Quiz/Homework 2

The questions are designed by the TAs or me. For information on how to contact the TAs, please refer to the course outline.

Due Date: **From** Monday February 26th to March 1st during your labs sessions.

Instructions

- Please ensure that you complete all questions and present your answers to your TAs during **one** of your **assigned** weekly lab sessions between Due Date. TAs may ask you questions and request you to work with your codes or present it. TAs will provide feedback and your grade right away. Grades will also be released on Avenue a few days later.
- The purpose of these Quiz/Homework assignments is not solely to test your programming skills. Instead, they are designed to provide you with simple and basic examples to better prepare you for your assignments. Consequently, you can work on the questions outside of the lab sessions. You may ask questions from your TAs at any point over Teams or during the lab sessions.
- Please attend only the labs for which you are enrolled to prevent classes from becoming overcrowded.
- You don't need to submit anything. Just keep your files organized to be presentable if the TAs asked you to work with your code.

1. **Find First Occurrence (Patrick, 0.5 points)**

   Given the array of random numbers

   ```
   int nums[] = {10,15,6,11,17,8,10,29,12,9};
   ```

   Write a C program that finds the first occurrence of a number from an array. Create a function called `int findFirstOccurence(int number)` that takes a number as input and returns the first occurrence of that number in the array (if it exists). Print the number and index to the command console. Store the program in a file named `q1.c`.

2. **Determine Ticket Discounts with If-Else Statements (Renjie, 0.5 points)**

   Write a C program that determines ticket discounts based on the user's age and student status:

   (a) Ask the user for their age and whether they are a student (**1** for yes, **0** for no).

(b) Apply the following rules to determine the discount:

- A student should be over 5 (> 5) and under 30 (< 30).
- If the person is under 18 or a student, they get a 30% discount.
- If the person is over 65 and under 120, they get a 50% discount.
- If the person is exactly 18 and not a student, they get a 20% discount.
- If the person is exactly 25 and a student, they get a 40% discount.
- If none of the above condition applies, there is no discount.
- If eligible for multiple discounts, the highest discount is applied.

(c) Include proper checks to ensure all the inputs are valid.

(d) Display the final discount to the user.

3. **Determine Ticket Discounts with Switch Statements (Renjie, 0.5 points)**

Write a C program that determines ticket discounts based on the user's age and discount code:

(a) Ask the user for their age.

(b) Ask if the user has a discount code (**1** for yes, **0** for no). If yes, ask for the code (**A**, **B**, **C**).

(c) Use a `switch` statement to categorize the user into an age group:

- 0-12 years: Child
- 13-19 years: Teen
- 20-64 years: Adult
- 65-120 years: Elder

(d) Apply the following discounts based on the age group:

- Child: 50% discount
- Teen: 25% discount
- Adult: No discount
- Elder: 30% discount

(e) If the user has a discount code, apply an extra discount:

- Code **A**: 5% extra discount
- Code **B**: 10% extra discount
- Code **C**: 15% extra discount

(f) Include proper checks to ensure all the inputs are valid.

(g) Calculate the final ticket discount after applying all discounts and display it to the user.

4. **Is it a Leap Year? (Patrick, 0.5 points)**

   Using your extensive knowledge of leap years (Hint: https://www.mathsisfun.com/leap-years.html), write a C program that determines if a given year is a leap year. Create a function called `bool leapYear(int year)` that takes an input year and returns `true` if it is a leap year. Make the code give a command line input for a year and print out if the year is a leap year or not. Store the program in a file named `q4.c`.

5. **Grade Analyzer (Nishant, 0.5 points)**

   Develop a C program to analyze student grades. The program uses an array to store the grades of five students. Write a C program that includes the following features:

   - Implement a function `inputGrades()` that takes an array as a parameter and prompts the user to input grades for five students. Ensure that the grades are between 0 and 100.
   - Implement a function `analyzeGrades()` that takes the array of grades as a parameter and calculates and prints the average grade, the highest grade, and the lowest grade.
   - In the main function, declare an array to store the grades of five students.
   - Call the `inputGrades()` function to input grades.
   - Call the `analyzeGrades()` function to analyze and display the statistics.

6. **Ticket Booking System (Nishant, 0.5 points)**

   Develop a simple ticket booking system for a theater using C. The theater has multiple rows, and each row has a different number of seats. Write a C program that uses functions, arrays, loops, and nested loops to achieve the following:

   - **Initialization:** Use global variables to define the number of rows = 5 and the maximum number of seats in each row = 10.
   - Create a function `void initializeSeatingChart()` which uses a `2D array` to represent the seating arrangement, initializing all seats as available ('O').
   - **Display Seating Chart:** Write a function `void displaySeatingChart()` to display the current seating chart. Seats should be represented by 'O' for available and 'X' for booked.
   - Display the row and seat numbers for easy identification.
   - **Seat Booking:** Implement a function `int bookSeat(int row, int seat)` to book a seat. Prompt the user to enter the row and seat numbers they want to book.
   - Check if the seat is available. If so, book the seat by updating the array, and display a success message.
   - If the seat is already booked or the input is invalid, display an appropriate error message.
   - **Display Total Booked Seats:** Write a function `void displayTotalBookedSeats()` to display the total number of booked seats.

   Example test scenario and output:

```c
int main()
{
    initializeSeatingChart();

    bookSeat(2, 3);
    bookSeat(3, 7);
    bookSeat(1, 5);

    displaySeatingChart();
    displayTotalBookedSeats();

    return 0;
}
```

```
Seat booked successfully!

Seat booked successfully!

Seat booked successfully!

Seating Chart:
Row 1: O O O O X O O O O O
Row 2: O O X O O O O O O O
Row 3: O O O O O O X O O O
Row 4: O O O O O O O O O O
Row 5: O O O O O O O O O O

Total Booked Seats: 3
```

7. **Fibonacci Sequence (Pedram, 0.5 points)**

During out lecture we talked about a Recursive function example computing the factorial of n ($n! = n \times (n-1)!$). Another classic example of a recursive function is calculating the nth term of the Fibonacci sequence. The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones. The sequence starts with 0 and 1. The Fibonacci numbers may be defined by the recurrence relation:

$$F_0 = 0, \text{ and } F_1 = 1$$

and

$$F_n = F_{n-1} + F_{n-2} \text{ for } n > 1.$$

Write the C code to take the input `n` from the user by `int main(int argc, char *argv[])`, and computes the $n-th$ number of Fibonacci.

Reminder:

- The input received through the `int main(int argc, char *argv[])` function parameters is typically referred to as command-line arguments. These are values provided by the user when running the program from the command line.
- The input received through `scanf` is typically referred to as standard input (`stdin`) or console input. This input method allows the user to input data during the program's execution via the console or terminal.

So do not use `scanf`. Keep the codes in `q7.c`.

8. **The differences between algorithms doing the same task (Pedram, 0.5 points)**

Take a look at the following functions.

```
// Function to calculate factorial iteratively
unsigned long long factorialIterative(int n) {
 unsigned long long result = 1;
 for (int i = 1; i <= n; ++i) {
  result *= i;
 }
 return result;
}

// Function to calculate factorial recursively
unsigned long long factorialRecursive(int n) {
```

```
    if (n == 0)
    return 1;
    else
    return n * factorialRecursive(n - 1);
}
```

They are two different algorithm to compute the factorial of n ($n! = n \times (n-1)!$), one with Recursive and the other with Iterative method. Write a C code which in which for inputs of `n` it can compute $n!$ using both functions. Use `<time.h>` library to measure the time taken by each function and print the time in milliseconds for each function. Also print the value of computer $n!$ with different functions. For example you should see in the terminal every time:

```
Factorial of 20 (Iterative): 2432902008176640000
Time taken (Iterative): 0.006000 milliseconds
Factorial of 20 (Recursive): 2432902008176640000
Time taken (Recursive): 0.004000 milliseconds
```

This is the result in my computer for 20! (`n=20`).

Keep your results in a q8.txt file for:

- `n=65`
- `n=66`
- `n=100000`
- `n=500000`

a) What is the highest value of `n` you can go to have a correct result for the factorial of n? (on your OS)

b) Why after that value the results are wrong?

c) What happens if we would change all `unsigned long long` to `unsigned long`? Do we get the same outputs?

d) Explain for `n=500000` **what**, and **why** it happens!?

e) Which one is faster, Recursive or Iterative method?

Keep the codes in `q8.c`.

9. **Practice with 2D arrays and basic array operations (Pedram, 0.5 points)**

Write a C program that performs various operations on a 2D array of double precision floating-point numbers.

The program should include the following functionalities:

1. Generation of a random 2D array of size n x m, where n and m are number of rows and columns and they are hard-code integers (`int m = 150` and `nt n = 150`).

2. Functions to find the maximum value in a given column (`column_s`), row (`row_s`), or the entire matrix.

3. Functions to find the average value in a given column (`column_s`), row (`row_s`), or the entire matrix.

4. Your program should not use pointers.

This is the general format that you code should look like:

```c
// Code here: include necessary library(s)

void generateRandomArray(int rows, int cols,double matrix[rows][cols]) {
 // Code here: the code to create a random double precision from 0 to 100
 // for each element in matrix
}

double findMaxInColumn(int row, int col, double matrix[row][col], int column_s) {
 // Code here: find the maximum in column index column_s
}

double findMaxInRow(int row, int col, double matrix[row][col], int row_s) {
 // Code here: find the minimum in row index row_s
}

double findMaxInMatrix(int row, int col, double matrix[row][col]) {
 // Code here: find the maximum in the whole matrix
}

double findAverageInColumn(int row, int col, double matrix[row][col], int column_s) {
 // Code here: find the average in the column index column_s
}

double findAverageInRow(int row, int col, double matrix[row][col], int row_s) {
 // Code here: find the average in the row index row_s
}

double findAverageInMatrix(int row, int col, double matrix[row][col]) {
 // Code here: find average in the whole matrix
}

int main() {
 int m = 150;
 int n = 150;
 double arr[m][n];

 // column_s is the specified column for average and max
 int column_s = 8;
 // row_s is the specified row for average and max
 int row_s = 5;
```

```
    // Code here: Call the functions and do th prints
}
```

This is what I get when `int m = 150`, `int n = 150`:

```
Maximum value in column 8: 86.317969
Maximum value in row 5: 85.047695
Maximum value in the entire matrix: 99.042461
Average value in column 8: 44.251738
Average value in row 5: 44.834173
Average value in the entire matrix: 51.814747
```

a) Explain what happens when `int m = 3000` and `int n = 3000`?

Keep the codes in `q9.c`.