Department of Mathematics and Computing
IIT(ISM) Dhanbad

Course Name: Data Structures Practical

Course Code: MCC507

Lab Location: Computer Lab, M&C

DATA STRUCTURES PRACTICAL
(Effective from academic year 2021-2022)
SEMESTER - I

Laboratory Code	**MCC507**	**Regular Assessment**	**50**
Number of Lab Hours/Week	**02 Hours**	**Examination**	**50**
Total Number of Lab Hours	**(12+1)×2=26 Hours**	**Exam Hours**	**02**

CREDITS-02

Course Objective: Data Structures is the basic course of Computer Science. It is required in every field of Computer Science. Objective of this course is to impart knowledge of Data Structures.

Descriptions (if any)
Implement all the experiments in C/C++ Language under Linux / Windows environment.

Ex. No.	**Laboratory Experiments Description**	**Page No.**
	Introduction to Data Structures	**1**
1	Design, Develop and Implement a menu driven Program in C/C++ for the following **Array** operations a. Creating an Array of N Integer Elements b. Display of Array Elements with Suitable Headings c. Inserting an Element at a given valid Position (**POS**) d. Deleting an Element at a given valid Position(**POS**) e. Exit. Support the program with functions for each of the above operations.	**1**
2	Design, Develop and Implement the following menu driven Programs in C/C++ using **Array** operations a. Write a program for Bubble Sort algorithm b. Write a program for Merge Sort algorithm c. Write a program for Radix Sort algorithm d. Write a program for Insertion Sort algorithm e. Write a program for Selection Sort algorithm	**4**
3	Design, Develop and Implement the following menu driven Programs in C/C++ for implementing a. Write a program for Heap Sort algorithm b. Write a program for Quick Sort algorithm c. Write a program for linear search algorithm d. Write a program for displaying a sparse matrix e. Fibonacci numbers f. Factorial of a number	**13**
4	Design, Develop and Implement a menu driven Program in C/C++ for the following operations on STACK of Integers (Array Implementation of Stack with maximum size **MAX**) a. **Push** an Element on to Stack b. **Pop** an Element from Stack **c.** Demonstrate how Stack can be used to check **Palindrome** d. Demonstrate **Overflow** and **Underflow** situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations	**22**
5	Design, Develop and Implement a Program in C/C++ for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (**Remainder**), ^ (**Power**) and **alphanumeric** operands.	**26**

6	Design, Develop and Implement a Program in C/C++ for the following Stack Applications **a.** Evaluation of **Suffix expression** with single digit operands and operators: **+, -, *, /, %, ^** b. Solving **Tower of Hanoi** problem with **n** disks	**29**
7	Design, Develop and Implement a menu driven Program in C/C++ for the following operations on **Circular QUEUE** of Characters a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations	**33**
8	Design, Develop and Implement a menu driven Program in C/C++ for the following operations on **Singly Linked List (SLL)** a. Create a **SLL**. b. Insert at Beginning c. Insert at Last d. Insert at any random location e. Delete from Beginning f. Delete from Last g. Delete node after specified location h. Search for an element i. Show j. Exit	**38**
9	Design, Develop and Implement a menu driven Program in C/C++ for the following operations on **Doubly Linked List (DLL)** a. Create a **DLL**. b. Print all the elements in DLL in forward traversal order c. Print all elements in DLL in reverse traversal order d. Exit	**54**
10	Design, Develop and Implement a menu driven Program in C/C++ for the following operations on **Binary Search Tree (BST)** of Integers a. Create a BST of **N** Integers: 8, 10, 3, 1, 6, 14, 7 b. Traverse the BST in Inorder c. Traverse the BST in Preorder d. Traverse the BST in and Post Order	**56**
11*	Design, Develop and Implement a Program in C/C++ for the following operations on **Graph(G)** a. Create a Graph **N** using Adjacency Matrix. b. Print all the nodes **reachable** from a given starting node in a digraph using **BFS** method c. Print all the nodes **reachable** from a given starting node in a digraph using **DFS** method.	**59**

***12 Lab sessions are assigned for experiments and 1 Lab session is assigned for Lab Examination. Experiments 1-10 will be carried out in 10 Lab sessions each and Experiment 11 will be carried out in 2 Lab sessions.**

Course outcomes: On the completion of this laboratory course, the students will be able to:

a. Analyze and compare various linear and non-linear data structures
b. Code, debug and demonstrate the working nature of different types of data structures and their applications
c. Implement, analyse, and evaluate the searching and sorting algorithms
d. Choose the appropriate data structure for solving real world problems

Graduate Attributes

1. Engineering Knowledge
2. Problem Analysis
3. Design/Development of Solutions
4. Modern Tool Usage

General Instructions (Do's & Don'ts):

1. Students should only use their assigned user accounts and not meddle with other's accounts.

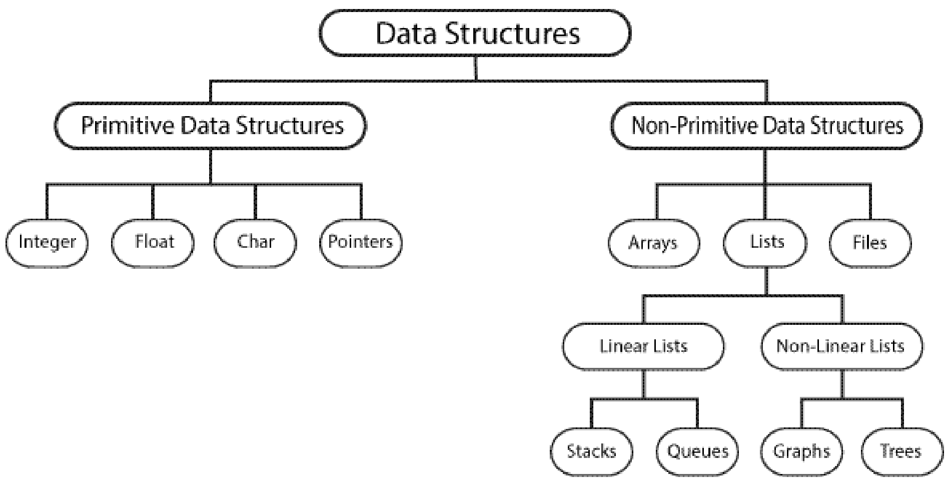
2. Students should not meddle with attached hardware components such as mouse, keyboards, etc.
3. Students should use internet connection only for Lab purpose and not for personal use.

Conduction of Practical Examination:

1. All laboratory experiments (**ELEVEN** nos.) are to be included for practical examination. Students are allowed to pick one experiment from the lot through lottery.
2. Strictly follow the instructions as printed on the cover page of answer script
3. Marks distribution: **Regular Assessment (50 Marks):** Attendance (10)+Conduct (10)+Average marks secured for all the experiments carried out in Lab classes (30)
   Examination Assessment (50 Marks): Algorithm & Execution of Program (35 Marks) + Viva (15 Marks)
4. **In Examination, second chance for choosing experiment through lottery is allowed only once (without deduction of marks). Each further chances (third onwards) for choosing experiment through lottery will invite deduction of 5 Marks.**

Introduction to Data Structures

Data Structure is defined as the way in which data is organized in the memory location. There are 2 types of data structures:



Linear Data Structure:
In linear data structure all the data are stored linearly or contiguously in the memory. All the data are saved in continuously memory locations and hence all data elements are saved in one boundary. A linear data structure is one in which we can reach directly only one element from another while travelling sequentially. The main advantage, we find the first element, and then it's easy to find the next data elements. The disadvantage, the size of the array must be known before allocating the memory.
The different types of linear data structures are:

- Array
- Stack
- Queue
- Linked List

Non-Linear Data Structure:
Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.
The various types of non-linear data structures are:

- Trees
- Graphs

EXPERIMENT 1

OBJECTIVE:

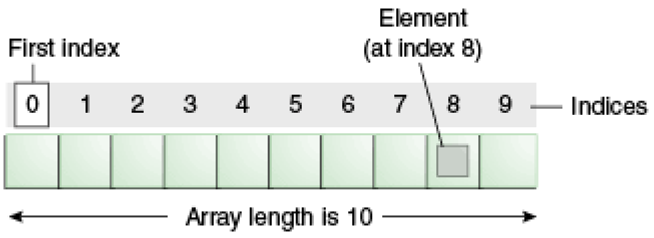
Design, Develop and Implement a menu driven Program in C for the following **Array** operations

a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Inserting an Element at a given valid Position (**POS**)
d. Deleting an Element at a given valid Position(**POS**)
e. Exit.

Support the program with functions for each of the above operations.

THEORY:

An array is collection of items stored at contiguous memory locations. The idea is to store multiple items of same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).



Visualization of an array

ALGORITHM:

```
Step 1: Start.
Step 2: Read N value.
Step 3: Read Array of N integer elements
Step 4: Print array of N integer elements.
Step 5: Insert an element at given valid position in an array.
Step 6: Delete an element at given valid position from an array.
Step 7: Stop.
```

PROGRAM IN C++

```
#include<iostream>
using namespace std;
/* Global variables declaration */
int a[10], n, elem, i, pos; //predefining array size to be maximum of size 10
/*Function Prototype and Definitions*/
void create() //creating an array
{
        cout<<"\nEnter the size of the array elements<=9: ";
        cin>>n;
        cout<<"\nEnter the elements for the array:\n";
        for(i=0; i<n; i++)
        cin>>a[i]; //end of create()
} //end of create()
void display() //displaying an array elements
{
        int i;
        cout<<"\nThe array elements are:\n";
        for(i=0; i<n; i++)
        {
                cout<<a[i]<<"\n";
        }
} //end of display()
void insert() //inserting an element in to an array
{
        cout<<"\nEnter the position for the new element: ";
        cin>>pos;
        cout<<"\nEnter the element to be inserted: ";
        cin>>elem;
        pos=pos-1;
        for(i=n-1; i>=pos; i--)
        {
                a[i+1] = a[i];
        }
        a[pos] = elem;
        n = n+1;
} //end of insert()
void del() //deleting an array element
{
        cout<<"\nEnter the position of the element to be deleted: ";
        cin>>pos;
        pos=pos-1;
        elem = a[pos];
        for(i=pos; i<n-1; i++)
        {
                a[i] = a[i+1];
        }
        n = n-1;
        cout<<"\nThe deleted element is\n"<<elem;
}//end of delete()
int main()
{

        int ch;
        create();
        display();

        cout<<"\n\n--------Menu ----------\n";
        cout<<"1.Insert\n 2.Delete\n 3.Exit\n";
        cout<<"_________________________________ ";
        cout<<"\nEnter your choice: ";
        cin>>ch;
        switch(ch)
        {
        case 1:         insert();
                        display();
                        break;
        case 2:         del();
                        display();
                        break;
        case 3:         return 0;
                        break;
        default:        cout<<"\nInvalid choice:\n";
                        break;
        }
}// end of main
```

PROGRAM OUTPUT:

```
Enter the size of the array elements: 5
Enter the elements for the array:
10 20 30 40 50
The array elements are:
10 20 30 40 50
---------Menu------------
1.Insert
4.Delete
5.Exit
Enter your choice: 1
Enter the position for the new element: 2
Enter the element to be inserted: 90
The array elements are:
10 90 20 30 40 50
```

QUESTIONS:

1. **Write a program in C/C++ to create an array of SIZE=10 by inserting the following elements 3, 20, 10, 12, 61, 23**

2. **Write a program in C/C++ to create the following array and insert an element 20 at the position 3 in the following array 2, 3, 12, 50, 18, 1**

3. **Write a program in C/C++ to create the following array and delete the element at the position 5 in the following array 6, 22, 11, 44, 20, 2**

EXPERIMENT 2

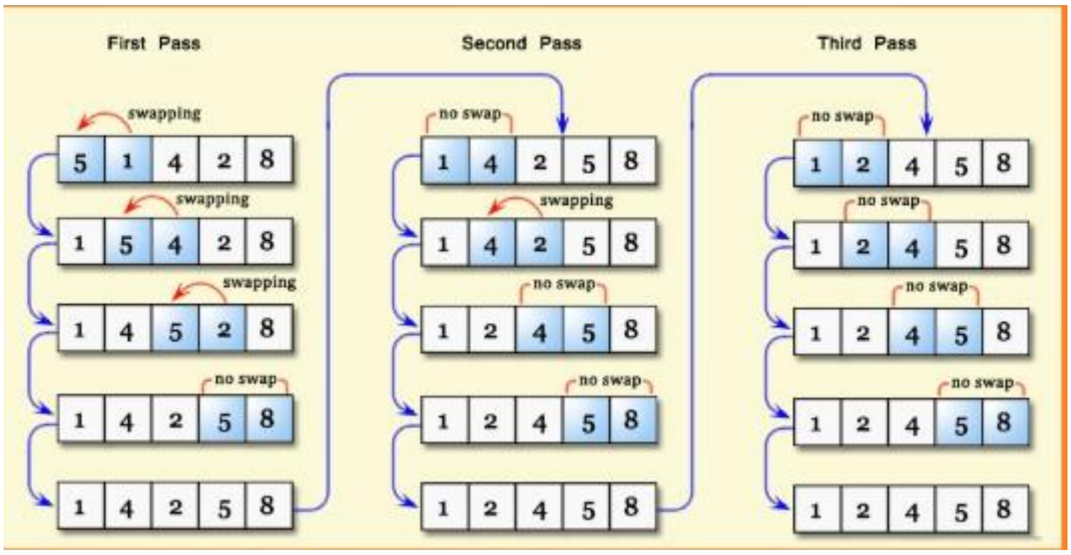
OBJECTIVE:

Design, Develop and Implement the following menu driven Programs in C/C++ using **Array** operations

(a) Write a program for Bubble sort algorithm
(b) Write a program for Merge Sort algorithm
(c) Write a program for Radix Sort algorithm
(d) Write a program for Insertion Sort algorithm
(e) Write a program for Selection Sort algorithm

a) THEORY (BUBBLE SORT):

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.



Visualization of bubble sort technique

ALGORITHM:

Step 1: Start.
Step 2: Starting with the first element(index = 0), compare the current element with the next element of the array.
Step 3: If the current element is greater than the next element of the array, swap them.
Step 4: If the current element is less than the next element, move to the next element.

Step 5: Repeat Step 1 until sorted.
Step 6: Stop

PROGRAM IN C++

```
#include<iostream>
using namespace std;
int main()
{
int n, i, arr[50], j, temp;
cout<<"Enter total number of elements :";
cin>>n;
cout<<"Enter "<<n<<" numbers :";
for(i=0; i<n; i++)
{
cin>>arr[i];
}
cout<<"Sorting array using bubble sort technique...\n";
for(i=0; i<(n-1); i++)
{
for(j=0; j<(n-i-1); j++)
{
if(arr[j]>arr[j+1])
{
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
}
}
}
cout<<"Elements sorted successfully!\n";
cout<<"Sorted list in ascending order:\n";
for(i=0; i<n; i++)
{
cout<<arr[i]<<" ";
}
return 0;
}
```

PROGRAM OUTPUT:

```
Enter total number of elements :5
Enter 5 numbers :56
32
2
99
67
Sorting array using bubble sort technique...
Elements sorted successfully!
Sorted list in ascending order:
2 32 56 67 99
```

b) THEORY (MERGE SORT):

Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.

PROGRAM IN C++ to Merge sort a given array: 12 11 13 5 6 7

```
#include <iostream>
using namespace std;

// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid, int const right)
{
   auto const subArrayOne = mid - left + 1;
   auto const subArrayTwo = right - mid;

   // Create temp arrays
   auto *leftArray = new int[subArrayOne],
        *rightArray = new int[subArrayTwo];

   // Copy data to temp arrays leftArray[] and rightArray[]
   for (auto i = 0; i < subArrayOne; i++)
      leftArray[i] = array[left + i];
   for (auto j = 0; j < subArrayTwo; j++)
      rightArray[j] = array[mid + 1 + j];

   auto indexOfSubArrayOne = 0, // Initial index of first sub-array
      indexOfSubArrayTwo = 0; // Initial index of second sub-array
   int indexOfMergedArray = left; // Initial index of merged array

   // Merge the temp arrays back into array[left..right]
   while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {
      if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
         array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
         indexOfSubArrayOne++;
      }
      else {
         array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
         indexOfSubArrayTwo++;
      }
      indexOfMergedArray++;
   }
   // Copy the remaining elements of
   // left[], if there are any
   while (indexOfSubArrayOne < subArrayOne) {
      array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
      indexOfSubArrayOne++;
      indexOfMergedArray++;
   }
   // Copy the remaining elements of
   // right[], if there are any
   while (indexOfSubArrayTwo < subArrayTwo) {
      array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
```

```
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}

// begin is for left index and end is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return; // Returns recursively

    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

// Function to print an array
void printArray(int A[], int size)
{
    for (auto i = 0; i < size; i++)
        cout << A[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    auto arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    cout << "\nSorted array is \n";
    printArray(arr, arr_size);
    return 0;
}
```

PROGRAM OUTPUT:

```
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
```

c) THEORY (RADIX SORT):

The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

PROGRAM IN C++ to Radix sort a given array: 170, 45, 75, 90, 802, 24, 2, 66

```
#include <iostream>
using namespace std;

// A utility function to get maximum value in arr[]
int getMax(int arr[], int n)
{
   int mx = arr[0];
   for (int i = 1; i < n; i++)
      if (arr[i] > mx)
         mx = arr[i];
   return mx;
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp)
{
   int output[n]; // output array
   int i, count[10] = { 0 };

   // Store count of occurrences in count[]
   for (i = 0; i < n; i++)
      count[(arr[i] / exp) % 10]++;

   // Change count[i] so that count[i] now contains actual
   // position of this digit in output[]
   for (i = 1; i < 10; i++)
      count[i] += count[i - 1];

   // Build the output array
   for (i = n - 1; i >= 0; i--) {
      output[count[(arr[i] / exp) % 10] - 1] = arr[i];
      count[(arr[i] / exp) % 10]--;
   }

   // Copy the output array to arr[], so that arr[] now
   // contains sorted numbers according to current digit
   for (i = 0; i < n; i++)
      arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n)
{
   // Find the maximum number to know number of digits
   int m = getMax(arr, n);

   // Do counting sort for every digit. Note that instead
   // of passing digit number, exp is passed. exp is 10^i
   // where i is current digit number
   for (int exp = 1; m / exp > 0; exp *= 10)
      countSort(arr, n, exp);
}

// A utility function to print an array
void print(int arr[], int n)
```

```
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
    int n = sizeof(arr) / sizeof(arr[0]);

      // Function Call
      radixsort(arr, n);
cout << "\nSorted array is: \n";
    print(arr, n);
    return 0;
}
```

PROGRAM OUTPUT:

```
Sorted array is:
2 24 45 66 75 90 170 802
```

d) THEORY (INSERTION SORT):

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

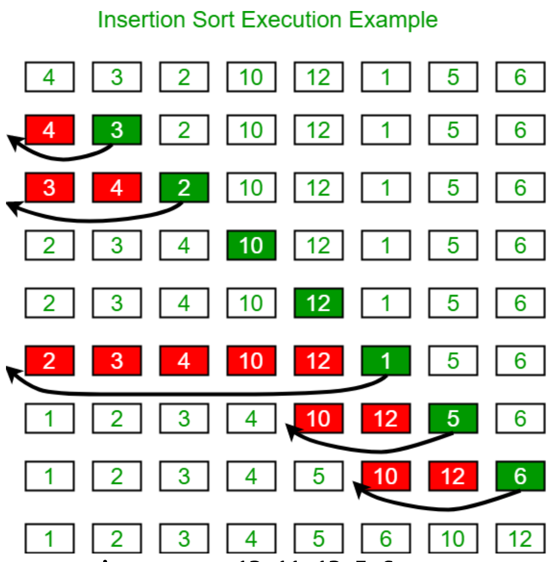
Algorithm

To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.

2: Compare the current element (key) to its predecessor.

3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.



PROGRAM IN C++ to Insertion sort a given array: 12, 11, 13, 5, 6

```
#include <iostream>
using namespace std;
```

```
 void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
      while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
cout << "\nSorted array is: \n";
    printArray(arr, n);

    return 0;
}
```

PROGRAM OUTPUT:

```
Sorted array is:
5 6 11 12 13
```

e) THEORY (SELECTION SORT):

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.

2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

```
arr[] = 64 25 12 22 11


// Find the minimum element in arr[0...4]

// and place it at beginning
```

```
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

PROGRAM IN C++ to Selection sort a given array: 64, 25, 12, 22, 11

```
#include <iostream>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
```

```
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}
 int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

PROGRAM OUTPUT:

```
Sorted array:
11 12 22 25 64
```

QUESTIONS:

1. **Write programs in C/C++ to perform Bubble/ Merge / Radix / Insertion / Selection sort algorithm for the following array**
   12, 3, 40, 1, 60, 2, 99, 101

EXPERIMENT 3

OBJECTIVE:

Design, Develop and Implement the following menu driven Programs in C/C++ for implementing

(a) Write a program for Heap Sort algorithm
(b) Write a program for Quick Sort algorithm
(c) Write a program for linear search algorithm
(d) Write a program for displaying a sparse matrix
(e) Fibonacci numbers
(f) Factorial of a number

THEORY

a) HEAP SORT:

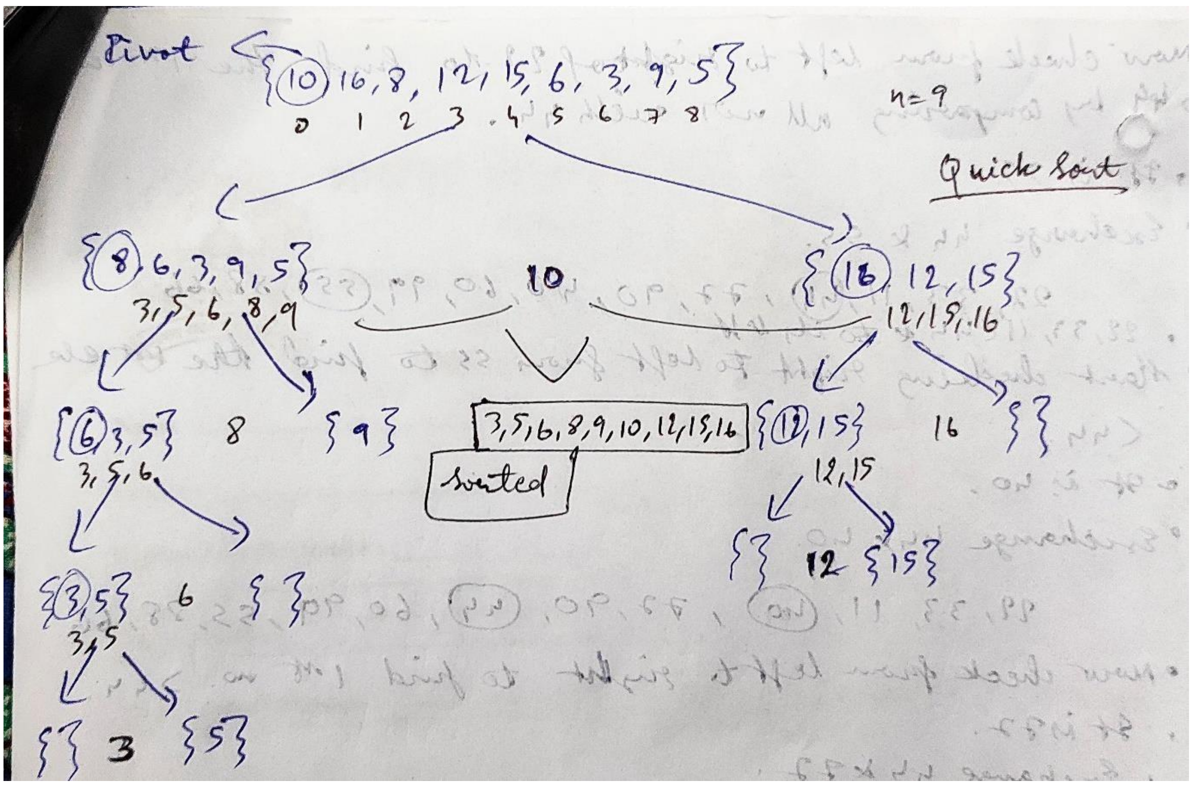
PROGRAM IN C++ to Heap sort a given array: 12, 11, 13, 5, 6, 7

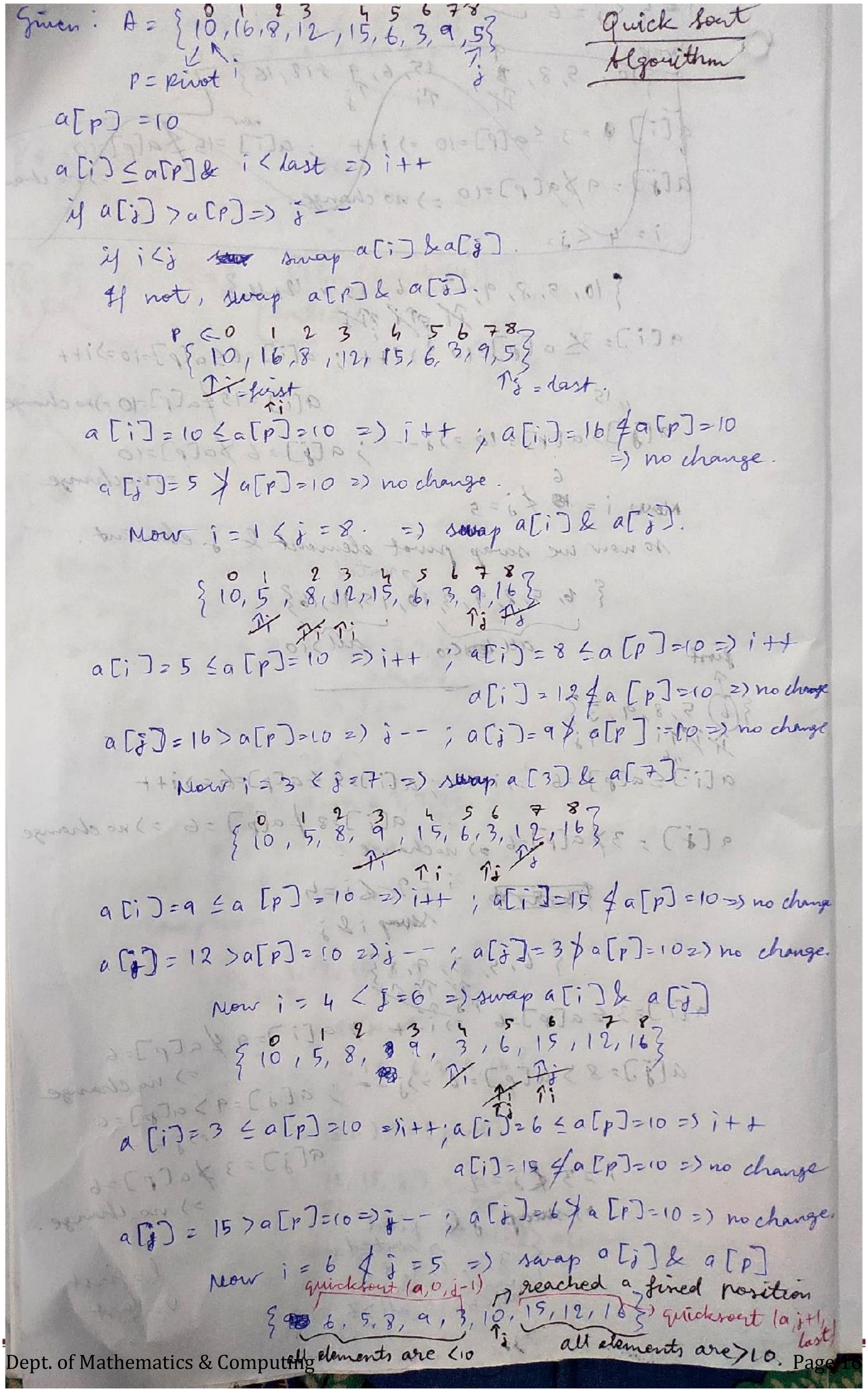
```
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        swap(arr[0], arr[i]);
        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

PROGRAM OUTPUT:

```
Sorted array is
5 6 7 11 12 13
```

b) QUICK SORT:





Quick Sort Algorithm

Given : $A = \left\{ \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 10, & 16, & 8, & 12, & 15, & 6, & 3, & 9, & 5 \end{array} \right\}$

p = pivot $\uparrow i$ $\uparrow j$

$a[p] = 10$

$a[i] \leq a[p]$ & $i <$ last $\Rightarrow i{+}{+}$

if $a[j] > a[p] \Rightarrow j{-}{-}$

if $i<j$ ~~swap~~ swap $a[i]$ & $a[j]$.

If not, swap $a[p]$ & $a[j]$.

$p \leftarrow \left\{ \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 10, & 16, & 8, & 12, & 15, & 6, & 3, & 9, & 5 \end{array} \right\}$

~~$\uparrow i$~~ = first $\uparrow i$ $\uparrow j$ = last.

$a[i] = 10 \leq a[p] = 10 \Rightarrow i{+}{+}$; $a[i] = 16 \nleq a[p] = 10$ $\Rightarrow$ no change.

$a[j] = 5 \ngtr a[p] = 10 \Rightarrow$ no change.

Now $i = 1 < j = 8$. $\Rightarrow$ ~~swap~~ swap $a[i]$ & $a[j]$.

$\left\{ \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 10, & 5, & 8, & 12, & 15, & 6, & 3, & 9, & 16 \end{array} \right\}$

~~$\uparrow i$~~ ~~$\uparrow i$~~ $\uparrow i$ $\uparrow j$ ~~$\uparrow j$~~

$a[i] = 5 \leq a[p] = 10 \Rightarrow i{+}{+}$; $a[i] = 8 \leq a[p] = 10 \Rightarrow i{+}{+}$

$a[i] = 12 \nleq a[p] = 10 \Rightarrow$ no change

$a[j] = 16 > a[p] = 10 \Rightarrow j{-}{-}$; $a[j] = 9 \ngtr a[p] = 10 \Rightarrow$ no change

Now $i = 3 < j = 7 \Rightarrow$ swap $a[3]$ & $a[7]$

$\left\{ \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 10, & 5, & 8, & 9, & 15, & 6, & 3, & 12, & 16 \end{array} \right\}$

~~$\uparrow i$~~ $\uparrow i$ $\uparrow j$ ~~$\uparrow j$~~

$a[i] = 9 \leq a[p] = 10 \Rightarrow i{+}{+}$; $a[i] = 15 \nleq a[p] = 10 \Rightarrow$ no change

$a[j] = 12 > a[p] = 10 \Rightarrow j{-}{-}$; $a[j] = 3 \ngtr a[p] = 10 \Rightarrow$ no change.

Now $i = 4 < j = 6 \Rightarrow$ swap $a[i]$ & $a[j]$

$\left\{ \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 10, & 5, & 8, & 9, & 3, & 6, & 15, & 12, & 16 \end{array} \right\}$

~~$\uparrow i$~~ ~~$\uparrow i$~~ $\uparrow j$ ~~$\uparrow j$~~ $\uparrow i$

$a[i] = 3 \leq a[p] = 10 \Rightarrow i{+}{+}$; $a[i] = 6 \leq a[p] = 10 \Rightarrow i{+}{+}$

$a[i] = 15 \nleq a[p] = 10 \Rightarrow$ no change

$a[j] = 15 > a[p] = 10 \Rightarrow j{-}{-}$; $a[j] = 6 \ngtr a[p] = 10 \Rightarrow$ no change.

Now $i = 6 \nless j = 5 \Rightarrow$ swap $a[j]$ & $a[p]$

quicksort (a, 0, j-1) reached a fixed position

$\{ \underbrace{6, 5, 8, 9, 3}_{\text{all elements are} < 10}, \overset{\uparrow j}{10}, \underbrace{15, 12, 16}_{\text{all elements are} > 10.} \}$ quicksort (a, j+1, last)

PROGRAM IN C to Quick sort:

```
#include <stdio.h>
 void quicksort(int a[40],int first,int last)
{
int i,j,pivot,temp;
if(first<last)
   {
       pivot=first;
       i=first;
       j=last;
       while(i<j)
       {
           while(a[i]<=a[pivot]&&i<last)
           i++;
           while(a[j]>a[pivot])
           j--;
if(i<j)
               {
               temp=a[i];
               a[i]=a[j];
               a[j]=temp;
               }
           temp=a[pivot];
           a[pivot]=a[j];
           a[j]=temp;
           quicksort(a,0,j-1);
           quicksort(a,j+1,last);
       }
   }
}
int main()
{
int a[40],n,i;
printf("How many elements are to be sorted?");
scanf("%d",&n);
printf("\nEnter array elements:\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
quicksort(a,0,n-1);
printf("\nArray after sorting:\n");
for(i=0;i<n;i++)
printf("%d ",a[i]);
return 0;
}
```

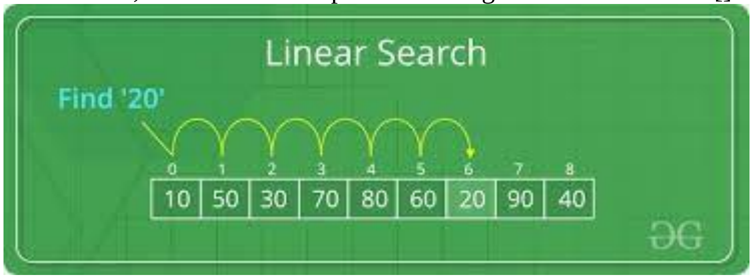
PROGRAM OUTPUT:

```
How many elements are to be sorted?4

Enter array elements:
22
```

```
46
71
3

Array after sorting:
3 22 46 71
```

c) THEORY (LINEAR SEARCH):

Given an array arr[] of n elements, Linear search helps to search a given element x in arr[].



Visualization of linear search technique

ALGORITHM:

```
Step 1: Start.
Step 2: Start from the left most element of arr[]
Step 3: One by one compare x with each element of arr[]
Step 4: If x matches with an element, return the index.
Step 5: If x doesn't match with any of elements, return -1
Step 6: Stop
```

PROGRAM IN C++

```
#include<iostream>
using namespace std;
int search(int arr[], int n, int x)
{
int i;
for (i = 0; i< n; i++)
if (arr[i] == x)
return i;
return -1;
}
int main(void)
{
int arr[] = { 2, 3, 4, 10, 40 };
int x = 4;
int n = sizeof(arr) / sizeof(arr[0]);
int result = search(arr, n, x);
(result == -1)? cout<<"Element is not present in array": cout<<"Element is present at index:\n" <<result;
return 0;
}
```

PROGRAM OUTPUT:

```
Element is present at index:
2
```

d) THEORY (SPARSE MATRIX):

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

ALGORITHM:

```
Step 1: Start
Step 2: Calculate Index of row, where non-zero element is located
Step 3: Calculate Index of column, where non-zero element is located
Step 4: CalculateValue of the non zero element located at index – (row,column)
Step 5: Print in the form of matrix
Step 6: Stop
```

PROGRAM IN C++

```
#include  <iostream>
using namespace std;
int main()
{
// Assume 4x5 sparse matrix
int sparseMatrix[4][5] =
{
{0 , 0 , 3 , 0 , 4 },
{0 , 0 , 5 , 7 , 0 },
{0 , 0 , 0 , 0 , 0 },
{0 , 2 , 6 , 0 , 0 }
};
int size = 0;
for (int i = 0; i< 4; i++)
for (int j = 0; j < 5; j++)
if (sparseMatrix[i][j] != 0)
size++;
// number of columns in compactMatrix (size) must be
// equal to number of non - zero elements in
// sparseMatrix
int compactMatrix[3][size];
// Making of new matrix
int k = 0;
for (int i = 0; i< 4; i++)
for (int j = 0; j < 5; j++)
if (sparseMatrix[i][j] != 0)
{
compactMatrix[0][k] = i;
compactMatrix[1][k] = j;
compactMatrix[2][k] = sparseMatrix[i][j];
k++;
}
for (int i=0; i<3; i++)
{
for (int j=0; j<size; j++)
cout<<compactMatrix[i][j]<<endl;
}
return 0;
}
```

PROGRAM OUTPUT:

```
0
0
1
1
3
3
2
4
2
3
1
2
3
4
5
7
2
6
```

e) THEORY (FIBONACCI NUMBERS):

In mathematics, the **Fibonacci numbers** form a sequence, the **Fibonacci sequence**, in which each number is the sum of the two preceding ones. The sequence commonly starts from 0 and 1, although some authors omit the initial terms and start the sequence from 1 and 1 or from 1 and 2. Starting from 0 and 1, the next few values in the sequence are:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,…

PROGRAM IN C++

```
#include  <iostream>
using namespace std;
int main()
      {
  int n1=0,n2=1,n3,i,number;
 cout<<"Enter the number of elements: ";
 cin>>number;
 cout<<n1<<" "<<n2<<" "; //printing 0 and 1
 for(i=2;i<number;i++) //loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  cout<<n3<<" ";
  n1=n2;
  n2=n3;
 }
   return 0;
```

```
    }
```

PROGRAM OUTPUT:

```
Enter the number of elements: 10
0 1 1 2 3 5 8 13 21 34
```

f) THEORY (FACTORIAL OF A NUMBER):

Factorial of a whole number 'n' is defined as the product of that number with every whole number till 1. For example, the factorial of 5 is 5×4×3×2×1, which is equal to 120. It is represented using the symbol '!' So, 120 is the value of 5!

PROGRAM IN C++

```
#include  <iostream>
using namespace std;
int main()
{
   int i,fact=1,number;
 cout<<"Enter any Number: ";
 cin>>number;
 for(i=1;i<=number;i++){
      fact=fact*i;
  }
  cout<<"Factorial of " <<number<<" is "<<fact<<endl;
  return 0;
}
```

PROGRAM OUTPUT:

```
Enter any Number: 5
Factorial of 5 is 120
```

QUESTIONS:

1. **Write programs in C/C++ to perform Heap / Quick Sort algorithm for the following array**
   12, 3, 40, 1, 60, 2, 99, 101

2. **Write a program in C/C++ to search the element 5 in the following array using linear search algorithm**
   20, 33, 12, 5, 8, 10

3. **Write a program in C/C++ to for displaying a sparse matrix for the following matrix**

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{pmatrix}$$

4. **Find out the value of 7! using a C/C++ program.**

EXPERIMENT 4

OBJECTIVE:

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size **MAX**)

a. **Push** an Element on to Stack
b. **Pop** an Element from Stack
c. Demonstrate how Stack can be used to check **Palindrome**
d. Demonstrate **Overflow** and **Underflow** situations on Stack
e. Display the status of Stack
f. Exit

Support the program with appropriate functions for each of the above operations

THEORY:

A stack is an abstract data type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack. A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.

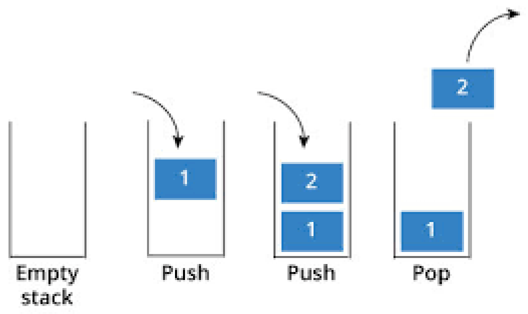
Basic Operations:

- **push() -** pushing (storing) an element on the stack.
- **pop() -** removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks;

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

Below given diagram tries to depict a stack and its operations −



ALGORITHM:

Step 1: Start.
Step 2: Initialize stack size MAX and top of stack -1.
Step 3: Push integer element on to stack and display the contents of the stack.
if stack is full give a message as ‘Stack is Overflow’.
Step 3: Pop element from stack along with display the stack contents.
if stack is empty give a message as ‘Stack is Underflow’.

Step 4: Check whether the stack contents are Palindrome or not.
Step 5: Stop.

PROGRAM IN C++

```
#include<iostream>
using namespace std;
#define MAX 10
int stack[MAX], item;
int ch, top=-1, count=0, status=0;
/*PUSH FUNCTION*/
void push(int stack[], int item)
{
if (top==(MAX-1))
cout<<"\n\nStack is Overflow";
else
{
stack[++top]=item;
status++;
}
}
int pop(int stack[])
{
int ret;
if(top==-1)
cout<<"\n\nStack is Underflow";
else
{
ret = stack[top--];
status--;
cout<<"\nPopped element is \n"<< ret;
}
return ret;
}
/* FUNCTION TO CHECK STACK IS
PALINDROME OR NOT */
void palindrome(int stack[])
{
int i, temp;
temp=status;
for(i=0; i<temp; i++)
{
if(stack[i]==pop(stack))
count++;
}
if(temp==count)
cout<<"\nStack contents are Palindrome";
else
cout<<"\nStack contents are not palindrome";
}
/*FUNCTION TO DISPLAY STACK*/
void display(int stack[])
```

```
{
int i;
cout<<"\nThe stack contents are:";
if(top==-1)
cout<<"\nStack is Empty";
else
{
for(i=top; i>=0; i--)
cout<<"\n"<< stack[i];
}
}
/*MAIN PROGRAM*/
int main()
{
do
{
cout<<"\n\n----MAIN MENU ---\n";
cout<<"\n 1. PUSH (Insert) in the Stack";
cout<<"\n 2. POP (Delete) from the Stack";
cout<<"\n 3. PALINDROME check using Stack";
cout<<"\n 4. Exit (End the Execution)";
cout<<"\n Enter Your Choice: ";
cin>>ch;
switch(ch)
{
case 1: cout<<"\nEnter a element to be pushed:";
cin>>item;
push(stack, item);
display(stack);
break;
case 2: item=pop(stack);
display(stack);
break;
case 3: palindrome(stack);
break;
case 4: return 0;
break;
default:
cout<<"\nEND OF EXECUTION";
}//end switch
}while (ch!= 4);
}
```

PROGRAM OUTPUT:

```
----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 1

Enter a element to be pushed:4
```

```
The stack contents are:
4

----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 1

Enter a element to be pushed:5

The stack contents are:
5
4

----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 1

Enter a element to be pushed:7

The stack contents are:
7
5
4

----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 2

Popped element is
7
The stack contents are:
5
4

----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 1

Enter a element to be pushed:4

The stack contents are:
4
5
```

```
4

----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 3

Popped element is
4
Popped element is
5
Popped element is
4
Stack contents are Palindrome

----MAIN MENU----

 1. PUSH (Insert) in the Stack
 2. POP (Delete) from the Stack
 3. PALINDROME check using Stack
 4. Exit (End the Execution)
 Enter Your Choice: 4
```

QUESTIONS:

1. **Write a program in C/C++ to insert the elements 12, 40, 19 into the following stack by Push function**
   3, 22, 34 (consider array size= 10)

2. **Write a program in C/C++ to delete 3 elements from the following stack by Pop function**
   30, 20, 40, 60, 90, 110

3. **Write a program in C/C++ to check whether the following stack is Palindrome or not**
   A S D F G F D S A

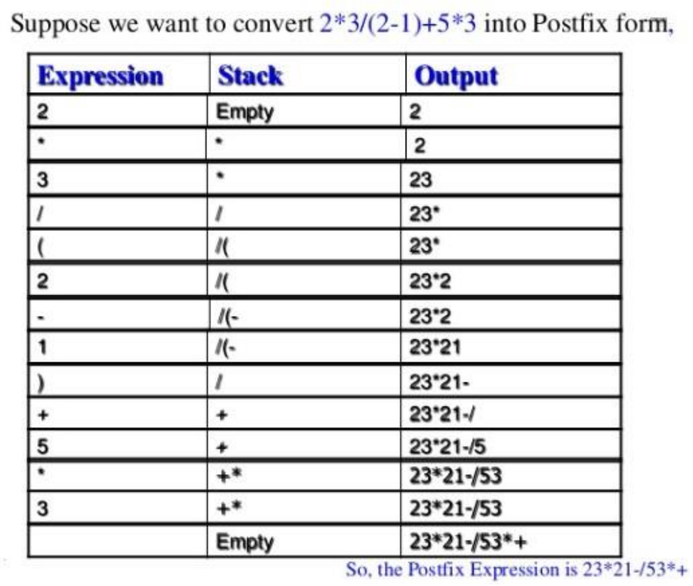
EXPERIMENT 5

OBJECTIVE:

Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (**Remainder**), ^ (**Power**) and **alphanumeric** operands.

THEORY:

Infix: Operators are written in-between their operands. Ex: X + Y
Prefix: Operators are written before their operands. Ex: +X Y
Postfix: Operators are written after their operands. Ex: XY+



Suppose we want to convert 2*3/(2-1)+5*3 into Postfix form,

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

So, the Postfix Expression is 23*21-/53*+

ALGORITHM:

```
Step 1: Start.
Step 2: Read an infix expression with parenthesis and without parenthesis.
Step 3: convert the infix expression to postfix expression.
Step 4: Stop
```

PROGRAM IN C

```
#include<stdio.h>
#include<stdlib.h>     #include<ctype.h>     #include<string.h>

#define SIZE 100

char stack[SIZE];
int top = -1;


void push(char x)
{
if(top >= SIZE-1)
{
printf("\nStack Overflow.");
}
else
{
top = top+1;
stack[top] = x;
}
}
char pop()
{
char item ;
if(top==-1)
{
printf("stack underflow: invalid infix expression");
getchar();
```

```
exit(1);
}
else
{
item = stack[top];
top = top-1;
return(item);
}
}
int priority(char x)
{
if(x=='(')
return 0;
if(x=='+'||x=='-')
return 1;
if(x=='*'||x=='/')
return 2;
if(x=='^')
return 3;
}
int main()
{
    char exp[20];
    char *e, x;
    printf("Enter the expression:");
    scanf("%s",exp);
    e=exp;
    while(*e!='\0')
    {
        if(isalnum(*e))
        printf("%c",*e);
        else if(*e=='(')
        push(*e);
        else if(*e==')')
        {
            while((x=pop())!='(')
            printf("%c",x);
        }
        else
        {
            while(priority(stack[top])>=priority(*e))
            printf("%c",pop());
            push(*e);
        }
        e++;
    }
        while(top!=-1)
    {
        printf("%c",pop());
    }
}
```

PROGRAM OUTPUT:

```
Enter the expression:(a+(b-c)*d)
abc-d*+
```

Manual check:

Symbol	Stack	Postfix
(	(	
a	(	a
+	(+	a
(	(+(	a
b	(+(	ab
-	(+(-	ab
c	(+(-	abc
)	(+	abc-
*	(+*	abc-
d	(+*	abc-d
)		abc-d*+

QUESTIONS:

1. **Write a program in C/C++ to convert the following Infix expression into Postfix expression a + b * c - (d / e + f ^ g ^ h)**

2. **Write a program in C/C++ to convert the following Infix expression into Postfix expression (6 + 2) * 5 – 8 / 4**

3. **Write a program in C/C++ to convert the following Infix expression into Postfix expression a + b * (c ^ d – e) ^ (f + g * h) – i**

EXPERIMENT 6

OBJECTIVE:

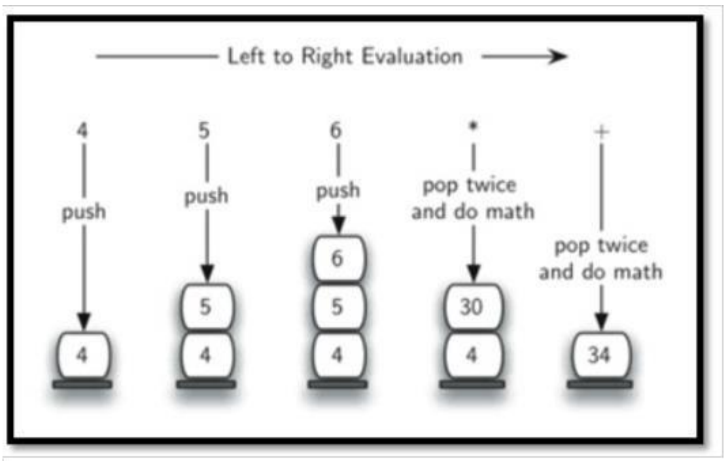
Design, Develop and Implement a Program in C for the following Stack Applications
a. Evaluation of **Suffix expression** with single digit operands and operators: **+, -, *, /, %,** ^
b. Solving **Tower of Hanoi** problem with **n** disks

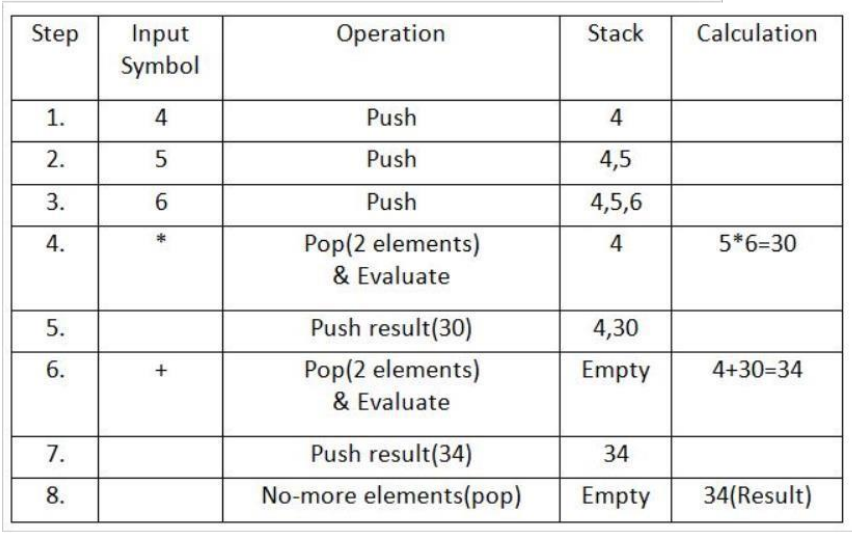
THEORY (SUFFIX/POSTFIX EXPRESSION EVALUATION):

Rules Of Evaluations

- While reading the expression from left to right, push the element in the stack if it is an operand.
- Pop the two operands from the stack, if the element is an operator and then evaluate it.
- Push back the result of the evaluation. Repeat it till the end of the expression.

Consider the following example:
To evaluate: 456*+





Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)

ALGORITHM:

Step 1: Start.
Step 2: Read the postfix/suffix expression.
Step 3: Evaluate the postfix expression based on the precedence of the operator.
Step 4: Stop.

PROGRAM IN C++:

PROGRAM 4A:

```
#include<iostream>
using namespace std;
#include<math>
#include<string>
double compute(char symbol, double op1, double op2)
{
        switch(symbol)
        {
                case '+': return op1 + op2;
                case' -': return op1 - op2;
                case '*': return op1 * op2;
                case '/': return op1 / op2;
                case '$':
                case '^': return pow(op1,op2);
                default: return 0;
        }
}
void main()
{
        double s[20], res, op1, op2;
        int top, i;
        char postfix[20], symbol;
        printf("\nEnter the postfix expression:\n");
        flushall();
        gets(postfix);
        top=-1;
        for(i=0; <strlen(postfix); i++)
        {
                symbol = postfix[i];
                if(isdigit(symbol))
                        s[++top] = symbol - '0';
                else
                {
                        op2 = s[top--];
                        op1 = s[top--];
                        res = compute(symbol, op1, op2);
                        s[++top] = res;
                }
        }
        res = s[top--];
        printf("\nThe result is : %f\n", res);
}
```

PROGRAM OUTPUT:

```
RUN1:

Enter the postfix expression:
23+
The result is: 5.000000
RUN2:
Enter the postfix expression:
23+7*
The result is: 35.000000
```

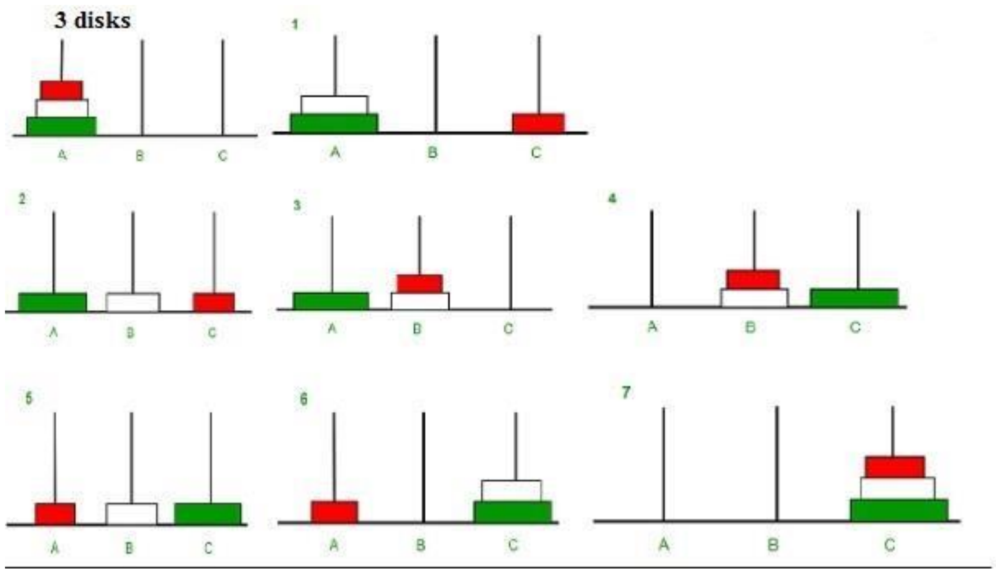
B. SOLVING TOWER OF HANOI PROBLEM WITH N DISKS

THEORY:

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of
another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2*n* - 1, where *n* is the number of disks.



Visualization of Tower of Hanoi problem

ALGORITHM:

Step 1: Start.
Step 2: Read N number of discs.
Step 3: Move all the discs from source to destination by using temp rod.
Step 4: Stop.

PROGRAM 4B:

```
#include<stdio.h>
#include<conio.h>
void TOWER(int, char[], char[], char[]);
void main()
{
   int N;
   clrscr;
   printf("Enter the number of disks to be transferred:");
   scanf("%d",&N);
   if(N<1)
   {
   printf("\nIncorrect value");
   }
   else
   {
      printf("\nThe following moves are required for n=%d\n\n",N);
      TOWER(N, "BEG", "AUX", "END");
   }
   getch();
}
void TOWER(int NUM, char A[50], char B[50], char C[50])
{
   if(NUM==1)
   {
      printf("%s->%s\t",A,C);
      return;
   }
   TOWER(NUM-1,A,C,B);
   printf("%s->%s\t",A,C);
   TOWER(NUM-1,B,A,C);
   return;
}
```

PROGRAM OUTPUT:

```
Enter the number of disks to be transferred:3

The following moves are required for n=3

BEG->END      BEG->AUX      END->AUX      BEG->END       AUX->BEG      AUX->END
BEG->END
```

QUESTIONS:

1. **Write a program in C/C++ to evaluate the following Postfix expression**
   2 3 1 * + 9 -

2. **Write a program in C/C++ to evaluate the following Postfix expression**
   5 3 + 8 2 - *

3. **Write a program in C/C++ to solve the Tower of Hanoi problem for 5 discs.**

EXPERIMENT 7

OBJECTIVE:

Design, Develop and Implement a menu driven Program in C for the following operations on **Circular QUEUE** of Characters

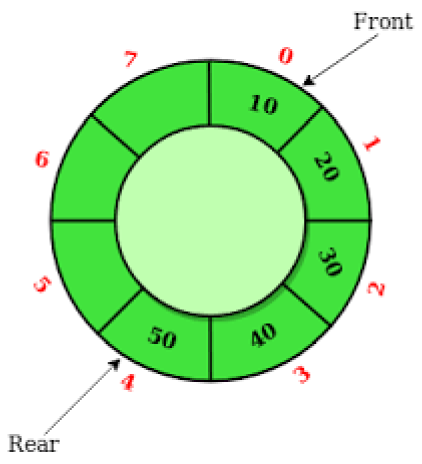
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit

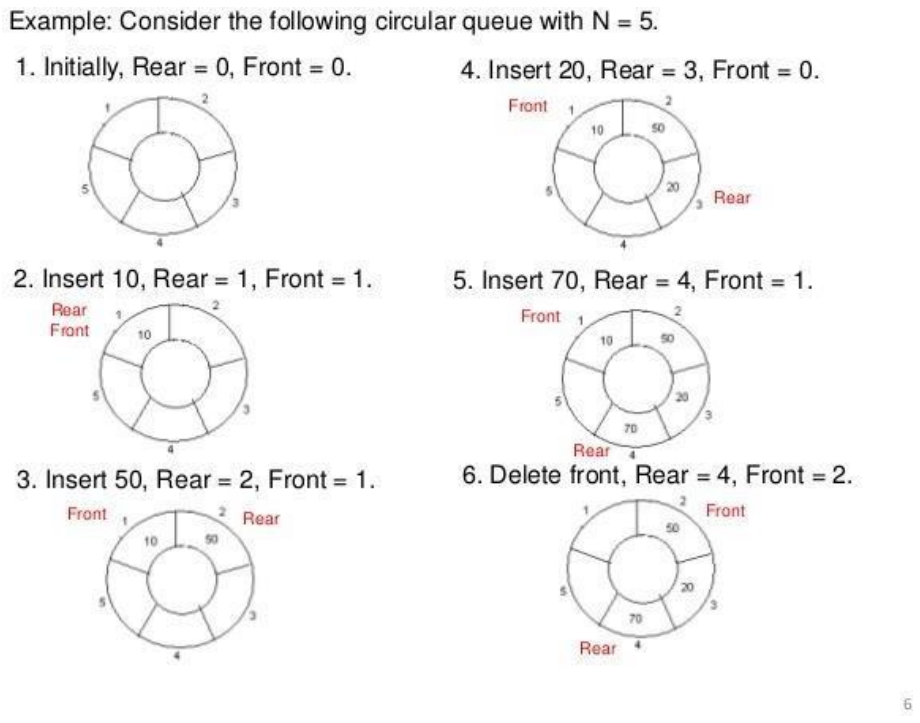
Support the program with appropriate functions for each of the above operations

THEORY:

Circular queue is a linear data structure. It follows FIFO principle. In **circular queue** the last node is connected back to the first node to make a **circle**. **Circular** linked list fallow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the **queue**. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

A circular queue looks like




Example: Consider the following circular queue with N = 5.

1. Initially, Rear = 0, Front = 0.
2. Insert 10, Rear = 1, Front = 1.
3. Insert 50, Rear = 2, Front = 1.
4. Insert 20, Rear = 3, Front = 0.
5. Insert 70, Rear = 4, Front = 1.
6. Delete front, Rear = 4, Front = 2.

ALGORITHM:

Step 1: Start.
Step 2: Initialize queue size to MAX.
Step 3: Insert the elements into circular queue. If queue is full give a message as ‘queue is overflow”
Step 4: Delete an element from the circular queue. If queue is empty give a message as ‘queue is underflow’.
Step 5: Display the contents of the queue.
Step 6: Stop.

PROGRAM IN C++

```
#include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1; /*The queue is empty*/ /*Deletion in front*/
int rear = -1; /*The queue is empty*/ /*Insertion in rear*/
void insert(int item)
```

```
{
if((front == 0 && rear == MAX-1) || (front == rear+1))
{
printf("Queue Overflow");
return;
}
if(front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
cqueue_arr[rear] = item ;
}
void delete()
{
if(front == -1)
{
printf("Queue Underflow");
return ;
}
printf("Element deleted from queue is : %d \n",cqueue_arr[front]);
if(front == rear)
{
front = -1;
rear=-1;
}
else
{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
}
void display()
  {
     int front_pos = front,rear_pos = rear;
     if(front == -1)
     {
        printf("Queue is empty");
        return;
     }
  printf("Queue elements :\n");
  if( front_pos <= rear_pos )
  while(front_pos <= rear_pos)
```

```
        {
           printf("%d ",cqueue_arr[front_pos]);
           front_pos++;
        }

else
        {
           while(front_pos <= MAX-1)
              {
              printf("%d ",cqueue_arr[front_pos]);
              front_pos++;
              }
           front_pos = 0;
           while(front_pos <= rear_pos)
              {
              printf("%d ",cqueue_arr[front_pos]);
              front_pos++;
              }
        }
   printf("\n");
   }
int main()
{
int choice,item;
do
{
printf("\n Circular Queue:");
printf("\n1.Insert");
printf("\n2.Delete");
printf("\n3.Display");
printf("\n4.Quit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1 :
printf("Input the element for insertion in queue : ");
scanf("%d", &item);
insert(item);
break;
case 2 :
delete();
break;
case 3:
display();
break;
case 4:
break;
default:
printf("Wrong choice");
}
}while(choice!=4);
```

```
return 0;
}
```

PROGRAM OUTPUT

```
 Circular Queue:
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Input the element for insertion in queue : 22

 Circular Queue:
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Input the element for insertion in queue : 56

 Circular Queue:
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Input the element for insertion in queue : 35

 Circular Queue:
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 1
Input the element for insertion in queue : 22

 Circular Queue:
1. Insert
2. Delete
3.Display
4.Quit
Enter your choice: 3
Queue elements  :
22 56 35 22

 Circular Queue:
1. Insert
2. Delete
3.Display
4.Quit
Enter your choice: 2
```

```
Element deleted from queue is : 22

 Circular Queue:
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice: 3
Queue elements :
56 35 22

 Circular Queue:
1. Insert
2.Delete
3.Display
4.Quit
Enter your choice: 4
```

QUESTIONS:

1. **Write a program in C/C++ to implement Circular Queue by inserting the following elements 9 , 20 , 2 , 10 , 13**

2. **Write a program in C/C++ to delete the element 30 from the following Circular Queue 12 , 10 , 20 , 30 , 130**

3. **Write a program in C/C++ to insert an element 60 after the element 20 in the following Circular Queue 100 , 70 , 20 , 40 , 80**

EXPERIMENT 8

OBJECTIVE:

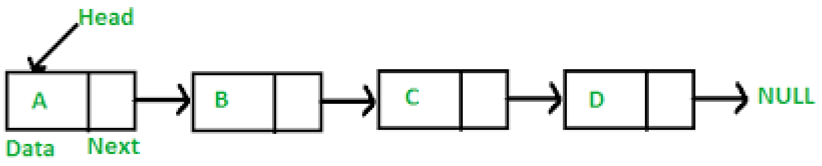
Design, Develop and Implement a menu driven Program in C/C++ for the following operations on **Singly Linked List (SLL)**

a. Create a **SLL**.
b. Insert at Beginning
c. Insert at Last
d. Insert at any random location
e. Delete from Beginning
f. Delete from Last
g. Delete node after specified location
h. Search for an element
i. Show
j. Exit

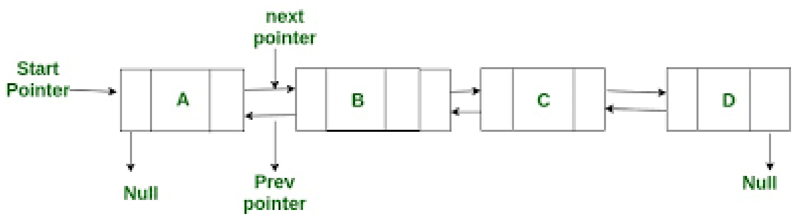
THEORY:

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.
They are a dynamic in nature which allocates the memory when required.
• Insertion and deletion operations can be easily implemented.

- Stacks and queues can be easily executed.
- Linked List reduces the access time.
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don’t need to know the size in advance.



Singly Linked List



Doubly Linked List

Types of Linked List:
Singly Linked List: Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.
Doubly Linked List: In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.

PROGRAM in C:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
   int choice =0;
   while(choice != 9)
   {
      printf("\n\n*********Main Menu*********\n");
      printf("\nChoose one option from the following list ...\n");
      printf("\n===============================================\n");
```

```
        printf("\n1.Insert at Beginning\n2.Insert at Last\n3.Insert at any random location\n4.Delete from
Beginning\n5.Delete from Last\n6.Delete node after specified location\n7.Search for an
element\n8.Show\n9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
switch(choice)
        {
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
            randominsert();
            break;
            case 4:
            begin_delete();
            break;
            case 5:
            last_delete();
            break;
            case 6:
            random_delete();
            break;
            case 7:
            search();
            break;
            case 8:
            display();
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice.");
        }
    }
}
//Insert at the Beginning
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }
}
//Insert at Last
```

```
void lastinsert()
{
   struct node *ptr,*temp;
   int item;
   ptr = (struct node*)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
else
   {
      printf("\nEnter value?\n");
      scanf("%d",&item);
      ptr->data = item;
      if(head == NULL)
      {
         ptr -> next = NULL;
         head = ptr;
         printf("\nNode inserted");
      }
      else
      {
         temp = head;
         while (temp -> next != NULL)
         {
            temp = temp -> next;
         }
         temp->next = ptr;
         ptr->next = NULL;
         printf("\nNode inserted");
        }
   }
}
//Insert at random position
void randominsert()
{
   int i,loc,item;
   struct node *ptr, *temp;
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
else
   {
      printf("\nEnter element value");
      scanf("%d",&item);
      ptr->data = item;
      printf("\nEnter the location after which you want to insert ");
      scanf("\n%d",&loc);
      temp=head;
      for(i=0;i<loc;i++)
      {
         temp = temp->next;
         if(temp == NULL)
         {
            printf("\nCan't insert\n");
            return;
         }
        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
    }
}
//Delete at the Beginning
void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }
}
//Delete at Last
void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }
else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}
//Delete at random position
void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;

        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted node %d ",loc+1);
}
//Search for an element
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("Item not found\n");
        }
    }
}
//Display the elements
void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values.........\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}
```

PROGRAM OUTPUT:

```
*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

1

Enter value

10

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element
```

```
8.Show

9.Exit


Enter your choice?

1

Enter value

20

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

1

Enter value

30

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location
```

```
4.Delete from Beginning
5.Delete from Last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit

Enter your choice?
8
printing values . . . . .
30
20
10
*********Main Menu*********
Choose one option from the following list ...
===============================================
1.Insert at Beginning
2.Insert at Last
3.Insert at any random location
4.Delete from Beginning
5.Delete from Last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit
Enter your choice?
2
Enter value?
40
Node inserted
*********Main Menu*********
Choose one option from the following list ...
```

```
===============================================
1.Insert at Beginning
2.Insert at Last
3.Insert at any random location
4.Delete from Beginning
5.Delete from Last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit

Enter your choice?
2
Enter value?
50
Node inserted
*********Main Menu*********
Choose one option from the following list ...
===============================================
1.Insert at Beginning
2.Insert at Last
3.Insert at any random location
4.Delete from Beginning
5.Delete from Last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit

Enter your choice?
8
printing values . . . . .
```

```
30

20

10

40

50

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

7

Enter item which you want to search?

10

item found at location 3

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location
```

```
7.Search for an element

8.Show

9.Exit


Enter your choice?

3

Enter element value 99

Enter the location after which you want to insert 2

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

8

printing values . . . . .

30

20

10

99

40

50

*********Main Menu*********
```

```
Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

4

Node deleted from the begining ...

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

8

printing values . . . . .

20
```

```
10

99

40

50

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

5

Deleted Node from the last ...

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit
```

```
Enter your choice?

8

printing values . . . . .

20

10

99

40

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

6

 Enter the location of the node after which you want to perform deletion

2

Deleted node 3

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location
```

```
4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

8

printing values . . . . .

20

10

40

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert at Beginning

2.Insert at Last

3.Insert at any random location

4.Delete from Beginning

5.Delete from Last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit


Enter your choice?

9
```

EXPERIMENT 9

OBJECTIVE:

Design, Develop and Implement a menu driven Program in C/C++ for the following operations on **Doubly Linked List (DLL)**

a. Create a **DLL**.
b. Print all the elements in DLL in forward traversal order
c. Print all elements in DLL in reverse traversal order
d. Exit

THEORY:

Doubly Linked List: In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence. In computer science, a **doubly linked list** is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called *links*, that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders. A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node. The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

PROGRAM IN C:

```
/* Doubly Linked List implementation */
#include<stdio.h>
#include<stdlib.h>

struct Node  {
	int data;
	struct Node* next;
	struct Node* prev;
};

struct Node* head; // global variable - pointer to head node.

//Creates a new Node and returns pointer to it.
struct Node* GetNewNode(int x) {
	struct Node* newNode
		= (struct Node*)malloc(sizeof(struct Node));
	newNode->data = x;
	newNode->prev = NULL;
	newNode->next = NULL;
	return newNode;
}
//Inserts a Node at head of doubly linked list
void InsertAtHead(int x) {
	struct Node* newNode = GetNewNode(x);
	if(head == NULL) {
		head = newNode;
		return;
	}
	head->prev = newNode;
```

```
        newNode->next = head;
        head = newNode;
}

//Inserts a Node at tail of Doubly linked list
void InsertAtTail(int x) {
        struct Node* temp = head;
        struct Node* newNode = GetNewNode(x);
        if(head == NULL) {
                head = newNode;
                return;
        }
        while(temp->next != NULL) temp = temp->next;
// Go To last Node
        temp->next = newNode;
        newNode->prev = temp;
}
//Prints all the elements in linked list in forward traversal order
void Print() {
        struct Node* temp = head;
        printf("Forward: ");
        while(temp != NULL) {
                printf("%d ",temp->data);
                temp = temp->next;
        }
        printf("\n");
}

//Prints all elements in linked list in reverse traversal order.
void ReversePrint() {
        struct Node* temp = head;
        if(temp == NULL) return; // empty list, exit
        // Going to last Node
        while(temp->next != NULL) {
                temp = temp->next;
        }
        // Traversing backward using prev pointer
        printf("Reverse: ");
        while(temp != NULL) {
                printf("%d ",temp->data);
                temp = temp->prev;
        }
        printf("\n");
}

//main function
int main() {

        /*Driver code to test the implementation*/
        head = NULL; // empty list. set head as NULL.

        // Calling an Insert and printing list both in forward as well as reverse direction.
                InsertAtTail(2); Print(); ReversePrint();
                InsertAtTail(4); Print(); ReversePrint();
                InsertAtTail(6); Print(); ReversePrint();
                InsertAtHead(1); Print(); ReversePrint();
                InsertAtHead(9); Print(); ReversePrint();


}
```

PROGRAM OUTPUT:

```
Forward: 1 2 4 6
Reverse: 6 4 2 1
Forward: 9 1 2 4 6
Reverse: 6 4 2 1 9
```

QUESTIONS:

1. **Write a program in C/C++ to implement Doubly Linked List by inserting the following data**
   1↔ 5 ↔ 8 ↔ 2 ↔ 7

2. **Write a program in C/C++ to insert an element 3 before the 1st node (having data: 1) in the following Doubly Linked List (Try yourself)**
   1↔ 7 ↔ 5 ↔ 2 ↔ 8

3. **Write a program in C/C++ to delete the last node from the following DLL and also show how the DLL can be used as Double Ended Queue (Try yourself)**
   10↔ 5 ↔ 2 ↔ 3 ↔ 1

EXPERIMENT 10

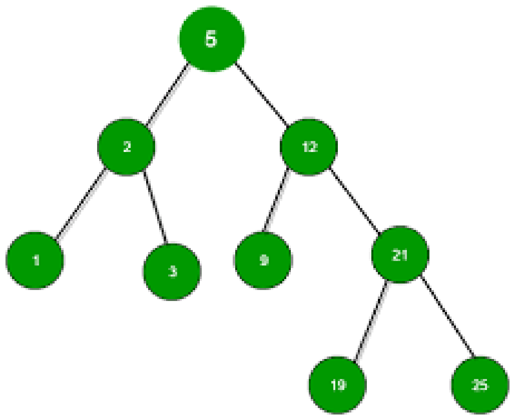
OBJECTIVE:

Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers

a. Create a BST of **N** Integers: 8, 10, 3, 1, 6, 14, 7
a. Traverse the BST in Inorder
b. Traverse the BST in Preorder
c. Traverse the BST in and Post Order

THEORY:

A **binary search tree** (BST) is a **tree** in which all nodes follows the below mentioned properties
• The left sub-**tree** of a node has key less than or equal to its parent node's key.
• The right sub-**tree** of a node has key greater than or equal to its parent node's key.
Thus, a binary search tree (BST) divides all its sub-trees into two segments; *left* sub-tree and *right* sub-tree and can be defined as
left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)



Example of BST

Following are basic primary operations of a tree which are following.

- **Search** − search an element in a tree.
- **Insert** − insert an element in a tree.
- **Preorder Traversal** − traverse a tree in a preorder manner.
- **Inorder Traversal** − traverse a tree in an inorder manner.
- **Postorder Traversal** − traverse a tree in a postorder manner.

Node definition: Define a node having some data, references to its left and right child nodes.

```
struct node
{
        int data;
        struct node *leftChild;
        struct node *rightChild;
};
```

ALGORITHM:

```
Step 1: Start.
Step 2: Create a Binary Search Tree for N elements.
Step 3: Traverse the tree in inorder.
Step 4: Traverse the tree in preorder
Step 6: Traverse the tree in postorder.
```

PROGRAM IN C:

```
#include<stdio.h>
#include<stdlib.h>

struct Node /*Structure of a basic Binary Tree*/
{
int data;
struct Node *left; /*Left child*/
struct Node *right; /*Right child*/
};

struct Node* createNode(int x) /*Creates a node in which we can insert values*/
{
struct Node* temp=(struct Node *)malloc(sizeof(struct Node));
temp->data=x;
temp->left=NULL;
temp->right=NULL;
return temp;
}
struct Node* insertElement(struct Node* root, int x)
{
   if(root==NULL) /*Tree is empty*/
   {
      struct Node* temp=createNode(x); /*Crate a new node. Return its address and update it.*/
      return temp;
   }
   if(root->data>=x) /*Value to be inserted is smaller or equal*/
   {
      root->left=insertElement(root->left,x);
   }
   else /*Value to be inserted is larger*/
   {
      root->right=insertElement(root->right,x);
   }
   return root;
}
void inorder(struct Node* root)
```

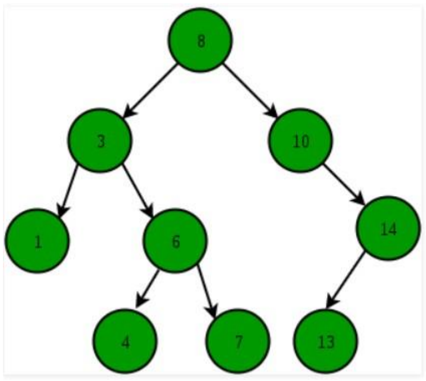
```
{
   if(root==NULL)
   return;
   inorder(root->left); /*Start checking form the left sub-tree*/
   printf("%d ",root->data); /*Print value of current node*/
   inorder(root->right); /*Check right sub-tree*/
   return;
}
void preorder(struct Node *root)
{
if(root!=NULL)
{
printf("%d ",root->data);
preorder(root->left);
preorder(root->right);
}
}
void postorder(struct Node *root)
{
if(root!=NULL)
{
postorder(root->left);
postorder(root->right);
printf("%d ", root->data);
}
}
int main()
{
   struct Node* root=NULL;
   root=insertElement(root,8);
   root=insertElement(root,10);
   root=insertElement(root,3);
   root=insertElement(root,1);
   root=insertElement(root,6);
   root=insertElement(root,14);
   root=insertElement(root,7);
   printf("Inorder traversal:\n");
   inorder(root);
   printf("\nPreorder traversal:\n");
   preorder(root);
   printf("\nPostorder traversal:\n");
   postorder(root);
   return 0;
}
```

PROGRAM OUTPUT:

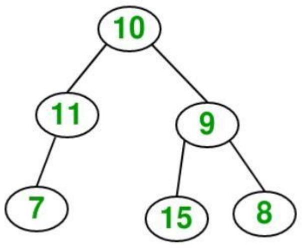
```
Inorder traversal:
1 3 6 7 8 10 14
Preorder traversal:
8 3 1 6 7 10 14
Postorder traversal:
1 7 6 3 14 10 8
```

QUESTIONS:

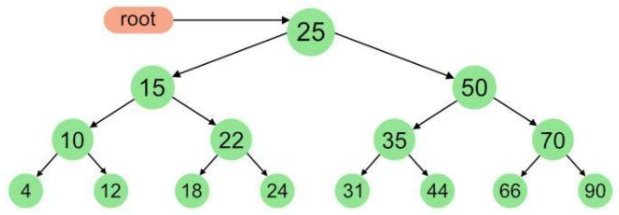
1. **Write a program in C/C++ to search the element 6 in the following binary tree (try yourself).**



2. **Write a program in C/C++ to insert the element 12 in the following binary tree (try yourself).**



3. **Write a program in C/C++ to perform Inorder, Preorder and Postorder tree traversal for the following binary tree.**



Note: If you want to perform Inorder, Preorder, Postorder, Searching, and Deletion in a single program, use switch case.

EXPERIMENT 11

OBJECTIVE:

Design, Develop and Implement a Program in C for the following operations on **Graph(G)** of Cities

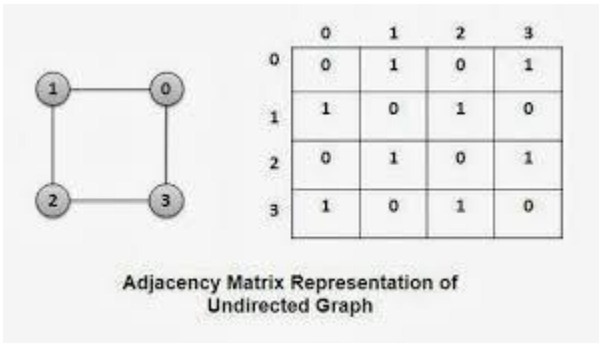
a. Create a Graph of **N** cities using Adjacency Matrix.
b. Print all the nodes **reachable** from a given starting node in a digraph using **BFS** method
c. Print all the nodes **reachable** from a given starting node in a digraph using **DFS** method
.

THEORY:

Adjacency Matrix

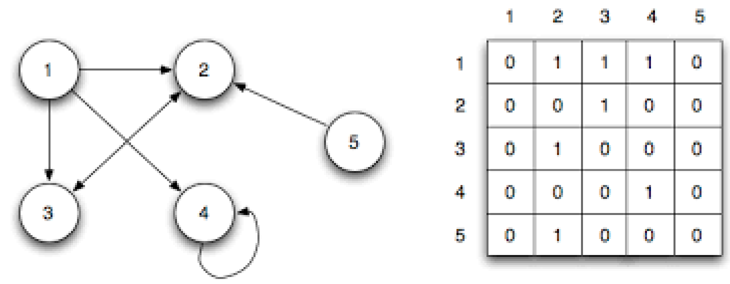
In **graph** theory, computer science, an **adjacency matrix** is a square **matrix** used to **represent** a finite **graph**. The elements of the **matrix** indicate whether pairs of vertices are adjacent or not in the **graph**. In the special case of a finite simple **graph**, the **adjacency matrix** is a (0, 1)-**matrix** with zeros on its diagonal. A graph G = (V, E) where v= {0, 1, 2, . . .n-1} can be represented using two dimensional integer array of size n x n.
a[20][20] can be used to store a graph with 20 vertices.
a[i][j] = 1, indicates presence of edge between two vertices i and j.
a[i][j] = 0, indicates absence of edge between two vertices i and j.
• A graph is represented using square matrix.
• Adjacency matrix of an undirected graph is always a symmetric matrix, i.e. an edge (i, j) implies the edge (j,i).

• Adjacency matrix of a directed graph is never symmetric, adj[i][j] = 1 indicates a directed edge from vertex i to vertex j.



	0	1	2	3
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0

Adjacency matrix representation of unidirected graph



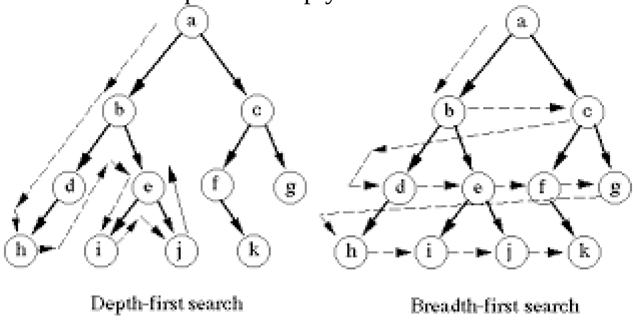
	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	1	0
5	0	1	0	0	0

Adjacency matrix representation of directed graph

BFS

Breadth-first search (**BFS**) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbor nodes first, before moving to the next level neighbors. Breadth First Search algorithm(BFS) traverses a graph in a breadth wards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

It employs following rules.
• **Rule 1** − Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
• **Rule 2** − If no adjacent vertex found, remove the first vertex from queue.
• **Rule 3** − Repeat Rule 1 and Rule 2 until queue is empty.

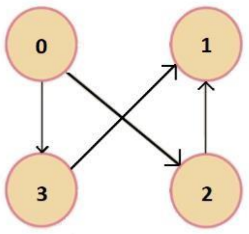


DFS

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

Depth-first search, or **DFS**, is a way to traverse the graph. Initially it allows visiting vertices of the graph only, but there are hundreds of algorithms for graphs, which are based on DFS. Therefore, understanding the principles of depth-first search is quite important to move ahead into the graph theory. The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack

PROGRAM IN C for BFS for



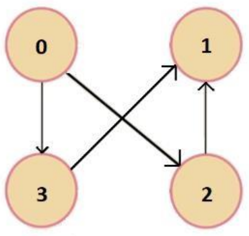
```
#include<stdio.h>
#include<stdlib.h>

struct queue
{
   int size;
   int  f;
   int r;
   int* arr;
};

int isEmpty(struct queue *q){
   if(q->r==q->f){
      return 1; //True
   }
   return 0; //False
}

int isFull(struct queue *q){
   if(q->r==q->size-1){
      return 1; //True
   }
   return 0; //False
}
void enqueue(struct queue *q, int val){
   if(isFull(q)){
      printf("This Queue is full\n");
   }
   else{
      q->r++;
      q->arr[q->r] = val;
}
}

int dequeue(struct queue *q){
   int a = -1;
   if(isEmpty(q)){
      printf("This Queue is empty\n");
   }
   else{
      q->f++;
      a = q->arr[q->f];
   }
   return a;
}

int main()
{
   struct queue q;
```

```
    q.size = 400;
    q.f = q.r = 0;
    q.arr = (int*) malloc(q.size*sizeof(int));

    // BFS implementation
    int node;
    int i=0;
    int visited[4]={0,0,0,0};
    int a[4][4]={
        {0,0,1,1},
        {0,0,0,0},
        {0,1,0,0},
        {0,1,0,0}
    };
 printf("%d",i);
    visited[i]=1;
    enqueue(&q,i); //Enqueue i for exploration
    while(!isEmpty(&q))
    {
        int node=dequeue(&q);
        for(int j=0;j<4;j++)
        {
            if(a[node][j]==1 && visited[j]==0)
            {
                printf("%d",j);
                visited[j]=1;
                enqueue(&q,j);
            }
        }
    }
}
```

PROGRAM OUTPUT:

```
0231
```

PROGRAM IN C for DFS for



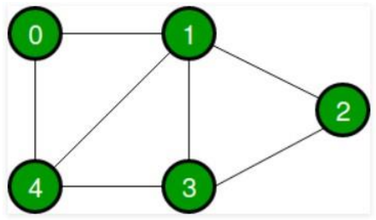
```
#include<stdio.h>
int G[10][10],visited[10],n;    //n is no of vertices and graph is sorted in array G[10][10]
void main()
{
    int i,j;
    printf("Enter number of vertices:");
scanf("%d",&n); //read the adjacency matrix
printf("\nEnter adjecency matrix of the graph:");
for(i=0;i<n;i++)
       for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
   for(i=0;i<n;i++)
        visited[i]=0; //visited is initialized to zero
   DFS(0);
}
void DFS(int i)
{
   int j;
printf("\n%d",i);
   visited[i]=1;
for(j=0;j<n;j++)
      if(!visited[j]&&G[i][j]==1)
          DFS(j);
}
```

PROGRAM OUTPUT:

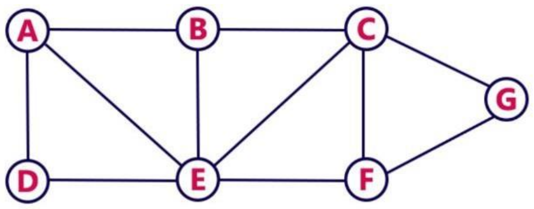
```
0
2
1
3
```

QUESTIONS:

1. **Write a program in C/C++ to create the following Graph using Adjacency Matrix.**



2. **Write a program in C/C++ to perform DFS traversal for the following graph**



3. **Write a program in C/C++ to perform BFS traversal for the following graph**