ASSIGNMENT 3

In this assignment, you will be implementing a basic parser for VHDL that can split a file into a set of tokens and then perform some simple processing on those tokens.

A **token**[1], for the purpose of this assignment, is defined as the smallest number of consecutive characters that can be separated by whitespace without changing the functionality of the code.
NOTE: To simplify your program, comment text should be treated as a single token.

The structure for both the parser (Tokenizer) and the storage element for the tokens (TokenList) have been provided. In the cases of the TokenList, we have also provided the implementation for appending new Tokens to the list. In this assignement, you will be focused on creating an implementation for the Tokenizer class.

The Tokenizer works by parsing a string into a set of tokens. To use the tokenizer you first call the tokenizer's `setString` function to set the current string to be parsed. The `isComplete` function lets you know if there are no remaining tokens in the string being parsed, and each call to `getNextToken`, returns the next token.

In order to split a string into tokens, you will need to identify a set of delimiters (character sequences) that can be used to identify boundaries between tokens and consider precedence between the delimiters. Examples include: whitespace characters, operators, and string quotes. **Note, for the purposes of this assignment you can assume that your input file has no syntax errors.**

Special cases (including but not limited to)**:**

- Bit vectors (''0010101'', x''1234ABCD'', '0', '1', etc)
- Comments (--)
  - The comment symbol should be an individual tokens (i.e. “--”). However, all text after the beginning of a comment should belong to the same token *independent of spaces* (see example below)
- Multi-character operators (eg. <=, :=, etc.)

Example (tokens are marked by a solid outline)

signal **control_line** : **std_logic** := **'0'** ; -- example comment

Note, gaps between boxes are whitespace, if there is no gap between two adjacent boxes then there was no whitespace between those two tokens.

Required: Before you begin the assignment familiarize yourself with the member functions of the C++ std string library. Specifically, you may find the functions: `find_first_not_of`, `find_first_of` **and** `find` **of particular interest.**

1 http://en.wikipedia.org/wiki/Lexical_analysis

Challenge Task: Challenge tasks are tasks that you should on perform if you have extra time, are keen, and want to show off. This challenge task is only worth 10% of your mark. If you don't complete the challenge task, the maximum score you can get on the lab is 90% (which is still an A+).

Your Challenge Task: Implement the `TokenList::deleteToken` function so that tokens can be removed from the doubly-linked TokenList class and implement the `removeComments` function to remove all comments from a list of tokens.

Building your assignment: For this assignment, we have provided you with a Makefile to help you build your assignment as well as to help you prepare your submission. To build your submission, simply type:

```
$ make
```

in the directory where the Makefile exists. Initially there will be some compiler errors as there are portions of code you need to complete (marked with /*Fill in implementation*/).

Testing: To assist you in your testing we have provided a sample main function (in assignment3.cpp) that you can use as a starting point to test your parser. The main function opens a test file, reads the file line-by-line and passes each line to the Tokenizer to be parsed. Upon completion, it calls the `removeComments` function and then outputs the list of tokens to stdout.

To test your parser, you should create test cases to cover as many possible valid syntax combinations as you can think of (feel free to share your test cases with each other on piazza). One way to confirm that you did not split a token into too many components is to take the outputted file and pass it through a vhdl syntax checker. Also, manually check that the number of keywords, operators, comments, etc. are correct. Be thorough with your testing!

For the challenge task, test your function that removes comments from the input file. Make sure it does not remove text past the end of a comment.

How will it be marked: We will be verifying the functionality of your parser by including your implementation of the Tokenizer, Token and TokenList classes in our own test application (Do **NOT** put any code you want marked in assignment3.cpp as it has been provided for testing purposes only). We will test your parser with a range of input files and will check for correctness in the generation of tokens and for the correct operation of the comment removing function (for the challenge task).

INSTRUCTIONS FOR SUBMITTING YOUR CODE

- ***Maintain the directory structure given to you. Do not modify the names or locations of any of the provided files.***

- For this assignment you will be submitting only the c++ files (i.e. “*.cpp” and “*.h”) and the Makefile for your assignment. (Again, do **NOT** put any code you want marked in assignment3.cpp as it will not be used in our testing of your submission)
- To submit your assignment, update the Makefile with your student number and run:

```
$ make tar
```

- **upload the resulting tar.gz file to the submission server.**