CS 280
Fall 2021
Programming Assignment 2

November 3rd, 2021

Due Date: Tuesday, November 16, 2021, 23:59
Total Points: 20

In this programming assignment, you will be building a parser for a simple programming language. A preliminary grammar rules of the language and its tokens were given in Programming Assignment 1. However, the following specification is an update to the previously introduced grammar rules of the language and its tokens. Your implementation of a parser to the language is based on the following grammar rules specified in EBNF notations.

```
Prog ::= PROGRAM IDENT StmtList END PROGRAM
DeclStmt ::= (INT | FLOAT) IdentList
IdentList ::= IDENT {, IDENT}
StmtList ::= Stmt; {Stmt;}
Stmt ::= DeclStmt | ControlStmt
ControlStmt ::= AssigStmt | IfStmt | WriteStmt
WriteStmt ::= WRITE ExprList
IfStmt ::= IF (LogicExpr) ControlStmt
AssignStmt ::= Var = Expr
ExprList ::= Expr {, Expr}
Expr ::= Term {(+|-) Term}
Term ::= SFactor {( *| / | % ) SFactor}
SFactor ::= (+ | -) Factor | Factor
LogicExpr ::= Expr (== | >) Expr
Var ::= IDENT
Factor = IDENT | ICONST | RCONST | SCONST | (Expr)
```

The following points describe the programming language. Note that not all of 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 3. These points are:

1. The language has two types: INT, and FLOAT.
2. The PLUS and MINUS operators in Expr represent addition and subtraction operations.

3. The MULT, DIV, and REM operators in Term represent multiplication, division and remainder operations.
4. The PLUS, MINUS, MULT, DIV, and REM operators are left associative.
5. MUL , DIV, and REM have higher precedence than PLUS and MINUS.
6. Unary operators for the sign have higher precedence than MULT, DIV, and REM.
7. All variables have to be declared in declaration statements.
8. An IfStmt evaluates a logical expression (LogicExpr) as a condition. If the logical expression value is true, then the Stmt block is executed, otherwise it is not.
9. A WriteStmt evaluates the list of expressions (ExprList), and prints their values in order from left to right.
10. The ASSOP operator in the AssignStmt that assigns a value to a variable. It evaluates the Expr on the right-hand side and saves its value in a memory location associated with the left-hand side variable (Var). The left-hand side variable can be assigned any of the numeric types (i.e., INT, FLOAT) of the language. For example, 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.
11. The binary operations for addition, subtraction, multiplication, and division are performed upon two numeric operands (i.e., INT or FLOAT) 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.
12. The binary operation for REM must be performed upon two integer operands. It is illegal to have a real operand for the REM operator.
13. The EQUAL and GTHAN relational operators operate upon two compatible operands. The evaluation of a logical expression, based on EQUAL or GTHAN operation, produces either a true or false value that controls whether the statement(s) of the selection IfStmt is executed or not.
14. It is an error to use a variable in an expression before it has been assigned.
15. The unary sign operators (+ or -) are applied upon a unary numeric operands (i.e., INT or FLOAT).

Parser Requirements:

Implement a recursive-descent parser for the given simple programming language. You may use the lexical analyzer you wrote for Programming Assignment 1, 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 Right Parentheis", "Undefined Variable", "Missing END", etc.

Provided Files

You are given the header file for the parser, "parse.h" and an incomplete file for the "parse.cpp". You should use "parse.cpp" to complete the implementation of the parser. In addition, "lex.h", "lex.cpp", and "prog2.cpp" files are also provided. The descriptions the files are as follows:

"Parse.h"

"parse.h" includes the following:

- Prototype definitions of the parser functions (e.g., Prog, StmtList, Stmt, etc.)

"Parse.cpp"

- 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 declared, otherwise it is false.
- A map container that keeps a record for each declared variable in the parsed program and its corresponding type, defined as:
  `map<string, Token> SymTable;`
  - The key of the SymTable is a variable name, and the value is a Token that is set to the type token (e.g., INT or FLOAT) when the variable is declared in a declaration statement.
- A function definition for handling the display of error messages, called `ParserError.`
- Functions to handle the process of token lookahead, GetNextToken and PushBackToken, defined in a namespace domain called Parser.
- Static int variable for counting errors, called `error_count,` and a function to return its value, called `ErrCount().`
- Implementations of some functions of the recursive-descent parser.

"prog2.cpp"

- You are given the testing program "prog2.cpp" that reads a file name from the command line. The file is opened for syntax analysis, as a source code for your parser.
- A call to Prog() function is made. If the call 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
  ```
- If the call to Prog() function succeeds, the program should stop and display the message "Successful Parsing ", and the program stops.

Vocareum Automatic Grading

- You are provided by a set of 19 testing files associated with Programming Assignment 2. Vocareum automatic grading will be based on these testing files. You may use them to check and test your implementation. These are available in compressed archive "PA2 Test

Cases.zip" on Canvas assignment. The testing case of each file is defined in the Grading table below.

- Automatic grading of clean source code testing files (testprog18, and testprog19) will be based on checking against the output message:

  Successful Parsing

- In each of the other testing files, there is one syntactic error at a specific line. The automatic grading process will be based on the statement number at which this error has been found and associated with one or more error messages.
- You can use whatever error message you like. There is no check against the contents of the error messages.
- A check of the number of errors your parser has produced and the number of errors printed out by the program is also made.

Submission Guidelines

- Submit your "parse.cpp" implementation through Vocareum. The "lex.h", "parse.h", "lex.cpp" and "prog2.cpp" files will be propagated to your Work Directory.
- **Submissions after the due date are accepted with a fixed penalty of 25%. No submission is accepted after Sunday 11:59 pm, November 21, 2021.**

Grading Table

Item	Points
Compiles Successfully	1
testprog1: Empty File	1
testprog2: Variable Redefinition	1
testprog3: Missing Program Name.	1
testprog4: Missing a Comma in Declaration Statement	1
testprog5: Missing a Semicolon	1
testprog6: Missing a Comma in expression list	1
testprog7: Incorrect Operator	1
testprog8: Missing right Parenthesis	1
testprog9: Missing left parenthesis in If condition	1
testprog10: Incorrect logic operator	1
testprog11: Missing assignment operator	1
testprog12: Missing PROGRAM at End of Program	1
testprog13: Missing END at the end of Program	1
testprog14: Missing PROGRAM at the beginning of a program	1
testprog15: Missing a right parenthesis after expression	1
testprog16: Missing operand after operator	1
testprog17: Undeclared Variable	1
testprog18: Clean program testing sign operator	1
testprog19: Clean Program	1
Total	20