CSC 3310 Lab 2 CalcInterp
Due: Wednesday, November 30, 2016 11:59pm

Design and implement a Visual Studio C++ "console" application CalcInterp that implements a statement-by-statement interpreter for the Calculator language example with grammar as shown in Lab 1. Your program must obtain a source-filename from the commandline that contains a calc-language program. Then CalcInterp reads through (by repeatedly calling the getToken( ) function developed for the previous CalcLex scanner lab) and parses (recursive descent) the statements in the program, executing each statement as it is encountered. Calc “READ” statements will input data values from the standard console input, and “WRITE” statements will output data values to the standard console output (screen).

Your CalcInterp program will require the design and implementation of a simple ID symbol table. Each entry in the symtab must store an identifier name and an associated ‘double’ value for that name. The first time an ID name is used in a calc-program, the ID name must be inserted into the symtab (use a default value of 0.0 if necessary). Each time a READ ID or assignment statement ID := expr statement is executed, the input or computed-expression value must be stored in the symtab for the ID (replacing the older value as necessary). Each time that a reference is made to an ID within an expression (e.g., in Factor), the associated value for that ID from the symtab must be located and used within the expression computation.

Assume that all expression values and computations are carried out as doubles.

Also assume that all keywords and IDs are not case-sensitive (e.g., read & READ & Read are all treated identically). This is most easily accomplished by either having the scanner or parser or symtab automatically convert all letters in IDs to upper (or lower) case, or consistently performing case-insensitive comparisons.

Design and implement a recursive descent parser that calls the scanner as needed to obtain tokens one at a time from the input file, and parses according to the grammar rules for the Calc language. Add semantic-evaluation code to each of the recursive descent functions to carry out the execution of each of the expressions and statements as the parse proceeds. For each statement, the interpreter should output a line representing the result of the interpretation of that line (see sample run for examples).

You'll find on Canvas the "EGrammar" notes with details and code examples.

<u>Turn in</u>:
Submit via Canvas a “cleaned” ZIP of your Visual Studio C++ CalcInterp project directory that contains the full interpreter and scanner functionality as a working project.

- ➔ Before ZIP’ing the directory, please do a **Build/Clean** within Visual Studio, and **manually delete** the very large *.sdf file and the ipch directory within the project.

Examples:

```
/* t1.calc */
read A      /* Input the values */
read B

/*****Perform the computations*****/
sum := A + B
write sum
write sum / 2.0
```

might produce the interpretation output

```
Read: Enter value for A> 14
Read: Enter value for B> 3
Assign: SUM = 17
Write: 17
Write: 8.5
Press any key to continue
```

```
/* t2.calc */
read A      /* Input the values */
read
B

/*****Perform the computations*****/
sum:=A+B
dif := A - B
mul := A * b   div := A / B
write sum
write dif
write mul
write div

write sum / 2.0
```

might produce

```
Read: Enter value for A> 15
Read: Enter value for B> 3
Assign: SUM = 18
Assign: DIF = 12
Assign: MUL = 45
Assign: DIV = 5
Write: 18
Write: 12
Write: 45
Write: 5
Write: 9
Press any key to continue
```