CS 280
Spring 2022
Programming Assignment 2

March 24, 2022

Due Date: Sunday, April 10, 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, called Pascal-Like Simple Language (PLSL), are given below using EBNF notations. The PLSL syntax has some features similar to the well-known Pascal Language. Your implementation of a parser to the language is based on the following grammar rules specified in EBNF notations.

1. `Prog ::= PROGRAM IDENT; DeclBlock ProgBody`
2. `DeclBlock ::= VAR {DeclStmt;}`
3. `DeclStmt ::= Ident {, Ident} : (Integer | Real | String)`
4. `ProgBody ::= BEGIN {Stmt;} END`
5. `Stmt ::= AssigStmt | IfStmt | WriteLnStmt | ForStmt`
6. `WriteLnStmt ::= WRITELN (ExprList)`
7. `IfStmt ::= IF ( LogicExpr ) THEN Stmt [ELSE Stmt]`
8. `ForStmt ::= FOR Var := ICONST (TO | DOWNTO) ICONST DO Stmt`
9. `AssignStmt ::= Var := Expr`
10. `ExprList ::= Expr {, Expr}`
11. `Expr ::= Term {(+|-) Term}`
12. `Term ::= SFactor {( * | / ) SFactor}`
13. `SFactor ::= [(+ | -)] Factor`
14. `LogicExpr ::= Expr (= | > | <) Expr`
15. `Var ::= IDENT`
16. `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:

Precedence	Highest	() parentheses
		Unary +, unary -
		*, / (Left Associative)
		+ (add), - (subtract) (Left Associative)
	Lowest	=, <, > (relational operators)

1. The language has three types: INTEGER, REAL and STRING.
2. The precedence rules of the PSLP language are as shown in the table
3. The PLUS, MINUS, MULT, and DIV operators are left associative.
4. All variables have to be declared in declaration statements in the declaration block.
5. 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. An else part for an IfSmt is optional.
6. A WriteLnStmt 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., INTEGER, REAL) 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. However, A left-hand side STRING variable can only be assigned a STRING value.
8. The binary operations for addition, subtraction, multiplication, and division are performed upon two numeric operands (i.e., INTEGER, 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.
9. The EQUAL LTHAN and GTHAN relational operators operate upon two compatible operands. The evaluation of a logical expression, based on EQUAL, LTHAN or GTHAN operation, produces either a true or false value that controls whether the statement(s) of the selection IfStmt is executed or not.
10. It is an error to use a variable in an expression before it has been assigned.
11. The unary sign operators (+ or -) are applied upon unary numeric operands (i.e., INTEGER, REAL).

Parser Requirements:

Implement a recursive-descent parser for the given PLSP 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 Parenthesis", "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 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;`
  - 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 23 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 as "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 (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 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, March 13, 2022.**

Grading Table

Item	Points
Compiles Successfully	1
testprog1: Incorrect Type	1
testprog2: Variable Redefinition	1
testprog3: Missing Program Name.	0.5
testprog4: Missing a Comma in Declaration Statement	0.5
testprog5: Invalid Operator Symbol	0.5
testprog6: Missing END of Program Body	0.5
testprog7: Missing Semicolon	1
testprog8: Missing left Parenthesis in WriteLn Statement	1
testprog9: Missing Right Parenthesis	1
testprog10: Invalid Logic Operator	1
testprog11: Missing Assignment Operator	1
testprog12: Missing Comma in Expression List	1
testprog13: Missing BEGIN of Program Body	1
testprog14: Missing Operand After Operator	1
testprog15: Undeclared Variable	1
testprog16: Using Undefined Variable	1
testprog17: Syntax Error in For Statement	1
testprog18: Missing Termination Value in For Statement	1
testprog19: Clean Program	1
testprog20: Missing Initialization Value in For Statement	0.5
testprog21: Missing PROGRAM Keyword	0.5
testprog22: Unrecognizable Declaration Block	0.5
testprog23: Syntax Error in IF Statement	0.5
Total	**20**