The University of Alabama in Huntsville
Electrical and Computer Engineering
Project 5 (20 points)
➔10% grading bonus if your program output matches the sample solution exactly←
Submit Your Solution Using ANGEL by Noon on Friday February 8, 2013
(A late submission drop box will be available on 02/8/13 from Noon to 2pm)

<Project 5 Directions>

On Project 5, you may only use concepts presented in Chapters 1 - 4 of your textbook!!

You are not allowed to use global variables. If necessary, global constants may be used. Global constants and variables are declared above the int main() line in a program.

Using your favorite text editor, type your solution and save it as a file named **Project_05.cpp** within your **CPE112_SPR13/Project_05** directory. If there are syntax errors, correct them and compile again. Once your program successfully compiles, run it and verify that the output for your program matches the output from the sample solution executable – **Project_05_solution**.

Once you are satisfied with your solution, submit Project_05.cpp via ANGEL.

The output of your program is to match as closely as possible the output of the sample solution. You may run the sample solution by typing the following at a command prompt in a terminal window (the terminal needs to be in the same directory that contains the downloaded input files)**:**

/home/work/cpe112/Executables/Project_05/Project_05_solution
Run the sample solution with the sample input files in P5_in.zip – Read the README.txt file
The README.txt file has instructions on input redirection with the input files provided

You can run the compare solution script for this project by typing the following at a command prompt in a terminal window that is in the directory containing your source code Project_05.cpp.

/home/work/cpe112data/Project_05/CompareSolution.bash Project_05.cpp

NOTE: make sure that you do not change the order in which the information is entered. An automatic script is used to process all lab submissions, and if the order of the input information is modified, the script will not work properly with your program.

<Project 5 Description>

For this project, you will write a complete C++ program that performs ***all of the tasks below***. Remember, your program must contain appropriate comments in order to receive full credit.

In this program a starting delimiter, ending delimiter and a line containing the delimiter will be read from the standard input stream (cin). The line read will be parsed based on the delimiters entered. The delimiters for a line will be unique and are to determine the sub string of the line to extract. The substring extracted does not contain the delimiters.

The output of the program consists of two parts. The first part outputs the contents of the extracted substrings followed by their length in brackets [ ]. The second part of the output consists of the number of characters in the two sub strings extracted, the number of characters in the original two lines read (this includes the characters in the delimiters, but not the new line character) and the percent of the characters of the lines that are contained within the delimiters.

The output of your program is to have the same format as the provided sample solution, and your output shall match the sample solution as closely as possible.

Steps/procedures of the program

(1) Read information for the two lines:

a. Prompt for, read and echo print the starting delimiter for the first line

b. Prompt for, read and echo print the ending delimiter for the first line

c. Prompt for, read (using getline) and echo print the first line to analyze

d. Repeat steps a, b and c for the second line

(2) Parse (divide) the two lines using the specified delimiters for the line.

a. The start and end delimiters used for parsing can appear anywhere in a line (starting delimiter first of course) and may have characters before or after them.

b. The delimiters are a unique character sequence in a line

c. Search for the delimiters based on their entire name – i.e. **<bold>**

(3) Output the content of the lines between the delimiters and the length of this content. The length is placed in brackets ([ ]) at the end of each line. Do not include the delimiters in this output.

(4) Output the total number of characters of the two lines contained within the delimiters

(5) Output the total number of characters in the two lines read. This total contains the characters in the delimiters as well; however, it does not include the new line character (this character is the only character not counted).

(6) Output the percent of the characters contained within the delimiters to the total number of characters for the lines. (item #4/Item#5*100.0) Use two decimal places in your answer. Watch out for integer division

<Project 5 Assumptions>

- Delimiters entered for a line will be contained in the line and they will be unique
- Starting delimiter will come before the end delimiter in a line.
- All input lines will be less than 200 characters in length.
- No testing of correctness of values entered is required.

<Project 5 Hints>

(1) You will likely need to use one or more of the built-in string functions from the C++ standard library in order to complete your program. (**Hint: length, find, substr**) To use these functions, include the **string header file** in your program.

(2) Read each line for processing into a separate string variable. All analysis of the lines is then performed on the contents of these string variables. Remember that the getline function will remove (but not keep) the end line (new line) character from the input stream

(3) Remember: character position counting starts at 0 not 1.

(4) Don't forget to echo print all values read from the standard input stream (cin)

(5) The length or size function can be used to determine how many characters are present in each delimiter that is entered. These lengths will be needed to correctly extract the substrings.

(6) You must search for the full delimiter name. Extraneous characters similar to the actual delimiters will be present in the line of data used for testing. The full delimiter names are the only unique strings in the line.

(7) Once you figure out how to parse the first line, copy and modify it for the next line (remember to make any necessary modifications to the variables in the code copied).

(8) Use setw() and left justification for the identifying phrases in the output. A value of 40 was used in the sample solution. To simplify your work, define an integer constant to represent the width, i.e. **const int WIDTH = 40;** then use cout << setw(WIDTH) << … Then if you need to modify the width to change the output appearance, you just have to change the value of the constant WIDTH.

(9) To extract the sub string between the delimiters,

a. First, find the starting character position of the first delimiter
b. Then find the starting character position of the second delimiter
c. Use the character positions found to determine the exact starting character position of the substring and the number of characters to extract. This step requires a little bit of thought and some basic math.

(10) Don't forget to use **string::size_type** for the data type of variables that hold the return values of the length(), size() and find() functions.

(11) The dashed lines contain 70 dashes in them. Each segment of code below prints out 70 dashes on a line. You can use either on in your program:

```
cout << setfill('-') << setw(70) << "-" << setfill(' ') << endl;
cout << string(70,'-') << endl;
```

(12) The program should determine/calculate the starting character positions and the number of characters to pull for the substrings – DO NOT USE literal values. The length(or size) and find functions are to be used to determine necessary values to create the substrings.