**K. J. Somaiya Institute of Technology**
**Sion, Mumbai – 400022**

**DEPARTMENT OF COMPUTER ENGINEERING**

**Academic Year: 2025-26 (Odd Semester)**

# Lab Manual

**Class – SY** **Semester - III**

**Subject: Data Structure Lab**
**Course code: CEL302**

**K. J. SOMAIYA INSTITUTE OF TECHNOLOGY**
**DEPARTMENT OF COMPUTER ENGINEERING**

**Class –S.Y** **Sem – III**

Course Code – CEL302 Course Name - Data Structures Lab

## List of Experiments

| Exp. No. | Name of Experiment |
|---|---|
| 1 | Implement Stack ADT using an array. |
| 2 | Convert an Infix expression to Postfix expression using stack ADT. |
| 3 | Application of Stack using array. |
| 4 | Implement Linear Queue ADT using an array. |
| 5 | Implement Circular Queue ADT using an array. |
| 6 | Implement a Singly Linked List. |
| 7 | Implement Doubly Linked List ADT. |
| 8 | Implement Binary Search Tree ADT using Linked List. |
| 9 | Implement Graph Traversal techniques :) Depth First Search b) Breadth First Search |
| 10 | To find an element in a sorted array using binary search technique |

## EXPERIMENT NO:-1

**AIM**- Implement Stack ADT using an array.

**THEORY**:

Stack can be represented using an array. Stack is open at one end and operations can be performed on a single end. We can have different primitive operations on Stack Data Structure.

**A. Push Operation on Stack :** **Push** is the operation used to add an element to the top of a stack.

**B. Pop Operation on Stack :** **Pop** is the operation used to remove the topmost element from the stack.

**ALGORITHM:**

**Push ( ):**

Description: Here STACK is an array with MAX locations. TOP points to the top most element and ITEM is the value to be inserted.

```
1. If (TOP == MAX-1) Then [Check for overflow]
2. Print: Overflow
3. Else
4. Set TOP = TOP + 1 [Increment TOP by 1]
5. Set STACK[TOP] = ITEM [Assign ITEM to top of STACK]
6. Print: ITEM inserted
[End of If]
7. Exit
```

**Pop ( ):**

Description: Here STACK is an array with MAX locations. TOP points to the top most element.

```
1. If (TOP == -1) Then [Check for underflow]
2. Print: Underflow
3. Else
4. Set ITEM = STACK[TOP] [Assign top of STACK to ITEM]
5. Set TOP = TOP - 1 [Decrement TOP by 1]
6. Print: ITEM deleted
```

```
[End of If]
7. Exit
```

**Conclusion:**

(Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. Conclusion carry 4 marks out of 10)

## Experiment No. 2

**Aim:** Convert an Infix expression to postfix expression using stack ADT.

**Theory:**

To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

**Algorithm**

infixToPostfix(infix)

**Input −** Infix expression.

**Output −** Convert infix expression to postfix form.

```
Begin
  initially push some special character say # into the stack
  for each character ch from infix expression, do
    if ch is alphanumeric character, then
      add ch to postfix expression
    else if ch = opening parenthesis (, then
      push ( into stack
    else if ch = ^, then            //exponential operator of higher precedence
      push ^ into the stack
    else if ch = closing parenthesis ), then

      while stack is not empty and stack top ≠ (,
        do pop and add item from stack to postfix expression
      done
      pop ( also from the stack
    else
      while stack is not empty AND precedence of ch <= precedence of stack top element, do
        pop and add into postfix expression
      done
      push the newly coming character.
  done
  while the stack contains some remaining characters, do
    pop and add to the postfix expression
  done
  return postfix
End
```

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. Conclusion carry 4 marks out of 10

## Experiment No. 3

**Aim: Stack application using Array.**

WAP to check if an expression contains balanced brackets: `()`, `{}`, and `[]`.

**Theory:**

A stack is a linear data structure that follows the LIFO (Last In, First Out) principle. The last element inserted (pushed) is the first to be removed (popped). Stacks can be implemented using arrays or linked lists. The Valid Parentheses Checker is a classic application of the stack. It verifies whether the opening and closing brackets in an expression are balanced and properly nested**.**

**Algorithm:**

- Use a stack to store opening brackets.
- For each character**:**
  - If it's an opening bracket (`(`, `{`, `[`), push it to the stack.
  - If it's a closing bracket (`)`, `}`, `]`):
    - Check if the stack is not empty and top of the stack is the matching opening bracket**.**
    - If yes, pop the top.
    - Else, it's invalid**.**
- After processing the string:
  - If the stack is empty**,** it's valid.
  - If not, it's invalid.

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10 )**

## Experiment No.4

**Aim:** Implement Linear Queue ADT using an array.

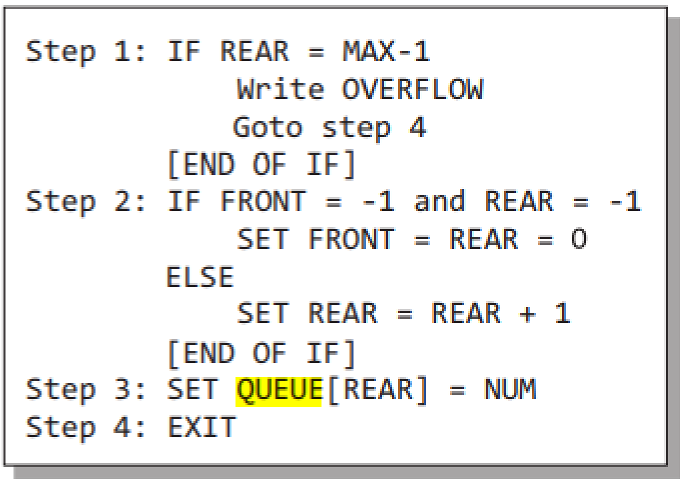
**Theory:**

### LinearQueues

A queue is an ordered list in which items may be added only at one end called the "rear" and items may be removed only at the other end called "front".

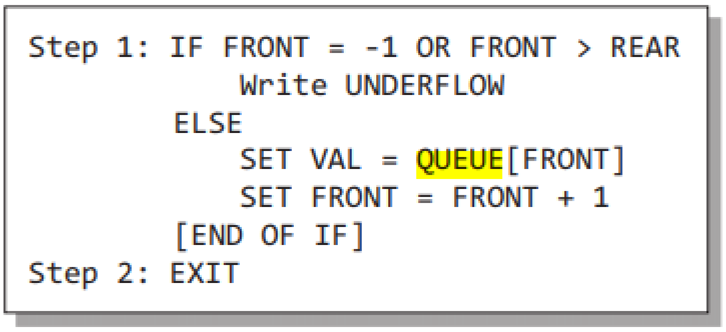
- Queues may be represented in the computer memory by means of linear arrays or linked list.
- There are two pointer variables namely FRONT and REAR.
- The FRONT denotes the location of the first element of the queue.
- The rear describes the location of the rear element of the queue.
- When the queue is empty, the values of front and rear are-1 and -1 respectively.
- The "max-size "represents the maximum capacity of the linear queue.
- The following is the condition to test the queue is full or not. (Rear== max size-1)
- To insert the first element both pointers should be altered to „0‟ Front =0 rear=0.
- Whenever an item is added to the queue, the value of REAR is incremented by 1. REAR =REAR+1;
- The following is the condition to test the queue is empty or not. Front==-1
- Whenever an "Item" is deleted from the queue, the value if the front is incremented by 1 Front=Front+1;
- To delete the last element, both pointers should be altered.

**Algorithm:**



```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

Algorithm to insert an element in a queue



```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Algorithm to delete an element from a queue

**Conclusion:**(Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )

## Experiment No. 5

**Aim:** Implementation of Circular Queue using Array

**Theory: Circular Queue**

A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a ***Ring Buffer***.

**Operations on Circular Queue:**

- Front: It is used to get the front element from the Queue.
- Rear: It is used to get the rear element from the Queue.
- enQueue(value): This function is used to insert the new value in the Queue. The new element is always inserted from the rear end.
- deQueue(): This function deletes an element from the Queue. The deletion in a Queue always takes place from the front end.

The steps of enqueue operation are given below:

- First, we will check whether the Queue is full or not.
- Initially the front and rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.
- When we insert a new element, the rear gets incremented, i.e., *rear=rear+1*.

The steps of dequeue operation are given below:

- First, we check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.
- When the element is deleted, the value of front gets decremented by 1.
- If there is only one element left which is to be deleted, then the front and rear are reset to -1.

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )

## Experiment No. 6

**Aim**: Implement a Singly Linked List.

**Theory:**

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

### Insertion in the Linked List

1. Insert at the beginning

   Allocate memory for new node
   Store data
   Change next of new node to point to head
   Change head to point to recently created node

2. Insert at the End

   Allocate memory for new node
   Store data
   Traverse to last node
   Change next of last node to recently created node

3. Insert at the Middle

   Allocate memory and store data for new node
   Traverse to node just before the required position of new node
   Change next pointers to include new node in between

### Delete from a Linked List

You can delete either from the beginning, end or from a particular position.

Delete from beginning

- Point head to the second node

2. Delete from end

   Traverse to second last element
   Change its next pointer to null

3. Delete from middle

   Traverse to element before the element to be deleted
   Change next pointers to exclude the node from the chain

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )

## Experiment No.7

**Aim:** Implement Doubly Linked List ADT.

**Theory:**

A Doubly Linked List (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

**Insertion**

A node can be added in four ways

1) At the front of the DLL

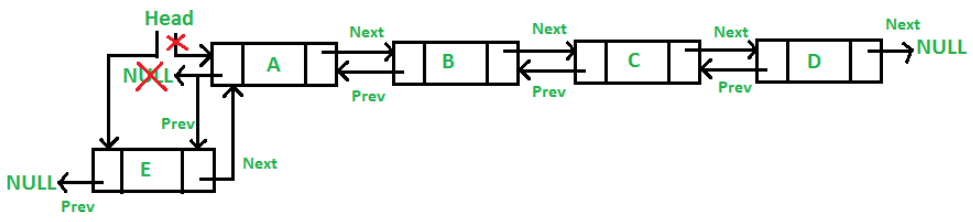
2) After a given node.

3) At the end of the DLL

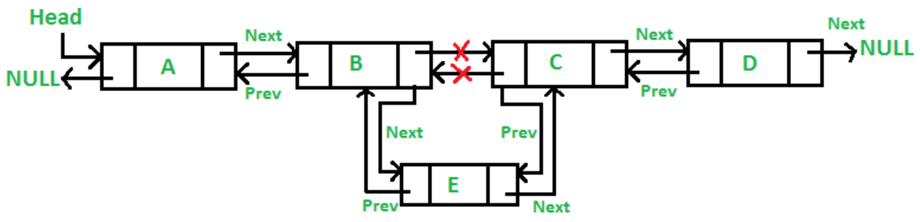
4) Before a given node.

1) Add a node at the front: (A 5 steps process)

The new node is always added before the head of the given Linked List. And newly added node becomes the new head of DLL.

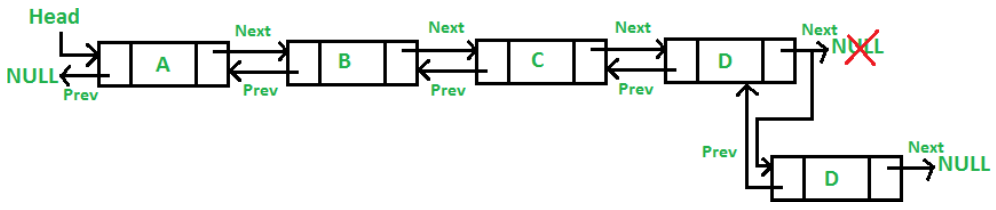


2) Insert node at a given position.We are given a pointer to a node as prev_node, and the new node is inserted after the given node.



3) Add a node at the end:

The new node is always added after the last node of the given Linked List.



**Delete a node in DLL.**

Deleting a node is similar to singly linked list,only make prev and next pointers NULL as per deletion.

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )

## Experiment No.8

**Aim: Binary search tree ADT**

**Theory:**

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node‟s key.
- The right subtree of a node contains only nodes with keys greater than the node‟s key.
- The left and right subtree each must also be a binary search tree.

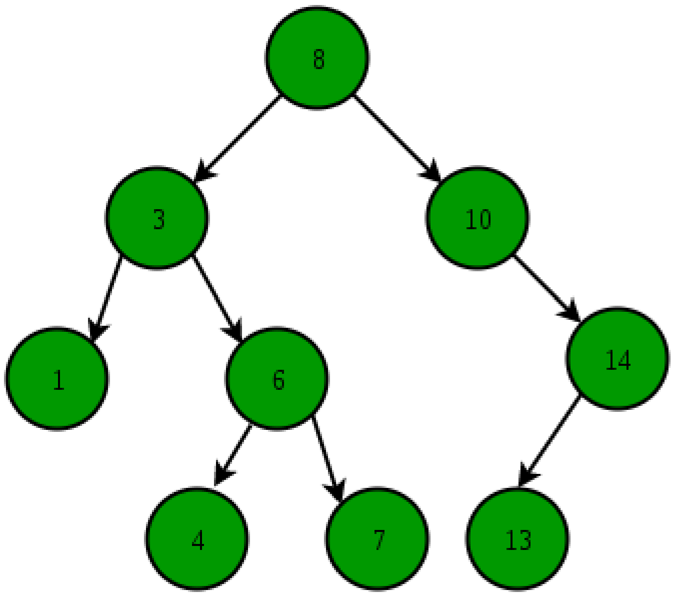


**Illustration to search 6 in below tree:**

1. Start from the root.

2.  Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

3. If the element to search is found anywhere, return true, else return false.

Insertion of a key

A new key is always inserted at the leaf. We start searching a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

```
        100                              100
       /   \       Insert 40            /    \
     20     500    --------->         20      500
    /  \                             /  \
  10    30                         10    30
                                           \
                                            40
```

**Delete a Node**

When we delete a node, three possibilities arise.

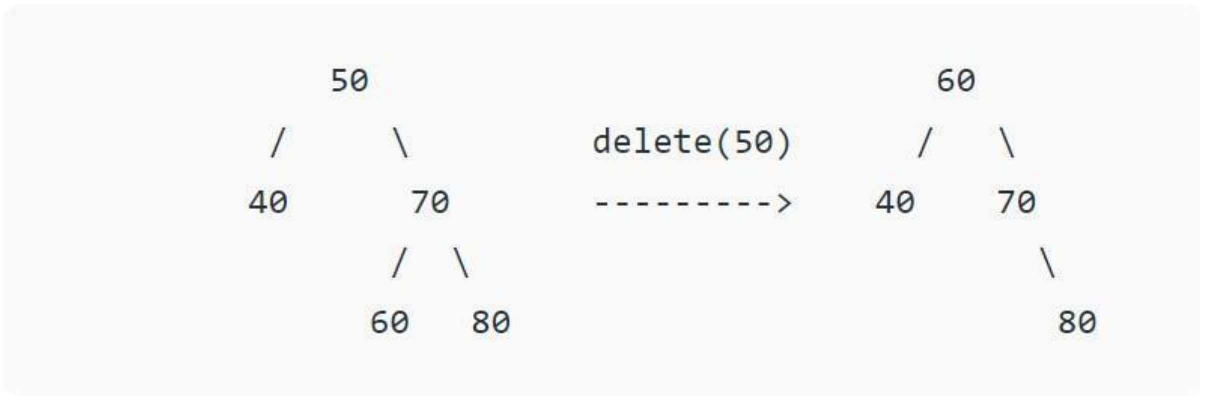
1) *Node to be deleted is the leaf:* Simply remove from the tree.

```
              50                            50
           /      \        delete(20)      /   \
          30       70      --------->    30     70
         /  \     /  \                     \    /  \
       20    40  60   80                   40  60   80
```

2) *Node to be deleted has only one child:* Copy the child to the node and delete the child

```
              50                            50
           /      \        delete(30)      /   \
          30       70      --------->    40     70
            \     /  \                          /  \
             40  60   80                       60   80
```

3) *Node to be deleted has two children:* Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.



```
       50                            60
    /      \        delete(50)     /   \
   40       70      --------->   40    70
           /  \                          \
          60   80                         80
```

The important thing to note is, inorder successor is needed only when the right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in the right child of the node.

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )

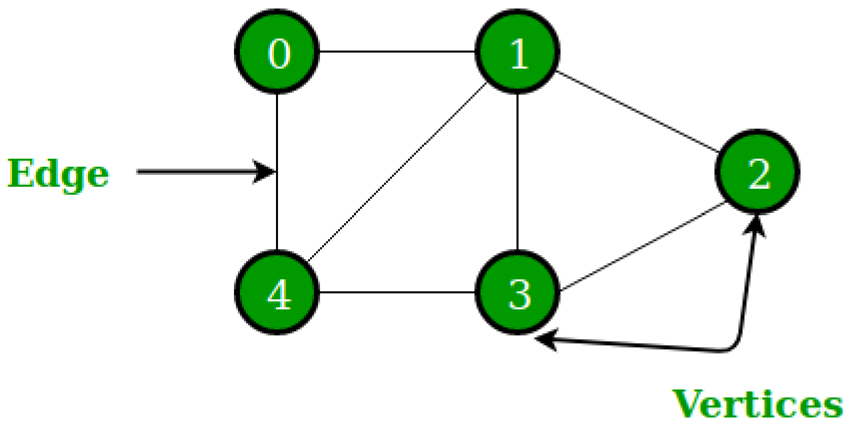
## Experiment No.9

**Aim: Graph Traversal**

**Techniques Theory:**

Graph

A graph can be defined as a group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent-child relationship.



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

**Graph Traversals:**

**BFS:Breadth First Search**

Breadth-First Traversal (or Search) for a graph is similar to Breadth-First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

```
procedure BFS(G, root) is
    let Q be a queue
    label root as explored
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as explored then
   label w as explored
   Q.enqueue(w)
```

## DFS:Depth First Search

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

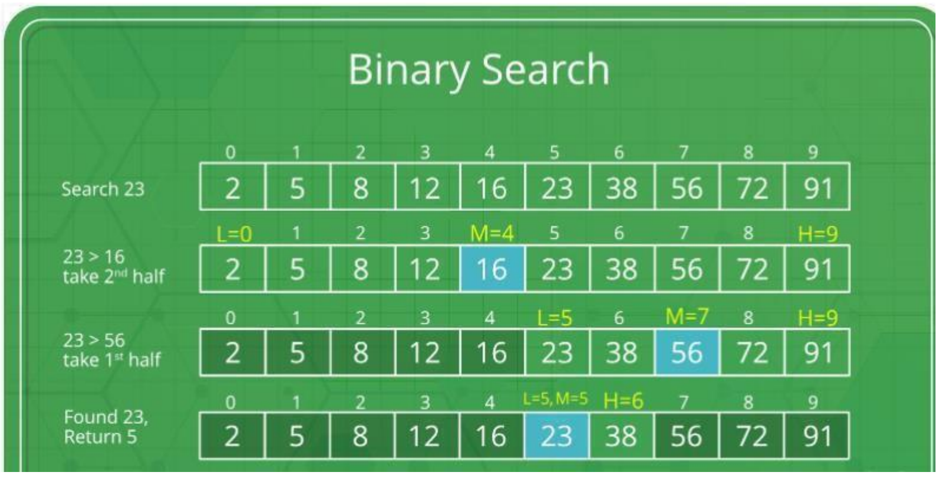
**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )

## Experiment No.10

**Aim:** To find an element in an sorted array using binary search technique.

**Theory:**

**Binary Search:** Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.



We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with the middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid element. So we recur for the right half.
4. Else (x is smaller) recur for the left half.

**Conclusion:** (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10** )