CSCI 1300/1310 Introduction to Computer Programming
Instructors: Knox/Hoenigman
Assignment 4
Due Friday, Sept 23, by 12:30 pm

For this assignment, the solution for all problems should be in a file called Assignment4_LastName.cpp, where LastName is your last name. Please include comments in your code that explain what your code is doing. The comments should also include your name, recitation TA, and the assignment number.

Measuring DNA Similarity

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

One of the challenges in computational biology is determining what the codons in a DNA sequence represent. A codon is a sequence of three nucleotides that form a unit of genetic code in a DNA or RNA molecule. For example, given the sequence GGGA, the codon could be a GGG or a GGA, depending on where the gene begins and the active bases in the gene. Clues about how to interpret a DNA sequence can be found by comparing an unknown DNA sequence to a known sequence and measuring their similarity. If sequences are similar, then it can be hypothesized that they have similar functions and proteins.

Assignment Details:

In this assignment, you will develop a few functions for DNA analysis. These functions will calculate common measures of DNA similarity, such as the Hamming distance and the Best Match Hamming distance between two DNA sequences. Each of the DNA sequences you need for this assignment can be copied from this write-up and stored in a variable in your program. There is a sample DNA sequence for a mouse, human, and an unknown species. Your mission is to determine the identity of the unknown by comparing it to the human and the mouse. If the unknown species is more similar to the human than it is to the mouse, then you can conclude that the unknown sequence is from a human. Otherwise, you can conclude that the unknown is from a mouse.

Your assignment needs to include at least the following functions for full credit:

```
void calculateSimilarity(int *similarity, string DNA1, string DNA2);

void calculateBestMatch(int *distance, int *index, string DNA1, string
DNA2);
```

Hamming distance and similarity between two strings

Hamming distance is one of the most common ways to measure the similarity between two strings of the same length. Hamming distance is a position-by-position comparison that counts the number of positions in which the corresponding characters in the string are different. Two strings with a small Hamming distance are more similar than two strings with a larger Hamming distance.

Example:
first string = "ACCT"
second string = "ACCG"

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

In this example, there are three matching characters and one mismatch, so the Hamming distance is one.

The similarity score for two sequences is then calculated as follows:

similarity_score = (string length-hamming distance) / string length

similarity_score = (4-1)/4=3/4=0.75

Two sequences with a high similarity score are more similar than two sequences with a lower similarity score.

The Best Match algorithm extends the Hamming distance calculation by finding the best overlap of the two strings using the Hamming distance calculation. For any two strings, calculate the Hamming distance between the string and substring starting at each position of the string.

calculateSimilarity(double*, string, string)

The *calculateSimilarity()* function should take two arguments that are both strings and an double pointer that stores the similarity between the strings. You can declare a double pointer just as you would an integer pointer:

```
double x;
double *dPtr = &x;
```

The function should calculate the similarity score for the two strings and update the similarity int with that score.

Note: when you test *calculateSimilarity()*, pass in strings where you can calculate the similarity by hand before passing it real data. That will help you identify errors in your algorithm.

calculateBestMatch(int*, int*, string, string)
The *calculateBestMatch()* function should take four arguments - two integer pointers and two strings. The integer pointers store the Hamming distance calculation and the index in the string where the best match starts. The two string arguments are the two strings to compare. The second string argument is the substring to search for. The first string is the string you are searching.

Note: you will need to be aware of the end of each string to make sure that you don't loop off the end of either string.

Functionality in main()
In your *main()* function, you will need to call the other functions you have written passing in the DNA samples shown below in this write-up. You should output the result of the function calls in the *main()* function. After calling *calculateSimularity()*, you need to output the identity of the unknown DNA sequence.

```
If the unknownDNA is more similar to the humanDNA
     print "Human"
Else If the unknownDNA is more similar to the mouseDNA
     print "Mouse"
Else unknownDNA is equally similar to both mouse and human
     print "Identity cannot be determined."
```

Before calling *calculateBestMatch(),* you need to prompt the user for a search string. For example, if you want to compare the search string to the mouse DNA, you would do something like the following:

```
cout<<"Enter a substring:;
getline(cin, subStr);
calculateBestMatch(hamDist, index, mouse, subStr);
```

After calling *calculateBestMatch()*, you need to display the DNA sequence that is the best match as well as the best match score. If there isn't a match longer than one character, print "`Match not found.`"

Other helpful hints
Write the code without pointers first. Once you are confident that your algorithms are correct, modify your functions to use pointers.

DNA samples
Use the following human, mouse, and unknown DNA strings in your program.

```
humanDNA =
"CGCAAATTTGCCGGATTTCCTTTGCTGTTCCTGCATGTAGTTTAAACGAGATTGCCAG
CACCGGGTATCATTCACCATTTTTCTTTTCGTTAACTTGCCGTCAGCCTTTTCTTTGAC
```

```
CTCTTCTTTCTGTTCATGTGTATTTGCTGTCTCTTAGCCCAGACTTCCCGTGTCCTTTC
CACCGGGCCTTTGAGAGGTCACAGGGTCTTGATGCTGTGGTCTTCATCTGCAGGTGTCT
GACTTCCAGCAACTGCTGGCCTGTGCCAGGGTGCAGCTGAGCACTGGAGTGGAGTTTTC
CTGTGGAGAGGAGCCATGCCTAGAGTGGGATGGGCCATTGTTCATG"

mouseDNA =
"CGCAATTTTTACTTAATTCTTTTTCTTTTAATTCATATATTTTTAATATGTTTACTAT
TAATGGTTATCATTCACCATTTAACTATTTGTTATTTTGACGTCATTTTTTTCTATTTC
CTCTTTTTTCAATTCATGTTTATTTTCTGTATTTTTGTTAAGTTTTCACAAGTCTAATA
TAATTGTCCTTTGAGAGGTTATTTGGTCTATATTTTTTTTTCTTCATCTGTATTTTTAT
GATTTCATTTAATTGATTTTCATTGACAGGGTTCTGCTGTGTTCTGGATTGTATTTTTC
TTGTGGAGAGGAACTATTTCTTGAGTGGGATGTACCTTTGTTCTTG"

unknownDNA =
"CGCATTTTTGCCGGTTTTCCTTTGCTGTTTATTCATTTATTTTAAACGATATTTATAT
CATCGGGTTTCATTCACTATTTTTCTTTTCGATAAATTTTTGTCAGCATTTTCTTTTAC
CTCTTCTTTCTGTTTATGTTAATTTTCTGTTTCTTAACCCAGTCTTCTCGATTCTTATC
TACCGGACCTATTATAGGTCACAGGGTCTTGATGCTTTGGTTTTCATCTGCAAGAGTCT
GACTTCCTGCTAATGCTGTTCTGTGTCAGGGTGCATCTGAGCACTGATGTGGAGTTTTC
TTGTGGATATGAGCCATTCATAGTGTGGGATGTGCCATAGTTCATG"
```

Additional practice problems:

If you are interested in additional problems for practice, here are a few more problems you can work on. We won't be grading them, but a little extra practice never hurt anyone.

Complement sequence function

DNA is double stranded even though we always write a single strand's nucleotide sequence. This is because we can infer the other stands sequence from the given strand. Every A on one strand has a complementary T on the opposite strand (every C has a G). Therefore we can create the complement strand by swapping the nucleotides with the complementary nucleotide. *Create a function to take the sequence and produce the complement sequence.*

Reverse complement sequence function

You now have the complement strand, but the DNA is read in the forward direction for the given strand and in the reverse direction for the complementary strand. Therefore, we must reverse the strand (first character becomes the last character, second becomes second to last, and so on.) to print it correctly. *Create a function to reverse the complement of a given DNA sequence.* Now we can search both strands of the given DNA sequences by using our previously created functions.