Vision of Institute:

"Empowerment through Knowledge".

Mission of Institute:

To educate the students to transform them as professionally competent and quality conscious engineers by providing conducive environment for teaching, learning, and overall personality development, culminating the institute into an international seat of excellence.

Vision of Department

Developing globally competent IT professional for sustainable growth of humanity.

Mission of Department

1. To empower students for developing systems and innovative products for solving real life problems.
2. To build strong foundation in computing knowledge that enables self-development entrepreneurship and Intellectual property.
3. To develop passionate IT professional by imparting global competencies and skills for serving society.

Program Specific Outcomes:

1. ***Placement*:** Enhance employability of students through design methodologies, application development tools and building knowledge base in Networking, Database, Web applications, Security etc.

2. ***Higher Studies:*** To prepare students to excel in postgraduate programmers or to succeed in industry by using knowledge of basic programming languages: syntax and semantics, problem analysis etc.

3. ***Project Development*:** Acquire ability to identify and solve IT problems by undertaking a team project and to improve communication skill and ability to work in team. Develop confidence for establishing their own IT companies in future.

4. ***Life-long Learning*:** To groom the students to Design, implement, and assess an IT based or computer based system to meet desired needs of society.

Program Educational Objectives:

1) ***Core Competence****:* To provide students with a solid foundation in mathematical, scientific and engineering fundamentals required to solve engineering problems and also to pursue higher studies.

2) ***Breadth****:* To train students with good scientific and engineering breadth so as to comprehend, analyse, design, and create novel products and solutions for the real life problems.

3) ***Professionalism****:* To inculcate in students professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, and an ability to relate engineering issues to broader social context.

4) ***Learning Environment****:* To provide student with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the life-long learning needed for a successful professional career.

5) ***Attainment***: To prepare students to excel in research or to succeed in industry/technical profession through global, rigorous education and research and to become future entrepreneurs.

Name-Harshita Totala Roll No- SEITA14

Div- A

Assignment-1
Searching and Sorting

AIM:

Consider a student database of SEIT class (at least 15 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of structure)

a) Design a roll call list, arrange list of students according to roll numbers in ascending order (Use Bubble Sort)

b) Arrange list of students alphabetically. (Use Insertion sort)

c) Arrange list of students to find out first ten toppers from a class. (Use Quick sort)

d) Search students according to SGPA. If more than one student having same SGPA, then print list of all students having same SGPA.

e) Search a particular student according to name using binary search without recursion. (all the student records having the presence of search key should be displayed**)**

THEORY:

Introduction Sorting:

Sorting means bringing some orderliness in the data, which are otherwise not ordered. For example, to print the list of students in ascending order of their birth dates, we need to sort the student information in ascending order of date of birth (student information stored in a file may not be in this order) before generating the print report. In most of the computer applications we need to sort the data either in ascending or descending order as per the requirements of the application. Sorting techniques are broadly classified as internal sort techniques and external sort techniques.

Internal sort: Internal sort is also known as Memory Sort, and it is used to sort the small volume of data in computer memory. Following are the some of internal or memory sort techniques.

1. Bubble sort
2. Selection sort
3. Insertion sort
4. Bucket sort
5. Radix sort
6. Quick sort
7. Merge sort

Name-Harshita Totala Roll No- SEITA14

Div- A

External sort:

The term External sort is referred to sorting of voluminous data. For example, to create a temporary file sorted on some key from the input file containing thousands of records, we need to divide the input file in small partitions, which can be sorted on the key by using any one of the known internal sort techniques and then merging those partitions on the sorted key to generate new partition of bigger size and this process is repeated until we get only one large sorted partition of size equal to the number of records in input file. In practice most of the times it is not possible to bring the entire data from input file to the memory of computer for sorting and we have to use some secondary storage to store the intermediate results (partitions). Thus in external sort input and output sorted files are stored on secondary storage i.e. disk.

Bubble sort:

Basic step:

From the remaining unsorted list, compare the first element with other elements and interchange them if they are not in the required order.

For the list containing 'n' elements,

Number of passes : n - 1

Time complexity : $O(n^2)$

Space complexity : No extra space required

Worst case : List in reverse order

Comparisons : $O(n^2)$ Exchanges : $O(n^2)$

Best case : Already sorted list

Comparisons: $O(n^2)$, Exchanges: 0

Average case : List on random order

Comparisons : $O(n^2)$, Exchanges : $O(n^2)$

Comments : Good for small value of 'n'

Name-Harshita Totala Roll No- SEITA14

Div- A

ALGORITHM:

```
 begin BubbleSort(list)

   for all elements of list       if list[i] > list[i+1]
swap(list[i], list[i+1])       end if    end for

   return list

end BubbleSort
```

Insertion sort:

Basic step:

To insert a record R into sequence of ordered records R1, R2, …, Ri, in such a way that resulting sequence of size (i+1) is also ordered.

For the list containing 'n' elements,

Number of passes : 1

Time complexity : O(n*n)

Space complexity : No extra space required

Worst case : List in reverse order
Comparisons : O(n*n), Push downs : O(n*n)

Best case : Already sorted list
Comparisons : O(n), Push downs : 0

Average case : List on random order
Comparisons : O(n*n), Push downs : O(n*n)

Comments : Good if the input list has very few entries Left Out of Order (LOO)

ALGORITHM:

Step 1 – If it is the first element, it is already sorted. return

Name-Harshita Totala Roll No- SEITA14

Div- A

```
1;
Step 2 – Pick next element
Step 3 – Compare with all elements in the sorted sub-list Step
4 – Shift all the elements in the sorted sub-list that is
greater than the          value to be sorted
Step 5 – Insert the value
Step 6 – Repeat until list is sorted
```

QUICK SORT : Quick sort scheme developed by C.A.R. Hoare, has the best average behaviour in terms of Time & Space Complexity among all the internal sorting methods.

BASIC STEP : In Quick Sort the key Ki (pivot – generally the first element in an array) controlling the process is placed at the right position with respect to the elements in an array i.e. if key Ki is placed at position s(i) then

$$Kj < Ks(i) \text{ for } j < s(i) \text{ and}$$

$$Kj >= Ks(i) \text{ for } j > s(i)$$

Thus after positioning of the element Ki has been made, the original array is partitioned into two sub arrays, one consisting of elements K1 to (Ks(i) – 1) and the other partition (Ks(i) + 1) to Kn. All the elements in first sub array are less than Ks(i) & those in the second sub array are greater or equal to Ks(i). These two sub arrays can be sorted independently

DIVIDE & CONQUER STRATEGY :

From the above description of Quick Sort, it is evident that Quick Sort is a good example of **Divide & Conquer** strategy for solving the problem. In Quick Sort, we divide the array of items to be sorted into two partitions and then call the same procedure recursively to sort the two partitions. Thus we **divide** the problem in two smaller problems and **conquer** by solving the smallest problem, in this case sorting of partition containing only one element at the end, which is trivial one. Thus Quick Sort is the best example of divide & conquer strategy, where the problem is solved (conquered) by dividing the original problem into two smaller problems and then applying the same strategy recursively until the problem is so small that it can be solved trivially. Thus the conquer part of the Quick Sort looks like this.

Name-Harshita Totala Roll No- SEITA14

Div- A

Initial Step

< Pivot1	Pivot1	>= Pivot1

Next Step

< Pivot2	Pivot2	>= Pivot2	Pivot1	>= Pivot1

EXAMPLE :

Pass No : **< -------- Input Array Elements -------- > Pivot m n i j**

Position : 0 1 2 3 4 5 6 7 8 9

Pass 1 : **2 1 22 34 4 7 30 1 21 20** 2 0 9 1 9

Pass 2 : **1 1** 2 34 4 7 30 22 21 20 1 0 1 1 1

Pass 3 : 1 1 2 **34 4 7 30 22 21 20** 34 3 9 4 9

Pass 4 : 1 1 2 **20 4 7 30 22 21** 34 20 3 8 4 8

Pass 5 : 1 1 2 **7 4** 20 30 22 21 34 7 3 4 4 4

Pass 6 : 1 1 2 4 7 20 **30 22 21** 34 30 6 8 7 8

Name-Harshita Totala Roll No- SEITA14

Div- A

Pass 7 : 1 1 2 4 7 20 **21 22** 30 34 30 6 7 7 7

Sorted Array : 1 1 2 4 7 20 21 22 30 34

1) Time Complexity :

a) Worst Case : Already sorted array is the worst case for Quick sort. Let us assume that there are 'n' elements in an array, then the number of elements in each partition for each pass will be,

Pass No	***Partitions***	***No. of comparisions***
Pass 1	: [0], [n - 1]	(n - 1)
Pass 2	: [0], [0], [n - 2]	(n - 2)
Pass 3	: [0],.[0], [0], [n - 3]	(n - 3)
.		.
.		.
.		.
Pass (n-1)	: [0],.[0], [0], …[0],[1]	1

Total number of comparisons would be = (n - 1) + (n - 2) + (n - 3) + …+ 1

= (n - 1) (n) / 2

= (n**2 / 2) – (n / 2)

= O (n**2) (ignoring lower order terms)

b) Best Case : Best case of Quick Sort would be the case where input elements in the array are such that in every pass **pivot** element is positioned at the middle such that the partition is divided in two partitions of equal size Then for a large value of n, the number of comparisons in each pass would be,

After 1st Pass : n comparisons + number of comparisons required to sort two

Partitions of size n / 2.

= n + 2 * [n / 2]

After 2nd Pass = n + 2 * [n / 2 + 2 * (n / 4)]

= n + n + 4*[n / 4]

= 2n + 4*[n / 4]

After 3rd Pass = 2n + 4* [n / 4 + 2 * (n / 8)]

= 2n + n + 8*[n / 8]

= 3n + 8*[n / 8] = (log 8) * 8 + 8*[8/8] for 8 elements array

= **8*log8** since number of comparisons required to sort partition of size 1 would be zero.

. . .

. . .

. . .

After nth Pass = n*log(n)

c) Average case : Experimental results show that number of comparisons required to sort 'n' element array using Quick Sort are **O(nlogn)** to an average.

2) Space Complexity : In recursive quick sort, additional space for stack is required, which is of the order of **O(logn)** for **best and average case** and **O(n)** for **worst case.**

ALGORITHM:

```
void quicksort (int a[], int p, int q)
/* sorts the elements a[p]…,a[q] which reside in the global array a[1:n] into ascending
order.*/ {
1. declare int j;
2. if(left >right)
   return;
```

3. j=partition(a, left ,right);
4. quicksort(a, left,j-1);
5. quicksort(a, j+1,right);

}

int part(int a[], int left ,int right)

/* within the array a[left: right] elements are arranged such that if pivot p is stored at location q then all elements less than p are stored from left to q-1 and all elements greater than p are stored from q+1 to right. */

{

1. **declare int i,j,temp,v;**
2. **pivot=a[left]; i=left; j=right+1;**
3. **do**

```
{
    do
{
i++;
    } while(a[i]<pivot);
do
    { j--;
    } while(a[j]>pivot);
if(i<j)
        {
          temp=a[i];
a[i]=a[j];          a[j]=temp;
        }
    }while(i<j);
```

4. **a[left]=a[j];**
5. **a[j]=pivot;**
6. **// Display intermediate results**
   Print partition position as j
   Print array

7. **return(j);**

}

Searching Techniques:

1. Sequential or Serial or Linear Search
2. Binary Search
3. Fibonacci Search

Sequential or Serial or Linear search:

In this method, entities are searched one by one starting from the first to the end. Searching stops as soon as we reach to the required entity or we reach to the end. For example, consider the array containing 'n' integers and to search a given number in array we start searching from the 1st element of the array to the end of array. Searching will stop as soon as the element of the array is equal to the given number (number found) or we reach the end of the array (number not found). This method is generally used when the **data stored is not in the order of the key** on which we want to search the table.

Binary Search:

If the **array is sorted on the key** & we want to search for a key having given value, then **Binary search** technique can be used which is more efficient as compared to serial search. Binary search uses the '**divide & conquer**' strategy as given below: 1) Divide the list into two equal halves.

2) Compare the given key value with the middle element of the array or list.
   There are three possibilities.
   a) middle element = key : key found and search terminates
   b) middle element > key : the value which we are searching is possibly in the first half of the list
   c) middle element < key : the value which we are searching is possibly in the second half of the list
3) Now search the key either in first half of second half of the list (b or c) 4) Repeat steps 2 and 3 till key is found or search fails in case key does not exist.

Binary Search:

a) Non recursive int bin_search_nonrec (char a[][20] , int

n, char key[20])

/* this procedure searches a record of given name from set of names and returns the record no to the calling program */ {

Name-Harshita Totala Roll No- SEITA14

Div- A

1) **Declare int** first, last, middle, passcnt=0;
2) **initialize** first=1, last =cnt;
3) **while** (last >= first)

```
{
    middle= ( first  + last )/2        if
( strcmp(key, a[middle] ) >0)
            first = middle + 1;
    else if (( strcmp(key, a[middle]) < 0)
            last = middle – 1;
    else
        return ( middle );
}
```

4) **return**(-1); //not found

}//end bin_search_nonrec

Name-Harshita Totala Roll No- SEITA14

Div- A

PROGRAM-

```
#include<iostream>
#include<string.h>
using namespace std;

typedef struct Students{
    int roll_no;
    char name[20];
    float marks;
}stu;                                   //stu array of structure
student.

void input(stu s[20], int n) {
    int i;
    for (i = 0; i < n; i++){
        cout<<"\nStudent "<<i+1<<" Information ";
        cout<<"\nEnter Name of student : ";
        cin>>s[i].name;
        cout<<"Enter student Roll Number : ";
        cin>>s[i].roll_no;

       int a;
        for(a=0;a<i;a++)
        {
           if(s[i].roll_no==s[a].roll_no)               //to check
same roll-no entered or not
            {
            cout<<"Please enter a unique roll no- ";
            cin>>s[i].roll_no;
            break;
            }
        }
         cout<<"Enter SGPA of student : ";
        cin>>s[i].marks;
    }
    i--;

}
```

Name-Harshita Totala Roll No- SEITA14

Div- A

```
void display(stu s[20], int n){
    cout<<"\n\t "<<"NAME"<<"\t      "<<"ROLL NO."<<"\t\t"<<"SGPA";
    cout<<"\n";
    cout<<"\t----------------------------------------";
    for (int i = 0 ; i < n ; i++)
    {
        cout<<"\n";
        cout<<"\t "<<s[i].name<<"\t\t
"<<s[i].roll_no<<"\t\t"<<s[i].marks<<endl;
    }
}


//bubble sort to sort roll number in ascending order.
void bubble_sort(stu s[20], int n){
    stu temp;
    for(int i = 0 ; i < n ; i++)
    {
        for(int j = 0 ; j < n - 1  ; j++)
        {
            if (s[j].roll_no> s[j+1].roll_no)
            {
                temp = s[j];
                s[j] = s[j+1];
                s[j+1] = temp;
            }
        }
    }

}

// insertion sort to sort names in ascending order
void insertion_sort(stu s[20], int n)
{
    int i,j;
    stu k;
    for (i = 1; i < n; i++)
```

```
    {
        k= s[i];
        j = i - 1;
        while (j >= 0 && strcmp(s[j].name, k.name) >0)
        {
            s[j + 1]= s[j];
            j = j - 1;
        }
            s[j + 1]= k;
    }
}


// quick sort to sort marks in desending order.
int partition(stu s[20], int l,int h){
    int i,j;
     stu temp, pivot;

    pivot=s[l];
    i=l+1;
    j=h;

    do
    {
        while(s[i].marks > pivot.marks ){
         i++;
        }

        while(pivot.marks > s[j].marks){
        j--;
        }

        if(i<j)
        {
            temp=s[i];
            s[i]=s[j];
            s[j]=temp;
        }
    }while(i<j);
```

```
    temp=s[l];
    s[l]=s[j];
    s[j]=temp;

    return(j);
}
void quicksort(stu s[20], int l, int h){

    if(l<h){
        int p=partition(s,l,h);
        quicksort(s,l,p-1);
        quicksort(s,p+1,h);
    }
}


// linear search for marks if more than one student have same marks
print all of them.
void linear_search(stu s[20], int n, float key)
{
    int f=-1;
    cout<<"\n\t "<<"NAME"<<"\t      "<<"ROLL NO."<<"\t\t"<<"SGPA";
    cout<<"\n";
    cout<<"\t----------------------------------------";
    for(int i = 0; i < n; i++)
    {
        if (key == s[i].marks){
            cout<<"\n\t "<<s[i].name<<"\t\t
"<<s[i].roll_no<<"\t\t"<<s[i].marks<<endl;
            f=0;
        }
    }
    if(f==-1){
        cout<<"\n\t---------ENTERED MARKS NOT FOUND --------"<<endl;
    }
}
```

Name-Harshita Totala Roll No- SEITA14

Div- A

```
int binary_search(stu s[20], char x[20], int low, int high)
{
    int mid;
    while( low <= high){

       int mid = (low + high)/2;

       if(strcmp(x,s[mid].name) == 0){
           while(strcmp(x,s[mid].name)==0)
           {
               cout<<"\n\t "<<s[mid].name<<"\t\t
"<<s[mid].roll_no<<"\t\t"<<s[mid].marks<<endl;
               mid--;
           }
            while(strcmp(x,s[mid].name)==0)
           {
               cout<<"\n\t "<<s[mid].name<<"\t\t
"<<s[mid].roll_no<<"\t\t"<<s[mid].marks<<endl;
               mid++;
           }
           return 0;
       }
       else if(strcmp(x,s[mid].name) > 0){
           low=mid+1;
       }
       else{
           high=mid-1;
       }
    }
    return -1;

}



int main(){
    stu s[20];
    int ch, n , r;
    float k;
```

Name-Harshita Totala Roll No- SEITA14

Div- A

```
    char x[20];

    do{
        cout<<"\nChoice of operations are : ";
        cout<<"\n1) Create Student Database";
        cout<<"\n2) Display Records";
        cout<<"\n3) Bubble Sort (for roll number)";
        cout<<"\n4) Insertion sort (for name)";
        cout<<"\n5) Quick sort (for marks)";
        cout<<"\n6) Linear Search (for marks)";
        cout<<"\n7) Binary Search  (for name)";
        cout<<"\n8) Exit"<<endl;

        cout<<"\nEnter your choice of operation : ";
        cin>>ch;

        switch (ch){
            case 1:
                cout<<"\nEnter number of Records to be entered : ";
                cin>>n;
                input(s, n);
                cout<<endl;
                break;

            case 2:
            display(s,n);
            break;

            case 3:
                bubble_sort(s, n);
                display(s,n);
                break;


            case 4:
                insertion_sort(s, n);
                display(s,n);
                break;
```

```
            case 5:
                quicksort(s,0,n-1);
                display(s,n);
                cout<<endl;
                break;


            case 6:
                cout<<"Enter the marks you want to search- ";
                cin>>k;
                linear_search(s, n ,k);
                break;


            case 7:
            cout<<"Enter name you want to search- ";
            cin>>x;
            insertion_sort(s, n);
            r=binary_search(s,x,0,(n-1));
            if(r==-1)
            {
                cout<<"\n----STUDENT NAME NOT PRESENT !----"<<endl;
            }
            else
            {
                cout<<"\n----STUDENT NAME PRESENT !----"<<endl;
            }
            break;


            case 8:
                break;

            default:
                cout<<"\nINVALID CHOICE. CHOOSE AGAIN !!!!\n"<<endl;
            }

        } while (ch != 8);
}
```

Name-Harshita Totala Roll No- SEITA14

Div- A

Output-

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 1

Enter number of Records to be entered : 4

Student 1 Information

Enter Name of student : Akash

Enter student Roll Number : 23

Enter SGPA of student : 6.78

Student 2 Information

Name-Harshita Totala Roll No- SEITA14

Div- A

Enter Name of student : Priya

Enter student Roll Number : 12

Enter SGPA of student : 4.56

Student 3 Information

Enter Name of student : Deep

Enter student Roll Number : 23

Please enter a unique roll no- 67

Enter SGPA of student : 8.90

Student 4 Information

Enter Name of student : Swati

Enter student Roll Number : 56

Enter SGPA of student : 7.21

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

Name-Harshita Totala Roll No- SEITA14

Div- A

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 3

NAME	ROLL NO.	SGPA
Priya	12	4.56
Akash	23	6.78
Swati	56	7.21
Deep	67	8.9

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

Name-Harshita Totala Roll No- SEITA14

Div- A

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 4

NAME	ROLL NO.	SGPA
Akash	23	6.78
Deep	67	8.9
Priya	12	4.56
Swati	56	7.21

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

Name-Harshita Totala Roll No- SEITA14

Div- A

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 5

NAME	ROLL NO.	SGPA
Deep	67	8.9
Swati	56	7.21
Akash	23	6.78
Priya	12	4.56

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

Name-Harshita Totala Roll No- SEITA14

Div- A

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 6

Enter the marks you want to search- 7.21

NAME	ROLL NO.	SGPA
Swati	56	7.21

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Name-Harshita Totala Roll No- SEITA14

Div- A

Enter your choice of operation : 7

Enter name you want to search- Deep

Deep 67 8.9

----STUDENT NAME PRESENT !----

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 8

Conclusion: Thus we had implemented bubble sort, insertion sort, quick sort and linear and binary search.

Name-Harshita Totala Roll No- SEITA14

Div-A

ASSIGNMENT-2
Stack

AIM: Implement stack as an abstract data type using singly linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix and prefix expression.

OBJECTIVE:

1) To understand the concept of abstract data type.
2) How different data structures such as arrays and a stacks are represented as an ADT.

THEORY:

1) What is an abstract data type?

An Abstract Data type is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects. There are no standard conventions for defining them. A broad division may be drawn between "imperative" and "functional" definition styles. In general terms, an abstract data type is a *specification* of the values and the operations that has two properties:

- It specifies everything you need to know in order to use the data type
- It makes absolutely no reference to the manner in which the data type will be implemented.

When we use abstract data types, our programs divide into two pieces:

- The Application: The part that uses the abstract data type.
- The Implementation: The part that implements the abstract data type.

2) What is stack? Explain stack operations with neat diagrams.

In computer science, a **stack** is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. The push operation adds an item to the top of the stack, hiding any items already on the stack, or initializing the stack if it is empty. A pop either reveals previously concealed items, or results in an empty stack. A stack is a restricted data structure, because only a small number of operations are performed on it. The nature of the pop and push operations also mean that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are those that have been on the stack the longest. A collection of items in which only the most recently added item may be removed. The

latest added item is at the top. Basic operations are push and pop. Often top and isEmpty are available, too. Also known as "last-in, first-out" or LIFO.

Operations

An abstract data type (ADT) consists of a data structure and a set of **primitive**

- **Push** adds a new element **Pop** removes a element

Additional primitives can be defined:

- **IsEmpty** reports whether the stack is empty
- **IsFull** reports whether the stack is full
- **Initialise** creates/initialises the stack
- **Destroy** deletes the contents of the stack (may be implemented by re-initialising
- the stack)

3) Explain how stack can be implemented as an ADT.

User can Add, Delete, Search, and Replace the elements from the stack. It also checks for Overflow/Underflow and returns user friendly errors. You can use this stack implementation to perform many useful functions. In graphical mode, this C program displays a startup message and a nice graphic to welcome the user. The program performs the following functions and operations:

- **Push**: Pushes an element to the stack. It takes an integer element as argument. If the stack is full then error is returned.
- **Pop**: Pop an element from the stack. If the stack is empty then error is returned. The element is deleted from the top of the stack.
- **DisplayTop** : Returns the top element on the stack without deleting. If the stack is empty then error is returned.

4) With an example explain how an infix expression can be converted to prefix and postfix form with the stack. (Ans to be written by students)

ALGORITHM:

Abstract Data Type Stack:

Define Structure for stack(Data, Next Pointer) **Stack**

Empty:

Return True if Stack Empty else
False. Top is a pointer of type
structure stack.
Empty(Top)
Step 1: If Top
= = NULL

Step 2: Return
1;
Step 3: Return 0;

Push Opertion:

Top & Node pointer of structure
Stack. Push(element)
Step 1: Node>data=element;
Step
2; Node->Next =
Top; Step 3: Top = Node
Step 3: Stop.

Pop Operation:

Top & Temp pointer of structure
Stack. Pop()
Step 1:If Top
!=
NULL
Then
i) Temp = Top;
ii) element=Te
mp->data; iiI)
Top =
(Top)>Next; iv)
delete temp; Step 2:
Else Stack is Empty.
Step 3:
return element;

Infix to Prefix Conversion:

String is an array of type character which store infix
expression. This function return Prefix expression.
InfixToPrefix(S tring) Step 1: I
= strlen(String);
Step 2:
Decrement I;
Step 3: Do Steps 4 to 9 while I >= 0
Step 4: If isalnum(String[I] Then PreExpression[J++] =
String[I]; Step 5: Else If String[I]=='('
Then a) Temp=Pop(&Top);//Pop Operator from stack
b) Do Steps while Temp->Operator!=')' and Temp!=NULL
i) PreExpression[J++] = Temp->Operator;
ii) Temp = Pop(&Top);}

Step 6: Else If String[I]==')' Then Push(&Top,Node);//push _)‘ in a stack
Step 7: Else

a) Temp = Pop(&Top);//Pop operator from stack
b) DO Steps while Temp->Operator != ')' And Temp != NULL And Priority(String[I])<Priority(Temp->Operator)
i) PreExpression[J++] = Temp->Operator; ii) Temp = Pop(&Top);}
c) If Temp!=NULL And Temp->Operator==')' Or Priority(String[I])>=Priority(Temp->Operator)) Then Push(&Top,Temp);

Step 8: Push(&Top,Node);//Push String[I] in a

stack; Step 9: Decrement I;

Step 10: Temp = Pop(&Top);//pop remaining operators from stack
Step 11: Do Steps while Temp != NULL //Stack is not Empty
i) PreExpression[J++] = Temp->Operator; ii)Temp = Pop(&Top);}
Step 12: PreExpression[J] = NULL;
Step 13: Reverse PreExpression;
Step 14:
Return PreExpression.

Infix to Postfix Conversion:

String is an array of type character which store infix expression. This function return Postfix expression.
InfixToPost fix(String)
Step 1: I = 0;
Step 2: Do Steps 3 to 8 while String[I] != NULL
Step 3: If isalnum(String[I] Then PostExpression[J++] = String[I]; Step 4: Else If String[I]==')'
Then a) Temp=Pop(&Top);//Pop Operator from stack
b) Do Steps while Temp->Operator!='(' and Temp!=NULL
i) PostExpression[J++] = Temp->Operator;
ii) Temp = Pop(&Top);}

Step 5: Else If String[I]=='(' Then Push(&Top,Node);//push _)‘ in a stack
Step 6: Else

a) Temp = Pop(&Top);//Pop operator from stack
b) DO Steps while Temp->Operator != '(' And Temp != NULL And Priority(String[I]) >= Priority(Temp->Operator) iii) PreExpression[J++] = Temp->Operator; iv) Temp = Pop(&Top);}
c) If Temp!=NULL And Temp->Operator=='(' Or Priority(String[I]) < Priority(Temp->Operator)) Then Push(&Top,Temp); Step 7: Push(&Top,Node);//Push String[I] in a stack; Step 8: Increment I;
Step 9: Temp = Pop(&Top);//pop remaining operators from stack
Step 10: Do Steps while Temp != NULL //Stack is not Empty
iii) PostExpres8 8(&Top);} Step 11: PostExpression[J] = NULL; Step 12: Return PostExpression.

PostFix Expression Evaluation:

String is an array of type character which store infix expression. Postfix_Evaluation(String) Step 1: Do Steps 2 & 3 while String[I] != NULL Step 2: If String[I] is operand Then Push it into Stack Step 3: If it is an operator Then
Pop two operands from Stack;
Stack Top is 1st Operand & Top-1 is 2nd Operand;
Perform Operation; Step 4: Return Result.

INPUT:

Test Case	O/P
If Stack Empty	Display message —Stack Empty‖
Stack Empty Full‖	Display message —Stack

Name-Harshita Totala Roll No- SEITA14

Div-A

INPUT:	**POSTFIX OUTPUT:**
(A+B) * (C-D)	AB+CD-*
A$B*C-D+E/F/(G+H)	AB$C*D-EF/GH+/+
((A+B)*C-(D-E))$(F+G)	AB+C*DE—FG+$
A-B/(C*D$E)	ABCDE$*/-
A^B^C	ABC^^

INPUT:	**PREFIX OUTPUT:**
(A+B) * (C-D)	*+AB-CD
A$B*C-D+E/F/(G+H)	+-*$ABCD//EF+GH
((A+B)*C-(D-E))$(F+G)	$-*+ABC-DE+FG
A-B/(C*D$E)	-A/B*C$DE
A^B^C	^A^BC

NOTE: Here $ AND ^ are used as raised to operator

Name-Harshita Totala Roll No- SEITA14

Div-A

Program-

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

typedef struct node
{
char data;
struct node *next;
}n;                              //n array of structure node

class stack
{
n *top;
public:
stack()                         //constructor of class stack
{
top=NULL;
}

int isempty()                   //check whether stack is empty or not.
{
if(top==NULL)
return 1;
else
return 0;
}

void push(char x)
{
n *p;
p=new node();
p->data=x;
p->next=top;
top=p;
}

char pop()
```

```
{
node *p;
char x;
p=top;
x=p->data;
top=top->next;
delete(p);
return x;
}

char topdata()
{
return top->data;
}
};

void infix_postfix(char infix[20],char postfix[20]);
int evaluate(int op1,int op2,char op);
void evaluate_postfix(char postfix[20]);
int precedence(char x);
void reverse(char a[20],char b[20]);
void infix_prefix(char infix[20],char prefix[20]);
void evaluate_prefix(char prefix[20]);
void infix_prefix_1(char infix[20],char prefix[20]);

int main()
{
char infix[20],token,postfix[20],prefix[20];
int ch,result;
do
{
cout<<"\n1. Infix to Postfix Expression";
cout<<"\n2. Evaluate Postfix";
cout<<"\n3. Infix to Prefix Expression";
cout<<"\n4. Evalute Prefix";
cout<<"\n5. Exit";
cout<<"\nEnter your choice: ";
cin>>ch;
switch(ch)
```

Name-Harshita Totala Roll No- SEITA14

Div-A

```
{

case 1:
cout<<"\nEnter Infix expression:  ";
cin>>infix;
infix_postfix(infix,postfix);
cout<<"\n------------------------------";
cout<<"\nPOSTFIX EXPRESSION- "<<postfix<<endl;
cout<<"------------------------------"<<endl;
break;

case 2:
evaluate_postfix(postfix);
break;

case 3:
cout<<"\nEnter Infix expression:  ";
cin>>infix;
infix_prefix(infix,prefix);
cout<<"\n------------------------------";
cout<<"\nPREFIX EXPRESSION- "<<prefix<<endl;
cout<<"------------------------------"<<endl;
break;

case 4:
evaluate_prefix(prefix);
break;

case 5:
cout<<"\nProgram Exited! ";
break;

default:
cout<<"WRONG CHOICE! Enter again...";
}

}while(ch!=5);
}
```

Name-Harshita Totala Roll No- SEITA14

Div-A

```
//infix----> postfix
void infix_postfix(char infix[20],char postfix[20])
{
stack s;
int i,j=0;
char token,x;
for(i=0;infix[i]!='\0';i++)
{
token=infix[i];

if(isalnum(token))                                    //will check
whether the token is number or an alphabet
{
postfix[j]=token;
j++;
}

else
{
if(token=='(')                                        //check whether
token is opening bracket, then push the token.
s.push(token);

else if(token==')')                                   //check whether
token is closing bracket.
{
while((x=s.pop())!='(')                              //pop all
elements until we get opening bracket.
{
postfix[j]=x;
j++;
}
}

else
{
while(s.isempty()==0 && precedence(token)<=precedence(s.topdata())
)            //first check stack is not empty and then the
precedence.
```

```
{
postfix[j]=s.pop();
j++;
}
s.push(token);
}
}
}

while(s.isempty()==0)
{
postfix[j]=s.pop();
j++;
}

postfix[j]='\0';

}

//evaluation of postfix.
void evaluate_postfix(char postfix[20])
{
stack s;
int i,op1,op2,result;
char token;
int x;
for(i=0;postfix[i]!='\0';i++)
{
token=postfix[i];

if(isalnum(token))
{
cout<<"Enter the value "<<token<<" :- ";
cin>>x;
s.push(char(x));
}

else
{
```

```
op2=s.pop();        //first pop
op1=s.pop();        //second pop
result=evaluate(op1,op2,token);
s.push(char(result));
}
}

result=s.pop();
cout<<"\n------------------------------";
cout<<"\nPOSTFIX EVALUATION- "<<result<<endl;
cout<<"------------------------------"<<endl;
}



//to reverse the given infix for prefix
void reverse(char x[20],char y[20])         //x will store infix and
y will store reverse of infix
{
int i,j=0;
for(i=0;x[i]!='\0';i++);
i--;                                       //since 'for' loop repeat
once again when the condition is false.

for(j=0;i>=0;j++,i--)
{
if(x[i]=='(')
y[j]=')';

else if(x[i]==')')
y[j]='(';

else
y[j]=x[i];
}
y[j]='\0';
}

//infix----> prefix
```

Name-Harshita Totala Roll No- SEITA14

Div-A

```
void infix_prefix(char infix[20],char prefix[20])
{
char rev[20],pre1[20];
reverse(infix,rev);
infix_prefix_1(rev,pre1);
reverse(pre1,prefix);
}


void infix_prefix_1(char infix[20],char prefix[20])
{
stack s;
int i,j=0;
char token,x;
for(i=0;infix[i]!='\0';i++)
{
token=infix[i];

if(isalnum(token))                                    //will check
whether the token is number or an alphabet
{
prefix[j]=token;
j++;
}
else
{
if(token=='(')                                       //check whether
token is opening bracket, then push the token.
s.push(token);

else if(token==')')                                  //check whether
token is closing bracket.
{
while((x=s.pop())!='(')                             //pop all
elements until we get opening bracket.
{
prefix[j]=x;
j++;
}
```

```
}
else
{
while(s.isempty()==0 && precedence(token)<precedence(s.topdata())
)            //first check stack is not empty and then the
precedence.
{
prefix[j]=s.pop();
j++;
}
s.push(token);
}
}
}
while(s.isempty()==0)
{
prefix[j]=s.pop();
j++;
}

prefix[j]='\0';

}



//evaluation of prefix.
void evaluate_prefix(char prefix[20])
{
stack s;
int i,op1,op2,result;
char token,pre[20];
int x;
reverse(prefix,pre);
for(i=0;pre[i]!='\0';i++)
{
token=pre[i];

if(isalnum(token))
```

```
{
cout<<"Enter the value "<<token<<" :- ";
cin>>x;
s.push(char(x));
}

else
{
op1=s.pop();        //first pop
op2=s.pop();        //second pop
result=evaluate(op1,op2,token);
s.push(char(result));
}
}

result=s.pop();
cout<<"\n------------------------------";
cout<<"\nPREFIX EVALUATION- "<<result<<endl;
cout<<"------------------------------"<<endl;
}



int precedence(char x)
{
if(x=='(')
return 0;
if(x=='+' || x=='-')
return 1;
if(x=='*' || x=='/')
return 2;
return 3;
}

int evaluate(int op1,int op2,char op)
{
if(op=='+')
return op1+op2;
if(op=='-')
```

Name-Harshita Totala Roll No- SEITA14

Div-A

```
return op1-op2;
if(op=='*')
return op1*op2;
if(op=='/')
return op1/op2;
if(op='%')
return op1%op2;
}
```

Name-Harshita Totala Roll No- SEITA14

Div-A

Output-

1. Infix to Postfix Expression
2. Evaluate Postfix
3. Infix to Prefix Expression
4. Evalute Prefix
5. Exit

Enter your choice: 1

Enter Infix expression: (5-2)+8/2-1

POSTFIX EXPRESSION- 52-82/+1-

---

1. Infix to Postfix Expression
2. Evaluate Postfix
3. Infix to Prefix Expression
4. Evalute Prefix
5. Exit

Enter your choice: 2
Enter the value 5 :- 5
Enter the value 2 :- 2
Enter the value 8 :- 8
Enter the value 2 :- 2
Enter the value 1 :- 1

POSTFIX EVALUATION- 6

---

1. Infix to Postfix Expression

2. Evaluate Postfix
3. Infix to Prefix Expression
4. Evalute Prefix
5. Exit
Enter your choice: 3

Enter Infix expression: (5-2)+8/2-1

PREFIX EXPRESSION- -+-52/821
---

1. Infix to Postfix Expression
2. Evaluate Postfix
3. Infix to Prefix Expression
4. Evalute Prefix
5. Exit
Enter your choice: 4
Enter the value 1 :- 1
Enter the value 2 :- 2
Enter the value 8 :- 8
Enter the value 2 :- 2
Enter the value 5 :- 5

PREFIX EVALUATION- 6
---

1. Infix to Postfix Expression
2. Evaluate Postfix
3. Infix to Prefix Expression
4. Evalute Prefix

5. Exit
Enter your choice: 5

Program Exited!

Conclusion: In this experiment we understood how to implement stack as an abstract data type using linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix/prefix expression.

Name- Harshita Totala Roll No-SEITA14

Div- A

Assignment-3

Circular Queue

AIM : Implement Circular Queue using Array. Perform following operations on it.

a) Insertion (Enqueue)

b) Deletion (Dequeue)

c) Display

Objectives :

1. To understand the concept of Circular Queue using Array as a data structure.

2. Applications of Circular Queue.

Theory :

1)Definition of Circular Queue –

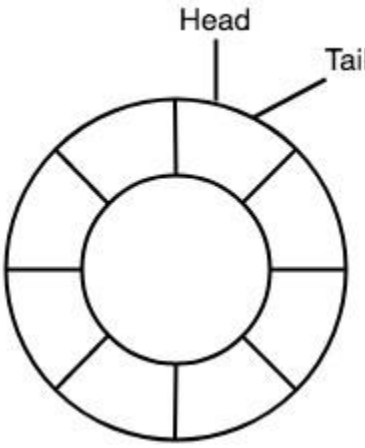
Circular Queue is a linear data structure, which follows the principle of **FIFO**(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

2) Definition of Array –

An **array**, is a data structure consisting of a collection of *elements* (values or variables), each identified by at least one *array index* or *key* .

Basic features of Circular Queue

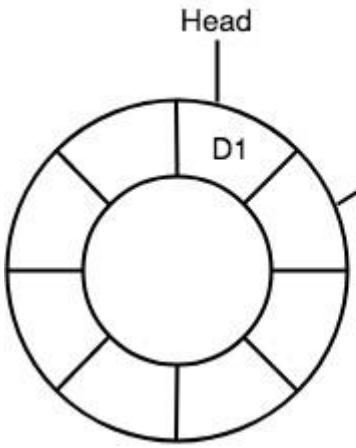
1. In case of a circular queue, `head` pointer will always point to the front of the queue, and `tail` pointer will always point to the end of the queue.
2. Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty.



Initially the queue is empty, as Head and Tail are at same location

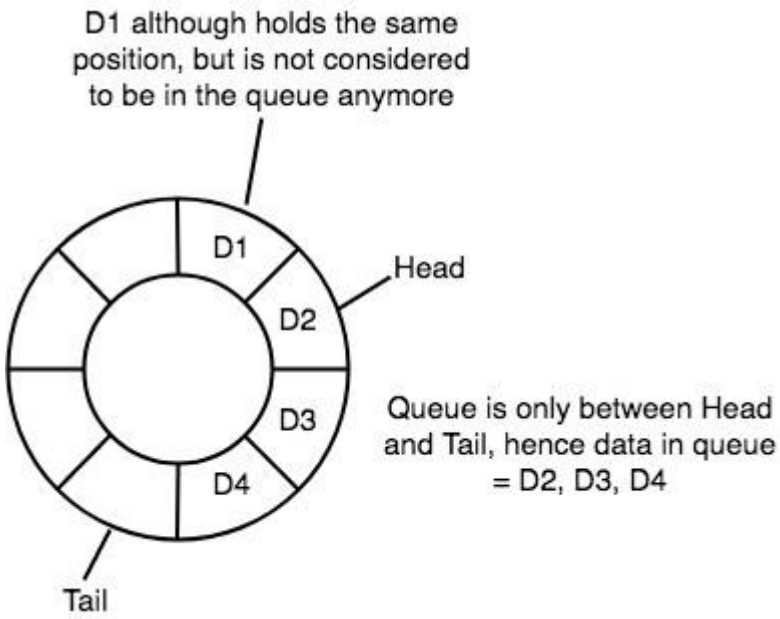
A simple circular queue with size 8

3. New data is always added to the location pointed by the `tail` pointer, and once the data is added, `tail` pointer is incremented to point to the next available location.

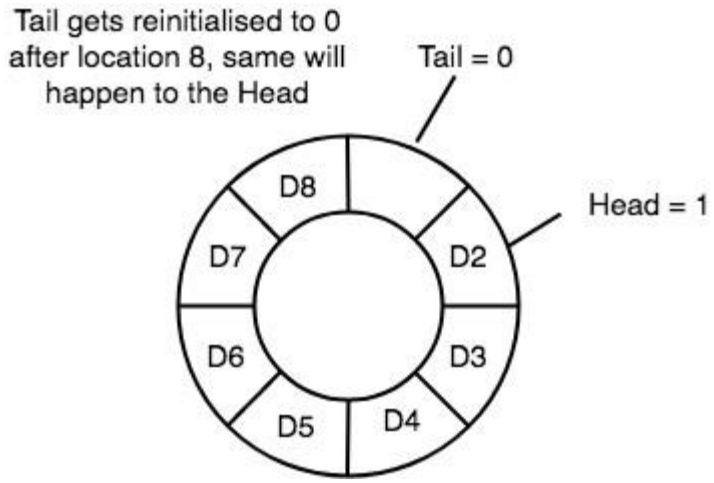


Tail always points to the location where new data will be inserted.

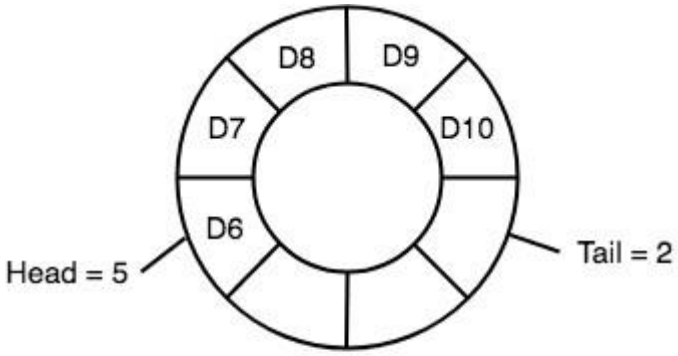
4. In a circular queue, data is not actually removed from the queue. Only the `head` pointer is incremented by one position when **dequeue** is executed. As the queue data is only the data between `head` and `tail`, hence the data left outside is not a part of the queue anymore, hence removed.



5. The `head` and the `tail` pointer will get reinitialised to **0** every time they reach the end of the queue.



6. Also, the `head` and the `tail` pointers can cross each other. In other words, `head` pointer can be greater than the `tail`. Sounds odd? This will happen when we dequeue the queue a couple of times and the `tail` pointer gets reinitialised upon reaching the end of the queue.



In such a situation the value of the Head pointer will be greater than the Tail pointer

Insertion :

enQueue(value) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

Steps:

1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.

Pseudo Code :

```
      void insertCQ(int val)
{     if ((front == 0 && rear == n - 1) || (front == rear +
1))
    {         cout << "Queue
Overflow \n";         return;
    }     if (front
== -1)
    {
front = 0;
rear = 0;
    }     else     {
if (rear == n - 1)
rear = 0;         else
rear = rear + 1;     }
cqueue[rear] = val;
}
```

Name- Harshita Totala Roll No-SEITA14

Div- A

Deletion :

deQueue() This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

Steps:

1. Check whether queue is Empty means check (front==-1).
2. If it is empty then display Queue is empty. If queue is not empty then step 3
3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

Pseudo Code :

```
      void deleteCQ()
{     if (front == -
1)
    {         cout << "Queue Underflow\n";         return;     }
cout << "Element deleted from queue is : " << cqueue[front] << endl;
if (front == rear)
    {         front
= -1;         rear
= -1;
    }     else     {
if (front == n - 1)
front = 0;          else
front = front + 1;
    } }
```

Display Circular Queue :

We can use the following steps to display the elements of a circular queue...

- **Step 1 -** Check whether **queue** is **EMPTY**. (**front == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front**'.
- **Step 4 -** Check whether '**front <= rear**', if it is **TRUE**, then display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i <= rear**' becomes **FALSE**.
- **Step 5 -** If '**front <= rear**' is **FALSE**, then display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until'**i <= SIZE - 1**' becomes **FALSE**.
- **Step 6 -** Set **i** to **0**.
- **Step 7 -** Again display '**cQueue[i]**' value and increment **i** value by one (**i++**). Repeat the same until '**i <= rear**' becomes **FALSE**.

Pseudo Code :

```
      void displayCQ_forward()
```

Name- Harshita Totala Roll No-SEITA14

Div- A

```
{     int f = front, r =
rear;     if (front == -1)
    {         cout << "Queue is empty" <<
endl;         return;
    }     cout << "Queue elements are
:\n";     if (f <= r)
    {         while (f
<= r)
        {             cout <<
cqueue[f] << " ";             f++;
        }     }     else
{         while (f <= n -
1)
        {             cout <<
cqueue[f] << " ";             f++;
}         f = 0;         while (f <=
r)
        {             cout <<
cqueue[f] << " ";             f++;
        }
}
    cout << endl;
}
```

Name- Harshita Totala Roll No-SEITA14

Div- A

Program-

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

int queue[5];
int front=-1 , rear=-1 , n = 5;

void insertion(int var)
{
    if ((front == 0 && rear == n-1) || (front == rear + 1)){          //check
whether queue if full or not.
        cout<<"\n-----QUEUE IS FULL-----\n"<<endl;
    }
    else if (front == -1){                                            //check
whether queue is empty or not.
        front = 0;
        rear = 0;
    }
    else{                                                             //insert
elements.
        rear =(rear + 1) % 5;
    }
     queue[rear] = var;
}



void deletion()
{
    if(front == -1){                                                  //check
whether queue is empty or not.
        cout<<"\n------QUEUE IS EMPTY------"<<endl;
        return;
    }
    cout<<"Element dequeued is : "<<queue[front]<<endl;

    if(front == rear){                                               //if queue is
having only one element.
        front = -1;
        rear = -1;
        }
    else{
        front=(front+1)%5;
    }
```

```
}


void display_forward()
{
    int f= front ,r = rear;
     if(front == -1){
         cout<<"\n------QUEUE IS EMPTY-----"<<endl;
         return ;
}
 cout<<"\nELEMENTS IN FORWARD QUEUE - "<<endl;
 if(f<=r)
 {
     while(f<=r){
         cout<<queue[f]<<" ";
         f++;
    }
 }

 else
 {
     while(f<=n-1){
         cout<<queue[f]<<" ";
         f++;
      }
       f=0;
        while(f<=r){
            cout<<queue[f]<<" ";
            f++;
        }
 }
 cout<<endl;
}


void display_reverse()
{
 int f = front;
 int r = rear;
 if(front == -1)
 {
     cout<<"\n-----QUEUE IS EMPTY-----"<<endl;
     return;
 }
 cout<<"\nELEMENTS IN REVERSE QUEUE -  "<<endl;
```

```
 if(f<=r)
 {
     while(f<=r){
         cout<<queue[r]<<" ";
         r--;
 }
 }
 else
 {
     while(r>=0){
         cout<<queue[r]<<" ";
         r--;
    }
    r=n-1;
    while(r>=f)
    {
        cout<<queue[r]<<" ";
        r--;
    }
 }
 cout<<endl;
}


int main(){
    int choice;
    cout<<"\n----** CIRCULAR QUEUE PROGRAM **----"<<"\n"<<endl;
    do{
        cout<<"Choice of operations are : "<<endl;
        cout<<"1] Insertion Queue"<<endl;
        cout<<"2] Deletion Queue"<<endl;
        cout<<"3] Display Forward Queue"<<endl;
        cout<<"4] Display Reverse Queue"<<endl;
        cout<<"5] Exit"<<endl;
        cout<<"\nEnter your choice of operations : ";
        cin>>choice;

        switch(choice)
        {
            case 1:
            int var;
            cout<<"Enter element : ";
            cin>>var;
            cout<<endl;
            insertion(var);
```

```
            break;

            case 2:
            deletion();
            cout<<"\n"<<endl;
            break;

            case 3:
            display_forward();
            cout<<"\n";
            break;

            case 4:
            display_reverse();
            cout<<"\n";
            break;

            case 5:
            cout<<"\nPROGRAM EXITED !";
            break;

            default:
            cout<<"WRONG CHOICE ! CHOOSE AGAIN !!"<<"\n"<<endl;
        }

    }while(choice!=5);
}
```

Name- Harshita Totala Roll No-SEITA14

Div- A

Output-

----** CIRCULAR QUEUE PROGRAM **----

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 1

Enter element : 4

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Name- Harshita Totala Roll No-SEITA14

Div- A

Enter your choice of operations : 1

Enter element : 7

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 1

Enter element : 8

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 3

Name- Harshita Totala Roll No-SEITA14

Div- A

ELEMENTS IN FORWARD QUEUE -

4 7 8

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 4

ELEMENTS IN REVERSE QUEUE -

8 7 4

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 2

Element dequeued is : 4

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 3

ELEMENTS IN FORWARD QUEUE -

7 8

Choice of operations are :

1] Insertion Queue

2] Deletion Queue

3] Display Forward Queue

4] Display Reverse Queue

5] Exit

Enter your choice of operations : 5

PROGRAM EXITED !

Conclusion:

Thus we had Implemented Circular Queue using Array and performed following operations :

1) Insertion (Enqueue)
2) Deletion (Dequeue)
3) Display

Name-Harshita Totala Roll No-SEITA14

Div-A

Assignment-4
Expression Tree

AIM: Construct and expression tree from postfix/prefix expression and perform recursive and non- recursive In-order, pre-order and post-order traversals.

OBJECTIVE:

1. Understand the concept of expression tree and binary tree.
2. Understand the different type of traversals (recursive & non-recursive).

THEORY:

1. **Definition of an expression tree with diagram.**

Algebraic expressions such as a/b + (c-d) e

The terminal nodes (leaves) of an expression tree are the variables or constants in the expression (a, b, c, d, and e). The non-terminal nodes of an expression tree are the operators (+, -, , and). Notice that the parentheses which appear in Equation do not appear in the tree. Nevertheless, the tree representation has captured the intent of the parentheses since the subtraction is lower in the tree than the multiplication.

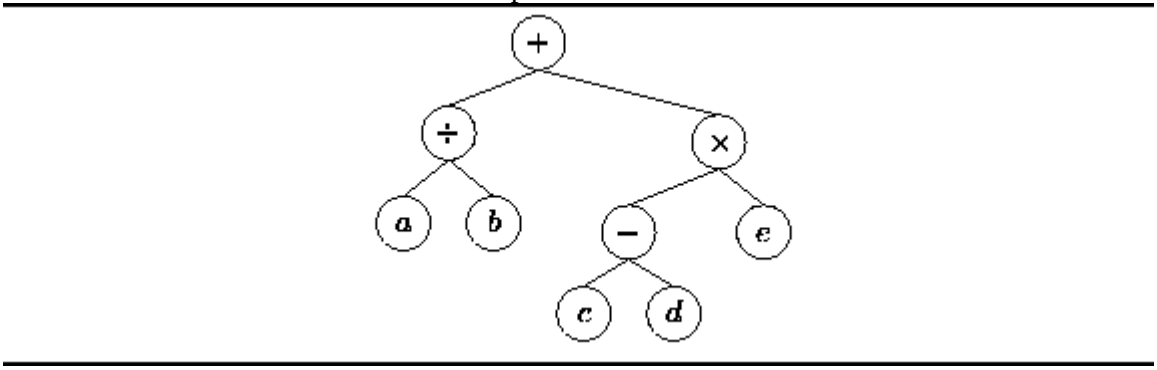


Figure: Tree representing the expression a/b+(c-d)e.

2. **Show the different type of traversals with example**

To traverse a non-empty binary tree in **preorder**,

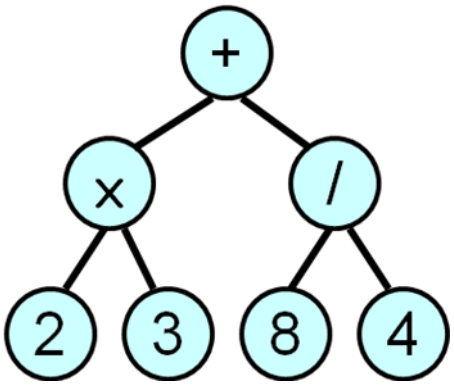
1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

To traverse a non-empty binary tree in **inorder**:

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

To traverse a non-empty binary tree in **postorder**,

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.



- **Pre-order (prefix)**
  + □ 2 3 / 8 4
- **In-order (infix)**
  2 □ 3 + 8 / 4
- **Post-order (postfix)**
  2 3 □ 8 4 / +

ALGORITHM:

Define structure for Binary Tree (Information, Left Pointer & Right Pointer) **Create Expression Tree:**

CreateTree()

Root& Node pointer variable of type structure. Stack is an pointer array of type structure. String is character array which contains postfix expression. Top & I are variables of type integer.

Step 1: Top = -1 , I = 0;

Step 2: Do Steps 3,4,5,6 While String[I] != NULL Step 3: Create Node of type of structure

Step 4:

Node>Data=String[I]

; Step 5: If
isalnum(String[I])
Then
Stack[Top++] =
Node; Else
Node->Right = Stack
[--Top]; Node->Left =
Stack[--Top]; Stack[
Top++] =
Node; Step 6: Increment
I;
Step 7: Root =
Stack[0]; Step
8: Return Root

Inorder Traversal Recursive : Tree
is pointer of type structure.
InorderR(Tree)
Step 1: If Tree != NULL
Step 2: InorderR(Tree->Left
) Step 3:
Print Tree->Data
Step 4: InorderR(Tree->Right)

Postorder Traversal
Recursive: Tree is
pointer of type
structure.
PostorderR(Tree)
Step 1: If Tree != NULL Step
2: PostorderR(Tree>Left)
Step 3:
PostorderR(Tree->Right)
Step 4: Print Tree->Data

Preorder Traversal Recursive:
Tree is pointer of type
structure.
PreorderR(Tree)
Step 1: If Tree

!= NULL Step
2: Print Tree-
>Data Step 3:
PreorderR(Tree>Left)
Step 4:
PreorderR(Tree->Right
)

Postorder Traversal Nonrecursive :

NonR_Postorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure.Top variable of type integer.
Step 1: Temp = Tree // current pointer pointing to root
Step 2: if Temp != NULL then push current pointer along with its initial flag value _L‘ on to the stack and traverse on the left side.
Step 3: Otherwise if the stack is not empty then pop an address from stack along with its flag value.
Step 4: For the current pointer if the flag value is _L‘ then change it to _R‘ and push current pointer along with its flag value _R‘ on to the stack and traverse on the right side.
Step 5: otherwise, For the current pointer if the flag value is _R‘ then display the element and make the current pointer NULL (i.e.temp=NULL) Step 6: repeat steps 2, 3, 4, 5 until the stack becomes empty.

Preorder Traversal Nonrecursive :

NonR_Preorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variable of type integer.
Step 1: Temp = Tree
Step 2: Do Steps3,4,5,6,7,& 8 While Temp != NULL And Stack is not
Empty Step 3: Do Steps 4,5&6 While Temp != NULL
Step 4: Print Temp->Data
Step 5: Stack[++ Top] = Temp //Push
Element Step 6: Temp = Temp->Left
Step 7: Temp = Stack [Top --] //Pop
Element Step 8: Temp = Temp->Right

Inorder Traversal Nonrecursive :

NonR_Inorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variable of type integer.
Step 1: Temp = Tree
Step 2: Do Steps3,4,5,6,7,&8 While Temp != NULL And Stack is not
Empty Step 3: Do Steps 4,5 While Temp != NULL
Step 4: Stack[++ Top] = Temp;//Push

Element Step 5: Temp = Temp->Left
Step 6: Temp = Stack[Top --]//Pop
Element Step 7: Print Temp->Data
Step 8: Temp = Temp->Right

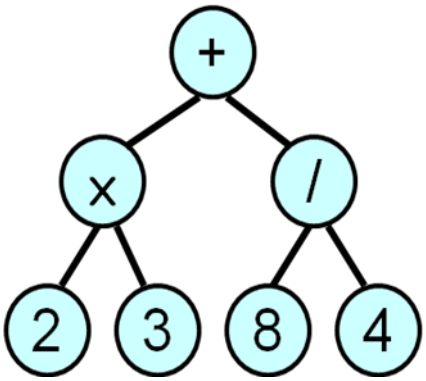
INPUT:

Postfix Expression: **2 3 □ 8 4 / +**

OUTPUT:

Display result of each operation with error checking.

Expression tree



Name-Harshita Totala Roll No-SEITA14

Div-A

Program-

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

typedef struct node                //node of tree
{
    char data;
    struct node *left;
    struct node *right;

}node;

typedef struct stacknode
{
    node* data;
    struct stacknode *next;
}stacknode;

class stack
{
    stacknode *top;
    public:
    stack()
    {
        top=NULL;
    }
    node* topp()
    {
        return (top->data);
    }
    int isempty()
    {
        if(top==NULL)
            return 1;

        else
          return 0;
    }

    void push(node* a)
    {
        stacknode *p;
```

```
        p=new stacknode();
        p->data=a;
        p->next=top;
        top=p;
    }

    node* pop()
    {
        stacknode *p;
        node* x;
        x=top->data;
        p=top;
        top=top->next;

        return x;
    }
};

node* create_pre(char prefix[10]);
node* create_post(char postfix[10]);
void inorder_non_recursive(node *t);
void inorder(node *p);
void preorder(node *p);
void postorder(node *p);
void preorder_non_recursive(node *t);
void postorder_non_recursion(node *t);

node* create_post(char postfix[10])
{
    node *p;
    stack s;
    for(int i=0;postfix[i]!='\0';i++)
    {
        char token=postfix[i];
        if(isalnum(token))
        {
            p=new node();
            p->data=token;
            p->left=NULL;
            p->right=NULL;
            s.push(p);
        }
        else
        {
            p=new node();
```

```
            p->data=token;
            p->right=s.pop();
            p->left=s.pop();
            s.push(p);
        }
    }
    return s.pop();


}

node* create_pre(char prefix[10])
{node *p;
stack s;
int i,j;
        for(i=0;prefix[i]!='\0';i++);
        i--;

    for( j=0;i>=0;j++,i--)
    {
        char token=prefix[i];
        if(isalnum(token))
        {
            p=new node();
            p->data=token;
            p->left=NULL;
            p->right=NULL;
            s.push(p);
        }
        else
        {
            p=new node();
            p->data=token;
            p->left=s.pop();
            p->right=s.pop();
            s.push(p);
        }
    }
    return s.pop();


}

int main()
{
```

```
    node *r=NULL;
    node *r1;
    char postfix[10],prefix[10];
    int x;
    int ch,choice;
do
{
    cout<<"\n---TREE OPERATIONS---\n1.Construct tree from postfix
expression/prefix expression\n2.Inorder traversal"
    "\n3.Preorder traversal\n4.Postorder traversal\n5.Exit\nEnter your
choice=";
    cin>>ch;
    switch(ch)
    {
        case 1:cout<<"Enter choice:\n1.Postfix expression\n2.Prefix
expression\n";
            cin>>choice;
            if(choice==1)
            {
                cout<<"\nEnter postfix expression=";
                cin>>postfix;
                r=create_post(postfix);
            }
            else
            {
                cout<<"\nEnter prefix expression=";
                cin>>prefix;
                r=create_pre(prefix);
            }
            cout<<"\n\nTree created successfully";
            break;


        case 2:cout<<"\nInorder Traversal of tree:\n";
                inorder(r);
                cout<<"\n Without recursion:\t";
                inorder_non_recursive(r);
                break;
        case 3:cout<<"\nPreorder Traversal of tree:\n";
                preorder(r);
                cout<<"\npreorder traversal without recursion:\t";
                preorder_non_recursive(r);
                break;
        case 4:cout<<"\nPostorder Traversal of tree:\n";
                postorder(r);
```

```
                cout<<"\npostorder traversal without recursion";
                postorder_non_recursion(r);
                break;
    }
}while(ch!=5);
    return 0;
}

void inorder(node *p)
{
        if(p!=NULL)
        {
            inorder(p->left);
            cout<<p->data;
            inorder(p->right);
        }
}

void preorder(node *p)
{
        if(p!=NULL)
        {
            cout<<p->data;
            preorder(p->left);
            preorder(p->right);
        }
}


void postorder(node *p)
{
        if(p!=NULL)
        {
            postorder(p->left);
            postorder(p->right);
            cout<<p->data;
        }
}




void inorder_non_recursive(node *t)
{
```

```
    stack s;
    while(t!=NULL)
    {
        s.push(t);
        t=t->left;
    }

    while(s.isempty()!=1)
    {
        t=s.pop();
        cout<<t->data;
        t=t->right;
        while(t!=NULL)
    {
        s.push(t);
        t=t->left;
    }


    }


}



void preorder_non_recursive(node *t)
{

    stack s;
    while(t!=NULL)
    {
        cout<<t->data;
        s.push(t);
        t=t->left;
    }

    while(s.isempty()!=1)
    {
        t=s.pop();

        t=t->right;
        while(t!=NULL)
    {
        cout<<t->data;
```

```
        s.push(t);
        t=t->left;
    }


    }
}

void postorder_non_recursion(node *t)
{
    stack s,s1;
node *t1;
    while(t!=NULL)
    {
        s.push(t);
        s1.push(NULL);
        t=t->left;
    }
    while(s.isempty()!=1)
    {
        t=s.pop();
        t1=s1.pop();
        if(t1==NULL)
        {
            s.push(t);
            s1.push((node *)1);
            t=t->right;
                while(t!=NULL)
                {
                    s.push(t);
                    s1.push(NULL);
                    t=t->left;
                }

        }
        else
        cout<<t->data;
    }


}
```

Name-Harshita Totala Roll No-SEITA14

Div-A

Output-

```
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=1
Enter choice:
1.Postfix expression
2.Prefix expression
1

Enter postfix expression=52-82/+1-


Tree created successfully
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=2

Inorder Traversal of tree:
5-2+8/2-1
 Without recursion:     5-2+8/2-1
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=3

Preorder Traversal of tree:
-+-52/821
preorder traversal without recursion:   -+-52/821
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
```

2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=4

Postorder Traversal of tree:
52-82/+1-
postorder traversal without recursion52-82/+1-
---TREE OPERATIONS---
1.Construct tree from postfix expression/prefix expression
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Exit
Enter your choice=5

Conclusion: Thus we have Constructed an expression tree from postfix/prefix expression and performed recursive and non- recursive In-order, pre-

Name-Harshita Totala Roll No-SEITA14
Div-A

Assignment-5
Binary search tree

AIM: Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry)
b) Delete
c) Search
d) Display tree (Traversal)
e) Display - Depth of tree
f) Display - Mirror image
g) Create a copy
h) Display all parent nodes with their child nodes
i) Display leaf nodes
j) Display tree level wise

OBJECTIVE:

1. To understand the concept of binary search tree as a data structure.
2. Applications of BST.

THEORY:

1. Definition of binary search tree

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

2. Illustrate the above operations graphically.

Searching

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method.We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search the right subtree. This process is repeated until the value is found or the indicated subtree is null. If the searched value is not found before a null subtree is reached, then the item must not be present in the tree.

Insertion

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or

left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

Deletion

There are three possible cases to consider:

- **Deleting a leaf (node with no children):** Deleting a leaf is easy, as we can simply remove it from the tree.
- **Deleting a node with one child:** Remove the node and replace it with its child.
- **Deleting a node with two children:** Call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R. Replace the value of N with the value of R, then delete R.

As with all binary trees, a node's in-order successor is the left-most child of its right subtree, and a node's in-order predecessor is the right-most child of its left subtree. In either case, this node will have zero or one children. Delete it according to one of the two simpler cases above.

ALGORITHM:

Define structure for Binary Tree (Mnemonics, Left Pointer, Right Pointer)

Insert Node:

Insert (Root, Node)
Root is a variable of type structure, represent Root of the Tree. Node is a variable of type structure ,represent new Node to insert in a tree.
Step 1: Repeat Steps 2,3 & 4 Until Node do not insert at appropriate position. Step 2: If
Node Data is less that Root Data & Root Left Tree is NULL Then
insert Node to Left.
Else Move Root to Left
Step 3: Else If Node Data is Greater that equal that Root Data & Root Right Tree is NULL Then
insert Node to Right.
Else Move Root to Right.
Step 4: Stop.

Search Node:

Search (Root, Mnemonics)
Root is a variable of type structure, represent Root of the Tree. Mnemonics is array of character. This function search Mnemonics in a Tree.

Step 1: Repeat Steps 2,3 & 4 Until Mnemonics Not find && Root != NULL Step 2: If Mnemonics Equal to Root Data Then
print message Mnemonics present.
Step 3: Else If Mnemonics Greater that Equal that Root Data Then Move Root to Right.
Step 4: Else Move Root to Left.
Step 5: Stop.

Delete Node:

Dsearch(Root, Mnemonics)

Root is a variable of type structure ,represent Root of the Tree. Mnemonics is array of character. Stack is an pointer array of type structure. PTree(Parent of Searched Node),Tree(Node to be deleted), RTree(Pointg to Right Tree),Temp are pointer variable of type structure; Step 1: Search Mnemonics in a Binary Tree Step 2: If Root = = NULL Then Tree is NULL Step 3:
Else //Delete Leaf Node

If Tree->Left = = NULL && Tree->Right = = NULL
Then a) If Root = = Tree Then Root = NULL;
a) If Tree is a Right Child PTree->Right=NULL;
Else PTree->Left=NULL;

Step 4: Else // delete Node with Left and Right children If
Tree->Left != NULL && Tree->Right != NULL Then a)
RTree=Temp=Tree->Right;
b) Do steps i && ii while Temp->Left !=NULL
i) RTree=Temp;
ii) Temp=Temp->Left;
c)RTree->Left=Temp->Right;
d) If Root == Tree//Delete Root Node
Root=Temp;
e) If Tree is a Right Child PTree->Right=Temp;
Else PTree->Left=Temp;
f) Temp->Left=Tree->Left;
g) If RTree!=Temp
Then Temp->Right = Tree->Right;

Step 5: Else //with Right child If
Tree->Right!= NULL
Then a) If Root==Tree Root = Root->Right;
b) If Tree is a Right Child PTree->Right=Tree->Right;
Else PTree->Left=Tree->Left;

Step 6: Else //with Left child If
Tree->Left != NULL
Then a) If Root==Tree Root = Root->Left;
b) If Tree is a Right Child PTree->Right=Tree->Left;
Else PTree->Left=Tree->Left; Step 7: Stop.

Display:

In order to display content of each node in exactly once, a BT can be traversed in either PreOrder, Post- Order, In-Order.
Here Pre-order Traversal is considered

1. Visit the root node R First
2. Then visit the left sub tree of R in Pre-Order fashion
3. Finally, visit the right sub tree of R in Pre-Order fashion

Depth of a Tree:

This function finds the depth of the linked binary tree.

1. Start from the ROOT node and store the address of the root node in **ptr.**
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then tree is empty.
3. Otherwise

a. Get the height of left sub tree, say leftHeight.
b. Get the height of right sub tree, say rightHeight.
c. Take the Max(leftHeight, rightHeight) and add 1 for the root and return. d. Call recursively.

Mirror Image:

Root is a variable of type structure ,represent Root of the Tree.
Queue is an pointer array of type structure. Front & Rear variable of type integer.
Mirror(Root)
Step 1: Queue[0]=Root;//Insert Root Node in a Queue Step 2:
Repeat Steps 3,4,5,6,7 & 8 Until Queue is Empty Step 3: Root
= Queue[Front++];
Step 4: Temp1 = Root->Left; Step 5: Root->Left
= Root->Right; Step 6:
Root->Right = Temp1; Step 7: If
Root->Left != NULL
Then Queue[Rear++] = Tree->Left;//insert Left SubTree
Step 8: If Root->Right != NULL

Then Queue[Rear++] = Root->Right;//insert Right SubTree Step 9:
Stop.

Create a Copy of a Tree:

This function will make a copy of the linked binary tree. The function should allocate memory for necessary nodes and copy respective contents in it.

1. Start from the ROOT node. The address of the root node will be in the **ptr.** Let newroot be the root of the new tree after the copy.
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then newroot = NULL
3. Otherwise

    (i) Allocate memory space for the new node
    (ii) Copy the data from current node from the old tree to node in the new tree.
    (iii) Traverse each node in the left subtrees from the old tree and repeat step (i) and (ii).
    (iv) Traverse each node in the right subtrees from the old tree and repeat step (i) and (ii).

4. Stop

Display Leaf-Nodes:

This function displays all the leaf nodes in the linked binary tree.

1. Start from the ROOT node and store the address of the root node in **ptr.**
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then tree is empty.
3. Otherwise

    a. Check whether the left child and right child of **ptr** are empty or not. If the left child and right child of **ptr** are NULL then display the content of the node.
    b. Traverse the left subtrees and the right subtrees and repeat step a for all the nodes.

4. Stop

Breath First Search(Levelwise Display):

Root is a variable of type structure ,represent Root of the Tree.
Queue is an pointer array of type structure. Front & Rear variable of type integer. BFS(Root)
Step 1:If Root == NULL Then Empty Tree;
Step 2: Else Queue[0] = Root;// insert Root of the Tree in a Queue

Step 3: Repeat Steps 4,5,6 & 7 Until Queue is Empty
Step 4: Tree=Queue[Front++]; //Remove Node From Queue Step
5: print Root Data
Step 6: If Root->Left != NULL
Then Queue[++Rear] = Tree->Left;//insert Left Subtree in a Queue Step
7: If Root->Right != NULL
Then Queue[++Rear] = Root->Right; //insert Left Subtree in a Queue Else
if Root->Right == NULL And Root->Left == NULL
Leaf++;//Number of Leaf Nodes Step
8: Stop.

Program-

```
#include <iostream>
#include<stdlib.h>
using namespace std;

struct node{
    int data;
    struct node *left;
    struct node *right;
};

node *insert(node *root, int val){
    if (root == NULL){
        node *temp;
        temp=new node;
        temp->data=val;
        temp->left=temp->right=NULL;
        return temp;
    }
    if (val < root->data){
        root->left = insert(root->left, val);
    }
    else{

        root->right = insert(root->right, val);
    }
    return root;
}

void inorder(node *root)
{
    if (root == NULL){
        return;
    }
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

node* inorderSucc(node* root){
    node* curr=root;
    while(curr && curr->left!=NULL){
        curr=curr->left;
```

```
    }
    return curr;
}

node *delete_ele(node *root, int key){
    if(key<root->data){
        root->left=delete_ele(root->left, key);
    }
    else if(key>root->data){
        root->right=delete_ele(root->right,key);
    }

    else{
        if(root->left==NULL){
            node* temp=root->right;
            free(root);
            return temp;
        }
        else if(root->right==NULL){
            node* temp=root->left;
            free(root);
            return temp;
        }
        node* temp=inorderSucc(root->right);
        root->data=temp->data;
        root->right=delete_ele(root->right, temp->data);
    }
    return root;
}

node* search(node* root, int val){
    if(root==NULL){
        return NULL;
    }
    if(val>root->data){
        return search(root->right, val);
    }
    else if(val<root->data){
        return search(root->left, val);
    }
    else{
        return root;
    }
}
```

```
void mirror(node* root){
    if(root==NULL){
        return;
    }
    else{
        struct node *temp;
        mirror(root->left);
        mirror(root->right);
        swap(root->left,root->right);
    }
}

node *copy(node *root){
    node *temp=NULL;
    if(root!=NULL){
        temp=new node();
        temp->data=root->data;
        temp->left=copy(root->left);
        temp->right=copy(root->right);
    }
    return temp;
}

void leafNodes(node* root){
    if(root==NULL){
        return;
    }
    if(!root->left && !root->right){
        cout<<root->data<<" ";
        return;
    }
    if(root->right)
       leafNodes(root->right);
    if(root->left)
       leafNodes(root->left);
}

int Height(node* root){
    if(root==NULL){
        return 0;
    }
    int lheight=Height(root->left);
```

```
    int rheight=Height(root->right);
    return max(lheight,rheight)+1;
}

node *Min(node *root){
    if(root==NULL){
        return NULL;
    }
    if(root->left)
        return Min(root->left);
    else
        return root;
}

node *Max(node *root){
    if(root==NULL){
        return NULL;
    }
    if(root->right)
       return Max(root->right);
    else
        return root;
}

int main()
{
    cout<<"\n** Welocme to Binary Search Tree **"<<endl;
    node *root=NULL, *temp;
    int ch;
    do{
        cout<<"\n---------------\n  Main Menu\n---------------"<<endl;
        cout<<"1) Insert" << endl;
        cout<<"2) Delete" << endl;
        cout<<"3) Search" << endl;
        cout<<"4) Create copy "<<endl;
        cout<<"5) Display leaf nodes "<<endl;
        cout<<"6) Height of tree"<<endl;
        cout<<"7) Find minimum"<<endl;
        cout<<"8) Find maximum"<<endl;
        cout<<"9) Mirror image"<<endl;
        cout<<"10) Exit"<<endl;
        cout<<"-----------------"<<endl;
        cout<<"\nEnter your choice: ";
        cin>>ch;
```

```
        switch (ch){
        case 1:
            cout << "Enter the element to be insert :  ";
            cin >> ch;
            root= insert(root, ch);
            cout << " Elements in Tree are ->  ";
            inorder(root);
            break;
        case 2:
            cout<<"Enter the element to be deleted :  ";
            cin>>ch;
            root=delete_ele(root, ch);
            cout<<"* Element deleted successfully *"<<endl;
            cout<<"Current elements in Tree are :  ";
            inorder(root);
            break;
        case 3:
            cout<<"Enter the element to search :   ";
            cin>>ch;
            temp=search(root, ch);
            if(temp==NULL){
                cout<<"*Element not found*";
            }
            else{
                cout<<"*Element found*";
            }
            break;
        case 4:
            cout<<"Copy of the Tree is :   ";
            root=copy(root);
            inorder(root);
            break;
        case 5:
            cout<<"The leaf nodes are :  ";
            leafNodes(root);
            break;
        case 6:
            cout<<"Height of the Tree is:  "<<Height(root);
            break;
        case 7:
            temp=Min(root);
            cout<<"\nMinimum element is :  "<<temp->data;
            break;
        case 8:
```

```
            temp=Max(root);
            cout<<"\nMaximum element is:   "<<temp->data;
            break;
        case 9:
            mirror(root);
            cout<<"Mirror image is:  ";
            inorder(root);
            break;
        case 10:
            cout<<"Thank you !! Do Visit Again";
        default:
            cout<<"\nInvalid choice !! Please enter your choice again";
        }
    }while(ch!=10);
}
```

Output-

```
** Welocme to Binary Search Tree **

---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 1
Enter the element to be insert :  4
 Elements in Tree are ->  4
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 1
Enter the element to be insert :  8
 Elements in Tree are ->  4 8
---------------
  Main Menu
---------------
```

1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 1
Enter the element to be insert : 3
Elements in Tree are -> 3 4 8

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 3
Enter the element to search : 3
Element found

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree

7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 4
Copy of the Tree is : 3 4 8

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 5
The leaf nodes are : 8 3

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 6
Height of the Tree is: 2

```
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 7

Minimum element is :  3
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 8

Maximum element is:   8
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
```

4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 9
Mirror image is: 8 4 3

Conclusion: Thus we had Implemented binary search tree and performed following operations:

a) Insert (Handle insertion of duplicate entry)
b) Delete
c) Search
d) Display tree (Traversal)
e) Display - Depth of tree
f) Display - Mirror image
g) Create a copy
h) Display all parent nodes with their child nodes
i) Display leaf nodes
j) Display tree level wise

Name-Harshita Totala Roll No-SEITA14 Div-IT A

ASSIGNMENT-6
Threaded Binary Tree

AIM : Implement In-order Threaded Binary Tree and traverse it in Inorder and Pre-order.

Objectives :

1) To understand the concept of Threaded Binary Tree as a data structure.
2) Applications of Threaded Binary Tree.

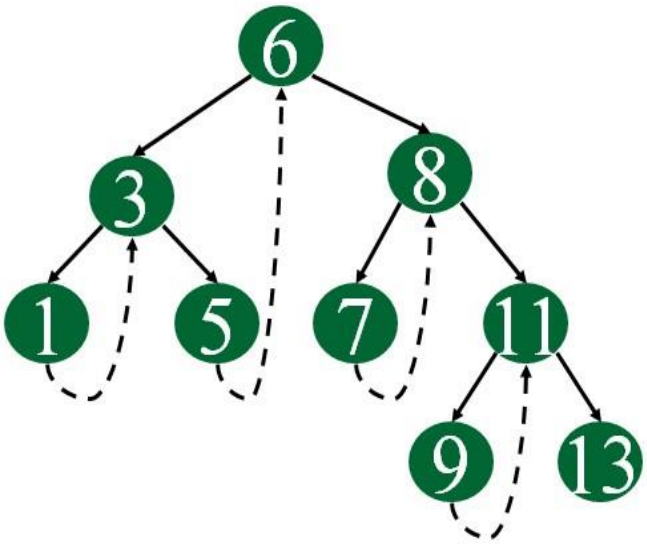
Theory :

Definition of Threaded Binary Tree :

A binary tree is threaded by making all right child pointers that would normally be null point to the inorder successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node**.**

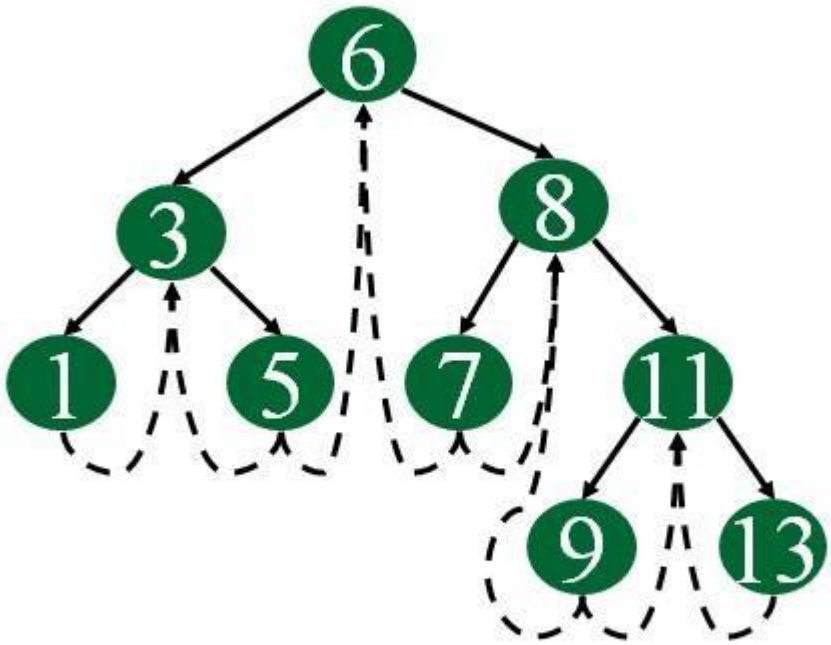
Types of threaded binary trees:

- **Single Threaded**: each node is threaded towards either the in-order predecessor or successor (left **or**right) means all right null pointers will point to inorder successor **OR** all left null pointers will point to inorder predecessor.



Inorder: 1 3 5 6 7 8 9 11 13

- **Double threaded:** each node is threaded towards both the in-order predecessor and successor (left **and**right) means all right null pointers will point to inorder successor **AND** all left null pointers will point to inorder predecessor.



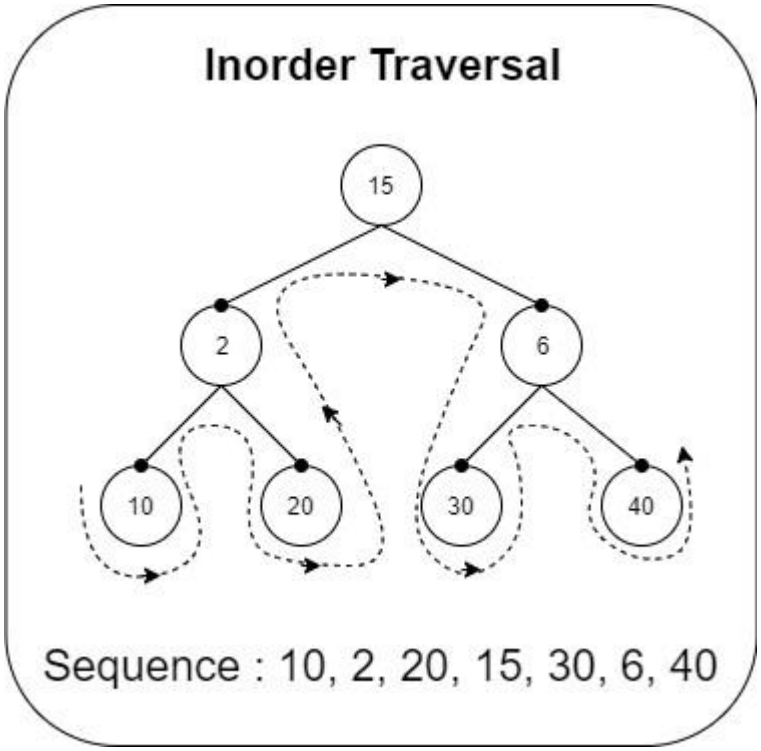
Double Threaded Binary Tree

Inorder: 1 3 5 6 7 8 9 11 13

In-order Traverse :

Name-Harshita Totala Roll No-SEITA14 Div-IT A

Inorder Traversal follows the order LNR (left, node, right).



Steps →

1. Create a stack **inorder** to fit elements of type TreeNode
2. Run a while loop until there are no more elements to process, i.e. the **root** is NULL and the stack is empty
3. In each iteration →

- Keep moving to the left child until it exists and push the elements onto the stack so that we can process them after we process the leftmost element
- Once we reach the left-most element of current subtree, print it and remove it from the stack.
- What's the next priority? Either the right subtree or current node or its parent.

- So we move to its right child if it exists. If it doesn't the parent node is already in the stack. Pop it and process it

Pseudo Code :

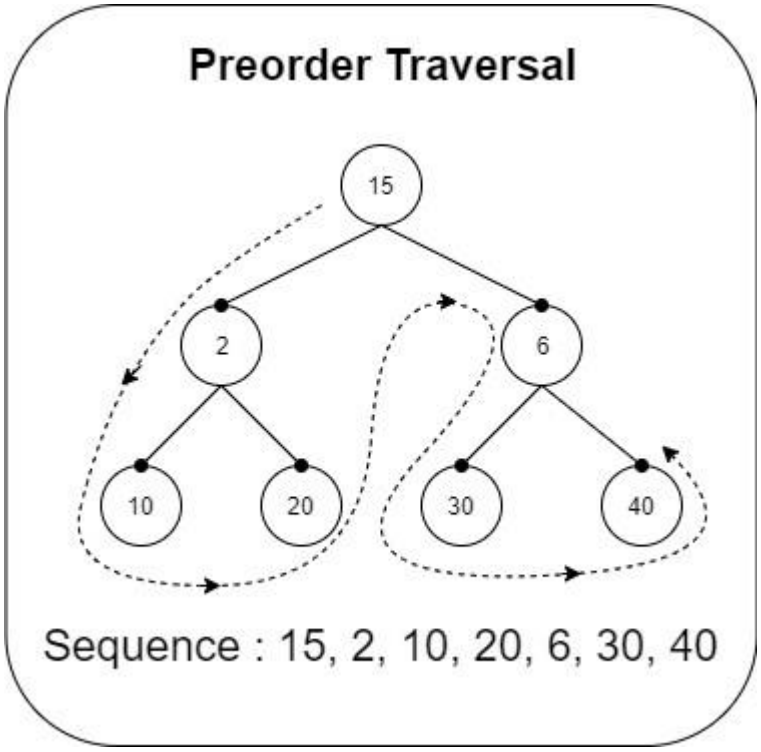
```
     void printInorder(TreeNode root)
{
    Create stack 'inorder' to fit elements of type TreeNode
while ( root != NULL or inorder.empty() != True )
    {
        // Find the leftmost node
while( root != NULL )
        {
            inorder.push(root)
root = root.left
        }

        root = inorder.peek()
        inorder.pop()
        print( root )
        root = root.right
    }                          }
```

Name-Harshita Totala Roll No-SEITA14 Div-IT A

Pre-order Traverse :

Preorder Traversal as we read above, traverse the tree in the order: Node, left child and then the right child.



Steps →

1. Create a stack **preorder** that will temporarily store the elements like the *function stack* in the above recursion.

2. Since priority is given to the root, push the root first to the stack.
3. Now we will run a loop until the stack empties, i.e., there are no more elements left to process.
4. In each iteration, print the node (i.e. the element at top of the stack) then push the right child in the stack and then the left child.

Pseudo Code :

```
     void printPreorder(TreeNode root)

{
    Create stack 'preorder' to fit elements of type TreeNode
preorder.push( root )
    while ( preorder.empty() != True )
    {
        TreeNode node = preorder.peek()
preorder.pop()          if ( node ==
NULL )              continue

        print ( node.val )
        // Note that right child is pushed first
// so that left child is on top
preorder.push( node.right )          preorder.push(
node.left )
    }  }
```

Name-Harshita Totala Roll No-SEITA14 Div-IT A

CODE-

```
#include<bits/stdc++.h>
using namespace std;
class Node{
public:
    int data;
    Node* left;
    Node* right;
    int leftThread; // leftThread=0 -> left pointer points to the inorder
predecessor
    int rightThread; // rightThread=0 -> right pointer points to the inorder
successor
    Node(int val){
        this->data = val;
    }
};
class DoubleThreadedBinaryTree{
private:
    Node* root;
public:
    DoubleThreadedBinaryTree(){
        // dummy Node with value as INT_MAX
        root = new Node(INT_MAX);
        root->left = root->right = root;
        root->leftThread = 0;
        root->rightThread = 1;
    }
    void insert(int data){
        Node* new_node = new Node(data);
        if(root->left == root && root->right == root){
            //Empty Tree
            new_node->left = root;
            root->left = new_node;
            new_node->leftThread = 0;
            new_node->rightThread = 0;
            root->leftThread = 1;
            new_node->right = root;
            return;
        }
        else{
            Node* current = root->left;
```

```
            while(true){
                if(current->data > data){
                    if(current->leftThread == 0 ){
                     // this is the last Node
                     new_node->left = current->left;
                     current->left = new_node;
                     new_node->leftThread = current->leftThread;
                     new_node->rightThread = 0;
                     current->leftThread = 1;
                     new_node->right = current;
                     break;
                    }
                    else{
                        current = current->left;
                    }
                }
                else{
                    if(current->rightThread == 0){
                        // this is the last Node
                        new_node->right = current->right;
                        current->right = new_node;
                        new_node->rightThread = current->rightThread;
                        new_node->leftThread = 0;
                        current->rightThread=1;
                        new_node->left = current;
                        break;
                    }
                    else{
                        current = current->right;
                    }
                }
            }
        }
    }
    Node* findNextInorder(Node* current){
        if(current->rightThread == 0){
            return current->right;
        }
        current = current->right;
        while (current->leftThread != 0)
        {
            current = current->left;
        }
        return current;
    }
```

```
    void inorder(){
        Node* current = root->left;
        while(current->leftThread == 1){
            current = current->left;
        }
        while(current != root){
            cout<<current->data<<" ";
            current = findNextInorder(current);
        }
        cout<<"\n";
    }
    void preorder(){
        Node* current = root->left;
        while(current != root){
            cout<<current->data<<" ";
            if(current->left != root && current->leftThread != 0)
                current= current->left;
            else if(current->rightThread == 1){
                current = current->right;
            }
            else{
                while (current->right != root && current->rightThread == 0)
                {
                    current = current->right;
                }
                if(current->right == root)
                    break;
                else
                {
                    current=current->right;
                }
            }
        }
        cout<<"\n";
    }


};

int main(){
    DoubleThreadedBinaryTree dtbt;
    dtbt.insert(10);
    dtbt.insert(45);
    dtbt.insert(1);
    dtbt.insert(7);
```

```
    dtbt.insert(76);
    dtbt.insert(34);
    dtbt.insert(61);
    dtbt.insert(100);
    cout<<"Inorder -"<<endl;
    dtbt.inorder();
    cout<<endl;
    cout<<"Preorder- "<<endl;
    dtbt.preorder();
   ;
    return 0;
}
```

OUTPUT-

Inorder -

1 7 10 34 45 61 76 100

Preorder-

10 1 7 45 34 76 61 100

Conclusion:

Thus we have implemented In-order Threaded Binary Tree and traversed it in In-order and Pre-order.

Name-Harshita Totala Roll No-SEITA14

Div-A

Assignment-7
Minimum Spanning tree

AIM : Implementation of Minimum Spanning tree using Prims and Kruskals Algorithm

DETAILED PROBLEM STATEMENT:

Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find minimum spanning tree

a) Using Kruskal's algorithm.
b) Using Prim's algorithm.

OBJECTIVE

1. To study Graph theory.
2. To study different graph traversal methods.
3. To understand the real time applications of graph theory.
4. To Implement a MST using Prims and Kruskals algorithm

OUTCOME :

1. Understand Non-linear data structure - graph.
2. Represent a graph using adjacency list / adjacency matrix.
3. Implement Prim's and Kruskal's Algorithm 4. Identify applications of Minimum Spanning Trees.
5. Analyze the Time and Space complexity.

THEORY:

Introduction Graph :

- **Definition :** A graph is a triple G = (V,E, φ)where • V is a finite set, called the vertices of G, E is a finite set, called the edges of G, and,φ is a function with domain E and codomain P2(V).
- **Loops:** A loop is an edge that connects a vertex to itself.
- **Degrees of vertices**: Let G = (V,E, φ) be a graph and v ∈ V a vertex. Define the degree of v, d(v) to be the number of e ∈ E such that v ∈ φ(e); i.e., e is Incident on v.
- **Directed graph:** A directed graph (or digraph) is a triple D = (V,E, φ) where V and E are finite sets and φ is a function with domain E and codomain V × V . We call E the set of edges of the digraph D and call V the set of vertices of D.
- **Path:** Let G = (V,E, φ) be a graph.

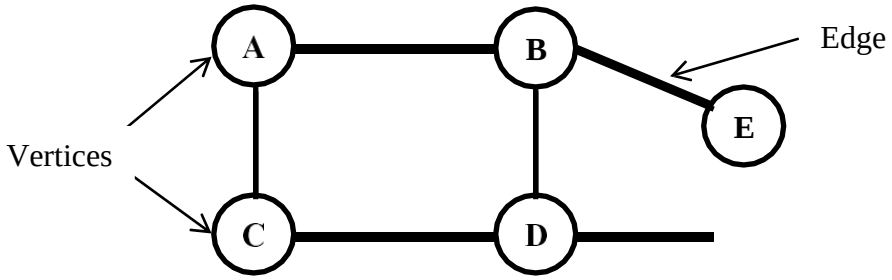
  Let e1, e2, . . . , en−1 be a sequence of elements of E (edges of G) for which there is a sequence a1, a2, . . . , an of distinct elements of V (vertices of G) such that φ(ei) = {ai, ai+1} for i = 1, 2, . . . , n − 1. The sequence of edges e1, e2, . . . , en−1 is called a path in G. The sequence of vertices a1, a2, . . . , an is called the vertex sequence of the path.

Circuit and Cycle: Let G = (V, E, φ) be a graph and let e1, . . . , en be a trail with vertex sequence a1, . . . , an, a1. (It returns to its starting point.) The subgraph G′of Ginduced by the set of edges{e1,...,en}is called a circuit of

G. The length of the circuit is n.

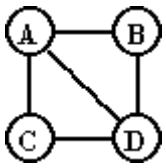
e.g.:

This graph G can be defined as G = (V, E) Where V= {A,B,C,D,E} and E={(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.



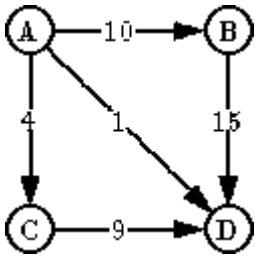
Different representations of graph:

1. **Adjacency matrix:** Graphs *G* = (*V, E*) can be represented by adjacency matrices *G* [$v_1..v_{|V|}$, $v_1..v_{|V|}$], where the rows and columns are indexed by the nodes, and the entries *G* [v_i, v_j] represent the edges. In the case of unlabeled graphs, the entries are just Boolean values.



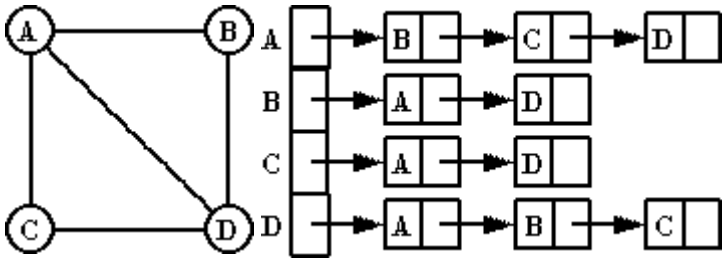
	A	**B**	**C**	**D**
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

In case of labeled graphs, the labels themselves may be introduced into the entries.



	A	**B**	**C**	**D**
A	∞	10	4	1
B	∞	∞	∞	15
C	∞	∞	∞	9
D	∞	∞	∞	∞

2. **Adjacency List:** A representation of the graph consisting of a list of nodes, with each node containing a list of its neighboring nodes.



- **Spanning Tree:**

A Spanning Tree of a graph G = (V, E) is a sub graph of G having all vertices of G and no cycles in it.

Minimal Spanning Tree: The cost of a graph is the sum of the costs of the edges in the weighted graph. A spanning tree of a graph G= (V, E) is called minimal cost spanning tree or simply the minimal spanning tree of G if its cost is minimum.

- When a graph G is connected, depth first or breadth first search starting at any vertex visits all the vertices in G.
- The edges of G are partitioned into two sets i.e. T for the tree edges & B for back edges. T is the set of tree edges and B for back edges. T is the set of edges used or traversed during the search & B is the set of remaining edges.
- The edges of G in T form a tree which includes all the vertices of graph G and this tree is called as spanning tree.

Definition: Any tree, which consists solely of edges in graph G and includes all the vertices in G, is called as spanning tree. Thus for a given connected graph there are multiple spanning trees possible. For maximal connected graph having ‗n' vertices the number of different possible spanning trees is equal to n.

Cycle: If any edge from set B of graph G is introduced into the corresponding spanning tree T of graph G then cycle is formed. This cycle consists of edge (v, w) from the set B and all edges on the path from w to v in T.

There are many approaches to computing a minimum spanning tree. We could try to detect cycles and remove edges, but the two algorithms we will study build them from the bottom-up in a greedy fashion.

Prim's Algorithm – starts with a single vertex and then adds the minimum edge to extend the spanning tree.

Kruskal's Algorithm – starts with a forest of single node trees and then adds the edge with the minimum weight to connect two components.

• **Prim's algorithm:** Prim‘s algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm was discovered in 1930 by mathematician Vojtech Jarník and later independently by computer scientist Robert C. Prim in 1957 and rediscovered by Edsger Dijkstra in 1959. Therefore it is sometimes called the DJP algorithm, the Jarník algorithm, or the Prim-Jarník algorithm.

Applications of spanning Trees:

- To find independent set of circuit equations for an electrical network. By adding an edge from set B to spanning tree we get a cycle and then Kirch off‘s second law is used on the resulting cycle to obtain a circuit equation.
- Using the property of spanning trees we can select the spanning tree with (n-1) edges such that total cost is minimum if each edge in a graph represents cost.
- Analysis of project planning
- Identification of chemical compounds
- Statistical mechanics, genetics, cybernetics, linguistics, social sciences

ALGORITHMS/ PESUDOCODE :

- **Prims algorithm :**

Data structure used:

Array: Two dimensional array (adjacency matrix) to store the adjacent vertices & the weights associated edges

One dimensional array to store an indicator for each vertex whether visited or not. #define max 20 int adj_ver[max][max], int edge_wt[max][max], int ind[max]

- **Algorithm to generate spanning tree by Prim's**

Prims(Weight ,Vertex)

//Weight is a two dimensional array having V no of rows and columns. KOWN ,cost , PRIEV are the l vectors.

Step 1: Repeat for I = 1 to V

i. KNOWN[I] = 0 ii.
PREV[I] = 0 iii.
cost[I] = 32767

Step 2: current= 1 //starting vertex of prims

Step 3: Total_V =0 //total vertex considered till the time

Step 4: KNOWN[current] =1 // Start is known now

Step 5: Repeat thru step 6 while Total_v != Vertex

i. mincost= 32767 ii.
Repeat for I =1 to V
iii. If (weight[current][I] != 0 AND KNOWN[I] =0)
i. If (cost[I] >= weight[current][I])

a. cost[I] = weight[current][I]
ii. end if
iv. Repeat for I =1 to V //finding min cost edge from current vertices
i. If (KNOWN[I] = 0 AND cost[I] <= mincost)
a. mincost =cost[I] //if min is Cost[i]
b. current = I //next node visited is I
ii. end if
v. end if
Step 6: KNOWN[current] = 1
Step 7: Toatal_v= Total_v + 1
Step 8: mincost = 0
Step 9: Repeat for I = 1 to V
i. WRITE(I , PREV[I]) //display mst edges
ii. If cost[I] != 32767
a. mincost =mincost + cost[I]
Step 10 : display final mincost
Step 11: end of prims

Trace of Prim's algorithm for graph G1 starting from vertex 1

Step No.	**Set A**	**Set (V-A)**	**Min cost Edge (u, v)**	**Cost**	**Set B**
Initial	{1}	{2,3,4,5,6,7}	--	--	{}
1	{1,2}	{3,4,5,6,7}	(1, 2)	1	{(1, 2)}
2	{1, 2, 3}	{4, 5, 6, 7}	(2, 3)	2	{(1,2},(2,3)}
3	{1,2,3,5}	{4, 6, 7}	(2, 5)	4	{(1,2),(2,3),(2,5)}
4	{1,2,3,5,4}	{6, 7}	(1,4)	4	{(1,2),(2,3),(2,5),(1,4)}
5	{1,2,3,5,4,7}	{6}	(4,7)	4	{(1,2),(2,3),(2,5),(1,4),(4,7)}
6	{1,2,3,5,4,7.6}	{ }	(7,6)	3	{(1,2),(2,3),(2,5),(1,4),(4,7),(7,6)}
		Total Cost		17	

Thus the minimum spanning tree for graph G1 is : A = { 1,2,3,4,5,7,6} B
={(1,2),(2,3),(2,5),(1,4),(4,7),(7,6)}, total Weight: 1+2+4+3+4+3=17

Kruskal's Algorithm:

- **Prerequisite for Kruskals** struct edge

```
{
    Number v1,v2,wt
} edge
```

- **Create edge Matrix**

AdjToEdges(int weight[][MAX], int n, edge E[])

Step 1: for i=0 to n do
Step 2: for j=i+1 to n do
i.i f(weight[i][j])
a. edge_matrix[k]. start=i
b. edge_matrix[k].end =j
c. edge_matrix[k++].Value=G[i][j] ii. end if
Step 3:end for
Step 4: end for return k;
Step 5: end AdjToEdges

- **Function to sort the edges according to weights Algorithm**

SortEdges(edge edge_matrix[], no_edge)

Step 1:for i=0 to no_edge
Step 2: for j=i+1 to no_edge
i. if(edge_matrix[i].value> edge_matrix[j].value)
a. t=edge_matrix[i]
b. edge_matrix[i]= edge_matrix[j]
c. edge_matrix[j]=t
ii. end if
Step 3 : end for j
Step 4: end for i
Step 5: end sort edge

Kruskals (G, N)

//G is a pointer to head of the adjacency List. N is max. number of vertices.

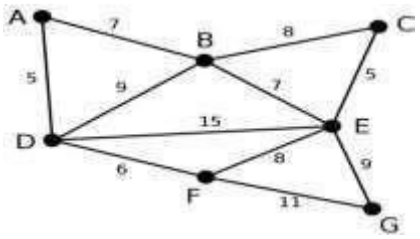
// L and K =0
Step 1: mincost = 0

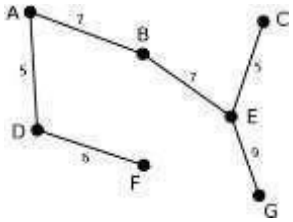
Step 2: EARRAY(G)

Step 3:Repeat for I=1 to N
i. set[I] = I

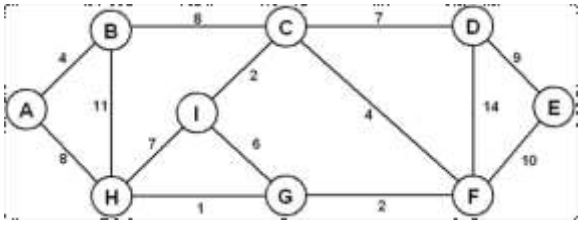
Step 4: Repeat Thru step 6 while L < Vertex-1 //select the min cost edge tillno of edges =Vertex-1
i. T = Edge_mat[K] // select min cost edge
ii. K = K+1 // to select next edge iii. Repeat for I = 1 to N
iv. PV1=FIND(set,T.V1) //check the set Membership of V1
v.PV2=FIND(set,T.V2) // check the Set membership of V2
vi. if (PV1 != PV2) //if both v1 and v2 are belongs to different set
//they are not forming cycle and can be added to fina
a.WRITE(V1(T),V2(T) , D(T))
b. mincost = micost + Dist[T]
c. L=L+1
d. for J=1 to N
1. If (C[J] = PV2)
i. C[J] □ PV1
2. end if
e. end for
vii. end if

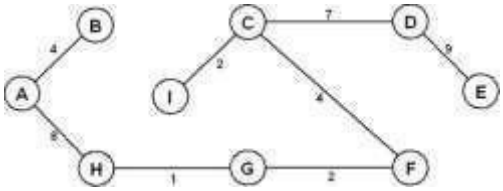
Step 5: display mincost
Step 6: end Kruskals





Cost of MST - 39





Cost of MST -37

- **Sample Input/Output:**

Input: Enter vertices of a graph: 1 2 3 4 5 6 7
Enter vertex wise adjacent vertices & cost of edges.

Vertex	Adjacent Vertex	Cost
Pune (1)	2	1
	4	4
	0	
Mumbai (2)	**1**	**1**
	4	**6**
	5	**4**
	3	**2**
	0	
Bangalore (3)	**5 2 5 2**	
	6	**8**
	0	
Hyderabad (4)	**1**	**4**
	2	**6**
	5	3
	5	3
	7	4
	0	
Chennai (5)	**2**	**4**
	3	**5**
	6	**8**
	7	**7**
	4	**3**
	0	
Delhi (6)	**3**	**8**
	5	**8**

	7 0	**3**
Ahmadabad (7)	**4**	**4**
	5	**7**
	6 0	**3**

OUTPUT : Display of adjacent matrix_adj vertices & cost of associated edges.

Pune (1): (0, 0) (2, 1) (0,0) (4, 4) (0, 0) (0, 0) (0, 0)
Mumbai (2): (1, 1) (0, 0) (3, 2) (4, 6) (5, 4) (0, 0) (0, 0)
Bangalore (3): (0, 0) (2, 2) (0, 0) (0, 0) (5, 5) (6, 8) (0, 0)
Hyderabad (4): (1, 4) (2, 6) (0, 0) (0, 0) (5, 3) (0, 0) (7, 4)
Chennai (5): (0, 0) (2, 4) (3, 5) (4, 3) (0, 0) (6, 8) (7, 7)
Delhi (6): (0, 0) (0, 0) (3, 8) (0, 0) (5, 8) (0, 0) (7, 3)
Ahmadabad (7): (0, 0) (0, 0) (0, 0) (4, 4) (5, 7) (6, 3) (0, 0)

Total cost: 17

Testcases :

a) Display the total number of comparisons required to construct the graph in computer memory.
b) Display the results as given in the sample o/p above.
c) Finally conclude on time & time space complexity for the construction of the graph and for generation of minimum spanning tree using Prim‘s algorithm.

Time Complexity:

Fortheconstructionofanundirectedgraphwith_n‘verticesand_e‘edgesusing adjacency list is O(n+e),since fore very vertex _v‘in Gweneed to store all adjacent edges to vertex v.

- ✓ In Prim‘s algorithm to get minimum spanning tree from an undirected graph with_n‘ vertices using adjacency matrix is $O(n^2)$.
- ✓ Using Kruskal‘salgorithm using adjacency matrix = $O(n^2)$.
  using adjacency list=O(eloge)

CODE FOR PRIM'S ALGORITHM:

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

class graph
{
 int G[20][20],n;

 public:

 void accept()
 {
    int i,j,e;
    int src,dest,cost;
      cout<<"\nEnter the no. of vertices: ";
      cin>>n;

      for(i=0;i<n;i++)
      {
      for(j=0;j<n;j++)
      {
      G[i][j]=0;
      }
      }

      cout<<"\nEnter the no. of  Edges: ";
      cin>>e;



      for(i=0;i<e;i++)
      {
      cout<<"\nEnter Source: ";
       cin>>src;
      cout<<"\nDestination: ";
       cin>>dest;
      cout<<"\nCost: ";
      cin>>cost;

      G[src][dest]=cost;
      G[dest][src]=cost;
     }
```

```
}




void display()
{
  int i,j;
 for(i=0;i<n;i++)
 {
  cout<<"\n";
  for(j=0;j<n;j++)
  {
      cout<<"\t"<<G[i][j];
  }
 }
}



void prims()
{
  int i,j,R[20][20];
  int src,dest,cost,count,min;
  int total=0;
  int visited[20];

  for(i=0;i<n;i++)
  {
      for(j=0;j<n;j++)
      {
          if(G[i][j]==0)
          {
              R[i][j]=999;
          }
      else
      R[i][j]=G[i][j];
      }
  }



  for(i=0;i<n;i++)
  {
```

```
        visited[i]=0;
    }

    cout<<"\nEnter start vertex: ";
    cin>>src;
    visited[src]=1;

    count=0;
    while(count<n-1)
    {
    min=999;

    for(i=0;i<n;i++)
    {

    if(visited[i]==1)
    for(j=0;j<n;j++)
    {
    if(visited[j]!=1)
    {
    if(min>R[i][j])
    {
        min=R[i][j];
        src=i;
        dest=j;
    }
    }
    }
    }


    cout<<"\nEdge from "<<src<<" to "<<dest<<" \twith cost:  "<<min;
    total=total+min;
    visited[dest]=1;
    count++;
    }

        cout<<"\nTotal Cost: "<<total<<"\n";
    }

};


  int main()
  {
```

```
graph g;
g.accept();
g.display();
g.prims();
}
```

OUTPUT FOR PRIM'S ALGORITHM:

Enter the no. of vertices: 4

Enter the no. of Edges: 5

Enter Source: 0

Destination: 1

Cost: 4

Enter Source: 1

Destination: 2

Cost: 6

Enter Source: 0

Destination: 2

Cost: 7

Enter Source: 3

Destination: 1

Cost: 1

Enter Source: 3

Destination: 2

Cost: 5

```
0    4    7    0
4    0    6    1
7    6    0    5
0    1    5    0
Enter start vertex: 0

Edge from 0 to 1      with cost:  4
Edge from 1 to 3      with cost:  1
Edge from 3 to 2      with cost:  5
Total Cost: 10
```

CODE FOR KRUSKAL'S ALGORITHM:

```
#include<iostream>
#define INFINITY 999
using namespace std;

class kruskal{
 typedef struct graph{
     int v1, v2, cost;
        }GR;
        GR G[20];

        public:
        int tot_edges, tot_nodes;
          void create();
          void spanning_tree();
          void get_input();
          int minimum(int);
};

int find(int v2, int parent[]){

    while(parent[v2!=v2]){
            v2 = parent[v2];
        }
        return v2;
}

void un(int i,int j,int parent[]){
    if(i<j)
     parent[j]=i;
     else
        parent[i]=j;
}


void kruskal :: get_input(){
        cout<<"\nEnter number of nodes : "<<endl;
            cin>>tot_nodes;
            cout<<"Enter number of edges : "<<endl;
            cin>>tot_edges;
}

void kruskal :: create(){
    for(int k=0; k<tot_edges; k++){
```

```
        cout<<"Enter V1 and V2 : ";
          cin>>G[k].v1;
          cin>>G[k].v2;
        cout<<"Enter cost : "<<endl;
          cin>>G[k].cost;
        }
}

int kruskal::minimum(int n){
    int i,small,pos;
    small=INFINITY;
        pos=-1;
        for(i=0;i<n;i++){
            if(G[i].cost<small){
                small=G[i].cost;
                pos=i;
            }
        }
        return pos;
}

void kruskal::spanning_tree(){

    int count,k,v1,v2,i,j,tree[10][10],pos,parent[10];
    int sum;
    count=0;
    k=0;
    sum=0;
        for(i=0;i<tot_nodes;i++)
            parent[i]=i;
    while(count!=tot_nodes-1){
                pos=minimum(tot_edges);
            if(pos==-1)
            break;
               v1=G[pos].v1;
               v2=G[pos].v2;
               i=find(v1,parent);
               j=find(v2,parent);
               if(i!=j){
                   tree[k][0]=v1;
                   tree[k][1]=v2;
                k++;
                count++;
                sum+=G[pos].cost;
```

```
                un(i,j,parent);
            }
            G[pos].cost=INFINITY;
        }
    if(count==tot_nodes-1){
         cout<<"\nSpanning tree is: "<<endl;


            for(i=0;i<tot_nodes-1;i++){
                cout<<"|"<<tree[i][0];
                cout<<" ";
                cout<<tree[i][1]<<"|"<<endl;
            }
        cout<<"\n--------------------------------- \n";
        cout<<"Cost of spanning tree is: "<<sum<<endl;
        }
        else{
            cout<<"There is no spanning tree "<<endl;
        }
}


int main(){

    kruskal obj;
    obj.get_input();
    obj.create();
    obj.spanning_tree();
}
```

OUTPUT FOR KRUSKAL'S ALGORITHM

Enter number of nodes :
4
Enter number of edges :
5
Enter V1 and V2 : 1 2
Enter cost :
4
Enter V1 and V2 : 2 3
Enter cost :

```
7
Enter V1 and V2 : 3 4
Enter cost :
7
Enter V1 and V2 : 4 1
Enter cost :
3
Enter V1 and V2 : 3 5
Enter cost :
5

Spanning tree is:
|4 1|
|1 2|
|3 5|

---------------------------------
Cost of spanning tree is: 12
```

CONCLUSION:

Hence we have studied the concept of prims algorithm using adjacency matrix and Kruskal's algorithm by using adjacency list.

ASSIGNMENT-8
Implementation of Dijikstra's algorithm

AIM: Represent a given graph using adjacency matrix /adjacency list and find the shortest path using Dijkstra's algorithm (single source all destination).

OBJECTIVE:

1. To understand the application of Dijkstra's algorithm

THEORY:

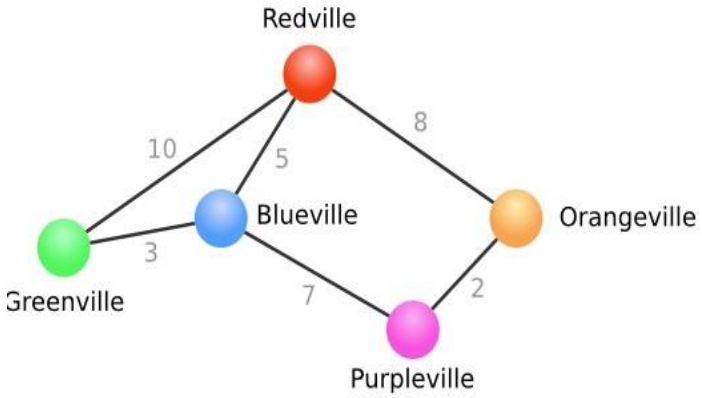
1. Explain in brief with examples how to find the shortest path using Dijkstra's algorithm.

Definition of Dijkstra's Shortest Path

1. To find the shortest path between points, the weight or length of a path is calculated as the sum of the weights of the edges in the path.
2. A path is a shortest if there is no path from x to y with lower weight.
3. Dijkstra's algorithm finds the shortest path from x to y in order of increasing distance from x. That is, it chooses the first minimum edge, stores this value and adds the next minimum value from the next edge it selects.
4. It starts out at one vertex and branches out by selecting certain edges that lead to new vertices.
5. It is similar to the minimum spanning tree algorithm, in that it is "greedy", always choosing the closest edge in hopes of an optimal solution.

Example:

It is easiest to think of the geographical distances, with the vertices being places, such as cities.



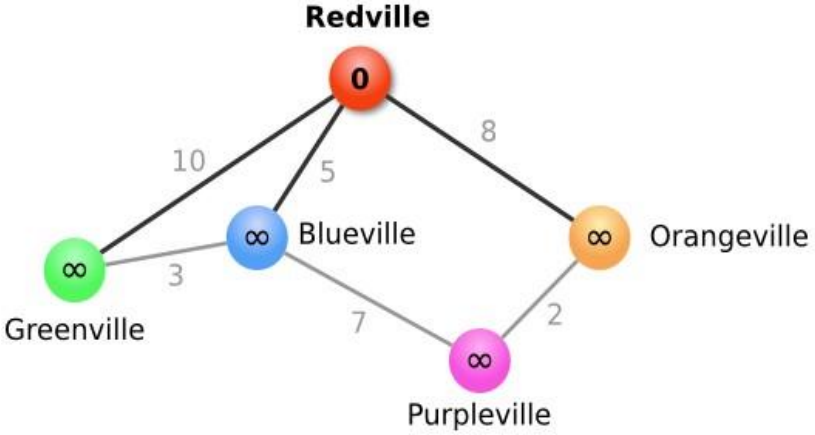
Imagine you live in Redville, and would like to know the shortest way to get to the surrounding towns: Greenville, Blueville, Orangeville, and Purpleville. You would be confronted with problems like: Is it faster to go through Orangeville or Blueville to get to Purpleville? Is it

faster to take a direct route to Greenville, or to take the route that goes through Blueville? As long as you knew the distances of roads going directly from one city to another, Dijkstra's algorithm would be able to tell you what the best route for each of the nearby towns would be.

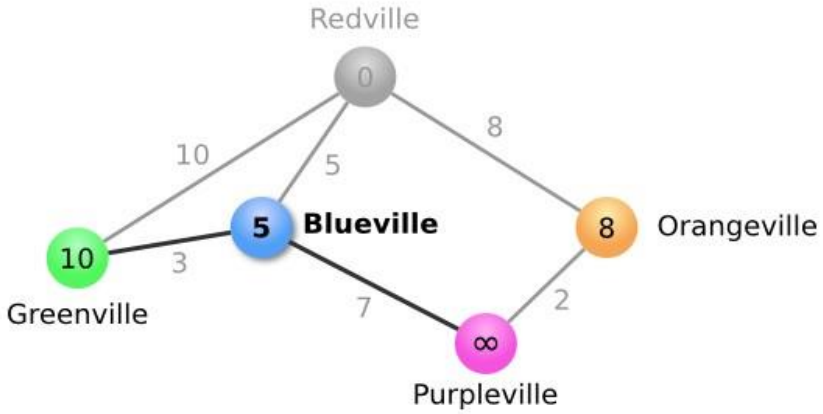
- Begin with the source node (city), and call this the current node. Set its value to 0. Set the value of all other nodes to infinity. Mark all nodes as unvisited.
- For each unvisited node that is adjacent to the current node (i.e. a city there is a direct route to from the present city), do the following. If the value of the current node plus the value of the edge is less than the value of the adjacent node, change the value of the adjacent node to this value. Otherwise leave the value as is.
- Set the current node to visited. If there are still some unvisited nodes, set the unvisited node with the smallest value as the new current node, and go to step 2. If there are no unvisited nodes, then we are done.

In other words, we start by figuring out the distance from our hometown to all of the towns we have a direct route to. Then we go through each town, and see if there is a quicker route through it to any of the towns it has a direct route to. If so, we remember this as our current best route.

Step I:
We set Redville as our current node. We give it a value of 0, since it doesn't cost anything to get to it from our starting point. We assign everything else a value of infinity, since we don't yet know of a way to get to them.

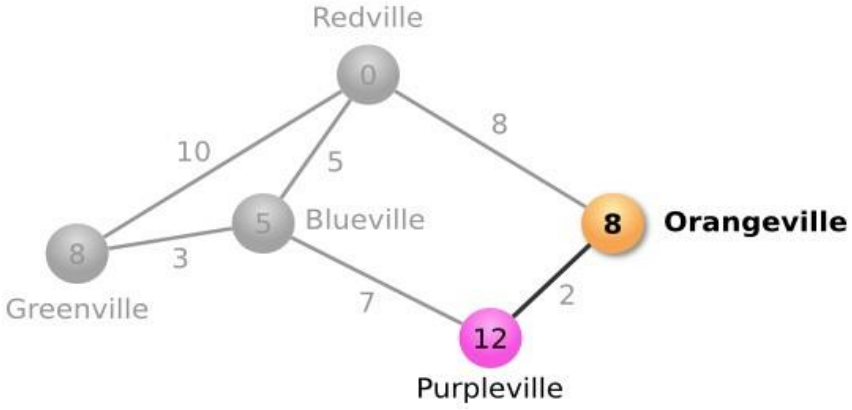


Step II:
Next, we look at the unvisited cities our current node is adjacent to. This means Greenville, Blueville and Orangeville. We check whether the value of the connecting edge, plus the value of our current node, is less than the value of the adjacent node, and if so we change the value. In this case, for all three of the adjacent nodes we should be changing the value, since all of the adjacent nodes have the value infinity. We change the value to the value of the current node (zero) plus the value of the connecting edge (10 for Greenville, 5 for Blueville, 8 for Orangeville). We now mark Redville as visited, and set Blueville as our current node since it has the lowest value of all unvisited nodes.



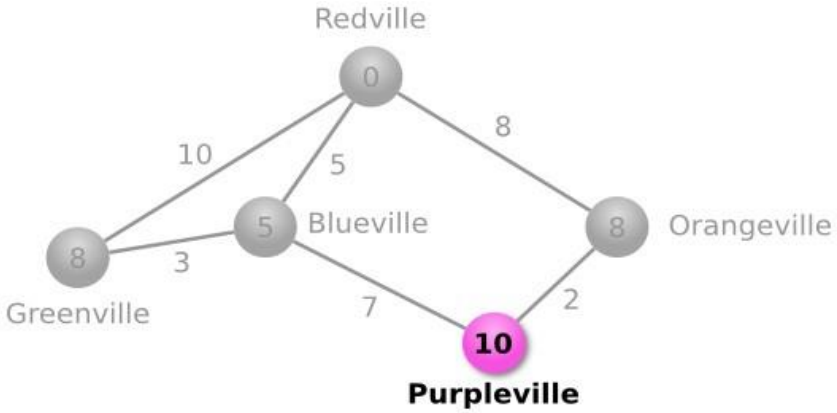
Step III:

The unvisited nodes adjacent to Blueville, our current node, are Purpleville and Greenville. So we want to see if the value of either of those cities is less than the value of Blueville plus the value of the connecting edge. The value of Blueville plus the value of the road to Greenvile is 5 + 3 = 8. This is less than the current value of Greenville (10), so it is shorter to go through Blueville to get to Greenville. We change the value of Greenville to 8, showing we can get there with a cost of 8. For Purpleville, 5 + 7 = 12, which is less than Purpleville‘s current value of infinity, so we change its value as well? We mark Blueville as visited. There are now two unvisited nodes with the lowest value (both Orangeville and Greenville have value 8). We can arbitrarily choose Greenville to be our next current node. However, there are no unvisited nodes adjacent to Greenville! We can mark it as visited without making any other changes, and make Orangeville our next current node.



Step IV:

There is only one unvisited node adjacent to Orangeville. If we check the values, Orangeville plus the connecting road is 8 + 2 = 10, Purpleville's value is 12, and so we change Purpleville's value to 10. We mark Orangeville as visited, and Purpleville is our last unvisited node, so we make it our current node. There are no unvisited nodes adjacent to Purpleville, so we're done!



All above steps can be simply put in a tabular form like this:

Current	Visited	Red	Green	Blue	Orange	Purple	Description
Red		0	Infinity	Infinity	Infinity	Infinity	Initialize Red as current, set initial values
Red		0	10	5	8	Infinity	Change values for Green, Blue, Orange
Blue	Red	0	10	5	8	Infinity	Set Red as visited, Blue as current
Blue	Red	0	8	5	8	12	Change value for Purple
Green	Red, Blue	0	8	5	8	12	Set Blue as visited, Green as current
Orange	Red, Blue, Green	0	8	5	8	12	Set Green as visited, Orange as current
Orange	Red, Blue, Green	0	8	5	8	10	Change value for Purple
Purple	Red, Blue, Green, Orange	0	8	5	8	10	Set Orange as visited, Purple as current
	Red, Blue, Green, Orange, Purple	0	8	5	8	10	Set Purple as visited

ALGORITHM:

College Area represented by Graph.
A graph G with N nodes is maintained by its adjacency matrix Cost. Dijkstra's algorithm find shortest path matrix D of Graph G.
Staring Node is 1.

Step 1: Repeat Step 2 for I = 1 to N
D[I] = Cost[1][I].
Step 2: Repeat Steps 3 & 4 for I = 1 to N

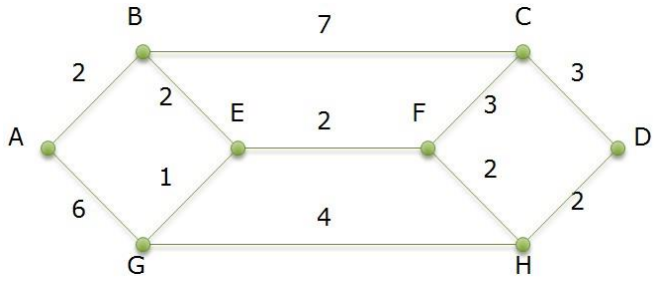
Step 3: Repeat Steps 4 for J = 1 to N
Step 4: If D[J] > D[I] + D[I][J]
Then D[J] = D[I] + D[I][J] Step
5: Stop.

INPUT:

The graph in the form of edges, nodes and corresponding weights, the source node and destination node.

OUTPUT:

The shortest path and length of the shortest path

INPUT	**OUTPUT**	Remark
	Shortest Path is A-B-E-F-H-D Cost - 10	Consider Source Vertex as node A and Destination Vertex as node D

FAQs:

1. What is shortest path?
2. What are the graph representation techniques?
3. What is adjacency Matrix?
4. What is adjacency list?
5. What is adjacency Multi-list?

PROGRAM CODE-

```
#include <iostream>
using namespace std;
class graph{
    int g[20][20];
    int e,v;
    public:
    void accept();
    void display();
    void dijkstra(int start);
};
void graph:: accept(){
    int src, dest, cost, i,j;
    cout<<"Enter the number of vertices:  ";
    cin>>v;
    cout<<"Enter the number of edges:  ";
    cin>>e;
    for(i=0; i<v; i++){
        for(j=0; j<v;j++){
            g[i][j]=0;
        }
    }
    for(i=0; i<e; i++){
        cout<<"\nEnter source vertex:  ";
        cin>>src;
        cout<<"Enter destination vertex:  ";
        cin>>dest;
        cout<<"Enter the cost of the edge:  ";
        cin>>cost;
        g[src][dest]=cost;
        //g[dest][src]=cost;
    }
}
void graph::display(){
    int i,j;
    for(i=0; i<v; i++){
        cout<<endl;
        for(j=0; j<v; j++){
            cout<<g[i][j]<<"\t";
        }
    }
}
void graph::dijkstra(int start){
    int r[20][20], visited[20],distance[20],from[20],i,j,cnt,mindst,next;
    for(i=0; i<v; i++){
        for(j=0; j<v; j++){
```

```
            if(g[i][j]==0){
                r[i][j]=999;
            }
            else{
                r[i][j]=g[i][j];
            }
        }
    }
    for(i=0; i<v; i++){
        visited[i]=0;
        from[i]=start;
        distance[i]=r[start][i];
    }
    distance[start]=0;
    visited[start]=1;
    cnt=v;
    while(cnt>0){
        mindst=999;
        for(i=0; i<v; i++){
            if((mindst>distance[i]) && visited[i]==0){
                mindst=distance[i];
                next=i;
            }
        }
        visited[next]=1;
        for(i=0; i<v; i++){
            if(visited[i]==0 && distance[i]>(mindst+r[next][i])){
                distance[i]=mindst+r[next][i];
                from[i]=next;
            }
        }
        cnt--;
    }
    for(i=0; i<v; i++){
        cout<<"\nDistance of "<<i<< " from "<<start<<" is
"<<distance[i]<<endl<<"Path "<<i;
        j=i;
        do{
            j=from[j];
            cout<<"<-"<<j;
        }
        while(j!=start);
    }
}
int main()
{
    graph g;
```

```
    int s;
    g.accept();
    g.display();
    cout<<"\nEnter the starting vertex: ";
    cin>>s;
    g.dijkstra(s);
return 0;
}
```

OUTPUT-

Enter the number of vertices: 7
Enter the number of edges: 12

Enter source vertex: 1
Enter destination vertex: 2
Enter the cost of the edge: 2

Enter source vertex: 2
Enter destination vertex: 5
Enter the cost of the edge: 10

Enter source vertex: 1
Enter destination vertex: 4
Enter the cost of the edge: 1

Enter source vertex: 2
Enter destination vertex: 4
Enter the cost of the edge: 3

Enter source vertex: 4
Enter destination vertex: 5
Enter the cost of the edge: 2

Enter source vertex: 5
Enter destination vertex: 7
Enter the cost of the edge: 6

Enter source vertex: 4
Enter destination vertex: 7
Enter the cost of the edge: 4

```
Enter source vertex:  4
Enter destination vertex:  6
Enter the cost of the edge:  8

Enter source vertex:  7
Enter destination vertex:  6
Enter the cost of the edge:  1

Enter source vertex:  4
Enter destination vertex:  3
Enter the cost of the edge:  2

Enter source vertex:  3
Enter destination vertex:  6
Enter the cost of the edge:  5

Enter source vertex:  3
Enter destination vertex:  1
Enter the cost of the edge:  4

0    0    0    0    0    0    0
0    0    2    0    1    0    0
0    0    0    0    3    10    0
0    4    0    0    0    0    5
0    0    0    2    0    2    8
0    0    0    0    0    0    0
0    0    0    0    0    0    0
Enter the starting vertex: 1

Distance of 0 from 1 is 999
Path 0<-1
Distance of 1 from 1 is 0
Path 1<-1
Distance of 2 from 1 is 2
Path 2<-1
Distance of 3 from 1 is 3
Path 3<-4<-1
Distance of 4 from 1 is 1
Path 4<-1
Distance of 5 from 1 is 3
Path 5<-4<-1
Distance of 6 from 1 is 8
Path 6<-3<-4<-1
```

Name-Harshita Totala Roll No-SEITA14 Div-IT-A

ASSIGNMENT-9
Heap Sort

AIM: Implement the heap sort to sort given set of values using max or min heap.

Objectives:

1.Understand the concept of Heap Sort.

2. Understand the types of heap sort and its applications.

THEORY :

There are several types of heaps, however in this chapter, we are going to discuss binary heap. A **binary heap** is a data structure, which looks similar to a complete binary tree. Heap data structure obeys ordering properties discussed below. Generally, a Heap is represented by an array. In this chapter, we are representing a heap by ***H***.

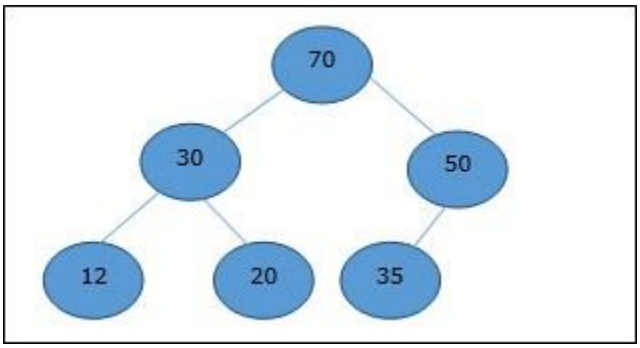
As the elements of a heap is stored in an array, considering the starting index as ***1***, the position of the parent node of **ith** element can be found at ⌊ ***i/2*** ⌋ . Left child and right child of **ith** node is at position ***2i*** and ***2i + 1***.

A binary heap can be classified further as either a ***max-heap*** or a ***min-heap*** based on the ordering property.

Max-Heap

In this heap, the key value of a node is greater than or equal to the key value of the highest child.

Hence, ***H[Parent(i)] ≥ H[i]***

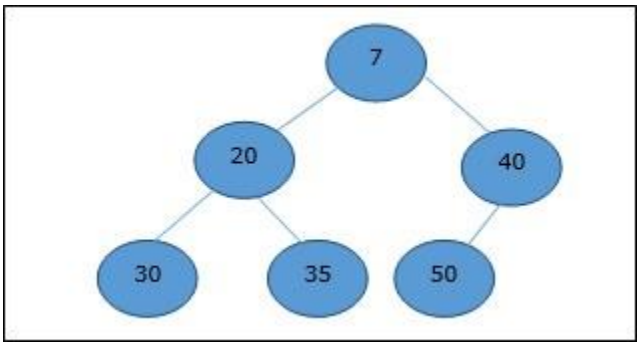


Min-Heap

In mean-heap, the key value of a node is lesser than or equal to the key value of the lowest child.

Hence, ***H[Parent(i)] ≤ H[i]***

In this context, basic operations are shown below with respect to Max-Heap. Insertion and deletion of elements in and from heaps need rearrangement of elements. Hence, **Heapify** function needs to be called.

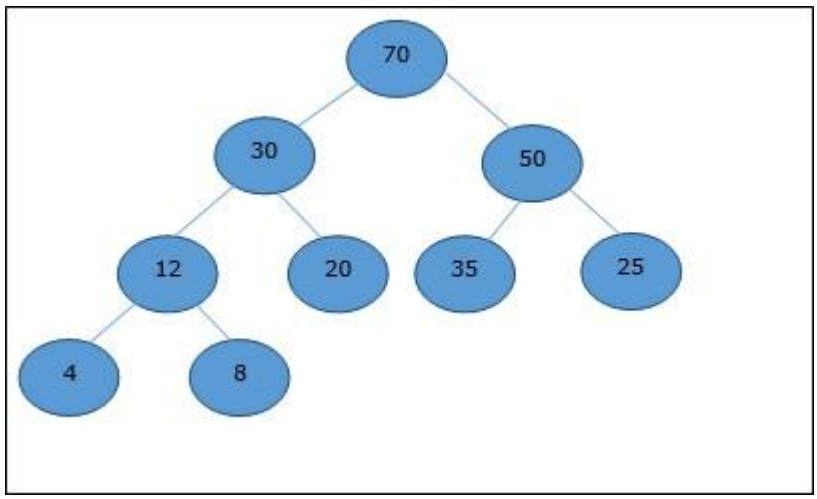


Array Representation

A complete binary tree can be represented by an array, storing its elements using level order traversal.

Let us consider a heap (as shown below) which will be represented by an array **H**.

Name-Harshita Totala Roll No-SEITA14 Div-IT-A



Considering the starting index as **0**, using level order traversal, the elements are being kept in an array as follows.

Index	0	1	2	3	4	5	6	7	8
elements	70	30	50	12	20	35	25	4	8

In this context, operations on heap are being represented with respect to Max-Heap.

To find the index of the parent of an element at index **i**, the following algorithm ***Parent (numbers[], i)*** is used.

```
Algorithm: Parent (numbers[], i)
if i == 1     return NULL  else
[i / 2]
```

The index of the left child of an element at index **i** can be found using the following algorithm, ***Left-Child (numbers[], i)***.

```
Algorithm: Left-Child (numbers[],
i)  If 2 * i ≤ heapsize     return
[2 * i]  else     return NULL
```

The index of the right child of an element at index **i** can be found using the following algorithm, ***Right-Child(numbers[], i)***.

Algorithm: Right-Child (numbers[], i)

```
if 2 * i < heapsize     return [2 * i
+ 1]  else     return NULL
```

Max-Heapify

Heapify method rearranges the elements of an array where the left and right sub-tree of **i**th element obeys the heap property.

Algorithm:

Max-Heapify(numbers[], i)

```
leftchild := numbers[2i]  rightchild
:= numbers [2i + 1]
if leftchild ≤ numbers[].size and numbers[leftchild] >
numbers[i]     largest := leftchild  else     largest
:= i
if rightchild ≤ numbers[].size and numbers[rightchild] >
numbers[largest]     largest := rightchild  if largest ≠
i
   swap numbers[i] with numbers[largest]
   Max-Heapify(numbers, largest)
```

Build-Max-Heap

When the provided array does not obey the heap property, Heap is built based on the following algorithm ***Build-Max-Heap (numbers[])***.

Algorithm:

Build-Max-Heap(numbers[])

```
numbers[].size := numbers[].length  fori
= ⌊ numbers[].length/2 ⌋ to 1 by -1
   Max-Heapify (numbers[], i)
```

Max-Heap-Sort

Algorithm:

1.Build Max Heap from unordered array;2.Find maximum element A[1];
3.Swap elements A[*n*] and A[1]:
now max element is at the end of the array!
4.Discard node *n* from heap(by decrementing heap-size variable)
5.New root may violate max heap property, but itschildren are max heaps. Run max_heapify to fix this.
6.Go to Step 2 unless heap is empty.

Min-Heapify

Heapify method rearranges the elements of an array where the left and right sub-tree of **i**th element obeys the heap property.

Algorithm:

```
Min-Heapify(numbers[], i) kg
leftchild := numbers[2i]  rightchild
:= numbers [2i + 1]
if leftchild ≤ numbers[].size and numbers[leftchild]<
numbers[i]
   smallest := leftchild
else     smallest := i
if rightchild ≤ numbers[].size and numbers[rightchild] <
numbers[smallest]     smallest := rightchild  if
smallest ≠ i
   swap numbers[i] with numbers[smallest]
   Min-Heapify(numbers, smallest)
```

Build-Min-Heap

When the provided array does not obey the heap property, Heap is built based on the following algorithm ***Build-Min-Heap (numbers[])***.

Algorithm:

```
Build-Min-Heap(numbers[])
numbers[].size := numbers[].length  fori
= ⌊ numbers[].length/2 ⌋ to 1 by -1
```

```
Min-Heapify (numbers[], i)
```

Min-Heap-Sort

Algorithm:

1.Build Min Heap from unordered array;2.Find maximum element A[1]; 3.Swap elements A[*n*] and A[1]:
now max element is at the end of the array!
4.Discard node *n* from heap(by decrementing heap-size variable)
5.New root may violate min heap property, but its children are min heaps. Run min_heapify to fix this.
6.Go to Step 2 unless heap is empty.

Name-Harshita Totala Roll No-SEITA14 Div-IT-A

CODE-

```
#include <iostream>
using namespace std;
void maxHeapify(int a[], int i, int n){
 int j, temp;
 temp = a[i];
 j = 2*i;
 while(j<=n){
 if(j<n && a[j+1]<a[j])
 j=j+1;
 if(temp<a[j])
 break;
 else if(temp>=a[j]){
 a[j/2] = a[j];
 j= 2*j;
 }
 }
 a[j/2] = temp;
 return;

}
void min_heapify(int a[], int i, int n){
 int j, temp;
 temp=a[i];
 j=2*i;
 while(j<=n){
 if(j<n && a[j+1]>a[j])
 j=j+1;
 if(temp>a[j])
 break;
 else if(temp<=a[j]){
 a[j/2]=a[j];
 j=2*j;
 }
 }
 a[j/2]=temp;
 return;
}
void build_maxheap(int a[], int n){
```

```
 int i;
 for(i=n/2 ; i>=1; i--){
 maxHeapify(a,i,n);
 }
}
void max_HeapSort(int a[], int n){
 int i, temp;
 for(i=n; i>=2; i--){
 temp = a[i];
 a[i] = a[1];
 a[1] = temp;
 maxHeapify(a, 1, i-1);
 }
}
void build_minheap(int a[], int n){
 int i;
 for(i=n/2; i>=1; i--){
 min_heapify(a,i,n);
 }
}

void min_HeapSort( int a[], int n){
 int i, temp;
 for(i=n; i>=2; i--){
 temp = a[i];
 a[i] = a[1];
 a[1] = temp;
 min_heapify(a, 1, i-1);
 }
}
void print(int arr[], int n){
 cout<<"Sorted data : ";
 for(int i=1; i<=n; i++){
 cout<<"->"<<arr[i];
 }
 return;
}
int main(){
 int n, i, ch;
 cout<<"\t*** Heap Sort ***\n"<<endl;
 cout<<"Enter the number of elements to be sorted: " ;
 cin>>n;
 int arr[n];
 for(i=1; i<=n; i++) {
```

```
 cout<<"Enter element "<<i<<" :";
 cin>>arr[i];
 }
 do{
 cout<<"\n\n1]Heap sort using Max Heap";
 cout<<"\n2]Heap sort using Min heap";
 cout<<"\n3]Exit";
 cout<<"\nEnter your choice: ";
 cin>>ch;
 switch(ch){

case 1:
build_maxheap(arr, n);
max_HeapSort(arr, n);
print(arr, n);

break;
case 2:
 build_minheap(arr, n);
 min_HeapSort(arr, n);
 print(arr, n);
 break;
 case 3:
 cout<<"\nProgram Exited Successfully !!"<<endl;
}
 }while(ch!=3);
return 0;
}
```

Name-Harshita Totala Roll No-SEITA14 Div-IT-A

OUTPUT-

*** Heap Sort ***

Enter the number of elements to be sorted: 5

Enter element 1 :34

Enter element 2 :76

Enter element 3 :55

Enter element 4 :21

Enter element 5 :8

1]Heap sort using Max Heap

2]Heap sort using Min heap

3]Exit

Enter your choice: 1

Sorted data : ->76->55->34->21->8

1]Heap sort using Max Heap

Name-Harshita Totala Roll No-SEITA14 Div-IT-A

2]Heap sort using Min heap

3]Exit

Enter your choice: 2

Sorted data : ->8->21->34->55->76

1]Heap sort using Max Heap

2]Heap sort using Min heap

3]Exit

Enter your choice: 3

Program Exited Successfully !!

Conclusion :

We have constructed max and min heap sort to sort array and perform deletion operation on it.

Assignment-10
Implementation of sequential file

AIM: Department maintains a student information. The file contains roll number, name, division and address. Write a program to create a sequential file to store and maintain student data. It should allow the user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If student record is found it should display the student details.

.

Objective

Type of File

- Binary File ○ The binary file consists of binary data
  - ○ It can store text, graphics, sound data in binary format ○ The binary files cannot be read directly ○ Numbers stored efficiently
- Text File ○ The text file contains the plain ASCII characters
  - ○ It contains text data which is marked by ‗end of line' at the end of each record ○ This end of record marks help easily to perform operations such as read and write ○ Text file cannot store graphical data.

File Organization :

The proper arrangement of records within a file is called as file organization. The factors that affect file organization are mainly the following:

- Storage device
- Type of query
- Number of keys
- Mode of retrieval/update of record Different types of File Organizations are as :
- Sequential file
- Direct or random access file
- Indexed sequential file
- Multi-Indexed file

Sequential file : In sequential file, records are stored in the sequential order of their entry. This is the simplest kind of data organization. The order of the records is fixed. Within each block, the records are in sequence . A sequential file stores records in the order they are entered. New records always appear at the end of the file.

Features of Sequential files :

- Records stored in pre-defined order.
- Sequential access to successive records.
- Suited to magnetic tape.
- To maintain the sequential order updating becomes a more complicated and difficult task. Records will usually need to be moved by one place in order to add (slot in) a

record in the proper sequential order. Deleting records will usually require that records be shifted back one place to avoid gaps in the sequence.

- Very useful for transaction processing where the hit rate is very high e.g. payroll systems, when the whole file is processed as this is quick and efficient.
- Access times are still too slow (no better on average than serial) to be useful in on-line applications.

Drawbacks of Sequential File Organization

- Insertion and deletion of records in in-between positions huge data movement
- Accessing any record requires a pass through all the preceding records, which is time consuming. Therefore, searching a record also takes more time.
- Needs reorganization of file from time to time. If too many records are deleted logically, then the file must be reorganized to free the space occupied by unwanted records

Primitive Operations on Sequential files

Open—This opens the file and sets the file pointer to immediately before the first record
Read-next—This returns the next record to the user. If no record is present, then EOF condition will be set.
Close—This closes the file and terminates the access to the file
Write-next—File pointers are set to next of last record and write the record to the file
EOF—If EOF condition occurs, it returns true, otherwise it returns false
Search—Search for the record with a given key
Update—Current record is written at the same position with updated values

- **Direct or random access file :** Files that have been designed to make direct record retrieval as easy and efficiently as possible is known as directly organized files. Though we search records using key, we still need to know the address of the record to retrieve it directly. The file organization that supports Files such access is called as direct or random file organization. Direct access files are of great use for immediate access to large amounts of information. They are often used in accessing large databases.
- ***Advantages of Direct Access Files :***
  - Rapid access to records in a direct fashion. ○ It doesn't make use of large index tables and dictionaries and therefore response times are very fast.
- **Indexed sequential file :** Records are stored sequentially but the index file is prepared for accessing the record directly. An index file contains records ordered by a record key. The record key uniquely identifies the record and determines the sequence in which it is accessed with respect to other records.
- A file that is loaded in key sequence but can be accessed directly by use of one or more indices is known as an indexed sequential file. A sequential data file that is indexed is called as indexed sequential file. A solution to improve speed of retrieving target is index sequential file. An indexed file contains records ordered by a record key. Each record contains a field that contains the record key.

- This system organizes the file into sequential order, usually based on a key field, similar in principle to the sequential access file. However, it is also possible to directly access records by using a separate index file. An indexed file system consists of a pair of files: one holding the data and one storing an index to that data. The index file will store the addresses of the records stored on the main file. There may be more than one index created for a data file e.g. a library may have its books stored on computer with indices on author, subject and class mark.

Characteristics of Indexed Sequential File

- Records are stored sequentially but the index file is prepared for accessing the record directly
- Records can be accessed randomly
- File has records and also the index
- Magnetic tape is not suitable for index sequential storage
- Index is the address of physical storage of a record
- When randomly very few are required/accessed, then index sequential is better
- Faster access method
- Addition overhead is to maintain index
- Index sequential files are popularly used in many applications like digital library

Primitive operations on Index Sequential files (IS)

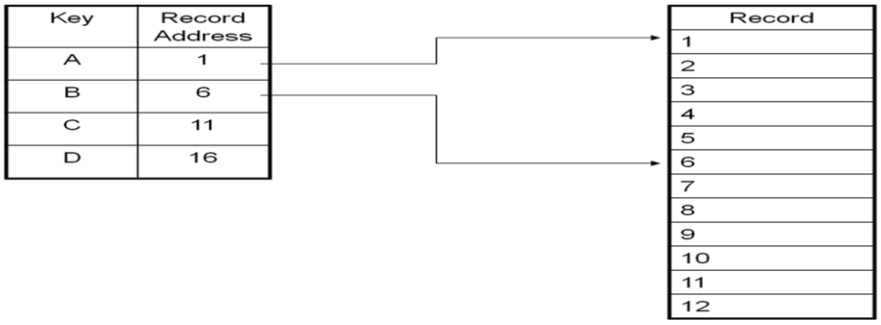
- ***Write (add, store) :*** User provides a new key and record, IS file inserts the new record and key.
- ***Sequential Access (read next) :*** IS file returns the next record (in key order)
- ***Random access (random read, fetch) :*** User provides key, IS file returns the record or "not there"
- ***Rewrite (replace) :*** User provides an existing key and a new record, IS file replaces existing record with new.
- ***Delete :*** User provides an existing key, IS file deletes existing record

Types of Indexed Files : There are two types of indexed files:

- Fully Indexed
- Indexed Sequential

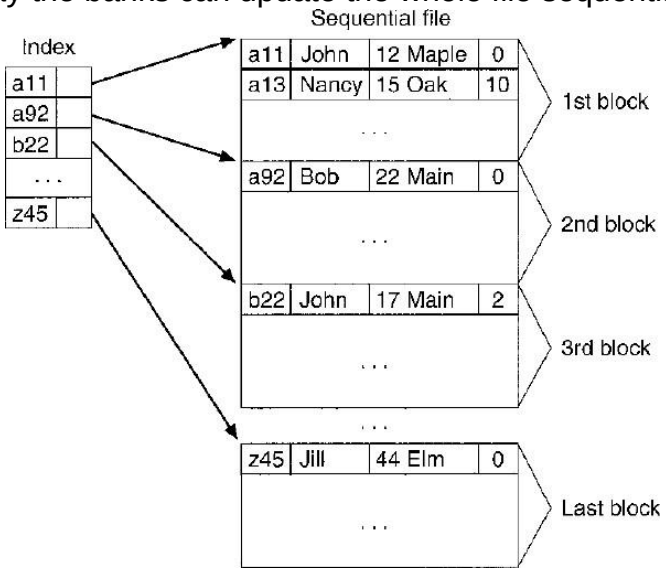
Fully Indexed Files :

An index to a fully indexed file will contain an entry for every single record stored on the main file. The records will be indexed on some key e.g. student number. Very large files will have correspondingly large indices. The index to a (large) file may be split into different index levels. When records are added to such a file, the index (or indices) must also be updated to include their relative position and change the relative position of any other records involved.



Indexed Sequential Files :

This is basically a mixture of sequential and indexed file organisation techniques. Records are held in sequential order and can be accessed randomly through an index. Thus, these files share the merits of both systems enabling sequential or direct access to the data. The index to these files operates by storing the highest record key in given cylinders and tracks. Note how this organisation gives the index a tree structure. Obviously this type of file organisation will require a direct access device, such as a hard disk. Indexed sequential file organisation is very useful where records are often retrieved randomly and are also processed in (sequential) key order. Banks may use this organisation for their auto-bank machines i.e. customers randomly access their accounts throughout the day and at the end of the day the banks can update the whole file sequentially.



Advantages of Indexed Sequential Files

1. Allows records to be accessed directly or sequentially.
2. Direct access ability provides vastly superior (average) access times.

Disadvantages of Indexed Sequential Files

1. The fact that several tables must be stored for the index makes for a considerable storage overhead.
2. As the items are stored in a sequential fashion this adds complexity to the addition/deletion of records. Because frequent updating can be very inefficient, especially for large files, batch updates are often performed.

Multi-Indexed file : In multi-indexed file, the data file is associated with one or more logically separated index files. Inverted files and multilist files are examples of multiindexed files

Algorithms :

1. Algorithm for main function

MAIN FUNCTION()
S1: Read the two filenames from user master and temporary.
S2: Read the operations to be performed from the keyboard
S3: If the operation specified is create go to the create function, if the operation specified is display go to the display function, if the operation specified is add go to the add function , if the operation specified is delete go to delete function, if the operation specified is
display particular record go to the search function, if the operation specified is exit go to step 4. S4: Stop

2. Algorithm for create function

S1: Open the file in the write mode ,if the file specified is not found or unable to open then display error message and go to step5 , else go to step2.
S2: Read the no: of records N to be inserted to the file .
S3: Repeat the step4 N number of times .
S4: Read the details of each student from the keyboard and write the same to the file .
S5: Close the file .
S6: Return to the main function

3. Algorithm for displaying all records S1:
Open the specified file in read mode.
S2: If the file is unable to open or not found then display error message and go to step 4 else go to Step 3
S3: Scan all the student details one by one from file and display the same at the console until end of file is reached.
S4: Close the file
S5: Return to the main function

4. Algorithm for add a record

S1: Open the file in the append mode ,if the file specified is not found or unable to open then display error message and go to step5 , else go to step2
S2: Scan all the student details one by one from file until end of file is reached.
S3: Read the details of the from the keyboard and write the same to the file
S4: Close the file .
S5: Return to the main function

5. Algorithm for deleting a record

S1: Open the file in the append mode ,if the file specified is not found or unable to open then display error message and go to step5 , else go to step2

S2:Accept the roll no from the user to delete the record
S3:Search for the roll no in file.If roll no. exits, copy all the records in the file except the one to be deleted in another temporary file.
S4:Close both files
S5:Now, remove the old file & name the temporary file with name same as that of old file name.

6. Algorithm for displaying particular record(search)

S1: Open the file in the read mode ,if the file specified is not found or unable to open then display error message and go to step6 , else go to step2.
S2: Read the roll number of the student whose details need to be displayed.
S3: Read each student record from the file.
S4: Compare the students roll number scanned from file with roll number specified by the user.
S5: If they are equal then display the details of that record else display required roll number not found message and go to step6.
S6: Close the file.
S7: Return to the main function.

Test Conditions:

1. Input valid filename.
2. Input valid record.
3. Check for opening / closing / reading / writing file errors

Sample Input Output MENU

Choice 1 : Create
Choice 2 : Display
Choice 3 : add
Choice 4 : Delete
Choice 5 : Display particular record
Choice 6 : Exit

Create

Accept the records and write into the file
File created successfully **Display**
Display all records present in Master file. **Add**
Accept the record to be add into the file at the end of the file. Record inserted successfully

Delete

Accept the record to be deleted.
Record deleted successfully **Search**
Accept the record to be displayed by search.
Display whether the record if it is present else record not present.

Program-

```
#include<iostream>
#include<fstream>
using namespace std;
int roll_num;
char name[10];
char division[10];
char address[20];
fstream f,r,temp;
void create(){
int n,i;
 f.open("data.txt",ios::out);
cout<<"How many records you want to enter:-";
cin>>n;
for(i=0; i<n; i++){
cout<<"\nInfo for Student "<<i+1<<endl;
cout<<"Enter Roll No.: ";
cin>>roll_num;
cout<<"Enter Name : ";
cin>>name;
cout<<"Enter Division : ";
cin>>division;
cout<<"Enter Address : ";
cin>>address;
 f<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t "<<address<<"\n";
}
f.close();
 cout<<"\n** Data created succesfully **"<<endl;
}
int search(int roll) {
r.open("data.txt",ios::in);
while(!r.eof()) {
r>>roll_num>>name>>division>>address;
 if(roll == roll_num) {
r.close();
return 1;
}
}
r.close();
return 0;
}
void modify_data(){
 int x,roll;
```

```
 cout<<"Enter roll no: ";
 cin>>roll;
 x = search(roll);
 if(x==1) {
 f.open("data.txt",ios::in);
 temp.open("temp.txt",ios::out);
 f>>roll_num>>name>>division>>address;
 while(!f.eof()) {
 if(roll == roll_num) {
 cout<<"Enter name : ";
 cin>>name;
 cout<<"Enter the division : ";
 cin>>division;
cout<<"Enter address : ";
cin>>address;
 temp<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t"<<address<<"\n";
 f>>roll_num>>name>>division>>address;
 continue;
}
temp<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t"<<address<<"\n";
f>>roll_num>>name>>division>>address;
}
 f.close();
 temp.close();
 f.open("data.txt",ios::out);
 temp.open("temp.txt",ios::in);
 temp>>roll_num>>name>>division>>address;
 while(!temp.eof()) {
f<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t "<<address<<"\n";
temp>>roll_num>>name>>division>>address;
}
f.close();
 temp.close();
}
else{
cout<<"\nRecord not exist!\n";
}
}
void delete_data()
{
int x,roll;
cout<<"Enter roll no to delete: ";
 cin>>roll;
 x = search(roll);
 if(x==1) {
 f.open("data.txt",ios::in);
```

```
 temp.open("temp.txt",ios::out);
 f>>roll_num>>name>>division>>address;
 while(!f.eof()){
 if(roll != roll_num) {
 temp<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t" <<address<<endl;
}
f>>roll_num>>name>>division>>address;
}
f.close();
temp.close();
f.open("data.txt",ios::out);
temp.open("temp.txt",ios::in);
temp>>roll_num>>name>>division>>address;
 while(!temp.eof()) {
f<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t "<<address<<"\n";
 temp>>roll_num>>name>>division>>address;
 }
 f.close();
 temp.close();
 }
 else {
cout<<"\nRecord not exist!\n";
}
}
void insert(){
int x,i,roll;
cout<<"Enter Stduent Details : "<<endl;
cout<<"Roll no : ";
 cin>>roll_num;
 x = search(roll);
if(x==1) {
cout<<"\nRecord already exist!\n";
}
else {
f.open("data.txt",ios::app);
 cout<<"Name : ";
 cin>>name;
 cout<<"Division : ";
 cin>>division;
cout<<"Address : ";
cin>>address;
 roll_num = roll;
 f<<roll_num<<"\t"<<name<<"\t\t "<<division<<"\t\t "<<address<<endl;
f.close();
 cout<<"\n** Record added succesfully **"<<endl;
 }
```

```
}
void display()
{
f.open("data.txt",ios::in);
cout<<"Roll No"<<"\t "<<"Name"<<"\t "<<"Division"<<"\t"<<"Address"<<"\n";
f>>roll_num>>name>>division>>address;
 while(!f.eof()) {
cout<<roll_num<<"\t "<<name<<" \t\t"<<division<<"\t\t"<<address<<endl;
 f>>roll_num>>name>>division>>address;
}
f.close();
}
int main(){
int choice;
cout<<"Weclome to Student Database Management\n"<<endl;
do{
cout<<"What do you want to perform ? Choose from options given below :"<<endl;
cout<<"1] Create DB"<<endl;
cout<<"2] Display DB"<<endl;
cout<<"3] Insert Data"<<endl;
cout<<"4] Delete Data"<<endl;
cout<<"5] Modify Data"<<endl;
cout<<"6] Exit"<<endl;
cout<<"\nEnter a choice : ";
cin>>choice;
switch(choice){
case 1:
create();
cout<<"\n";
break;
case 2:
display();
cout<<"\n";
break;
case 3:
insert();
cout<<"\n";
break;
case 4:
delete_data();
cout<<"\n";
display();
break;
case 5:
modify_data();
cout<<"\n";
```

```
break;
case 6:
cout<<"Thank You !! Your Data has been saved .";
cout<<"\n";
break;
default:
cout<<"*** Enter a valid choice!! ***";
cout<<"\n";
}
}while(choice!=6);
}
```

Output-

```
Weclome to Student Database Management

What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
5] Modify Data
6] Exit

Enter a choice : 1
How many records you want to enter:-3

Info for Student 1
Enter Roll No.: 12
Enter Name : harsh
Enter Division : g
Enter Address : pune

Info for Student 2
Enter Roll No.: 34
Enter Name : divya
Enter Division : s
Enter Address : mumbai

Info for Student 3
Enter Roll No.: 90
Enter Name : priya
Enter Division : h
Enter Address : nashik

** Data created succesfully **

What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
5] Modify Data
6] Exit
```

```
Enter a choice : 2
Roll No  Name    Division       Address
12       harsh         g             pune
34       divya         s             mumbai
90       priya         h             nashik

What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
5] Modify Data
6] Exit

Enter a choice : 3
Enter Stduent Details :
Roll no : 67
Name : om
Division : b
Address : abad

** Record added succesfully **

What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
5] Modify Data
6] Exit

Enter a choice : 4
Enter roll no to delete: 90

Roll No  Name    Division       Address
12       harsh         g             pune
34       divya         s             mumbai
0        om            b             abad
What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
```

5] Modify Data
6] Exit

Enter a choice : 5
Enter roll no: 67

Record not exist!

What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
5] Modify Data
6] Exit

Enter a choice : 5
Enter roll no: 12
Enter name : Divya
Enter the division : w
Enter address : pune

What do you want to perform ? Choose from options given below :
1] Create DB
2] Display DB
3] Insert Data
4] Delete Data
5] Modify Data
6] Exit

Conclusion:

Thus we have implemented sequential file and performed all the primitive operations on it.