Assignment 1: Computing with DNA sequences

Summary

Deoxyribonucleic acid (DNA) is a nucleic acid containing the genetic instructions used in the development and functioning of all known living organisms. In this assignment you will be extracting information from strands of DNA, which you can simply think of as sequences of the letters `a`, `c`, `g`, and `t` in any order (assume that the letters are always in lower case). Although you do not need to know anything else about DNA other than what is written here, it would be useful to read my tutorial containing biological background information (found here: http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci135/biobackground.pdf) to get a better perspective on the project.

You will write a program that performs several computations given strands of DNA read from a file. The results of computations should be written to the standard output stream and any error messages should be sent to the standard error stream.

Input

The input DNA strings will come from a file that the program is expected to open and read. The program should expect that the file is named "`dna.txt`" and is in its current working directory. However, if the program cannot open this file for any reason, it should report this error on the standard error stream and then exit gracefully. This might happen for a number of reasons, including that the name is wrong, the file is not there, or the permissions on the file are wrong.

The file will contain two DNA strings, on separate lines. There will be no blank lines in the file. The program should not assume that the strings are valid DNA strings. If for some reason, any of the strings have characters other than `a`, `c`, `g`, or `t` it should report the error on the standard error stream and gracefully exit.

The DNA strings in the file will not be larger than 4095 characters long. This is an arbitrary length chosen just for the assignment.

Computational Tasks

The program must be able to perform the following computational tasks. Each of these must be implementing by a user-defined function.

GC Content: The *GC content* of a single DNA strand is the ratio of the total number of `c`'s and `g`'s to the length of the strand. For example, the sequence '`atcgtttgga`' is of length 10 and has a total of 4 `c`'s and `g`'s, so its GC content is 0.4.

Given a DNA string, count the GC content of the string and report this as the fraction (number of `c`'s and `g`'s)/(length of string). The result of this computation is therefore a number between 0 and 1 inclusive.

Poly-T sequences: A *poly-T sequence* of length at least N is a sequence of N or more consecutive `t` nucleotides. It is maximal if it is not contained in a longer poly-T sequence.

Given a DNA string, count the number of occurrences of maximal poly-T sequences of length at least N in the string and report this as a whole number.

For example, `actttaattttactttcctta` has 3 poly-t sequences of length 3 or more:

```
actttaattttactttcctta
```

and only one of length 4 or more:

```
actttaattttactttcctta.
```

Note: The sequence `tttt` has only one maximal poly-T sequence of length at least 3. Similarly, the sequence `ttttttt` has only one maximal poly-T sequence of length at least 3. This is because we are looking for sequences of 3 or more consecutive `t`'s.

Mutations: A *mutation* is a change in one or more bases in the DNA sequence. Mutations are essential to evolution.

Given two DNA strands, not necessarily of the same length, count and report the positions at which they differ. If one is longer than the other, then treat each extra positions in the longer as a mutation. For example, the two strings '`aaaataa`' and '`aaaaaaaccc`' differ in just one common position, the fifth letter, but the second has three extra letters, so the result should be 4 (1 plus 3).

Phylogenetic distance: The *phylogenetic distance* between two DNA strands is the number of mutations that occurred in the transition from one strand to the other.

Given two DNA strands of equal size, compute and output the phylogenetic distance between them, measured as the number of mutations between them divided by the length of the strings, producing a value between 0 and 1. For example the phylogenetic distance between '`aaaataa`ggg' and '`aaaaaaaccc`' is $4/10 = 0.4$.

User Interface

This program provides a menu driven user interface. It should display the following menu on the screen:

1 compute GC content of DNA string 1
2 compute GC content of DNA string 2
3 compute number of poly-T sequences of length N or more in DNA string 1
4 compute number of poly-T sequences of length N or more in DNA string 2
5 compute number of mutations between the two DNA strings
6 compute phylogenetic distance between the two DNA strings
7 quit

After displaying the menu the program should display a prompt and wait for the user to enter one of the above numbers. When the user enters a valid choice (a digit 1, 2, 3, 4, 5, 6, or 7) the task should be performed and then menu should be redisplayed unless the user chooses to quit.

When the user enters either 3 or 4 to compute the number of poly-T sequences of length N or more, he/she should be prompted for the value of N. N can be any integer greater than zero. Your program should verify that the number entered is greater than zero. If the user types anything other than one of the numbers 1 through 6, the program should display the message "`Invalid choice. Try again.`"

Programming Rules

Your program must conform to the programming rules described in the Programming Rules document on the course website. Remember in particular that:

- You must write your program yourself. I will on occasion ask you to explain the code to me, so you need to make sure that you understand the code that you write. If you want to use features of C++ that we did not cover in class, you need to understand them thoroughly.
- You must document your program (preamble, function pre- and post-conditions, comments throughout the code, appropriately choosing variable and function names).
- Input from the standard input stream and output to the standard output and error streams should be performed by the `main` function only, i.e. no other functions should perform any I/O operations).
- Late programs lose 20% for each 24 hours they are late.

Also, you must implement and use at least four functions that perform the four computational tasks described above. Each of these functions has to take at least one parameter, i.e. the DNA string. It may take other parameters if you want. It is up to you whether the main program or your functions should validate that the string is a valid DNA string. (Remember that a valid DNA string contains only the letters `a`, `c`, `g`, and `t` in any order and in lower case.)

Grading

The program will be graded based on the following rubric.

- Does the program compile (on a cslab machine): 15%
- Correctness and implementation of the four functions: 45%
  - compute GC content (10%)
  - compute number of poly-T sequences of length N or more (15%)
  - compute mutation between two strings (10%)
  - compute phylogenetic distance (10%)
- Correctness and implementation of the main function and any other functions in the program: 20%
- Documentation: 15%
- Style and proper naming: 5%

Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on October 4, 2012. You need to submit a single file for this assignment. Its name should be *username*`_hwk1.cpp.` where *username* is to be replaced by your username on our system. Do not name it anything else. If it is named anything else, it loses 3% of the program's value. Before you submit the assignment, make sure that it compiles and runs correctly on one of the cslab machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

You are to put your file into the directory

```
/data/biocs/b/student.accounts/cs135_sw/cs135projects/project1
```

Give it permission 600 so that only you have access to it. To do this, cd to the above directory and run the command

```
chmod 600 yourfilename
```

where *yourfilename* is the name of the file you put in that directory.

If you put a program there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date.