School of Computing Sciences

Module: CMP-5015A Programming 2
Assignment: C++ Coursework

Set by: Gavin Cawley (g.cawley@uea.ac.uk)
Date set: *date missing*
Value: 30%

Date due: 22nd January 2021 9:30am
Returned by: N/A
Submission: Blackboard

Learning outcomes

On completing this exercise, you will have demonstrated the ability to write a relatively simple class in the C++ programming language, including encapsulation, the use of constructors, accessor methods, operator overloading (particularly for use in stream-based I/O) etc. You will also have demonstrated your ability to perform unit testing of your class and integrating it into an application written by someone else, simulating the way programs are typically produced in industry — by a team rather than an individual programmer. The other parts of the application however may have a number of errors, so the exercise also tests your ability to understand code written by others and to debug it and improve it, which are also key skills for any programmer, as maintenance is an important part of the software life-cycle for most non-trivial programs.

Specification

Overview

The aim of this assignment is to develop programming, testing, debugging and refactoring skills, by contributing a class to a C++ application that performs a simple analysis of a database of proteins, and then testing, debugging and improving it.

Description

Proteins are the components of the mechanisms of our cells, and are composed of a string of elementary building blocks, known as amino acids. The function of proteins in our bodies depends on their shape, which is determined by this sequence. Most proteins naturally fold to form a compact shape, but some don't and are described as "intrinsically disordered proteins", and are of considerable scientific and medical interest, for instance the prion protein that gives rise to Creutzfeldt–Jakob disease. Some amino acids are hydrophilic (like being in water) and others are hydrophobic (dislike being in water), and many proteins fold such that the hydrophilic amino acids are on the outside of the protein and the hydrophobic amino acids are hidden inside the folded protein in the watery cytoplasm within the cells of our body. Amino acids are also able to carry an electrical charge and this is also thought to affect the ability of a protein to fold to form a stable shape in solution.

Table 1: Table showing the hydrophobicity of amino acids, according to the Kyte-Dootittle scale, and the electrical charge.

Amino Acid	Symbol	Hydrophobicity	Charge
Alanine	A	+1.8	+0
Arginine	R	-4.5	+1
Asparagine	N	-3.5	+0
Aspartic acid	D	-3.5	-1
Cysteine	C	+2.5	+0
Glutamine	Q	-3.5	+0
Glutamic acid	E	-3.5	-1
Glycine	G	-0.4	+0
Histidine	H	-3.2	+1
Leucine	L	+3.8	+0
Isoleucine	I	+4.5	+0
Lysine	K	-3.9	+1
Methionine	M	+1.9	+0
Phenylalanine	F	+2.8	+0
Proline	P	-1.6	+0
Serine	S	-0.8	+0
Threonine	T	-0.7	+0
Tryptophan	W	-0.9	+0
Tyrosine	Y	-1.3	+0
Valine	V	+4.2	+0

Prilusky et al. [2005] suggest a simple mathematical method, called "FoldIndex", to predict whether a protein is likely to be intrinsically disordered, based on the hydrophobicity and charge of its amino acid sequence:

$$I = 2.785 \times H - R - 1.151,$$

where H is the average (arithmetic mean) hydrophobicity of the amino acids, and R is the absolute magnitude of the difference between the number of positively charged and negatively charged amino acids, divided by the length of the sequence. Table 1 shows the electrical charge and hydrophobicity (according to the Kyte-Doolittle scale) for each amino acid. Note that before the hydrophobicities are used to calculate the FoldIndex, they must be scaled to lie in the range 0 to 1 (i.e. such that Arginine has a normalised hydrophobicity of zero and Isoleucine a normalised hydrophobicity of one). Positive values of FoldIndex suggest the protein will be folded, negative values suggest the propensity for intrinsic disorder — the more negative the value, the more disordered the protein is likely to be.

The program is intended to load a database of proteins, update it to include calculated values for the FoldIndex for each protein, and then save the database to another file, sorted in descending order of the propensity for intrinsic disorder. The main part of the prototype application `main.cpp` and the input database, `disprot.csv` are provided on BlackBoard. Your main task is to implement a class called `Protein`, to represent all of the information about a given protein, comprised of:

`ID` - a unique identifier assigned to each protein (stored as a `string`).

`sequence` - a `string` of upper-case characters describing the sequence of amino acids comprising the protein (c.f. Table 1).

`label` - a `string` of the same length, where each character indicates that the corresponding amino acid is either part of a folded region, indicated by a '–', or part of an intrinsically disordered region, indicated by a '#'.

`foldIndex` - a value of the FoldIndex for that protein (or zero if the value has yet to be computed), stored as a `double`.

The class must provide default and ordinary constructors, an appropriate set of accessor methods, a method called `setFoldIndex` used to set the value of the FoldIndex for the protein, a full set of relational operators that order `Protein` objects according to their `foldIndex` value and operators (<< and >>) for stream-based I/O. The required format for both operators is illustrated by the format of `disprot.csv`. Note that the format used by both << and >> must be identical, so that I/O operators are mutually compatible (hint: `std::quoted`). The interfaces for the methods/functions must be compatible with their usage in `main.cpp`. Note: implementing this class does not require any understanding of protein structure, beyond that given in this assignment brief.

The next step is to write a test-harness to perform "unit testing" of the class you have written. This should be implemented in a source code file called `ProteinTest.cpp`, which should provide a function called `testProtein` that has no parameters and returns `true` if the code passes all tests and `false` otherwise. It should document the steps taken to test your component of the program, and should be easily repeatable so that if the source code is modified, it is easy to check that a regression error had not been introduced. The test harness should not generate large amounts of diagnostic information, especially where the code passes the tests, as this would make testing for regression errors unduly time consuming.

The third step is to integrate your class with the prototype program, provided via BlackBoard. Sadly all non-trivial programs contain bugs, and this program is no exception. Test the system as a whole, and identify and correct any bugs that you find. These may prevent the program from compiling, they may cause the program to crash, or they may cause the program to run, but produce incorrect answers.

The last step is to improve the efficiency of the program, by *refactoring* `main.cpp` so that it stores a `vector` of pointers to `Protein` objects, rather than storing the objects directly (which may result in a lot of unnecessary copying). This sort of modification and improvement of programs is a common actvity in the maintenance phase of the software life-cycle.

References

Jaime Prilusky, Clifford E. Felder, Tzviya Zeev-Ben-Mordehai, Edwin H. Rydberg, Orna Man, Jacques S. Beckmann, Israel Silman, and Joel L. Sussman. FoldIndex©: a simple tool to predict whether a given protein sequence is intrinsically unfolded. *Bioinformatics*, 21(16): 3435–3438, 06 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bti537. URL https://doi.org/10.1093/bioinformatics/bti537.

Relationship to formative assessment

This assignment build on skills developed through the lab exercises on this module and those on CMP-4008Y - Programming 1.

Deliverables

Your solution must be formatted using the PASS program, available on all laboratory machines, and also via the server at `http://pass.cmp.uea.ac.uk`, to produce a PDF file containing the source code of the program, the compiler messages (if any) and the output of the program and the specified file generated by the program. Make sure you familiarise yourself with both versions of PASS in case of network/server problems on the day of the test. Your submission must include at least the following files:

- `Protein.h`
- `Protein.cpp`
- `ProteinTest.cpp`
- `main.cpp` - an initial prototype provided via BlackBoard

The PASS program uses the `gcc/g++` compiler from the GNU Compiler Collection tools. It will be set to conform to the C++17 standard. It is your responsibility to ensure that your program is compatible with that compiler and that standard.

The PASS program is not able to provide input to your program via the keyboard, so programs with a menu system, or which expect user input of some kind are not compatible with PASS. Design your program to operate correctly without user input from the keyboard.

If you develop your solution on a computer other than the laboratory machines, make sure that you leave time to test it properly with PASS, in case of any unforeseen portability issues. If your program works correctly under CLion on the lab machines, but does not operate correctly using PASS, then it is likely that the data file (`disprot.csv`) you are using has become corrupted in some way. PASS downloads a fresh version of the file to test your submission, so this is perhaps the most likely explanation.

Resources

- `https://stackoverflow.com/` - An excellent site for finding information about specific issues relating to various programming languages, including Java. It is important however not to become too reliant on sites such as `StackOverflow`, which are great for details, but don't give the "big picture", so it is difficult to get a good understanding of programming in this way. Note that if you re-use or modify code found on-line, then you must provide a comment giving the URL and a brief explanation of what modifications have been made. Re-using code found on-line without proper attribution would constitute plagiarism. Obviously it would also be an offence to ask questions directly relating to this assignment on StackOverflow or other programming forums.
- `https://cppreference.com` - An excellent resource for C and C++ programming, providing detailed documentation on both the language and the libraries. This is the site I use most often when I am programming (as well as using books).
- It is likely that the lectures and live programming demonstrations for this module provide many examples of C++ programming techniques and idioms that may be useful for this assignment.

Marking Scheme

Marks will be awarded according to the proportion of specifications successfully implemented, programming style (indentation, good choice of identifiers, commenting etc.), and appropriate use of programming constructs. It is not sufficient that the program generates the correct output, professional programmers are required to produce maintainable code that is easy to understand, easy to debug when (rather than if) bug reports are received and easy to extend. The code needs to be modular, where each module has a well-defined interface to the rest of the program. The function of modules should be made as generic as possible, so that it not only solves the problem specified today, but can easily be modified or extended to implement future requirements without undue cost. The code should also be reasonable efficient. Marks may also be awarded for correct use of more advanced programming constructs not covered in the lectures.

- **Implementation of `Protein.h` and `Protein.cpp` -** Marks will be awarded according to the quality and correctness of the code, according to the specifications. [15 marks]
- **Unit Testing -** A file called `ProteinTest.cpp` must be implemented, demonstrating the steps taken to test `Protein.h` and `Protein.cpp`. Marks will be awarded according to the quality and thoroughness of the tests. [5 marks]
- **System testing** - Marks are awarded for identifying and correcting the errors/bugs in the file `main.cpp` and for fully and accurately describing the nature of the error in the revision history in the block comment at the start of the file. [5 marks]
- **Improving efficiency -** The original prototype is somewhat inefficient as it is based on a `vector` of `Protein` objects, which may result in unnecessary copying of objects. Marks will be awarded according to the quality of the refactoring. Note that substantially re-writing `main.cpp` will not attract a good mark as the new code will require testing and debugging, so adding large amounts of code is likely to *reduce* the reliability of the program. [5 marks]