CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Project 1: DNA, Deoxyribonucleic acid
Due: Saturday, September 28th, by 6:00 pm
No bonus points for early submission

Three components (Moodle CodeRunner attempts, zip file, and inerivew grading) must be completed and submitted by Saturday, September 28th, 6:00 pm for your homework to receive points. Project 1 requires you to have an interview grading with your TA by October 12th.

---

0. Table of Contents

---

1. Objectives

- Understand and work with `string`.
- Understand and work with `while` loops and `for` loops.
- Be able to create menu-driven applications
- 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, then 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` in the above `for` loop?

---

3. Submission Requirements

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

1. ***Work on questions on your Cloud 9 workspace:*** You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **project1** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Project 1 CodeRunner**. You will find one programming quiz question for each problem in the assignment. Submit your solution for each 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.

3. ***Submit a .zip file to Moodle:*** After you have completed all 7 questions from the Moodle assignment, zip all 7 files you compiled in Cloud9 and submit the zip file through the **Project 1** link on Moodle.

---

4. Rubric

Aside from the points received from the **Project 1 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Project 1** link on Moodle and assign points for the following:

Style, Comments, Algorithm:

Style:

- Your code should be well-styled, and we expect your code to follow some basic guidelines on whitespace, naming variables and indentation, to receive full credit. Please refer to the **CSCI 1300 Style Guide** on Moodle.

Comments:

- 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 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Project 1 - Problem # ...
```

Algorithm:

- 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:

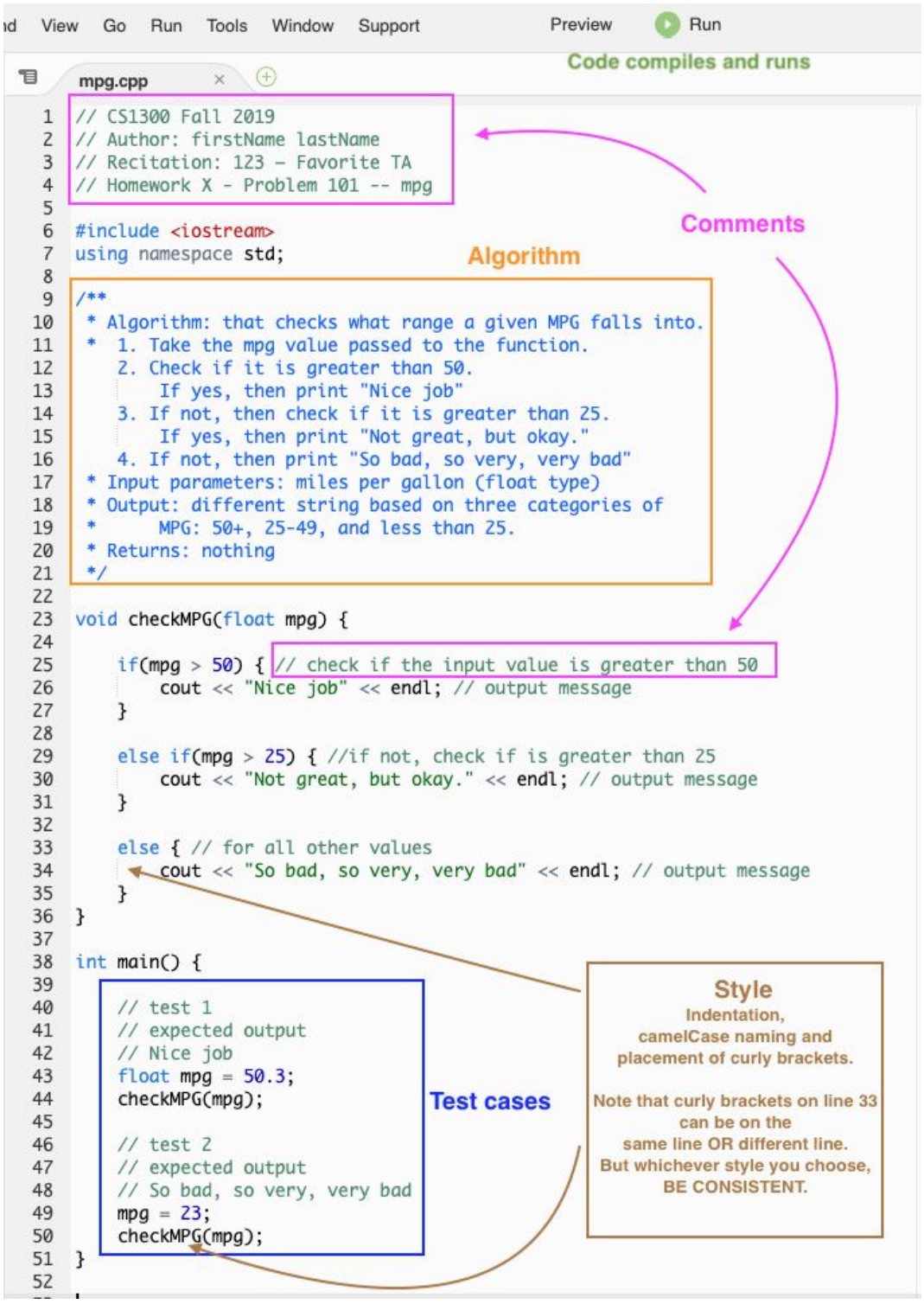
```
double convertUSDtoEuros(double dollars)
{
     double exchange_rate = 0.86; //declaration of exchange
rate
     double euros = dollars * exchange_rate; //conversion
     return euros; //return the value in euros
}
```

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

In your Cloud9 solution file, for each function, (except problem 3 and problem 8), You should have enough test cases to verify: every condition with true and false values, every branch of every decision, and every loop for executing 0 times, and 1 or more times. Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.




```
1  // CS1300 Fall 2019
2  // Author: firstName lastName
3  // Recitation: 123 - Favorite TA
4  // Homework X - Problem 101 -- mpg
5
6  #include <iostream>
7  using namespace std;
8
9  /**
10  * Algorithm: that checks what range a given MPG falls into.
11  *  1. Take the mpg value passed to the function.
12     2. Check if it is greater than 50.
13         If yes, then print "Nice job"
14     3. If not, then check if it is greater than 25.
15         If yes, then print "Not great, but okay."
16     4. If not, then print "So bad, so very, very bad"
17  * Input parameters: miles per gallon (float type)
18  * Output: different string based on three categories of
19  *      MPG: 50+, 25-49, and less than 25.
20  * Returns: nothing
21  */
22
23 void checkMPG(float mpg) {
24
25     if(mpg > 50) { // check if the input value is greater than 50
26         cout << "Nice job" << endl; // output message
27     }
28
29     else if(mpg > 25) { //if not, check if is greater than 25
30         cout << "Not great, but okay." << endl; // output message
31     }
32
33     else { // for all other values
34         cout << "So bad, so very, very bad" << endl; // output message
35     }
36 }
37
38 int main() {
39
40     // test 1
41     // expected output
42     // Nice job
43     float mpg = 50.3;
44     checkMPG(mpg);
45
46     // test 2
47     // expected output
48     // So bad, so very, very bad
49     mpg = 23;
50     checkMPG(mpg);
51 }
52
```

5. Start With These Problems…

… but be sure to have a look at #6 to plan your time, because it is a much larger task! These will prepare you for tackling #6, DNA analyzer

Problem 1(4 points): printCharInString

Write a function **printCharInString** that takes a string argument and prints all the characters in the string on a new line.

- Your function **MUST** be named **printCharInString**
- Your function takes **one parameter**: a string
- If your function receives an empty string, then it must print "`Given string is empty!`"
- If your function receives a string that is not empty, then it must print all the characters in the string with each character on a new line.
- Your function should **NOT return** anything

Examples:

- When the argument is "`Computer`", the function should print:

```
C
o
m
p
u
t
e
r
```

- When the argument is equal to `""`, the function should print:

```
Given string is empty!
```

In Cloud9 the file should be called **printCharInString.cpp** and it will be one of the files you need to zip together for the **Project 1** on Moodle.

Don't forget to head over to Moodle to the link **Project 1 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 (8 points): countMatches

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 **countMatches** that searches the substring in the given string and returns how many times the substring appears in the string.

- Your function MUST be named **countMatches**
- Your function takes two strings parameters in the following order
  - String where the substring is searched
  - Substring whose count is to be found.
- Your function should return the number of matches as an integer value
- Your function should return -1 if one or both parameters are empty. If not, your function should return the number of occurrences of a substring in a given string.

Example:

- "mis**si**s**si**ppi", "si" -> 2
- "mississ**ipp**i", "ipp", -> 1

In Cloud9 the file should be called **countMatches.cpp** and it will be one of the files you need to zip together for the **Project 1** on Moodle.

Don't forget to head over to Moodle to the link**Project 1 CodeRunner**. For Problem 2, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button.

Problem 3 (8 points): zootopia1300


The Zootopia Police Department is recruiting new officers and has come up with an innovative equation to hire. They define `hireScore` as a weighted sum of `agility`, `strength` and `speed`.

`hireScore` = 1.8 * `agility` + 2.16 * `strength` + 3.24 * `speed`

The candidates for this hiring season are foxes, bunnies and sloths. Chief Bogo has requested you to write a program that takes in these attributes and produces a `hireScore` for the candidate.

The program should provide a menu to choose the anthropomorphic animal. The menu should have the following options (`printMenu` function is posted on Moodle):

1. Fox
2. Bunny
3. Sloth
4. Quit

Once an animal is selected, the program should ask two of the characteristics based on the animal. The rules for which are:

1. Input `agility` and `strength` for a Fox
2. Input `agility` and `speed` for a Bunny
3. Input `strength` and `speed` for a Sloth

The program should then compute the `hireScore` based on the inputs using the weighted sum formula. The computed `hireScore` should be displayed on the screen. The menu will run in a loop, continually offering Bogo four options until he chooses to quit.

- You MUST submit the entire program including the main
- There are 4 types of user inputs in total
    - menu option which is of type Integer
    - `agility` which is of type double
    - `strength` which is of type double
    - `speed` which is of type double
- Your program needs to print the menu, input prompts and the `hireScore` as shown in the examples below.

Note: For every animal we only ask for two parameters out of the three, assume the other one to be zero. For example, the `speed` of a fox can be assumed to be zero.

Example 1:

```
Select a numerical option:
=== menu ===
1. Fox
2. Bunny
3. Sloth
4. Quit
1
Enter agility:
10.5
Enter strength:
20.0
Hire Score: 62.1
Select a numerical option:
=== menu ===
1. Fox
2. Bunny
3. Sloth
4. Quit
4
```

Example 2:

```
Select a numerical option:
=== menu ===
1. Fox
2. Bunny
3. Sloth
4. Quit
3
Enter strength:
0.5
```

```
Enter speed:
0.2
Hire Score: 1.728
Select a numerical option:
=== menu ===
1. Fox
2. Bunny
3. Sloth
4. Quit
4
```

In Cloud9 the file should be called **zootopia1300.cpp** and it will be one of the files you need to zip together for the **Project 1** on Moodle.

Don't forget to head over to Moodle to the link **Project 1 CodeRunner**. For Problem 3, in the Answer Box, paste **your entire program**(including your main). Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. DNA Analyzer

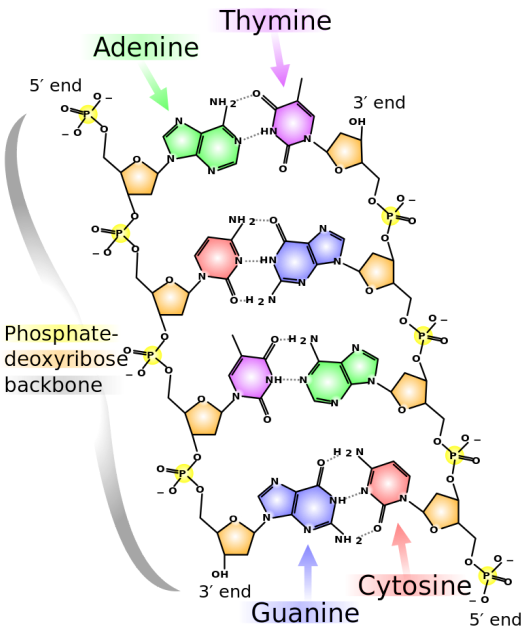
(4 problems, 80 points)

Background

Measuring DNA Similarity

DNA is the hereditary material in humans and other species. Almost every cell in a person's body has the same DNA. All information in DNA is stored as code in four chemical bases: adenine (**A**), guanine (**G**), cytosine (**C**) and thymine (**T**). The differences in the order of these bases is a means of specifying different information.

We refer to an organism's complete set of chemical bases as their *genome*. Every genome looks something like this:

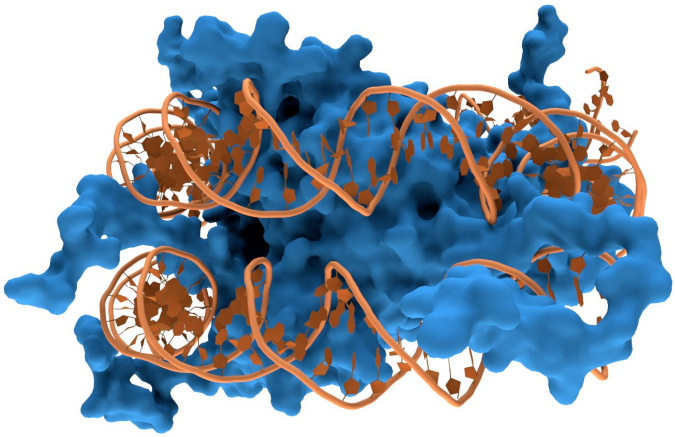


GATCAATGAGGTGGACACCAGAGGCGGGGACTTGTAAATAACACTGGGCTGTAGGAGTGATGGGGTTCACCTCTAATTCTAAGATGGCTAGATAAT
GCATCTTTCAGGGTTGTGCTTCTATCTAGAAGGTAGAGCTGTGGTCGTTCAATAAAAGTCCTCAAGAGGTTGGTTAATACGCATGTTTAATAGTACA
GTATGGTGACTATAGTCAACAATAATTTATTGTACATTTTTAAATAGCTAGAAGAAAAGCATTGGGAAGTTTCCAACATGAAGAAAAGATAAATGGTC
AAGGGAATGGATATCCTAATTACCCTGATTTGATCATTATGCATTATATACATGAATCAAAATATCACACATACCTTCAAACTATGTACAAATATTATAT
ACCAATAAAAAATCATCATCATCATCTCCATCATCACCACCCTCCTCCTCATCACCACCAGCATCACCACCATCATCACCACCACCATCATCACCAC
CACCACTGCCATCATCATCACCACCACTGTGCCATCATCATCACCACCACTGTCATTATCACCACCACCATCATCACCAACACCACTGCCATCGTCA
TCACCACCACTGTCATTATCACCACCACCATCACCAACATCACCACCACCATTATCACCACCATCAACACCACCACCCCCATCATCATCATCACTAC
TACCATCATTACCAGCACCACCACCACTATCACCACCACCACCACAATCACCATCACCACTATCATCAACATCATCACTACCACCATCACCAACACC
A

Human Genome: First 1000 lines of Chromosome 1

The human genome has about 3 billion DNA base pairs. That means the entire human genome can be represented by a (very long) string of ~3 billion characters, where every character in the string is A, C, G, or T. You can find the first 1000 lines of Chromosome 1 of the Human Genome at this webpage.

In the lectures, there have been examples of string representation and use in C++. In this assignment, we will create strings that represent DNA sequences. We will be implementing a number of functions that are used to search for substrings that represent sequences within the DNA.



Interaction of DNA (orange) with histones (blue), a DNA binding protein. These proteins' basic amino acids bind to the acidic phosphate groups on DNA.
Thomas Splettstoesser

One of the challenges in computational biology is determining where a DNA binding protein will bind in a genome. Each DNA binding protein has a preference for a specific sequence of nucleotides. This preference sequence is known as a motif. The locations of a motif within a genome are important to understanding the behavior of a cell's response to a changing environment.

To find each possible location along the DNA, we must consider how well the motif matches the DNA sequence at each possible position. The protein does not require an exact match to bind and can bind when the sequence is similar to the motif.

Another common DNA analysis function is the comparison of newly discovered DNA genomic sequences to the large databases of known DNA sequences and using the similarity of the sequences to help identify the origin of the new sequences.

Goal of your project

Suppose that the emergency room of a hospital sees a sudden and drastic increase in patients presenting with a particular set of symptoms. Doctors determine the cause to be bacterial, but without knowing the specific species involved they are unable to treat patients effectively. One way of identifying the cause is to obtain a DNA sample and compare it against known bacterial genomes. With a set of similarity scores, doctors can make more informed decisions regarding treatment, prevention, and tracking of the disease.

Write a program that contains three main parts:

1) calculate the similarity score,
2) find the best similarity score in the given genome,
3) given three genomes and a sequence from the unknown bacteria, determine the most probable match.

We've nicely broken down this program into three functions. You're welcome to write additional helper functions as you need. Write your code and test each problem on your Cloud 9. Then, once you complete, you can submit it on Moodle coderunner to make sure it's fully functional.

Problem 4 (5 points): calcSimScore

Write a function calcSimScore that returns the similarity score for two sequences. The similarity score for two sequences is calculated as follows:

similarity_score = (string_length - hamming_distance) / string_length

Where hamming distance is the number of positions at which the corresponding characters are different. Two strings with a small Hamming distance are more similar than two strings with a larger Hamming distance. The two strings must always be the same length when calculating a Hamming distance. If the length of the genomes is different, then the similarity score cannot be calculated. Similarity score will be in the range [0.0,1.0].

Example: first string = “ACCT” second string = “ACCG”

A C C T
| | | *
A C C G

In this example, string_length = 4. Since there is one mismatch, the hamming_distance = 1 . And the similarity score for the two strings on this example would be, similarity_score = (4-1)/4 = 3/4 = 0.75

- Your function MUST be named **calcSimScore**
- Your function should take two parameters in this order:
  - Sequence 1, a string
  - Sequence 2, a string
- Your function should return the similarity score as a double
- If the length of the strings is the same, then you can calculate the similarity score. If not, the similarity score would be 0.
- Your function should not print/display/cout anything to the screen.

Examples:
Arguments and their expected return values for the calcSimScore

sequence1	**sequence2**	**Similarity score**
ATGC	ATGA	0.75
CCDCCD	CCDCCD	1
ATG	GAT	0
ACCDT	ACT	0

In Cloud9 the file should be called **calcSimScore.cpp** and it will be one of the files you need to zip together for the **Project 1** on Moodle. Make sure that your main has all the test cases that are needed to check the function. After developing in Cloud9, this function will be one of the functions needed for Problem 8.

Once you test your code on your Cloud 9 workspace, head over to Moodle to the project 1 CodeRunner. For this problem, 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 (10 points): findBestSimScore

Write a function called findBestSimScore that takes a genome and a sequence and returns the highest similarity score found in the genome as a double.

Note: the term *genome* refers to the string that represents the complete set of genes in an organism, and *sequence* to refer to some substring or sub-sequence in the genome.

- Your function MUST be named **findBestSimScore**
- Your function should take two parameters in this order:
    - a string parameter for the genome (complete set of genes)
    - a string parameter for the sequence (sub-sequence of the genome)
- Your function should return the highest similarity score as a double.
- If the length of the sequence is longer than the genome, then the function should return 0. (the best similarity score is 0)
- Your function should not print anything.
- The best similarity scores is [0.0,1.0]

Note: Our sequence is "ACT", which is a string of length 3. That means we need to compare our sequence with all the 3 character long sub-sequences (substrings) in the genome. Follow the example below:

genome sub-sequence	sequence	similarity score
ATACGC	**ACT**	0.33
A**TAC**GC	**ACT**	0
AT**ACG**C	**ACT**	0.66
ATA**CGC**	**ACT**	0

← **findBestSimScore** returns 0.66, since that is the highest similarity score found

In Cloud9 the file should be called **findBestSimScore.cpp** and it will be one of the files you need to zip together for the **Project 1** on Moodle. Make sure that your main has all the test cases that needed to check the function. After developing in Cloud9, this function will be one of the functions needed for Problem 8.

Once you test your code on your Cloud 9 workspace, head over to Moodle to project 1 CodeRunner. For this problem, 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 6 (10 points): findMatchedGenome

Write a function called findMatchedGenome that takes three genomes and a sequence and prints the list of matched genomes.

- Your function MUST be named **findMatchedGenome**
- Your function should take four parameters in this order:
    - Genome 1, a string
    - Genome 2, a string
    - Genome 3, a string
    - sequence (sub-sequence of the genomes), a string
- Your function **should not return anything.**
- Unexpected values might be passed into the function and your function should be able to handle them without crashing the program:
    - Case 1: If one or more genomes or the sequence is an empty string, then your function should **only** print "`Genomes or sequence is empty.`"
    - Case 2: If the length of the three genomes are different, your function should **only** print "`Lengths of genomes are different.`"
    - Your function must check these cases in the order specified above; i.e. first check for empty genomes or sequences, then check for different length genomes.

We compute the highest similarity scores found in each genome. Among the three highest similarity scores, the best matched genome is the one with the highest similarity score. If two or more genomes have the same highest similarity scores, then it lists the genomes with the highest similarity score.

Example1: One of the genomes has the best match

genome 1	AATGTTTCAC
genome 2	GACCGACTAA
genome 3	AAGGTGCTCC
sequence	TACTA
Genome 2 is the best match.	

Explanation:
Since the length of the three genomes (genome1, genome2 and genome3) are the same, we calculate the best similarity score with each genome. The best-matched genome is then calculated based on the highest of the previously calculated best similarity scores.

		Highest similarity score
genome 1	AATGT**TTCAC**	0.4
genome 2	GACC**GACTA**A	0.8
genome 3	AAGG**TGCTC**C	0.6

The best similarity score for genome1 and sequence is 0.4, for genome2 and sequence is 0.8, and for genome3 and sequence is 0.6. Since genome 2 has the highest similarity scores among the three genomes, genome 2 is best matched.

Example2: Two genomes match with a sequence

genome 1	AACT
genome 2	AACT
genome 3	AATG
sequence	AACT
Genome 1 is the best match. Genome 2 is the best match.	

Explanation:
The best-matched genome is calculated based on the highest similarity scores given in each sequence.

		Highest Similarity Score
genome 1	**AACT**	1.0
genome 2	**AACT**	1.0
genome 3	AATG	0.5

The length of the three genomes are the same, so we calculate the best similarity score. The best similarity for genome1 and genome2 will be 1.0, while the genome3 is 0.5. Since genome 1 and 2 have higher similarity scores than the genom3, it will be printed in the format specified.

Example 3 (edge case): length of genomes is different

genome 1	AACTC
genome 2	AACT
genome 3	AATG

sequence	AACT
Lengths of genomes are different.	

Example 4 (edge case): One or more genomes or sequence is empty

genome 1	AACTC
genome 2	
genome 3	AATG
sequence	AACT
Genomes or sequence is empty.	

In Cloud9 the file should be called **findMatchedGenome.cpp** and it will be one of the files you need to zip together for the **Project 1** on Moodle. Make sure that your main has all the test cases that needed to check the function. After developing in Cloud9, this function will be one of the functions needed for Problem 8.

Once you test your code on your Cloud 9 workspace, head over to Moodle to the link project 1 CodeRunner. For this problem, 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 7 (15 points): Put it all together

To make it usable for doctors and researchers, we'll create a menu option and call the appropriate functions you have created.

The menu has the following options:

1. Calculate similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit

We've provided the showMenu function (proj1Menu.cpp) on Moodle. The menu will run on a loop, continually offering the user four options until they opt to quit.
You need to fill in the code for each of the options.
You should make use of the functions you wrote above, call them, and process the values they return.

Option1: Find similarity score
The program asks two sequences (sequence1 and sequence2), then it displays the similarity score between the two sequences.

```
Enter sequence 1:
AACT
Enter sequence 2:
AATC
Similarity score: 0.5
```

Option2: Find the best similarity score
The program asks for a genome and sequence (subset of the genome) and it displays the highest similarity scores found in the genome.

```
Enter genome:
AATCTCTTTAA
Enter sequence:
TCA
Best similarity score: 0.666667
```

Option3: Analyze the genome sequences
In this option, the program takes 3 genomes and a sequence and shows the best matched genome.

```
Enter genome 1:
AATGTTTCAC
Enter genome 2:
GACCGACTAA
Enter genome 3:
AAGGTGCTCC
Enter sequence:
TACTA
Genome 2 is the best match.
```

Option4: Quit
When the user opt this option, the program prints "`Good bye!`" And exits the program.

Invalid option
Your program should be able to handle the user error (i.e. users might choose the option that is not listed). If an invalid option is entered, the program should display "`Invalid option.`" and show the menu options again.

```
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
```

```
3. Analyze the genome sequences
4. Quit
10
Invalid option.
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit
4
Good bye!
```

Below is an example of running the main program.

```
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit
1
Enter sequence 1:
ATTCT
Enter sequence 2:
TAACT
Similarity score: 0.4
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit
2
Enter genome:
ATTCCCTAA
Enter sequence:
TAC
Best similarity score: 0.666667
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit
```

```
3
Enter genome 1:
ACCTT
Enter genome 2:
TCCTT
Enter genome 3:
TCCTT
Enter sequence:
CCT
Genome 1 is the best match.
Genome 2 is the best match.
Genome 3 is the best match.
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit
6
Invalid option.
Select a numerical option:
=== menu ===
1. Find similarity score
2. Find the best similarity score
3. Analyze the genome sequences
4. Quit
4
Good bye!
```

For the last problem, the file should be called **analyzeDNA.cpp** in your Cloud9. Your task is to complete the `main` function using the functions you have created.

Once you test your code on your Cloud 9 workspace, head over to Moodle to the link project 1 CodeRunner. For this problem, in the Answer Box, **paste the entire program**, including all the functions you use. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

7. Submitting files and Interview grading

Submitting the assignment:

1. Complete the **Project 1 CodeRunner**.
2. Zip the following 1 file into one zip file, name it: **project1_lastname.zip**
    - printCharInString.cpp
    - countMatches.cpp
    - zootopia1300.cpp
    - calcSimScore.cpp
    - findBestSimScore.cpp
    - findMatchedGenome.cpp
    - analyzeDNA.cpp
3. Submit the zip file to **Project 1**
4. Sign up for the interview grading slot on Moodle. Please make sure that you sign-up and complete an interview grading with your TA by October 12th. The schedulers for interview grading will be available after the deadline of this project.

    You are allowed to reschedule without any penalty only one time during the semester. If you miss a second time, you will be allowed to reschedule but you will incur a 25 points (out of 100) penalty, then 50 points for the third time.

8. Project 1 points summary

Criteria	**Pts**
CodeRunner	60
Interview grading (style, comments, algorithm)	40
Recitation attendance (week 5)*	-30
Total	100

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