CS 280
Fall 2022
Programming Assignment 2

November 3rd, 2022

Due Date: Sunday, November 20, 2022, 23:59
Total Points: 20

In this programming assignment, you will be building a parser for a simple programming language. The syntax definitions of the small programming language are given below using EBNF notations. Your implementation of a parser to the language is based on the following grammar rules specified in EBNF notations.

```
1.  Prog ::= PROGRAM IDENT StmtList END PROGRAM
2.  StmtList ::= Stmt; { Stmt; }
3.  Stmt ::= DeclStmt | ControlStmt
4.  DeclStmt ::= ( INT | FLOAT | BOOL ) VarList
5.  VarList ::= Var { ,Var }
6.  ControlStmt ::= AssigStmt | IfStmt | PrintStmt
7.  PrintStmt ::= PRINT (ExprList)
8.  IfStmt ::= IF (Expr) THEN StmtList [ ELSE StmtList ] END IF
9.  AssignStmt ::= Var = Expr
10. Var ::= IDENT
11. ExprList ::= Expr { , Expr }
12. Expr ::= LogORExpr ::= LogANDExpr { || LogANDRxpr }
13. LogANDExpr ::= EqualExpr { && EqualExpr }
14. EqualExpr ::= RelExpr  [== RelExpr ]
15. RelExpr ::= AddExpr  [ ( < | > ) AddExpr ]
16. AddExpr :: MultExpr { ( + | - ) MultExpr }
17. MultExpr ::= UnaryExpr { ( * | / ) UnaryExpr }
18. UnaryExpr ::= ( - | + | ! ) PrimaryExpr | PrimaryExpr
19. PrimaryExpr ::= IDENT | ICONST | RCONST | SCONST | BCONST | ( 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:

Table of Operators Precedence Levels

Precedence	Operator	Description	Associativity
1	Unary +, -, and !	Unary plus, minus, and logical NOT	Right-to-Left (ignored)
2	*, /	Multiplication and Division	Left-to-Right
3	+, -	Addition and Subtraction	Left-to-Right
4	<, >	Relational operators < and >	(no cascading)
5	==	Equality operator	(no cascading)
6	&&	Logical AND	Left-to-Right
7	||	Logical OR	Left-to-Right

1. The language has three types: INT, FLOAT and BOOL.
2. The precedence rules of operators in the language are as shown in the table of operators precedence levels.
3. The PLUS, MINUS, MULT, DIV, AND, OR operators are left associative.
4. A variable has to be declared in a declaration statement.
5. An IfStmt evaluates a logical expression (Expr) as a condition. If the logical condition value is true, then the StmtList in the Then-part are executed, otherwise they are not. An else part for an IfSmt is optional. Therefore, If an Else-part is defined, the StmtList in the Else-part are executed when the logical condition value is false.
6. A PrintStmt evaluates the list of expressions (ExprList), and prints their values in order from left to right followed by a newline.
7. The ASSOP operator (=) in the AssignStmt 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). A left-hand side variable of a numeric type can be assigned a value of either one 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. A BOOL var in the left-hand side of an assignment statement must be assigned a Boolean value.
8. The binary operations for addition, subtraction, multiplication, and division are performed upon two numeric operands (i.e., INT, 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. The binary logic operations for the AND and OR are applied on two Boolean operands.
9. The LTHAN and GTHAN relational operators and the EQUAL operator operate upon two operands of compatible types. The evaluation of a relational expression, based on LTHAN or GTHAN operators, or an Equality expression, based on the Equal operator, produce either a true or false value.
10. The unary sign operators (+ or -) are applied upon unary numeric operands (i.e., INT, FLOAT). While the unary NOT operator is applied upon a Boolean operand (i.e., BOOL).
11. It is an error to use a variable in an expression before it has been assigned.

<u>Parser Requirements:</u>

Implement a recursive-descent parser for the given 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 "Missing semicolon at end of Statement.", "Incorrect Declaration Statement.", “Missing Right Parenthesis”, “Undefined Variable”, “Missing END”, etc.

<u>Provided Files</u>

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

“Parse.h”

“parse.h” includes the following:

- Prototype definitions of the parser functions (e.g., Prog, DeclBlock, ProgBody, etc.)

“Parse.cpp”

- A map container that keeps a record of the defined variables in the parsed program, defined as: `map<string, bool> defVar;`
  - o 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 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 test cases 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 as “PA2 Test Cases.zip” on Canvas assignment. The testing case of each file is defined in the Grading table below.
- The automatic grading of a clean source code file 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 the number of associated error messages with this syntactic error.
- 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 are 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 Wednesday 11:59 pm, November 23, 2022.**

Grading Table

Item	**Points**
Compiles Successfully	1
testprog1: Incorrect Type	1
testprog2: Variable Redefinition	1
testprog3: Missing Program Name.	1
testprog4: Missing a Comma in Declaration Statement	1
testprog5: Missing PROGRAM Keyword	1
testprog6: Missing END of Program Body	1
testprog7: Missing Semicolon	1
testprog8: Missing left Parenthesis in PRINT Statement	1
testprog9: Missing Right Parenthesis in an expression	1
testprog10: Illegal Equality Expression	1
testprog11: Missing Assignment Operator	1
testprog12: Missing Operand After Operator	1
testprog13: Missing END of IF Statement	1
testprog14: Undeclared Variable	1
testprog15: Missing Then in IF Statement	1
testprog16: Illegal Relational Expression	1
testprog17: Clean Program 1 (Declaration within If-Statement	1
testprog18: Clean Program 2 (If-Then-Else)	1
testprog19: Clean Program 3 (Logic Expression)	1
Total	**20**