

In this programming assignment, you will be building a parser for a small programming language and write a C++ program to test it. The syntax definitions and tokens of the programming language were given in Programming Assignment 2, and are given below using EBNF notations.

```
Prog := begin StmtList end
StmtList := Stmt; { Stmt;}
Stmt := PrintStmt | AssignStmt | IfStmt
PrintStmt := print ExprList
IfStmt := if (Expr) then Stmt
AssignStmt := Var = Expr
ExprList := Expr {, Expr}
Expr := Term {(+|-) Term}
Term := Factor {(*/) Factor}
Var := ident
Factor := ident | iconst | rconst | sconst | (Expr)
```

The following points describe the programming language, note that not all these points will be addressed in this assignment. However, they are listed in order to give you an understanding of the language semantics and what to be considered for implementing an interpreter for the language in Programming Assignment 4. These points are:

1. The language has three types: Integer, Real, and String.
2. The PLUS and MINUS operators in Expr represent addition and subtraction.
3. The MULT and DIV operators in Term represent multiplication and division.
4. The PLUS, MINUS, MULT and DIV operators are left associative.
5. MULT and DIV have higher precedence than PLUS and MINUS.
6. The type of a variable is the type of the value assigned to it.
7. An IfStmt evaluates the expression (Expr), which must be an integer. If the expression value is nonzero, then the Stmt is executed, otherwise it is not.
8. A PrintStmt evaluates the list of expressions (ExprList), and prints their values in order from left to right.
9. An IfSTMT statement whose Expr is not of an integer type is an error.

10. The EQ operator in the AssignStmt assigns a value to a variable. It evaluates the Expr on the right-hand side and saves its value in memory associated with the left-hand side (an IDENT). If the IDENT does not exist, it is created. If the IDENT already exists, its value is replaced.
11. The type of a variable (Integer, Real, and String) is determined from the first time it has been assigned a value in an assignment statement. Any reassignments for the same variable have to be for a value of the same type or a compatible one. For example, it is an error to assign a numeric value to a variable of a string type, or to assign a string value to a numeric variable (i.e., integer or real). However, an integer variable can be assigned a real value, and a real variable can be assigned an integer value. In either case, conversion of the value to the type of the variable must be applied.
12. The binary operations for addition, subtraction, multiplication, and division are performed upon two numeric operands (i.e., integer or real) of the same or different types. If the operands are of the same type, the type of the result is the same type as the operator's operands. Otherwise, the type of the result is real.
13. It is an error to use a variable in an expression before it has been assigned.

### **Parser Requirements:**

Implement a recursive-descent parser for the given programming language. You may use the lexical analyzer you wrote for Programming Assignment 2, OR you may use the provided implementation when it is posted. The parser should provide the following:

- The results of an unsuccessful parsing are a set of error messages printed by the parser functions, as well as the error messages that might be detected by the lexical analyzer.
- If the parser fails, the program should stop after the parser function returns.
- The assignment does not specify the exact error messages that should be printed out by the parser; however, the format of the messages should be the line number, followed by a colon and a space, followed by some descriptive text. Suggested messages might include "No statements in program", "Invalid statement", "Missing THEN", "Undefined Variable", "Missing END", etc.

You are given a header file for the parser, “parser.h”, and the “lex.h”. “parse.h” includes the following:

- Prototype definitions of the parser functions (e.g., Prog, StmtList, Stmt, etc.)
- A map container that keeps a record of the defined variables in the parsed program, defined as: `map<string, bool> defVar;`  
The key of the `defVar` is a variable name, and the value is a boolean that is set to true when the first time the variable has been assigned, otherwise it is false.
- A function definition for handling the display of error messages, called `ParserError`.
- Functions to handle the process of token lookahead.
- Static int variable for counting errors, called `error_count`.

### **Program Requirements:**

1. There can be at most one file name specified on the command line. If more than one filename is provided, the program should print “ONLY ONE FILE NAME ALLOWED” and it should stop running.
2. If the program cannot open a filename that is given, the program should print “CANNOT OPEN THE FILE {arg}”, where {arg} is the filename given, and it should stop running.
3. The result of an unsuccessful parsing is a set of error messages printed by the parser functions.

functions.

4. If the call to `Prog()` function fails, the program should stop and display a message as "Unsuccessful Parsing ", and display the number of errors detected. For example:

```
Unsuccessful Parsing
```

```
Number of Syntax Errors: 3
```

5. If the call to `Prog()` function succeeds, the program should stop and display a message such as "Successful Parsing ", and the program stops.
6. Testing files are provided which represent various cases of syntactic error types. They can be used for testing your implementation. You can also come up with your own testing cases.