CSCI 135 Section 03 – Spring 2014

Programming Assignment #3

Program Description

Phylogenetics is the study of evolutionary relationships among various organisms. Over time, the DNA sequence of a particular organism undergoes various mutational transformations and at certain points organisms diverge enough to form new species. DNA sequence analysis, using computer programs, is an invaluable tool for phylogenetic analysis of different organisms to determine how closely related they are.

In this project, you will develop an object-oriented DNA sequence analysis tool. You will be provided two input files; one containing a number of DNA sequences, while the second contains potential mutated sequences. Your goal will be to identify whether each source sequence (from the first file) can be mutated to match a sequence from the second file and write a report of your findings to an output file.

Your program will consider 5 possible types of DNA mutations:

(1) Rotate left mutation
(2) Rotate right mutation
(3) Phenylalanine mutation
(4) Glycine deletion mutation
(5) Isoleucine insertion mutation

These mutations will be described in more detail below.

Your program will check each source sequence, from the first file, to see if a series of mutations exists (between one and eight) that, when applied to the source sequence, make it identical to any of the mutated sequences from the second file. For example, you will compare the original source sequence after a single mutation; since there are five possible mutations (listed above) you will have five resulting sequences to compare. If none matched, you would then compare the original source sequence after two mutations were performed; with five possible choices for the first mutation and five possible for the second, you will have twenty-five resulting sequences to compare. You'll repeat this process for all combinations of three mutations applied, four mutations applied, and so on, until you have either found a matching mutated sequence from the second file or you have attempted all possible combinations of eight mutations without success. If a match is found, you must also provide the order and names of the mutations that were applied in your output. You are not guaranteed a match, but if you find one, you do not need to check other target mutations for matches of that source sequence (so, stop on the first match).

The mutations are now described in greater detail. Note that for (3), (4), and (5), the match may start anywhere in the original sequence and need not be preceded by a full codon.

1. Rotate Left Mutation: The original DNA sequence is rotated left by 1 position: this effectively shifts all the letters left by 1 position and re-inserts the first left shifted letter to the end of the sequence and makes it the last letter. An example is shown below:

   `Original Sequence:  GATTACA`

   `Result of Mutation: ATTACAG`

2. Rotate Right Mutation: The original DNA sequence is rotated right by 1 position: this effectively shifts all the letters right by 1 position and re-inserts the last right shifted letter to the beginning of the sequence and makes it the first letter. An example is shown below:

   `Original Sequence:  GATTACA`

   `Result of Mutation: AGATTAC`

3. Phenylalanine (Phe) Mutation: The amino acid phenylalanine can be represented with both TTT and TTC. This mutation changes the first and last occurrences of TTT to TTC and the first and last occurrences of TTC to TTT in the original string. The changes are made all at once, regardless of codon boundaries. A few examples are shown below:

   Example A:

   `Original Sequence:  GATTTATTCA`

   `Result of Mutation: GATTCATTTA`

   Note that the TTT was not preceded by a full codon; the match may start anywhere in the original sequence

   Example B:

   `Original Sequence:  GATTCGTTCGATTCA`

   `Result of Mutation: GATTTGTTCGATTTA`

4. Glycine (Gly) Deletion Mutation: The amino acid glycine can be represented with GGT, GGC, GGA, and GGG. This mutation removes the last letter of the first and last occurrences of any of these sequences. A few examples are shown below:

   Example A:

   `Original Sequence:  GAGGCAAATA`

   `Result of Mutation: GAGGAAATA`

   Example B:

`Original Sequence:  GAGGGTTTCGGAA`

`Result of Mutation: GAGGTTTCGGA`

5. Isoleucine (Ile) Insertion Mutation: The amino acid isoleucine can be represented with ATT, ATC and ATA. This mutation inserts an extra T after the first and last occurrences of any of these sequences. A few examples are shown below:

   Example A:

   `Original Sequence:  GAATTAAATA`

   `Result of Mutation: GAATTTAAATAT`

   Example B:

   `Original Sequence:  GAATGAAATA`

   `Result of Mutation: GAATGAAATAT`

User Interface

There is no user interface for this program. All input comes from the input files. All output should be saved in the output file.

Input

The program expects the names of the two input files on the command line. If either file does not exist or cannot be opened for any reason, the program should display an error message and exit. The input files will have the format described below; your code does not need to check for incorrectly formatted data.

Un-mutated Sequences file

The un-mutated sequences file contains a list all source DNA sequences you'll be checking, in the following format. Each line will denote a single sequence:

```
SEQUENCE_STRING
SEQUENCE_STRING
```

- There may be any number of blank lines in the file
- A sequence will only contain the letters A, C, G and T
- A sequence will be 60 characters long, at most
- You may assume that each provided sequence is unique
- The file will contain a maximum of 20 sequences

Mutated Sequences file

The mutated sequences file contains the list of possible mutated DNA sequences, which you'll be checking. There may be more mutated sequences than un-mutated. Each line in the file will denote a single mutated sequence.

SEQUENCE_STRING
SEQUENCE_STRING

- There may be any number of blank lines in the file
- A sequence will only contain the letters A, C, G and T
- A sequence will be 68 characters long, at most
- You may assume that each provided sequence is unique
- The file will contain a maximum of 30 sequences

Output

The program expects the name of the output file on the command line (in addition to the names for the two input files). Your program will write your results to this file, using the following format:

```
Un-mutated sequence: SEQUENCE_STRING
will resemble mutated sequence: SEQUENCE STRING
after the following mutations are performed:
- mutation1
- mutation2
- mutation3…
BLANKLINE
```

- The list of mutations performed will correspond to the **ordered** list of mutations that were performed. Each mutation should be output with its full descriptive name (e.g. Rotate Right Mutation).
- After the result for a single sequence, you will output a blank line.
- If no match could be found for an un-mutated sequence, then you will adjust the output to display:

```
Un-mutated sequence #N: SEQUENCE_STRING
could not be mutated to find a match
BLANKLINE
```

Programming Rules

You must document your program (preamble, function pre- and post-conditions, comments throughout the code, class documentation, appropriately choosing variable and function names).

You must implement and use a class called **Sequence** to represent a **single DNA sequence**. Each DNA sequence from the two input files should be stored in an instance of your Sequence class. The class should implement 5 public member functions to perform the 5 mutations described above. You may add other data and function members as desired.

You must implement and use an algorithm to correctly mutate each source sequence through all possible combination of mutations and compare it with the mutated target sequences for a match. This may be done by writing a second class, by writing additional non-class functions, within the Sequence class, or any combination of the three.

Your class should respect the OOP principles of information hiding, data abstraction and encapsulation.

The program must accept the names of the input files and output file as arguments from the command line (you cannot hard-code the names of the files in your final submission).

You are expected to handle missing input files gracefully, as described in the `Input` section intro above. Otherwise, you may assume the input files will be correctly formatted.

You must use multiple files in your implementation (separate compilation). Your class definitions should be placed in a separate header file(s). The implementations of your class functions should be placed in a separate implementation file(s).

Grading

Does the program compile (on one of the cslab machines): 10 pnts
Correctness and implementation of the Sequence class: 30 pnts
Correctness and implementation of the mutate-and-compare routine: 20 pnts
Correctness and implementation of the file I/O operations: 10 pnts
Correctness and implementation of the main function and any other functions in the program: 20 pnts
Correctness and use of multiple file implementation: 10 pnts
Documentation: 10 pnts.

What to submit and when

This assignment is due by the end of the day (i.e. 11:59PM, EST) on May 14, 2014. You must create a zip archive of your source code files (.cpp and .h files) and submit the zip archive to Blackboard for this assignment. Do not include any input, output or executable files. Attach the zip file to your submission in Blackboard - do not paste the code into your submission. You should test your program on the cslab machines to make sure it compiles and runs correctly there.