NATIONAL INSTITUTE OF TECHNOLOGY SIKKIM
Department of Computer Science and Engineering

CS13201 Data Structure and Algorithms Laboratory

Odd Semester, July–December 2024

Laboratory Assignment #2

NB: Solve the following programming problems according to instructions given during the laboratory sessions. Along with the requirements mentioned in the text of the questions, additional instructions may be given by the course instructor to produce the desired output. All programs should be written in the C programming language. Do not use the *string.h* header.

1. Write a program to partition an array of integers into two subarrays by rearranging its elements in such a way that the first subarray contains odd numbers and the second subarray contains even numbers, without using a second array, and without changing the relative ordering among the odd numbers and the relative ordering among the even numbers. Take the numbers of the array as input from the user, rearrange the numbers of the array to partition it, and then display the modified array as the final output of your program. For example, the array {17, 14, 15, 10, 25, 40, 56, 46, 98, 77} will be partitioned and modified to {17, 15, 25, 77, 14, 10, 40, 56, 46, 98}.

2. Similar to Question no. 1 above, write a program to partition an array of integers into two subarrays of prime numbers and composite numbers. Take the array as input from the user, rearrange its elements, and display the modified array as the final output of your program. For example, the array {17, 14, 15, 7, 9, 31, 67, 89, 90, 19} will be partitioned and modified to {17, 7, 31, 67, 89, 19, 14, 15, 9, 90}.

3. Write a program to find the product of two matrices. Take two matrices as input from the user in row-major order, check whether the two matrices are compatible for multiplication, and then, if they are compatible, multiply them and create the product matrix. Display the product matrix as the final output of your program in row-major order. If the matrices are not compatible for multiplication, display an appropriate message.

4. Write a program to find the sum of the elements of the principal diagonal of a matrix. Take a matrix of real numbers as input from the user in row-major order, and then print the sum of the principal diagonal elements of the matrix as the final output of your program. Also, print the elements of the principal diagonal of the matrix.

5. Write a program to store a matrix of names and then print the names contained in the principal diagonal of the matrix. Take the dimensions of the matrix as input from the user, store names in the matrix (where the matrix is implemented with a three-dimensional character array) in row-major order so that each element of the matrix is a name, and then display the list of names contained in the principal diagonal of the matrix, in the correct order from the top left element to the bottom right element of the principal diagonal, as the final output of your program.

6. Write a program to store the elements of a two-dimensional array A_2 in an array of structures S where each element of the array of structures S is a structure object containing a one-dimensional array A_1 that stores one row of the two-dimensional array A_2. To elaborate, the one-dimensional array A_1 which is a member of the structure object which is the i^{th} element of the array of structures S will store the i^{th} row of the two-dimensional array A_2. Take the elements of the two-dimensional array A_2 as input from the user in row-major order. Deciding the data type of A_2 is left to the programmer's discretion. Print the contents of the array of structures S in a reader-friendly format as the final output of your program.

7. Write a program to store the elements of the Pascal's triangle in a matrix such that the i^{th} row of the matrix stores the i^{th} row of the Pascal's triangle in the correct order. Print the matrix with appropriate spacing so that the Pascal's triangle is displayed in the conventional way that looks like an isosceles triangle. Take the number of rows of the Pascal's triangle as a command-line argument.

   Sample Execution (for showing 5 rows of Pascal's triangle)

```
$ ./pascal_triangle.out 5 ↲
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

8. Write a function called **add_to_sortedlist** which takes a sorted array of numbers called *list* and a number *num* as its arguments, and then inserts *num* at the correct position in *list* so that *list* remains sorted. In your **main** function, take a set of *n* random numbers as input from the user, and then call **add_to_sortedlist** for *n* times to store the numbers in an array *A* so that *A* always remains sorted. Print *A* in the correct order as the final output of your program. Do not use any second array.

9. Write a program to create a menu-based railway reservation system for a train where the passenger details are stored in an array of structures. Each structure instance will have at least four members: *pnr_number*, *name*, *age*, *seat_number*; more members may be defined by the programmer if required. The user interface will show a menu with the following options: 1) **Book ticket**, 2) **Cancel ticket**, 3) **Print ticket**, 4) **Print chart**, 5) **Quit**. The menu will be repeatedly displayed in a loop until the user chooses to "**Quit**". In "**Book ticket**" option, the user will be allowed to book a seat by entering her name and age if seats are available in the train (assume that the size of the array of structures is the total number of seats in the train), and will be provided with a unique ticket identification number called a PNR number, otherwise will be shown an appropriate "Not available" message. In "**Cancel ticket**" option, the user will be allowed to cancel her ticket by providing her PNR number, and the corresponding seat in the train will be marked as available again. In "**Print ticket**" option, a passenger will be able to print his ticket details by entering his PNR number. In "**Print chart**" option, a user will be able to print the details of all tickets booked for the train. Note the constraint that the same seat cannot be booked by two passengers and that a passenger whose age is above 60 years of age (called a senior citizen) must be provided a lower berth if available, where a lower berth is defined to be a seat number denoted by *num* where *num* is divisible by 3; if a lower berth is not available, then that senior citizen will be provided any other berth. The details of every ticket will be stored in an element of the array of structures declared in the program. The format of the output, satisfying the requirements specified herein, is left to the discretion of the programmer.

10. Write a program to store in an array of structures the runs of players of two teams playing a Test cricket match and then find out the result of the match. Store details of players of two teams India and Australia in two arrays of structures *I* and *A* respectively (i.e. *I* will store the details of India and *A* will store the details of Australia), where each structure instance will store (at least) three values: i) name of a player, ii) runs scored by the player in 1st innings, and iii) runs scored by the player in 2nd innings. Note the constraint that there can be only 11 players in each team. Take the aforementioned values as input from the user, and then display the result, i.e. which team beat which team by how many runs, as the final output of your program. Also display the complete scoreboard of the match elegantly, showing the runs scored by individual players and the runs scored by both teams in both innings, along with the result. The format of the output, satisfying the requirements specified herein, is left to the discretion of the programmer. [NB: The team with the higher aggregate number of runs in both innings wins the match.]

--------------------------------------x--------------------------------------