NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES
ISLAMABAD CAMPUS
OBJECT ORIENTED PROGRAMMING (CS103) - SPRING 2020
ASSIGNMENT-2

Due Date: FEB 21, 2020(5:00 PM)
Instructions:

• Make sure that you read and understand each and every instruction. If you have any questions or Comments you are encouraged to discuss with your colleagues and instructors on Piazza (https://piazza.com/nu.edu.pk/spring2020/cs217/home).
• Create each problem solution in a separate .cpp file, i.e. you must name the file containing solution of
Q1 as 'q1.cpp', Q2 as 'q2.cpp' and Q3 as 'q3.cpp'.
– Combine all your work in one .zip file.
– Name the .zip file as ROLL-NUM SECTION.zip (e.g. 19i-0001 B.zip).
– Submit the .zip file on Google Classroom within the deadline.
• Start early otherwise you will struggle with the assignment.
• You must follow the submission instructions to the letter, as failing to do so will get you a zero in the assignment.
• All the submitted evaluation instruments (quizzes, assignments, lab work, exams, and the project) will be checked for plagiarism. If found plagiarized, both the involved parties will be awarded zero marks in the relevant evaluation instrument, all of the instruments, or even an F grade in the course.

Q1: Write a recursive method named **Combination** that accepts two integers' n and r as parameters and returns the number of unique combinations of r items from a group of n items. For given values of n and r, this value **c(n,r)** can be computed as follows:
C(n,r) = n! / r! * (n - r)!
For example **Combination (4,3)=4!/3!*(4-3)!** should return 4. Your function prototype must be as follows:
https://study.com/academy/lesson/how-to-calculate-the-probability-of-combinations.html
long Combination (int n, int r)

Q2: Write a recursive function that takes three square matrix and checks whether the values are equal or not. It returns true if they are equal else false.

bool equal(int matrixOne, int** matrixTwo,int** matrixThree, int row, int column)**

Q3: Write a program that takes a pointer to array of integers. Create a dynamic array of integers through **CreateArray()** function (without recursion).After that pass it to the sum function to calculate sum of perfect numbers using recursion.

NOTE: A perfect number is a number that is numerically equal to the sum of its divisors. For example, 6 is a perfect number because the divisors of 6 are 1,2,3 and 1+2+3=6.
int sum(int *array, int size)

Q4: Draw the following patterns using the *recursion* mechanism.**(You have to run the code on terminal to see the output pattern.)**

NOTE:The solution should be generic (for input N) and should not contain any "Loops" and for the test run it on terminal with n number of values.

(A) Arrow Shape (with size N)

```
Input : n = 5
Output :

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

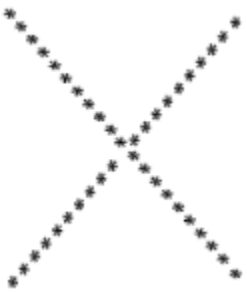
```
Input : n = 7
Output :

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

(B) Rhombus Star pattern: (stars are separated by a space character):
Input: N (number of lines)

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

(C) X start pattern:
Input: N (number of symbols in a line)



Q5: Consider the following structure definitions:

```
struct name
{
  char  F_Name[20];
  char L_Name[20];
};
```

```
struct student
{
      string reg_no;
      name  student_name;
      int marks[5];
      float GPA;
};
```

Ask the user to enter number of students and then create a dynamic array of students. Read all of the above data for all the students except GPA. For each student calculate GPA according to his/her average marks (considering the grading scheme mentioned below).

For example :A student who obtained marks in five subjects as follows "60, 80, 90, 50, 60" will result in average marks of 68 and therefore his/her GPA will be "2.87"

GPA	Marks (Percentage)
4.00	90-100
3.62	80-89
3.10	70-79
2.87	60-69
1.80	50-59
0.00	Below 50

(1)After calculating GPA for each student, display first names (*f_name*) and registration number (*reg_no*) of all those students whose GPA is above 3.0

(2)Sort data of all students (for example using bubble sort) so that students getting a higher GPA are stored first as compared to the students getting a lower GPA.

https://www.techopedia.com/definition/3757/bubble-sort

Q6: Consider the following structure definitions:

```
struct Employee
{
  char  name[20];
  int scale;
  Salary YearlySalary[3];
  Tax YearlyTax[3];
  Address addr;
  Employee* next;
};
```

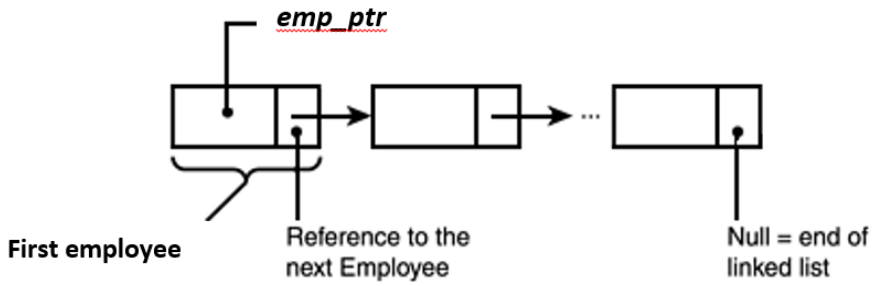
```
struct Address
{
  char  city[20];
  int postcode;
  char email_id[20];
};
```

```
struct Salary
{
  int Year;
  float gross_salary;
};
```

```
struct Tax
{
  int Year;
  float tax_due;
  float net_salary;
};
```

Write a C++ program that asks the user to enter data for 10 employees. The following information should be entered by the user: *name*, *scale* (1 to 17), gross *salary* for the last 3 years, and *address*. The program should calculate the yearly tax (5% of gross salary for scale 1 to 10 employees and 7.5% for scale 11 to 17) and net_salary (gross_salary – tax_due).

For an employee, the structure variable should be allocated ***dynamically on heap memory*** using **new** operator. A pointer ***emp_ptr*** (of employee type) should be used to point to the first employee (initially ***emp_ptr*** will be NULL). The second employee should be referred by the first employee (**Employee* next** structure member). This will result in a chain of records in the heap (linked with each other using **next** pointer) as shown below.



Next, find the "average" net_salary for all the 10 employees. Also, display the name and address of the employee who has paid the highest tax (for any year).

Q7: Write a program that stores the following data about a Soccer player in a structure:

```
struct SoccerPlayer
 { char PlayerName[20];
     int Player_Number;
     int Points_Scored;
};
```

The program should keep a dynamic array of these structures. Each element is for a different player of the team. When the program runs it should display a menu asking the user to enter the data. The menu should provide the following options:

I) Add new players' information: *It will add information of a new player.*

2) Insert a new player: *It will insert information at the desired location.(for example at index 2).*

3) Delete the player's information *(Example: Delete player at index = 5)*

4) Display Player information: *Display information in a tabular format (for all players)*

5) Display: *The number and name of the player who has earned the most points should also be displayed.*

Input Validation: Do not accept negative values for players' numbers or points scored.

Q8: Write a program that uses these structure to store the following weather data for a particular month:

```
struct Temperature{
{
double max_temperature;
double min_temperature;
};
```

```
struct Rainfall{
int max_rainfall;
int min_rainfall;
};
```

```
   struct weather{
Rainfall total_rain;
Tempearture temp;
};
```

The program should have an array of 12 structures to hold weather data for an entire year. When the program runs, it should ask the user to enter data for each month. (You are required to calculate average temperature and rain fall for the year). The program then should display month wise average rain fall and temperature similarly it should display average rain fall and temperature of the entire year.

Input Validation: Only accept temperatures within the range between -100 and +140 degrees Fahrenheit and rain fall 1 mm to 25mm.