CS 280
Fall 2022
Programming Assignment 3

November 22, 2022

Due Date: Sunday, December 11, 2022, 23:59
Total Points: 20

In this programming assignment, you will be building an interpreter for our programming language. The grammar rules of the language and its tokens were given in Programming Assignments 1 and 2. You are required to modify the parser you have implemented for the language to implement an interpreter for it. The specifications of the grammar rules are described 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. You have already implemented all of the syntactic rules of the language as part of the parser implementation. The points related to the dynamic semantics (i.e. run-time checks) of the language must be implemented in your interpreter. These include points 5-12 in the following list. 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. A used variable in an expression must have been defined previously by an assignment statement.
4. It is illegal to use the name of the program as a variable anywhere in the program.
5. The scope of a declared variable extends from the point of its declaration statement to the End of the Program.
6. The PLUS, MINUS, MULT, DIV, AND, OR operators are left associative.
7. A variable has to be declared in a declaration statement.
8. 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.
9. A PrintStmt evaluates the list of expressions (ExprList), and prints their values in order from left to right followed by a newline.
10. 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 only.
11. 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 FLOAT.
12. The binary logic operations for the AND and OR operators are applied on two Boolean operands only.
13. 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.

14. The unary sign operators (+ or -) are applied upon one numeric operand (i.e., INT, FLOAT). While the unary NOT operator is applied upon a one Boolean operand (i.e., BOOL).
15. It is an error to use a variable in an expression before it has been assigned.

Interpreter Requirements:

I. Implement an interpreter for the language based on the recursive-descent parser developed in Programming Assignment 2. You need to modify the parser functions to include the required actions of the interpreter for evaluating expressions, determining the type of expression values, executing the statements, and checking run-time errors. You may use the lexical analyzer you wrote for Programming Assignment 1 and the parser you wrote for Programming Assignment 2. Otherwise you may use the provided implementations for the lexical analyzer and parser when they are posted. Rename the parse.cpp file as parserInt.cpp to reflect the applied changes on the current parser implementation for building an interpreter. The interpreter should provide the following:

- It performs syntax analysis of the input source code statement by statement, then executes the statement if there is no syntactic or semantic error.
- It builds information of variables types in the symbol table for all the defined variables.
- It evaluates expressions and determines their values and types. **You need to implement the overloaded operator functions for the Value class.**
- The results of an unsuccessful parsing and interpreting are a set of error messages printed by the parser/interpreter functions, as well as the error messages that might be detected by the lexical analyzer.
- **Any failures due to the process of parsing or interpreting the input program should cause the process of interpretation to stop and return back.**
- In addition to the error messages generated due to parsing, **the interpreter generates error messages due to its semantics checking**. The assignment does not specify the exact error messages that should be printed out by the interpreter. However, the format of the messages should be the line number, followed by a colon and a space, followed by some descriptive text, similar to the format used in Programming Assignment 2. Suggested messages of the interpreter's semantics check might include messages such as "Run-Time Error-Illegal Mixed Type Operands", " Run-Time Error-Illegal Assignment Operation", "Run-Time Error-Illegal Division by Zero", etc.

Provided Files

You are given the following files for the process of building an interpretation. These are "lex.h', "lex.cpp", "val.h", "parseInt.h", and "parseInt.cpp" with definitions and partial implementations of some functions. You need to complete the implementation of the interpreter in the provided copy of "parseInt.cpp. "parse.cpp" will be posted later on.

"val.h" includes the following:

- A class definition, called Value, representing a value object in the interpreted source code for constants, variables or evaluated expressions. It includes:

- Four data members of the Value class for holding a value as either an integer, float, string, or boolean.
- A data member holding the type of the value as an enum type defined as:
  ```
  enum ValType { VINT, VREAL, VSTRING, VBOOL, VERR };
  ```
- Getter methods to return the value of an object based on its type
- Getter methods to return the type of the value of an object.
- Setter methods to update the value of an object.
- Overloaded constructors to initialize an object based on the type of their parameters.
- Member methods to check the type of the Value object.
- Overloaded operators' functions for the arithmetic operators (+, -, *, and /)
- Overloaded operators' functions for the relational operators (==, >, and <)
- Overloaded operators' functions for the logical operators (&&, ||, and !)
- A friend function for overloading the operator<< to display value of an object based on its type.

- You are required to provide the implementation of the Value class in a separate file, called "val.cpp", which includes the implementations of the overloaded operator functions specified in the Value class definition (operator+, operator-, operator*, operator/, operator==, operator>, operator<, operator&&, operator||, and operator!).

"parserInt.h" includes the prototype definitions of the parser functions as in "parse.h" header file with the following applied modifications:

- `extern bool Var(istream& in, int& line, LexItem & tok);`
- `extern bool LogANDExpr(istream& in, int& line, Value & retVal);`
- `extern bool Expr(istream& in, int& line, Value & retVal);//or LogORExpr`
- `extern bool EqualExpr(istream& in, int& line, Value & retVal);`
- `extern bool RelExpr(istream& in, int& line, Value & retVal);`
- `extern bool AddExpr(istream& in, int& line, Value & retVal);`
- `extern bool MultExpr(istream& in, int& line, Value & retVal);`
- `extern bool UnaryExpr(istream& in, int& line, Value & retVal);`
- `extern bool PrimaryExpr(istream& in, int& line, int sign, Value & retVal);`

"parseInt.cpp" includes the following:

- Map containers definitions given in "parse.cpp" for Programming Assignment 2.
- The declaration of a map container for temporaries' values, called `TempsResults`. Each entry of `TempsResults` is a pair of a string and a Value object, representing a variable name, and its corresponding Value object.
- A map container SymTable that keeps a record of each declared variable in the parsed program and its corresponding type.
- The declaration of a pointer variable to a queue container of Value objects.
- Implementations of the interpreter actions in some functions.

“parse.cpp”

- Implementations of parser functions in “parse.cpp” from Programming Assignment 2.
    - It will be provided after the deadline of PA 2 submission (including any extensions).

“prog3.cpp”:

- You are given the testing program “prog3.cpp” that reads a file name from the command line. The file is opened for syntax analysis and interpretation, as a source code of the language.
- A call to Prog() function is made. If the call fails, the program should stop and display a message as "Unsuccessful Interpretation ", and display the number of errors detected. For example:
  ```
  Unsuccessful Interpretation
  Number of Syntax Errors: 3
  ```
- If the call to Prog() function succeeds, the program should stop and display the message `"Successful Execution"`, and the program stops.

Vocareum Automatic Grading

- You are provided by a set of 19 testing files associated with Programming Assignment 3. 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 “PA3 Test Cases.zip” on Canvas assignment. The testing case of each file is defined in the Grading table below.
- Automatic grading of testing files with no errors will be based on checking against the generated outputs by the executed source program and the output message:
  `Successful Execution`
- In each of the other testing files, there is one semantic error at a specific line. The automatic grading process will be based on identifying 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.

<u>Submission Guidelines</u>

- Submit your “parseInt.cpp” implementation through Vocareum. The “lex.h”, “parseInt.h”, “lex.cpp”, “val.h”, and “prog3.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, December 14, 2022.**

Grading Table

Item	Points
Compiles Successfully	1
testprog1: Undefined variable	1
testprog2: Illegal operand type for the NOT operator	1
testprog3: Compatible numeric mixed types	1
testprog4: If_Stmt test with relational expression	1
testprog5: Testing All Logic operators	1
testprog6: Illegal mixed type operands for a logic operator	1
testprog7: Illegal mixed type operands for a relational operator	1
testprog8: Testing Divide by zero operation	1
testprog9: Illegal operand type for sign operator	1
testprog10: Illegal assignment operation	1
testprog11: Clean Program testing If-stmt Then-part	1
testprog12: Clean Program testing If-stmt Else-part	1
testprog13: Clean Program testing expression evaluation with nested parentheses	1
testprog14: Illegal use of program name as a variable name	1
testprog15: Illegal mixed type operands for equality operator	1
testprog16: Testing mixed type operands for equality operator	1
testprog17: Illegal operands type for Add operator	1
testprog18: Illegal mixed type operands for multiplication operator	1
testprog19: Illegal mixed type operands for subtraction operator	1
Total	20