Data Structures and Algorithms LAB – Spring 2026

(BS-IT-F24 Morning & Afternoon)

Lab # 2 *(Ungraded Lab)*

Instructions:

- *Although, this is an **Ungraded Lab**, but I will assume in the upcoming Labs, Assignments, Quizzes, and Exams that you have solved all of these tasks.*
- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

Task # 1

Develop a C++ class **`Student`** that has private member variables to store following attributes:

- **Roll Number**: An integer variable that holds the student's roll number
- **Name**: A c-string that holds the student's name (assume that max length of name is 40) (*Note: Declare the size of the c-string as a named constant before the start of class declaration*)
- **Number of Quizzes**: An integer to store the number of quizzes taken by the student (*Note: Number of quizzes can be different for each student*)
- **Marks**: An **`int*`** through which you will dynamically allocate an array (according to the number of quizzes taken by the student) to store the marks obtained by the student in different quizzes
- **Total Marks**: An **`int*`** through which you will dynamically allocate an array (according to the number of quizzes taken by the student) to store the total marks for each quiz that a student has taken

Now, carry out the following tasks in the given order:

1. Implement a **Default Constructor** for **`Student`** class in which roll number should be initialized to 0, name should be initialized to an empty c-string, number of quizzes should be initialized to 0, and the pointers marks and total marks should be initialized as NULL pointers.

2. Implement an **Overloaded Constructor** for **`Student`** class that accepts 3 arguments: student's roll number, student's name, and number of quizzes taken by the student. The values supplied in the arguments should be used to initialize the corresponding member variables, and dynamically allocate the arrays for storing the **marks** and **total marks** of the quizzes. All elements of these two dynamically allocated arrays should be initialized to 0.

3. Implement the **Destructor** for **`Student`** class which should deallocate all dynamically allocated memory (if any).

4. Implement a public member function **`getInputFromUser`** of the **`Student`** class, which should, firstly, ask the user to enter the following 3 attributes: Roll No., Name, and No. of Quizzes taken by the student. After storing this data in appropriate member variables, this function should allocate the arrays Marks and Total Marks. If these arrays have been previously allocated make sure to deallocate them first (see the description of Overloaded Constructor above in **step 2**). After that, this function should call the private member function **`inputMarks`** (see **step 5**) to take the **Total marks** and **Obtained marks** of each quiz from the user.

5. Implement a **private** member function **`inputMarks`** of the **`Student`** class, which should ask the user to enter the **Total marks** and **Obtained marks** of all quizzes (one-by-one) and store these marks in the arrays inside the calling Student object. Also perform **input validation** on the quiz marks entered by the user. Marks of each quiz should NOT be negative, or greater than the total marks of the quiz.

6. Implement a public member function **`display()`** of the **`Student`** class which should display the roll number, name, marks obtained by the student in different quizzes, **highest percentage marks** obtained in a quiz, and the **lowest percentage marks** obtained in a quiz taken by the student.

7. Implement the **Copy Constructor** for the **`Student`** class.

8. Implement a **global function** **`void printStudent (Student)`**. This function should take a Student object which is passed by value into it. This function should display the details of that Student object on screen by calling its member function **`display()`**. There will be only a single statement in the body of this function ☺. The purpose of this function is to test the implementation of the ***Copy Constructor*** which you implemented above in **step 7**.

9. Implement a public member function **`storeInFile(ofstream&)`** of the **`Student`** class that stores all information of the Student object in the text file which has been opened through the file handle which is passed into this function.

10. Write a driver program which should take the number of students (**`n`**) from the user and dynamically allocate an array of **`Student`**s of size **`n`**. Then, it should ask the user to enter all details for each student. After that, all the Student objects from the array should be stored in a text file by using the member function **`storeInFile(…)`** that you implemented above in **step 9**. Decide about the format of the file yourself, but make sure that all necessary information is stored in the file in a manner so that it can be read later on (see the next step). At the end, your program should properly deallocate all the dynamically allocated memory.

11. Implement a public member function **`readFromFile(ifstream&)`** of the **`Student`** class that reads all information of a Student from the text file (which has been opened through the file handle which is passed into this function) and stores this information in the member variables of the Student object on which this function has been called.

12. Now, write a driver program which should open the text file created in **step 10** and dynamically allocates an array of **`Student`**s. After that, your program should read the details of all the **`Student`** objects (present in the text file) into the dynamically allocated array. Here, you will be using the function **`readFromFile(…)`** of the **`Student`** class that you implemented above in **step 11**. After reading all objects from the file, your program should display the details of all Student objects on screen (using the **`display()`** function on each Student object). At the end, your program should properly deallocate all the dynamically allocated memory.

13. Implement the **Overloaded Assignment operator** for the **`Student`** class, and write a driver program to test the working of the Assignment operator.

Task # 2

A ***ragged array*** is a 2-D array which contains a varying number of elements in each row. The ***Pascal triangle*** can be used to compute the coefficients of the terms in the expansion of $(a+b)^n$. In this task, you are required to write a C++ program to create a ragged array representing the Pascal triangle. The Pascal triangle of size **7** is shown below. Note that the 1st row contains 1 element, 2nd row contains 2 elements, 3rd row contains 3 elements, and so on. Also note that all elements in the first column are 1's. Similarly, all the diagonal elements are also 1's. Each of the other elements in the Pascal triangle is the sum of the element directly above it and the element to the left of the element directly above it. For example, the value 10 (highlighted in *Yellow*) is the sum of the elements 6 and 4 (highlighted in *Blue*).

1						
1	1					
1	2	1				
1	3	3	1			
1	4	6	4	1		
1	5	10	10	5	1	
1	6	15	20	15	6	1

You are required to implement the following functions:

Task # 2.1 `int** createPascalTriangle (int n);`

This function will take an integer (**`n`**) as argument and create a Pascal triangle consisting of **`n`** rows. It will dynamically allocate the two-dimensional *ragged array*, fill up its elements, and return a pointer to this filled ragged array (Pascal triangle).

Task # 2.2 `void displayPascalTriangle (int** pt, int n);`

This function will take a pointer **`pt`** which is pointing to a Pascal triangle consisting of **`n`** rows. It will display the Pascal triangle on screen.

Task # 2.3 `void deallocatePascalTriangle (int** pt, int n);`

This function will take a pointer **`pt`** which is pointing to a Pascal triangle consisting of **`n`** rows. This function will deallocate the two-dimensional array containing the Pascal triangle.

Task # 2.4

Write a **`main`** function which asks the user to specify the value of **`n`**. After that, the main should call the above functions to create a Pascal triangle of size **`n`**, display it on screen, and, finally, deallocate all the dynamically allocated memory.

Task # 2.5

There is one problem with the function **`deallocatePascalTriangle`** that you have implemented above in ***Task # 2.3***. Can you see the problem?

Once you have figured out the problem, change the prototype and implementation of the function to fix this problem.

Task # 3

Assume that you have a text file **`phrases.txt`** which contains several phrases (one on each line). The first line in the file indicates the number of phrases present in the file. Write a C++ program which should read all phrases from the file into a dynamically allocated **array of strings**. After that your program should traverse this array to determine the **longest phrase** and the **shortest phrase**. See the following example:

If the file "phrases.txt" contains:

```
6
Wild Goose Chase
Break The Ice
Deafening Silence
On the Same Page
In a Pickle
Down To The Wire
```

Then, a sample run of your program will look like:

```
The input file contains 6 phrases

Longest phrase:
      Deafening Silence (Length: 17)

Shortest phrase:
      In a Pickle       (Length: 11)
```

Task # 4

Write a function that copies the contents of a one-dimensional array of **`n`** elements into a two-dimensional array of **`r`** rows and **`c`** columns. The resulting 2-D array must be allocated dynamically. The data will be inserted into this 2-D array in ***row major order***; that is, the first **`c`** elements will be placed in row 0, the next **`c`** elements in row 1, and so forth until all rows have been filled.

If **`n ≠ r * c`** , the function returns a NULL pointer. Otherwise, it returns a pointer to the dynamically allocated (and filled) 2-D array. A pointer to the 1-D array, and integers **`r`**, **`c`**, and **`n`** will be passed as parameters into this function.

Also write a driver main program to test the working of your function.

VERY IMPORTANT

In the next Lab, you will need some or all of the functions from Today's Lab. So, make sure that you have the working implementation of **ALL** the functions of Today's Lab, when you come to the next Lab.