Cairo University
Faculty of Computers and Information
CS213: Programming-II - 2018-2019 – First Semester
Dr. Mohammad El-Ramly, Dr. Mohammad Nassef

Assignment 4

(900 Points scaled to 9 Marks + Bonus 1.5 Marks)

Due Date of Phase-1: Saturday 1 December 11:50 PM
Due Date of Phase-2: Saturday 8 December 11:50 PM

After graduation, you might be employed in one of the software companies or institutions that are responsible for developing software for people in other domains. In this assignment, you are required to develop software that can be used by Biologists to manipulate their biological data. The main job and interest of Biologists is to efficiently analyze the biological data of living creatures (including humans, animals and plants). The analysis of biological data can for example result in great scientific discoveries for diseases. It can also give a clue about the physical similarities and differences between different persons according to their genetic attributes. They can do that in laboratories, but it usually takes very long time to reach satisfactory research discoveries.

By the emergence of more powerful technological and computational equipments, it became easy to convert the biological data into sequence of characters through a process called ***sequencing***. Sequencing machines take a biological material (like DNA) as an input, and then it maps this biological material to a sequence of characters stored in text file. It becomes the role of Computer Scientists develop the algorithms and tools that can efficiently store, analyze and visualize the different types of biological sequences located in huge text files or databases.

Objectives

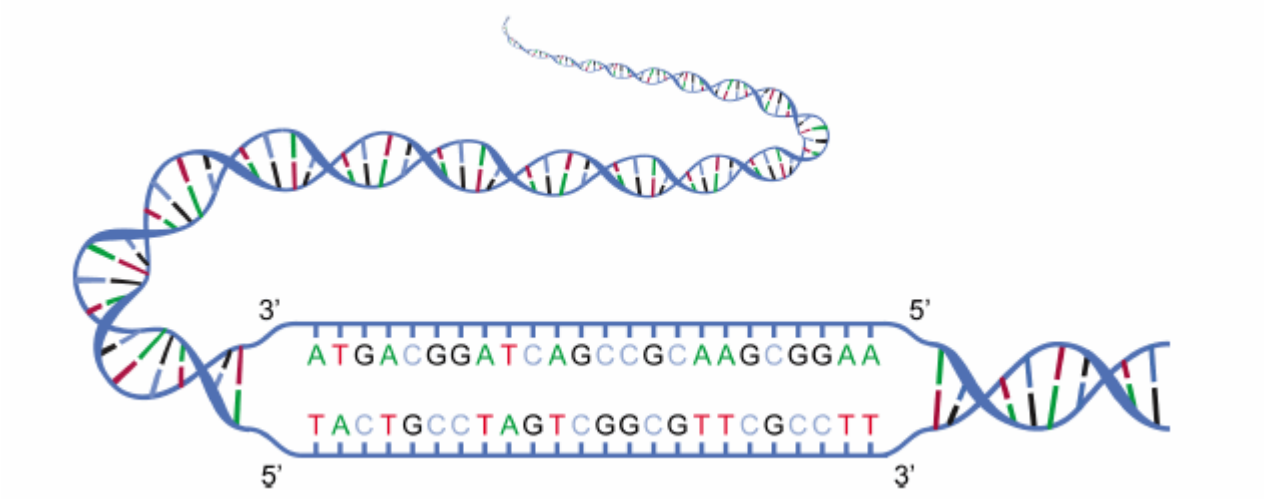
This assignment aims to:
1. Train students on using inheritance, polymorphism, templates, and exception handling among other OOP concepts in C++.
2. Train students on program modularization and separate compilation.
3. Train students on team work and cooperation.
4. Train students on the role of programming in the development of other scientific domains.

Hints

1. **Work in your same teams. Each team consists of the same three students as in the previous assignment.**
2. If you have any problem or difficulty, ask your TA or the professor. **We love to help you.**
3. Please submit **only work that you did yourselves as a team**. If you copy work from your friends or book or the net **you will fail the course**.
4. Submission will be through Acadox before the deadlines.

Details

You should deliver a complete program that helps biologists manage and analyze their sequence data while storing it in a computer. The biological data can be divided into 3 types of sequences: DNA, RNA, and Protein. A DNA sequence can be of type (`promoter`, `motif`, `tail`, `noncoding`). Each DNA sequence consists of 2 strands, each strand is a string that is formed by the following 4 characters (nucleotides): `A`,`C`,`G`,`T` and each strand has its own direction (like 2-way street). `ACAAGG`, `CGATACAG`, `TTACGCCAT`, and `GACCCCTA` are examples of single DNA strands. The second DNA strand is the reversed complement of the first strand. For example, the following are 2 *connected* DNA strands:



A strand is read in direction from 5' to 3'. The second strand can be easily built from the first strand as follows:

1) Convert each T to A, each A to T, each C to G, and each G to C, then
2) Reverse the overall string (to be correctly read from 5' to 3').

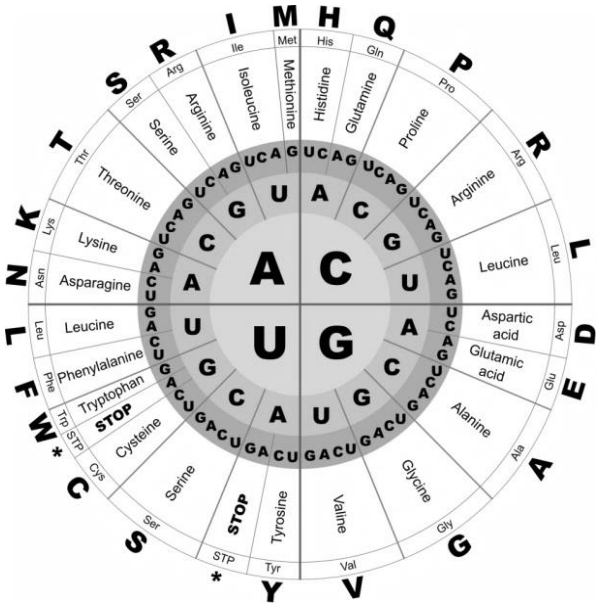
A DNA sequence can be converted to RNA sequence which is then converted to a Protein sequence. RNA results from one DNA strand. To convert a DNA strand to an RNA sequence, replace each `T` by `U`. So, the RNA sequences corresponding to the past DNA examples are `ACAAGG`, `CGAUACAG`, `UUACGCCAU`, and `GACCCCUA` respectively. Note that the actual process of

converting DNA to RNA is more complicated. RNA sequence can be of type (`mRNA`, `pre_mRNA`, `mRNA_exon`, `mRNA_intron`). When converting a DNA strand to an RNA, the resulting RNA should be of type `mRNA` unless the user has decided to alter this default type.

When converting a DNA strand to its complementary strand or to an RNA strand, this should be done from a start index to an end index. If the user entered -1 for both indices, then you should convert the overall DNA strand.

Finally, The RNA sequence can be converted into a Protein sequence. Each 3 characters of the RNA sequence, called `a codon`, can be converted/mapped into one Protein character (called AminoAcid). The mapping of codons into AminoAcids is shown in the following Table or Wheel. So, the RNA sequence `ACGUACGUA`**`UAA`** is converted into the Protein sequence `TYV`. The last codon `UAA` is not converted into an AminoAcid because it is a *stop codon*. Proteins come in different types (`Hormon`, `Enzyme`, `TF`, `Cellular_Function`). When converting an RNA strand to a protein, the protein type should be Cellular_Function unless the user has decided to change the protein type.

1	AAA	K	33	GAA	E
2	AAC	N	34	GAC	D
3	AAG	K	35	GAG	E
4	AAU	N	36	GAU	D
5	ACA	T	37	GCA	A
6	ACC	T	38	GCC	A
7	ACG	T	39	GCG	A
8	ACU	T	40	GCU	A
9	AGA	R	41	GGA	G
10	AGC	S	42	GGC	G
11	AGG	R	43	GGG	G
12	AGU	S	44	GGU	G
13	AUA	I	45	GUA	V
14	AUC	I	46	GUC	V
15	AUG	M	47	GUG	V
16	AUU	I	48	GUU	V
17	CAA	Q	49	UAA	
18	CAC	H	50	UAC	Y
19	CAG	Q	51	UAG	
20	CAU	H	52	UAU	Y
21	CCA	P	53	UCA	S
22	CCC	P	54	UCC	S
23	CCG	P	55	UCG	S
24	CCU	P	56	UCU	S
25	CGA	R	57	UGA	
26	CGC	R	58	UGC	C
27	CGG	R	59	UGG	W
28	CGU	R	60	UGU	C
29	CUA	L	61	UUA	L
30	CUC	L	62	UUC	F
31	CUG	L	63	UUG	L
32	CUU	L	64	UUU	F



Alignment is a very important process that Biologists need to perform in order to discover similarities and differences between sequences of the same type. This can help in finding similar cell attributes and functions. Moreover, it can help in discovering the cause of some disease based on mutations (changes) in the original sequence.

For two sequences to be aligned, Dynamic Programming should be used. It is based on constructing a matrix with dimensions equal to the length of the two sequences, and then start filling the matrix cells according to the used alignment algorithm. We have three famous algorithms: LCS, Global Alignment (also called *Needleman–Wunsch* Algorithm), and Local Alignment (also called *Smith-Waterman* Algorithm). In LCS, you should find the Longest Common Subsequence (substring) between two sequences. This is done by filling the cells of alignment matrix S using the following recurrence:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} & +0 \\ s_{i,j-1} & +0 \\ s_{i-1,j-1} & +1 \text{ if } v_i = w_j \end{cases}$$

The last cell in the matrix reflects the length of the LCS. You can then use Backtracking to read the characters of the LCS from the alignment matrix. Your LCS function between two sequences `V` and `W` can be as follows:

```
LCS(v, w)
   for i ← 0 to |v|
      s_{i,0} ← 0
   for j ← 0 to |w|
      s_{0,j} ← 0
   for i ← 1 to |v|
      for j ← 1 to |w|
         s_{i,j} = max{s_{i-1,j}, s_{i,j-1}, s_{i-1,j-1} + 1 (if v_i = w_j)}
```

Your tasks:

Phase 1: Modelling Submit a simple **class diagram** representing the different classes needed in this application. Moreover, you should manage which attributes/functions should be in the base class and which of them should be in one of the child classes. **Done!**

Phase 2: Implementation: Each class should has its own .h and .cpp files. In addition, you should allow the user of your program to repeatedly choose what type of sequence he/she wants to enter during runtime, and whether he/she wants to convert it into another type of sequence. As a **bonus** part, you should save results into a text file and provide the user with an option to load/save the past sequences he/she processed. Don't forget to apply operator overloading on sequences using

the +, ==, !=, <<, >> operators. You should also decide which functions might be overridden. Overloading the + operator should enable you to concatenate any two sequences of the same type.

Templates:

Classes should be implemented as template classes because sometimes the biologist has his/her data in the format of ASCII codes corresponding the ordinary alphabetic characters. For example, the DNA sequence `CGAT` can be represented as sequence of characters: `'C', 'G', 'A', 'T'`, or as a sequence of ASCII codes: `67, 71, 65, 84`.

Exception Handling:

Your implementation should handle all possible exceptions. For example, your program should throw an exception in case the user entered a DNA sequence that contain a strange character (all characters except A, C, G, T) or a delimiter. The same should be applied for RNA and Protein sequences according to their aforementioned alphabet.

How to program this assignment?

- Firstly, start by building your classes in a class diagram.
- After validating the class diagram, implement your classes with `string` attributes, not `char*`.
- Then, start building your child classes with all the required functionalities.
- After that, start converting the `string` attributes into `char*`, and handle the required changes in constructors, destructors, and the = operator.
- Finally, apply templates and exception handling to the overall hierarchy.
- It is important to take regular backups of your stable code before starting in applying a new feature.

Hints:

- Each class should have its own .h and .cpp files.
- As **bonus**, functions to `LoadSequenceFromFile(char* filename)` and `SaveSequenceToFile(char* filename)` are pure virtual functions and should be overridden for all the child classes.
- A menu should periodically appear to the user allowing him/her to choose executing any of the available operations.
- For simplicity, you can implement the alignment using the Longest-CommonSubsequence (LCS) algorithm. Other alignment techniques such as Global Alignment and Local Alignment will be considered as **bonus**.

Online Helpful Resources:

- https://www.youtube.com/watch?v=h5mJbP23Buo
- https://www.youtube.com/watch?v=_POdWsii7AI
- https://www.youtube.com/watch?v=0Elo-zX1k8M - https://www.youtube.com/watch?v=gG7uCskUOrA –
- **Alignment:**
    - https://en.wikipedia.org/wiki/Sequence_alignment#Dynamic_programming
    - https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm
    - https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Grading Criteria: 900 points + 150 bonus points

Feature	Grade
Phase 1: Modeling (1 Dec.)	**100**
Phase 2: Implementation (8 Dec.)	
Comments & Readability	25
Good User Interface	25
Enums	50
`char *` instead of `string`	50
Parent Class	50
Child Classes	150
Codon Data Structure	75
Loading Codons from File	25
Separate .h and .cpp Files	50
Virtual Functions & Polymorphism	100
Templates	50
Exception Handling	75
LCS Alignment	75
Bonus:	
Files for Loading/Saving Sequences	50
Global/Local Alignment	50
Code Versioning.. Github	50