101: Byg egen compiler

undertitel

Mathias
Mirza
Mads

Gruppemedlemmere:

Gruppe Opgave

KØBENHAVNS UNIVERSITET

Datalogisk Institut
Compilers 2015

For at løse opgaver i task 1, kigger vi kun på `Lexer.lex`, `Parser.grm`, `Interpreter.sml`, `Typerchecker.sml` og `codegen.sml`.

Hvis vi, tager det slavisk, som i den rækkefølge de skal løses i. Det vil sige, for at implentere `TRUE` og `FALSE`, så skal de først defineres under keywords, der indsættes

Lexer.lex

```
42        | "true"          => Parser.TRUE pos
43        | "false"         => Parser.FALSE pos
```

her efter skal vi definere dem som tokens og expresseions

Parser.grm

```
13 %token <(int*int)> TRUE FALSE
```

Parser.grm

```
68          | TRUE            { Constant (BoolVal true, $1) }
69          | FALSE           { Constant (BoolVal false, $1) }
```

Som vi gjorde med `TRUE` og `FALSE`, så skal der også defineres en rule token for `TIMES` og `DIVIDE`, gøres næsten det samme

Lexer.lex

```
133  Parser.DIVIDE (getPos lexbuf) )
134  and action_10 lexbuf = (
135  Parser.TIMES  (getPos lexbuf) )
```

Det samme gælder for selve token, hvor det er en `(int*int)`, i `Parser.grm`. Selve precedence level af `TIMES` og `DIVIDE` er den højste, dvs. at $4+2\cdot3$ betyder $4+(2\cdot3)$.

Dog i `TIMES, DIVIDE`, skal vi bruge 3 registre, altså

Parser.grm

```
76          | Exp TIMES Exp   { Times($1, $3, $2) }
77          | Exp DIVIDE Exp  { Divide($1, $3, $2)}
```

Da vi skal tage højde for det Fasto, skal vi tjekke det abstrakte syntax træ,

Interpreter.sml

```
160 | evalExp ( Times(e1, e2, pos), vtab, ftab ) =
161     let
162       val res1   = evalExp(e1, vtab, ftab)
```

```
163       val res2   = evalExp(e2, vtab, ftab)
164     in
165       case (res1, res2) of
166           (IntVal n1, IntVal n2) => IntVal (n1 * n2)
167         | _ => invalidOperands
168             "Times on non-integral args: "
169                 [(Int, Int)] res1 res2 pos
170     end
171
172 | evalExp ( Divide(e1, e2, pos), vtab, ftab ) =
173     let
174       val res1   = evalExp(e1, vtab, ftab)
175       val res2   = evalExp(e2, vtab, ftab)
176     in
177       if res2 = IntVal 0 then raise Fail "Division by zero"
178         else
179           case (res1, res2) of
180             (IntVal n1, IntVal n2) => IntVal( Int.quot(n1, n2))
181             | _ => invalidOperands
182                 "Divide on non-integral args: "
183                     [(Int, Int)] res1 res2 pos
184     end
```

For at kunne tage højde for at alle operationer i Fasto, bliver udført lovligt, så har vi med en type-checker, at det bliver overholdt

TypeChecker.sml

```
160 | In.Times (e1, e2, pos)
161   => let val (_, e1_dec, e2_dec) =
162             checkBinOp ftab vtab (pos, Int, e1, e2)
163       in (Int,
164           Out.Times (e1_dec, e2_dec, pos))
165       end
166
167 | In.Divide (e1, e2, pos)
168   => let val (_, e1_dec, e2_dec) =
169             checkBinOp ftab vtab (pos, Int, e1, e2)
170       in (Int,
171           Out.Divide (e1_dec, e2_dec, pos))
172       end
```

text her.

CodeGen.sml

```
133   | Times (e1, e2, pos) =>
134       let val t1 = newName "times_L"
135           val t2 = newName "times_R"
136           val code1 = compileExp e1 vtable t1
137           val code2 = compileExp e2 vtable t2
138       in  code1 @ code2 @ [Mips.MUL (place,t1,t2)]
139       end
140   | Divide (e1, e2, pos) =>
141       let val t1 = newName "divide_L"
142           val t2 = newName "divide_R"
143           val code1 = compileExp e1 vtable t1
144           val code2 = compileExp e2 vtable t2
145       in  code1 @ code2 @ [Mips.DIV (place,t1 ,t2)]
146       end
```

Selve `AND` og `OR` følger meget samme fremgangs metode, som `TRUE, FALSE` i lexeren og parseren, dog vil de to operationer i `Interpreter.sml`. Hvor vi i `OR`, kigger på, at hvis den første variabel er `TRUE`, returneres der `TRUE`

Interpreter.sml

```
190 | evalExp (Or (e1, e2, pos), vtab, ftab) =
191         let
192           val r1 = evalExp(e1, vtab, ftab)
193         in
194           case r1 of
195                BoolVal true => BoolVal true
196              | BoolVal false  => evalExp(e2, vtab, ftab)
197              | _ => raise Fail "First operand none boolean"
198         end
199
200   | evalExp ( Not(e, pos), vtab, ftab ) =
201         let
202           val r1 = evalExp(e, vtab, ftab)
203         in
204           case r1 of
205                BoolVal true => BoolVal false
206              | BoolVal false  => BoolVal true
207              | _ => raise Fail "First operand none boolean"
208         end
```

CodeGen.sml

```
408   | And (e1, e2, pos) =>
```

```
409     let val t1 = newName "and_L"
410         val t2 = newName "and_R"
411         val falseLabel = newName "false"
412         val trueLabel = newName "true"
413         val code1 = compileExp e1 vtable t1
414         val code2 = compileExp e2 vtable t2
415     in
416       code1 @
417       [ Mips.LI (place,"0")
418       , Mips.BEQ (t1, "0", falseLabel) ]
419       @ code2 @
420       [ Mips.BEQ (t2, "0", falseLabel)
421       , Mips.LI (place, "1")
422       , Mips.LABEL falseLabel ]
423     end
424
425   | Or (e1, e2, pos) =>
426     let val t1 = newName "and_L"
427         val t2 = newName "and_R"
428         val falseLabel = newName "false"
429         val trueLabel = newName "true"
430         val code1 = compileExp e1 vtable t1
431         val code2 = compileExp e2 vtable t2
432     in
433             code1 @
434             [ Mips.LI (place,"0")
435             , Mips.BEQ (t1, "1", trueLabel) ]
436             @ code2 @
437             [ Mips.BEQ (t2, "0", falseLabel)
438             , Mips.LABEL trueLabel
439             , Mips.LI (place, "1")
440             , Mips.LABEL falseLabel ]    end
```