Appendix D
SML Calculator Compiler

This is the listing of the calc structure in the calc.sml file.

```
structure calc =
struct
open RegisterAllocation;
open calcAS;

     structure calcLrVals =
         calcLrValsFun(structure Token = LrParser.Token)

     structure calcLex =
         calcLexFun(structure Tokens = calcLrVals.Tokens)

     structure calcParser =
         Join(structure Lex= calcLex
              structure LrParser = LrParser
              structure ParserData = calcLrVals.ParserData)

     val input_line =
       fn f =>
          let val sOption = TextIO.inputLine f
          in
            if isSome(sOption) then
               Option.valOf(sOption)
            else
               ""
          end

     val calcparse =
         fn filename =>
           let val instrm = TextIO.openIn filename
               val lexer = calcParser.makeLexer
                   (fn i => input_line instrm)
               val _ = calcLex.UserDeclarations.pos := 1
               val error = fn (e,i:int,_) =>
                     TextIO.output(TextIO.stdOut," line " ^
                     (Int.toString i) ^ ", Error: " ^ e ^ "\n")
           in
                calcParser.parse(30,lexer,error,())
                    before TextIO.closeIn instrm
           end

     (* These functions are needed for
        if-then-else expressions and functions *)
     val label = ref 0;
```

```

fun nextLabel() =
    let val lab = !label
    in
      label := !label + 1;
      "L"^Int.toString(lab)
    end

val relOpOpposites = [("=","<>"),("<>","="),("<=",">"),
    (">=","<"),("<",">="),(">","<=")];

exception notLocated;

fun opposite(relOp) =
  let fun mappedVal x nil = raise notLocated
        | mappedVal (x:string) ((y,z)::rest) =
            if x = y
            then z else mappedVal x rest
  in
    mappedVal relOp relOpOpposites
  end

(* These functions are needed for function
   and constant bindings *)

fun forloop (0, f, x) = 0
  | forloop (y, f, x) = (f x; forloop(y-1, f, x));


exception unboundId;

datatype Type = function' of string
              | constant' of string;

fun boundTo(name,[]) =
    let val idname = (case name of
                          function'(s) => s
                        | constant'(s) => s)
    in
      TextIO.output(TextIO.stdOut, "Unbound identifier "^
          idname^" referenced or type error!\n");
      raise unboundId
    end

  | boundTo(name,(n,ol,depth)::t) =
      if name=n then ol else boundTo(name,t);

fun depthOf(name,[]) =
    let val idname = (case name of
                          function'(s) => s
                        | constant'(s) => s)
    in
      TextIO.output(TextIO.stdOut, "Unbound identifier "^
          idname^" referenced or type error!\n");
          raise unboundId
        end

      | depthOf(name,(n,ol,depth)::t) =
          if name=n then depth else depthOf(name,t);

    val frameSize = 88;

    (* This is the code generation for the compiler *)

    exception Unimplemented;

    fun codegen(add'(t1,t2),outFile,bindings,offset,depth) =
        let val _ = codegen(t1,outFile,bindings,offset,depth)
            val _ = codegen(t2,outFile,bindings,offset,depth)
            val reg2 = popReg()
            val reg1 = popReg()
        in
          TextIO.output(outFile,reg1 ^ ":="^reg1^"+"^reg2^"\n");
          delReg(reg2);
          pushReg(reg1)
        end

      | codegen(sub'(t1,t2),outFile,bindings,offset,depth) =
        let val _ = codegen(t1,outFile,bindings,offset,depth)
            val _ = codegen(t2,outFile,bindings,offset,depth)
            val reg2 = popReg()
            val reg1 = popReg()
        in
          TextIO.output(outFile,reg1 ^ ":="^reg1^"-"^reg2^"\n");
          delReg(reg2);
          pushReg(reg1)
        end

      | codegen(integer'(i),outFile,bindings,offset,depth) =
        let val r = getReg()
        in
          TextIO.output(outFile, r ^ ":=" ^
              Int.toString(i) ^ "\n");
          pushReg(r)
        end


  | codegen(_,outFile,bindings,offset,depth) =
        (TextIO.output(TextIO.stdOut,
            "Attempt to compile expression"^
            " not currently supported!\n");
         raise Unimplemented)


    fun compile filename  =
        let val (ast, _) = calcparse filename
            val outFile = TextIO.openOut("a.ewe")
        in
          TextIO.output(outFile,"SP:=100\n");
          TextIO.output(outFile,"PR0 := 0\n");
          TextIO.output(outFile,"PR1 := 0\n");
          TextIO.output(outFile,"PR2 := 0\n");
          TextIO.output(outFile,"PR3 := 0\n");
          TextIO.output(outFile,"PR4 := 0\n");
          TextIO.output(outFile,"PR5 := 0\n");
          TextIO.output(outFile,"PR6 := 0\n");
          TextIO.output(outFile,"PR7 := 0\n");
          TextIO.output(outFile,"PR8 := 0\n");
          TextIO.output(outFile,"PR9 := 0\n");
          TextIO.output(outFile,"cr := 13\n");
          TextIO.output(outFile,"nl := 10\n");
          TextIO.output(outFile,"nullchar:=0\n");
          let val s = codegen(ast,outFile,
              [(function'("output"),"output",0),
               (function'("input"),"input",0)],0,0)
              val reg1 = popReg()
          in
            TextIO.output(outFile,
                "writeInt("^reg1^")\nhalt\n\n");
            delReg(reg1);
            TextIO.output(outFile,
                "###### input function ######\n");
            TextIO.output(outFile,"input:  readInt(PR9)"^
                "\t\t# read an integer into"^
                " function result register\n");
            TextIO.output(outFile,"SP:=M[SP+1]"^
                "\t\t# restore the stack pointer\n");
            TextIO.output(outFile,"PC:=PR8"^
                "\t\t\t# return from whence you came\n");
            TextIO.output(outFile,
                "###### output function ######\n");
            TextIO.output(outFile,"output:  writeInt(PR9)"^
                "\t\t# write the integer in function"^
                " parameter register\n");
            TextIO.output(outFile,"writeStr(cr)\n");
            TextIO.output(outFile,"SP:=M[SP+1]"^
                "\t\t# restore the stack pointer\n");
            TextIO.output(outFile,"PC:=PR8"^
                "\t\t\t# return from whence you came\n");
            TextIO.output(outFile,"equ PR0 M[0]\n");
            TextIO.output(outFile,"equ PR1 M[1]\n");
            TextIO.output(outFile,"equ PR2 M[2]\n");
            TextIO.output(outFile,"equ PR3 M[3]\n");
            TextIO.output(outFile,"equ PR4 M[4]\n");
            TextIO.output(outFile,"equ PR5 M[5]\n");
            TextIO.output(outFile,"equ PR6 M[6]\n");
            TextIO.output(outFile,"equ PR7 M[7]\n");
            TextIO.output(outFile,"equ PR8 M[8]\n");
            TextIO.output(outFile,"equ PR9 M[9]\n");
            TextIO.output(outFile,"equ MEM M[12]\n");
            TextIO.output(outFile,"equ SP M[13]\n");
            TextIO.output(outFile,"equ cr M[14]\n");
```

```
             TextIO.output(outFile,"equ nl M[15]\n");
             TextIO.output(outFile,"equ nullchar M[16]\n");
             printRegs(!regList,outFile);
             TextIO.closeOut(outFile)
           end
         end
         handle _ => (TextIO.output(TextIO.stdOut,
             "An error occurred while compiling!\n\n"));


     fun run(a,b::c) = (compile b; OS.Process.success)
       | run(a,b) = (TextIO.print("usage: sml @SMLload=calc\n");
                     OS.Process.success)
end
```