This assignment is due **Saturday, February 17th, by 6 pm**

- ***All components*** **(Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by Saturday, February 17th 6:00 pm for your solution to receive points.**
- **Recitation attendance is required to receive credit.**

Please follow the same submission guidelines outlined in Homework 3 description regarding Style, Comments and Test Cases. Here's a review below on what you need to submit for Recitation 5.

Develop in Cloud9: For this recitation assignment, write and test your solution using Cloud9.
Submission: All three steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.

1. ***Make sure your Cloud 9 workspace is shared with your TA:*** Your recitation TA will review your code by going to your Cloud9 workspace. *TAs will check the last version that was saved before the submission deadline*.
    - Create a directory called **Rec5** and place all your file(s) for this assignment in this directory.
    - Make sure to *save* the final version of your code (File > Save). Verify that this version displays correctly by going to File > File Version History.
    - The file(s) should have all of your functions, test cases for the functions in main function(s), and adhere to the style guide. Please read the **Test Cases** and **Style and Comments** sections included in the **Homework 3** write up for more details.
2. ***Submit to the Moodle Autograder:*** Head over to Moodle to the link **Rec 5**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.
3. ***Submit a zip file to Moodle:*** After you have completed all the questions and checked them on Moodle, ***you must submit a zip file with the .cpp file(s) you wrote in Cloud9***. Submit this file going to **Rec 5 (File Submission)** on moodle.

While Loops

Loops allow us to run a section of code multiple times. They will repeat execution of a single statement or group of statements as long as a specified condition continues to be satisfied. If the condition is not true, then the statement will not be executed.

Syntax and Form:

```
while (condition)
{
      //statement to do something;
}
```

where *while* is a C++ reserved word, *condition* is a boolean-expression that will evaluate to a true or false statement, and *statement to do something* is enclosed by curly brackets. If the condition is true, the specified statement within the loop is executed. After running once, the boolean-expression is then re-evaluated. If the statement is true, then it is executed again. This process of evaluation and execution is repeated until the condition becomes false.

Example 1:

```
int userChoice = 1;
while (userChoice != 0)
{
      cout << "Do you want to see this question again? Press 0 for no,
      any other number for yes.";
      cin >> userChoice;
}
```

Entering '0' will terminate the loop, but any other number will cause the loop to run again. Note how we have to initialize the condition before the loop starts. Setting *userChoice* equal to 1 ensures that the while loop will run at least once.

Example 2:

```
int i = 0
while (i < 5)
{
      cout<< i << endl;;
      i = i + 2;
}
```

Notice how you must manually initialize *i* to equal 0 and then manually increment i by 2.

Inserting print statements into your loops is a quick way to debug your code if something isn't working.

Strings

In C++ string is a type just like int or float. Strings, however, represent sequences of characters instead of a numeric value. A string literal can be defined using double quotes. So "Hello, world!", "3.1415", and "int i" are all strings. We can access the character at a particular index in a string using square brackets. Importantly, strings in C++ are zero-indexed. This means that the first character in a string is located at index 0. For example:

```
string s = "Hello, world!";
cout << s[0] << endl; //prints the character 'H' to the screen
cout << s[4] << endl; //prints the character 'o' to the screen
cout << s[6] << endl; //prints the character ' ' to the screen
cout << s[12] << endl; //prints the character '!' to the screen
```

There are a huge number of very useful functions available in C++ for string manipulation. One of the simplest is `length`. We can use this function to determine the number of characters in a string. This allows us to loop over a string character by character:

```
string s = "Hello, world!";
Cout << s.length();                    //This will print 13
for (int i = 0; i < s.length(); i++)
{
      cout << s[i] << endl;
}
```

This will print each character in the string "Hello, world!" to the screen one per line. Notice how the `length` function is called: `s.length()` and not `length(s)`. This is a special kind of function associated with objects, usually called a method, which we will discuss later in the course.

What happens in the above code snippet if we try to print characters beyond the length of the string? In particular, what happens when we replace `s.length()` with `s.length()+3`?

Problems Set:

Write a **function** for each of the following problems. You should first write your solution in Cloud9. Then copy and paste your function into the moodle quiz questions.

Note: You are required to also submit .cpp file(s) with your functions and a `main()` function with 2 tests for each of your functions.

Problem 1

A space (' ') is a single *character* in C++, just like 'a', 'A' , '5', or '%' in C++.

Write a function that takes a sentence and returns the number of words in the sentence.

- Your function should take **one** parameter:
  - a string parameter for the sentence
- Your function should return the number of words in the sentence.
- Your function should not print anything.
- Your function *MUST* be named **getWordCount**.

Examples:

- "Go" -> 1 word
- "I went" -> 2 words
- "Colorless green ideas dream furiously" -> 5 words
- "The rat the cat the dog bit chased escaped" -> 9 words

You can assume that all words will be separated by single spaces in the sentence. Think about the relationship between the number of space characters (' ') and the number of words.

Edge cases:

Make sure your function returns 0 if the string is empty (the length of the string is 0).

- sentence = "" -> 0 words

Problem 2

A digit is a character in the range 0-9.

Write a function to count the number of digits in a string.

- Your function should take **one** parameter:
  - a string parameter

- Your function should return the number of digit characters in the string
- Your function should not print anything.
- Your function *MUST* be named **getDigitCount**

Examples:

- "12345" -> 5 digits
- "a blue house" -> 0 digits
- "a0aaa" -> 1 digit
- "abre1567" -> 4 digits
- "009cD8" -> 4 digits
- "!%$o9bf&^" -> 1 digit

Problem 3

A *substring* refers to a string that is a continuous segment of a larger string. The list of all substrings of the string "apple" would be:

- ➔ "apple",
- ➔ "appl", "pple",
- ➔ "app", "ppl", "ple",
- ➔ "ap", "pp", "pl", "le",
- ➔ "a", "p", "p", "l", "e"
- ➔ ""

Write a function that takes a candidate substring and returns the number of times it occurs in another string.

- Your function should take **two** parameters *in this order*:
  - a string parameter for the substring
  - an string parameter for the string
- Your function should return the number of times the substring occurs in the string.
- Your function should not print anything.
- Your function MUST be named **getMatchCount**

Examples:

- "si", "mississippi" -> 2
- "ipp", "mississippi" -> 1

Edge cases:

Make sure your function can handle the following edge cases:

- substring is empty -> return -1
- string is empty -> return -1
- substring is longer than string -> return -2

Problem 4

Write a function to return the last index at which a character is used in a string.

- Your function should take **two** parameters *in this order*:
  - a char parameter
  - an string parameter
- Your function should return the last index at which the character occurs in the string.
- Your function should not print anything.
- Your function MUST be named **getLastIndex**

Examples:

- "o", "lollipop" -> index 6

l	**o**	**l**	**l**	**i**	**p**	**o**	**p**
0	1	2	3	4	5	6	7

- "p", “mississippi” -> index 9

m	**i**	**s**	**s**	**i**	**s**	**s**	**i**	**p**	**p**	**i**
0	1	2	3	4	5	6	7	8	9	10

Edge cases:

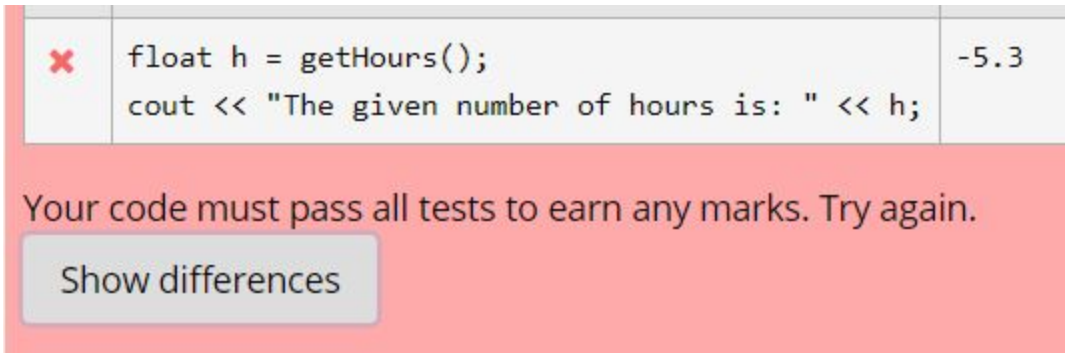
Make sure your function can handle the following edge cases:

- character not found in string -> return -2
- string is empty -> return -1

Submitting Your Code to the Autograder on Moodle

You must name the functions as indicated in each problem description. **Importantly**, *the cout formats provided for each problem are not suggestions – they **MUST** be followed precisely, word-for-word and including all punctuation marks*, otherwise the autograder will not recognize your results and you will not receive credit.

If there are errors in your solution to a particular problem, a button labeled "Show differences" will appear below the table of tests after you hit "check". This can be a very useful tool in helping you find small typos, especially in cout statements.



For example, below we hit "check" for a solution to problem 1 on this recitation and have failed all the test cases despite getting the correct values. Hitting "Show differences", we can see that a colon (:) is missing at the end of our prompt. When characters are in the expected output but not in your output they are highlighted in the "Expected" column.

Expected	Got	
Please enter how many hours you worked this week:↩ The given number of hours is: 36.5	Please enter how many hours you worked this week↩ The given number of hours is: 36.5	✖
Please enter how many hours you worked this week:↩ The given number of hours is: 12	Please enter how many hours you worked this week↩ The given number of hours is: 12	✖
Please enter how many hours you worked this week:↩ The given number of hours is: 0	Please enter how many hours you worked this week↩ The given number of hours is: 0	✖
Please enter how many hours you worked this week:↩ The given number of hours is: -5.3	Please enter how many hours you worked this week↩ The given number of hours is: -5.3	✖

On the other hand, when we include extra, unexpected characters in output they are highlighted in the "Got" column. Below we added the word "have" plus a space to the prompt.

Expected	Got	
Please enter how many hours you worked this week:↩ The given number of hours is: 36.5	Please enter how many hours you have worked this week:↩ The given number of hours is: 36.5	✖
Please enter how many hours you worked this week:↩ The given number of hours is: 12	Please enter how many hours you have worked this week:↩ The given number of hours is: 12	✖
Please enter how many hours you worked this week:↩ The given number of hours is: 0	Please enter how many hours you have worked this week:↩ The given number of hours is: 0	✖
Please enter how many hours you worked this week:↩ The given number of hours is: -5.3	Please enter how many hours you have worked this week:↩ The given number of hours is: -5.3	✖

You are required to also submit a .cpp file with your functions and a `main()` function with 2 tests for each of your functions.