

# Appendix D

## SML Calculator Compiler

---

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

```
1  structure calc =
2  struct
3  open RegisterAllocation;
4  open calcAS;
5
6      structure calcLrVals =
7          calcLrValsFun(structure Token = LrParser.Token)
8
9      structure calcLex =
10         calcLexFun(structure Tokens = calcLrVals.Tokens)
11
12     structure calcParser =
13         Join(structure Lex= calcLex
14             structure LrParser = LrParser
15             structure ParserData = calcLrVals.ParserData)
16
17     val input_line =
18         fn f =>
19             let val sOption = TextIO.inputLine f
20             in
21                 if isSome(sOption) then
22                     Option.valOf(sOption)
23                 else
24                     ""
25                 end
26
27     val calcparse =
28         fn filename =>
29             let val instrm = TextIO.openIn filename
30             val lexer = calcParser.makeLexer
31                 (fn i => input_line instrm)
32             val _ = calcLex.UserDeclarations.pos := 1
33             val error = fn (e,i:int,_) =>
34                 TextIO.output(TextIO.stdOut," line " ^
35                     (Int.toString i) ^ ", Error: " ^ e ^ "\n")
36             in
37                 calcParser.parse(30,lexer,error,())
38                 before TextIO.closeIn instrm
39             end
40
41     (* These functions are needed for
42        if-then-else expressions and functions *)
43     val label = ref 0;
```

```
44
45 fun nextLabel() =
46   let val lab = !label
47   in
48     label := !label + 1;
49     "L"^Int.toString(lab)
50   end
51
52 val relOpOpposites = [("<=", "<"), ("<=", "<"), ("<=", "<"),
53   ("<=", "<"), ("<=", "<"), ("<=", "<")];
54
55 exception notLocated;
56
57 fun opposite(relOp) =
58   let fun mappedVal x nil = raise notLocated
59       | mappedVal (x:string) ((y,z)::rest) =
60         if x = y
61         then z else mappedVal x rest
62   in
63     mappedVal relOp relOpOpposites
64   end
65
66 (* These functions are needed for function
67    and constant bindings *)
68
69 fun forloop (0, f, x) = 0
70   | forloop (y, f, x) = (f x; forloop(y-1, f, x));
71
72
73 exception unboundId;
74
75 datatype Type = function' of string
76               | constant' of string;
77
78 fun boundTo(name, []) =
79   let val idname = (case name of
80     function'(s) => s
81     | constant'(s) => s)
82   in
83     TextIO.output(TextIO.stdOut, "Unbound identifier "^
84       idname^" referenced or type error!\n");
85     raise unboundId
86   end
87
88   | boundTo(name, (n,ol,depth)::t) =
89     if name=n then ol else boundTo(name,t);
90
91 fun depthOf(name, []) =
92   let val idname = (case name of
93     function'(s) => s
94     | constant'(s) => s)
95   in
96     TextIO.output(TextIO.stdOut, "Unbound identifier "^
97       idname^" referenced or type error!\n");
```

---

```

98         raise unboundId
99     end
100
101     | depthOf(name, (n, ol, depth)::t) =
102         if name=n then depth else depthOf(name, t);
103
104     val frameSize = 88;
105
106     (* This is the code generation for the compiler *)
107
108     exception Unimplemented;
109
110     fun codegen(add' (t1,t2), outFile, bindings, offset, depth) =
111         let val _ = codegen(t1, outFile, bindings, offset, depth)
112             val _ = codegen(t2, outFile, bindings, offset, depth)
113             val reg2 = popReg()
114             val reg1 = popReg()
115         in
116             TextIO.output(outFile, reg1 ^ " := " ^ reg1 ^ "+" ^ reg2 ^ "\n");
117             delReg(reg2);
118             pushReg(reg1)
119         end
120
121     | codegen(sub' (t1,t2), outFile, bindings, offset, depth) =
122         let val _ = codegen(t1, outFile, bindings, offset, depth)
123             val _ = codegen(t2, outFile, bindings, offset, depth)
124             val reg2 = popReg()
125             val reg1 = popReg()
126         in
127             TextIO.output(outFile, reg1 ^ " := " ^ reg1 ^ "-" ^ reg2 ^ "\n");
128             delReg(reg2);
129             pushReg(reg1)
130         end
131
132     | codegen(integer' (i), outFile, bindings, offset, depth) =
133         let val r = getReg()
134         in
135             TextIO.output(outFile, r ^ " := " ^
136                 Int.toString(i) ^ "\n");
137             pushReg(r)
138         end
139
140
141     | codegen(_, outFile, bindings, offset, depth) =
142         (TextIO.output(TextIO.stdOut,
143             "Attempt to compile expression" ^
144             " not currently supported!\n");
145         raise Unimplemented)
146
147
148     fun compile filename =
149         let val (ast, _) = calcparse filename
150             val outFile = TextIO.openOut("a.ewe")
151         in

```

```

152     TextIO.output(outFile,"SP:=100\n");
153     TextIO.output(outFile,"PR0 := 0\n");
154     TextIO.output(outFile,"PR1 := 0\n");
155     TextIO.output(outFile,"PR2 := 0\n");
156     TextIO.output(outFile,"PR3 := 0\n");
157     TextIO.output(outFile,"PR4 := 0\n");
158     TextIO.output(outFile,"PR5 := 0\n");
159     TextIO.output(outFile,"PR6 := 0\n");
160     TextIO.output(outFile,"PR7 := 0\n");
161     TextIO.output(outFile,"PR8 := 0\n");
162     TextIO.output(outFile,"PR9 := 0\n");
163     TextIO.output(outFile,"cr := 13\n");
164     TextIO.output(outFile,"nl := 10\n");
165     TextIO.output(outFile,"nullchar:=0\n");
166     let val s = codegen(ast,outFile,
167         [(function'("output"),"output",0),
168          (function'("input"),"input",0)],0,0)
169     val reg1 = popReg()
170 in
171     TextIO.output(outFile,
172         "writeInt("^reg1^")\nhalt\n\n");
173     delReg(reg1);
174     TextIO.output(outFile,
175         "##### input function #####\n");
176     TextIO.output(outFile,"input:  readInt (PR9) "^
177         "\t\t# read an integer into"^
178         " function result register\n");
179     TextIO.output(outFile,"SP:=M[SP+1]"^
180         "\t\t# restore the stack pointer\n");
181     TextIO.output(outFile,"PC:=PR8"^
182         "\t\t\t# return from whence you came\n");
183     TextIO.output(outFile,
184         "##### output function #####\n");
185     TextIO.output(outFile,"output:  writeInt (PR9) "^
186         "\t\t# write the integer in function"^
187         " parameter register\n");
188     TextIO.output(outFile,"writeStr(cr)\n");
189     TextIO.output(outFile,"SP:=M[SP+1]"^
190         "\t\t# restore the stack pointer\n");
191     TextIO.output(outFile,"PC:=PR8"^
192         "\t\t\t# return from whence you came\n");
193     TextIO.output(outFile,"equ PR0 M[0]\n");
194     TextIO.output(outFile,"equ PR1 M[1]\n");
195     TextIO.output(outFile,"equ PR2 M[2]\n");
196     TextIO.output(outFile,"equ PR3 M[3]\n");
197     TextIO.output(outFile,"equ PR4 M[4]\n");
198     TextIO.output(outFile,"equ PR5 M[5]\n");
199     TextIO.output(outFile,"equ PR6 M[6]\n");
200     TextIO.output(outFile,"equ PR7 M[7]\n");
201     TextIO.output(outFile,"equ PR8 M[8]\n");
202     TextIO.output(outFile,"equ PR9 M[9]\n");
203     TextIO.output(outFile,"equ MEM M[12]\n");
204     TextIO.output(outFile,"equ SP M[13]\n");
205     TextIO.output(outFile,"equ cr M[14]\n");

```

---

```
206         TextIO.output(outFile,"equ nl M[15]\n");
207         TextIO.output(outFile,"equ nullchar M[16]\n");
208         printRegs(!regList,outFile);
209         TextIO.closeOut(outFile)
210     end
211 end
212 handle _ => (TextIO.output(TextIO.stdOut,
213     "An error occurred while compiling!\n\n"));
214
215
216 fun run(a,b::c) = (compile b; OS.Process.success)
217 | run(a,b) = (TextIO.print("usage: sml @SMLload=calc\n");
218     OS.Process.success)
219 end
```