

101: Byg egen compiler

undertitel

Mathias

Mirza

Mads

Gruppemedlemmere:

Gruppe Opgave



Datalogisk Institut
Compilers 2015

Indhold

Litteratur

4

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 implementere `TRUE` og `FALSE`, så skal de først defineres i `Lexer.lex` under keywords, der indsættes

Lexer.lex

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

i `Parser.grm` er begge blevet implementeret som tokens og ekspresion

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` inde i `Lexer.lex`, og der laves det samme som med `TRUE`. Dog, er den lidt anderledes

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 $4 + 2 \cdot 3$ betyder $4 + (2 \cdot 3)$

Dog i `TIMES`, `DIVIDE`, skal vi bruge 3 registre, som det ses i

Parser.grm

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

I `Interpreter.sml`, så skal man tjekke det abstrakte syntax træ, som skal udføres

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

```

tekst

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

```

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

```

Litteratur