CS 241L – Data Organization
Spring 2022

Project 5

Total points: 70

Due on Monday, 4/11/22

In this project, you will write a C program that uses a linked list to store and mutate an amino acid sequence.

(1) Background

Our genome encodes all information that our cells need to create proteins. During transcription, RNA is created from DNA, and in translation, this RNA is read to create the protein. Each three letters in the RNA sequence (a codon) translates to one amino acid. Amino acids are the building blocks of proteins. There are 22 amino acids that can compose proteins. A list of the 22 amino acids and their letter code is provided below.

Alanine	A
Arginine	R
Asparagine	N
Aspartic acid	D
Cysteine	C
Glutamic acid	E
Glutamine	Q
Glycine	G
Histidine	H
Isoleucine	I
Leucine	L

Lysine	K
Methionine	M
Phenylalanine	F
Proline	P
Serine	S
Threonine	T
Tryptophan	W
Tyrosine	Y
Valine	V
Selenocysteine	U
Pyrrolysine	O

During translation, each codon encodes an amino acid, and a sequence of codons results in a sequence of amino acids that are chained together to form a protein. The protein folds to create a 3-dimensional structure that performs essential functions in cells.

Viruses contain genetic code that enables them to replicate, but they are unable to replicate by themselves. They need to hijack a living cell translation mechanism to create viral proteins that are then used to create new viruses. Viruses can mutate their genetic code quickly. When letters are deleted or swapped in the genetic code, the codons also change

and amino acids in the protein may change as well. These small changes to the amino acid sequence of the protein can sometimes have a big effect. Some mutations, for example, can make the virus more transmissible.

Some mutations to the SARS-CoV-2 virus (the virus that transmits COVID-19) make it more transmissible. The alpha variant (B.1.1.7) appeared in early 2021. For this variant, there are multiple mutations in the entire genetic code of SARS-CoV-2, but more notably, there are many mutations in the part of its genome that encodes instructions for making the spike protein. The spike protein on the surface of SARS-CoV-2 viruses is the protein responsible for binding the virus to human cells. The B.1.1.7 mutations on the spike protein make it attach more strongly to cells.

(2) Mutations in the spike protein of SARS-CoV-2

a) The input file: `spike.in`

In this lab, you are given the input file `spike.in`. This file contains the original (not mutated) sequence of amino acids that make the spike protein of SARS-CoV-2. There are 1273 amino acids in this protein. The sequence is shown below:

```
MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS
NVTWFHAIHVSGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIV
NNATNVVIKVCEFQFCNDPFLGVYYHKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLE
GKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINITRFQT
LLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETK
CTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISN
CVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIAD
YNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEIYQAGSTPC
NGVEGFNCYFPLQSYGFQPTNGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVN
FNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPCSFGGVSVITP
GTNTSNQVAVLYQDVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSY
ECDIPIGAGICASYQTQTNSPRRARSVASQSIIAYTMSLGAENSVAYSNNSIAIPTNFTI
SVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQE
VFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDC
LGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAM
QMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASALGKLQDVVNQNAQALN
TLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRA
SANLAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPA
ICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDP
LQPELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQKEIDRLNEVAKNLNESLIDL
QELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCCKFDEDD
SEPVLKGVKLHYT
```

This sequence was downloaded from the Uniprot database and can be found at: https://www.uniprot.org/uniprot/P0DTC2.

You will write a C program called `protein.c` that stores each amino acid in this sequence in a linked list. Then, you will mutate the spike protein to create the B.1.1.7 spike protein sequence, which is output in the file `mutated.out`. The mutated amino acids are given as command line arguments.

b) The mutations (command line arguments)

Your program should read the list of amino acids to mutate (or delete) from the command line arguments. You will run your program as:

```
./protein B.1.1.7 d69 d70 d144 N501Y A570D P681H T716I S982A D1118H < spike.in > mutated.out
```

The first argument after the call to run the program is `B.1.1.7`, which is simply the name of the variant. After the variant name, there is a list of mutations to perform, of two kinds:

dXXXX: Delete amino acid at position XXXX in spike.in file. These mutations are deletions. Eg. `d69` means that amino acid in position 69 in spike.in is deleted.

&XXXX$: Find amino acid & in position XXXX and replace it by $. Eg. `N501Y` means that amino acid N at position 501 in spike.in is replaced by a Y.

Your C program should work for any mutation and any protein sequence that is entered. Therefore, you should use dynamic arrays to store information given to the program in the command line arguments.

c) The output file: `mutated.out`

Once mutations and deletions are performed, you will print your new linked list in an output file, `mutated.out` (using stdout redirection, > mutated.out). Your `mutated.out` file must match the file `b117.out` that is provided to you.

In `b117.out`, you will find a header line explaining that the file contains the protein sequence for the variant entered in the command line arguments. The amino acids are grouped by 10. The line printed above the amino acids provides labels (by every 10 amino acids) for the locations of the amino acids in the new sequence.

(3) The Makefile

We will be using a Makefile for the first time in this project. You will **not** need to modify it. Its contents are shown below:

```
CC=gcc
CFLAGS=-pedantic -ansi -Wall
B117 = B.1.1.7 d69 d70 d144 N501Y A570D P681H T716I S982A D1118H

all: b117

b117: protein
	./protein $(B117) < spike.in > mutated.out
```

```
protein: protein.o
	$(CC) $(CFLAGS) -o protein protein.o

protein.o:
	$(CC) $(CFLAGS) -c protein.c

clean:
	rm protein protein.o
```

The first 3 lines define constant strings that are used in the instructions. Eg. $(CC) is replaced by the string defined in CC, which is gcc.

The instruction `all` contains what happens when the user types `make` on the terminal screen. It is linked to the instruction `b117`, which runs the program with the appropriate command line arguments in $(B117).

Instructions to build the object file and the executable are in `protein.o` and `protein`, respectively. The last instruction, `clean`, removes the executable and object file when the user types `make clean` on the terminal. It is important to clean whenever you make changes to the code and need to compile again.

(4) OPTIONAL: Extra credit (5 points)

What happens if the mutations entered in the command line arguments are not in the expected format? For example, if you enter:

./protein variant1 d 70 < spike.in

d 70 is not in the expected format for deletions, which would be d70.

Modify your protein.c program to handle errors in the command line arguments. Some of the errors you can expect are extra spaces, wrong characters, wrong amino acid letter at position, missing amino acid letter in mutations, etc. Think about all possible errors that could occur. Your program should output an error message and not build a linked list with mutations if an error is encountered. If your program `protein.c` can handle these errors, submit it as `proteinEM.c`. Your program `proteinEM.c` will be tested with the regular grader file that is used with `protein.c`, plus 5 other unknown error cases. One point is given for each error case that is handled correctly.

What to submit:

The files:

`protein.c`
`mutated.out`
`proteinEM.c` **(extra credit, optional: submit instead of protein.c)**

IMPORTANT CHANGE FROM PREVIOUS PROJECTS:

Submit ALL FILES a single zip or tar file. Name the compressed file:

Project5_FirstNameLastName

Grading Rubric:

+ 1 pts: Your C program starts with a comment on top of the file with your name and description of the program.
+ 1 pts: Your C program compiles with the `-ansi -pedantic -Wall` options without errors or warnings.
+ 1 pts: Your C program follows the class coding standards, including function comments.
+ 2 pts: You follow all directions in this PDF.
+ 15 pts: You save command line arguments in dynamic arrays using dynamic memory allocation.
+ 15 pts: Your output file `mutated.out` passes a diff test with `b117.out`
+ 15 pts: Your program runs without errors or memory leaks in Valgrind.
+ 20 pts: Your program correctly mutates an unknown protein (matches a diff test with the grader file).