ENSC 251 – Summer 2017 – Course Project Part 1

Due: May 31, 2017, 10:00 pm

In this part of the course project you will be implementing a basic tokenizer, for a subset of the C++ language, that can convert a sequence of characters passed to the actual compilation stage of a compiler (or interpreting stage of an interpreter) for our subset of the C++ language into a sequence of strings each holding a token from the subset language. Any remaining preprocessing directives in the input should be stripped out. For an overview of the stages of building a single-file program written in C or C++, see:

https://calleerlandsson.com/the-four-stages-of-compiling-a-c-program/

for more information on the preprocessor, see:

https://gcc.gnu.org/onlinedocs/cpp/Preprocessor-Output.html

A token, for the purposes of this Course Project and in a syntactically correct program, is defined as the smallest number of consecutive characters that can be surrounded by whitespace without changing the functionality of the code.

‘Space, tab, linefeed, carriage-return, formfeed, vertical-tab, and newline characters are called "white-space characters" because they serve the same purpose as the spaces between words and lines on a printed page — they make reading easier. Tokens are delimited (bounded) by white-space characters and by other tokens, such as operators and punctuation.’ https://msdn.microsoft.com/en-us/library/e9a023cx.aspx

‘The punctuation and special characters in the C character set have various uses, from organizing program text to defining the tasks that the compiler or the compiled program carries out. They do not specify an operation to be performed. [In the full C-language,] some punctuation symbols are also operators (see Operators). The compiler determines their use from context.’ https://msdn.microsoft.com/en-us/library/eb1htw0t.aspx

String literals are a type of token. You will have to research what they and character literals can look like. Hint: here is a link with a useful tool to add and remove escape characters from a string: http://www.freeformatter.com/java-dotnet-escape.html -- it should be valid for C++ too.

Our subset of C++ language will use the following punctuators:

1. (- opening parenthesis
2.) - closing parenthesis
3. : - colon
4. ; - semicolon
5. * - indirection

Here is the list of operators for our subset of C++ language:

Category	**Operator**	**Associativity**
Postfix	++ --	Left to right
Unary	~	Right to left
Divisive	/ %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Conditional Operator	?	Right to left
Assignment	= += -= /= %= >>= <<=	Right to left

For example,

errno = 3+2;

can be split into the following set of tokens:

errno	=	3	+	2	;

> "When the compiler interprets tokens, it includes as many characters as possible in a single token before moving on to the next token. Because of this behavior, the compiler may not interpret tokens as you intended if they are not properly separated by white space. Consider the following expression:
> `i+++j`
>
> In this example, the compiler first makes the longest possible operator (`++`) from the three plus signs, then processes the remaining plus sign as an addition operator (`+`). Thus, the expression is interpreted as `(i++) + (j)`, not `(i) + (++j)`." https://msdn.microsoft.com/en-us/library/44kh05a0.aspx

A template including the required function for you to implement has been provided. You may implement any other helper functions also in the Part1Tokenizer.cpp file, but you are not allowed to remove or change the structure of the function 'tokenizeCodeStrip'. Take care with floating point numbers. For example, 1.575E1 is equivalent to 15.75 – they represent the same number. So,"1.575E1" is one token.

Refer to the BNF text file in the provided template. It contains a Backus-Naur Form (BNF) grammar for the tokens in our project.

Note, when testing your code we may use some tests that are correct according to our C++-subset grammar, some tests that include things not present in the grammar (like operators or punctuators not present in the grammar), and some tests that have other sorts of errors (like unterminated strings). When you find something not included in our subset grammar, including unterminated strings, issue a warning, ignore the problematic characters, push an empty string onto the back of the vector of strings to be returned so far, and then continue on tokenizing the remaining characters.

INSTRUCTIONS FOR SUBMITTING YOUR CODE:

- Do not modify the names of the source code files given to you.
- Please write high quality code with comments and good choice of variable names.