

## CSC 112 Lab 5: Protein Sequence Searching Fall 2009

Due: Sunday, October 4th, 9:00pm

### *Purpose*

---

This lab is intended to:

- Provide additional chances to work with dynamic memory allocation, particularly array allocation
- Reinforce the notion of looping control structures
- Use simple file input/output and command line arguments
- Simulate a real-world problem in the domain of bioinformatics

### *Introduction to Biological Sequences: Proteins*

---

Almost all vital biological processes are controlled by proteins. A protein is, quite simply, a chain of bonded amino acids folded into a particular shape. Important for us, a protein can just be thought of as a long sequence of characters from a limited alphabet. The 20-character amino acid alphabet is as follows:

Amino Acid	Symbol	Amino Acid	Symbol
Alanine	A	Leucine	L
Arginine	R	Lysine	K
Asparagine	N	Methionine	M
Aspartic acid	D	Phenylalanine	F
Cysteine	C	Proline	P
Glutamic acid	E	Serine	S
Glutamine	Q	Threonine	T
Glycine	G	Tryptophan	W
Histidine	H	Tyrosine	Y
Isoleucine	I	Valine	V

As an example of an encoded protein is below (the H1N1 neuraminidase protein):

```
MNPNQKIITIGSVCMTIGMANLILQIGNIISIWIHSHSIQLGNQNQIETCNQSVITYENNTWVNQTYVNIS
NTNFAAGQSVVSVKLAGNSSLCPVSGWAIYSKDNSIRIGSKGDVVFVIREPFISCSPLECRTFFLTQGALL
NDKHSNGTIKDRSPYRTLMSCPIGEVPSPYNSRFESVAWSASACHDGINWLTIGISGPDNGAVAVLKYNG
IITDTIKSWRNNILRTQESECACVNGSCFTVMTDGPDSGQASYKIFRIEKGKIVKSVEMNAPNYHYEACS
CYPDSSEITCVCARDNWHGSRNPWVSFNQNLEYQIGYICSGIFGDNPRPNDKTGSCGPVSSNGANGVKGFS
FKYGNVWIGRTKSISSRNGFEMIWDPNGWTGTDNNFSIKQDIVGINEWSGYSGSFVQHPELTGLDCIRP
CFWVELIRGRPKENTIWTSGSSISFCGVNSDVTGWSWPDGAELPFTIDK
```

## General Problem: Searching for Short Protein Sequences

A common task encountered in working with protein sequences is searching for whether a short protein sequence of interest, hereafter referred to as a *segment*, is part of a larger protein sequence of interest.

For example, assume the segment *VNGSC* is known to be a common site where two proteins can bind together. One might wish to search to determine if any instances of *VNGSC* occur in a newly sequenced protein of interest. For our H1N1 neuraminidase protein, the answer would be yes, as an instance of *VNGSC* is present in the neuraminidase sequence as highlighted (underlined) below:

```
MNPNQKIITIGSVCM TIGMANLILQIGNIISIWI SHSIQLGNQNQIETCNQSVITYENNTWVNQTYVNIS
NTNFAAGQSVSVKLAGNSSLCPVSGWAIYSKDNSIRIGSKGDV FVIREPFISCSPLECRTFFLTQGALL
NDKHSNGTIKDRSPYRTLMS CPIGEVPSYNSRFESVAWSASACHDGINWLTIGISGPDNGAVAVLKYNG
IITDTIKSWRNNILRTQESE CACVNGSCFTVMTDGP SDGQASYKIFRIEKGKIVKSVEMNAPNYHYECS
CYPDSSEITCVCRDNWHG SNRPWVSFNQNLEYQIGYICSGIFGDNPRPNDKTGSCGPVSSNGANGVKGFS
FKYGNVWIGRTKSISSRNG FEMIWDPNGWTGTDNNFSIKQDIVGINESGYSGSFVQHPELTGLDCIRP
CFWVELIRGRPKENTI IWTSGSSISFCGVNSDTV GWSWPDGAELPFTIDK
```

## Programming Task #1: M-Length Segment Database

Assume that your employer, a bio-tech company, maintains a list of protein segments of interest to them. This list is maintained as a text file with the following format.

```
numberOfEntries segmentLength
entry1
entry2
entry3
...
entryN
```

For any given segment file, all segments within the file are the same length, as specified by the variable *segmentLength*.

Write a function called `loadDatabase` which, given a filename of a validly formatted text file, dynamically generates and returns a perfectly sized array of the strings that represent the segments of interest, the number of segments that were read, and the segment length that was read. Also write a function called `saveDatabase` which saves the database back out in the appropriate text format.

The function headers for the two functions above should be as follows:

```
void loadDatabase(const string & filename, string* & database,
int & numberOfEntries, int & segmentLength)
void saveDatabase(const string & filename, string* database, int
numberOfEntries, int segmentLength)
```

## Programming Task #2: Update Database

Since the list of segments of interest might be very long, the task of maintaining the database is automated. Assume that to add new segments of interest to the database or remove segments of interest, a 2<sup>nd</sup> text file is written which has the following format:

1. The first line of the update text file has the number of changes to be made
2. Each following line starts with either an A or a D, followed by a space, followed by a segment string.
3. If the line starts with an A, then the segment is to be added to the database unless it is already there (so no repeats).
4. If the line starts with a D, then the segment is to be removed from the database if it is in the database. If it isn't already in the database, nothing is changed.

Write a function called `updateDatabase` which, given the database of segments and the filename of an update file, opens the update file and adds and removes the sequences as indicated in the update file. At the end of this function, one should also have a perfectly sized (fit to the actual number of segments remaining in the database) array in memory.

The function header for the function above should be as follows:

```
void updateDatabase(const string & updateFilename, string* & database, int & numberOfEntries)
```

This is the most complicated function – you may want to write tasks 3 and 4 first!

## Programming Task #3: Clean Database

Occasionally, the database is not updated through the `updateDatabase` function but by hand. In this situation, someone else at your company opens up the actual segment database file, adds new entries at the bottom, and updates the count of segments at the top. Remember, however, that it is required that the database contains no repeat entries (a database file only updated using the `updateDatabase` method guarantees this, by rule 3 of the update rules). Since your colleague may introduce repeats through his manual updates, a process for occasionally cleaning up the database is required.

Write a function called `cleanDatabase` which, given the in-memory database of segments, goes through and removes any repeat sequences. Note that you should only do this work on the in-memory database. At the end of this function, one should also have a perfectly sized array.

The function header for the function described above should be as follows:

```
void cleanDatabase(string* & database, int & numberOfEntries)
```

### *Programming Task #4: Search Database*

---

Assume you are given a protein sequence of interest, P, whose size is larger than the segment size in your database. You are tasked with writing code that will search for matches of segments against your sequence. You have been told you should follow the following basic technique to search:

Given an m-character segment length, loop down the protein sequence of interest P, extracting characters 0-(m-1), 1-m, 2-(m+1), 3-(m+2), and so on. For each extracted set of characters, compare against each entry in the segment database. If there is a match write to the screen the segment that matched and its starting position in the protein of interest (assume the first amino acid in the protein is position 0).

As an example, using the protein sequence: MNPNQKIITIG  
and a segment database of:

```
3
PNQK
SRAT
ITIG
```

the following strings should be searched for in the segment database:

```
MNPN // positions (0-3) of protein
NPNQ // positions (1-4) of protein
PNQK // positions (2-5) of protein
NQKI // .. and so on
QKII
KIIT
IITI
ITIG
```

and the following output should be printed:

```
PNQK at position 2
ITIG at position 7
```

Write a function called `searchDatabase` which, when given the in-memory database of interest, a string representing a protein sequence of interest, and an output filename, implements the functionality described above and prints the output to the specified file.

For this lab, you **must** extract short pieces from the large sequence of interest and search in the database (as shown in the example above), instead of selecting database segments and searching in the large sequence of interest. You will likely want to make use of the string class `length()` and `substr(int start, int lengthToExtract)` methods. (<http://www.cplusplus.com/reference/string/string/>)

The function header for the function described above should be as follows:

```
void searchDatabase(string* database, const string &
proteinOfInterest, int numberOfEntries, int segmentLength, const
string & outputFilename)
```

### *Programming Task #5: main() Function*

---

Write a main function you feel is appropriate for testing the lab. I will use my own main function in grading the lab, but will evaluate yours for whether or not it appropriately tested the different parts of your lab. *Also note that your function headers must exactly match those specified in the lab so that an external main function can replace yours without causing problems.*

### *Completing the Lab*

---

Please adhere to all of the following points to receive the maximum credit for this program:

- Turn-in electronically the one C++ source file for this program. Submit your `lab5.cpp` file under the “Lab 5” section of the Assignments Menu on Blackboard by the due date and time. You are not required to tar and gzip your code this time (since it’s only one file).
- Assume valid inputs.
- Use dynamic memory allocation for this program. You are responsible for deleting any memory you dynamically allocate.
- Adhere to documentation and coding style standards