Friday, May 20, 2022.

Data Structures and Algorithms LAB – Spring 2022

(BS-IT-F20 Morning & Afternoon)

Lab # 6 (Online Lab)

Submission Deadline: ***Saturday, May 21, 2022 (till 8:00 PM)***

Instructions:

- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an **"F"** grade in this course and lab.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Do NOT use any **global** or **static** variable(s). However, global named constants may be used.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.
- Late submissions will NOT be accepted, whatever your excuse may be.

Submission Procedure:

i. There are 10 tasks in this lab. You are required to submit C++ programs for the **first 3 tasks**. The remaining tasks are for your practice and you are not required to submit them. But I will assume in quizzes and exams that you have solved all of these questions/tasks ☺

ii. Create 3 empty folders. The folder names MUST be **Task-1**, **Task-2**, and **Task-3**. Put all of the ***.CPP and .H files*** of each task in its appropriate folder. Do NOT include any other files in your submission, otherwise your submission will NOT be graded. Don't forget to mention your Name, Roll Number and Section in comments at the top of each CPP file.

iii. Now, put all the folders created in the previous step into another folder.

iv. Now, compress the folder (that you created in previous step) in **.RAR** or **.ZIP** format. Then rename the RAR or ZIP file ***exactly*** according to the following format:

`Mor-Lab6-BITF20M123` (for Morning section) OR

`Aft-Lab6-BITF20A456` (for Afternoon section),

where the text shown in **BLUE** should be your **complete Roll Number**.

v. Finally, submit the *single* **.RAR** or **.ZIP** file through **Google Classroom**. Make sure that you press the **SUBMIT** button after uploading your file.

Note: If you do not follow the above submission procedure, your Lab will NOT be graded and you will be awarded ZERO marks.

Task # 1

As discussed in the lecture, in Queue data structure, elements are inserted at the **back** end and removed from the **front** end of the queue. A **Double Ended Queue** (Deque) (pronounced *"deck"*) is a queue in which you can insert and remove elements **both at the *front* and at the *back* of the queue**. In this task, you are required to implement a class **`Deque`** (which will be used to store integers). The **`Deque`** class should have the following public member functions:

- **`Deque (int n)`** Constructor to create a Deque which can contain up to **`n`** integers.
- **`~Deque ()`** Destructor to destroy the Deque.
- **`bool isEmpty ()`** returns true if the Deque is empty, and false otherwise.
- **`bool isFull ()`** returns true if the Deque is full, and false otherwise.
- **`void display ()`** to display the elements of Deque **from front to back**.
- **`void makeEmpty ()`** to make the Deque empty.
- **`bool insertAtFront (int val)`** to insert **`val`** at the front of the queue.
- **`bool insertAtBack (int val)`** to insert **`val`** at the back of the queue.
- **`bool removeFromFront (int& val)`** to remove an integer from the front of the queue and store it in the variable **`val`**.
- **`bool removeFromBack (int& val)`** to remove an integer from the back of the queue and store it in the variable **`val`**.

Note that:

- The **time complexity** of **all member functions** (except the **`display()`** function) of the **`Deque`** class should be **constant i.e. *O*(1)**.
- Your **`Deque`** class should refuse the insertion of a new element ONLY when there are no empty slots left in the **`Deque`**. So, make sure that all the boundary cases are properly handled.

You are also required to write a menu-driven program which allows the user to use all of the aforementioned functionalities of the **`Deque`** class. A sample menu is shown below (Text shown in **Red** is entered by the user):

```
Enter the size of Deque: 10
1. Insert value at Front
2. Insert value at Back
3. Remove value from Front
4. Remove value from Back
5. Display the Deque
6. Make the Deque Empty
7. Exit
```

Note: *Following tasks require you to write* ***recursive*** *functions. NO credit will be given for iterative (loop-based) implementation. Also write a* ***main*** *function to test the working of each function that you implement. In these recursive functions, you are* ***NOT*** *allowed to declare/use any* ***global or static variables***

Task # 2

Implement a **recursive** C++ function which takes an array of integers (**arr**) and the starting (**start**) and ending (**end**) indices of a *portion* (part) of this array, and returns the **index of the largest element** present in that portion of the array **arr**. The prototype of your function should be:

```
int findLargestIndex (int* arr, int start, int end)
```

For example, the function call **findLargestIndex(arr,3,8)** should determine and return the ***index*** of the largest element present in the array **arr** between the indices **3** and **8** (both inclusive).

Task # 3

3.1 Implement a **recursive** C++ function which takes an integer array (**arr**) and the starting (**start**) and ending (**end**) indices of that array, and prints all the values present in that array in a zig zag way i.e. initially the first and the last elements will be displayed, then the second and the second-last elements will be displayed, and so on. For example:

If the array contains **{3,6,2,1,4}** then the output should be **3 4 6 1 2**

If the array contains **{7,1,9,3,5,2}** then the output should be **7 2 1 5 9 3**

The prototype of your function must be:

```
void printZigZag (int* arr, int start, int end)
```

3.2 Implement a modified version of printZigZag which takes as arguments the **pointers** to the first and the last element of an integer array. The prototype of this function should be:

```
void printZigZagUpdated (int* pStart, int* pEnd)
```

Task # 4

Implement a **recursive** C++ function which takes two integers **a** and **b** as arguments and returns the value of a^b. The prototype of your function should be:

```
int power (int a, int b)
```

Task # 5

Implement a **recursive** C++ function which takes two integers **`a`** and **`b`** as arguments and returns their **product**. The prototype of your function should be:

```
int product (int a, int b)
```

Hint: Think about repeated addition.

Task # 6

6.1 Implement a **recursive** C++ function which takes a character (**`ch`**) and a positive integer (**`n`**) and prints the character **`ch`**, **`n`** times on the screen. The prototype of your function should be:

```
void printChar (char ch, int n)
```

For example, calling **`printChar('Z',5)`** should display **`ZZZZZ`** on screen.

Note**:** You are NOT allowed to use any loop in your function.

6.2 Now, implement a **recursive** C++ function which takes a character (**`ch`**) and a positive integer (**`n`**) and, and prints a pattern on screen (see below). For example, it should display the following pattern when the arguments are **`'*'`** and **`4`**, respectively:

```
****
***
**
*
```

The prototype of your function should be:

```
void printPattern1 (char ch, int n)
```

Hint**:** Use the function **`printChar`** that you implemented above in **Task # 6.1**.

6.3 Implement another **recursive** C++ function which prints the following pattern on screen when its arguments are **`'#'`** and **`6`**, respectively:

```
#
##
###
####
#####
######
```

The prototype of your function should be:

```
void printPattern2 (char ch, int n)
```

Hint**:** Use the function **`printChar`** that you implemented above in **Task # 6.1**.

Task # 7

Implement a **recursive** C++ function which takes two integers `num` and `den` as arguments and returns the **integer quotient** that will result when `num` is divided by `den`. The prototype of your function should be:

```
int quotient (int num, int den)
```

***Hint*:** Think about repeated subtraction.

Task # 8

Implement a **recursive** C++ function which takes an array of integers (`arr`) and the starting (`start`) and ending (`end`) indices of a *portion* (part) of this array, and returns the **largest element** present in that portion of the array `arr`. The prototype of your function should be:

```
int findLargest (int* arr, int start, int end)
```

For example, the function call `findLargest(arr,3,6)` should determine and return the largest element present in the array `arr` between the indices `3` and `6` (both inclusive).

Task # 9

Implement a **recursive** C++ function which takes two integer arrays and their sizes as arguments, and determines whether these two arrays are equal or not. Note that two arrays are equal if they contain the same number of elements and the elements of both arrays occur in the same order. The prototype of your function should be:

```
bool areArraysEqual (int* a, int aSize, int* b, int bSize)
```

Task # 10

Implement a **recursive** C++ function which takes a c-string and its length as parameters and returns the number of vowels present in that c-string. The prototype of your function should be:

```
int countVowels (char* str, int length)
```

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 tasks** and **ALL functions** of Today's Lab, before the start of next lab (**Lab # 7**).