Homework 3 (15 points)

Recursive-descendent parsing

Study the Mini-Pascal BNF grammar attached to this exercise sheet and do the following tasks:

1. Transform the grammar into a form suitable for a recursive-descent parsing as follows:
    a. Eliminate the left recursion; **(1 point)**
    b. Left factorize the productions (replace non-terminals with their right hand side, if necessary); **(1 point)**
    c. Convert the grammar into the EBNF form; **(1 point)**
2. Declare the set of tokens in an enumerated type and modify the lexical analyser from Homework 1 to return the correct token when matching a regular expression; **(1 point)**
3. Write a recursive-descent parser that uses the lexical analyser implemented in the previous task to get the next lookahead token; **(10 points)**
4. Modify your Mini-Pascal test program to contain at least five syntactic errors; **(0.5 points)**
5. Stop the parse at the first syntactic error with a meaningful error message including the line number. **(0.5 points)**

Mini-Pascal BNF Grammar

start → **PROGRAM IDENT ;** *varDec subProgList compStmt* **.**

varDec → **VAR** *varDecList*
| ε

varDecList → *varDecList identListType* **;**
| *identListType* **;**

identListType → *identList* **:** *type*

identList → *identList* **, IDENT**
| **IDENT**

type → *simpleType*
| **ARRAY [NUM .. NUM] OF** *simpleType*

simpleType → **INTEGER**
| **REAL**
| **BOOLEAN**

subProgList → *subProgList subProgHead varDec compStmt* **;**
| ε

subProgHead → **FUNCTION IDENT** *args* **:** *type* **;**
| **PROCEDURE IDENT** *args* **;**

args → **(** *parList* **)**
| ε

parList → *parList* **;** *identListType*
| *identListType*

compStmt → **BEGIN** *stmtList* **END**

stmtList → *stmtList* **;** *statement*
| *statement*

statement → *procCall*
| *assignStmt*
| *compStmt*
| *ifStmt*
| *whileStmt*

procCall → **IDENT**
| **IDENT** *params*

params → **(** *exprList* **)**

assignStmt → **IDENT :=** *expr*
| **IDENT** *index* **:=** *expr*

index → **[** *expr* **]**
| **[** *expr* **..** *expr* **]**

ifStmt → **IF** *expr* **THEN** *statement elsePart*

elsePart → **ELSE** *statement*
| ε

whileStmt → **WHILE** *expr* **DO** *statement*

exprList → *exprList* **,** *expr*
| *expr*

expr → *simpleExpr relOp simpleExpr*
| *simpleExpr*

simpleExpr → *simpleExpr addOp term*
| *term*

term → *term mulOp factor*
| *factor*

factor → **NUM**
| **FALSE**
| **TRUE**
| **IDENT**
| **IDENT** *index*
| **IDENT** *params*
| **NOT** *factor*
| **–** *factor*
| **(** *exp* **)**

relOp → **<** | **<=** | **>** | **>=** | **=** | **<>**

addOp → **+** | **–** | **OR**

mulOp → ***** | **/** | **DIV** | **MOD** | **AND**