CSCI 1300 CS1: Starting Computing
Ashraf, Cox, Spring 2020
Project 1: DNA, Deoxyribonucleic acid
Due: Saturday, February 22, by 6 pm
No bonus points for early submission

Objectives

- Understand and work with functions and strings
- Be able to test functions

You can find **project1 note: functions and strings** on Moodle.

Submissions

- **Honor code questions (mcq)**. There are a few multiple-choice questions about the honor code. Don't forget to complete it!
- **C++ files**. All files should be named as specified in each question, and they should compile and run on Cloud 9 to earn full points. TAs will be grading styles of your code and comments. Please see the style guide on Moodle and the summary note on Moodle. At the top of each file, write your name with the following format:

```
// CS1300 Spring 2020
// Author: Punith Sandhu
// Recitation: 123 - Favorite TA
// Project 1 - Problem # …

/*
 * This function converts a temperature in fahrenheit to celsius
 * and prints the equivalence.
 * Parameters: fahrenheit - degrees fahrenheit
 * Return: equivalent temperature in celsius
*/

double fahrenheit_to_celsius(double fahrenheit){
      double celsius = (fahrenheit - 32) * (5/9.0);
      return celsius;
}
```

  For each question, create a program that calls the function in the main. Here is an example.

- **Code runner**. Your program will be graded by the code runner. You can modify your code and re-submit (press Check again) as many times as you need to, up until the assignment due date.

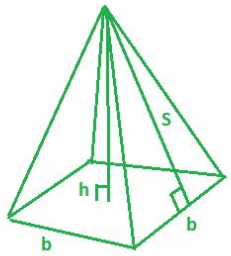
Interview grading

Sign up, between Feb. 19 and Feb. 22, for the interview grading slot on Moodle. The interviews will take place between Feb. 23 and March 9. If you don't sign-up between Feb.19 and Feb.22 and you miss your interview, then no points will be awarded for the project. The schedulers for interview grading will be available a couple days before the deadline of this project.

Start with these problems …

Question 1(5pt): pyramid, surface area

A Pyramid with a square base, 4 triangular faces, and an apex is a square pyramid. Write a function **`surfaceArea`** to calculate and return the surface area of a square pyramid. The function takes **two double parameters** - the base length(b) and slant height(s) of a square pyramid, and calculates and returns the surface area as a **double**.



Function specifications:

- The function name: **surfaceArea**
- The function parameters (in this order)
    - The base length, `double`
    - The slant height, `double`
- The function return value: surface area, `double`
- The function should not print anything.
- Edge cases: if one or more arguments are not positive numbers, then it should return -1

Sample run 1 (**bold** is user input)

```
Enter the base length: 3
Enter the slant height: 4
The surface area: 33
```

Sample run 2 (**bold** is user input)

```
Enter the base length: 3
Enter the slant height: -1
Invalid values
```

The file should be named as `pyramid.cpp`. Don't forget to head over to the code runner on Moodle and paste only your function in the answer box! In the main, make sure that you call the function and output the surface area.

Question 2(5pt): Braking Distance

Displacement, D (i.e. a distance) for a braking vehicle can be calculated with the initial velocity, **I**, the final velocity, **F**, and deceleration, **a**.

$$\frac{\left(F^2 - I^2\right)}{\left(-2 \cdot a\right)} = D$$

Write a function called **`displacement`** that takes as parameters: initial velocity, **`I`**, final velocity **`F`**, and deceleration **`a`.**

Function specifications:

- Your function must be named **displacement**
- Your function takes three parameters (all double, in this order)
    - Initial velocity **`I`**
    - Final velocity **`F`**
    - Deceleration **`a`**
- Your function should return the appropriate displacement, **`D`**, of the vehicle as `double`. If one or more values are invalid, the function print all the error messages in this order and return `0`.
- If your function receives negative velocities then it must print "`Velocity should be greater than zero.`"
- If your function receives a zero deceleration then it must print "`No brakes were applied.`"
- If your function receives a negative deceleration then it must print "`The vehicle is speeding up.`"
- If your function receives **`F > I`** then it must print "`Error in acceleration values.`"

For example: Bob is riding his bicycle to the store at a velocity of 4 m/s, when an aardvark runs out in front of him. He quickly brakes to a complete stop, with a deceleration of 2 m/s. What is his displacement?

Note that because Bob is stopped, the final velocity, `F = 0`. His initial velocity, `I = 4` m/s. The deceleration is `2` m/s.

Therefore the function should return 4.

$$\frac{\left(0^2 - 4^2\right)}{\left(-2 \cdot 2\right)} = 4$$

Sample run 1 (**bold** is user input)

```
Enter initial velocity: 10
Enter final velocity: 1
Enter deceleration: 5
Displacement is 9.9
```

Sample run 2 (**bold** is user input)

```
Enter initial velocity: 1
Enter final velocity: 2
Enter deceleration: 0
No brakes were applied.
Error in acceleration values.
```

The file should be named as `displacement.cpp`. Don't forget to head over to the code runner on Moodle and paste only your function in the answer box! In the main, make sure that you call the function and output the displacement.

Question 3(5pt): change to upper case

Write a function called **`toUpper`** that has a single parameter and returns a new string with all characters converted to uppercase.

Function specifications:

- The function name: **toUpper**
- The function parameter
    - The string that will be converted to upper cases, `string`
- The function return value: the string with all upper cases, `string`
- The function should not print anything.
- Edge cases: Non alphabetical characters remain unchanged.

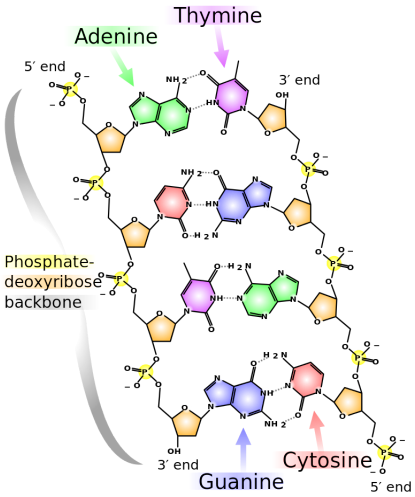
Sample run 1 (**bold** is user input)

```
Enter the string: Convert this string to UPPER case!!!
CONVERT THIS STRING TO UPPER CASE!!!
```

The file should be named as `toUpper.cpp`. Don't forget to head over to the code runner on Moodle and paste only your function in the answer box! In the main, make sure that you call the function and output the surface area.

DNA Analyzer

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.

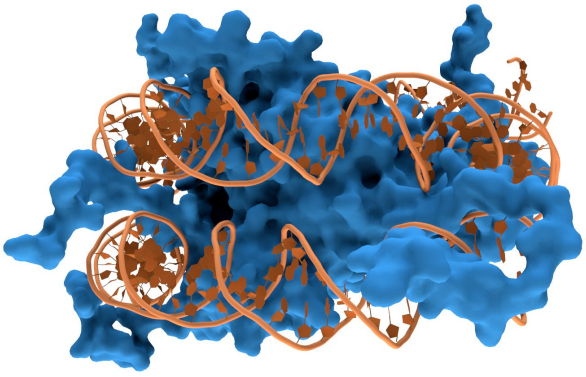
We refer to an organism's complete set of chemical bases as their *genome*. Every genome looks something like this:

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.

For this project (questions 4 - 7) you will be writing a program that performs three main functionalities:

1) Calculate the similarity score (This will be question 4)
2) Find the best similarity score in the given genome (This will be question 5)
3) Given three genomes and a sequence from the unknown bacteria, determine the most probable match (This will be question 6)

So, in question 7, you will be developing the `main` function using the functions you create in problems 4-6. 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 function on your Cloud 9. Then, once you complete, you can submit it on Moodle coderunner to make sure it's fully functional. All the functions and the `main` function should be in one file, `DNAanalyzer.cpp`.

Question 4(10pt): 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

Function specifications:

- The function name: **`calcSimScore`**
- The function parametesr(in this order):
  - Sequence 1, `string`
  - Sequence 2, `string`
- The function should return the similarity score as a `double`
- If the length of the strings is the same, then the function calculates the similarity score. If not, the similarity score would be 0.
- The function should not print anything.

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

Once you have tested your code on Cloud 9 / VS code, then head over to code runner on Moodle and only your function in the answer box! Make sure that your unit tests are in the main function (commented out).

Question 5(10pt): 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.

Function specifications:

- The function name: **`findBestSimScore`**
- The function 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)
- The 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)
- The 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

Once you have tested your code on Cloud 9 / VS code, then head over to code runner on Moodle and only your functions in the answer box! Make sure that your unit tests are in the main function (commented out).

Question 6(10pt): findMatchedGenome
Write a function called findMatchedGenome that takes three genomes and a sequence and prints the list of matched genomes.

Function specifications:

- The function name: **`findMatchedGenome`**
- The function parameters(in this order):
    - Genome 1, a string
    - Genome 2, a string
    - Genome 3, a string
    - sequence (sub-sequence of the genomes), a string
- The 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 genome3, 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.	

Once you have tested your code on Cloud 9 / VS code, then head over to code runner on Moodle and paste only your functions in the answer box! Make sure that your unit tests are in the main function (commented out).

Question 7(5pt): 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

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. Be sure to use the functions you wrote in questions 4-6.

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.

Sample run (**bold** is user input)

```
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!
```

Once you have tested your code on Cloud 9 / VS code, then head over to code runner on Moodle and paste the entire program (functions from questions 4 - 6 and the `main` function from this question 7) in to the answer box!

Project 1 checklist

Here is a checklist for submitting the assignment:

1. Complete the **Honor code questions**
2. Complete the code **project 1 CodeRunner**
3. Submit one zip file to **project 1 zip file submission**. The zip file should be named, **project1_lastname.zip**. It should have the following 4 files:
    - **pyramid.cpp**
    - **displacement.cpp**
    - **toUpper.cpp**
    - **DNAanalyzer.cpp**
4. Sign up, between Feb. 19 and Feb. 22, for the interview grading slot on Moodle. The interviews will take place between Feb. 23 and March 9. If you don't sign-up between Feb.19 and Feb.22 and you miss your interview, then no points will be awarded for the project. The schedulers for interview grading will be available a couple days before the deadline of this project.

During the interview grading, your TAs will be checking styles and comments of your code, test cases. They will also ask about your implementations and conceptual questions.

Project 1 summary

Criteria	**Pts**
Honor code MCQ questions	10
CodeRunner (problem 1 - 7)	50
Interview grading	40
Recitation attendance (Feb 17 - Feb 18)*	-30
Total	100

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