School of Computing & Information Technology

# CSCI251/CSCI851 Advanced Programming
# Spring 2018

## Assignment 3 (Worth 10%)

Due 11:55pm Friday $26^{th}$ October 2018. (End of Week Thirteen)

This assignment involves implementing a container to support some basic genetic related manipulations.

The genetic code is described in as a sequence of nucleotides or bases. Those bases describe our basic alphabet and for our standard genetics there are 4 bases in the alphabet.



| Base symbol | A | G | C | T |
|---|---|---|---|---|
| Base name | Adenine | Guanine | Cytosine | Thymine |

Sequences of three bases, referred to as triplets or codons, map to amino acids. Chains of amino acids make proteins.



There are now attempts to develop non-standard genetics with synthetic bases and unnatural amino acids. Those would typically extend the base set but we are going to use a fictictious base set with only 3 bases in the alphabet.



| Base symbol | A | C | B |
|---|---|---|---|
| Base name | Adenine | Cytosine | Bob |

To simplify processing we go to assume these also code to amino acids using sequences of 3 bases, which isn't strictly necessary in nonstandard amino acids.

The provided data files, `Codon3.txt` and `Codon4.txt`, give example mappings between codons and amino acids & instructions for the two different alphabets. The `Start` and `Stop` codons are instructions, not amino acids. They tell us when we begin and end a coding sequence. There will only be one line for each per file. The amino acids/instructions can generally have multiple codons that map to them.

You should only load the correspondence table once.

## Classes: A container and two bases

You are to implement one container class template for storing an ordered list of objects associated with the nucleotides.

To avoid confusion with Base classes in the sense of inheritence I will refer to the classes for our symbols as being Alphabet classes. These classes store information about the symbols available, as in the tables above, and also information about the codons and their mapping to amino acids.

1. AlphabetFour: The standard 4 base alphabet.
2. AlphabetThree: The non–standard 3 base alphabet.

Each class should have the following functionality, although they can be organised differently:

1. Loading the mapping between codons and amino acids from the data files. This should only be done once.
2. Checking if the mapping covers all possible codons, so all possible combinations of 3 alphabet symbols.
3. Mapping three elements from the alphabet, so a codon, to an amino acid or an instruction.

Your container should have the following functionality:

1. Report on and validate the length of the container content. The length must be a multiple of 3.
2. Report on the symbol distribution for the container.
3. List the complete content of the container. This should display 60 elements per line, as in the sample data files, `GeneTest4-1.txt`.
4. Display all the encoded active amino acid chains. These means we read and process codons from our data sequence, so symbols 3 at a time, until we reach a Start instruction. We then output the amino acids using the names seperated by dashes, until a Stop instruction is reached. The number of amino acids in each active amino acid chain should be reported. You then continue reading and processing codons as before.
   (a) If you are in an active sequence and reach the end of your data without reaching a Stop instruction this should be reported.
   (b) You also need to report the number of codons not displayed in between Stop and Start instructions, and at the beginning or end as appropriate.



The output based on the sequence processing should be something like the following:

```
5 codons read and ignored.

Sequence 1: Start-Phe-Lys-Stop.
Contains 2 amino acids.

10 codons read and ignored.

Sequence 2: Start-Try-Asp-Ala-Pro-Pro-The-Iso-Stop.
Contains 7 amino acids.
...
```

Note that other than for counting the symbols you only deal with them in blocks of three. So AATGCA is considered as AAT–GCA, not A-ATG-CA... with ATG being the Start symbol.

## Genetics

Your code should compile on Banshee according to instructions you provide in a `Readme.txt` file. The program shold be named `Genetics`. The program should run as follows:

```
$ ./Genetics 3 Codons3.txt Datafile.txt
$ ./Genetics 4 Codons4.txt Datafile.txt
```

where the 3 or 4 distinguishes whether we are dealing with an `AlphabetThree` case or an `AlphabetFour` case. That first value must be a 3 or 4. The `Codons3.txt` and `Codons4.txt` files could be named something else but will follow the codon to amino acid mapping files given. The file `Datafile.txt` will follow the structure of `GeneTest3-1.txt` and `GeneTest4-1.txt` and that's the data you operate on.

## So what do you do?

Based on 3 or 4 you get set up to deal with data of that type, so reading from the `Codons3.txt` or `Codons4.txt` type file specified as appropriate. Test and report on the validity of the table.

You then read from the provided data, into your container. Carry out the length validation and report on the length. Report on the distribution of symbols within your sequence. Print out the entire sequence.

The display all of the encoded active amino acid chains.

The user shouldn't need to give any input other than that given on the command line. If there is a problem with the data, so an invalid symbol or invalid Codons file, report the problem and abort.



## General notes

These are some general rules about what you should and shouldn't do.

1. Your assignment should be sensibly organised with the same kind of expectations in this area as A1.
2. Your code must compile on Banshee with the compilation instructions you provide in `Readme.txt`. If it doesn't you will likely be given zero for the programming part of this assignment.

Submission is via Moodle. **Please submit a single zip file with all your assignment files in it. Do not include subdirectories, that slows down marking!**

1. The deadline is 11:55pm Friday $26^{th}$ October 2018.
2. Late submissions will be marked with a 25% deduction for each day, including days over the weekend.
3. Submissions more than three days late will not be marked, unless an extension has been granted.
4. If you need an extension apply through SOLS, if possible **before** the assignment deadline.
5. Plagiarism is treated seriously. Students involved will likely receive zero.