INDEX

Sr. No.	Practical	Date	Sign	Remarks
1	**Implementation of different sorting techniques** a) To implement bubble sort b) To implement insertion sort c) To implement selection sort d) To implement shell sort e) To implement radix sort f) To implement quick sort	27/12/2022		
2	**Implementation of different searching techniques** a) To implement linear search b) To implement binary search	27/12/2022		
3	**Implementation of Stacks (Using arrays and Linked List)** a) To implement Stack using array b) To implement Stack using linked list	27/12/2022		
4	**Implementation of Stack Application** a) To implement Postfix evaluation b) To implement Balancing of Parenthesis	28/12/2022		
5	**Implement all different types of queues** a) To implement Circular Queue	28/12/2022		
6	**Demonstrate application of queue** a) To implement Priority Queue	01/01/2023		
7	**Implementation of all types of linked list** a) To implement Single Linked List b) To implement Double Linked list c) To implement Circular Linked List	09/01/2023		
8	**Demonstrate application of linked list** a) To implement Polynomial Addition b) To implement Sparse Matrix	09/01/2023		

9	**Create and perform various operations on BST** a) Inserting node in BST b) Deleting the node from BST c) To find height of Tree d) To perform Inorder e) To perform Preorder f) To perform Postorder g) To find Maximum value of tree	09/01/2023	
10	**Implementing Heap with different operations performed** a) To perform insertion operation b) To create Heap using Heapify method c) To perform Heap sort d) To delete the value in heap	09/01/2023	
11	**Create a graph storage structure** a) Adjacency Matrix	10/02/2023	
12	**Perform various hashing techniques with Linear Probe as collision resolution scheme.**	10/02/2023	
13	**Create a minimum spanning tree using any method Kruskal's algorithm or Prim's algorithm**	14/02/2023	
14	**Implementation of Graph Traversal** a) Implement Depth First Search (DFS) b) Implement Breath First Search (BFS)	14/02/2023	

Practical No. 01

Problem Statement: **Implementation of different Sorting Techniques.**

❖ Bubble Sort

➢ Algorithm:

BubbleSort(int a[], int n)

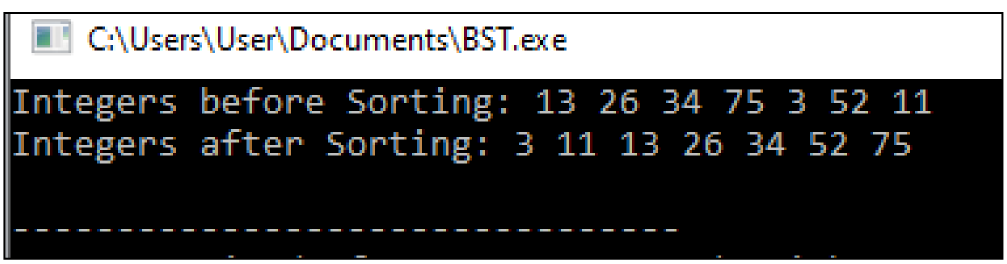
Step 1: Begin
Step 2: for i = 1 to n-1
sorted = true
Step 3: for j = 0 to n-1-i
if a[j] > a[j+1]
temp = a[j]
Step 4: a[j] = a[j+1]
a[j+1] = temp
sorted = false
end for
Step 5: if sorted
break from i loop
end for
Step 6: End

➢ Code:

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

void Bubble(int entry[],int size)
{
for(int round=1;round<size;round++)
{
 for(int i=0;i<size-round;i++)
 {
 if(entry[i]>entry[i+1])
 {
 int temp=entry[i];
```

```
 entry[i]=entry[i+1];
 entry[i+1]=temp;
 }  }
}   }

void Display(int entry[], int size)
{
 for(int i=0;i<size;i++)
 {
 cout<<entry[i]<<" ";
 }
 cout<<endl;
}
int main()
{
 int entryInt[]={13,26,34,75,3,52,11};
 cout<<"Integers before Sorting: ";
 Display(entryInt,7);
 cout<<"Integers after Sorting: ";
 Bubble(entryInt,7);
 Display(entryInt,7);
 return 0;
}
```

- **Output:**



❖ Insertion Sort

➢ Algorithm:

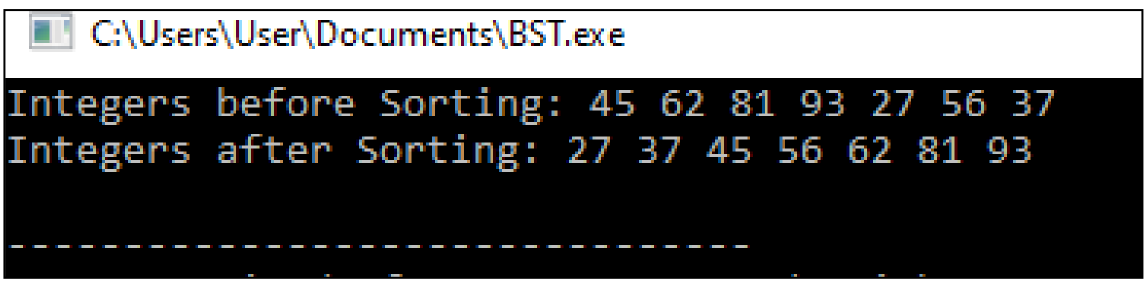
insertionSort(array A)
Step 1: begin
for i = 1 to length[A] - 1 do
Step 2: begin
value = A[i];
Step 3: j = i - 1;
while j >= 0 and A[j] > value do
Step 4: begin
A[j + 1] = A[j];
j = j - 1;
end;
Step 5: A[j + 1] = value;
end;
end;

➢ Code:

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

void Insertion(int entry[],int size)
{
for(int i=1;i<size;i++)
 {
 int live=entry[i];
 int k;
 for(k=i-1;k>=0 && entry[k]>live;k--)
 entry[k+1]=entry[k];
 entry[k+1]=live;
 }
}

void Display(int entry[], int size)
{
 for(int i=0;i<size;i++)
 {
```

```
 cout<<entry[i]<<" ";
 }
 cout<<endl;
}

int main()
{
 int entryInt[]={45,62,81,93,27,56,37};
 cout<<"Integers before Sorting: ";
 Display(entryInt,7);
 cout<<"Integers after Sorting: ";
 Insertion(entryInt,7);
 Display(entryInt,7);
 return 0;
}
```

- **Output:**



❖ Selection Sort

➢ Algorithm:

SelectionSort(array A)

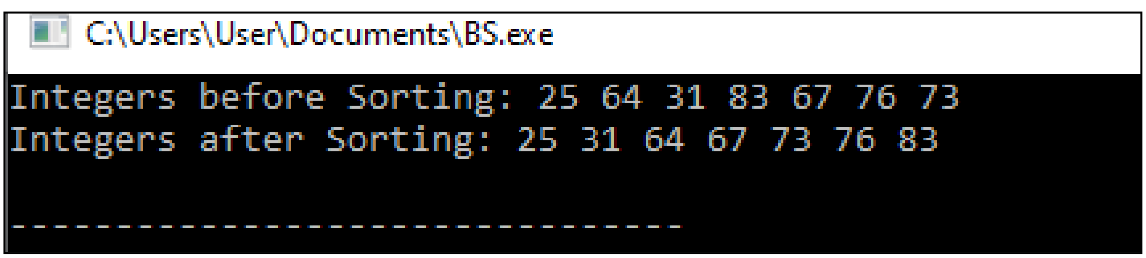
```
Step 1:     begin
            For I = 0 to N-1 do:
            Smallsub = I
Step 2:     For J = I + 1 to N-1 do:
             If A(J) < A(Smallsub)
             Smallsub = J
             End-If
             End-For
Step 3:     Temp = A(I)
             A(I) = A(Smallsub)
             A(Smallsub) = Temp
             End-For
```

➢ Code:

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

void Selection(int entry[],int size)
{
 for(int i=0;i<size;i++)
 {
 int LiveMin=entry[i];
 int LiveMinIndex=i;
 for(int j=i+1;j<size;j++)
 {
 if(LiveMin>entry[j])
 {
 LiveMin=entry[j];
 LiveMinIndex=j;
 } }
 if(LiveMinIndex != i)
 {
 entry[LiveMinIndex]=entry[i];
 entry[i]=LiveMin;
```

```
 }    }         }
void Display(int entry[], int size)
{
 for(int i=0;i<size;i++)
 {
 cout<<entry[i]<<" ";
 }
 cout<<endl;
}
int main()
{
 int entryInt[]={25,64,31,83,67,76,73};
 cout<<"Integers before Sorting: ";
 Display(entryInt,7);
 cout<<"Integers after Sorting: ";
 Selection(entryInt,7);
 Display(entryInt,7);
 return 0;
}
```

- **Output:**



❖ Shell Sort

➢ Algorithm:

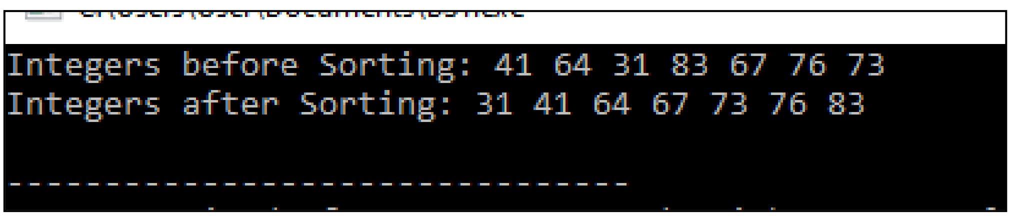
Step 1: Caculate gap size
Step 2: WHILE gap is greater than 0
FOR each element of the list, that is gap apart
Extract the current item
Step 3: Locate the position to insert
Insert the item to the position
END FOR
Step 4: Calculate gap size
END WHILE

➢ Code:

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

void Shell(int entry[],int size)
{
 int gap,i,j;
 for(int gap=size/2;gap>=1;gap=gap/2)
 {
 for(j=gap;j<size;j++)
 {
 for(i=j-gap;i>=0;i=i-gap)
 {
 if(entry[i+gap]>entry[i]){
 break;
 }
 else
 {
 int temp=entry[i+gap];
 entry[i+gap] =entry[i];
 entry[i]=temp;
 } } }
}}
void Display(int entry[], int size)
{
```

```
 for(int i=0;i<size;i++)
 {
 cout<<entry[i]<<" ";
 }
 cout<<endl;
}
int main()
{
 int entryInt[]={41,64,31,83,67,76,73};
 cout<<"Integers before Sorting: ";
 Display(entryInt,7);
 cout<<"Integers after Sorting: ";
 Shell(entryInt,7);
 Display(entryInt,7);
 return 0;
}
```

- **Output:**



❖ Radix Sort

➢ Algorithm:

Radix-Sort(A, d)

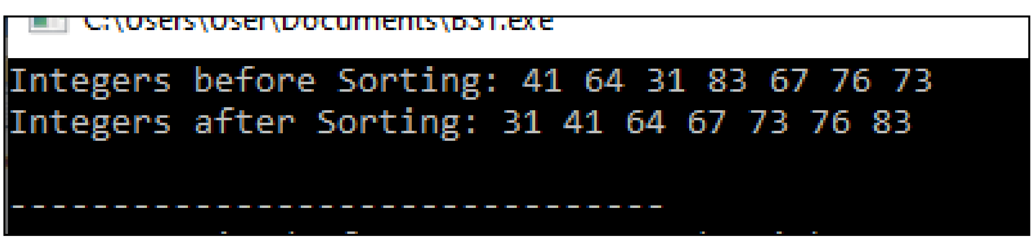
Step 1: for j = 1 to d do
    int count[10] = {0};
Step 2: for i = 0 to n do
    count[key of(A[i]) in pass j]++
Step 3: for k = 1 to 10 do
    count[k] = count[k] + count[k-1]
Step 4: for i = n-1 downto 0 do
    result[count[key of(A[i])]] = A[j]
    count[key of(A[i])]--
Step 5: for i=0 to n do
    A[i] = result[i]
Step 6: end for(j)
    end func

➢ Code:

```
#include <iostream>
using namespace std;
int FindMax(int entry[],int size)
{
 int large =entry[0];
 for(int i=1;i<size;i++)
 if(entry[i]>large)
 large=entry[i];
 return large;
}
void num(int entry[],int size,int base)
{
 int n[10]={0};
 int i;
 int res[size];
 for(i=0;i<size;i++)
 n[(entry[i]/base)%10]++;
 for(i=1;i<=9;i++)

 n[i]=n[i]+n[i-1];
```

```
for(i=size-1;i>=0;i--)
{
res[n[(entry[i]/base)%10]-1]=entry[i];
n[(entry[i]/base)%10]--;
}
for(i=0;i<size;i++)
entry[i]=res[i];}
void Radix(int entry[],int size){
int large=FindMax(entry,size);
int base;
for(base=1;large/base>0;base=base*10)
{
num(entry,size,base);
}}
void Display(int entry[], int size){
for(int i=0;i<size;i++){
cout<<entry[i]<<" "; }
cout<<endl;}
int main()
{
int entryInt[]={41,64,31,83,67,76,73};
cout<<"Integers before Sorting: ";
Display(entryInt,7);
cout<<"Integers after Sorting: ";
Radix(entryInt,7);
Display(entryInt,7);
return 0;
}
```

- **Output:**



❖ <u>Quick Sort</u>

➢ Algorithm:

QuickSort(double[] a)

Step 1: if (a.length ≤ 1)
return;

Step 2: Select a pivot;
Partition a[] in 2 halves:
left[]: elements ≤ pivot
right[]: elements > pivot;

Step 3: Sort left[];
Sort right[];

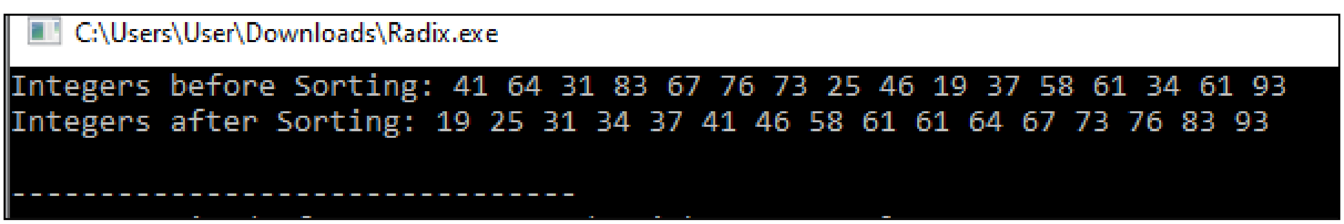
Step 4: Concatenate: left[] pivot right[]

➢ Code:

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

static void Quick(int entry[], int lside, int rside);
static int part(int entry[], int lside, int rside);
static void Display(int entry[], int size);

static void Quick(int entry[], int lside, int rside) {
  if (lside < rside) {
    int rotate = part(entry, lside, rside);
    Quick(entry, lside, rotate-1);
    Quick(entry, rotate+1, rside);
  }
}

static int part(int entry[], int lside, int rside) {
  int i = lside;
  int rotate = entry[rside];
  int temp;
  for(int j = lside; j <=rside; j++) {
    if(entry[j] < rotate) {
      temp = entry[i];
      entry[i] = entry[j];
```

```
      entry[j] = temp;
      i++;
    }
  }
  temp = entry[rside];
  entry[rside] = entry[i];
  entry[i] = temp;
  return i;
}
void Display(int entry[], int size)
{
 for(int i=0;i<size;i++)
 {
 cout<<entry[i]<<" ";
 }
 cout<<endl;
}
int main()
{
 int entryInt[]={41,64,31,83,67,76,73,25,46,19,37,58,61,34,61,93};
 cout<<"Integers before Sorting: ";
 Display(entryInt,16);
 int n = sizeof(entryInt) / sizeof(entryInt[0]);
 cout<<"Integers after Sorting: ";
 Quick(entryInt,0,n-1);
 Display(entryInt,16);
 return 0;
}
```

- **Output:**



Practical No. 02

Problem Statement: **Implementation of Searching Techniques.**

- ## Linear Search

  - **Algorithm:**

    LinearSearch(a,n,item,loc)
    Step 1: Begin
    Step 2: for i = 0 to (n - 1) by 1 do
    Step 3: if (a[i] = item) then
    set loc = i
    Exit
    Step 4: endif
    endfor
    Step 5: set loc = -1
    End

  - **Code:**

```
#include<iostream>
using namespace std;
int LinearSearch(int arr[],int sel,int n)
{
 int loc;
 for(int i=0;i<n;i++)
 {
 if(arr[i]==sel)
 {
 loc=i;
 break;
 }
 else
 {
 loc=0;
 }
 }
```

```
 return loc;
}
int main()
{
 int n,sel,loc,arr[5];
 cout<<"Series Size: ";
 cin>>n;
 cout<<endl<<"Values of Series: ";
 for(int i=0;i<n;i++)
 cin>>arr[i];
 cout<<endl<<"Value to be Searched: ";
 cin>>sel;
 loc=LinearSearch(arr,sel,n);
 if(loc==-1)
 cout<<endl<<"No such Value Exist!";
 else
 cout<<endl<<"Index of the Value is: "<<loc;
 return loc;
}
```

- **Output:**

```
C:\Users\User\Downloads\LS.exe
Series Size: 7

Values of Series: 64 83 29 57 19 28 44

Value to be Searched: 57

Index of the Value is: 3
--------------------------------
Process exited after 40.57 seconds with return value 3
Press any key to continue . . .
```

❖ Binary Search

➢ Algorithm:

BinarySearch

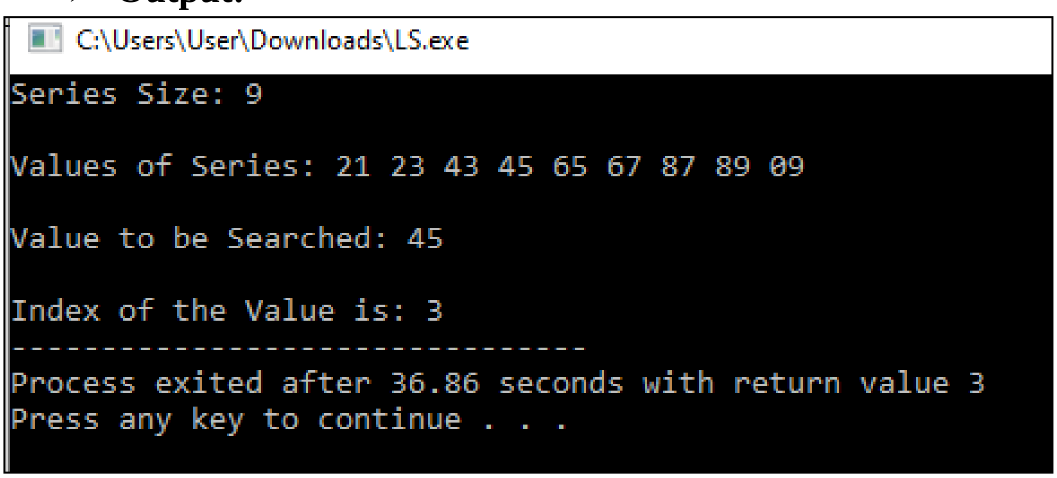
```
Step 1:   Begin
Step 2:   Set beg = 0
          Set end = n-1
          Set mid = (beg + end) / 2
Step 3:   while ( (beg <= end) and (a[mid] ≠ item) ) do
Step 4:   if (item < a[mid]) then
          Set end = mid - 1
          else
          Set beg = mid + 1
          endif
Step 5:   Set mid = (beg + end) / 2
          Endwhile
Step 6:   if (beg > end) then
          Set loc = -1
          else
          Set loc = mid
          Endif
Step 7:   End
```

➢ Code:

```
#include<iostream>
using namespace std;
int BinarySearch(int arr[],int sel,int n)
{
 int  begin=0;
 int stop=n-1;
 int loc;
 int cen=int((begin+stop)/2);
 while(begin<=stop && arr[cen]!=sel){
 if(sel<arr[cen])
 stop=cen-1;
 else
 begin=cen+1;
 cen=int((begin+stop)/2);
 }
```

```
if(arr[cen]==sel)
loc=cen;
else
loc=-1;
return cen;
}
int main()
{
int n,sel,loc,arr[5];
cout<<"Series Size: ";
cin>>n;
cout<<endl<<"Values of Series: ";
for(int i=0;i<n;i++)
cin>>arr[i];
cout<<endl<<"Value to be Searched: ";
cin>>sel;
loc=BinarySearch(arr,sel,n);
if(loc==-1)
cout<<endl<<"No such Value Exist!";
else
cout<<endl<<"Index of the Value is: "<<loc;
return loc;
}
```

- **Output:**



Practical No. 03

Problem Statement: **Implementation of Stacks.**

- **Implementation of Stack using Array**

  - **Code:**

```
#include<iostream>
using namespace std;
#define MSize 25
int stack[MSize];
int top = -1;
void push(){
 int item;
 if(top==MSize-1){
 cout<<"STACK FULL\n";
 }
 else{
 cout<<"Enter values to be intrested: ";
 cin>>item;
 stack[++top]=item;
 }
}
void pop(){
 int item;
 if(top==-1)
 cout<<"EMPTY STACK"<<endl;
 else{
 item=stack[top-1];
 cout<<"Deleted Element: "<<item;
 }
}
void traverse(){
 if(top==-1)
 cout<<"EMPTY STACK"<<endl ;
 else{
 cout<<"Values in Stack: "<<endl ;
 for(int i=top;i>=0;i--)
 cout<<"\n"<<stack[i];
```

```
 }
}
int main(){
 int choice;
 char ch;
 do{
 cout<<"**** Stack Operation ****\n\n";
 cout<<"1-Push Value\n\n2- Pop Value\n\n3- Traverse\n\n4-Exit\n";
 cin>>choice;
 switch(choice){
 case 1:
 push();
 break;
 case 2:
 pop();
 break;
 case 3:
 traverse();
 break;
 default:
 cout<<"\n Invalid Choice";
 }
 cout<<"\n Enter (Y|y) to Continue.";
 cin>>ch;
 }
 while(ch=='Y'||ch=='y');
 return 0;
}
```

- **Output:**

```
C:\Users\User\Downloads\LS.exe
**** Stack Operation ****

1-Push Value

2- Pop Value

3- Traverse

4-Exit
1
Enter values to be intrested: 45

 Enter (Y|y) to Continue.y
**** Stack Operation ****

1-Push Value

2- Pop Value

3- Traverse

4-Exit
3
Values in Stack:

45
 Enter (Y|y) to Continue.y
**** Stack Operation ****

1-Push Value

2- Pop Value

3- Traverse

4-Exit
2
Deleted Element: 0
 Enter (Y|y) to Continue.
```

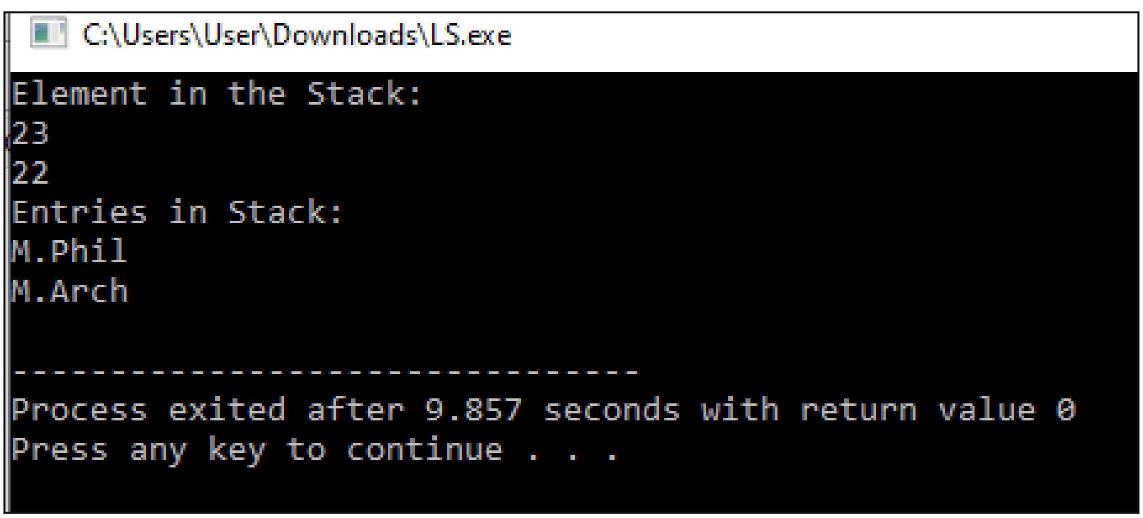
❖ Implementation of Stack using Linked List

➢ Code:

```
#include<iostream>
using namespace std;
template<typename E>
class SNode{
 public:
 E element;
SNode<E>* next;};
template<typename E>
class SLinkL{
 public:
 SLinkL();
 ~SLinkL();
 void addFore( const E& e);
 void del();
 void show();
 private:
 SNode<E>* head;};
template<typename E>
SLinkL<E>::SLinkL(){
head=NULL; }
template<typename E>
SLinkL<E>::~SLinkL(){
del();}
template<typename E>
void SLinkL<E>::addFore( const E& e){
 SNode<E>* v=new SNode<E>;
 v-> element =e;
 v->next=head;
 head=v;}
template<typename E>
void SLinkL<E>::del(){
if(head==NULL)
cout<<"Stack is Empty";
else {
 SNode<E>* old=head;
 head=old->next;
```

```
 delete old; }}
template<typename E>
void SLinkL<E>::show(){
SNode<E>* T;
 for(T=head; T!=NULL;T=T->next)
 cout<<T->element<<endl;
}
int main()
{
 SLinkL <int> a;
 a.addFore(22);
 a.addFore(23);
 cout<<"Element in the Stack: "<<endl;
 a.show();
 SLinkL <string> b;
 b.addFore("M.Arch");
 b.addFore("M.Phil");
 cout<<"Entries in Stack: "<<endl;
 b.show();
 return 0;
}
```

- **Output:**



Practical No. 4: Implementation of stack application

a) Aim: Write a program in c++ to implement postfix evaluation

Algorithm:

Step 1: Declare an Integer Stack.

Step 2: Scan postfix expression from left to right and repeat step 3 and 4 for each element of the expression until end of the expression.

Step 3: If an operand is encountered, put it on stack.

Step 4: If an operator @ is encountered, then

- Remove the top two elements of stack, where A is the top element and B is the top-1 position.
- Evaluate B @ A

Step 5: Set the result as the top element in stack.

Step 6: Exit

Code:

```
#include<iostream>

using namespace std;

#define MAXSIZE 20

int stack[MAXSIZE];

int top=-1;

int pop()

{

        int item;

        if(top==-1)

        cout<<"the stack is empty\n";

        else

        item=stack[top--];

        return item;
```

```
}
void push(int item)
{
	if(top==MAXSIZE-1)
	cout<<"the stack is full\n";
	else
	stack[++top]=item;
}
void evaluatePostfix(char expr[])
{
	int i;
	for(i=0;expr[i];++i)
	{
		if(expr[i]==' ')
		continue;
		else if(isdigit(expr[i]))//reading integer
		{
			int num=0;
			while(isdigit(expr[i]))
			{
				num=num*10+(int)(expr[i]-'0');
				i++;
			}
			push(num);
			//push(expr[i]-'0');
		}
		else
		{
```

```
                    int A=pop();
                    int B=pop();
                    switch(expr[i])
                    {
                        case '+':
                            push(B+A);
                            break;
                        case '-':
                            push(B-A);
                            break;
                        case '*':
                            push(B*A);
                            break;
                        case '/':
                            push(B/A);
                            break;
                    }
                }
            }
            cout<<stack[top];
        }
        int main()
        {
            char expr[]="50 20 + 3 * 9 3 / -";
            evaluatePostfix(expr);
            return 0;
}
```

Output:

```
207
--------------------------------
Process exited after 0.146 seconds with return value 0
Press any key to continue . . .
```

b) Aim: Write a program in c++ to implement balancing of parenthesis

Algorithm:

Step 1: Declare a char stack

Step 2: Now traverse the expression from left to right till end of the expression.

- If the character is open bracket ‘(‘ or ‘{‘ or ‘[‘ then push on stack
- If the character is closing bracket ‘)’ or ‘}’ or ‘]’ then the top character.
- If the popped character is matching with the open bracket then it is balanced otherwise not balanced.

Step 3: After complete traversal if there is any open bracket left in stack then it is also not balanced.

Code:

```
#include<iostream>

#include<string.h>

using namespace std;

#define MAX 20

class Stack //creating stack

{

        public:

                char stk[MAX];

                int top;

};

Stack s;//stk and top
```

```
//to access stk and top we have to write s.stk[],s.top
void push(char item)
{
	if(s.top==(MAX-1))
	cout<<"stack is full\n";
	else
	{
		s.top++;//0,1,2,3,4
		s.stk[s.top]=item;
	}
}
void pop()
{
	if(s.top==-1)
	{
		cout<<"stack is empty\n";
	}
	else
	{
		s.top=s.top-1;
	}
}
bool balancedParenthesis(string expr)
{
	int i=0;
	char x;
	s.top=-1;
	for(i=0;i<expr.length();i++)
```

```
{
    if(expr[i]=='('||expr[i]=='{'||expr[i]=='[')
    {
        push(expr[i]);
        continue;
    }
    switch(expr[i])
    {
        case ')':
            x=s.stk[s.top];
            pop();
            if(x=='{'||x=='[')
            return false;
            break;
        case '}':
            x=s.stk[s.top];
            pop();
            if(x=='('||x=='[')
            return false;
            break;
        case ']':
            x=s.stk[s.top];
            pop();
            if(x=='{'||x=='(')
            return false;
            break;
    }
}
```

```
        if(i==expr.length()&&s.top==-1)
        cout<<"stack is balanced\n";
        else
        cout<<"stack is not balanced\n";
}
int main()
{
        string expr;
        cout<<"enter the expression\n";
        cin>>expr;
        balancedParenthesis(expr);
        return 0;
}
```

Output:

```
enter the expression
()[{}]
stack is balanced

--------------------------------
Process exited after 32.5 seconds with return value 0
Press any key to continue . . .
```

Practical No. 5: Implement all different types of queues.

a) Aim: Write a program in c++ to implement circular queue

Algorithm: Circular queue

Step 1: Include all the header files which are used in a program and define a constant SIZE with specific value.

Step 2: Declare all user defined functions used in circular queue implementation.

Step 3: Create a one dimensional array with above defined SIZE (int cQueue[SIZE])

Step 4: Define two integer variables front and rear and initialize both with -1 (int front = -1, rear = -1)

Algorithm: Enqueue (insert values into queue)

Step 1: Check whether queue is FULL. ((rear = = SIZE-1 && front = = 0 || (front = = rear+1))

Step 2: If it is FULL, then display “Queue is full” and terminate the function

Step 3: If it is NOT FULL, then check rear = = SIZE-1 && front ! = 0 if it is true then set rear = -1.

Step 4: Increment rear value by one (rear++) set queue[rear] = value and check front = = -1 If it is TRUE, then set front=0

Algorithm: Dequeue (deleting values from the queue)

Step 1: Check whether queue is EMPTY. (front = = -1 && rear = = -1)

Step 2: If it is EMPTY, then display “Queue is empty” and terminate the function

Step 3: If it is NOT EMPTY, then display queue[front] as deleted element and increment the front value by one(front++). Then check whether front = = SIZE, if it is TRUE, then set front = 0. Then check whether both front-1 and rear are equal (front-1 = = rear), if it is TRUE, then set both front and rear to -1 (front = rear = -1)

Code:

```
#include<iostream>
#define max 4
using namespace std;


class CircularQ
{
        public:
                int cq[max];
                int front,rear;
                CircularQ();
                void enqueue();
                void dequeue();
                void display();
};
CircularQ::CircularQ()
{
        front=rear=-1;
}
void CircularQ::enqueue()
{
        int num;
        //checking overflow
        if(front==(rear+1)%max)//use if(front==0 && rear==max-1)
        {
                cout<<"Queue is full \n";
                return;
        }
        else
        {
```

```
                cout<<"Enter the number ";
                cin>>num;
                //queue is empty
                if(front==-1)
                rear=front=0;
                else
                rear=(rear+1)%max;
        }
        cq[rear]=num;
        cout<<num<<"is inserted..";
}
void CircularQ::dequeue()
{
        int num;
        if(front==-1)
        cout<<"Queue is empty";
        else
        {
                num=cq[front];
                cout<<"deleted item is"<<num;
                if(front==rear)
                front=rear=-1;
                else
                front=(front+1)%max;
        }
}
void CircularQ::display()
{
```

```
		int i;
		if(front==-1)
		cout<<"Queue is empty";
		else
		{
			cout<<"\n Queue elements are \n";
			for(i=front;i<=rear;i++)
			cout<<cq[i]<<"\t";
		}
		if(front>rear)
		{
			for(i=front;i<max;i++)
			cout<<cq[i]<<"\t";
			for(i=0;i<=rear;i++)
			cout<<cq[i]<<"\t";
		}
	}
	int main()
	{
		CircularQ c;
		int choice;
		while(1)
		{
			cout<<"\n ---- Circular Queue Operation -----\n";
			cout<<"\n 1. Enqueue \n 2. Dequeue \n 3. Display \n 4. Exit \n";
			cout<<"Enter the choice ";
			cin>>choice;
			switch(choice)
```

```
            {
                case 1:
                    c.enqueue();
                    break;
                case 2:
                    c.dequeue();
                    break;
                case 3:
                    c.display();
                    break;
                case 4:
                    exit(0);
                default:
                    cout<<"wrong choice";
            }
        }
        return 0;
    }
```

Output:

```
1    . Enqueue
2    Dequeue
3    D T spley
4    Ex It
Ente n the r ho T r e 1
Enten the nutr be r 2 3
23 i s in s erted..
          C T nr u1a n gueue Ope nat hon
```

```
1    . Enqueue
2    Dequeue
3    D T spley
4    Ex It
Enten  the   r ho Tr e 3
```

```
Queue e1emen t c e ce
```

```
1    . Enqueue
2    Dequeue
3    D T spley
4    Ex It
Enten  the   r ho Tr e 2
de1eted it em i s 23
          C T nr u1a n gueue Ope nat hon
```

```
1    . Enqueue
2    Dequeue
3    D T spley
4    Ex It
Enten  the   r ho Tr e 3
```

```
1 . Enqueue
2.   Dequeue
3.   D T spley
4.   Ex It
Enten  the   r ho Tr e 4
```

```
P ror e s s ex ited aTten 12.86 s er and s with return va lue 0
```

Practical No. 6: Demonstrate application of queue

a) Aim: Write a program in c++ to implement priority queue

Algorithm: Priority queue

Step 1: Create a new node with DATA and PRIORITY

Step 2: Check if HEAD has low priority. If true go to step 3 and 4 and end else go to step 5.

Step3:PointNEXT>NEXT=HEAD

Step 4: Assign HEAD as a NEW

Step 5: Set TEMP to head of the list

Step6:CheckthatifTEMP>NEXT!=NULL TEMPNEXTPRIORITY>PRIORITY

Step 7: Assign TEMP as NEXT. End the loop

Algorithm: POP

Step 1: Set the head of the list to the next node in the list. HEAD = HEADNEXTStep

2: Free the node at the head of the list.

Step 3: End

Code:

```
#include<iostream>

using namespace std;

struct node

{

        int priority;

        int data;

        struct node *link;

};
```

```
class PriorityQueue
{
	private:
		node* front;
	public:
		PriorityQueue();
		void  insert();
		void deleteItem();
		void display();
};
PriorityQueue::PriorityQueue()
{
	front=NULL;
}
//This function will insert a data and its priority
void PriorityQueue::insert()
{
	node *tmp,*q;
	int added_item,item_priority;
	tmp=new struct node;
	cout<<"\nInput the item value to be added in the queue:";
	cin>>added_item;
	cout<<"\nEnter its priority:";
	cin>>item_priority;
	tmp->data = added_item;
	tmp->priority = item_priority;
	/*Queue is empty or item to be added has priority more than first item*/
```

```
        if(front == NULL || item_priority > front->priority)
        {
                tmp->link = front;
                front = tmp;
        }
        else
        {
                q = front;
                while(q->link != NULL && q->link->priority >= item_priority)
                q=q->link;
                tmp->link = q->link;
                q->link = tmp;
        }
}

void PriorityQueue::deleteItem()
{
        node* tmp;
        if(front == NULL)
        cout<<"\nQueue Underflow\n";
        else
        {
                tmp = front;
                cout<<"\nDeleted item is %d\n"<<tmp->data;
                front = front->link;
                delete(tmp);
        }
}/*End of del()*/
```

```
void PriorityQueue::display()
{
	node* ptr;
	ptr = front;
	if(front == NULL)
	cout<<"\nQueue is empty\n";
	else
	{
		cout<<"\nQueue is:\n";
		cout<<"\nPriority Item\n";
		while(ptr != NULL)
		{
			cout<<"|"<<ptr->priority<<" "<<ptr->data<<"|->";
			ptr = ptr->link;
		}
	}/*End of else */
}/*End of display() */
int main()
{
	int choice;
	PriorityQueue p;
	while(1)
	{
		cout<<"\n 1.Insert \n";
		cout<<"\n 2.Delete \n";
		cout<<"\n 3.Display \n";
		cout<<"\n 4.Quit \n";
		cout<<"\n Enter your choice ";
```

```
                cin>>choice;
                switch(choice)
                {
                        case 1:
                                p.insert();
                                break;
                        case 2:
                                p.deleteItem();
                                break;
                        case 3:
                                p.display();
                                break;
                        case 4:
                                exit(1);
                        default :
                                cout<<"\nWrong choice\n";
                }/*End of switch*/
        }/*End of while*/
}/*End of main()*/
```

Output:

```
 1.Insert

 2.Delete

 3.Display

 4.Quit

 Enter your choice 1

Input the item value to be added in the queue:23

Enter its priority:1

 1.Insert

 2.Delete

 3.Display

 4.Quit

 Enter your choice 3

Queue is:

Priority Item
|1 23|->
 1.Insert

 2.Delete

 3.Display

 4.Quit

 Enter your choice 2

Deleted item is %d
23
 1.Insert

 2.Delete

 3.Display

 4.Quit
 Ente n youn c ho be e 3

tro‹      i‹ d a+‹   i9.89  ‹ord ›‹ i‹ w  ‹vr valv i

 2 . De lete

 Ente n youn c ho be e 4
```

Practical No. 7: Implement all types of linked list

a) Aim: Write a program in c++ to implement Single Linked List.

Algorithm: Insert node at beginning

Step 1: Input DATA to be inserted
Step 2: Create a NewNode
Step 3: NewNode → DATA = DATA
Step 4: If (SATRT equal to NULL)
  (a) NewNode → Link = NULL
Step 5: Else
  (a) NewNode → Link = START
Step 6: START = NewNode
Step 7: Exit

Algorithm: Insert node at end

Step 1: Input DATA to be inserted

Step 2: Create a NewNode
Step 3: NewNode → DATA = DATA
Step 4: NewNode → Next = NULL
Step 5: If (SATRT equal to NULL)
  (a) START = NewNode
Step 6: Else
  (a) TEMP = START
  (b) While (TEMP → Next not equal to NULL)
    (i) TEMP = TEMP → Next
Step 7: TEMP → Next = NewNode
Step 8: Exit

Algorithm: Insert node at specific position

Step 1: Input DATA and POS to be inserted
Step 2: intialise TEMP = START; and j = 0
Step 3: Repeat the step 3
  while(k is less than POS)
  (a) TEMP = TEMP è Next
  (b) If (TEMP is equal to NULL)
    (i) Display "Node in the list less than the position"
    (ii) Exit (c) k = k + 1
Step 4: Create a New Node
Step 5: NewNode → DATA = DATA
Step 6: NewNode → Next = TEMP → Next

Step 7: TEMP → Next = NewNode
Step 8: Exit

Algorithm: Deleting a node

Step 1: Input the DATA to be deleted

Step 2: if ((START → DATA) is equal to DATA)
(a) TEMP = START
(b) START = START → Next
(c) Set free the node TEMP, which is deleted
(d) Exit
Step 3: HOLD = START
Step 4: while ((HOLD → Next → Next) not equal to NULL))
(a) if ((HOLD → NEXT → DATA) equal to DATA)
(i) TEMP = HOLD → Next
(ii) HOLD → Next = TEMP → Next
(iii) Set free the node TEMP, which is deleted
(iv) Exit
(b) HOLD = HOLD → Next
Step 5: if ((HOLD → next → DATA) == DATA)
(a) TEMP = HOLD → Next
(b) Set free the node TEMP, which is deleted
(c) HOLD → Next = NULL
(d) Exit
Step 6: Display "DATA not found"
Step 7: Exit

Algorithm: Searching node

Step 1: Input the DATA to be searched
Step 2: Initialize TEMP = START; POS =1;
Step 3: Repeat the step 4, 5 and 6 until (TEMP is equal to NULL)
Step 4: If (TEMP → DATA is equal to DATA)
(a) Display "The data is found at POS"
(b) Exit
Step 5: TEMP = TEMP → Next
Step 6: POS = POS+1
Step 7: If (TEMP is equal to NULL)
(a) Display "The data is not found in the list"
Step 8: Exit

Code:

```
#include<iostream>
using namespace std;
class SLinked_List
{
    //create a node
    struct node
    {
            int info;//data section
            struct node* link;//address section
    };
    struct node* head;
    public:
            SLinked_List()
            {
                    head = NULL;
            }
            void createList(int);
            void addAtBeg(int);
            void addAfterPos(int,int);
            void deleteData();
            void display();
};
void SLinked_List::createList(int data)//insert 10,insert 20 head is not NULL
{
    //create a node
    struct node *temp,*q;
    temp = new struct node;
    temp->info=data;
    temp->link = NULL;
    if(head == NULL)
    {
            head = temp;//temp as starting node
    }
    else
    {
            q = head;
            while(q->link!=NULL)
            q = q->link;
            q->link = temp;
    }
}
void SLinked_List::addAtBeg(int data)
```

```
{
      struct node*  temp;
      temp = new struct node;
      temp->info=data;
      temp->link=head;
      head=temp;
}
void SLinked_List::addAfterPos(int data,int pos)//15 at pos = 3 swipe to 4 one
{
      struct node* temp,*q;
      int i;
      q = head;
      for(i=0;i<pos-1;i++)
      {
            q = q->link;
            if(q==NULL)
            {
                  cout<<"\n there are less than "<<pos<<"elements";
                  return;
            }
      }
      temp = new struct node;
    temp->link=      q->link;
    temp->info=data;
      q->link=temp;

}
void SLinked_List::deleteData()
{
      struct node* temp,*q;
      int data;
      if(head == NULL)
      {
            cout<<"list is empty";
            return;
      }
      cout<<"\n enter the element for deletion";
      cin>>data;
      if(head->info==data)//if the data is in first node
      {
            temp=head;
            head = head->link;
            delete(temp);
```

```
                return;
        }
        //if the data is in between the list
        q=head;
        while(q->link != NULL)
        {
                if(q->link->info==data)
                {
                        temp=q->link;
                        q->link = temp->link;
                        delete(temp);
                }
                        q=q->link;
        }

    //if data is at last
    if(q->link->info==data)
    {
        temp=q->link;
        delete(temp);
        q->link = NULL;
        return;
    }
    }
    void SLinked_List::display()
    {
        struct node *q;
        if(head==NULL)
        {
                cout<<"\n List is empty";
                return;
        }
        q=head;
        cout<<"\n List of elements:";
        while(q!=NULL)
        {
                cout<<q->info<<" ";
                q=q->link;//passing to next address
        }
    }
    int main()
    {
        int choice,size,element,pos;
```

```
SLinked_List sl;
while(1)
{
	cout<<"1:Create  list\n";
	cout<<"2: Add element at first\n";
	cout<<"3: Add  after\n";
	cout<<"4: Delete\n";
	cout<<"5: Display\n";
	cout<<"6: Quit\n";
	cout<<"Enter choice:\n";
	cin>>choice;
	switch(choice)
	{
		case 1:
			cout<<"\nHow many nodes to create:";
			cin>>size;
			for(int i=0;i<size;i++)
			{
				cout<<"Enter the Element:";
				cin>>element;
				sl.createList(element);
			}
			break;
		case 2:
			cout<<"\n enter the element:";
			cin>>element;
			sl.addAtBeg(element);
			break;
		case 3:
			cout<<"\n enter the element: ";
			cin>>element;
			cout<<"enter the position where the elemenet to be 
insterted";
			cin>>pos;
			sl.addAfterPos(element,pos);
			break;
		case 4:
			sl.deleteData();
			break;
		case 5:
			sl.display();
			break;
		case 6:
```

```
                    exit(0);
            default:
                    cout<<"\nwrong choice";
        }
    }
}
```

Output:

```
1:Create list
2: Add element at first
3: Add after
4: Delete
5: Display
6: Quit
Enter choice:
1

How many nodes to create:3
Enter the Element:895
Enter the Element:246
Enter the Element:123
1:Create list
2: Add element at first
3: Add after
4: Delete
5: Display
6: Quit
Enter choice:
5

 List of elements: 895

246

123

1:Create list
2: Add element at first
3: Add after
4: Delete
5: Display
6: Quit
Enter choice:
2

 enter the element:0001
```

s : 9u I
E n te c h o ce :

24 6

s : 9u I
E n te c h o ce :

en te Ihe etc men I : 2
en te Ihe po I o tche re Ihe etc acne I Io be te te fi2

s : 9u I
E n te c h o ce :

b) **Aim: Write a program in c++ to implement Double Linked List.**

Algorithm: Insertion at beginning

Step 1: Input the DATA and POS

Step 2: Initialize TEMP = START; i = 0
Step 3: Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
Step 4: TEMP = TEMP → RPoint; i = i +1
Step 5: If (TEMP not equal to NULL) and (i equal to POS)

(a) Create a New Node
(b) NewNode → DATA = DATA
(c) NewNode → RPoint = TEMP → RPoint
(d) NewNode → LPoint = TEMP
(e) (TEMP → RPoint) → LPoint = NewNode
(f) TEMP → RPoint = New Node

Step 6: Else

(a) Display "Position NOT found"

Step 7: Exit

Algorithm: For deleting a node

Step 1: Input the POS

Step 2: Initialize TEMP = START; i = 0

Step 3: Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)

Step 4: TEMP = TEMP → RPoint; i = i +1

Step 5: If (TEMP not equal to NULL) and (i equal to POS)

(a) Create a New Node
(b) NewNode → DATA = DATA
(c) NewNode → RPoint = TEMP → RPoint
(d) NewNode → LPoint = TEMP
(e) (TEMP → RPoint) → LPoint = NewNode
(f) TEMP → RPoint = New Node

Step 6: Else

(a) Display "Position NOT found"

Step 7: Exit

Code:

```
#include<iostream>
using namespace std;
class dlinked_list {
struct node {
    int data;
    struct node* prev;
    struct node* next;
};
struct node* head;
int data;
public:
    dlinked_list();
    void insertAtFront();
    void insertAtEnd();
    void insertAtposition(int);
    void  deleteAtFront();
    void deleteAtend();
    void deleteAtpos(int);
    void display();//traverse
};
dlinked_list :: dlinked_list() {
    head=NULL;
}
void dlinked_list :: insertAtFront() {
    struct node* temp;
    cout<<"enter data into the node";
    cin>>data;
```

```
            temp=new struct node;
            temp->data=data;
            temp->prev=NULL;
        temp->next=NULL;
        if(head==NULL) {
            head=temp;
        } else {
            temp->next=head;
            head->prev=temp;
            head=temp;
        }

    }
    void dlinked_list :: insertAtEnd() {

        struct node* temp,*t;

        cout<<"enter data into the node";
        cin>>data;
        temp=new struct node;
        temp->data=data;
        temp->prev=NULL;
        temp->next=NULL;
        if(head==NULL) {
            head=temp;
        } else {
            t=head;
            while(t->next!=NULL) {
```

```
                t=t->next;
            }
            t->next=temp;
            temp->prev=t;
        }
}
void dlinked_list :: insertAtposition(int pos) {
        struct node* temp,*pr,*aft;
        int index=0;
        cout<<"enter data into the node";
        cin>>data;
        temp=new struct node;
        temp->data=data;
        temp->prev=NULL;
        temp->next=NULL;
        if(head==NULL) { //if it is empty
            head=temp;
        } else {
            pr=aft=head;
            if(pos==0) {
                temp->next=head;
                head=temp;

            } else {
                while(index<pos) {
                    index++;
                    pr=aft;
                    aft=aft->next;
```

```
                }
                pr->next=temp;
                temp->prev=pr;
                temp->next=aft;
                aft->prev=temp;
            }
        }
}
void dlinked_list :: deleteAtFront() {
        struct node* t;
        t=head;
        head=head->next;
        head->prev=NULL;
        cout<<t->data<<"deleted successfully";
        delete(t);
}
void dlinked_list :: deleteAtend() {
        struct node *pr,*aft;
        pr=aft=head;
        if(head==NULL)
            cout<<"list is empty";
        else {

            while(aft->next!=NULL) {
                pr=aft;
                aft=aft->next;
```

```
                }
                pr->next=NULL;
                cout<<aft->data<<"deleted successfully";
                delete(aft);
        }
}
void dlinked_list :: deleteAtpos(int pos){
        struct node *pr,*aft;
        pr=aft=head;
        int count=0;
        if(head==NULL)
                cout<<"list is empty";

        else
        {
                if(pos==0){
                        deleteAtFront();}
                else{


                                while(count<pos) {
                                count++;
                                pr=aft;
                                aft=aft->next;

                }

                pr->next=aft->next;
```

```
            aft->next->prev=pr;
            cout<<aft->data<<"is deleted";
            delete(aft);
}
        }
}
void dlinked_list :: display() {
        struct node *t;
        if(head==NULL) {
            cout<<"list is empty"<<endl;
        } else {
            cout<<"the elements in the list are"<<endl;
            t=head;
            while(t!=NULL) {
                cout<<t->data<<"<=>";
                t=t->next;//incrementing the node
            }
        }
}
int main() {
        int choice,size;
        int element,pos;
        dlinked_list d1;
        while(1) {

            cout<<"1: Add Element at First\n";
            cout<<"2: Add At end\n";
            cout<<"3: add at position\n";
```

```
cout<<"4: delete Element at First\n";
cout<<"5: delete At end\n";
cout<<"6: delete element at position\n";
cout<<"7: Display\n";
cout<<"8: Quit\n";
cout<<"1: Enter Choice\n";
cin>>choice;
switch(choice) {
    case 1:

        d1.insertAtFront();
        break;

    case 2:

        d1.insertAtEnd();
        break;

    case 3:
        cout<<"enter the position"<<endl;
        cin>>pos;
        d1.insertAtposition(pos);
        break;

    case 4:
        d1.deleteAtFront();
        break;
    case 5:
```

```
                d1.deleteAtend();
                break;
            case 6:
                cout<<"enter the position"<<endl;
                cin>>pos;
                d1.deleteAtpos(pos);
                break;
            case 7:
                d1.display();
                break;

            case 8:
                exit(0);

            default:
                cout<<"Wrong Choice";
        }
    }

}
```

Output:

```
1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
1
enter data into the node1
1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
2
enter data into the node3
1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
3
enter the position
1
enter data into the node2
1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
```

```
enter data into the node100
1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
7
the elements in the list are
100<=>2<=>3<=> 1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
6
enter the position
1
2is deleted1: Add Element at First
2: Add At end
3: add at position
4: delete Element at First
5: delete At end
6: delete element at position
7: Display
8: Quit
1: Enter Choice
7
the elements in the list are
100<=>3<=> 1: Add Element at First
```

c) **Aim: Write a program in c++ to implement Circular Linked List.**

Algorithm:

Step 1: IF PTR = NULL

Write OVERFLOW
Go to Step 11
[END OF IF]

Step 2**:** SET NEW_NODE = PTR

Step 3**:** SET PTR = PTR -> NEXT

Step 4**:** SET NEW_NODE -> DATA = VAL

Step 5: SET TEMP = HEAD

Step 6: Repeat Step 8 while TEMP -> NEXT != HEAD

Step 7: SET TEMP = TEMP ->

NEXT [END OF LOOP]

Step 8: SET NEW_NODE -> NEXT =

HEAD **Step 9:** SET TEMP → NEXT =

NEW_NODE **Step 10:** SET HEAD =

NEW_NODE

Step 11: EXIT

Code:

```
#include<iostream>
using namespace std;
struct node
{
        int data;
        struct node* link;
};
class CList
{
        int data;
        struct node* head;
        public:
                CList();
                void insertFront();
                void insertEnd();
                void insertPos(int);
                void display();
                void deleteFront();
                void deleteEnd();
```

```
        void deletePos(int);
};
CList::CList()
{
    head=NULL;
}
void CList::insertFront()
{
    struct node *temp, *t;
    temp = new struct node;
    cout<<"enter element"<<endl;
    cin>>data;
    temp->data=data;
    if(head == NULL)
    {
        head=temp;
        temp->link=head;
    }
    else
    {
```

```
            temp->link=head;
            t = head;
            while(t->link != head)
            {
                t = t->link;
            }
            t->link= temp;
            head = temp;
        }
        cout<<"inserted successfully"<<endl;
    }
    void CList::insertEnd()
    {
        struct node* temp,*t;
        int data;
        cout<<"enter data to insert";
        cin>>data;
        temp=new struct node;
        temp->data=data;
        temp->link=NULL;
```

```
if(head==NULL)//if there no element in the list
{
        head=temp;
        temp->link=head;
}
else
{
        t=head;
        if(t->link==head)//list containing one node
        {
                t->link=temp;
                temp->link=t;
        }
        else
        {
                while(t->link!=head)
                {
                        t=t->link;
                }
                t->link=temp;
```

```
                    temp->link=head;
                }
        }
        cout<<"node inserted successfully";
}
void CList::insertPos(int pos)
{
        struct node *temp, *t;
        int i;
        temp = new struct node;
        cout<<"enter element"<<endl;
        cin>>data;
        temp->data = data;
        if(head == NULL)
        {
                cout<<"List is empty";
        }
        else
        {
```

```
                t = head;
                for(i=0; i<=pos-3; i++)
                {
                        t = t->link;
                }
                temp->link = t->link;
                t->link = temp;
                cout<<"inserted sucessfully"<<endl;
        }
}
void CList::deleteFront()
{
        struct node *temp,*t;
        t=head;
        data=head->data;
        while(t->link!=head)
        {
                t=t->link;
        }
        temp=head;
```

```
		head=head->link;

		t->link=head;

		delete(temp);

		cout<<data<<" deleted sucessfully"<<endl;

	}

	void CList::deleteEnd()

	{

		struct node *t,*temp;//here t is current node and temp is previous node

		t=head;

		while(t->link!=head)

		{

			temp=t;

			t=t->link;

		}

		temp->link=head;

		data=t->data;

		delete(t);

		cout<<data<<" deleted successfully"<<endl;

	}

	void CList::deletePos(int pos)
```

```
{
    struct node *temp,*t;
    t=head;
    int count=0;
    if(head==NULL)
    cout<<"list is empty";
    else
    {
        if(pos==0)
        {
            deleteFront();
        }
        else
        {
            while(count<pos) {
                count++;
                temp=t;
                t=t->link;
        }
        temp->link=t->link;
```

```
                cout<<t->data<<"is deleted";
                delete(t);
            }
    }
}
void CList::display()
{
            struct node *t;
            if(head==NULL)
            {
                cout<<"list is empty\n";
            }
            else
            {
                t=head;
                if(t->link==head){//if there is only one node
                    cout<<t->data<<"->";
                }
                else
                {
```

```
                    cout<<t->data<<"->";
                    t=t->link;
                    while(t!=head)
                    {
                        cout<<t->data<<"->";
                        t=t->link;
                    }
                }
            }
        }
        int main()
        {
            CList c;
            int choice,pos;
            while(1)
            {
                cout<<"\nCList operations:\n";
                cout<<"\n 1.insertAtFront\n 2.insertAtEnd\n 3.insertAtpos\n
                4.deleteAtFront\n 5.deleteAtEnd\n 6.deleteAtPos\n 7.display\n 8.exit\n";
                cout<<"\n enter choice ";
```

```
cin>>choice;
switch(choice)
{
    case 1:
        c.insertFront();
        break;
    case 2:
        c.insertEnd();
        break;
    case 3:
        cout<<"enter position"<<endl;
        cin>>pos;
        c.insertPos(pos);
        break;
    case 4:
        c.deleteFront();
        break;
    case 5:
        c.deleteEnd();
        break;
```

```
                        case 6:
                            cout<<"enter the position"<<endl;
                            cin>>pos;
                            c.deletePos(pos);
                            break;
                        case 7:
                            c.display();
                            break;
                        case 8:
                            exit(0);
                        default:cout<<"wrong choice";
                }
        }
        return 0;
}
```

Output:

```
T . In se ntAt F nont
2.  In  sentAt End
3. In  se ntAtpo s
4. de1eteAt F nont
5. de1eteAt End
6. deleteAt Pos
7.di splay

enter choice 1
ente n e1ement
10
i n se n te d s u c c e s s l u lly

T . In se ntAt F nont
2.  In  sentAt End
3. In  se ntAtpo s
4. de1eteAt F nont
5. de1eteAt End
6. deleteAt Pos
7.di splay

enter choice 2
enten  data   to    In se nt 50
node   In  sented  suse essTu11\

T . In se ntAt F nont
2.  In  sentAt End
3. In  se ntAtpo s
4. de1eteAt F nont
5. de1eteAt End
6. deleteAt Pos
7.di splay
```

```
ente c e lesen t
1.insertAtFront
CList operations:
2. In se nt At End
4. de leteAt Fcont
5. de1eteAt End
6. deleteAt Po s
7. di sp lay
8. exIt

enter choice 4
10 deleted sucessfully

[redacted]
2. In se nt At End
[redacted]
4. de leteAt Fcont
5. de1eteAt End
6. deleteAt Po s
7. di sp lay
8. exIt

enter choice 5
50 deleted successfully

[redacted]
2. In se nt At End
[redacted]
4. de leteAt Fcont
5. de1eteAt End
6. deleteAt Po s
7. di sp lay
8. exIt

ente n cho Ice 6
49 i s de leted
```

```
1 . InsentAt Fnont
2. In se ntAtEnd

4. de1eteAt Fnont
 5. de1eteAt End
6. de1ct eAt Pos
7.di splay

T . In se ntAt F nont
2. In  sentAt End
3. In  se ntAtpo s
4. de1eteAt F nont
5. de1eteAt End
6. deleteAt Pos
7.di splay

enten   cho ice    8


P no c e s s exited aPte n 9 5.76 second s uTth netunn va1ue 0
```

Practical No. 8: To demonstrate application of linked list

a) **Write a program in c++ to implement polynomial addition**

Algorithm:

Step 1: loop around all values of linked list and follow step 2& 3.

Step 2: if the value of a node's exponent. is greater copy this node to result node and head towards the next node.

Step 3: if the values of both node's exponent is same add the coefficients and then copy the added value with node to the result.

Step 4: Print the resultant node.

Code:

```
#include<iostream>
using namespace std;
class PolyAdd
{
        private: //creation of node
        struct polynode
        {
                float coeff;
                int exp;
                polynode *link;
        };
        struct polynode* head;
        public:
                PolyAdd();
                void createpoly(float c, int e);
                void displaypoly();
                void addpoly(PolyAdd &p1,PolyAdd &p2);
                //~PolyAdd();
```

```
};
PolyAdd :: PolyAdd()
{
	head=NULL;
}
void PolyAdd :: createpoly(float c, int e)
{
	polynode *temp,*ptr;
	temp=new struct polynode;
	temp->coeff=c;
	temp->exp=e;
	temp->link=NULL;
	if(head==NULL || e>head->exp)
	{
		temp->link=head;
		head=temp;
	}
	else
	{
		ptr=head;
		while(ptr->link!=NULL && ptr->link->exp>e)
		{
			ptr=ptr->link;
		}
		ptr->link=temp;
		ptr=ptr->link;
	}
}
```

```
void PolyAdd ::addpoly(PolyAdd &p1, PolyAdd &p2)
{
	struct polynode* result;
	if(p1.head==NULL && p2.head==NULL)
	{
		return;
	}
	polynode *temp1,*temp2;
	temp1=p1.head;
	temp2=p2.head;
	while(temp1!=NULL && temp2!=NULL)
	{
		if(head==NULL)
		{
			head=new polynode;
			result=head;
		}
		else
		{
			result->link=new polynode;
			result=result->link;
		}
		if(temp1->exp < temp2->exp)
		{
			result->coeff=temp2->coeff;
			result->coeff=temp2->exp;
			temp2=temp2->link;
		}
```

```
        else if(temp1->exp > temp2->exp)
        {
                result->coeff=temp1->coeff;
                result->exp=temp1->exp;
                temp1=temp1->link;
        }
        else if(temp1->exp==temp2->exp)
        {
                result->coeff=(temp1->coeff)+(temp2->coeff);
                result->exp=temp1->exp;
                temp1=temp1->link;
                temp2=temp2->link;
        }
}
while(temp1!=NULL)
{
        if(head==NULL)
        {
                head=new polynode;
                result=head;
        }
        else
        {
                result->link=new polynode;
                result=result->link;
        }
        result->coeff=temp1->coeff;
        result->exp=temp1->exp;
```

```
                temp1=temp1->link;
        }
        while(temp2!=NULL)
        {
                if(head==NULL)
                {
                        head=new polynode;
                        result=head;
                }
                else
                {
                        result->link=new polynode;
                        result=result->link;
                }
                result->coeff=temp2->coeff;
                result->exp=temp2->exp;
                temp2=temp2->link;
        }
        result->link=NULL;
}
void PolyAdd :: displaypoly()
{
        polynode *q;
        q=head;
        while(q!=NULL)
        {
                if(q->exp!=0)
                {
```

```
                cout<<q->coeff<<"x^"<<q->exp;
                cout<<"+";
            }
            else
            {
                cout<<q->coeff;
            }
            q=q->link;
        }
}
int main()
{
        PolyAdd p1;
        p1.createpoly(3,3);
        p1.createpoly(5,4);
        p1.createpoly(5,0);
        cout<<"the first polynomial is: "<<endl;
        p1.displaypoly();
        PolyAdd p2;
        p2.createpoly(4,4);
        p2.createpoly(2,3);
        p2.createpoly(8,0);
        cout<<"the second polynomial is: "<<endl;
        p2.displaypoly();
        PolyAdd p3;
        p3.addpoly(p1,p2);
        cout<<"\nThe resultant polynomial is: "<<endl;
        p3.displaypoly();
```

```
        return 0;
}
```

Output:

```
the first polynomial is:
5x^4+3x^3+5the second polynomial is:
4x^4+2x^3+8
The resultant polynomial is:
9x^4+5x^3+13
--------------------------------
Process exited after 2.08 seconds with return value 0
Press any key to continue . . .
```

b) Write a program in c++ to implement Sparse Matrix

Code:

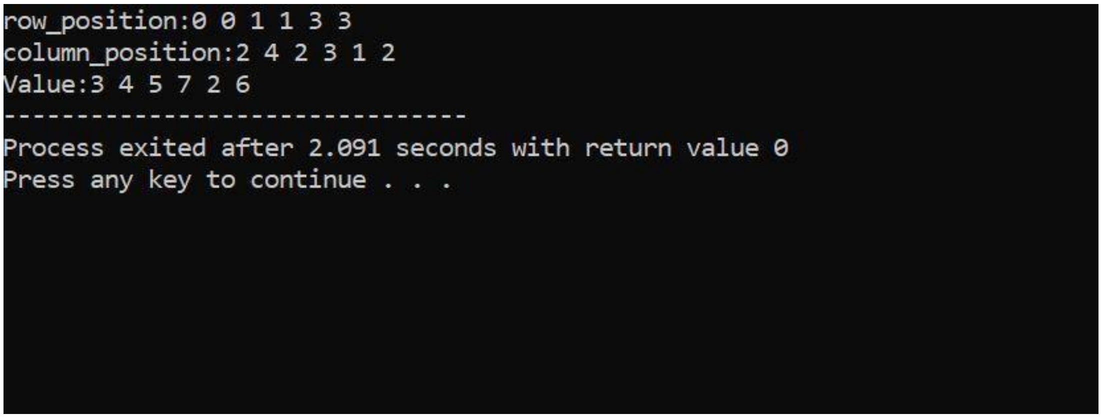
```
// C++ program for sparse matrix representation.
// Using Link list
#include<iostream>
using namespace std;
// Node class to represent link list
class Node
{
        public:
                int row;
                int col;
                int data;
        Node *next;
};
```

```
// Function to create new node
void create_new_node(Node **p, int row_index,int col_index, int x)
{
	Node *temp = *p;
	Node *r;
	// If link list is empty then
	// create first node and assign value.
	if (temp == NULL)
	{
		temp = new Node();
		temp->row = row_index;
		temp->col = col_index;
		temp->data = x;
		temp->next = NULL;
		*p = temp;
	}
	// If link list is already created
	// then append newly created node
	else
	{
		while (temp->next != NULL)
		temp = temp->next;
		r = new Node();
		r->row = row_index;
		r->col = col_index;
		r->data = x;
		r->next = NULL;
		temp->next = r;
```

```
        }
}
// Function prints contents of linked list
// starting from start
void printList(Node *start)
{
        Node *ptr = start;
        cout << "row_position:";
        while (ptr != NULL)
        {
                cout << ptr->row << " ";
                ptr = ptr->next;
        }
        cout << endl;
        cout << "column_position:";
        ptr = start;
        while (ptr != NULL)
        {
                cout << ptr->col << " ";
                ptr = ptr->next;
        }
        cout << endl;
        cout << "Value:";
        ptr = start;
        while (ptr != NULL)
        {
                cout << ptr->data << " ";
                ptr = ptr->next;}}

int main()
{
```

```
int sparseMatrix[4][5] = { { 0 , 0 , 3 , 0 , 4 },
                           { 0 , 0 , 5 , 7 , 0 },
                           { 0 , 0 , 0 , 0 , 0 },
                           { 0 , 2 , 6 , 0 , 0 } };

// Creating head/first node of list as NULL
Node *first = NULL;
for(int i = 0; i < 4; i++){
        for(int j = 0; j < 5; j++){
                // Pass only those values which
                // are non - zero
                if (sparseMatrix[i][j] != 0)
                create_new_node(&first, i,
                j,sparseMatrix[i][j]);}}

printList(first);
return 0;
}
```

Output:



Practical No. 9: **Create and perform various operations on BST**

a) Inserting node in BST
b) Deleting the node from BST
c) To find height of Tree
d) To perform Inorder
e) To perform Preorder
f) To perform Postorder
g) To find Maximum value of tree

Algorithm: Insert operation

Step 1: Input the DATA to be pushed and ROOT node of the tree.

Step 2: NEWNODE = Create a New Node.
Step 3: If (ROOT == NULL)
(a) ROOT=NEW NODE
Step 4: Else If (DATA < ROOT → Info)
(a) ROOT = ROOT → Lchild
(b) GoTo Step 4
Step 5: Else If (DATA > ROOT → Info)
(a) ROOT = ROOT → Rchild
(b) GoTo Step 4THE TREES 261
Step 6: If (DATA < ROOT → Info)
(a) ROOT → LChild = NEWNODE
Step 7: Else If (DATA > ROOT → Info)
(a) ROOT → RChild = NEWNODE
Step 8: Else
(a) Display ("DUPLICATE NODE")
(b) EXIT

Step 9: NEW NODE → Info = DATA
Step 10: NEW NODE → LChild = NULL
Step 11: NEW NODE → RChild = NULL
Step 12: EXIT

Algorithm: delete operation

Step 1: Find the location NODE of the DATA to be deleted.
Step 2: If (NODE = NULL)

(a) Display "DATA is not in tree"

(b) Exit
Step 3: If(NODE → Lchild = NULL)
(a) LOC = NODE
(b) NODE = NODE → RChild
Step 4: If(NODE → RChild= =NULL)

(a) LOC = NODE
(b) NODE = NODE → LChild

Step 5: If((NODE → Lchild not equal to NULL) && (NODE → Rchild not equal to NULL))

(a) LOC = NODE → RChild

Step 6: While(LOC → Lchild not equal to NULL)

(a) LOC = LOC → Lchild

Step 7: LOC → Lchild = NODE → Lchild

Step 8: LOC → RChild= NODE → RChild

Step 9: Exit

Algorithm: Preorder traversal

Step 1: Visit the root node

Step 2: Traverse the left sub tree in preorder

Step 3: Traverse the right sub tree in preorder

Algorithm: Postorder traversal

Step 1: Traverse the left sub tree in post order
Step 2: Traverse the right sub tree in post order
Step 3: Visit the root node

Algorithm: Postorder traversal

Step 1: Traverse the left sub tree in order
Step 2: Visit the root node
Step 3: Traverse the right sub tree in order

Code:

```
#include<iostream>
using namespace std;
#define SPACE 10
//creating a tree node
class treenode
{
        public:
        int data;
        treenode* left;
        treenode* right;
        treenode()
        {
                data=0;
                left=NULL;
```

```
            right=NULL;

        }

};
class BinarySearchTree
{
        public:
        treenode* root;//node
        BinarySearchTree()
        {
                root=NULL;
        }
        void insertNode(treenode* newnode)
        {

                if(root==NULL)//there is no node in the tree
                {
                        root=newnode;
                        cout<<"node is inserted at root level"<<endl;
                }
                else
                {
                        treenode* temp=root;//to traverse the tree
                        while(temp!=NULL)
                        {
                                if(newnode->data==temp->data)
```

```
				{
					cout<<"duplicacy is not allowed"<<endl;
					return;
				}
				else if((newnode->data<temp->data)&&(temp
				>left==NULL))
				{
					temp->left=newnode;
					cout<<"the node is inserted at left"<<endl;
					break;
			}
			else if(newnode->data<temp->data)
			{
				temp=temp->left;
			}
			else if((newnode->data>temp->data)&&(temp->right==NULL))
			{
				temp->right=newnode;
				cout<<"the node is inserted at right"<<endl;
				break;
			}
			else
			{
				temp=temp->right;
			}
			}
		}

	}

	treenode* deleteNode(treenode* r,int v)
	{
		bool found=false;
		if(root==NULL)
		{
			cout<<"tree is empty"<<endl;
			return NULL;
		}
		treenode* curr;
		treenode* parent;
		curr=root;
		while(curr!=NULL)
		{
			if(curr->data==v)
			{
				found=true;
```

```
				break;
			}
			else
			{
				parent=curr;
				if(v>curr->data)
				{
					curr=curr->right;
				}
				else
				{
					curr=curr->left;
				}
			}
		}
		if(!found)
		{
			cout<<"the value is not present"<<endl;
			return NULL;
		}

		if(r==NULL)
		{
			return NULL;
		}
		else if(v<r->data)
		{
			r->left=deleteNode(r->left,v);
		}
		else if(v>r->data)
		{
			r->right=deleteNode(r->right,v);
		}
		else
		{
			if(r->left==NULL)
			{
				treenode* temp=r->right;
				delete r;
				return temp;
			}
			else if(r->right==NULL)
			{
				treenode* temp=r->left;
				delete r;
				return temp;
```

```
                }
                else
                {
                        treenode* temp=minValueNode(r->right);
                        r->data=temp->data;
                        r->right=deleteNode(r->right,temp->data);
                }
        }
}
treenode* minValueNode(treenode* node)
{
        treenode* curr=node;
        while(curr->left!=NULL)
        {
                curr=curr->left;
        }
        return curr;
}
void display(treenode* r, int space)
{
        if(r==NULL)
        {
                return;
        }
        space+=SPACE;
        display(r->right,space);
        cout<<endl;
        for(int i=SPACE;i<space;i++)
        {
                cout<<" ";
        }
        cout<<r->data<<"\n";
        display(r->left,space);
}
int height(treenode* r)
{
        if(r==NULL)
        {
                return -1;
        }
        else
        {
                int lheight=height(r->left);
                int rheight=height(r->right);
                if(lheight>rheight)
                {
```

```
                return (lheight+1);
            }
            else
            {
                return (rheight+1);
            }
        }
}
void printPreorder(treenode* r)
{
        if(r==NULL)
        {
            return;
        }
        cout<<r->data<<" ";//read node
        printInorder(r->left);//read left
        printInorder(r->right);//read right
}
void printInorder(treenode* r)
{
        if(r==NULL)
        {
            return;
        }
        printPreorder(r->left);//read left
        cout<<r->data<<" ";//read node
        printPreorder(r->right);//read right
}
void printPostorder(treenode* r)
{
        if(r==NULL)
        {
            return;
        }
        printPostorder(r->left);//read left
        printPostorder(r->right);//read right
        cout<<r->data<<" ";//read node
}
int findMax(treenode* root)
{
        if(root==NULL)
        {
            return 0;
        }
        int res=root->data;
        int lres= findMax(root->left);
```

```
		int rres=findMax(root->right);
		if(lres>res)
		{
			res=lres;
		}
		if(rres>res)
		{
			res=rres;
		}
		return res;
	}
};
int main()
{
	BinarySearchTree bst;
	int choice,val;
	while(1)
	{
		cout<<"select bst opertaion:"<<endl;
		cout<<"1.insert node\n";
		cout<<"2.delete node\n";
		cout<<"3.display\n";
		cout<<"4.display height of tree\n";
		cout<<"5. preorder \n";
		cout<<"6. Inorder \n";
		cout<<"7. Postorder \n";
		cout<<"8. Find max value \n";
		cout<<"9.exit\n";
		cin>>choice;
		treenode t;
		treenode* newnode=new treenode();
		switch(choice)
		{
			case 1:
				cout<<"enter value to be inserted in the node:"<<endl;
				cin>>val;
				newnode->data=val;
				bst.insertNode(newnode);
				break;
			case 2:
				cout<<"enter value to delete"<<endl;
				cin>>val;
				bst.deleteNode(bst.root,val);
				break;
			case 3:
				bst.display(bst.root,5);
```

```
                case 4:
                        int h;
                        cout<<"height is:"<<endl;
                        h=bst.height(bst.root);
                        cout<<h<<endl;
                        break;
                case 5:
                        bst.printPreorder(bst.root);
                        break;
                case 6:
                        bst.printInorder(bst.root);
                        break;
                case 7:
                        bst.printPostorder(bst.root);
                        break;
                case 8:
                        int max;
                        max=bst.findMax(bst.root);
                        cout<<"max value is:"<<max<<endl;
                        break;
                case 9:
                        exit(0);
                default:
                        cout<<"wrong choice"<<endl;
            }
        }
    }
```

Output:

```
select bst opentaion :
1 . i nse at node
2 . dei et e node
3 . di spt ay
4. d i s pJ ay  her ght   o    I see
5 . pceord er
7    Posl or den
8. Find rrax value
9. exit
1
enten va I ue to be i       nse at ed        he  node :

node i s i nse at ed at        root  l  e\ e1
select bst opentaion :
1 . i nse at node
2 . dei et e node
3 . di spt ay
4. d i s pJ ay  her ght   o    I see
5 . pceord er
7    Posl or den
8. Find rrax value
9. exit

enten   va T ue  to   be  i nseat ed      I he node :
4
I he node i s i nse at ed at  I  e I-I
select bst opentaion :
1 . i nse at node
2 . dei et e node
3 . di spt ay
4. d i s pJ ay  her ght   o    I see
5 . pceord er
7    Posl or den
8. Find rrax value
9. exit

enten   va T ue  to   be  i nseat ed      I he node :
B
```

```
enter value to be inserted in the node:
60
the node is inserted at right
select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
1
enter value to be inserted in the node:
35
the node is inserted at left
select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
1
enter value to be inserted in the node:
56
the node is inserted at left
```

```
select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
2
enter value to delete
60
select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
3

     67

                                        56

                    45

                                        35
height is:
2
```

```
select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
5
67 35 45 56 select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
6
45 35 56 67 select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
7
35 56 45 67 select bst opertaion:
```

```
35 56 45 67 select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
8
max value is:67
select bst opertaion:
1.insert node
2.delete node
3.display
4.display height of tree
5. preorder
6. Inorder
7. Postorder
8. Find max value
9.exit
```

Practical No. 10: Implementing Heap with different operations performed

a) To perform insertion operation
b) To create Heap using Heapify method
c) To perform Heap sort
d) To delete the value in heap

Algorithm: Insertion

Step 1: Input n elements in the heap H.

Step 2: Add new node by incrementing the size of the heap H: n = n + 1 and LOC = n

Step 3: Repeat step 4 to 7 while (LOC < 1)

Step 4: PAR = LOC/2

Step 5: If (data <= HA[PAR])

(a) HA[LOC] = data

(b) Exit

Step 6: HA[LOC] = HA[PAR]

Step 7: LOC = PAR

Step 8: HA[1] = data

Step 9: Exit

Algorithm: Heap sort

Step 1: Input n elements in the heap H.

Step 2: Add new node by incrementing the size of the heap H: n = n + 1 and LOC = n

Step 3: Repeat step 4 to 7 while (LOC < 1)

Step 4: PAR = LOC/2

Step 5: If (data <= HA[PAR])

(a) HA[LOC] = data

(b) Exit

Step 6: HA[LOC] = HA[PAR]

Step 7: LOC = PAR

Step 8: HA[1] = data

Step 9: Exit

Algorithm: Delete

Step 1: Input n elements in the heap H

Step 2: Data = HA[1]; last = HA[n] and n = n – 1

Step 3: LOC = 1, left = 2 and right = 3

Step 4: Repeat the steps 5, 6 and 7 while (right <= n)

Step 5: If (last >= HA[left]) and (last >= HA[right])

(a) HA[LOC] = last

(b) Exit

Step 6: If (HA[right] <= HA[left])

(i) HA[LOC] = HA[left]

(ii) LOC = left

(b) Else

(i) HA[LOC] = HA[right]

(ii) LOC = right

Step 7: left = 2 × LOC; right = left +1

Step 8: If (left = n) and (last < HA[left])

(a) LOC = left

Step 9: HA[LOC] = last

Step 10: Exit

Code:

```
#include<iostream>
using namespace std;
#define height 10
int arr[20],n;
//Function to insert an element to the heap
```

```
void insert(int num,int loc)//35 4
{
	int par;
	while(loc>0)//0
	{
		par = (loc-1)/2;//1st element,0th
		if (num<=arr[par])//[0]=15,[1]=35 [4]=20
		{
			arr[loc]=num;
			return;
		}
		arr[loc]=arr[par];//
		loc=par;//recursive,loc=4,loc=1,loc=0
	}/*End of while*/
	arr[0]=num;
}/*End of insert()*/
//This function to create a heap
void create_heap()
{
	int i;
	for(i=0;i<n;i++)
	//maxHeapify( arr, n,largest);
	insert(arr[i],i);
}/*End of create_heap()*/
//Function to display the elements in the array
void display()
{
	int i;
```

```
        for(i=0;i<n;i++)
        cout<<arr[i]<<endl;
        cout<<" ";
}/*End of display()*/
void maxHeapify(int arr[],int n, int i)
{
        int largest = i;//i=3
        int l=2*i;//6,
        int r=(2*i)+1;//7
        //comparing the root with its left and right child
        while(l<= n && arr[l]>arr[largest])
        {
                largest=l;
        }
        while(r<=n && arr[r]>arr[largest])
        {
                largest=r;
        }
        if(largest!=i)
        {
                int temp=arr[i];
                arr[i]=arr[largest];
                arr[largest]=temp;
                maxHeapify( arr, n,largest);
        }
}
void build(int a[],int n)//create heap
{
```

```
        int i;
        for(i=n/2;i>=0;i--)
        maxHeapify(a,n,i);
}
void del_root(int last)
{
        int left,right,i,temp;
        i=0; /*Since every time we have to replace root with last*/
        /*Exchange last element with the root */
        temp=arr[i];
        arr[i]=arr[last];
        arr[last]=temp;
        left=2*i+1; /*left child of root*/
        right=2*i+2;/*right child of root*/
        while( right < last)
        {
                if ( arr[i]>=arr[left] && arr[i]>=arr[right] )
                return;
                if ( arr[right]<=arr[left] )
                {
                        temp=arr[i];
                        arr[i]=arr[left];
                        arr[left]=temp;
                        i=left;
                }
                else
                {
                        temp=arr[i];
```

```
                        arr[i]=arr[right];
                        arr[right]=temp;
                        i=right;
                    }
                left=2*i+1;
                right=2*i+2;
                }/*End of while*/
                if(left==last-1 && arr[i] < arr[left])
                //if (left==last–1 && arr[i] < arr[left] )/*right==last*/
                {
                    temp=arr[i];
                    arr[i]=arr[left];
                    arr[left]=temp;
                }
            }/*End of del_root*/
            void deleteRoot(int arr[],int n)
            {
                int lastElement = arr[n-1];
                arr[0]=lastElement;
                n=n-1;
                maxHeapify(arr,n-2,0);
            }
            //Function to sort an element in the heap
            void heap_sort()
            {
                int last;
                for(last = n-1; last>=0;last--)
                del_root(last);
```

```
}
int main()
{
	int i;
	cout<<"enter number of elements:";
	cin>>n;
	for(i=0;i<n;i++)
	{
		cout<<"enter elements:";
		cin>>arr[i];
	}
	cout<<"\nEntered list is :\n";
	display();
	//create_heap();
	//maxHeapify(arr,n,i);
	build(arr,n);
	cout<<"\nHeap is :\n";
	//del_root(n-1);
	//display();
	display();
	heap_sort();
	//deleteRoot(arr,n);
	cout<<"\nSorted Heap is :\n";
	display();
	return 0;
}
```

Output:

```
ent er e?eve nt s : 45

Entered list is :
34

 2

45

 eap

45
3

12

Sorted Heap zs :
1

3
4
p
```

Practical No. 11: Create a graph storage structure

a) Aim: Write a program in c++ to implement adjancency matrix

Algorithm: Create a graph

Step 1: Input the total number of vertices in the graph, say n

Step 2: Allocate the memory dynamically for the vertices to store in list array.

Step 3: Input the first vertex and the vertices through which it has edges by linking the node from lost array through nodes.

Step 4: Repeat the process by incrementing the list array to add other vertices and edges.

Step 5: Exit

Algorithm: Searching and deleting from a graph

Step 1: Input an edge to be searched.

Step 2: Search for an initial vertex of edge in list arrays by incrementing the array index.

Step 3: Once it is found, search through the linked list for the terminal vertex of the edge.

Step 4: If found display “the edge is present in the graph”.

Step 5: Then delete the node where the terminal vertex is found and rearrange the linked list.

Step 6: Exit

Code:

```
#include<iostream>
using namespace std;
#define MAX 20
class Graph
{
    int adj[MAX][MAX];
    int node;
    int edge;
```

```
    public:
        Graph()
        {
            int i,j;
            for(i=0;i<MAX;i++)
            for(j=0;j<MAX;j++)
            adj[i][j]=0;
            node=0;
            edge=0;
        }
        void createGraph();//bidirectional
        void createDGraph();
        void display();

};
void Graph::createGraph()
{
    int origin,dest,i;
    cout<<"enter the no of nodes"<<endl;
    cin>>node;
    cout<<"neter the no of edges";
    cin>>edge;
    for(i=1;i<=edge;i++)
    {
        cout<<"\enter edge"<<i<<endl;
        cout<<"enter origin"<<endl;
        cin>>origin;
        cout<<"enter dest";
```

```
                cin>>dest;
                adj[origin][dest]=1;
                adj[dest][origin]=1;
                cout<<endl;
        }
}
void Graph::createDGraph()
{
        int origin,dest,i;
        cout<<"enter the no of nodes"<<endl;
        cin>>node;
        cout<<"neter the no of edges";
        cin>>edge;
        for(i=1;i<=edge;i++)
        {
                cout<<"\enter edge"<<i<<endl;
                cout<<"enter origin"<<endl;
                cin>>origin;
                cout<<"enter dest";
                cin>>dest;
                adj[origin][dest]=1;
                adj[dest][origin]=0;
                cout<<endl;
        }
}
void Graph::display()
{
        int i,j;
```

```
    for(i=1;i<=node;i++)
    {
        for(j=1;j<=node;j++)
        cout<<adj[i][j]<<" ";
        cout<<endl;
    }
}
int main()
{
    Graph g;
    g.createGraph();
    g.display();
    g.createDGraph();
    g.display();
}
```

Output:

```
ente n the no oT nodes

ente r the no oT ed8" 8
+nte r ed8"

ente r de s t2

ente r de s t4

ente r de s to

0 1 0

0 0 0
ente n the no oT node s

ente r the no oT ed8" 8
+nte r ed8"

ente r de s t4

ente r de s to

0 1
0 0

P ro r e s s ex ited aTten 34. 02 s e r and s with return va1ue 0
```

Practical No. 12: Perform various hashing techniques with Linear Probe as collision resolution scheme.

Write a program in c++ to implement linear probing

Code:

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
using namespace std;
class digit
{
	long arr[10];
	public:
	void hash()
		{
			long temp,no,pos,n;
			for(int i=0;i<10;i++)
			arr[i]=0;
			cout<<"\n\n hashing with linear probing \n";
			cout<<"\n enter how many numbers you want \n";
			cin>>n;
			for(int i=1;i<=n;i++)
			{
				cout<<"\n\n enter 8 no of 6 digit\n";
				cin>>no;
				//pos=no%9+1;
				pos=((2*no)+3)%10;
```

```
                    for(int j=0;j<10;j++)
                    {
                        if(arr[pos]==0)
                        {
                            arr[pos]=no;
                            break;
                        }
                        if(pos==9)
                        pos=0;
                        else
                        {
                            pos++;
                        }
                    }
                }
                for(int i=0;i<=9;i++)
                cout<<"\n arr["<<i<<"]= "<<arr[i]<<"\n";
            }
        };
        int main()
        {
            //clrscr();
            digit d;
            d.hash();
        }
```

Output:

[redacted]

enten 8 no of 6 digit

enten 8 no of 6 digit

enten 8 no of 6 digit

ann [0 = 0

ann [1 = 0

ann [2 = 0

ann [3 = 0

ann [4§ = 0

[redacted]

ann [6 = 0

[redacted]

ann [8 = 0

ann[9d = 3

[redacted]

Practical No. 13: Create a minimum spanning tree using any method Kruskal's algorithm or Prim's algorithm

Write a program in c++ to implement minimum spanning tree

Algorithm:

Step 1: Initialize the spanning tree T to contain all the vertices in the graph G but no edges.

Step 2: Choose the edge e with lowest weight from graph G.

Step 3: Check if both vertices from e are within the same set in the tree T, for all such sets of T. If it is not present, add the edge e to the tree T, and replace the two sets that this edge connects.

Step 4: Delete the edge e from the graph G and repeat the step 2 and 3 until there is no more edge to add or until the spanning tree T contains (n-1) vertices.

Step 5: Exit

Code:

```
#include<iostream>

#define MAX 20

using namespace std;

struct edge

{

    int u;

    int v;

    int weight;

    struct edge *link;

}*front=NULL;

int father[MAX];

struct edge tree[MAX];

int n;
```

```
int wt_tree=0;
int count=0;
void make_tree();
void insert_tree(int i,int j,int wt);
void insert_pque(int i,int j,int wt);
struct edge* del_pque();
void create_graph()
{
    int i,wt,max_edges,origin,destin;
    cout<<"enter no of nodes";
    cin>>n;
    max_edges=n*(n-1)/2;
    for(i=1;i<=max_edges;i++)
    {
        cout<<"enter edges"<<i;
        cin>>origin>>destin;
        if((origin==0 )&&(destin==0))
        break;
        cout<<"enter weight for the edge:";
        cin>>wt;
        if(origin>n||destin>n||origin<=0||destin<=0)
        {
            cout<<"ivalid edge"<<endl;
            i--;
        }
        else
        {
            insert_pque(origin,destin,wt);
```

```
		}
	}//end of for
	if(i<n-1)
	{
		cout<<"spanning tree not possible";
		exit(1);
	}
}
int main()
{
	int i;
	create_graph();
	make_tree();
	cout<<"edges to be included in spanning tree"<<endl;
	for(i=1;i<=count;i++)
	{
		cout<<tree[i].u;
		cout<<tree[i].v;
		cout<<endl;
	}
	cout<<"\nweight of minimum spanning tree is"<<wt_tree;
}
void make_tree()
{
	struct edge* temp;
	int node1,node2,root_n1,root_n2;
	while(count<n-1)
	{
```

```
                temp=del_pque();
                node1=temp->u;
                node2=temp->v;
                cout<<"n1="<<node1;
                cout<<"n2="<<node2;
                while(node1>0)
                {
                        root_n1=node1;
                        node1=father[node1];
                }
                while(node2>0)
                {
                        root_n2=node2;
                        node2=father[node2];
                }
                cout<<"rootn1="<<root_n1<<endl;
                cout<<"rootn2="<<root_n2<<endl;
                if(root_n1!=root_n2)
                {
                        insert_tree(temp->u,temp->v,temp->weight);
                        wt_tree=wt_tree+temp->weight;
                        father[root_n2]=root_n1;
                }
        }
}
void insert_tree(int i,int j,int wt)
{
        cout<<"the edges inserted in the spanning tree:"<<endl;
```

```
        count++;
        tree[count].u=i;
        tree[count].v=j;
        tree[count].weight=wt;
}
void insert_pque(int i,int j,int wt)
{
        struct edge* temp,*q;
        //temp=(struct edge*)malloc(sizeof(struct edge));
        temp=new struct edge();
        temp->u=i;
        temp->v=j;
        temp->weight=wt;
        if(front==NULL || temp->weight<front->weight)
        {
                temp->link=front;
                front=temp;
        }
        else
        {
                q=front;
                while(q->link!=NULL && q->link->weight<=temp->weight)
                q=q->link;
                temp->link=q->link;
                q->link=temp;
                if(q->link==NULL)
                temp->link=NULL;
        }
```

```
}
struct edge* del_pque()
{
        struct edge* temp;
        temp=front;
        cout<<"edge processed is"<<temp->u<<" "<<temp->v<<" "<<temp->weight;
        front=front->link;
        return temp;
}
```

Output:

```
enter no of nodes 2
enter edges11
2
enter weight for the edge:1
edge processed is1 2 1n1=1n2=2rootn1=1
rootn2=2
the edges inserted in the spanning tree:
edges to be included in spanning tree
12

weight of minimum spanning tree is1
--------------------------------
Process exited after 9.943 seconds with return value 0
Press any key to continue . . .
```

Practical No. 14: Implementation of graph traversal

a) **Aim: Write a program in c++ to implement Depth First Search (DFS)**

Algorithm:

Step 1: Set status = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its status = 2 (waiting state)

Step 3: Repeat steps 4 and 5 until stack is empty

Step 4: Pop the top node N. Process it and set its status = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose status = 1) and set their status = 2 (waiting state)

Step 6: Exit

Code:

```
#include<iostream>

#include<stdio.h>

#define max 10

using namespace std;

/* a function to build adjacency matrix of a graph */

void buildadjm(int adj[][max], int n)

{

        int i,j;

        for(i=1;i<=n;i++)

        for(j=1;j<=n;j++)

        {

                cout<<"enter 1 or 0:"<<i<<j;

                cin>>adj[i][j];

        }
```

```
}
/* a function to visit the nodes in a depth first order */
void dfs(int x,int visited[],int adj[][max],int n)
{
	int j;
	visited[x] = 1;
	//printf("\nThe node visited id %d\n",x);
	cout<<"the node visited id is"<<x;
	for(j=1;j<=n;j++)
	{
		if (adj[x][j] ==1 && visited[j] ==0)
		dfs(j,visited,adj,n);
	}
}
int main()
{
	int adj[max][max],node,n;
	int i, visited[max];
	cout<<"enter the no of nodes"<<endl;
	cin>>n;
	buildadjm(adj,n);
	for(i=1;i<=n;i++)
	visited[i] =0;
	cout<<"enter start node";
	cin>>node;
	if(visited[node] ==0)
	dfs(node,visited,adj,n);
}
```

Output:

```
enter the no of nodes
3
enter 1 or 0:111
enter 1 or 0:120
enter 1 or 0:131
enter 1 or 0:212
enter 1 or 0:220
enter 1 or 0:233
enter 1 or 0:310
enter 1 or 0:324
enter 1 or 0:335
enter start node1
the node visited id is1the node visited id is3
--------------------------------
Process exited after 19.32 seconds with return value 0
Press any key to continue . . .
```

b) Write a program in c++ to implement Breath First Search (BFS)

Algorithm:

Step 1: Input the vertices of the graph and its edges G = (V, E)

Step 2: Input the source vertex and assign it to the variable S.

Step 3: Add or Push the source vertex to the queue.

Step 4: Repeat step 5 and 6 until the queue is empty (front > rear)

Step 5: Pop the front element of the queue and display it as visited.

Step 6: Push the vertices, which is neighbor to just popped element. If it is not in the queue and displayed (not visited)

Step 7: Exit

Code:

```
#include<iostream>

#define MAX 50

using namespace std;

struct node
```

```
{
        int vertex;
        node *next;
};
node *adj[MAX];
int totNodes;//number of nodes in graph
int queue[MAX],front=-1,rear=-1;
void enqueue(int item)
{
        rear=rear+1;
        queue[rear]=item;
        if(front==-1)
        front=0;
}
int dequeue()
{
        int delItem=queue[front];
        if(front==rear)
        front=rear=-1;
        else
        front=front+1;
        return(delItem);
}
int isQueueEmpty()
{
        if(front==-1)
        return  1;
        else
```

```
	return 0;
}
void createGraph()
{
	node *new1,*last;
	int neighbours,neighbour_val;
	cout<<"Proceeding for graph creation..."<<endl;
	cout<<"enter the number of nodes"<<endl;
	cin>>totNodes;
	for(int i=1;i<=totNodes;i++)
	{
		last=NULL;//store address of next node
		cout<<"enter num of nodes neighbour to"<<i<<endl;
		cin>>neighbours;
		cout<<"neighbours of"<<i<<"are: "<<endl;
		for(int j=1;j<=neighbours;j++)
		{
			cout<<"enter name of the neighbour: "<<endl;
			cin>>neighbour_val;
			new1 = new node;//creation of node
			new1->vertex=neighbour_val;
			new1->next=NULL;
			if(adj[i]==NULL)
			adj[i]=last=new1;
			else
			{
				last->next=NULL;
				last=new1;
```

```
                }
            }
        }
}
void BFS_traversal()
{
        node *temp;
        int startNode,status[MAX],N,v;
        const int ready=1,wait=2,processed=3;
        cout<<"enter the start node";
        cin>>startNode;
        for(int i=1;i<=totNodes;i++)
        {
            status[i]=ready;
        }
        enqueue(startNode);
        status[startNode]=wait;
        while(isQueueEmpty()!=1)
        {
            N=dequeue();
            status[N]=processed;
            cout<<" "<<N;
            temp=adj[N];
            while(temp!=NULL)
            {
                v=temp->vertex;
                if(status[v]==ready)
                {
```

```
                    enqueue(v);
                    status[v]=wait;
                }
                temp=temp->next;
            }
        }
    }
    int main()
    {
        createGraph();
        cout<<"BFS Traverse: "<<endl;
        BFS_traversal();
        return 0;
    }
```

Output:

```
Proceeding for graph creation...
enter the number of nodes
4
enter num of nodes neighbour to1
2
neighbours of1are:
enter name of the neighbour:
2
enter name of the neighbour:
3
enter num of nodes neighbour to2
1
neighbours of2are:
enter name of the neighbour:
3
enter num of nodes neighbour to3
1
neighbours of3are:
enter name of the neighbour:
4
enter num of nodes neighbour to4
0
neighbours of4are:
BFS Traverse:
enter the start node1
  1  2  3  4
--------------------------------
Process exited after 35.92 seconds with return value 0
Press any key to continue . . .
```