Pandit Deendayal Energy University

School of Technology

Department of Computer Science & Engineering

Odd Semester 2021-2022

LAB MANUAL FOR DATA STRUCTURES

BY

DR. ADITYA SHASTRI

Practical No. 1 [Revision of Arrays]

Programs covering basics of Arrays: searching, sorting, minimum and maximum elements etc.

1. Write a program in C to perform linear and binary search.

2. Write a program in C to perform bubble sort, insertion sort and selection sort. Take the array size and array elements from user.

3. Write a program in C that obtains the minimum and maximum element from the array. Modify this program to give the second largest and second smallest element of the array.

Practical No. 2 [Revision of Structures]

Programs covering structure definition, array of structure, nested structures etc.

1. Create a structure Student in C with student name, student roll number and student address as its data members. Create the variable of type student and print the values.

2. Modify the above program to implement arrays of structure. Create an array of 5 students and print their values.

3. Create a structure Organization with organization name and organization ID as its data members. Next, create another structure Employee that is nested in structure Organization with employee ID, employee salary and employee name as its data members. Write a program in such a way that there are two organizations and each of these contains two employees.

Practical No. 3 [Revision of Pointers]

Programs covering use of pointers with arrays, structures, functions etc.

1. Write a program in C to implement arrays of pointers and pointers to arrays.

2. Write a program in C to implement pointers to structures.

3. Write a program in C to perform swapping of two numbers by passing addresses of the variables to the functions.

Experiment No. 4 [Stack]

1. Implement a stack using an array having following functionalities:
    a. isEmpty – to check if the stack if empty or not
    b. isFull – to check if the stack if full or not
    c. push – to insert the element into the stack
    d. pop – to delete an element from the stack
    e. print_top – to print the top most element of the stack.

2. Given a stack, sort it using recursion. Use of any loop constructs like `while`, `for`, etc. is not allowed. We can only use the following functions on Stack S:
    a. isEmpty(S): Tests whether stack is empty or not.
    b. push(S): Adds new element to the stack.
    c. pop(S): Removes top element from the stack.
    d. top(S): Returns value of the top element.

Experiment No. 5 [Stack Applications]

1. Write a program to evaluate the following given postfix expressions:
    a. 2 3 1 * + 9 – Output: -4
    b. 2 2 + 2 / 5 * 7 + Output: 17

2. Convert the given infix expression into postfix expression using stack.
    Example- Input: $a + b * (c\hat{}d - e)\hat{}(f + g * h) - i$
    Output: $abcd\hat{}e - fgh * +\hat{} * +i -$

3. Given an expression, write a program to examine whether the pairs and the orders of "{", "}", "(", ")", "[", "]" are correct in the expression or not.
    Example: Input: exp = "[( )]{ }{[( )( )]( )}" Output: Balanced
    Input: exp = "[( ])" Output: Not Balanced

Practical No. 6 [Queue]

1. Implement the following functionalities of the **Circular queue using Arrays**:
    a. isFull – to check if the queue is full or not.
    b. isEmpty – to check if the queue is empty or not.
    c. enqueue – to insert the element in the queue.
    d. dequeue – to delete the element from the queue.
    e. front and rear – to print the front and rear element of the queue.

2. Implement various functionalities of Queue using Linked Lists. Again, you can implement operation given above.

3. Implement Double Ended Queue that supports following operation:
    a. insertFront(): Adds an item at the front of Deque.
    b. insertLast(): Adds an item at the rear of Deque.
    c. deleteFront(): Deletes an item from the front of Deque.
    d. deleteLast(): Deletes an item from the rear of Deque.

4. Implement Double Ended Queue that supports following operation:
    a. getFront(): Gets the front item from the queue.
    b. getRear(): Gets the last item from queue.
    c. isEmpty(): Checks whether Deque is empty or not.
    d. isFull(): Checks whether Deque is full or not.

Practical No. 7 [Linked List]

1. Write a program to ***insert a new node*** into the linked list. A node can be added into the linked list using three ways: [Write code for all the three ways.]
    a. At the front of the list
    b. After a given node
    c. At the end of the list.

2. Write a program to ***delete a node*** from the linked list. A node can be deleted from the linked list using three ways: [Write code for all the three ways.]
    a. Delete from the beginning
    b. Delete from the end
    c. Delete from the middle.

3. Write a program that takes two sorted lists as inputs and merge them into one sorted list.
   For example, if the first linked list ***A*** is 5 =>10 =>15, and the other linked list ***B*** is 2 => 3 => 20, then output should be 2 => 3 => 5 => 10 => 15 => 20.

4. Implement the ***circular linked list*** and perform the operation of traversal on it. In a conventional linked list, we traverse the list from the head node and stop the traversal when we reach NULL. In a circular linked list, we stop traversal when we reach the first node again.

5. Implement the ***doubly linked list*** and perform the deletion and/ or insertion operation on it. Again, you can perform insertion deletion according to the three ways as given above. Implement all of them according to availability of time.

Practical No. 8 [Trees]

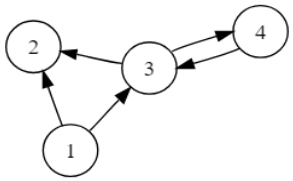
1. Implement the Binary Tree and perform **any one** of the following three types of traversals: **(Implement iterative method of traversal, not recursive)**
    a. Pre-order Traversal
    b. In-order Traversal
    c. Post-order Traversal

2. Write a program to insert an element, delete an element and search an element in the Binary Tree.

3. Given a preorder traversal sequence of a Binary Search Tree, construct the corresponding Binary Search Tree.

Practical No. 9 [Graphs]

1. Implement a graph $G = (V, E)$ using a linked list. This should contain the node list and edge list connected using a linked list as below for the given the graph.



Node List Edge list
$1 \rightarrow 2 \rightarrow 3$
$2 \rightarrow Null$
$3 \rightarrow 2 \rightarrow 4$
$4 \rightarrow 3$

2. For a given graph $G = (V, E)$, study and implement the Breadth First Search (or traversal) i.e., BFS. Also, perform complexity analysis of this algorithm in-terms of time and space.

3. For a given graph $G = (V, E)$, study and implement the Depth First Search (or traversal) i.e., DFS. Also, perform complexity analysis of this algorithm in-terms of time and space.

Practical No. 10 [Hashing]

1. Implement a suitable hashing function to insert elements in hash table. In case of collision, use separate chaining as a resolution technique. Print the hash table after inserting all the values.

2. Given a limited range array containing both positive and non-positive numbers, i.e., elements are in the range from -MAX to +MAX. Our task is to search if some number is present in the array or not in O(1) time.

3. Given two arrays: *A* and *B*. Find whether *B* is a subset of *A* or not using Hashing. Both the arrays are not in sorted order. It may be assumed that elements in both arrays are distinct.