CS 280
Spring 2021
Programming Assignment 3

April 19, 2021

Due Date: Sunday, May 2nd, 2021, 23:59
Total Points: 20

In this programming assignment, you will be building an interpreter for our Fortran-Like programming language. You are required to modify the parser you have implemented for the programming language to implement an interpreter for it. The syntax definitions and tokens of the programming language were given in Programming Assignments 1 and 2, and are given below using EBNF notations with some modification. **The ReadStmt and VarList grammar rules have been deleted from the language.**

```
Prog = PROGRAM IDENT {Decl} {Stmt} END PROGRAM IDENT
Decl = (INTEGER | REAL | CHAR) : IdList
IdList = IDENT {,IDENT}
Stmt = AssigStmt | IfStmt | PrintStmt
PrintStmt = PRINT , ExprList
IfStmt = IF (LogicExpr) THEN {Stmt} END IF
AssignStmt = Var = Expr
ExprList = Expr {, Expr}
Expr = Term {(+|-) Term}
Term = SFactor {(*|/) SFactor}
SFactor = Sign Factor | Factor
LogicExpr = Expr (== | <) Expr
Var = IDENT
Sign = + | -
Factor = IDENT | ICONST | RCONST | SCONST | (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. The points related to the dynamic semantics (i.e. runt-time checks) of the language must be implemented in your interpreter. These include points 7-15 in the following list.

1. The language has three types: INTEGAR, REAL, and CHARacter (as 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. Unary operators for the sign have higher precedence than MULT and DIV.
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. Though the language does not include a Boolean type, Boolean expressions are considered as the type of evaluating LogicExpr.
9. A PrintStmt evaluates the list of expressions (ExprList), and prints their values in order from left to right.
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 memory associated with the left-hand side variable (Var). The left-hand side variable must be of a type that is compatible with the type of the expression value on the right-hand side. 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, it is an error to assign a numeric value to a variable of a CHAR type, or to assign a CHAR value to a numeric variable (i.e., integer or real).
11. 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. It is an error to have string operand to any of the arithmetic operations.
12. The EQUAL and LTHAN relational operators operate upon two compatible operands. The evaluation of a logical expression, based on EQUAL or LTHAN operation, produces either a true or false value that controls whether the statement(s) of the selection IfStmt is executed or not.
13. It is an error to use a variable in an expression before it has been assigned.
14. The unary sign operators (+ or -) are applied upon a unary numeric operand (i.e., INTEGER or REAL).
15. The name of the program at its header must match the name of the program at its end.

Interpreter Requirements:

I. Implement an interpreter for the simple Fortran-Like programming 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.
- 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.

II. You are given header files for the process of interpretation. These are "val.h", and "parserInt.h", in addition to "lex.h", "lex.cpp" and "prog2.cpp" codes are also provided.

"val.h": includes the following:

- A class, called Value, representing a value object in the interpreted source code for constants, variables or evaluated expressions. It includes:
  - Three data members of the Value class for holding a value as either an INTEGER, a REAL, or a CHAR.
  - A data member holding the type of the value as an enum type defined as:
    `enum ValType { VINT, VREAL, VCHAR, 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 parameter.
  - 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 <)
  - A friend function for overloading the operator<< to display value of an object based on its type.
- The implementations of the overloaded operators' functions: `operator+()`, `operator-()`, `operator*()`, and `operator\()`.
- The implementations of the overloaded operators' functions `operator<()` and `operator==()`.
- The declaration of a map container for the construction of a repository of temporaries, called `TempsResults`. Each entry of `TempsResults` is a pair of a string and a Value object, representing a variable name, a constant, or an expression value and its corresponding Value object.
- The declaration of a pointer variable to a queue container of Value objects.

“parserInt.h”: includes the prototype definitions of the parser function as in “parse.h” header file with following applied modifications:

- `extern bool Var(istream& in, int& line, LexItem & tok);`
- `extern bool Expr(istream& in, int& line, Value & retVal);`
- `extern bool LogicExpr(istream& in, int& line,Value & retVal);`
- `extern bool Term(istream& in, int& line, Value & retVal);`
- `extern bool SFactor(istream& in, int& line, Value & retVal);`
- `extern bool Factor(istream& in, int& line, Value & retVal);`

NOTE:

All code segments available in “parserInt.h” have to be moved to your “parserInt.cpp” file, except the defined functions prototypes. This process is needed in order to satisfy the requirements for linking program files on Linux when you are submitting your work to Vocareum.

“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.

III. Vocareum Automatic Grading

- You are provided by a set of 14 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 as “PA3 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 files will be based on checking against the whatever output to be displayed by the interpreter and the output message:
  ```
  Successful Execution
  ```
- In each of the other testing files, there is one semantics checking related 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/interpreter has produced and the number of errors printed out by the program is also made.

Submission Guidelines

- Submit your “parserInt.cpp” implementation through Vocareum. The “lex.h”, “parserInt.h”, “lex.cpp”, “val.h”, “val.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 Monday 11:59 pm, May 5, 2021.**

Grading Table

Item	**Points**
Compiles Successfully	1
Cleanprog: Clean program with assignstmt and ifstmt	2
Expreval: Expression evaluation	2
mixedAssign2: Mixed type assignment	2
Printexpr: Clean program testing Print statement and expression evaluation	2
Ifstmt: Ifstmt test	2
mixedAssign1: Mixed type assignment	1
Strexpr: Assignment of CHAR variable	1
Strassign: Testing assignment and printing of string variables	1
Progname: Matching program name	1
undefinedVar: Use of Undefined variable case	1
Invlogexpr: Invalid logic expression mixed operands	1
illexpr1: Illegal mixed type operands	1
Divbyzero: Divide by zero execution error	1
Illstrexpr: Illegal arithmetic add operation	1
Total	20