CSCI 1300 CS1: Starting Computing
Instructor: Fleming/Wong, Spring 2019
Homework 5
Due Saturday, February 23, by 6 pm
+5% bonus if submitted by Friday, February 22, 11:59 pm

All 3 components (Cloud9 workspace, Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, February 23, by 6 pm for your homework to receive points.

---

1. Objectives

- Understand and work with strings.
- Traverse strings using loops.
- Make decisions based on the value of each character in a given string.
- Writing and testing C++ functions
    - Understand problem description
    - Design your function:
        - come up with a step by step algorithm,
        - convert the algorithm to pseudocode
        - imagine many possible scenarios and corresponding sample runs or outputs
    - Convert the pseudocode into a program written in the C++ programming language
    - Test it in the Cloud9 IDE and submit it for grading on Moodle

---

2. Background

Strings

In C++, `string` is a data 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 location within a string by using square brackets, which enclose an ***index*** which you can think of as the ***address*** of the character within the string. Importantly, strings in C++ are indexed starting from zero. This means that the first character in a string is located at index 0, the second character has index 1, and so on. 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 many useful functions available in C++ to manipulate strings. 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 (i.e. *traverse the string*):

```
string s = “Hello, world!”;
cout << s.length() << endl;            //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.

The correct way:

- `s.length()`

Common mistakes:

- `length(s)`
- `s.length`
- `s.size()`

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`?

3. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Create Hmwk5 directory on your Cloud 9 workspace:*** 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 **Hmwk5** 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 **submission file instructions** under Week 4.

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 5 CodeRunner**. 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 6 questions from the Moodle assignment, zip all 6 files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 5 (File Submission)** link on Moodle.

4. Rubric

Aside from the points received from the **Homework 5 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 5 (File Submission)** link on Moodle and assign points for the following:

Style and Comments **(5 points):**

- Your code also should be well styled. Code that has good style makes it easier for others to understand what you are doing in your program. The coding style includes white space, indentation, using global variables and naming variables. In professional work environments, you're expected to obey the company's coding style guide to keep the consistency within the company. The style guide is posted on Moodle under week 6.

- Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to help other developers understand how your code works. These comments should begin with two backslashes (//) or the multi-line comments (/* … *comments here*… */) .

- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Spring 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/…
// Homework 5 - Problem # ...
```

Algorithm **(5 points):**

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the function.
- This is an example C++ solution. Look at the code and the algorithm description for an example of what is expected.

Example 1:

```
/*
 * Algorithm: convert money from U.S. Dollars (USD) to Euros.
 *    1. Take the value of number of dollars involved in the transaction.
 *    2. Current value of 1 USD is equal to 0.86 euros
 *    3. Multiply the number of dollars got with the currency
 *       exchange rate to get Euros value
 *    4. Return the computed Euro value
 * Input parameters: Amount in USD (double)
 * Output (prints to screen): nothing
 * Returns: Amount in Euros (double)
 */
```

Example 2:

```
double convertUSDtoEuros(double dollars)
{
	double exchange_rate = 0.86; //declaration of exchange rate
	double euros = dollars * exchange_rate; //conversion
	return euros; //return the value in euros
}
```

The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

```
/*
 * conversion
 */
double convertUSDtoEuros(double dollars)
{
	double euros = dollars * 0.86;
	return euros;
}
```

Test Cases (20 points):

1. *Code compiles and runs* (**6 points**):
    - The zip file you submit to Moodle should contain **6** full programs (each with a `main()` function), saved as .cpp files. It is important that your programs can be compiled and run on Cloud9 with no errors. The functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**14 points**):
   For this week's homework, all 6 problems are asking you to create a function. In your Cloud9 solution file for each function, you should have 2 test cases present in their respective `main()` function, for a total of 12 test cases (see the diagram on the next page). Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.

Please make sure that your submission files follow the the **submission file instructions** under week 6.

5. Problem Set

***Note: To stay on track for the week, we recommend to finish/make considerable progress on problems 1-3 by Wednesday. Students with recitation on Thursday are encouraged to come to recitation with questions and have made a start on* all *of the problems*.**

Problem 1 (5 points): getDigitCount

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 argument, of type `string`.
- Your function should return an integer
    - The return value will be the number of digit characters in the string
- Your function should not print/display/`cout` anything to the screen.
- Your function *MUST* be named **getDigitCount**

Examples:

- Input parameter: **`"12345"`** ➔ return **`5`**;
- Input parameter: **`"a blue house"`** ➔ return **`0`**;
- Input parameter: **`"a0aaa"`** ➔ return **`1`**;
- Input parameter: **`"abre1567"`** ➔ return **`4`**;
- Input parameter: **`"009cD8"`** ➔ return **`4`**;
- Input parameter: **`"!%o9bf&^"`** ➔ return **`1`**;

In Cloud9 the file should be called **getDigitCount.cpp** and it will be one of 5 files you need to zip together for the **Homework 5 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 CodeRunner**. For Problem 1, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 2 (10 points): getWordCount

A space (`' '`) is a single *character* in C++, just like `'a'`, `'A'` , `'5'`, and `'%'` (for example). In English, a space is used to separate individual words. Write a function that takes an English 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**.
- ***Note****: You can assume there is exactly one single space between any two words.*

Examples:

- Input parameter: `""` (an empty string) ➔ return `0`;
- Input parameter: `"Go"` ➔ return `1`;
- Input parameter: `"I went"` ➔ return `2`;
- Input parameter: `"Colorless green ideas dream furiously"` ➔ return `5`;
- Input parameter: `"The rat the cat the dog bit chased escaped"` ➔ return `9`;

In Cloud9 the file should be called **getWordCount.cpp** and it will be one of 5 files you need to zip together for the **Homework 5 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 CodeRunner**. For Problem 2, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 3 (20 points): split

Write a function **split** which takes two input arguments: a string to be split, a character to split on (also known as a ***delimiter***). The function will split the input string into smaller substrings, separated by the delimiter, and print each substring. Your function will return the number of pieces the string was split into.

- Your function should be named **split**
- Your function takes **two** input parameter arguments in the following order:
  - The `string` to be split.

    - A delimiter `character`, which marks where the input string should be split up.
- Your function must **return** the number of substrings that the input string was split into as an `integer`.
- Your function prints to the screen each substring that the delimiter separates the string into, each on its own line.

Note 1: It's possible that the string to be split will have the delimiter character more than once in a row. Sequences of repeated delimiters should be treated as a single delimiter. It's also possible for a string to start or end with a delimiter, or a sequence of delimiters. These "leading" and "trailing" delimiters should be ignored by your function; see the examples.

Note 2: Consider the case where the delimiter is not present. What, then, should be the substring(s) that your function prints? How many substrings does such a string have?

Example output:

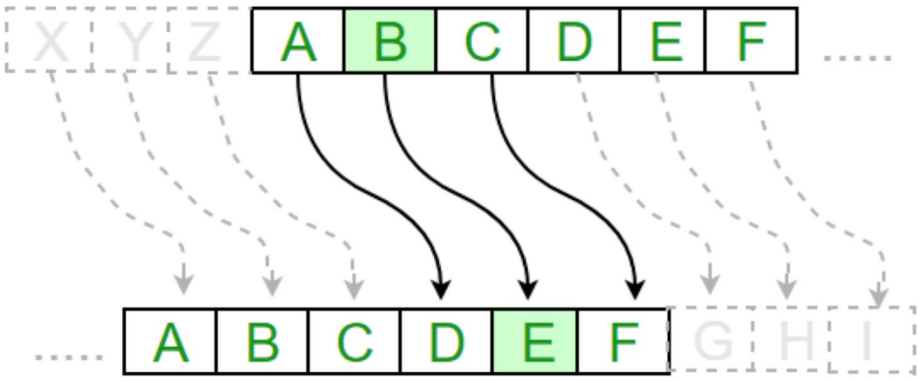
Input	split returns...	Output to screen...
`split("one small step", ' ');`	3	one small step
`split(" one small step  ", ' ');`	3	one small step
`split("cow/big pig/fish", '/');`	3	cow big pig fish
`split("cow/big pig//fish", '/');`	3	cow big pig fish
`split("unintentionally", 'n');`	5	u i te tio ally

In Cloud9 the file should be called **split.cpp** and it will be one of 5 files you need to zip together for the **Homework 5 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 CodeRunner**. For Problem 3, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 4 (15 points): Caesar Ciphers caesarCipher

In cryptography, a ***cipher*** is an algorithm for encrypting and decrypting some given message through a series of well-defined steps. Commonly called codes, there are various types of ciphers ranging from fairly simple to complex. One of the most widely known encryption schemes is called a *Caesar Cipher* or *Shift Cipher*. The Caesar Cipher encrypts a letter by substituting for it with another letter that is a fixed distance, or *shift*, away in the alphabet. For example, to use a shift of +3 in a Caesar Cipher, we would substitute the letter D for A, E for B, and so on, as illustrated below.



As you can see in the image above, each letter is being replaced with the letter 3 positions forward in the alphabet. For this problem you will be creating a function which performs both encryption and decryption of a message using this scheme.

Write a function **caesarCipher** which will either encrypt a message or decrypt a coded message using a Caesar Cipher.

- Your function **MUST** be named **caesarCipher**
- Your function takes three input arguments *in this order*: a `string message`, an `int key`, and a `bool flag`
- Your function must **return** the resulting string from the encoding or decoding algorithm. It should **NOT print** the resulting string using `cout`.

- `message` will be a string in all capital letters with spaces. Unencoded messages will be strings such as `"I LIKE CHOCOLATE"` or `"HELLO WORLD"`, and encoded messages will be strings such as `"L OLNH FKRFRODWH"` or `"KHOOR ZRUOG"`.
- `key` will be an integer that specifies how many positions in the alphabet are being shifted in the encoding or decoding process. A valid `key` must be between 0 and 25. If your function is given a value outside of this range, it should `return "ERROR";`
- `flag` will be a boolean variable that controls whether your function will be encrypting or decrypting the given message. A value of `true` will mean your function is encoding the given message, and a value of `false` will mean your function is decoding it.

Example:

- If the input arguments are `caesarCipher("ABCD", 2, true)`, the function should **return** the string `"CDEF"`.
- If the input arguments are `caesarCipher("TOY STORY", 27, true)`, the function should **return** the string `"ERROR"`.
- If the input arguments are `caesarCipher("ABCD", 2, false)`, the function should **return** the string `"YZAB"`.

In Cloud9 the file should be called **caesarCipher.cpp** and it will be one of 5 files you need to zip together for the **Homework 5 (File Submission)**on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 CodeRunner**. For Problem 4, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 5 (20 points): Vigenere Cipher

The ***Vigenere cipher*** is a type of polyalphabetic cipher. For the *Caesar cipher* each letter of the alphabet is substituted by a new letter based on a constant *shift value*. That makes the Caesar cipher easy enough to break. For the *Vigenere cipher,* the *shift value* is variable for each letter of the *input text*. The *shift value* is calculated based on the letters from the string *keyword* (explained below) rather than a fixed, constant number. One letter from the *keyword* is chosen in a *dynamic* fashion based on the position of the input letter to be encrypted. Let's look at the encryption algorithm through the example below.

There are two components to this type of encryption, *keyword* and the *input text*. If the *keyword* is TIGER and the *input text* we are trying to encode is BATMAN, encryption happens as follows:

The first letter ‘B’ of BATMAN is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S

The encrypted alphabet is a regular alphabet just shifted to start from T (first letter of the *keyword)*. Therefore, the letter ‘B’ becomes ‘U’ on encryption.

The second letter ‘A’ of BATMAN is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H

Again, the encrypted alphabet is a regular alphabet just shifted to start from I (second letter of the *keyword)*. Hence, the letter ‘A’ becomes ‘I’ on encryption.

The third letter ‘T’ of BATMAN is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F

The letter ‘T’ becomes ‘Z’ on encryption.

The fourth letter ‘M’ of BATMAN is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D

The letter ‘M’ becomes ‘Q’ on encryption.

The fifth letter ‘A’ of BATMAN is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q

The letter ‘A’ is encrypted as ‘R’. *Notice that this ‘A’ is encrypted differently than the first ‘A’*

The sixth letter ‘N’ of BATMAN is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S

The letter ‘N’ becomes ‘G’ on encryption.

We can summarize this process in three steps:

1. For each input letter find the corresponding letter from the *keyword*. This is done by looking at the position of the letter in the *input text*, then, locating the letter from the *keyword* at the same position.

   Note: if the *keyword* is shorter (has less characters) than the *input text* we need to loop back to the start, and use the letters from the *keyword* again. In the above example, notice that TIGER is shorter than BATMAN. TIGER has 5 characters. 'B' is the character at position 0 in BATMAN, and 'N' is the character at position 5, and they both follow the same encryption rules, based on the character at position 0 from TIGER, the letter 'T'

2. Create a new alphabet sequence shifted to start from the above computed letter.

3. Find the 'encrypted letter' for the input letter from the above created alphabet sequence.

- Your function **MUST** be named **vigenereCipher.**
- Your function takes three input arguments:
  - **message** - a string in all capital letters with spaces. Unencoded messages will be strings such as `"I LIKE CHOCOLATE"` or `"HELLO WORLD"`, and encoded messages will be strings such as `"B TOOV VPUGFEIZI"` or `"AMRPF PWXPU"`.
  - **key** - a **`string`** which is used to encrypt the message. It should be all capital letters. For e.g "TIGER", "CATS".
  - **flag** - a **`bool`** variable that controls whether your function will be encrypting or decrypting the given message. A value of `true` will mean your function is encrypting the given message, and a value of `false` will mean your function is decrypting it.
- Your function should not print anything.
- Your function should return the encrypted/decrypted message: a **string** value**.**

Examples:

- If the input arguments are `vigenereCipher("UNICORNS", "TIGER", true)`, the function should return `NVOGFKVY` as the encrypted message.
- If the input arguments are `vigenereCipher("UNI CORNS", "TIGER", true)`, the function should return `NVO GFKVY` as the encrypted message.
- If the input arguments are `vigenereCipher("NVOGFKVY", "TIGER", false)`, the function should return `UNICORNS` as the encrypted message.

In Cloud9 the file should be called **vigenereCipher.cpp** and it will be one of 5 files you need to zip together for the **Homework 5 (File Submission)** on Moodle.
Don't forget to head over to Moodle to the link **Homework 5 CodeRunner**. For Problem 6, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

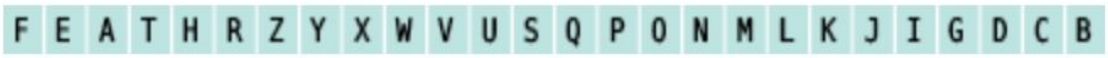
Extra Credit Problem (15 points): Random Monoalphabet Ciphers

keywordCipher

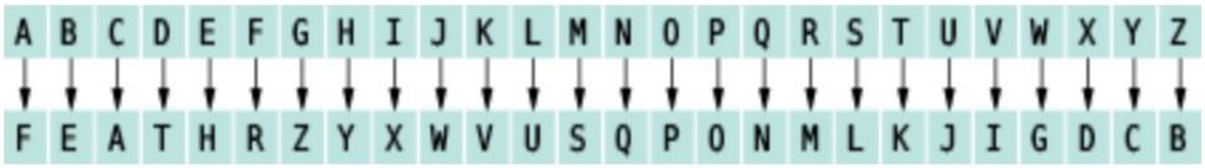
The Caesar cipher shifts all letters by a fixed number. That is good and all, but it is relatively easy to break those codes and decrypt messages that are not meant for you!

Instead, here is a better idea. As the key, don't use numbers but *words* instead. A ***keyword cipher*** is a form of monoalphabetic substitution. A keyword is used as the key, and it determines the letter matchings of the cipher alphabet to the plain alphabet.

For example, suppose the key word is FEATHER. First, remove duplicate letters, yielding FEATHR. Then, append the other 20 letters of the alphabet in **reverse order**:



Now encryption of any letter in your message is done as follows:



With FEATHER as the keyword, all A's become F's, all B's become E's, all C's become A's and so on. Hence, the message "UGLY DUCKLING", will be encrypted to "JZUC TJAVUXQZ" using the above cipher. To decrypt a message, run the encryption in reverse and match letters in the top row to those in the bottom. For example, the encrypted message "EFKSFQ" will be decrypted to "BATMAN".

Write a function **keywordCipher** which will encrypt or decrypt a message using a keyword cipher.

- Your function **MUST** be named **keywordCipher.**
- Your function takes three input argument:
    - **message** - a **`string`** in all capital letters, and can also have spaces. For example, `message` can be “ZOMBIE HERE”, “BATMAN”.
    - **key** - a **`string`** which is used to encrypt the message. It should be in all capital letters, and can have spaces. For e.g “SECRET”, “STAR WARS”.
    - **flag** - a **`bool`** variable that controls whether your function will be encrypting or decrypting the given message. A value of `true` will mean your function is encrypting the given message, and a value of `false` will mean your function is decrypting it.

- Your function **MUST RETURN** the resulting encrypted or decrypted **`string`**.
- You may assume that all input that your function will be tested against will contain only capital (uppercase) letters and spaces. That is, you do not need to implement a encryption method for lowercase letters, numbers, and other special symbols.

Notes:

- Your function should remove duplicate characters from the key before proceeding to do anything else. *Hint: Iterate through the key and store each character in a new string. Before storing, check if the character already exists in the new string.*
- Your function should handle spaces in the message. *Hint: check if the character is equal to ‘ ‘, then just append it to the encrypted/decrypted/cipher string as it is.* The space ‘ ‘ characters do not get encrypted.

Examples:

- If the input arguments are `keywordCipher(“ZOMBIE HERE”, “SECRET”, true)`, the function should return `ANPEWT XTKT` as the encrypted message.
- If the input arguments are `keywordCipher(“ANPEWT XTKT”, “SECRET”, false)`, the function should return `ZOMBIE HERE` as the decrypted message.

In Cloud9 the file should be called **keywordCipher.cpp** and submit it to the **Homework 5 -- Extra Credit Problem (File Submission)** on Moodle (submit cpp file, not a zip file).

Don’t forget to head over to Moodle to the link **Homework 5 CodeRunner Quiz -- Extra Credit Problem** For this extra credit, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. Homework 5 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 5 CodeRunner**
2. Submit one zip file to **Homework 5 (File Submission)**. The zip file should be named, **<firstName>_<lastName>_homework5.zip**. It should have following 6 files:
    - **getDigitCount.cpp**
    - **getWordCount.cpp**
    - **split.cpp**
    - **caesarCipher.cpp**
    - **vigenereCipher.cpp**
3. If you work on the extra credit problem (keywordCipher), submit your solution to **Homework 5 CodeRunner Quiz -- Extra Credit Problem** and **keywordCipher.cpp** to **Homework 5 -- Extra Credit Problem (File Submission)** on Moodle (cpp file, not a zip file).

7. Homework 5 point summary

Criteria	**Pts**
CodeRunner (problem 1 - 6)	70
Style and Comments	5
Algorithms	5
Test cases	20
Recitation attendance (Feb 19 or Feb 21)*	-30
Total	100
5% early submission bonus	+5%
Extra credit question	+15

* if your attendance is not recorded, you will lose points. Make sure your attendance is recorded on Moodle.