Lab # 2

Object Oriented Programming

BS SE F17 Morning

Instructions:

• **Attempt the following tasks exactly in the given order.**

• Indent your code properly.

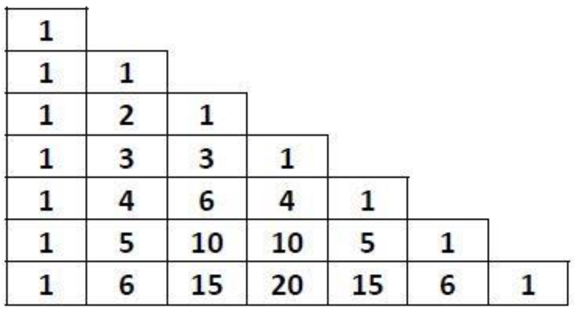
• Use meaningful variable and function names. Follow the naming conventions.

• Use meaningful prompt lines and labels for all input/output.

• Make sure that there are **NO dangling pointers** or **memory leaks** in your program.

Task # 1

A *ragged array* is an 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 a Pascal triangle, each element is the sum of the element directly above it (if any) and the element to the left of the element directly above it (if any). A Pascal triangle of size 7 is shown below:



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.

<u>Task 1.1</u>

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 array (Pascal triangle).

<u>Task 1.2</u>

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.

<u>Task 1.3</u>

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.

<u>Task 1.4</u>

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 # 2.1
Write a function that copies a one-dimensional array of **n** integers into a 2-D array of **r** rows and **c** columns. The resulting 2-D array should be allocated dynamically. The data will be inserted into the 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. The *input array* and the integers **n**, **r**, and **c** will be passed as parameters into this function. If ***n ≠ r*c***, the function returns a NULL pointer. Otherwise, it returns the pointer to the newly created 2-D array.

Task # 2.2
Modify the function written in **Task # 2.1**, so that it copies the data into the 2-D array in *column major order* i.e. the first **r** elements will be placed in column 0, the next **r** elements in column 1, and so forth.

Task # 3

```
struct Book
{
	char name[50];
	char publisher[50];
	int ISBNNumber;
};
```
	```
	struct Library
	{
		Book * data;
		int noOfBooks;
		int capacity;
	};
	```

Struct Description:
Book:
It will store the item information which user wants to buy.

- 'name' attribute will store book name
- 'publisher' will store name of the publisher.
- 'ISBNNumber' will store the ISBN number of book.

Library:
It will store/contain all available books.

- 'data' attribute will point to an array of 'Book' on heap, which will contain the book available at library.
- 'noOfBooks' represents the total no of Books available in library.
- 'capacity' represents the size of library i.e. the size of array pointed by 'data'.

Operations:

- void initializeLibrary(Library & ly, const int capacity);
- bool addBook(Library & ly, const Book & bo);
- bool removeBook (Library& ly, char * BookName, int count = 1);
- void DisplayAllBooksInformation(Library & ly);
- void resize(Library& ly , int s=5);