Aflevering 1

Thomas Vinther, 201303874 Jens Kristian Nielsen, 201303862

4 februar 2019

Opgave 9

Kode

```
def eval(e: Exp): Int = e match {
  case IntLit(c) => trace("Integer "+c+" found"); c
  case BinOpExp(leftexp, op, rightexp) => trace("Binary operation found,
      evaluating left and right")
    val leftval = eval(leftexp)
    val rightval = eval(rightexp)
    op match {
      case PlusBinOp() => val res = leftval+rightval; trace("Addition yields "+
          res); res
      case MinusBinOp() => val res = leftval - rightval; trace("Subtraction
          yields "+res); res
      case MultBinOp() => val res = leftval*rightval; trace("Multiplication
          yields "+res); res
      case DivBinOp() =>
        if (rightval == 0)
          throw new InterpreterError(s"Division by zero", op)
          val res = leftval / rightval; trace("Dividing yields "+res); res
      case ModuloBinOp() => val res = leftval%rightval; trace("Modulating
          yields "+res); res
      case MaxBinOp() =>
        if (leftval>rightval) {
          trace("Calculating maximum and found: " + leftval)
          leftval
        } else { trace("Calculating maximum and found: "+rightval)
          rightval }
    }
  case UnOpExp(op, exp) =>
    trace("Unary expression found")
    val expval = eval(exp)
    op match {
      case NegUnOp() => trace("Negation of expression"); -expval
  }
}
def trace(msg: String): Unit =
  if (Options.trace)
    println(msg)
```

Beskrivelse

Trace-mekanismen fungerer ved at printe en relativt passende tekst om udførselen. Dette opnåes ved at indsætte printsætninger via trace funktionen der tjekker om Options.trace er true og i bekræftende fald printes der. Dette undgår kode duplikering fordi vi slipper for en masse if statements.

Når fortolkeren køres med argumenterne `-run -trace examples/calc1.s` fås:
`Binary operation found, evaluating left and right`

```
Binary operation found, evaluating left and right
Integer 1 found
Integer 2 found
Addition yields 3
Integer 3 found
Multiplication yields 9
Output: 9
```

Opgave 10

Kode

```
def unparse(n: AstNode): String = n match {
  case IntLit(c) => c.toString()
  case BinOpExp(leftexp, op, rightexp) =>
    val leftString = unparse(leftexp)
    val rightString = unparse(rightexp)
  op match {
    case PlusBinOp() => leftString+"+"+rightString
    case MinusBinOp() => leftString+parenthesize("","-",rightString,1)
    case MultBinOp() => parenthesize(leftString,"*",rightString,2)
    case DivBinOp() => parenthesize(leftString,"/",rightString,2)
    case ModuloBinOp() => parenthesize(leftString,"%",rightString,2)
    case MaxBinOp() => parenthesize(leftString,"max",rightString,2)
  }
  case UnOpExp(op,exp) =>
    val expString = unparse(exp)
    op match{
      case NegUnOp() => parenthesize("","-",expString,1)
  }
}
private def parenthesize(leftString: String, op: String, rightString: String,
    option: Int): String = {
  var parLeftString = ""
  var parRightString = rightString
  if(option == 2) {
    try {
      leftString.toInt
      parLeftString = leftString
    }
    catch {
      case e: Exception => parLeftString = "(" + leftString + ")"
    }
  }
  try {
    rightString.toInt
  }
  catch{
    case e: Exception => parRightString = "("+rightString+")"
  }
  parLeftString + op + parRightString
}
```

Beskrivelse

Unparse-mekanismen fungerer ved at den tager et expression som input og løber det igennem på tilsvarende vis som Parser.parse() har pakket det ind og pakker det ud igen, hvormed Unparse.unparse() bliver den inverse funktion til Parser.parse(). For at få pæne paranteser har vi lavet en hjælpe funktion parenthesize der tjekker om længden på venstre hhv højre streng er længere end 1