CSE 310 Assignment #6
(Max. Points: 30)

Due on: Thursday, Nov. 16th, 2017, 11:59pm Arizona time

General Instructions:

- This is an individual assignment, please do not collaborate. *Make sure that you write every line of your own code. Using code written by someone else will be considered a violation of the academic integrity and will result in a report sent to the Dean's office.*
- It must be submitted online through Blackboard submission link! We don't accept submissions through emails.
- Submission link will be closed automatically once the due date/time is past and **no late assignment will be accepted**.
- You will be allowed 3 times to submit the assignment, but we will only grade your last submission.

Objectives

- Implementation to solve the LCS problem by dynamic programing.

Assignment Description

The longest common subsequence problem is given two sequences, *A* and *B*, find the longest sequence that is a subsequence of both of them. This can be done by an $O(nm)$ dynamic programming algorithm as we learned in Chp.15. In this assignment, you're required to implement the algorithm in C++.

1. Intro. Application of the algorithm

We knew that DNA is a sequence of four nucleotides: **A**denine, **C**ytosine, **G**uanine, and **T**hymine. In general, we represent a DNA digitally by long strings of A, C, G, and T. In Biology, one of the most simplistic tests to determine genetic similarity is the longest *subsequence* of nucleotides common to both species. Determining this similarity value is of great use to bioinformatics scientists to determine their positions on an evolutionary tree.

Also in computer world, computers execute *machine code*, a series of 0s and 1s. It is a common problem to determine the longest common sub*string* of two strings (0s and 1s) in web search engines.

It is simple to write a brute-force program to solve above problems, but DNA and a machine code can be millions of characters long, thus a brute-force method could take months (or even years) to give results for real-world data. This assignment asks you to implements a dynamic programming method to efficiently solving the longest common subsequence problem.

2. Programming requirements

1) You need to implement the basic algorithm which takes $O(nm)$ time & space to find the subsequence. Your algorithm should identify the longest common sub-sequence (and not just give its length).

2) Input, Error checking and Output

Program Input: your program should prompt the user to enter two strings from keyboard, representing two DNA sequences or two strings of 0s ad 1s.

Error checking: before executing the algorithm, your program should check both strings to ensure only acceptable DNA letters (A, C, G, T) or valid strings (0s and 1s) are entered. Do not run the algorithm if they contain any errors inside. In the event of an error, indicate how many characters are illegal, and show the original string with the illegal letters highlighted, for example show an error message of the following type

```
Input contains error
Error in String #1:   aacgttcOgMa
Error in String #2:   ggataccaSat
```

Program Output: your program should output the following on screen:

(1) The length of the longest subsequence
(2) The actual subsequence string.
(3) The dynamic subsequence table

3. Implementation/Documentation Requirements

- You need to implement this program using C++ and it need to read from keyboard.
- Your program needs to compile and execute in *general.asu.edu*.
- Use functions to modularize your code. The following are required to organize your functions.

`functions.h`	header file for functions
`functions.cpp`	implementation of the functions
`main.cpp`	main program implementation that calls the functions

- Your code should be well documented and commented, and every file should have your name at its top.
- Also you are not allowed to use any predefined data structures (such as ones in the library in C++, etc.) except arrays and strings, you need to build your own data structures and operations associated with them (such as insert or search).

Copy any codes from other people's programs is considered to be cheating and will lead to a report to the Dean and you will be penalized. Also check the rules stated in the syllabus of the course regarding the academic integrity.

- At the top of each source code file, add the following information:

```
// Name of Author(s):
// ASU ID:
// Homework Number:
// Description:
```

4. Sample Test Cases

Below please find 4 test cases and their results for this program (user input is in **bold**)

Test case #1

```
String #1: 11011010111010000111101010111110000 1
String #2: 00000100010000111101000101010101001

The LCS is: 0100010000111110101011110 01
Length = 26
```

(Below your program should show the table C (length computation table) and B (arrow table)

Test case #2

```
String #1: ATCCGACAAC
String #2: ATCGCATCTT

The LCS is: ATCGCAC
Length = 7
```

(Below your program should show the table C (length computation table) and B (arrow table)

Test case #3

```
String #1: AACGTTCOGMA
String #2: GGATACCASAT

Input contains error
Error in String #1:   aacgttcOgMa
Error in String #2:   ggataccaSat
```

Test case #4

```
String #1: AGGTCCAACTTGCCTTAAATTTCCCACGGT
String #2: GGGCCTTAACTTTTAACGCGGGCCTA

The LCS is: GGCCAACTTTTAACCCCT
Length = 18
```

(Below your program should show the table C (length computation table) and B (arrow table)

What to turn in

Under the submission link on Blackboard, submit a zip file named ***youFirstNameyourLastName.zip*** containing the following:

1. One or more files of your well documented C++ source code (with a file extension *.cpp* or *.h*) implementing the commands required.

2. A Makefile that compiles your program to an executable named "***A6p1***" on the Linux machine *general.asu.edu*. Our TA will write a script to compile and run all student submissions on *general.asu.edu*; therefore executing the make command must produce the executable file "***A6p1***" in the same folder as your source code files. After such ***A6p1*** file is produced, then we should be able to execute your program using the command:

```
./A6p1 < sampleinput.txt
```

where "*sampleinput.txt*" is a file name which contains the test cases.

Your program must read from a keyboard, not a file.

Error Handling

Your program will be tested with other input besides the sample input given, thus is expected to be robust.

Grading Rubric

- 2 pts – Documentation (correct and complete code description and comments, header for each function/method, each file should have your name at its top) and good indentation and spacing (easy to read)
- 1 pt – makefile is submitted and it works correctly to generate A5p1 executable file
- 12 pts – Each of the following functions are defined correctly: *functions.h, functions.cpp and main.cpp*.
- 15 pts – Correct output for the test case(s)

Total points: 30