**Project #1 - Lexical Scanner**

Version 1.5, Revised: 02/13/2016 14:48:19  
Due Date: Tuesday February 9th

**Purpose:**

The first two projects this Spring focus on building software tools for code analysis. We will emphasize C++ code but want our tools to be easily extendable to other similar languages like C# and Java.

Code analysis consists of extracting lexical content from source code files, analyzing the code's syntax from its lexical content, and building an Abstract Syntax Tree (AST) that holds the results of our analysis. It is then fairly easy to build several backends that can do further analyses on the AST to construct code metrics, search for particular constructs, or some other interesting features of the code.

You will find it useful to look at the [Parsing](http://www.ecs.syr.edu/faculty/fawcett/handouts/Webpages/BlogParser.htm) blog for a brief introduction to parsing and code analysis.

In this first project we will build and test a lexical scanner in C++ that consists of two packages:

* **Tokenizer**

extracts words, called tokens, from a stream of characters. Token boundaries are white-space characters, transitions between alphanumeric and punctuator characters, and comment and string boundaries. Certain classes of punctuator characters belong to single character or two character tokens so they require special rules for extraction.

* **SemiExpression**

groups tokens into sets, each of which contain all the information needed to analyze some grammatical construct without containing extra tokens that have to be saved for subsequent analyses. SemiExpressions are determined by special terminating characters: semicolon, open brace, closed brace, newline when preceeded on the same line with '#', and colon when preceded by one of the three tokens "public", "protected", or "private".

**Requirements:**

Your Scanner Solution:

1. **Shall** use Visual Studio 2015 and its C++ Windows Console Projects, as provided in the ECS computer labs.
2. **Shall** use the C++ standard library's streams for all I/O and new and delete for all heap-based memory management1.
3. (**2**) **Shall** provide C++ packages for Tokenizing, collecting SemiExpressions, and a scanner interface, ITokCollection.
4. (**4**) **Shall** provide a Tokenizer package that declares and defines a Toker class that implements the State Pattern2 with an abstract ConsumeState class and derived classes for collecting the following token types:
   * alphanumeric tokens
   * punctuator tokens
   * special one3 and two4 character tokens with defaults that may be changed by calling setSpecialSingleChars(string ssc) and/or setSpecialCharPairs(string scp).
   * C style comments returned as a single token
   * C++ style comments returned as a single token
   * quoted strings5
5. (**1**) The Toker class, contained in the Tokenizer package, **shall** produce one token for each call to a member function getTok().
6. (**4**) **Shall** provide a SemiExpression package that contains a class SemiExp used to retrieve collections of tokens by calling Toker::getTok() repeatedly until one of the SemiExpression termination conditions, below, is satisfied.
7. (**5**) **Shall** terminate a token collection after extracting any of the single character tokens: semicolon, open brace, closed brace. Also on extracting newline if a "#" is the first character on that line. Also, the extraction of a single character token consisting of ":" if immediately preceeded by one of the tokens: "public", "protected", or "private".
8. (**2**) **Shall** provide a facility providing rules to ignore certain termination characters under special circumstances. You are required to provide a rule to ignore the (two) semicolons within parentheses in a for(;;) expression6.
9. (**2**) The SemiExp class **Shall** implement the interface ITokenCollection with a declared method get().
10. (**5**) **Shall** include an automated unit test suite that exercises all of the special cases that seem appropriate for these two packages7.
11. That means that you are not allowed to use any of the C language I/0, e.g., printf, scanf, etc, nor the C memory management, e.g., calloc, malloc, or free.
12. <https://en.wikipedia.org/wiki/State_pattern>
13. Special one character tokens: <, >, [, ], (, ), {, }, :, =, +, -, \*, \n
14. Special two character tokens: <<, >>, ::, ++, --, ==, +=, -=, \*=, /=
15. "abc" becomes the token abc and the outer quotes are discarded. "\"abc\"" becomes the token "abc" with the outer quotes discarded.
16. This will be discussed in class.
17. This is in addition to the construction tests you include as part of every package you submit.

**What you need to know:**

In order to successfully meet these requirements you will need to know:

1. Basics of the C++ language: [http://CppReference.com](http://cppreference.com/)
2. How to implement a simple class hierarchy. This will be covered briefly in lecture #3 and in more detail later.
3. The [STL Containers](http://www.ecs.syr.edu/faculty/fawcett/handouts/CoreTechnologies/Cpp/Code/STL-Containers).
4. How to use Visual Studio. We will discuss this in one of the Help Sessions.