CodeQuotient

Department of Computer Science & Engineering

LAB MANUAL

Advance Data Structures_03-12-2024

Student Name	**ALISHA VASHISHT**
Email	**alisha1602.be23@chitkara.edu.in**
Course Name	**Advance Data Structures_03-12-2024**

Faculty Incharge

_____

Table of Contents

19	Find the number of swaps in Selection Sort	Not Attempted
20	FInd the numbers of shifts in Insertion sort	Not Attempted
21	Matrix Multiplication	Not Attempted
22	Row or Column sum	Not Attempted
23	Spirally traversing a matrix	Not Attempted
24	Rotate a 2-D array by 90 degrees	Not Attempted
25	Reverse the order of words of a string	Not Attempted
26	String is subsequence or not	Not Attempted
27	Technical Issue With The Keyboard	Not Attempted
28	Implement atoi and itoa functions	Not Attempted
29	Implement strcat function	Not Attempted
30	Count words	Not Attempted
31	Spell the number	Not Attempted
32	Check if strings are rotations or not	Not Attempted
33	Print the List	Not Attempted
34	Copy first list to second list	Not Attempted
35	Move the Smallest and largest to head and tail of list	Not Attempted
36	Check List for Palindrome	Not Attempted
37	Find the loop in Linked list	Not Attempted
38	Reverse a linked list	Not Attempted
39	Add Two Numbers Represented by Lists	Not Attempted

40	Delete a Node in Linked list given access to only that Node	Not Attempted
41	Swap Two Nodes of Doubly Linked List	Not Attempted
42	Rotate the Doubly Linked List by K elements	Not Attempted
43	Rearrange the Even-Odd Nodes of Doubly Linked List	Not Attempted
44	Given list is circular or not	Not Attempted
45	Insert Nodes in a Circular Linked List	Not Attempted
46	Delete in Circular Linked List	Not Attempted
47	Count the Number of Nodes in Circular Linked List	Not Attempted
48	Insert in a sorted circular linked list	Not Attempted
49	Split the Circular Linked List in two parts	Not Attempted
50	Implement the stack using array	Not Attempted
51	Reverse a string using stack	Not Attempted
52	Implement the stack using Linked List	Not Attempted
53	Design stack for push, pop and min functions	Not Attempted
54	Find the next greater element on right side	Not Attempted
55	Find the minimum bracket reversals for balanced expression	Not Attempted
56	Evaluate the postfix expression using Stack	Not Attempted
57	Evaluate the prefix expression using Stack	Not Attempted
58	Implement the queue using array	Not Attempted
59	Reverse a given queue	Not Attempted

60	Reverse first N elements of a given queue	Not Attempted
61	Create a binary tree from array	Not Attempted
62	Print binary tree with level order traversal	Not Attempted
63	Print nodes at odd levels of the binary tree	Not Attempted
64	Write iterative version of inorder traversal	Not Attempted
65	Complete the inorder(), preorder() and postorder() functions for traversal with recursion	Not Attempted
66	Construct tree from given inorder and post order traversal	Not Attempted
67	Count the number of leaf and non-leaf nodes in a binary tree	Not Attempted
68	Print all paths to leaves and their details of a binary tree	Not Attempted
69	Find the right node of a given node	Not Attempted
70	Convert a binary tree into its mirror tree	Not Attempted
71	Print cousins of a given node in Binary Tree	Not Attempted
72	Print nodes in a top view of Binary Tree	Not Attempted
73	Find out if the tree can be folded or not	Not Attempted
74	Given two trees are identical or not	Not Attempted
75	Find maximum depth or height of a binary tree	Not Attempted
76	Evaluation of expression tree	Not Attempted
77	Given binary tree is binary search tree or not	Not Attempted
78	Find the kth smallest element in the binary search tree	Not Attempted

79	Convert Level Order Traversal to BST	Not Attempted
80	Find a lowest common ancestor of a given two nodes in a binary search tree	Not Attempted
81	Find the floor and ceil of a key in binary search tree	Not Attempted

Aim: Factorial using recursion

Write a recursive function **factorial** that accepts an integer *n* as a parameter and returns the factorial of *n*, or *n!*.
A factorial of an integer is defined as the product of all integers from 1 through that integer inclusive. For example, the call of **factorial(4)** should return 1 * 2 * 3 * 4, or 24. The factorial of 0 and 1 are defined to be 1.
You may assume that the value passed is non-negative and that its factorial can fit in the range of type int.

Input Format:
The first line of input contains number of testcases , T.
Then T lines follow, which contains an integer,n.

Output Format:
For each testcase print the factorial in new line

Sample Input
2
4
3

Sample Output
24
6

Solution:

```
/*
 * Complete the function 'factorial' given below
 * @params
 *  n -> an integer whose factorial is to be calculated
 * @return
 *  The factorial of integer n
 */
int factorial(int n) {
 if(n==0||n==1){
    return 1;
 }
   return n*factorial(n-1);
}
```


ALISHA VASHISHT
alisha1602.be23@chitkara.edu.in

Aim: Sum of all the digits using recursion

Write a recursive function **sumOfDigits** that accepts an integer as a parameter and returns the sum of its digits. For example, calling sumOfDigits(1729) should return 1 + 7 + 2 + 9, which is 19. If the number is negative, return the negation of the value. For example, calling **sumOfDigits**(-1729) should return -19.
Constraints: Do not declare any global variables. Do not use any loops; you must use recursion. Also do not solve this problem using a string. You can declare as many primitive variables like ints as you like. You are allowed to define other "helper" functions if you like; they are subject to these same constraints.

Solution:

```
int sumOfDigits(int n)
{
  if(n==0){
      return 0;
  }
    return n%10+sumOfDigits(n/10);
}
```


ALISHA VASHISHT
alisha1602.be23@chitkara.edu.in

Aim: power(base, exp)

Write a recursive function **power** that accepts two integers representing a *base* and an *exponent*, and returns the base raised to that exponent. For example, the call to power(3, 4) should return 3^4 i.e. 81. If the exponent passed is negative, then return -1.
Do not use loops or auxiliary data structures; Solve the problem recursively. Also do not use the provided library pow function in your solution.
Expected Time Complexity: O(log(n)); *here n denotes the exponent*
Input Format:
The first line of input conatins an integer T, denoting the number of test cases.
The second line of input contains 2 integers base and exponent seperated by space.
Output Format:
Print the answer when base is raised to the exponent.
Constraints:
-10 <= base <= 10
þÿ-15 <= exponent <= 15
Sample Input
2 // Test Cases
2 3
5 2
Sample Output
8
25

Solution:

```
long power(int base, int exp) {
  if(exp<0){
      return -1;
  }
    if(exp==0){
        return 1;
    }
   return base* power(base,exp-1);
}
```


ALISHA VASHISHT
alisha1602.be23@chitkara.edu.in