Lab Manual

Computer Engineering for Session 2019 (Semester Fall-2019)

Programing Fundamentals

(Lab 10)

More on functions & arrays

Note: It is expected that all the functions will be solved in modular fashion i.e. proper functions will be created for independent tasks. You will write no code without adapting modular approach. Also, be careful on your code indentation and formatting. Comments will come out more helpful now.

First five questions will be evaluated in lab based on the CLO1 and CLO4.

1. Write a program that calls a function for five times using loop. The function uses a static variable initialized to 0. Each time the function is called, the value of static variable is incremented by 1 and is displayed on the screen. Explore the use, advantages and disadvantages of static and global variables.
2. Write a program that inputs two integers in main() function, passes these integers to a function by reference. The function swaps the values. The main() function should display the values begore and after swapping. Explore concept of pass by value and pass by reference.
3. Write a program which tosses a biased coin and presents in the information in the form of a histogram. Biased coin has 60% chance of having a face value equal to tail and heads otherwise. Based on this, you need to write function **flipCoin** which flips the biased coin and return 1 for heads and 0 for tails. You will take **n** as input from user i.e. how many times user wants to flip the coin. After all the flips, you will print the histogram of possible values. Note, you also need to create a histogram function. Think on its attributes.
4. (Airplane Seating Assignment) Write a program that can be used to assign seats for a commercial airplane. The airplane has 13 rows, with six seats in each row. Rows 1 and 2 are first class, rows 3 through 7 are business class, and rows 8 through 13 are economy class. Your program must prompt the user to enter the following information:
   a. Ticket type (first class, business class, or economy class)
   b. Desired seat
   Output the seating plan in the following form:

	A	B	C	D	E	F
Row 1	*	*	X	*	X	X
Row 2	*	X	*	X	*	X
Row 3	*	*	X	X	*	X
Row 4	X	*	X	*	X	X
Row 5	*	X	*	X	*	*
Row 6	*	X	*	*	*	X
Row 7	X	*	*	*	X	X
Row 8	*	X	*	X	X	*
Row 9	X	*	X	X	*	X
Row 10	*	X	*	X	X	X
Row 11	*	*	X	*	X	*
Row 12	*	*	X	X	*	X
Row 13	*	*	*	*	X	*

Here, * indicates that the seat is available; X indicates that the seat is occupied. Make this a menu-driven program; show the user's choices and allow the user to make the appropriate choices.

5. Write a function dotProduct which takes two vectors as input and returns a scalar value which is equal to the dot product of the given vectors. For example, if x = {1, 2, 3} and z = {5, 7, 1} then answer will be 22.
6. (Total Sales) Use a two-dimensional array to solve the following problem. A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each different type of product sold. Each slip contains: a) The salesperson number b) The product number c) The total dollar value of that product sold that day Thus; each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all the slips for last month is available. Write a program that will read all this information for last month's sales and summarize the total sales by salesperson by product. All totals should be stored in the two-dimensional array sales. After processing all the information for last month, print the results in tabular format with each column representing a salesperson and each row representing a product. Cross total each row to get the total sales of each product for last month; cross total each column to get the total sales by salesperson for last month. Your tabular printout should include these cross totals to the right of the totaled rows and to the bottom of the totaled columns.

Homework Questions:

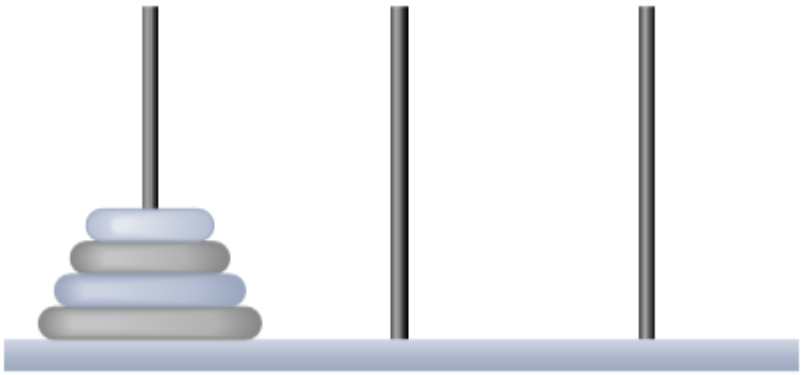
1. (**Matrix Multiplication**) A two-dimensional matrix can be multiplied by another matrix to give a matrix whose elements are the sum of the products of the elements within a row from the first matrix and the associated elements of a column of the second matrix. Both matrices should either be square matrices, or the number of columns of the first matrix should equal the number of rows of the second matrix. To calculate each element of the resultant matrix, multiply the first element of a given row from the first matrix and the first element of a given column in the second matrix, add that to the product of the second element of the same row and the second element of the same

column, and keep doing so until the last elements of the row and column have been multiplied and added to the sum. Write a program to calculate the product of 2 matrices and store the result in a third matrix.

2. **(Power set):** Write a program which takes a set as input and print its power set.

3. (**Intersection**): Write a program in C++ which takes two sets as input and returns another set which is equal to the intersection of the given sets. Also, create a function to print the set. Your code must have intersection, printing sets and input taking functions etc.

4. **(Towers of Hanoi)** Every budding computer scientist must grapple with certain classic problems, and the Towers of Hanoi (see Fig. 5.23) is one of the most famous of these. Legend has it that in a temple in the Far East, priests are attempting to move a stack of disks from one peg to another. The initial stack had 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from this peg to a second peg under the constraints that exactly one disk is moved at a time, and at no time may a larger disk be placed above a smaller disk. A third peg is available for temporarily holding the disks. Supposedly the world will end when the priests complete their task, so there's little incentive for us to facilitate their efforts. Let's assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that will print the precise sequence of disk-to-disk peg transfers. If we were to approach this problem with conventional methods, we'd rapidly find ourselves hopelessly knotted up in managing the disks. Instead, if we attack the problem with recursion in mind, it immediately becomes tractable. Moving n disks can be viewed in terms of moving only n – 1 disks (and hence the recursion) as follows: a) Move n – 1 disks from peg 1 to peg 2, using peg 3 as a temporary holding area.



b) Move the last disk (the largest) from peg 1 to peg 3.
c) Move the n – 1 disks from peg 2 to peg 3, using peg 1 as a temporary holding area.
The process ends when the last task involves moving n = 1 disk, i.e., the base case. This is accomplished by trivially moving the disk without the need for a temporary holding area. Write a program to solve the Towers of Hanoi problem. Use a recursive function with four parameters:
a) The number of disks to be moved

b) The peg on which these disks are initially threaded
c) The peg to which this stack of disks is to be moved
d) The peg to be used as a temporary holding area Your program should print the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from peg 1 to peg 3, your program should print the following series of moves:
1 → 3 (This means move one disk from peg 1 to peg 3.)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3