Advanced Programming 2017
Assignment 1

Tom Jager
dgr418@alumni.ku.dk

Tobias Ludwig
fqj315@alumni.ku.dk

September 26, 2017

1 New grammar

```
Expr ::= Term1 ExprOpt
       | Term1

ExprOpt ::= ',' Expr

Term1 ::= Ident '=' Term1
        | Term2

Term2 ::= Term3 '===' Term3
        | Term3 '<' Term3
        | Term3

Term3 ::= Term4 '+' Term4
        | Term4 '-' Term4
        | Term4

Term4 ::= Atom '*' Atom
        | Atom '%' Atom
        | Atom

Atom ::= Number
       | String
       | 'true'
       | 'false'
       | 'undefined'
       | Ident
       | '[' Exprs ']'
       | '[' ArrayFor ']'
       | '(' Expr ')'
```

```
Exprs ::= $\epsilon$
        | Expr1 CommaExprs

CommaExprs ::= $\epsilon$
             | ',' Expr1 CommaExprs

ArrayFor ::= 'for' '(' Ident 'of' Expr1 ')' ArrayCompr

ArrayIf ::= 'if' '(' Expr1 ')' ArrayCompr

ArrayCompr ::= Expr1
             | ArrayFor
             | ArrayIf
```

We transformed the given grammar by hand in order to make the code easier.

1. Abolishing **Left-recursion**: this is the case for the definition of `Expr` which consists of top-level instructions (again `Exprs`!) seperated by comma. We can prevent the left-recursion by handling the case of a single vs. multiple inputs seperately and by introducing a helper `ExprOpt` which calls `Expr` again circular.

2. **Precedence** is possible by defining the operators explicitly on different levels which we called `Term1` through `Term4` sticking to the numbering in the task description where level 1 corresponds to the lowest precedence. Because the parser will work through the grammar top-down, we will parse those first. This way we get a hierarchy of operators each of which can only be called with terms from lower levels.

3. **Associativity** comes into the game for for the arithmetic operators is

Further aspects: Type checking: - Ident

Note if our hierarchy were that simple we could not use lower precedence level (e.g. Assignment, Term1) in computations of higher precedence, e.g. in `3 + (x=2)` could not be parsed. However, we have a remedy for that. We include the `'(' Expr ')'` as an option on the `Atom` level thereby closing the circle to the top of the hierarchy. Hence a proper nesting of expressions is possible maintaining a new frame of precedence in every paranthesized expression.

2 Parsers for Number, Ident and String

2.1 Number

Number is supposed to be a 9-digit signed integer. Parsec provides a function `count` which aids here.

3 The fuzzy gap between Parser and Interpreter

It is always a question of design how much interpreting is already done by the parser, with other words - where does Syntax end and Semantics start? For example we decided not to do any type-checking on the parser level.