40:
```

```
1: (* Generated: Tue Jan 28 16:24:35 PST 2020 *)
2: type variable_table_t = (string, float) Hashtbl.t
3: type array_table_t = (string, float array) Hashtbl.t
4: type unary_fn_table_t = (string, float -> float) Hashtbl.t
5: type binary_fn_table_t = (string, float -> float -> float) Hashtbl.t
6: type label_table_t = (string, Absyn.program) Hashtbl.t
7: val variable_table : variable_table_t
8: val array_table : array_table_t
9: val unary_fn_table : unary_fn_table_t
10: val binary_fn_table : binary_fn_table_t
11: val label_table : label_table_t
12: val init_label_table : Absyn.program -> unit
13: val dump_label_table : unit -> unit
```

```
1: (* $Id: tables.ml,v 1.6 2020-01-27 18:45:28-08 - - $ *)
2:
3: type variable_table_t = (string, float) Hashtbl.t
4: type array_table_t = (string, float array) Hashtbl.t
5: type unary_fn_table_t = (string, float -> float) Hashtbl.t
6: type binary_fn_table_t = (string, float -> float -> float) Hashtbl.t
7: type label_table_t = (string, Absyn.program) Hashtbl.t
8:
9: let variable_table : variable_table_t = Hashtbl.create 16
10: let _ = List.iter (fun (label, value) ->
11:     Hashtbl.add variable_table label value)
12:     ["e" , exp 1.0;
13:      "eof", 0.0;
14:      "pi" , acos ~-.1.0;
15:      "nan", nan]
16:
17: let array_table : array_table_t = Hashtbl.create 16
18:
19: let unary_fn_table : unary_fn_table_t = Hashtbl.create 16
20: let _ = List.iter (fun (label, value) ->
21:     Hashtbl.add unary_fn_table label value)
22:     ["+" , (~+.);
23:      "-" , (~-.);
24:      "abs" , abs_float;
25:      "acos" , acos;
26:      "asin" , asin;
27:      "atan" , atan;
28:      "ceil" , ceil;
29:      "cos" , cos;
30:      "exp" , exp;
31:      "floor" , floor;
32:      "log" , log;
33:      "log10" , log10;
34:      "log2" , (fun x -> log x /. log 2.0);
35:      "round" , Float.round;
36:      "sin" , sin;
37:      "sqrt" , sqrt;
38:      "tan" , tan]
39:
40: let binary_fn_table : binary_fn_table_t = Hashtbl.create 16
41: let _ = List.iter (fun (label, value) ->
42:     Hashtbl.add binary_fn_table label value)
43:     ["+", (+.);
44:      "-", (-.);
45:      "*", (*.);
46:      "/", (/.);
47:      "%", mod_float;
48:      "^", (^)]
49:
```

```
50:
51: let label_table : label_table_t = Hashtbl.create 16
52:
53: let rec init_label_table program =
54:   let rec init program = match program with
55:     | [] -> ()
56:     | (_, Some label, _)::rest ->
57:       (Hashtbl.add label_table label program; init rest)
58:     | _::rest -> init rest
59:   in (Hashtbl.reset label_table; init program)
60:
61: let dump_label_table () =
62:   let dump key value = match value with
63:     | [] -> ()
64:     | (line, _, _)::_ ->
65:       Printf.fprintf stderr
66:         "label_table: \"%s\" -> line %d\n%!" key line
67:   in Hashtbl.iter dump label_table
68:
```

```
1: (* Generated: Tue Jan 28 16:24:36 PST 2020 *)
2: val quote : string -> string
3: val join : string -> string -> string -> string list -> string
4: val string_of_option : ('a -> string) -> 'a option -> string
5: val string_of_ctor : string -> string list -> string
6: val string_of_list : ('a -> string) -> 'a list -> string
7: val string_of_printable : Absyn.printable -> string
8: val string_of_memref : Absyn.memref -> string
9: val string_of_expr : Absyn.expr -> string
10: val string_of_stmt : Absyn.stmt -> string
11: val dump_procline : int * string option * Absyn.stmt option -> unit
12: val dump_program : Absyn.program -> unit
```

```
1: (* $Id: dumper.ml,v 1.17 2020-01-27 14:28:07-08 - - $ *)
2:
3: open Printf;;
4:
5: let quote string =
6:   let regex = Str.regexp "\""
7:   and subst _ = "\\\""
8:   in "\"" ^ Str.global_substitute regex subst string ^ "\"";;
9:
10: let join start sep stop list =
11:   let rec join' list' = match list' with
12:     | [] -> stop
13:     | [unit] -> unit ^ stop
14:     | head::tail -> head ^ sep ^ " " ^ join' tail
15:   in match list with
16:     | [] -> start ^ stop
17:     | _::_ -> start ^ join' list;;
18:
19: let string_of_option str_fn item = match item with
20:   | None -> "None"
21:   | Some thing -> "Some (" ^ str_fn thing ^ ")";;
22:
23: let string_of_ctor ctor args =
24:   join (ctor ^ " (") ", " " " args;;
25:
26: let string_of_list str_fn list =
27:   join "[" "; " "]" (List.map str_fn list);;
28:
29: let rec string_of_printable printable = match printable with
30:   | Absyn.Printexpr expr ->
31:     string_of_ctor "Printexpr" [string_of_expr expr]
32:   | Absyn.String string ->
33:     string_of_ctor "String" [quote string]
34:
35: and string_of_memref memref = match memref with
36:   | Absyn.Arrayref (ident, expr) ->
37:     string_of_ctor "Arrayref" [quote ident; string_of_expr expr]
38:   | Absyn.Variable ident -> string_of_ctor "Variable" [quote ident]
39:
40: and string_of_expr expr = match expr with
41:   | Absyn.Number number ->
42:     string_of_ctor "Number" [string_of_float number]
43:   | Absyn.Memref memref ->
44:     string_of_ctor "Memref" [string_of_memref memref]
45:   | Absyn.Unary (oper, expr) ->
46:     string_of_ctor "Unary" [quote oper; string_of_expr expr]
47:   | Absyn.Binary (oper, expr1, expr2) ->
48:     string_of_ctor "Binary"
49:       [quote oper; string_of_expr expr1; string_of_expr expr2];;
50:
```



```
51:
52: let string_of_stmt (stmt: Absyn.stmt) = match stmt with
53:   | Absyn.Dim (ident, expr) ->
54:     string_of_ctor "Dim"
55:       [quote ident ^ ", " ^ string_of_expr expr]
56:   | Absyn.Let (memref, expr) ->
57:     string_of_ctor "Let"
58:       [string_of_memref memref; string_of_expr expr]
59:   | Absyn.Goto label ->
60:     string_of_ctor "Goto" [quote label]
61:   | Absyn.If (expr, label) ->
62:     string_of_ctor "If" [string_of_expr expr; quote label]
63:   | Absyn.Print printable'list ->
64:     string_of_ctor "Print"
65:       [string_of_list string_of_printable printable'list]
66:   | Absyn.Input memref'list ->
67:     string_of_ctor "Input"
68:       [string_of_list string_of_memref memref'list];;
69:
70: let dump_progline (linenr, label'option, stmt'option) =
71:   Printf.fprintf stderr "program: %d %s: %s\n%!" linenr
72:     (string_of_option quote label'option)
73:     (string_of_option string_of_stmt stmt'option);;
74:
75: let dump_program (program : Absyn.program) =
76:   List.iter dump_progline program;;
77:
```

```
1: (* $Id: interp.mli,v 1.7 2020-01-24 12:57:06-08 - - $ *)
2:
3: (*
4:  * Interpreter for Silly Basic
5:  *)
6:
7: val want_dump : bool ref
8:
9: val interpret_program : Absyn.program -> unit
10:
```

```
1: (* $Id: interp.ml,v 1.9 2020-01-28 13:33:00-08 - - $ *)
2:
3: open Absyn
4:
5: exception Unimplemented of string
6: let no_expr reason = raise (Unimplemented reason)
7: let no_stmt reason continuation = raise (Unimplemented reason)
8:
9: let want_dump = ref false
10:
11: let rec eval_expr (expr : Absyn.expr) : float = match expr with
12: | Number number -> number
13: | Memref memref -> no_expr "eval_expr Memref"
14: | Unary (oper, expr) -> no_expr "eval_expr Unary"
15: | Binary (oper, expr1, expr2) -> no_expr "eval_expr Binary"
16:
17: let rec interpret (program : Absyn.program) = match program with
18: | [] -> ()
19: | firstline::continuation -> match firstline with
20: | _, _, None -> interpret continuation
21: | _, _, Some stmt -> (interp_stmt stmt continuation)
22:
23: and interp_stmt (stmt : Absyn.stmt) (continuation : Absyn.program) =
24:   match stmt with
25:   | Dim (ident, expr) -> no_stmt "Dim (ident, expr)" continuation
26:   | Let (memref, expr) -> no_stmt "Let (memref, expr)" continuation
27:   | Goto label -> no_stmt "Goto label" continuation
28:   | If (expr, label) -> no_stmt "If (expr, label)" continuation
29:   | Print print_list -> interp_print print_list continuation
30:   | Input memref_list -> interp_input memref_list continuation
31:
32: and interp_print (print_list : Absyn.printable list)
33:   (continuation : Absyn.program) =
34:   let print_item item =
35:     (print_string " ";
36:      match item with
37:      | String string ->
38:        let regex = Str.regexp "\\(.*\\)"
39:        in print_string (Str.replace_first regex "\\1" string)
40:      | Printexpr expr ->
41:        print_float (eval_expr expr))
42:   in (List.iter print_item print_list; print_newline ());
43:   interpret continuation
44:
```

```
45:
46: and interp_input (memref_list : Absyn.memref list)
47:   (continuation : Absyn.program) =
48:   let input_number memref =
49:     try let number = Etc.read_number ()
50:       in (print_float number; print_newline ())
51:     with End_of_file ->
52:       (print_string "End_of_file"; print_newline ())
53:   in List.iter input_number memref_list;
54:   interpret continuation
55:
56: let interpret_program program =
57:   (Tables.init_label_table program;
58:    if !want_dump then Tables.dump_label_table ();
59:    if !want_dump then Dumper.dump_program program;
60:    interpret program)
61:
```

```
1: (* $Id: main.ml,v 1.3 2020-01-24 12:57:06-08 - - $ *)
2:
3: (*
4:  * Main program reads a file and prints to stdout.
5:  *)
6:
7: let interpret_source filename =
8:   try (let sourcefile =
9:         if filename = "-"
10:        then stdin
11:        else open_in filename in
12:        let lexbuf = Lexing.from_channel sourcefile in
13:        let abstract_syntax = Parser.program Scanner.token lexbuf in
14:        Interp.interpret_program abstract_syntax)
15:   with Sys_error (string) -> Etc.die [string]
16:
17: let _ = if !Sys.interactive
18:         then ()
19:         else match Array.length Sys.argv with
20:              | 1 -> interpret_source "-"
21:              | 2 -> interpret_source Sys.argv.(1)
22:              | _ -> Etc.usage_exit ["[filename.sb]"]
23:
```

```

1: /* $Id: parser.mly,v 1.3 2020-01-28 13:31:11-08 - - $ */
2:
3: %{
4:
5: let linenr() = (symbol_start_pos()).Lexing.pos_lnum
6:
7: let syntax() = Etc.syntax_error (symbol_start_pos()) ["syntax error"]
8:
9: %}
10:
11: %token <string> RELOP EQUAL ADDOP MULOP POWOP
12: %token <string> IDENT NUMBER STRING
13: %token COLON COMMA LPAR RPAR LSUB RSUB EOL EOF
14: %token DIM LET GOTO IF PRINT INPUT
15:
16: %type <Absyn.program> program
17:
18: %start program
19:
20: %%
21:
22: program      : stmt_list EOF                {List.rev $1}
23:
24: stmt_list    : stmt_list stmt EOL           {$2::$1}
25:               | stmt_list error EOL         {syntax(); $1}
26:               |                             {[]}
27:
28: stmt         : label action                 {(linenr(), Some $1, Some $2)}
29:               | action                     {(linenr(), None, Some $1)}
30:               | label                      {(linenr(), Some $1, None)}
31:               |                           {(linenr(), None, None)}
32:
33: label        : IDENT COLON                 {$1}
34:
35: action       : DIM IDENT LSUB expr RSUB     {Absyn.Dim ($2, $4)}
36:               | LET memref EQUAL expr       {Absyn.Let ($2, $4)}
37:               | GOTO IDENT                  {Absyn.Goto $2}
38:               | IF relexpr GOTO IDENT       {Absyn.If ($2, $4)}
39:               | PRINT print_list            {Absyn.Print $2}
40:               | PRINT                      {Absyn.Print ([]) }
41:               | INPUT input_list            {Absyn.Input $2}
42:
43: print_list   : print COMMA print_list      {$1::$3}
44:               | print                      {[ $1]}
45:
46: print        : expr                        {Absyn.Printexpr $1}
47:               | STRING                     {Absyn.String $1}
48:
49: input_list   : memref COMMA input_list     {$1::$3}
50:               | memref                     {[ $1]}
51:

```

```
52:
53: memref      : IDENT                {Absyn.Variable $1}
54:             | IDENT LSUB expr RSUB {Absyn.Arrayref ($1, $3)}
55:
56: relexpr     : expr RELOP expr      {Absyn.Binary ($2, $1, $3)}
57:             | expr EQUAL expr      {Absyn.Binary ($2, $1, $3)}
58:
59: expr        : expr ADDOP term       {Absyn.Binary ($2, $1, $3)}
60:             | term                  {$1}
61:
62: term        : term MULOP factor     {Absyn.Binary ($2, $1, $3)}
63:             | factor                {$1}
64:
65: factor      : primary POWOP factor  {Absyn.Binary ($2, $1, $3)}
66:             | primary                {$1}
67:
68: primary     : LPAR expr RPAR        {$2}
69:             | ADDOP primary          {Absyn.Unary ($1, $2)}
70:             | NUMBER                 {Absyn.Number (float_of_string $1)}
}
71:             | memref                 {Absyn.Memref $1}
72:             | IDENT LPAR expr RPAR   {Absyn.Unary ($1, $3)}
73:
```

```
1: (* $Id: scanner.mll,v 1.2 2019-11-26 14:09:51-08 - - $ *)
2:
3: {
4:
5: let lexerror lexbuf =
6:   Etc.syntax_error (Lexing.lexeme_start_p lexbuf)
7:     ["invalid character \" ^ (Lexing.lexeme lexbuf) ^ \""]
8:
9: let newline lexbuf =
10:   let incr pos =
11:     {pos with Lexing.pos_lnum = pos.Lexing.pos_lnum + 1;
12:       Lexing.pos_bol = pos.Lexing.pos_cnum}
13:   in (lexbuf.Lexing.lex_start_p <- incr lexbuf.Lexing.lex_start_p;
14:       lexbuf.Lexing.lex_curr_p <- incr lexbuf.Lexing.lex_curr_p)
15:
16: let lexeme = Lexing.lexeme
17:
18: }
19:
20: let letter      = ['a'-'z' 'A'-'Z' '_' ]
21: let digit       = ['0'-'9' ]
22: let fraction    = (digit+ '.'? digit* | '.' digit+)
23: let exponent    = (['E' 'e'] ['+' '-']? digit+)
24:
25: let comment     = ('#' [^'\n']* )
26: let ident       = (letter (letter | digit)*)
27: let number      = (fraction exponent?)
28: let string      = ('"' [^'\n' '"' ]* '"')
29:
```



```
30:
31: rule token      = parse
32:   eof            { Parser.EOF }
33:   [' ' '\t']     { token lexbuf }
34:   comment        { token lexbuf }
35:   "\n"           { newline lexbuf; Parser.EOL }
36:   ":"            { Parser.COLON }
37:   ","            { Parser.COMMA }
38:   "("            { Parser.LPAR }
39:   ")"            { Parser.RPAR }
40:   "["            { Parser.LSUB }
41:   "]"            { Parser.RSUB }
42:   "="            { Parser.EQUAL (lexeme lexbuf) }
43:   "!="           { Parser.RELOP (lexeme lexbuf) }
44:   "<"            { Parser.RELOP (lexeme lexbuf) }
45:   "<="          { Parser.RELOP (lexeme lexbuf) }
46:   ">"            { Parser.RELOP (lexeme lexbuf) }
47:   ">="          { Parser.RELOP (lexeme lexbuf) }
48:   "+"            { Parser.ADDOP (lexeme lexbuf) }
49:   "-"            { Parser.ADDOP (lexeme lexbuf) }
50:   "*"            { Parser.MULOP (lexeme lexbuf) }
51:   "/"            { Parser.MULOP (lexeme lexbuf) }
52:   "%"            { Parser.MULOP (lexeme lexbuf) }
53:   "^"            { Parser.POWOP (lexeme lexbuf) }
54:   "dim"          { Parser.DIM }
55:   "goto"         { Parser.GOTO }
56:   "if"           { Parser.IF }
57:   "input"        { Parser.INPUT }
58:   "let"          { Parser.LET }
59:   "print"        { Parser.PRINT }
60:   number         { Parser.NUMBER (lexeme lexbuf) }
61:   string         { Parser.STRING (lexeme lexbuf) }
62:   ident          { Parser.IDENT (lexeme lexbuf) }
63:   _              { lexerror lexbuf; token lexbuf }
64:
```

```
1: # $Id: Makefile,v 1.16 2020-01-28 16:12:50-08 - - $
2:
3: #
4: # General useful macros
5: #
6:
7: MKFILE      = Makefile
8: MAKEFLAGS += --no-builtin-rules
9: DEPSFILE    = ${MKFILE}.deps
10: NOINCLUDE   = ci clean spotless
11: NEEDINCL    = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
12: GMAKE       = ${MAKE} --no-print-directory
13: OCAMLOPT    = ocamlpt -g
14:
15: #
16: # File macros
17: #
18:
19: EXECBIN      = sbinterp
20: OBJCMX       = etc.cmx parser.cmx scanner.cmx tables.cmx \
21:               dumper.cmx interp.cmx main.cmx
22: OBJCMI       = ${OBJCMX:.cmx=.cmi} absyn.cmi
23: OBJBIN       = ${OBJCMX:.cmx=.o}
24: MLSOURCE     = absyn.mli etc.mli etc.ml tables.mli tables.ml \
25:               dumper.mli dumper.ml interp.mli interp.ml main.ml
26: GENLEXYACC   = parser.mli parser.ml scanner.ml
27: GENSOURCE    = dumper.mli tables.mli ${GENLEXYACC}
28: GENFILES     = ${GENSOURCE} parser.output ${DEPSFILE}
29: OTHERFILES   = ${MKFILE} ${DEPSFILE} using .ocamlinit
30: ALLSOURCES   = ${MLSOURCE} parser.mly scanner.mli ${OTHERFILES}
31: LISTING      = Listing.ps
32:
33: #
34: # General targets
35: #
36:
37: all : ${EXECBIN}
38:
39: ${EXECBIN} : ${OBJCMX} ${OBJBIN}
40:             ${OCAMLOPT} str.cmx ${OBJCMX} -o ${EXECBIN}
41:
42: %.cmi : %.mli
43:         ${OCAMLOPT} -c $<
44:
45: %.o %.cmx : %.ml
46:         ${OCAMLOPT} -c $<
47:
48: %.ml : %.mli
49:         ocamllex $<
50:
51: %.mli %.ml : %.mly
52:         ocamlyacc -v $<
53:
```

```
54:
55: MAKEMLI      = (echo "(* Generated: $$ (date) *)"; ${OCAMLOPT} -i $<) >$@
56:
57: tables.mli : tables.ml absyn.cmi
58:     ${call MAKEMLI}
59:
60: dumper.mli : dumper.ml absyn.cmi
61:     ${call MAKEMLI}
62:
63: #
64: # Misc targets
65: #
66:
67: clean :
68:     - rm ${OBJCMI} ${OBJCMX} ${OBJBIN} ${GENSOURCE}
69:
70: spotless : clean
71:     - rm ${EXECBIN} ${GENFILES} ${LISTING} ${LISTING:.ps=.pdf}
72:
73: ci : ${ALLSOURCES}
74:     - checksource ${ALLSOURCES}
75:     cid + ${ALLSOURCES}
76:
77: deps : ${MLSOURCE} ${GENSOURCE}
78:     @ echo "# Generated: $$ (date)" >${DEPSFILE}
79:     ocamldep ${MLSOURCE} ${GENSOURCE} >>${DEPSFILE}
80:
81: ${DEPSFILE} : tables.mli
82:     @touch ${DEPSFILE}
83:     ${GMAKE} deps
84:
85: lis : ${ALLSOURCES}
86:     mkpspdf ${LISTING} ${ALLSOURCES}
87:
88: again :
89:     ${GMAKE} spotless
90:     ${GMAKE} deps
91:     ${GMAKE} ci
92:     ${GMAKE} all
93:     ${GMAKE} lis
94:
95: ifeq "${NEEDINCL}" ""
96: include ${DEPSFILE}
97: endif
98:
```

```
1: # Generated: Tue Jan 28 16:24:35 PST 2020
2: absyn.cmi :
3: dumper.cmo : \
4:     absyn.cmi \
5:     dumper.cmi
6: dumper.cmx : \
7:     absyn.cmi \
8:     dumper.cmi
9: dumper.cmi : \
10:     absyn.cmi
11: dumper.cmi : \
12:     absyn.cmi
13: etc.cmo : \
14:     etc.cmi
15: etc.cmx : \
16:     etc.cmi
17: etc.cmi :
18: interp.cmo : \
19:     tables.cmi \
20:     etc.cmi \
21:     dumper.cmi \
22:     absyn.cmi \
23:     interp.cmi
24: interp.cmx : \
25:     tables.cmx \
26:     etc.cmx \
27:     dumper.cmx \
28:     absyn.cmi \
29:     interp.cmi
30: interp.cmi : \
31:     absyn.cmi
32: main.cmo : \
33:     scanner.cmo \
34:     parser.cmi \
35:     interp.cmi \
36:     etc.cmi
37: main.cmx : \
38:     scanner.cmx \
39:     parser.cmx \
40:     interp.cmx \
41:     etc.cmx
42: parser.cmo : \
43:     etc.cmi \
44:     absyn.cmi \
45:     parser.cmi
46: parser.cmx : \
47:     etc.cmx \
48:     absyn.cmi \
49:     parser.cmi
50: parser.cmi : \
51:     absyn.cmi
52: scanner.cmo : \
53:     parser.cmi \
54:     etc.cmi
55: scanner.cmx : \
56:     parser.cmx \
57:     etc.cmx
58: tables.cmo : \
```

```
59:      absyn.cmi \  
60:      tables.cmi  
61: tables.cmx : \  
62:      absyn.cmi \  
63:      tables.cmi  
64: tables.cmi : \  
65:      absyn.cmi  
66: tables.cmi : \  
67:      absyn.cmi
```

```
1: let rcs = "( * $Id: using,v 1.3 2019-01-24 17:15:07-08 - - $ *)";;
2:
3: print_endline rcs;;
4:
5: #load "str.cma";;
6:
7: #mod_use "absyn.mli";;
8: #mod_use "etc.ml";;
9:
10: #mod_use "parser.ml";;
11: #mod_use "scanner.ml";;
12:
13: #mod_use "tables.ml";;
14: #mod_use "dumper.ml";;
15:
16: #mod_use "interp.ml";;
17: #mod_use "main.ml";;
18:
19: open Interp;;
20: open Main;;
21:
22: want_dump := true;;
23:
```

```
1: let rcs = "( * $Id: .ocamlinit,v 1.6 2019-01-24 18:40:26-08 - - $ *)";;
2:
3: print_endline rcs;;
4:
5: #load "str.cma";;
6:
7: #mod_use "absyn.mli";;
8: #mod_use "etc.ml";;
9:
10: #mod_use "parser.ml";;
11: #mod_use "scanner.ml";;
12:
13: #mod_use "tables.ml";;
14: #mod_use "dumper.ml";;
15:
16: #mod_use "interp.ml";;
17: #mod_use "main.ml";;
18:
19: open Interp;;
20: open Main;;
21:
22: want_dump := true;;
23:
```