CSCI 135 Section 2
Programming Project 3

Due: Friday, December 12th, 2014, 11:59:00 pm

1 Introduction

In this project you have to combine your skills to create an Biological Analysis program. Some tools that you should use are file I/O, arrays and classes. Note that this project should be completed in an Object Oriented way, and I suggest you first think about what sort of classes you will have and then run them by me before starting to code them up.

1.1 Programming Rules

Under "Course Information" on Blackboard (for CSCI 135), you will see a document called "Programming Rules". Please read it and make sure you follow the rules described therein.

1.2 Commenting

Note that you will lose points if you do not comment your code. Apart from explaining what a block of code is doing, you should also write a brief description of your program at the top of your `.cpp` files using multiline comments `(/* ... */)`. Meaningful comments will also assist the instructor in understanding your code easily.

1.3 Design

It's a bad idea to jump right away to coding. Always think about what the problem is asking for, then start formulating your solution on paper, and then keep on thinking about the problem. You will have to decide whether you should write code to test your formula, or use a calculator - there is no general rule, and it varies from problem to problem.

2 Biological Sequence Analysis

In the past few decades, computers have helped unravel mysteries in biology. Problems which were too difficult to work out by hand can now be computed in a very short period of time, and we can develop models to predict how certain structures will form from basic components. Computer programs also help in determining properties of sequences (sequence analysis). In this project, you will develop software that can be used to perform sequence analysis on DNAs and RNAs. Note that the user interaction in this program is different: your program will not interact directly with the user! Instead, you will use command line arguments to direct the program's behavior. See "Program flow" for more details.

Before you proceed, make sure you have read the document named "Biology Background" in the project folder.

2.1 Program Flow

The program should start and end without asking the user for any input or showing any output on the console, except how many sequences were processed or an error (if there is one). Your program has two options: 1) DNA Sequence Analysis, and 2) Protein Sequence Prediction. The user will provide this option, "dna" or "rna", as the first parameter to your program. The second argument is the input file, which contains the sequences. The third parameter will be output file name. Therefore, you should expect your program to be run like this:

```
$ ./hw3 dna MyDnaFile.txt output
```

or

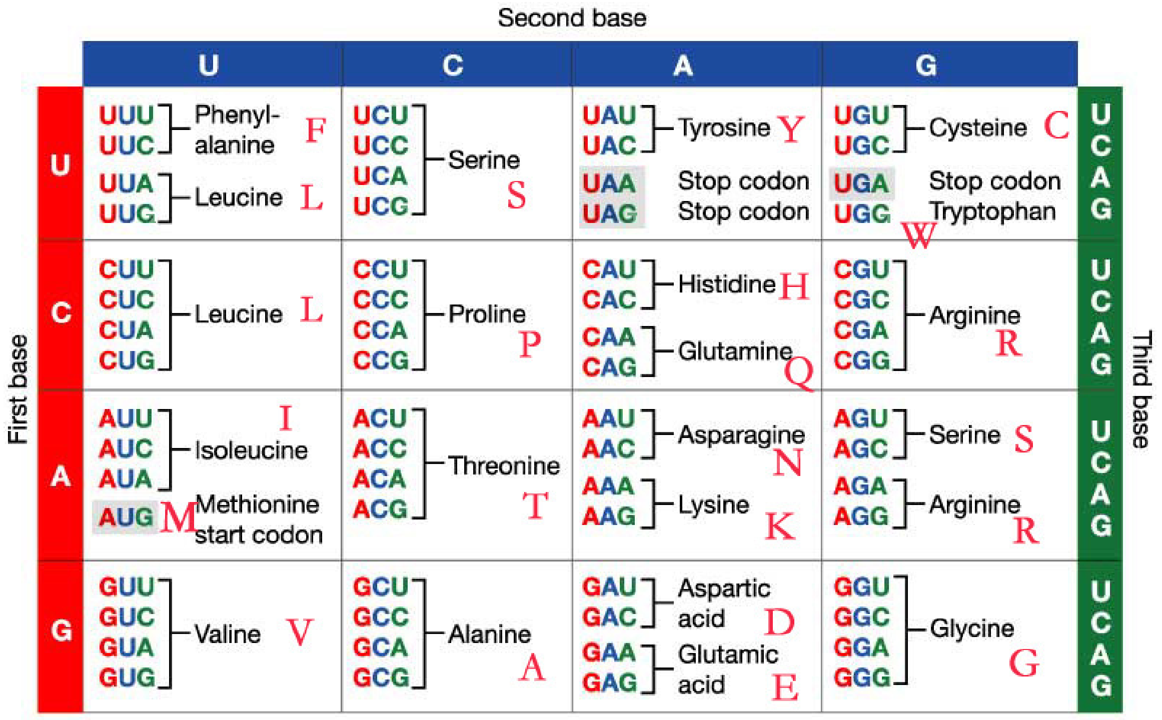
```
$ ./hw3 rna MyRnasFile.txt rnaoutput.txt
```

2.2 DNA Sequence Analysis

In the first part, you have to implement some methods to analyze DNA sequences. You will ask the user to provide a file (filename to be provided via the command line) with a single strand ($5'$ to $3'$) of one or more DNA sequences (one on each line), then perform the following tasks on each sequence, and then write the output in a single file, whose name is also provided as a command line argument.

For each strand, your program should:

1. Find the other strand ($3'$ to $5'$ direction), but then print it in the $5'$ to $3'$ direction. For example, if you are given `GTATCCAATGCC` in the $5'$ to $3'$ direction, it's complement in $3'$ to $5'$ will be `CATAGGTTACGG`. However, you need to print this in $5'$ to $3'$, so the output should be `GGCATTGGATAC`.
2. Find the locations of all poly-T sequences of a length at least 4. Note that a poly-T sequence of length N is a substring of N *or more* T's.
3. Find the GC content.
4. Find the locations of all CpG islands.



First base	Second base U	Second base C	Second base A	Second base G	Third base
U	UUU Phenyl-alanine F	UCU Serine S	UAU Tyrosine Y	UGU Cysteine C	U
U	UUC Phenyl-alanine F	UCC Serine S	UAC Tyrosine Y	UGC Cysteine C	C
U	UUA Leucine L	UCA Serine S	UAA Stop codon	UGA Stop codon	A
U	UUG Leucine L	UCG Serine S	UAG Stop codon	UGG Tryptophan W	G
C	CUU Leucine L	CCU Proline P	CAU Histidine H	CGU Arginine R	U
C	CUC Leucine L	CCC Proline P	CAC Histidine H	CGC Arginine R	C
C	CUA Leucine L	CCA Proline P	CAA Glutamine Q	CGA Arginine R	A
C	CUG Leucine L	CCG Proline P	CAG Glutamine Q	CGG Arginine R	G
A	AUU Isoleucine I	ACU Threonine T	AAU Asparagine N	AGU Serine S	U
A	AUC Isoleucine I	ACC Threonine T	AAC Asparagine N	AGC Serine S	C
A	AUA Isoleucine I	ACA Threonine T	AAA Lysine K	AGA Arginine R	A
A	AUG M Methionine start codon	ACG Threonine T	AAG Lysine K	AGG Arginine R	G
G	GUU Valine V	GCU Alanine A	GAU Aspartic acid D	GGU Glycine G	U
G	GUC Valine V	GCC Alanine A	GAC Aspartic acid D	GGC Glycine G	C
G	GUA Valine V	GCA Alanine A	GAA Glutamic acid E	GGA Glycine G	A
G	GUG Valine V	GCG Alanine A	GAG Glutamic acid E	GGG Glycine G	G

Figure 1: RNA codon to amino acid mapping

2.3 From RNAs to Proteins

As you already know by now, proteins are created via RNAs through a process called *translation*. The RNA strands contain codons (groups of 3 bases), each of which encode a certain amino acid. This mapping is described in detail in Figure 1.

Your program will take a file of RNAs as input, and output the corresponding sequences of amino acids in another file. Both filenames will be provided via the command line. You can expect the second argument to be the input file's name, and the third argument to be the output file's name.

Suppose one of the sequences is:

```
AUGCUUAUUAACUGAAAACAUAUGGGUAGUCGAUGA
```

The set of codons for this sequence is `AUG CUU AUU AAC UGA AAA CAU AUG GGU AGU CGA UGA`.

The first codon `AUG` is the start codon. This means that we should start reading the sequence. Also note that this codes for Methionine (M). The next 3 code for Leucine (L), Isoleucine (I) and Asparagine (N). The 5th codon is the stop codon, so we pause the mapping but continue reading. We skip over 6 and 7, but since the 8th codon is the start codon (AUG) we start mapping again. AUG, GGU and AGU code for M, G and S respectively. The last codon is a stop codon, so that is not included. Therefore, the output sequence of amino acids is M-L-I-N-M-G-S.

Make sure you check that the RNA sequence provided by the user is a valid protein coding RNA. For this, you need to ensure that it's length is a multiple of three, that it starts and ends with appropriate codons, and that the characters of the sequence belong to the set $\{A, C, G, U\}$.

3 Reading Command Line Arguments

Command line arguments are parameters that you type after the name of your program before hitting ENTER. For example, in the command `$ cp file1 file2` to copy a file, 'file1' and 'file2' are command line arguments.

Command line arguments are actually parameters of the `main()` function - yes, C++ has two versions of `main()`. The overloaded definition of main can be used to pass information into a program when you run it. The second version of main is `main(int argc, char *argv[])`. You should have already guessed that `argc` and `argv` are arbitrary and are used as convention, because the names of the parameters are irrelevant to the declaration.

`argc` is an integer that holds the number of arguments on the command line. It will always be at least 1, because the name of the program qualifies as the first argument. `argv` is a pointer to an array of character pointers Each pointer in `argv` array points to a string containing a command line argument.

In other words, `argv[1]`, `argv[2]`, . . . , `argv[argc-1]` are all command line arguments. `argv[0]` contains the name of the program.

4 Approach

You should use classes to represent RNAs and DNAs. Since there are different functions for RNA and DNA analysis, you should have one class for DNAs and one for RNAs. Note that each DNA/RNA object should correspond to **one** DNA/RNA string. These classes should implement functions that analyze the given string. Consider the following code:

```
RNA fooRNA("AAUUCGAUGCUGCAU");
if(fooRNA.isValidRNA())
    cout << fooRNA.convertToAminoAcids() << endl;
else
    cout << "This is not a valid RNA string" << endl;
```

As you can see, there is a class RNA and we create an object `fooRNA`. Then we call various functions on this object to test its validity and convert it to a protein. There should be a similar class for DNAs. For example, you should have something like this for the sample DNA string from above:

```
DNA aDNAString("GTATCCAATGCC");
cout << aDNAString.otherStrand() << endl; //Prints GGCATTGGATAC
```

Note that the `cout` is here just to exemplify this function's usage. Your actual program will write everything to a file.

5 Submission

Please archive all your source files in to a zip file, rar file or tar ball before submitting. Name this archive file in the following format: `<LAST_NAME>_hw3.<EXTENSION>`

Submit it in the folder for Project 3.