```
# $Id: README,v 1.1 2009-02-03 17:24:36-08 - - $

NAME
   sbtran - translator from SB to SBIR

SYNOPSIS
   sbtran sbsource >sbirobject

DESCRIPTION
   The sbtran utility translate Silly Basic source code into
   Silly Basic Intermediate Representation, which looks like
   Scheme.  You do not need to understand scanning, parsing,
   or O'Caml to use this.  See the SBIR sources in another
   directory.

```

```
(* $Id: etc.mli,v 1.1 2009-02-03 17:24:36-08 - - $ *)

(*
* Main program and system access.
*)

val execname : string

val quit : unit -> unit

val eprint : string list -> unit

val lexeprint : Lexing.position -> string list -> unit

val usageprint : string list -> unit

```

```
(* $Id: absyn.ml,v 1.2 2019-11-26 14:07:46-08 - - $ *)

(*
* Abstract syntax definitions for SB->IR.
*)

type linenr     = int
and  variable   = string
and  label      = string
and  number     = float
and  oper       = string
and  array      = variable * expr
and  fncall     = variable * expr

and  print      = Printexpr of expr
                | String of string

and  memref     = Array of array
                | Variable of variable

and  expr       = Binop of oper * expr * expr
                | Unop of oper * expr
                | Memref of memref
                | Constant of number
                | Fncall of fncall

and  stmt       = Dim of array
                | Let of memref * expr
                | Goto of label
                | If of expr * label
                | Print of print list
                | Input of memref list

and  program    = (linenr * label option * stmt option) list

```

```
(* $Id: etc.ml,v 1.1 2009-02-03 17:24:36-08 - - $ *)

open Lexing
open Printf

let execname = Filename.basename Sys.argv.(0)

let exit_code_ref = ref 0

let quit () = exit !exit_code_ref

let eprintlist message =
    (exit_code_ref := 1;
     flush_all ();
     List.iter (fprintf stderr "%s") message;
     fprintf stderr "\n";
     flush_all ())

let eprint message = eprintlist (execname :: ": " :: message)

let lexeprint position message =
    eprint (position.pos_fname :: ": "
            :: string_of_int position.pos_lnum :: ": "
            :: message)

let usageprint message =
    eprintlist ("Usage: " :: execname :: " " :: message)

```

```
(* $Id: main.ml,v 1.2 2019-11-26 14:07:46-08 - - $ *)

open Absyn
open Etc
open Lexing
open Printf

(*
* Functions for printing out the absyn tree in Scheme format.
*)

let
rec pr’print file print = match print with
    | Printexpr (expr)           -> fprintf file " %a" pr’expr expr
    | String (string)            -> fprintf file " %s" string

and pr’memref file memref = match memref with
    | Array (array)              -> pr’array file array
    | Variable (variable)        -> fprintf file "%s" variable

and pr’expr file expr = match expr with
    | Binop (oper, expr1, expr2) -> fprintf file "(%s %a %a)"
                                    oper pr’expr expr1 pr’expr expr2
    | Unop (oper, expr)          -> fprintf file "(%s %a)"
                                    oper pr’expr expr
    | Memref (memref)            -> pr’memref file memref
    | Constant (number)          -> fprintf file "%.15g" number
    | Fncall (fncall)            -> pr’fncall file fncall

and pr’stmt file stmt = match stmt with
    | None                       -> ()
    | Some (Dim (array))         -> fprintf file "(dim %a)"
                                    pr’array array
    | Some (Let (memref, expr))  -> fprintf file "(let %a %a)"
                                    pr’memref memref pr’expr expr
    | Some (Goto (label))        -> fprintf file "(goto %s)" label
    | Some (If (expr, label))    -> fprintf file "(if %a %s)"
                                    pr’expr expr label
    | Some (Print (prints))      -> fprintf file "(print%a)"
                                    pr’prints prints
    | Some (Input (memrefs))     -> fprintf file "(input%a)"
                                    pr’inputs memrefs

and pr’array file (variable, expr) =
    fprintf file "(asub %s %a)" variable pr’expr expr

and pr’fncall file (ident, expr) =
    fprintf file "(%s %a)" ident pr’expr expr

and pr’line file (line, label, stmt) =
    let s’label = match label with
        | None         -> ""
        | Some (label) -> label
    in  fprintf file "(%5d %-8s %a)\n" line s’label pr’stmt stmt

and pr’input   file inputs  = fprintf file " %a" pr’memref inputs

and pr’prints  file prints  = List.iter (pr’print file) prints

and pr’inputs  file memrefs = List.iter (pr’input file) memrefs

and pr’lines   file lines   = List.iter (pr’line file) lines

and pr’program file program = fprintf file "(\n%a)\n" pr’lines program

(*
* Main program reads a file and prints to stdout.
*)

let translatefile filename =
    try (printf ";;File: %s\n" filename;
         let sourcefile =
             if filename = "-" then stdin else open_in filename in
         let lexbuf = Lexing.from_channel sourcefile in
         let absyn = Parser.program Lexer.token lexbuf in
         printf "%a" pr’program absyn)
    with Sys_error (string) -> eprint [string]

let _ = if !Sys.interactive
        then ()
        else match Array.length Sys.argv with
             | 1 -> translatefile "-"
             | 2 -> translatefile Sys.argv.(1)
             | _ -> usageprint ["[filename.sb]"]

```

```
/* $Id: parser.mly,v 1.2 2019-11-26 14:07:46-08 - - $ */

%{
(******** BEGIN PARSER SEMANTICS ********)

open Absyn
open Etc
open Lexing

let syntax () = lexeprint (symbol_start_pos ()) ["syntax error"]

let linenr () = (symbol_start_pos ()).pos_lnum

(******** END PARSER SEMANTICS ********)
%}

%token <string> RELOP EQUAL ADDOP MULOP POWOP
%token <string> IDENT NUMBER STRING
%token COLON COMMA LPAR RPAR LSUB RSUB EOL EOF
%token DIM LET GOTO IF PRINT INPUT

%type <Absyn.program> program

%start program

%%

program : stmts EOF              { List.rev $1 }

stmts   : stmts stmt EOL         { $2::$1 }
        | stmts error EOL        { syntax (); $1 }
        |                        { [] }

stmt    : label action           { (linenr (), Some $1, Some $2) }
        | action                 { (linenr (), None, Some $1) }
        | label                  { (linenr (), Some $1, None) }
        |                        { (linenr (), None, None) }

label   : ident COLON            { $1 }

action  : DIM array              { Dim ($2) }
        | LET memref EQUAL expr { Let ($2, $4) }
        | GOTO ident             { Goto ($2) }
        | IF relexpr GOTO ident { If ($2, $4) }
        | PRINT prints           { Print ($2) }
        | PRINT                  { Print ([]) }
        | INPUT inputs           { Input ($2) }

prints  : print COMMA prints     { $1::$3 }
        | print                  { [$1] }

print   : expr                   { Printexpr ($1) }
        | STRING                 { String ($1) }

inputs  : memref COMMA inputs    { $1::$3 }
        | memref                 { [$1] }

memref  : ident                  { Variable ($1) }
```

```
          | array                   { Array ($1) }

array     : ident LSUB expr RSUB  { ($1, $3) }

relexpr : expr RELOP expr       { Binop ($2, $1, $3) }
          | expr EQUAL expr       { Binop ($2, $1, $3) }

expr      : expr ADDOP term       { Binop ($2, $1, $3) }
          | term                  { $1 }

term      : term MULOP factor     { Binop ($2, $1, $3) }
          | factor                { $1 }

factor    : primary POWOP factor  { Binop ($2, $1, $3) }
          | primary               { $1 }

primary : LPAR expr RPAR        { $2 }
          | ADDOP primary         { Unop ($1, $2) }
          | NUMBER                { Constant (float_of_string ($1)) }
          | memref                { Memref ($1) }
          | ident LPAR expr RPAR  { Fncall ($1, $3) }

ident     : IDENT                 { $1 }
          | DIM                   { "dim" }
          | GOTO                  { "goto" }
          | IF                    { "if" }
          | INPUT                 { "input" }
          | LET                   { "let" }
          | PRINT                 { "print" }

```

```
(* $Id: lexer.mll,v 1.4 2019-11-26 14:07:46-08 - - $ *)

{
(******** BEGIN LEXER SEMANTICS ********)

open Absyn
open Etc
open Lexing
open Parser
open Printf

let lexerror lexbuf =
    lexeprint (lexeme_start_p lexbuf)
              ["invalid character `" ^ (lexeme lexbuf) ^ "'"]

let newline lexbuf =
    let incrline pos =
        {pos with pos_lnum = pos.pos_lnum + 1; pos_bol = pos.pos_cnum}
    in  (lexbuf.lex_start_p <- incrline lexbuf.lex_start_p;
         lexbuf.lex_curr_p <- incrline lexbuf.lex_curr_p)

let list lexbuf =
    let pos = lexeme_start_p lexbuf
    in  (if pos.pos_bol = pos.pos_cnum
         then printf ";;%4d: " pos.pos_lnum;
         printf "%s" (lexeme lexbuf))

(******** END LEXER SEMANTICS ********)
}

let letter          = ['a'-'z' 'A'-'Z' '_']
let digit           = ['0'-'9']
let fraction        = (digit+ '.'? digit* | '.' digit+)
let exponent        = (['E' 'e'] ['+' '-']? digit+)

let comment         = ('#' [^'\n']*)
let ident           = (letter (letter | digit)*)
let number          = (fraction exponent?)
let string          = '"' [^'\n' '"']* '"'

rule token          = parse
    | eof           { EOF }
    | [' ' '\t']    { list lexbuf; token lexbuf }
    | comment       { list lexbuf; token lexbuf }
    | "\n"          { list lexbuf; newline lexbuf; EOL }
    | ":"           { list lexbuf; COLON }
    | ","           { list lexbuf; COMMA }
    | "("           { list lexbuf; LPAR }
    | ")"           { list lexbuf; RPAR }
    | "["           { list lexbuf; LSUB }
    | "]"           { list lexbuf; RSUB }
    | "="           { list lexbuf; EQUAL (lexeme lexbuf) }
    | "!="          { list lexbuf; RELOP (lexeme lexbuf) }
    | "<"           { list lexbuf; RELOP (lexeme lexbuf) }
    | "<="          { list lexbuf; RELOP (lexeme lexbuf) }
    | ">"           { list lexbuf; RELOP (lexeme lexbuf) }
    | ">="          { list lexbuf; RELOP (lexeme lexbuf) }
    | "+"           { list lexbuf; ADDOP (lexeme lexbuf) }
```

```
        | "-"           { list lexbuf; ADDOP (lexeme lexbuf) }
        | "*"           { list lexbuf; MULOP (lexeme lexbuf) }
        | "/"           { list lexbuf; MULOP (lexeme lexbuf) }
        | "%"           { list lexbuf; MULOP (lexeme lexbuf) }
        | "^"           { list lexbuf; POWOP (lexeme lexbuf) }
        | "dim"         { list lexbuf; DIM }
        | "goto"        { list lexbuf; GOTO }
        | "if"          { list lexbuf; IF }
        | "input"       { list lexbuf; INPUT }
        | "let"         { list lexbuf; LET }
        | "print"       { list lexbuf; PRINT }
        | number        { list lexbuf; NUMBER (lexeme lexbuf) }
        | string        { list lexbuf; STRING (lexeme lexbuf) }
        | ident         { list lexbuf; IDENT (lexeme lexbuf) }
        | _             { list lexbuf; lexerror lexbuf; token lexbuf }

```

```
# $Id: Makefile,v 1.11 2019-11-26 14:08:20-08 - - $

#
# General useful macros
#

MKFILE     = Makefile
MAKEFLAGS += --no-builtin-rules
DEPSFILE   = ${MKFILE}.deps
NOINCLUDE  = ci clean spotless
NEEDINCL   = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}

#
# File list macros
#

EXECBIN    = sbtran
OBJCMX     = absyn.cmx etc.cmx parser.cmx lexer.cmx main.cmx
OBJCMI     = ${patsubst %.cmx, %.cmi, ${OBJCMX}}
OBJBIN     = ${patsubst %.cmx, %.o, ${OBJCMX}}
MLSOURCE   = etc.mli absyn.ml etc.ml main.ml
GENSOURCE  = parser.mli parser.ml lexer.ml
GENFILES   = ${GENSOURCE} parser.output
ALLSOURCES = README ${MLSOURCE} parser.mly lexer.mll ${MKFILE}
LISTING    = Listing.ps

#
# General targets
#

all : ${EXECBIN}

${EXECBIN} : ${OBJCMX}
	ocamlopt ${OBJCMX} -o ${EXECBIN}

%.cmi : %.mli
	ocamlc -c $<

%.cmx : %.ml
	ocamlopt -c $<

%.ml : %.mll
	ocamllex $<

%.mli %.ml : %.mly
	ocamlyacc -v $<

#
# Misc targets
#

clean :
	- rm ${OBJCMI} ${OBJCMX} ${OBJBIN}

spotless : clean
	- rm ${EXECBIN} ${GENFILES}

ci : ${ALLSOURCES} ${SBFILES}
```

```
        cid + ${ALLSOURCES} ${SBFILES}

deps : ${MLSOURCE} ${GENSOURCE}
        @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
        ocamldep ${MLSOURCE} ${GENSOURCE} >>${DEPSFILE}

${DEPSFILE} :
        @touch ${DEPSFILE}
        ${MAKE} deps

lis : ${ALLSOURCES}
        mkpspdf ${LISTING} ${ALLSOURCES}

again :
        ${MAKE} spotless
        ${MAKE} deps
        ${MAKE} ci
        ${MAKE} all
        ${MAKE} lis

ifeq "${NEEDINCL}" ""
include ${DEPSFILE}
endif

```