Aflevering 3

Studerende 1, 2017xxxxx Studerende 2, 2017xxxxx

X. YYYY 20ZZ

Opgave 36

```
  def unparse(e: AstNode): String =
    ???
```

Opgave 37

```
def eval(e: Exp, env: VarEnv): Val = e match {
    case IntLit(c) => IntVal(c)
    case BoolLit(c) => ???
    case FloatLit(c) => ???
    case StringLit(c) => ???
    case VarExp(x) => env.getOrElse(x, throw new InterpreterError(s"Unknown
        identifier '$x'", e))
    case BinOpExp(leftexp, op, rightexp) =>
      val leftval = eval(leftexp, env)
      val rightval = eval(rightexp, env)
        // ...
        case MinusBinOp() => ???
        case MultBinOp() => ???
        case DivBinOp() => ???
        case ModuloBinOp() => ???
        case EqualBinOp() => ???
        case LessThanBinOp() => ???
        case LessThanOrEqualBinOp() => ???
        case MaxBinOp() => ???
        case AndBinOp() => ???
        case OrBinOp() => ???
      }
    case UnOpExp(op, exp) =>
      val expval = eval(exp, env)
      op match {
        // ...
        case NotUnOp() => ???
      }
    case IfThenElseExp(condexp, thenexp, elseexp) => ???
    // ...
    case MatchExp(mexp, cases) =>
      val matchval = eval(mexp, env)
      matchval match {
        case TupleVal(vs) =>
          for (c <- cases) {
            if (vs.length == c.pattern.length) {
              ???
            }
          }
          throw new InterpreterError(s"No case matches value ${valueToString(
              matchval)}", e)
```

```
        case _ => throw new InterpreterError(s"Tuple expected at match, found ${
            valueToString(matchval)}", e)
      }
  }
```

Opgave 38

```
  def typeCheck(e: Exp, tenv: TypeEnv): Type = e match {
    case IntLit(_) => IntType
    case BoolLit(_) => ???
    case FloatLit(_) => ???
    case StringLit(_) => ???
    case VarExp(x) => ???
    case BinOpExp(leftexp, op, rightexp) =>
      val lefttype = typeCheck(leftexp, tenv)
      val righttype = typeCheck(rightexp, tenv)
      op match {
        // ...
        case MinusBinOp() => ???
        case MultBinOp() => ???
        case DivBinOp() => ???
        case ModuloBinOp() => ???
        case EqualBinOp() => ???
        case LessThanBinOp() => ???
        case LessThanOrEqualBinOp() => ???
        case MaxBinOp() => ???
        case AndBinOp() => ???
        case OrBinOp() => ???
      }
    case UnOpExp(op, exp) => ???
    case IfThenElseExp(condexp, thenexp, elseexp) => ???
    case BlockExp(vals, exp) => ???
    case TupleExp(exps) => TupleType(???)
    case MatchExp(mexp, cases) =>
      val mexptype = typeCheck(mexp, tenv)
      mexptype match {
        case TupleType(ts) =>
          for (c <- cases) {
            if (ts.length == c.pattern.length) {
              ???
            }
          }
          throw new TypeError(s"No case matches type ${Unparser.unparse(mexptype)}"
              , e)
        case _ => throw new TypeError(s"Tuple expected at match, found ${Unparser.
            unparse(mexptype)}", e)
      }
  }
```