Project 1
Implementation of a Lexer
Due Tuesday, April 14, 2020

Note: You need to COMPLETELY follow the instructions in this sheet.

1. **Problem:**

In this assignment you are requested to use the tool ANTLR to write a scanner for a small language Cactus. The language Cactus contains the following categories of token types:

1. An *identifier* is a sequence of letters and digits; the first character must be a letter. The underscore '_' counts as a letter. All identifiers are returned as a single token type.
2. The following identifiers are reserved for use as *keywords*, and may not be used otherwise:
   else fi if int main return while read write
   Each keyword is returned as a distinct token type.
3. An *integer constant* consists of a sequence of digits. All integer constants are returned as a single token type.
4. *Operators* include
   + - * / % == != > >= < <= && || ! = ( ) { } ;
   Each operator is returned as a distinct token type.
5. *Blanks*, *tabs*, and *newlines* are ignored except as they serve to separate tokens.
6. *Comments* are ignored except as they serve to separate tokens. The characters /* introduces a comment, which terminates with the characters */. Comments do not nest.

You can follow the following steps:

1. Edit a grammar Cactus.g4 that contains the regular expressions for each of the token types as follows.

```
// The grammar for Cactus language
grammar Cactus;

// Parser rules
token : (ELSE | … | ID | CONST | ADD | … )*
```

```
// lexer rules
ELSE : 'else'
   …
ID : …
CONST : …
ADD : '+'
   …
WHITESPACE : …
COMMENT : …
```

2. Use the ANTLR tool to generate the scanner and parser java code.

```
$antlr4 Cactus.g4
```

3. Compile the generated java code.

```
$javac Cactus*.java
```

4. Use the ANTLR tool to execute the scanner and parser.

```
$grun Cactus token -tree
```

If the input is as follows:

```
/* A program to sum 1 to n */
main()
{
        int n;
        int s;
        int i;

        read n;
        if ( n < 1 ) {
            write -1;
            return;
        } else {
            s = 0;
```

```
        } fi
        i = 1;
        while ( i <= n) {
            s = s + i;
            i = i + 1;
        }
        write s;
        return;
}
```

The output should be

(token main ( ) { int n ; int s ; int i ; read n ; if (n < 1) { write - 1 ; return ; } else { s = 0 ; } fi i = 1 ; while (i <= n) { s = s + i ; i = i + 1 ; } write s ; return ; })

2. Handing in your program:

To turn in the assignment, upload a compressed file proj1 containing Cactus.g4, Cactus.tokens, Cactus*.java, and Cactus*.class to eCourse2 site.

4. Grading

The grading is based on the correctness of your program. The correctness will be tested by a number of test cases designed by the instructor and teaching assistants.