# WEEK - 1

## EX. NO. 01 : Programs on recursion : DATE:

### AIM : Write a c++ program to compute : a.)Factorial of a given number. b.)Tower of Hanoi.

**c.)GCD of a number.**

**a.)C++ program to compute factorial of a given number . PROGRAM :**

//Program to implement factorial of a given number #include<iostream>

using namespace std; int factorial(int);

int main()

{

int n;

cout<<"Enter n value: "; cin>>n;

cout<<"Factorial value = "<<factorial(n);

}

int factorial(int n)

{

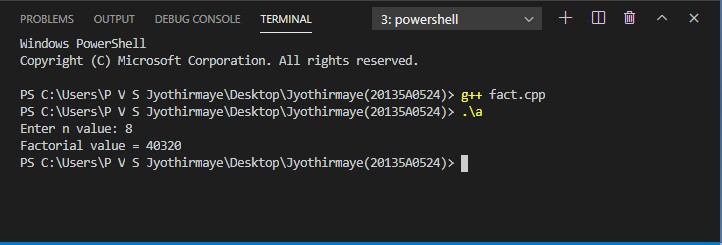
int fact=1;

for(int count=1;count<=n;count++)

fact=fact\*count; return fact;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement tower of Hanoi problem. PROGRAM :

//Program to implement Tower of Hanoi #include <iostream>

#include <bits/stdc++.h> using namespace std;

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 1)

{

cout << "Move disk 1 from rod " << from\_rod << " to rod " << to\_rod<<endl; return;

}

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

cout << "Move disk " << n << " from rod " << from\_rod <<" to rod " << to\_rod << endl; towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

int main()

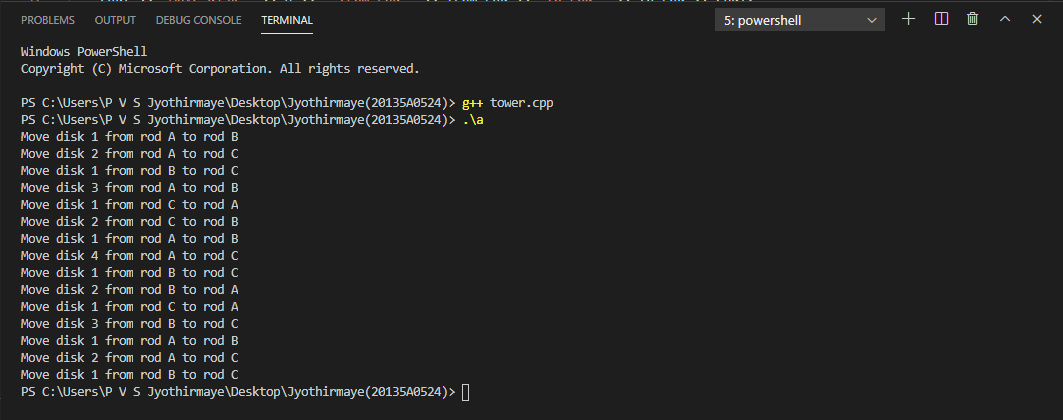
{

int n = 4;

towerOfHanoi(n, 'A', 'C', 'B'); return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### a.)C++ program to implement GCD. PROGRAM :

//Program to find GCD of two numbers #include<iostream>

using namespace std; int gcd(int n1, int n2); int main()

{

int n1,n2;

cout<<"Enter values of n1 and n2 "; cin>>n1>>n2;

cout <<"GCD of "<< n1 <<" and "<< n2 <<" is "<<gcd(n1,n2)<<"\n"; return 0;

}

int gcd(int n1, int n2)

{

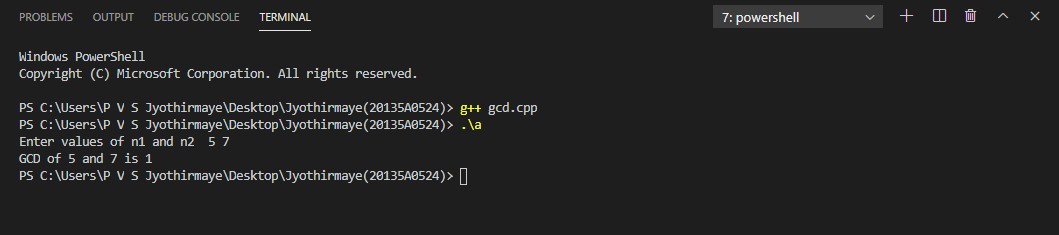
if(n2!=0)

return gcd(n2,n1%n2); else

return n1;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

# WEEK - 2

## Ex. No. 2 : Programs to implement search algorithms. Date :

### AIM : Write a C++ program to implement : a.)Linear search.

**b.)Binary search. c.)Fibonacci search.**

**a.)C++ program to implement Linear search . PROGRAM :**

//Program to implement Linear search #include<iostream>

using namespace std; int main()

{

int arr[5],key,n,j;

cout<<"Enter no. of elements :"; cin>>n;

cout<<"Enter "<<n<<" elements "; for(int i=0;i<n;i++)

cin>>arr[i];

cout<<"Enter the key element to search : "; cin>>key;

for(j=0;j<n;j++)

{

if(arr[j]==key)

{

cout<<"Element found at : "<<j; break;

}

}

if(j==n)

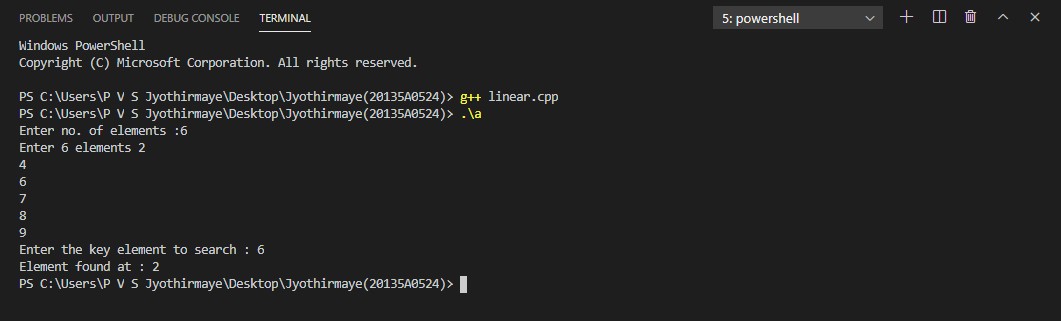
{

cout<<"element not found ";

}

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement Binary search. PROGRAM :

//Program to implement Binary search #include<iostream>

using namespace std; int main()

{

int i,arr[10],key,first,last,mid,n; cout<<"Enter no. of elements "; cin>>n;

cout<<"Enter "<<n<<" elements "; for(int i=0;i<n;i++)

cin>>arr[i];

cout<<"Enter element to search: "; cin>>key;

first=0; last=n-1;

mid=(first+last/2); while(first<=last)

{

if(arr[mid]<key) first=mid+1;

else if(arr[mid]==key)

{

cout<<"\n The number "<<key<<" found at "<<mid; break;

}

else

last=mid-1; mid=(first+last)/2;

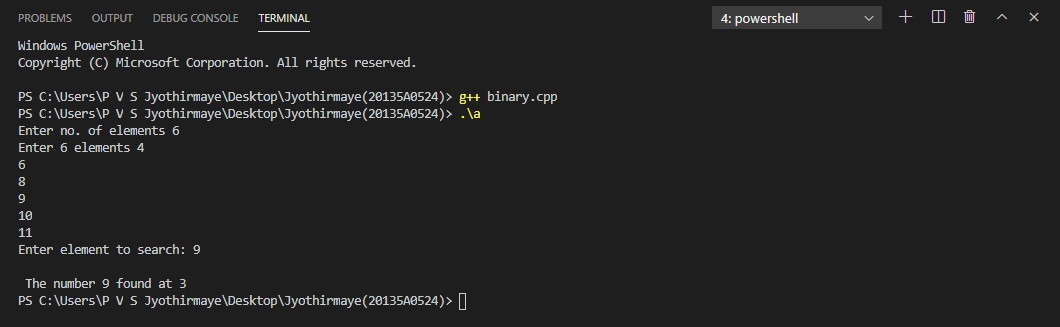
}

if(first>last)

cout<<"\n The number "<<key<<" not found";

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### c.)C++ program to implement Fibonacci search. PROGRAM :

//Program to implement fibonacci search #include <iostream>

using namespace std; int min(int x, int y)

{

return (x<=y)? x : y;

}

int fibSearch(int arr[], int x, int n)

{

int fib2 = 0; int fib1 = 1;

int fib = fib2 + fib1; while (fib < n)

{

fib2 = fib1; fib1 = fib;

fib = fib2 + fib1;

}

int offset = -1; while (fib > 1)

{

int i = min(offset+fib2, n-1); if (arr[i] < x)

{

fib = fib1; fib1 = fib2;

fib2 = fib - fib1; offset = i;

}

else if (arr[i] > x)

{

fib = fib2;

fib1 = fib1 - fib2; fib2 = fib - fib1;

}

else return i;

}

if(fib1 && arr[offset+1]==x)return offset+1; return -1;

}

int main()

{

int arr[] = {10, 22, 35, 40, 45, 50, 80, 82,

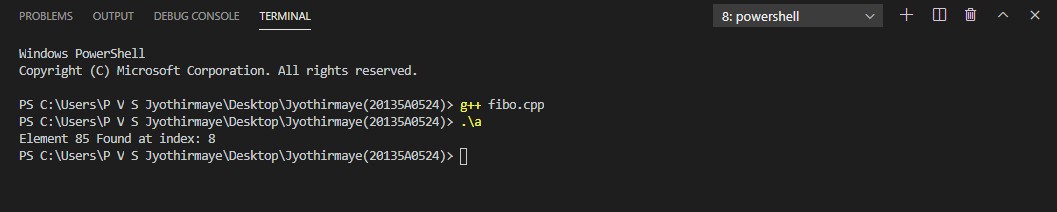
85, 90, 100};

int n = sizeof(arr)/sizeof(arr[0]); int x = 85;

cout<<"Element "<<x<<" Found at index: "<<fibSearch(arr, x, n);

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

# WEEK - 3

## Ex. No. 3 : Programs to implement sorting algorithms. Date :

### AIM : Write a C++ program to implement : a.)Bubble sort.

**b.)Insertion sort. c.)Quick sort. d.)Merge sort.**

**a.)C++ program to implement Bubble sort. PROGRAM :**

//Program for Bubble sort #include<iostream>

using namespace std; int main()

{

int a[50];

int i,j,n,temp;

cout<<"Enter no. of elements "; cin>>n;

cout<<"Enter "<<n<<" elements "; for(i=0;i<n;i++)

cin>>a[i]; for(i=0;i<n;i++)

{

for(j=0;j<(n-i);j++)

{

if(a[j]>a[j+1])

{

temp=a[j]; a[j]=a[j+1]; a[j+1]=temp;

}

}

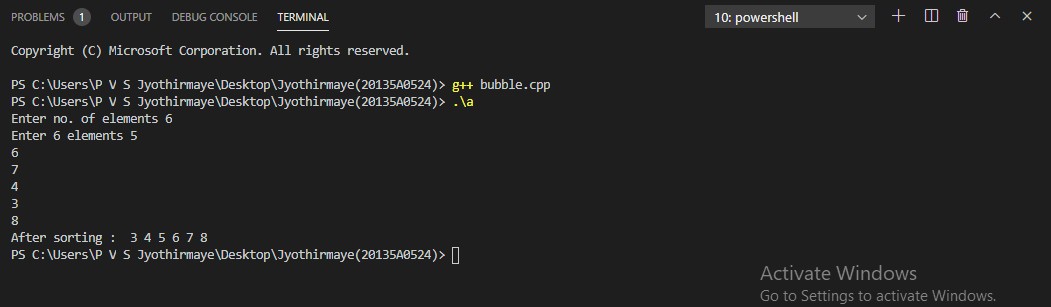
}

cout<<"After sorting : "; for(i=0;i<n;i++)

cout<<" "<<a[i]; return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement Insertion sort. PROGRAM :

//Program for Insertion sort #include<iostream>

using namespace std; int main()

{

int n, arr[50], i, j, temp; cout<<"Enter Array Size : "; cin>>n;

cout<<"Enter Array Elements : "; for(i=0; i<n; i++)

{

cin>>arr[i];

}

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

{

temp=arr[i]; j=i-1;

while((temp<arr[j]) && (j>=0))

{

arr[j+1]=arr[j]; j=j-1;

}

arr[j+1]=temp;

}

cout<<"Array after sorting : \n"; for(i=0; i<n; i++)

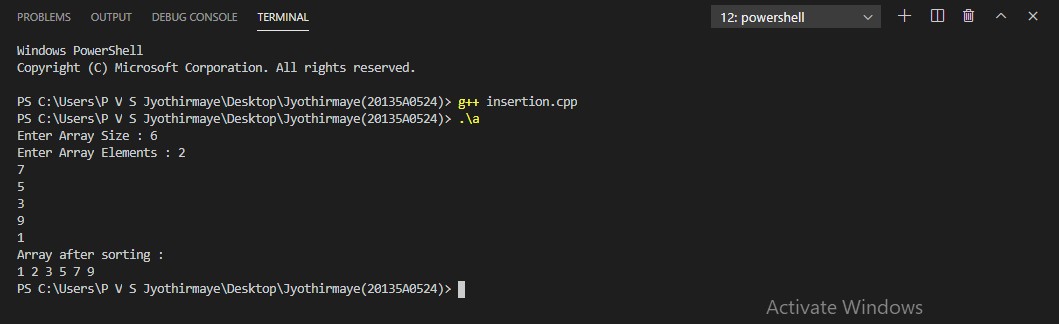
{

cout<<arr[i]<<" ";

}

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### c.)C++ program to implement Quick sort. PROGRAM :

//Program for Quick sort #include <iostream> using namespace std;

void quick\_sort(int[],int,int); int partition(int[],int,int);

int main()

{

int a[50],n,i;

cout<<"How many elements ? "; cin>>n;

cout<<"\nEnter array elements: "; for(i=0;i<n;i++)

cin>>a[i]; quick\_sort(a,0,n-1);

cout<<"\nArray after sorting: "; for(i=0;i<n;i++)

cout<<a[i]<<" "; return 0;

}

void quick\_sort(int a[],int l,int u)

{

int j; if(l<u)

{

j=partition(a,l,u); quick\_sort(a,l,j-1); quick\_sort(a,j+1,u);

}

}

int partition(int a[],int l,int u)

{

int v,i,j,temp; v=a[l];

i=l; j=u+1; do

{

do

i++;

while(a[i]<v&&i<=u); do

j--;

while(v<a[j]);

if(i<j)

{

temp=a[i]; a[i]=a[j]; a[j]=temp;

}

}while(i<j);

a[l]=a[j]; a[j]=v; return(j);

}

### C:\Users\P V S Jyothirmaye\Desktop\quick.PNGOUTPUT :

**RESULT :** The above program is executed successfully and verified .

### d.)C++ program to implement Merge sort. PROGRAM :

//Program for Merge sort

#include <iostream> using namespace std;

void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1; int n2 = r - m;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) L[i] = arr[l + i];

for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

int i = 0; int j = 0; int k = l;

while (i < n1 && j < n2) { if (L[i] <= R[j]) {

arr[k] = L[i]; i++;

}

else {

arr[k] = R[j]; j++;

} k++;

}

while (i < n1)

{

arr[k] = L[i]; i++;

k++;

}

while (j < n2)

{

arr[k] = R[j]; j++;

k++;

}

}

void mergeSort(int arr[],int l,int r)

{

if(l>=r)

{

return;

}

int m = (l+r-1)/2; mergeSort(arr,l,m); mergeSort(arr,m+1,r); merge(arr,l,m,r);

}

void printArray(int A[], int size)

{

for (int i = 0; i < size; i++) cout << A[i] << " ";

}

int main()

{

int n;

cout<<"Enter size of the Array: "; cin>>n;

int arr[n],i;

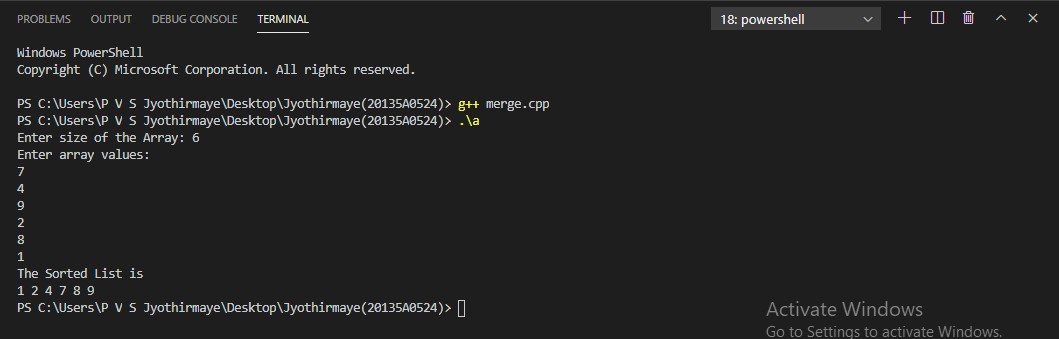
cout<<"Enter array values:\n"; for(i=0;i<n;i++)

cin>>arr[i]; mergeSort(arr,0,n-1); cout<<"The Sorted List is\n"; for(i=0;i<n;i++)

cout<<arr[i]<<" "; return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

# WEEK - 4

## Ex. No. 4 : Program on Hashing. Date :

### AIM : Write a C++ program to implement different types of Hash functions . PROGRAM :

//Program on hashing #include<iostream> using namespace std; int n;

void insertion(); void deletion(); void displaying(); void searching();

using namespace std; struct Node

{

int data;

Node\* next;

};

Node \*head = NULL , \*a[100]; void insertion()

{

int element;

cout<<"Enter the element to insert "<<endl;

cin>>element;

int i= element%n;

Node \*p1 = new Node(); p1->data=element;

p1->next=NULL; if(a[i]==NULL)

{

a[i]=p1;

}

else

{

Node \*temp = a[i]; while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = p1;

}

}

void deletion()

{

int item,count=0;

cout<<"Enter the element that you want to delete "<<endl;

cin>>item; int i=item%n; if(a[i]==NULL)

cout<<"Element not found and list is empty "<<endl; else

{

Node \*ptr,\*pre; ptr=a[i];

if(ptr->next==NULL and ptr->data==item)

{

a[i]=NULL;

free(ptr);

}

else if(ptr->next==NULL and ptr->data!=item) cout<<"not found"<<endl;

else

{

if(ptr!=NULL and ptr->data==item)

{

a[i]=ptr->next; free(ptr);

}

else

{

while(ptr->next!=NULL)

{

if(ptr->data==item)

{

pre->next=ptr->next; count++;

free(ptr); break;

}

pre=ptr; ptr=ptr->next;

}

if(count==0)

{

int c=0;

Node \*p1,\*p2; p1=a[i]; while(p1!=NULL)

{

if(p1->data==item)

{

c++;

break;

}

p2=p1; p1=p1->next;

}

if (c>0)

{

p2->next=NULL; free(p1);

}

else

{

cout<<"Element not found"<<endl;

}

}

}

}

}

}

void searching()

{

int element,i;

cout<<"Enter the element to search : "; cin>>element;

i=element%n; Node \*ptr;

ptr = a[i]; while(ptr!=NULL)

{

if(ptr->data==element)

{

cout<<"Element found"<<endl; break;

}

ptr=ptr->next;

}

if(ptr==NULL)

{

cout<<"Element not found"<<endl;

}

}

void displaying()

{

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

{

cout<<i<<" ";

Node \*ptr; ptr = a[i];

while(ptr!=NULL)

{

cout<<ptr->data<<" "; ptr = ptr->next;

}

cout<<endl;

}

}

int main()

{

int choice;

cout<<"Enter the SIZE of the HASH TABLE.."<<endl; cin>>n;

for(int j=0;j<n;j++)

{

a[j]=NULL;

}

while(1)

{

cout<<"1. Insertion "<<endl; cout<<"2. Deletion "<<endl; cout<<"3. Displaying "<<endl; cout<<"4. Searching "<<endl; cout<<"5. Exit"<<endl; cout<<"Enter Your Option "<<endl; cin>>choice;

switch(choice)

{

case 1: insertion(); break;

case 2: deletion(); break;

case 3: displaying(); break;

case 4: searching(); break;

case 5: cout<<"EXIT"<<endl; exit(0);

break;

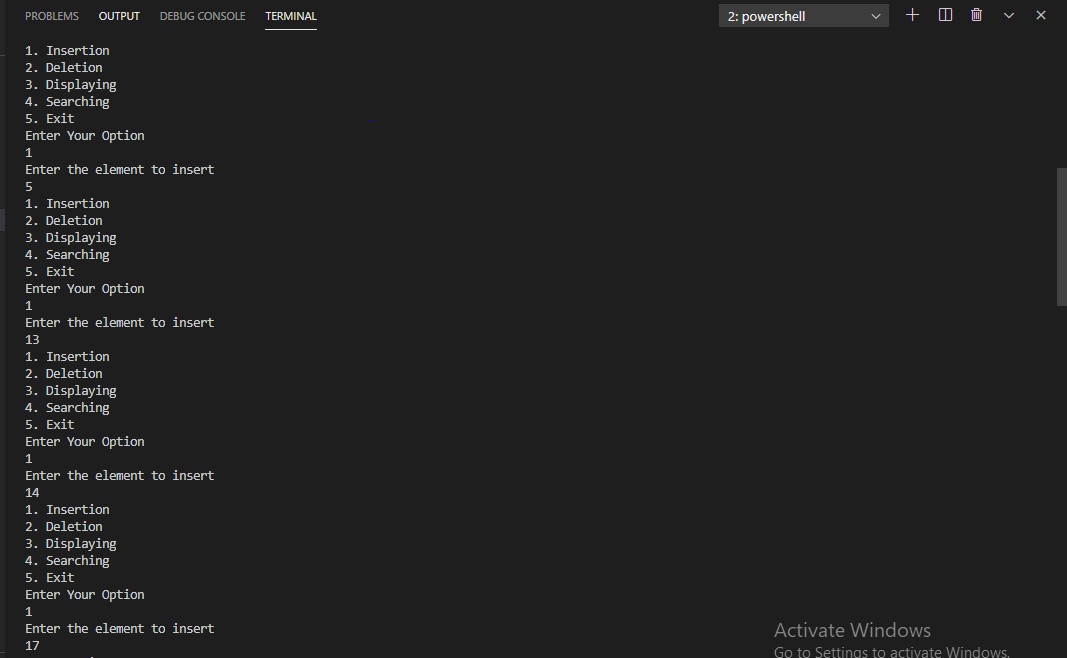
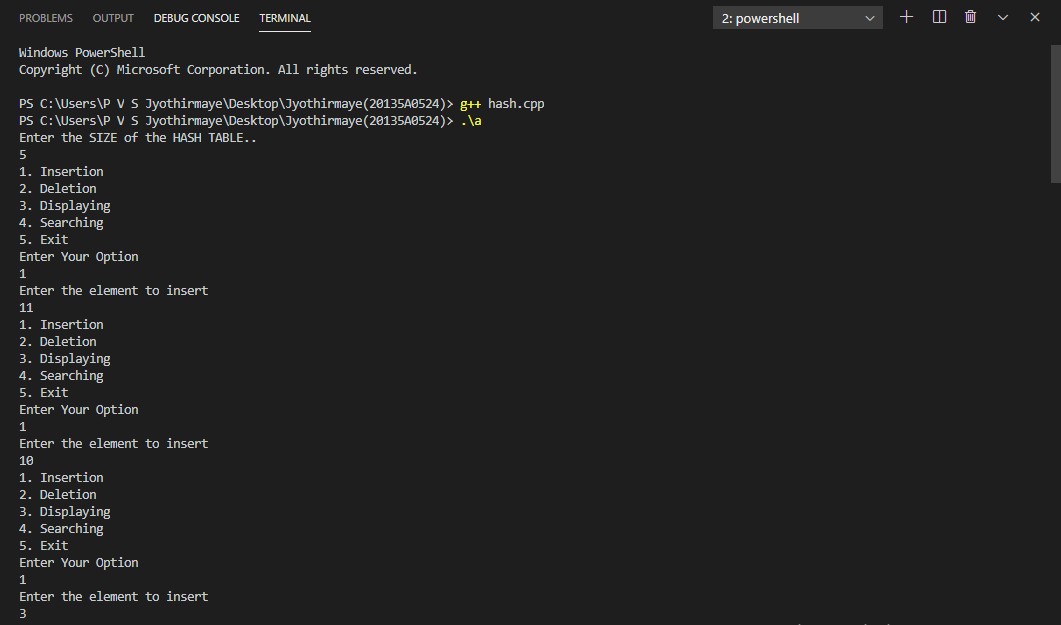
default :cout<<"Invalid Choice"<<endl;

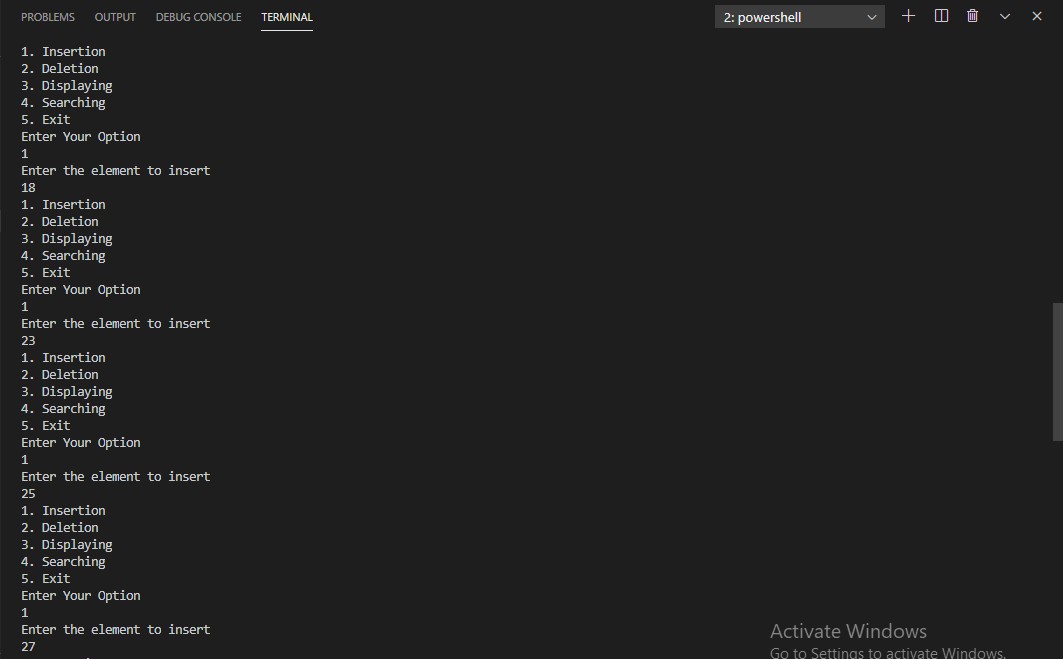
}

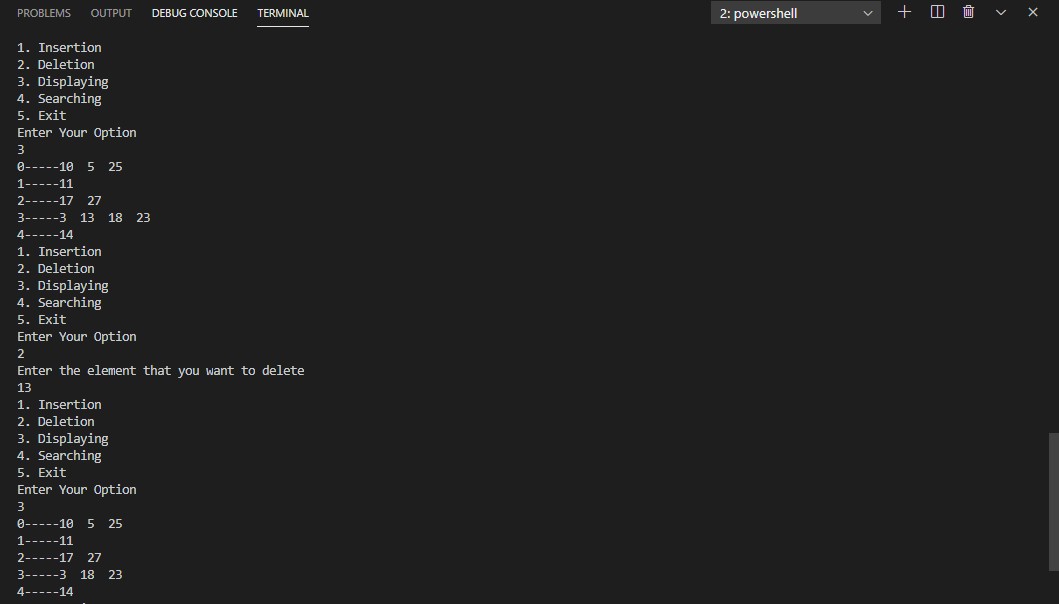
}

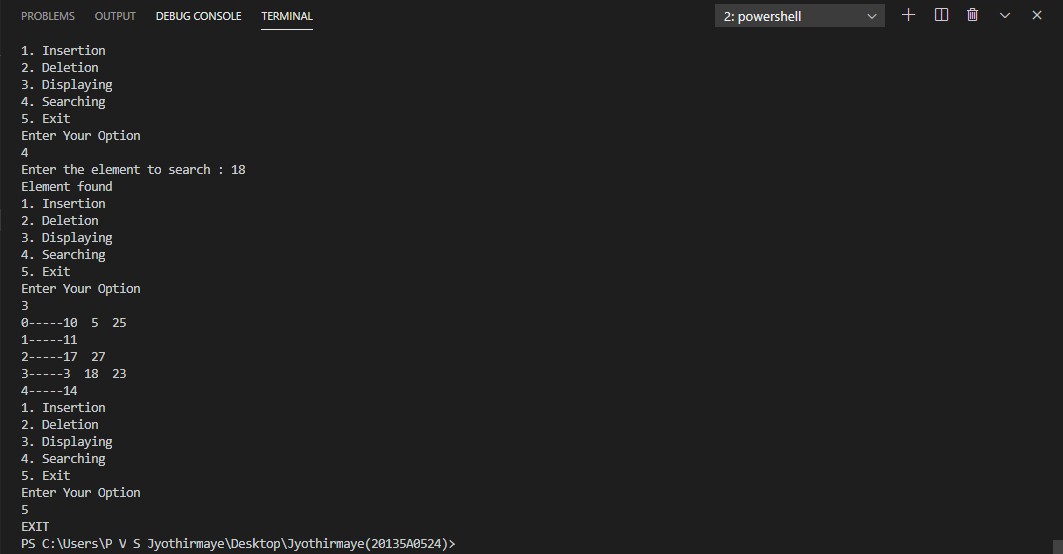
}

### OUTPUT :









**RESULT :** The above program is executed successfully and verified .

# WEEK - 5

## Ex. No. 5 : Programs to implement stacks and queues using arrays . Date :

### AIM : Write a C++ program to implement : a.)Stack .

**b.)Queue.**

**a.)C++ program to implement Stack. PROGRAM :**

//Program on stack #include<iostream> using namespace std;

int stack[100], n=100, top=-1; void push(int val)

{

if(top>=n-1)

cout<<"Stack Overflow"<<endl; else

{

top++; stack[top]=val;

}

}

void pop()

{

if(top<=-1)

cout<<"Stack Underflow"<<endl; else

{

cout<<"The popped element is "<<stack[top]<<endl; top--;

}

}

void display()

{

if(top>=0)

{

cout<< "Stack elements are:"; for(int i=top; i>=0; i--)

cout<< stack[i] <<" "; cout<< endl;

}

else

cout<< "Stack is empty";

}

int main()

{

int ch, val; do

{

cout<<"1) Push in stack"<<endl; cout<<"2) Pop from stack"<<endl; cout<<"3) Display stack"<<endl; cout<<"4) Exit"<<endl; cout<<"Enter choice:"<<endl; cin>>ch;

switch(ch)

{

case 1:

{

cout<<"Enter the value to be pushed:"<<endl; cin>>val;

push(val); break;

}

case 2:

{

pop(); break;

}

case 3:

{

display();

break;

}

case 4:

{

cout<<"Exit"<<endl; break;

}

default:

{

cout<<"Invalid choice"<<endl;

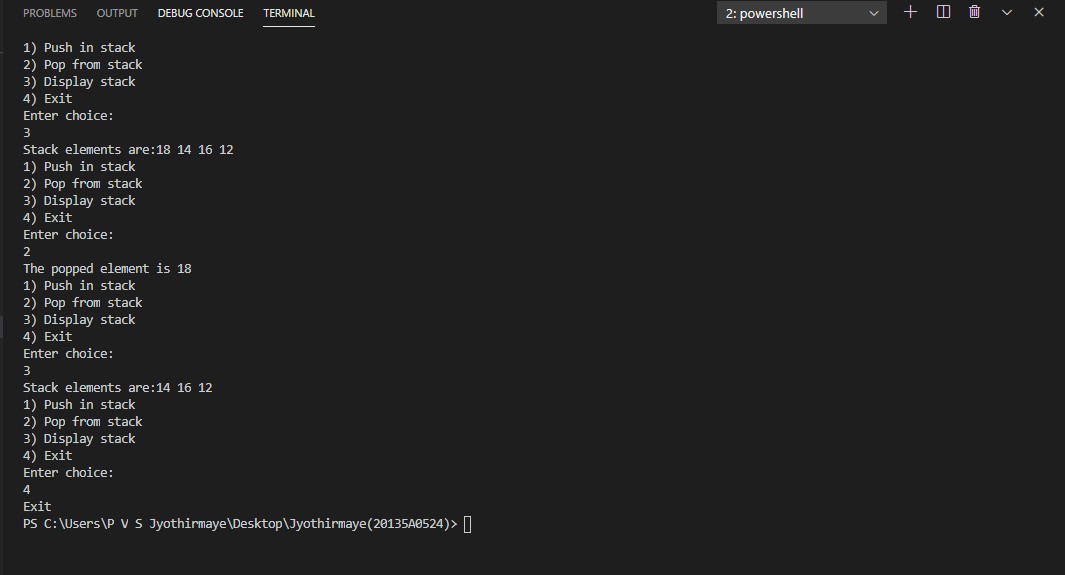
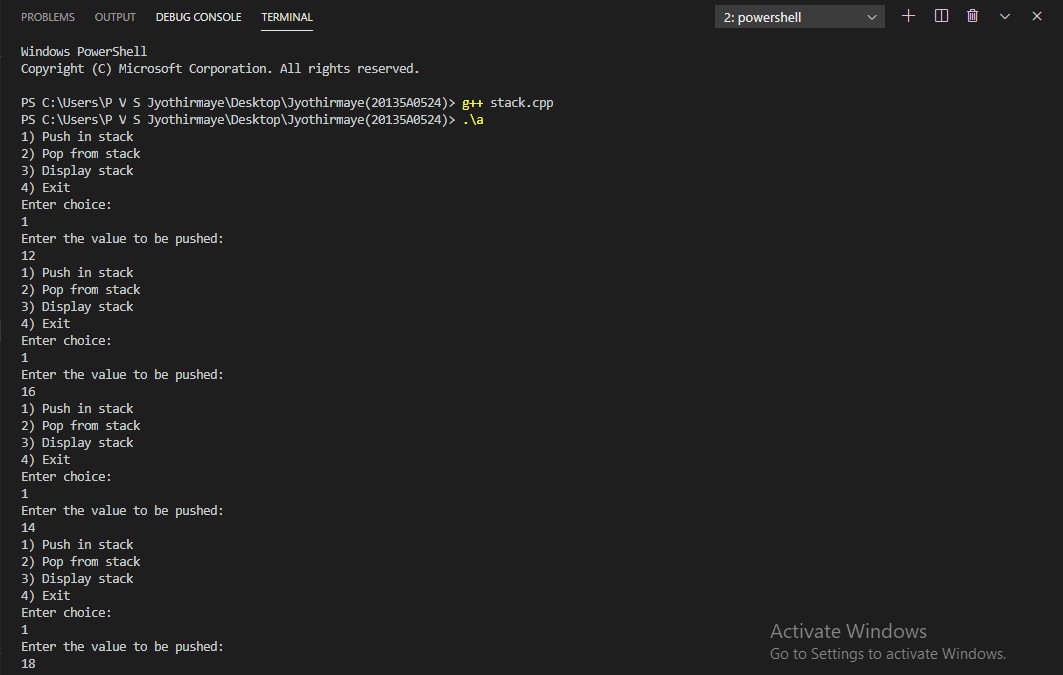
}

}

}while(ch!=4); return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement Queue . PROGRAM :

//Program on queue #include <iostream> using namespace std;

int queue[100], n = 100, front = - 1, rear = - 1; void Insert() {

int val;

if (rear == n - 1)

cout<<"Queue Overflow"<<endl; else {

if (front == - 1) front = 0;

cout<<"Insert the element in queue : "<<endl; cin>>val;

rear++; queue[rear] = val;

}

}

void Delete() {

if (front == - 1 || front > rear) { cout<<"Queue Underflow "; return ;

} else {

cout<<"Element deleted from queue is : "<< queue[front] <<endl; front++;;

}

}

void Display() { if (front == - 1)

cout<<"Queue is empty"<<endl; else {

cout<<"Queue elements are : "; for (int i = front; i <= rear; i++) cout<<queue[i]<<" ";

cout<<endl;

}

}

int main() { int ch;

cout<<"1) Insert element to queue"<<endl; cout<<"2) Delete element from queue"<<endl; cout<<"3) Display all the elements of queue"<<endl; cout<<"4) Exit"<<endl;

do {

cout<<"Enter your choice : "<<endl; cin>>ch;

switch (ch) { case 1: Insert(); break;

case 2: Delete(); break;

case 3: Display(); break;

case 4: cout<<"Exit"<<endl; break;

default: cout<<"Invalid choice"<<endl;

}

} while(ch!=4); return 0;

}

### C:\Users\P V S Jyothirmaye\Desktop\queue1.PNGC:\Users\P V S Jyothirmaye\Desktop\queue2.PNGOUTPUT :

**RESULT :** The above program is executed successfully and verified .

# WEEK - 6

## Ex. No. 6 : Programs to implement Stack applications . Date :

### AIM : Write a C++ program to implement : a.)Factorial using stack . b.)Evaluation of Postfix Expression. c.)Number conversion.

**a.)C++ program to implement Factorial using stack. PROGRAM :**

//Program on factorial using stack #include<iostream>

using namespace std; class stack

{

int arr[100],ptr; public:

stack()

{

ptr=0;

}

void push(int n)

{

arr[ptr++]=n;

}

int pop()

{

return arr[--ptr];

}

};

int main()

{

int n,i; stack s;

cout<<"Enter the number to find factorial :"; cin>>n;

for(i=n;i>0;i--)

{

s.push(i);

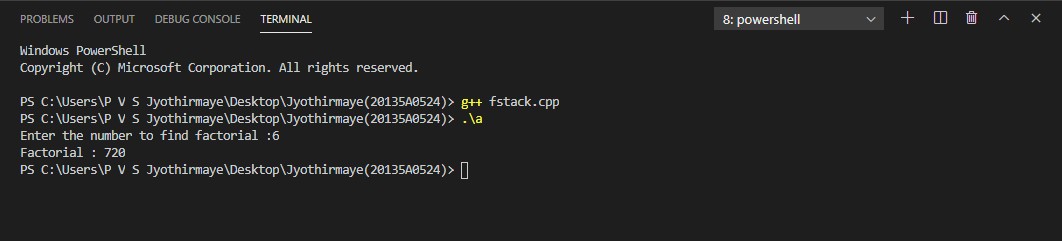
}

int fact=1;

while(n--)fact\*=s.pop(); cout<<"Factorial : "<<fact<<endl; return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to Evaluate postfix expression using stack . PROGRAM :

//Program on Evaluation of postfix expression #include<iostream>

using namespace std; class evaluation

{

public:

int st[50]; int top; char str[50]; evaluation()

{

top=-1;

}

void push(int item)

{

top++; st[top]=item;

}

int pop()

{

int item=st[top]; top--;

return item;

}

int operation(int a,int b,char opr)

{

switch(opr)

{

case '+': return a+b; case '-': return a-b; case '\*': return a\*b; case '/': return a/b; default: return 0;

}

}

int calculatepostfix();

};

int evaluation::calculatepostfix()

{

int index=0; while(str[index]!='#')

{

if(isdigit(str[index]))

{

push(str[index]-'0');

}

else

{

int x=pop(); int y=pop();

int result=operation(x,y,str[index]); push(result);

}

index++;

}

return pop();

}

int main()

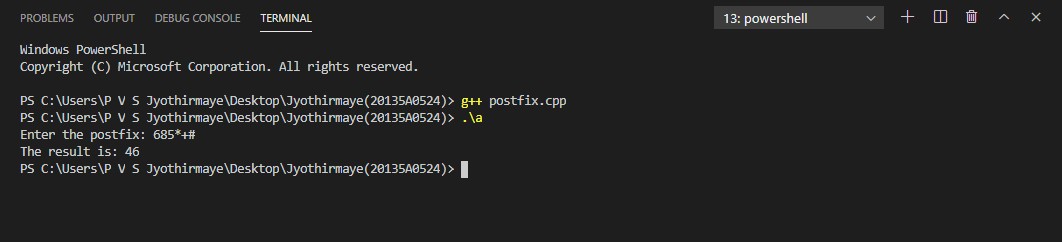
{

evaluation eval; cout<<"Enter the postfix: "; cin>>eval.str;

int result=eval.calculatepostfix(); cout<<"The result is: "<<result<<"\n";

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### c.)C++ program to implement number conversion using stack . PROGRAM :

//Program on number conversion #include <iostream>

#include <cmath> using namespace std;

int BinaryToDecimal(long long n)

{

int decimalNumber = 0, i = 0, remainder; while (n!=0)

{

remainder = n%10; n /= 10;

decimalNumber += remainder\*pow(2,i);

++i;

}

return decimalNumber;

}

long long convertDecimalToBinary(int n)

{

long long binaryNumber = 0; int remainder, i = 1, step = 1;

while (n!=0)

{

remainder = n%2;

cout << "Step " << step++ << ": " << n << "/2, Remainder = " << remainder << ", Quotient = "

<< n/2 << endl; n /= 2;

binaryNumber += remainder\*i; i \*= 10;

}

return binaryNumber;

}

int main()

{

int ch;

do

{

cout<<"1.BINARY TO DECIMAL \n"; cout<<"2.DECIMAL TO BINARY \n";

cout<<"3.Exit\n"; cout<<"Enter choice : "; cin>>ch;

switch(ch)

{

case 1:

{

long long n;

cout << "Enter a binary number : "; cin >> n;

cout << n << " in binary = " << BinaryToDecimal(n) << " in decimal \n"; break;

}

case 2:

{

int n, binaryNumber;

cout << "Enter a decimal number : "; cin >> n;

binaryNumber = convertDecimalToBinary(n);

cout << n << " in decimal = " << binaryNumber << " in binary " << endl ;

break;

}

case 3:

{

cout<<"Exit\n"; break;

}

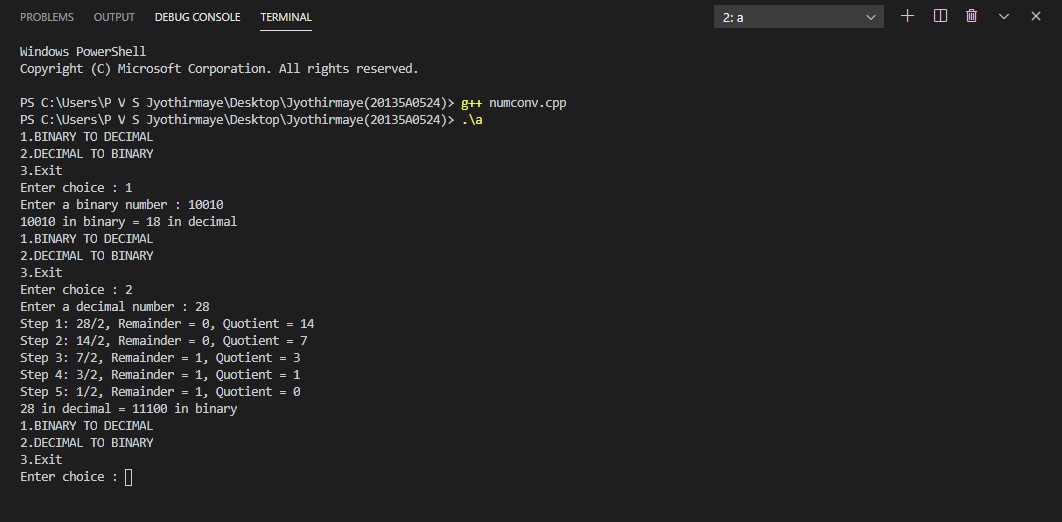
default: cout<<"Invalid choice\n";

}

}while(ch!=3);

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

# WEEK - 7

## Ex. No. 7 : Programs to implement types of queues . Date :

### AIM : Write a C++ program to implement : a.)Priority queue.

**b.)Circular queue.**

**a.)C++ program to implement Priority queue. PROGRAM :**

//Program on priority queue #include<iostream>

#define N 20

using namespace std; int Q[N],Pr[N];

int r = -1,f = -1;

void enqueue(int data,int p)

{

int i; if((f==0)&&(r==N-1))

cout<<"Queue is full"; else

{

if(f==-1)

{

f = r = 0;

Q[r] = data; Pr[r] = p;

}

else if(r == N-1)

{

for(i=f;i<=r;i++) {

Q[i-f] = Q[i];

Pr[i-f] = Pr[i]; r = r-f;

f = 0;

for(i = r;i>f;i--)

{

if(p>Pr[i])

{

Q[i+1] = Q[i];

Pr[i+1] = Pr[i];

}

else

break; Q[i+1] = data; Pr[i+1] = p; r++;

}

}

}

else

{

for(i = r;i>=f;i--)

{

if(p>Pr[i])

{

Q[i+1] = Q[i];

Pr[i+1] = Pr[i];

}

else

break;

}

Q[i+1] = data; Pr[i+1] = p; r++;

}

}

}

void print()

{

int i;

for(i=f;i<=r;i++)

{

cout<<"Element = "<<Q[i]<<" Priority = "<<Pr[i]<<endl;

}

}

int dequeue()

{

if(f == -1)

{

cout<<"Queue is Empty";

}

else

{

cout<<"deleted Element = "<<Q[f]<<endl; cout<<"Its Priority = "<<Pr[f]<<endl;

if(f==r)

f = r = -1;

else

f++;

}

return 0;

}

int main()

{

int opt,n,i,data,p;

cout<<"Enter Your Choice : "<<endl; do{

cout<<"1. Insert the Data in Queue\n2. Display the Data in Queue \n3. Delete the data from t he Queue\n4. Exit"<<endl;

cout<<"Enter the choice : "; cin>>opt;

switch(opt){ case 1:

cout<<"Enter the number of data : "<<endl; cin>>n;

cout<<"Enter your data and Priority of data : "<<endl; i=0;

while(i<n){ cout<<"Enter data : "; cin>>data;

cout<<"Enter priority : "; cin>>p; enqueue(data,p);

i++;

}

break; case 2:

print();

break; case 3:

dequeue(); break;

case 4:

cout<<"Exit"<<endl; break;

default:

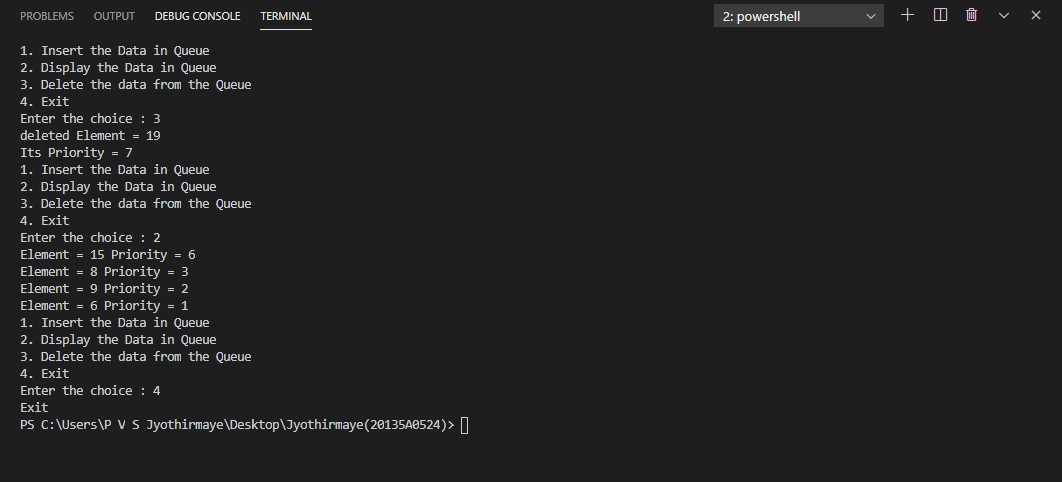
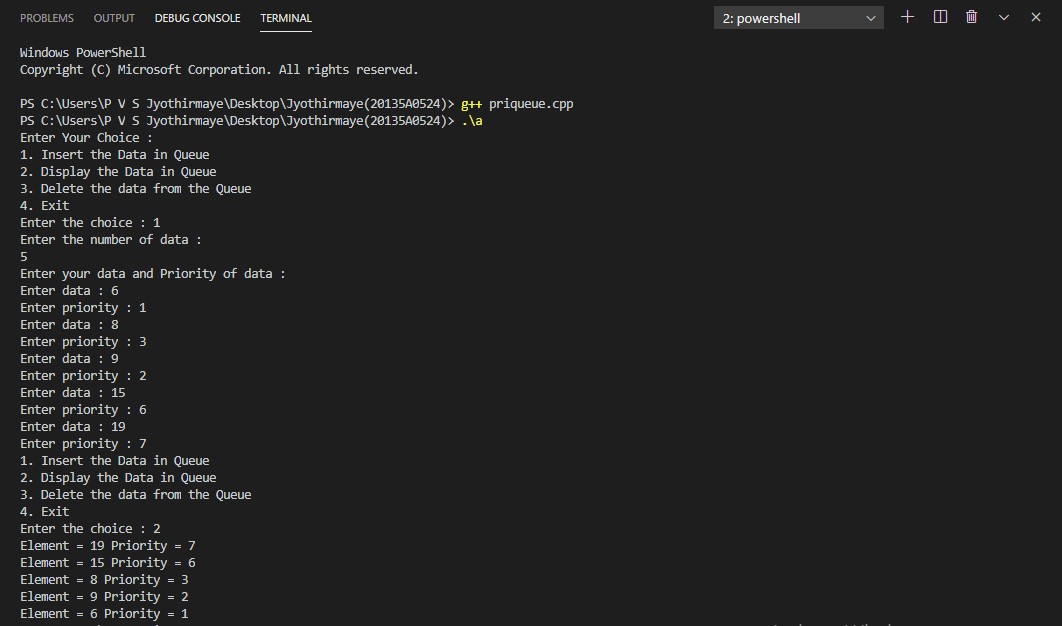
cout<<"Incorrect Choice"<<endl;

}

}while(opt!=4); return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement Circular queue . PROGRAM :

//Program to implement circular queue #include<iostream>

using namespace std; int cqueue[5];

int front=-1,rear=-1,n=5; void insertcq(int val)

{

if((front==0&&rear==n-1)||(front==rear+1))

{

cout<<"Queue overflow\n";

}

if(front==-1)

{

front=0; rear=0;

}

else

{

if(rear==n-1) rear=0;

else

rear=rear+1;

}

cqueue[rear]=val;

}

void deletecq()

{

if(front==-1)

{

cout<<"Queue underflow\n"<<endl; 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;

}

}

void displaycq()

{

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;

}

int main()

{

int ch, val; cout<<"1)Insert\n"; cout<<"2)Delete\n"; cout<<"3)Display\n"; cout<<"4)Exit\n";

do

{

cout<<"Enter choice : "<<endl; cin>>ch;

switch(ch)

{

case 1:

{

cout<<"Input for insertion : "<<endl; cin>>val;

insertcq(val); break;

}

case 2:

deletecq(); break;

case 3:

displaycq(); break;

case 4:

cout<<"Exit\n"; break;

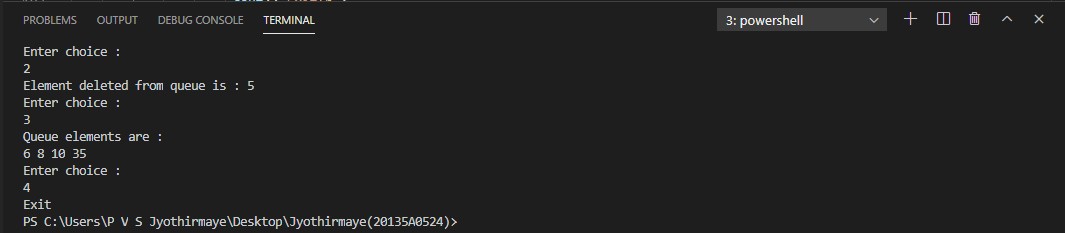
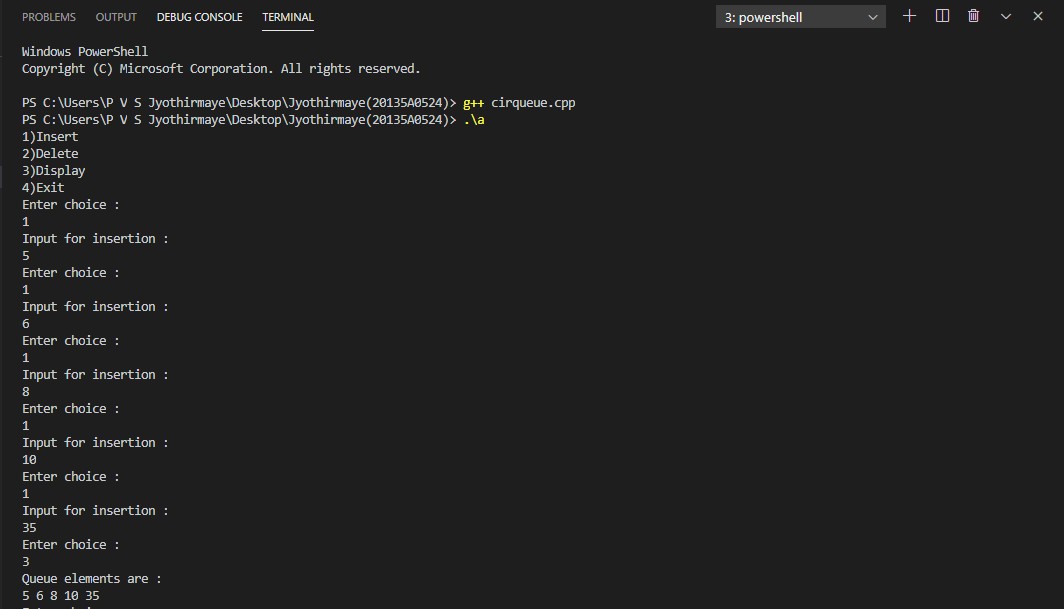
default: cout<<"Incorrect!\n";

}

} while(ch != 4); return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

# WEEK - 8

## Ex. No. 8 : Programs on Linked Lists . Date :

### AIM : Write a C++ program to implement : a.)Singly Linked List.

**b.)Doubly Linked List.**

**a.)C++ program to implement Singly linked list . PROGRAM :**

//C++ Program to Implement Singly Linked List #include<iostream>

#include<cstdio> #include<cstdlib> using namespace std; struct node

{

int info;

struct node \*next;

}\*start;

class single\_llist

{

public:

node\* create\_node(int); void insert\_begin();

void insert\_pos();

void insert\_last(); void delete\_pos(); void sort();

void search(); void update(); void reverse(); void display(); single\_llist()

{

start = NULL;

}

};

main()

{

int choice, nodes, element, position, i; single\_llist sl;

start = NULL; while (1)

{

cout<<endl<<"Operations on singly linked list"<<endl; cout<<endl<<" "<<endl; cout<<"1.Insert Node at beginning"<<endl; cout<<"2.Insert node at last"<<endl;

cout<<"3.Insert node at position"<<endl;

cout<<"4.Sort Link List"<<endl; cout<<"5.Delete a Particular Node"<<endl; cout<<"6.Update Node Value"<<endl; cout<<"7.Search Element"<<endl; cout<<"8.Display Linked List"<<endl; cout<<"9.Reverse Linked List "<<endl; cout<<"10.Exit "<<endl;

cout<<"Enter your choice : "; cin>>choice;

switch(choice)

{

case 1:

cout<<"Inserting Node at Beginning: "<<endl; sl.insert\_begin();

cout<<endl; break;

case 2:

cout<<"Inserting Node at Last: "<<endl; sl.insert\_last();

cout<<endl; break;

case 3:

cout<<"Inserting Node at a given position: "<<endl; sl.insert\_pos();

cout<<endl; break;

case 4:

cout<<"Sort Link List: "<<endl; sl.sort();

cout<<endl; break;

case 5:

cout<<"Delete a particular node: "<<endl; sl.delete\_pos();

break; case 6:

cout<<"Update Node Value: "<<endl; sl.update();

cout<<endl; break;

case 7:

cout<<"Search element in Link List: "<<endl; sl.search();

cout<<endl; break;

case 8:

cout<<"Display elements of link list"<<endl; sl.display();

cout<<endl; break;

case 9:

cout<<"Reverse elements of Link List"<<endl; sl.reverse();

cout<<endl; break;

case 10: cout<<"Exiting..."<<endl; exit(1);

break; default:

cout<<"Wrong choice"<<endl;

}

}

}

node \*single\_llist::create\_node(int value)

{

struct node \*temp, \*s; temp = new(struct node); if (temp == NULL)

{

cout<<"Memory not allocated "<<endl; return 0;

}

else

{

temp->info = value; temp->next = NULL; return temp;

}

}

void single\_llist::insert\_begin()

{

int value;

cout<<"Enter the value to be inserted: "; cin>>value;

struct node \*temp, \*p; temp = create\_node(value); if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

p = start; start = temp;

start->next = p;

}

cout<<"Element Inserted at beginning"<<endl;

}

void single\_llist::insert\_last()

{

int value;

cout<<"Enter the value to be inserted: "; cin>>value;

struct node \*temp, \*s; temp = create\_node(value); s = start;

while (s->next != NULL)

{

s = s->next;

}

temp->next = NULL; s->next = temp;

cout<<"Element Inserted at last"<<endl;

}

void single\_llist::insert\_pos()

{

int value, pos, counter = 0;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*s, \*ptr; temp = create\_node(value);

cout<<"Enter the postion at which node to be inserted: "; cin>>pos;

int i;

s = start;

while (s != NULL)

{

s = s->next; counter++;

}

if (pos == 1)

{

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

ptr = start; start = temp;

start->next = ptr;

}

}

else if (pos > 1 && pos <= counter)

{

s = start;

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

{

ptr = s;

s = s->next;

}

ptr->next = temp; temp->next = s;

}

else

{

cout<<"Positon out of range"<<endl;

}

}

void single\_llist::sort()

{

struct node \*ptr, \*s; int value;

if (start == NULL)

{

cout<<"The List is empty"<<endl; return;

}

ptr = start;

while (ptr != NULL)

{

for (s = ptr->next;s !=NULL;s = s->next)

{

if (ptr->info > s->info)

{

value = ptr->info; ptr->info = s->info; s->info = value;

}

}

ptr = ptr->next;

}

}

void single\_llist::delete\_pos()

{

int pos, i, counter = 0; if (start == NULL)

{

cout<<"List is empty"<<endl; return;

}

cout<<"Enter the position of value to be deleted: "; cin>>pos;

struct node \*s, \*ptr; s = start;

if (pos == 1)

{

start = s->next;

}

else

{

while (s != NULL)

{

s = s->next; counter++;

}

if (pos > 0 && pos <= counter)

{

s = start;

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

{

ptr = s;

s = s->next;

}

ptr->next = s->next;

}

else

{

cout<<"Position out of range"<<endl;

}

free(s);

cout<<"Element Deleted"<<endl;

}

}

void single\_llist::update()

{

int value, pos, i; if (start == NULL)

{

cout<<"List is empty"<<endl; return;

}

cout<<"Enter the node postion to be updated: "; cin>>pos;

cout<<"Enter the new value: "; cin>>value;

struct node \*s, \*ptr; s = start;

if (pos == 1)

{

start->info = value;

}

else

{

for (i = 0;i < pos - 1;i++)

{

if (s == NULL)

{

cout<<"There are less than "<<pos<<" elements"; return;

}

s = s->next;

}

s->info = value;

}

cout<<"Node Updated"<<endl;

}

void single\_llist::search()

{

int value, pos = 0;

bool flag = false; if (start == NULL)

{

cout<<"List is empty"<<endl; return;

}

cout<<"Enter the value to be searched: "; cin>>value;

struct node \*s; s = start;

while (s != NULL)

{

pos++;

if (s->info == value)

{

flag = true;

cout<<"Element "<<value<<" is found at position "<<pos<<endl;

}

s = s->next;

}

if (!flag)

cout<<"Element "<<value<<" not found in the list "<<endl;

}

void single\_llist::reverse()

{

struct node \*ptr1, \*ptr2, \*ptr3; if (start == NULL)

{

cout<<"List is empty"<<endl; return;

}

if (start->next == NULL)

{

return;

}

ptr1 = start;

ptr2 = ptr1->next; ptr3 = ptr2->next; ptr1->next = NULL; ptr2->next = ptr1; while (ptr3 != NULL)

{

ptr1 = ptr2; ptr2 = ptr3;

ptr3 = ptr3->next; ptr2->next = ptr1;

}

start = ptr2;

}

void single\_llist::display()

{

struct node \*temp; if (start == NULL)

{

cout<<"The List is Empty "<<endl; return;

}

temp = start;

cout<<"Elements of list are: "<<endl; while (temp != NULL)

{

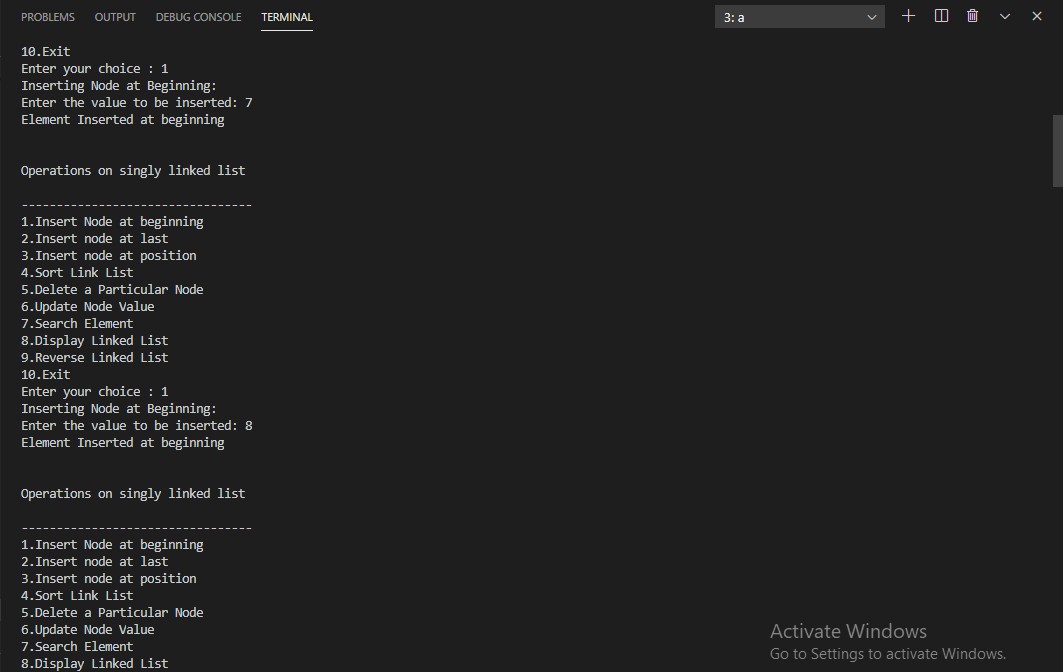
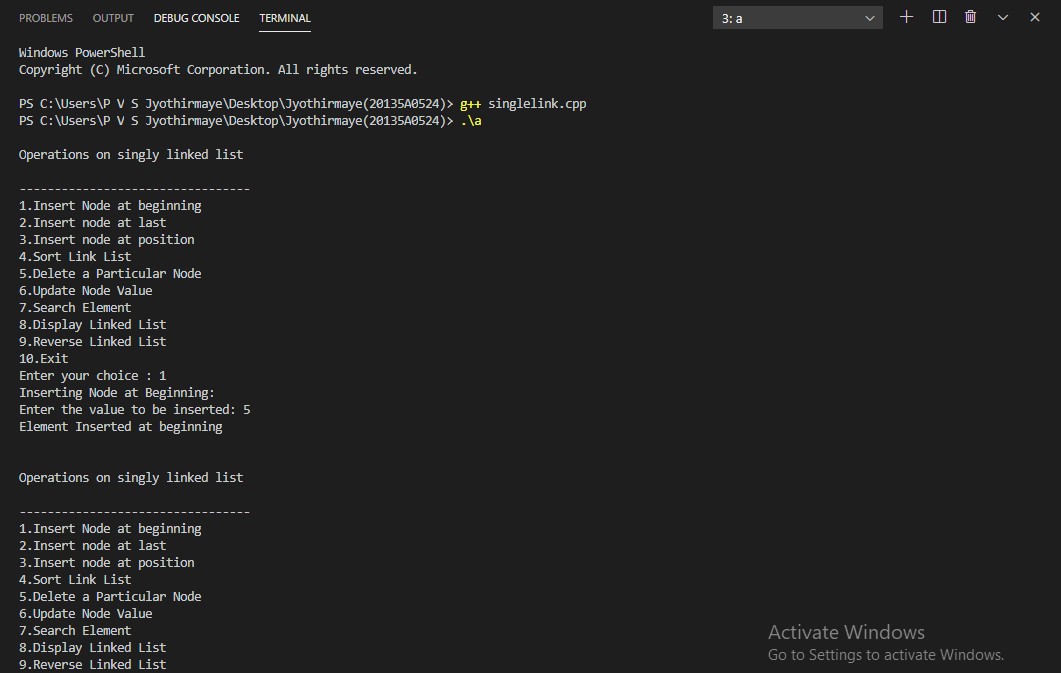
cout<<temp->info<<"->"; temp = temp->next;

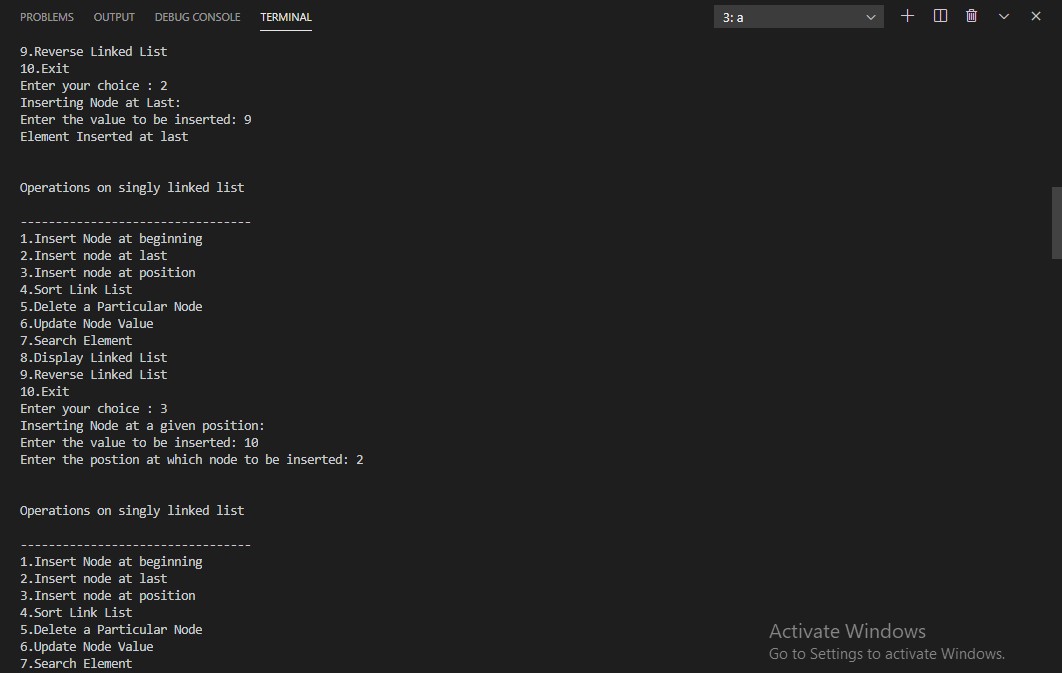
}

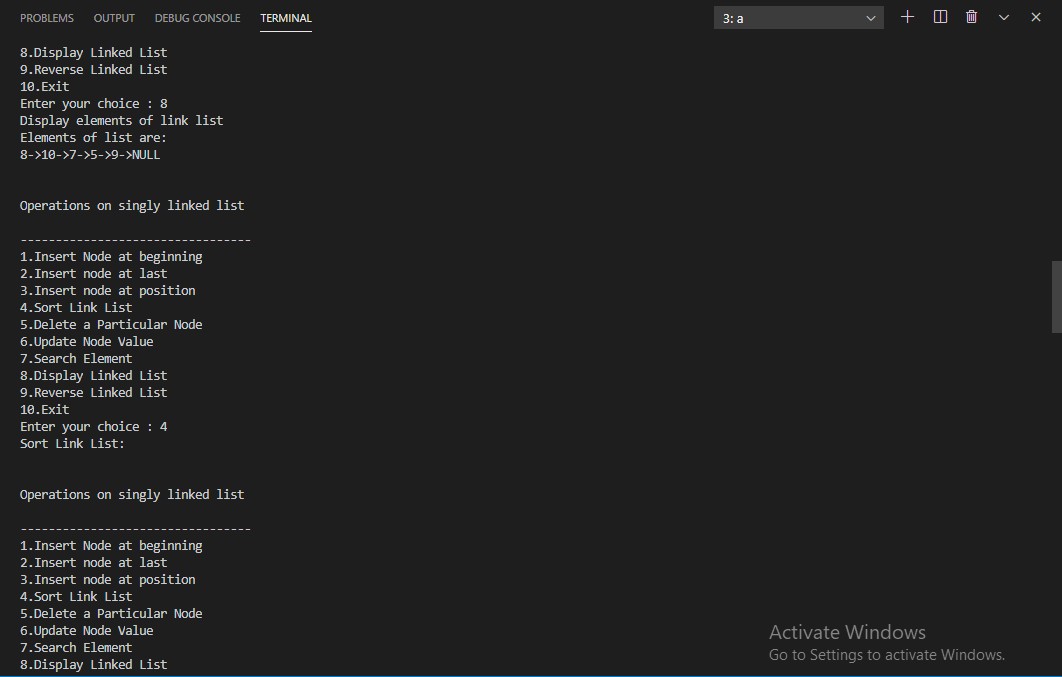
cout<<"NULL"<<endl;

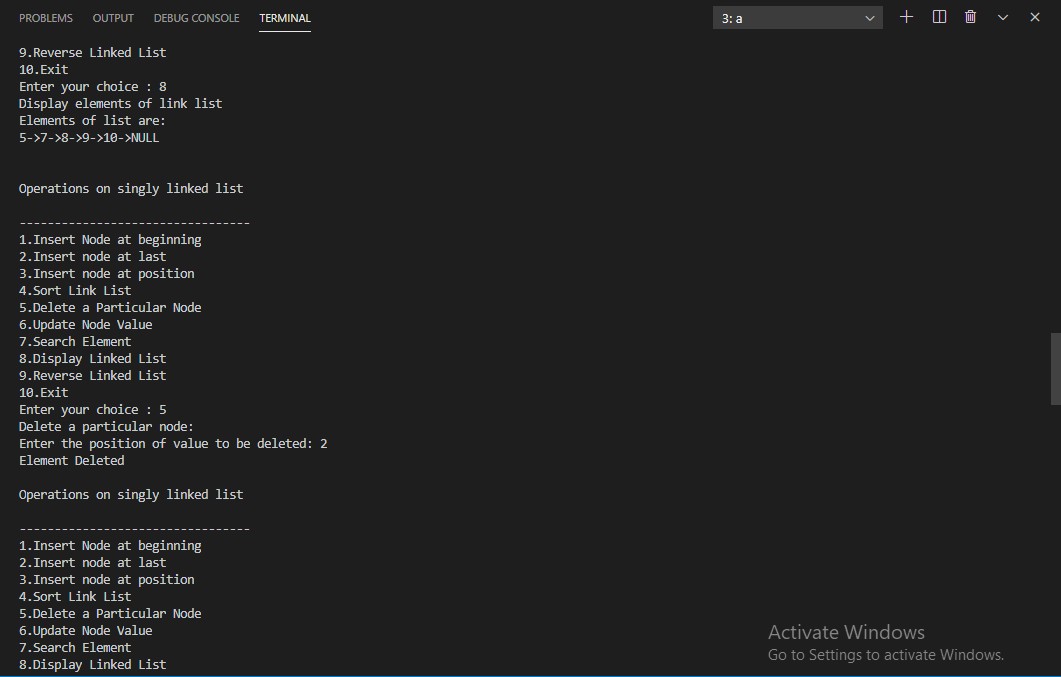
}

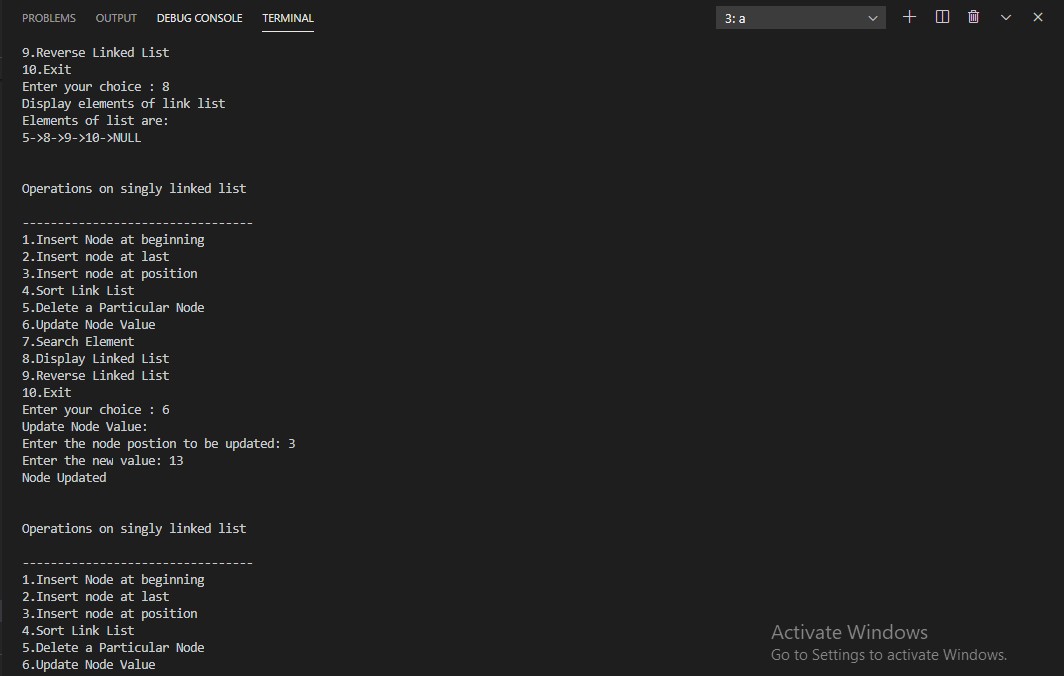
### OUTPUT :

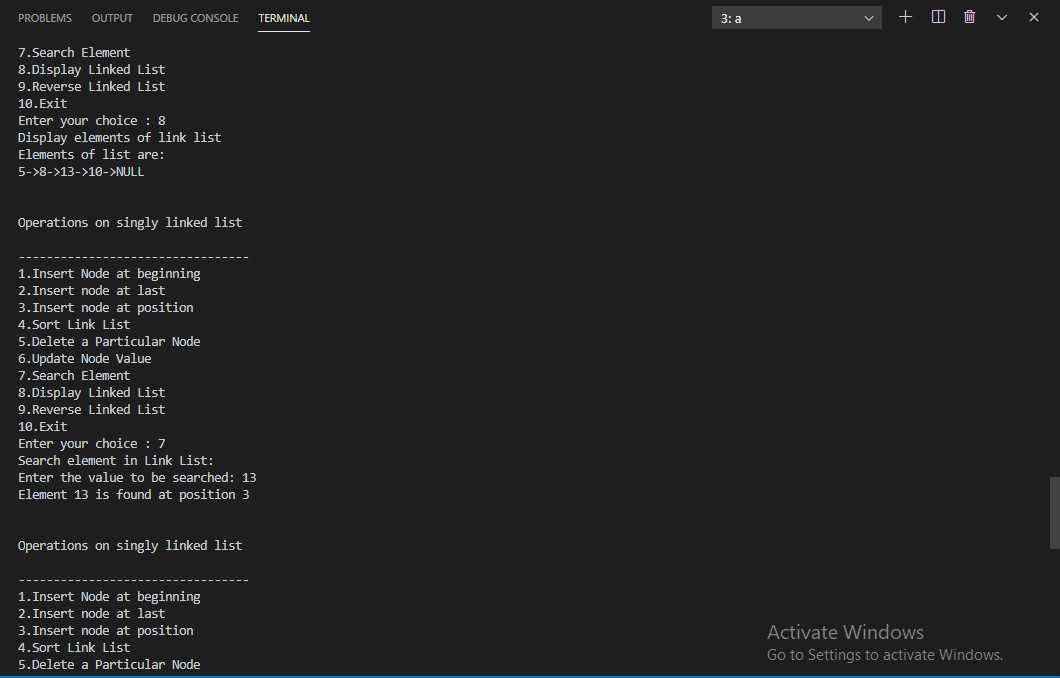


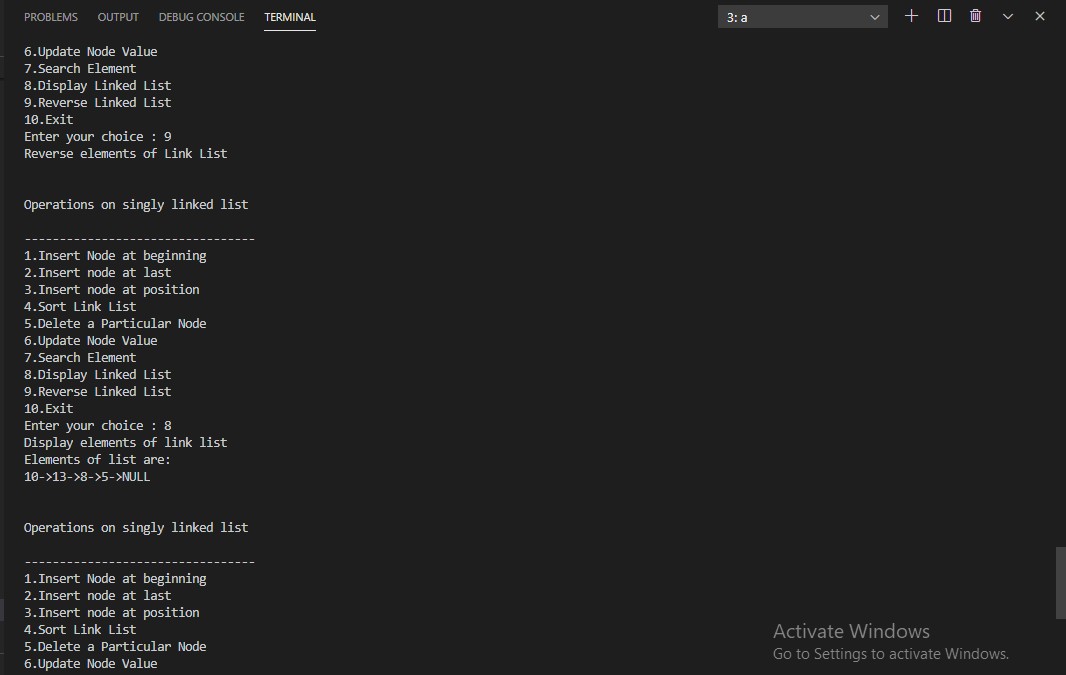


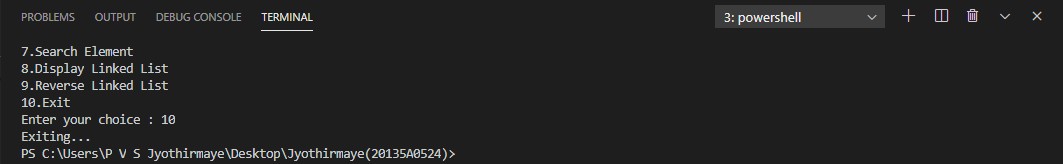












**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement Doubly linked list . PROGRAM :

#include<iostream> using namespace std; struct node

{

int value;

struct node\* next; struct node\* prev;

};

struct node\* head; struct node\* tail; void init()

{

head=NULL; tail=NULL;

}

void insertFirst(int element)

{

struct node\* newItem; newItem=new node; if(head==NULL)

{

head=newItem; newItem->prev=NULL;

newItem->value=element; newItem->next=NULL; tail=newItem;

}

else

{

newItem->next=head; newItem->value=element; newItem->prev=NULL; head->prev=newItem; head=newItem;

}

}

void insertLast(int element)

{

struct node\* newItem; newItem=new node;

newItem->value=element; if(head==NULL)

{

head=newItem; newItem->prev=NULL; newItem->next=NULL; tail=newItem;

}

else

{

newItem->prev=tail; tail->next=newItem; newItem->next=NULL; tail=newItem;

}

}

void insertAfter(int old, int element)

{

struct node\* newItem; newItem=new node; struct node\* temp; temp=head; if(head==NULL)

{

cout<<"could not insert"<<endl; return;

}

if(head==tail)

{

if(head->value!=old)

{

cout<<"could not insert"<<endl; return;

}

newItem->value=element; head->next=newItem; newItem->next=NULL; head->prev=NULL; newItem->prev=head; tail=newItem;

return;

}

if(tail->value==element)

{

newItem->next=NULL; newItem->prev=tail; tail->next=newItem; tail=newItem;

return;

}

while(temp->value!=old)

{

temp=temp->next; if(temp==NULL)

{

cout<<"Could not insert"<<endl; cout<<"element not found"<<endl; return;

}

}

newItem->next=temp->next; newItem->prev=temp; newItem->value=element; temp->next->prev=newItem; temp->next=newItem;

}

void deleteFirst()

{

if(head==NULL)

{

return;

}

if(head==tail)

{

struct node\* cur; cur=head; head=NULL; tail=NULL; delete cur; return;

}

else

{

struct node\* cur; cur=head; head=head->next; head->prev=NULL; delete cur;

}

}

void deleteLast()

{

if(head==NULL) return; if(head==tail)

{

struct node\* cur; cur=head; head=NULL; tail=NULL; delete cur; return;

}

else

{

struct node\* cur; cur=tail;

tail=tail->prev; tail->next=NULL; delete cur;

}

}

void deleteItem(int element)

{

struct node\* temp; temp=head; if(head==tail)

{

if(head->value!=element)

{

cout<<"could not delete"<<endl; return;

}

head=NULL; tail=NULL; delete temp; return;

}

if(head->value==element)

{

head=head->next; head->prev=NULL; delete temp; return;

}

else if(tail->value==element)

{

temp=tail; tail=tail->prev; tail->next=NULL; delete temp; return;

}

while(temp->value!=element)

{

temp=temp->next; if(temp==NULL)

{

cout<<"element not found"<<endl; return;

}

}

temp->next->prev=temp->prev; temp->prev->next=temp->next; delete temp;

}

struct node\* searchItem(int element)

{

struct node\* temp; temp=head; while(temp!=NULL)

{

if(temp->value==element)

{

return temp; break;

}

temp=temp->next;

}

return NULL;

}

void printList()

{

struct node\* temp; temp=head; while(temp!=NULL)

{

printf("%d->",temp->value); temp=temp->next;

}

puts("");

}

void printReverse()

{

struct node\* temp; temp=tail; while(temp!=NULL)

{

cout<<temp->value<<"->"; temp=temp->prev;

}

cout<<endl;

}

int main()

{

init();

int choice; while(1)

{

printf("1.InsertFirst 2. InsertLast 3. InsertAfter 4.DeleteFirst 5. DeleteLast 6. DeleteItem 7.

SearchItem 8. PrintList 9. PrintReverse 10. Exit \n"); cin>>choice;

if(choice==1)

{

int element; cout<<"Enter element : "; cin>>element; insertFirst(element); printList();

}

else if(choice==2)

{

int element; cout<<"Enter element : "; cin>>element; insertLast(element);

printList();

}

else if(choice==3)

{

int old,newitem; cout<<"Enter Old Item : "; cin>>old;

cout<<"Enter new Item : "; cin>>newitem; insertAfter(old,newitem); printList();

}

else if(choice==4)

{

deleteFirst(); printList();

}

else if(choice==5)

{

deleteLast(); printList();

}

else if(choice==6)

{

int element;

cout<<"Enter element to delete : "; cin>>element; deleteItem(element);

printList();

}

else if(choice==7)

{

int item;

cout<<"Enter Item to Search : "; cin>>item;

struct node\* ans=searchItem(item);

if(ans!=NULL) cout<<"FOUND "<<ans->value<<endl; else cout<<"NOT FOUND"<<endl;

}

else if(choice==8)

{

printList();

}

else if(choice==9)

{

printReverse();

}

else if(choice==10)

{

break;

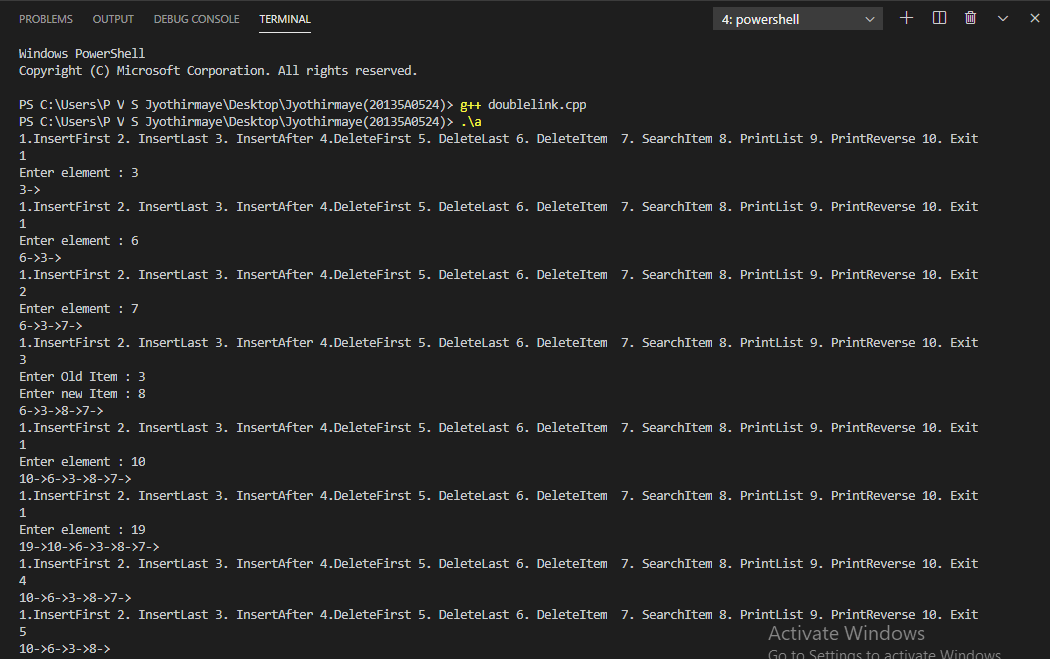
}

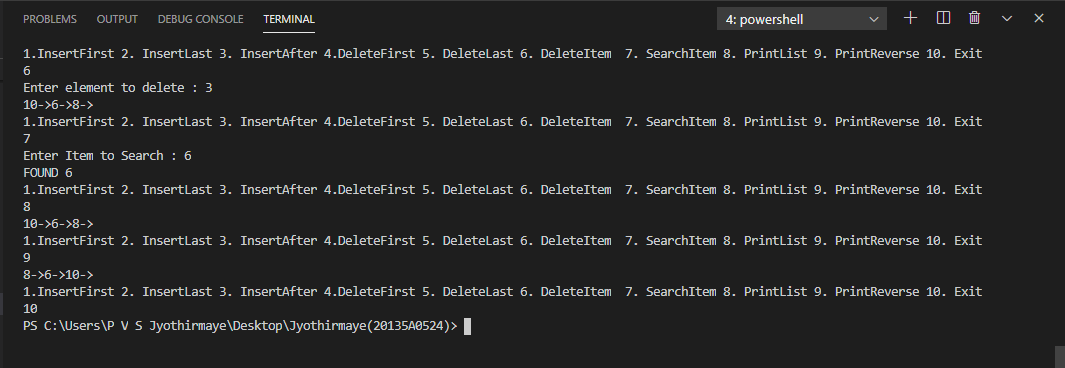
}

return 0;

}

### OUTPUT :





**RESULT :** The above program is executed successfully and verified .

# WEEK - 9

## Ex. No. 9 : Programs on Binary Tree . Date :

### AIM : Write a C++ program to implement : a.)Program to implement Binary Tree . b.)Inorder Traversal.

**c.)Postorder Traversal. d.)Preorder Traversal.**

**a.)C++ program to implement Binary Tree . PROGRAM :**

//Program to implement binary tree #include <iostream>

using namespace std; struct Node

{

int data;

Node\* left, \* right;

};

Node\* create(int data)

{

Node\* node = new Node ; node->data = data;

node->left = node->right = NULL; return (node);

}

Node\* insert(int arr[], Node\* root, int i, int n)

{

if (i < n)

{

Node\* temp = create(arr[i]); root = temp;

root->left = insert(arr,

root->left, 2 \* i + 1, n);

root->right = insert(arr, root->right, 2 \* i + 2, n);

}

return root;

}

void inorder(Node\* root)

{

if (root != NULL)

{

inorder(root->left);

cout << root->data <<" "; inorder(root->right);

}

}

int main()

{

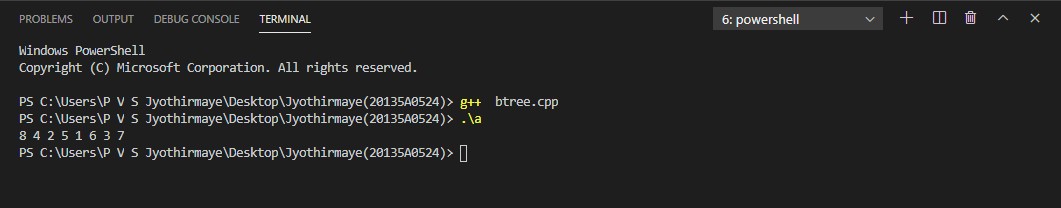
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8};

int n = sizeof(arr)/sizeof(arr[0]); Node\* root = insert(arr, root, 0, n); inorder(root);

return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to implement Inorder Traversal. PROGRAM :

//Program on inorder traversal #include <iostream>

using namespace std; struct Node

{

int data;

struct Node\* left, \*right; Node(int data)

{

this->data = data; left = right = NULL;

}

};

void printInorder(struct Node\* node)

{

if (node == NULL) return;

printInorder(node->left); cout << node->data << " "; printInorder(node->right);

}

int main()

{

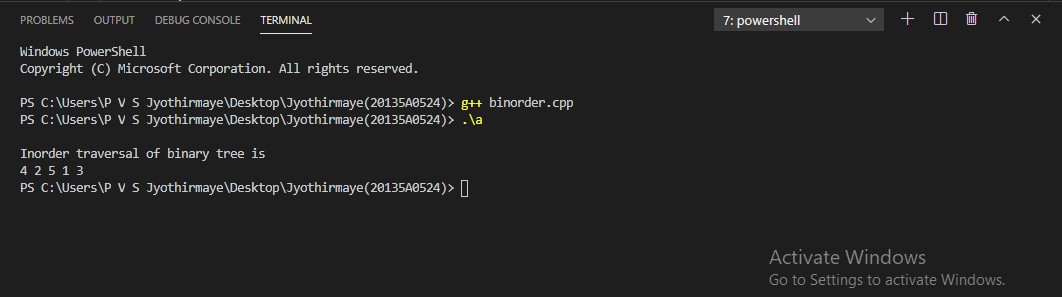
struct Node \*root = new Node(1); root->left = new Node(2); root->right = new Node(3); root->left->left = new Node(4); root->left->right = new Node(5);

cout << "\nInorder traversal of binary tree is \n"; printInorder(root);

return 0;

}

### OUTPUT :



**RESULT :** The above program is executed successfully and verified .

### c.)C++ program to implement Postorder Traversal. PROGRAM :

//Program on postorder traversal #include <iostream>

using namespace std; struct Node

{

int data;

struct Node\* left, \*right; Node(int data)

{

this->data = data; left = right = NULL;

}

};

void printPostorder(struct Node\* node)

{

if (node == NULL) return;

printPostorder(node->left); printPostorder(node->right); cout << node->data << " ";

}

int main()

{

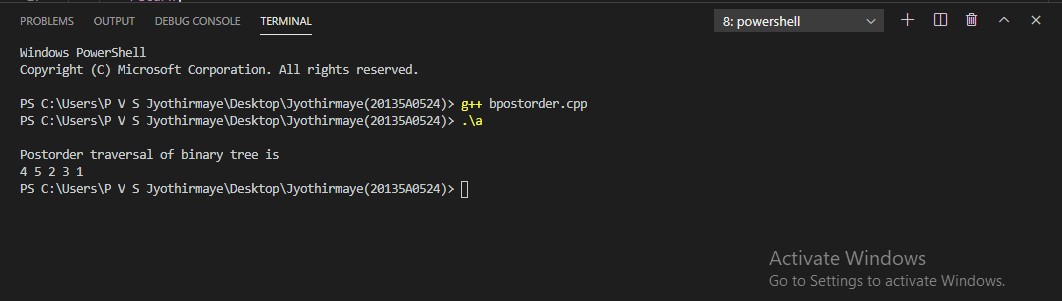
struct Node \*root = new Node(1); root->left = new Node(2); root->right = new Node(3); root->left->left = new Node(4); root->left->right = new Node(5);

cout << "\nPostorder traversal of binary tree is \n"; printPostorder(root);

return 0;

}

### OUTPUT :



.

### d.)C++ program to implement Preorder Traversal. PROGRAM :

//Program on preorder traversal #include <iostream>

using namespace std; struct Node

{

int data;

struct Node\* left, \*right; Node(int data)

{

this->data = data; left = right = NULL;

}

};

void printPreorder(struct Node\* node)

{

if (node == NULL) return;

cout << node->data << " "; printPreorder(node->left); printPreorder(node->right);

}

int main()

{

struct Node \*root = new Node(1); root->left = new Node(2); root->right = new Node(3); root->left->left = new Node(4); root->left->right = new Node(5);

cout << "\nPreorder traversal of binary tree is \n"; printPreorder(root);

return 0;

}

### OUTPUT :



# WEEK - 10

## Ex. No. 10 : Programs on Binary Search Tree . Date :

### AIM : Write a C++ program to implement :

**a.)Insert an element into Binary Search Tree . b.)Delete an element from Binary Search Tree . c.)Search for key element in a Binary Search Tree .**

**a.)C++ program to Insert an element into Binary Search Tree . PROGRAM :**

#include<iostream> using namespace std; template<class t> class bst

{

struct NODE

{

t data;

NODE\* left, \*right;

};

public:

NODE\* createnewnode(t data)

{

NODE\* createnewnode = new NODE(); createnewnode->data = data;

createnewnode->left = createnewnode->right = NULL; return createnewnode;

}

NODE\* Insert(NODE\* root,t data)

{

if(root == NULL)

{

root = createnewnode(data);

}

else if(data <= root->data)

{

root->left = Insert(root->left,data);

}

else

{

root->right = Insert(root->right,data);

}

return root;

}

void inorder(NODE\* root )

{

if(root != NULL)

{

inorder(root->left); cout<<root->data<<" "; inorder(root->right);

}

}

bst()

{

NODE\* root = NULL; t a,b,c,d,e,f;

cout<<endl<<"Enter the data"<<endl; cin>>a;

cin>>b; cin>>c; cin>>d; cin>>e; cin>>f;

root = Insert(root,a); root = Insert(root,b); root = Insert(root,c); root = Insert(root,d); root = Insert(root,e); root = Insert(root,f);

cout<<"\*\*\*THE BINARY SEARCH TREE\*\*\* "<<endl;

cout<<"Inorder:"; inorder(root); cout<<endl; insertion(root);

}

NODE\* insertion(NODE\* root)

{

t n;

cout<<"Enter the ELEMENT to be inserted in the BST"<<endl; cin>>n;

root = Insert(root,n);

cout<<"\*\*\*THE BINARY SEARCH TREE(2)\*\*\*"<<endl; cout<<"Inorder:";

inorder(root); return 0;

}

};

int main()

{

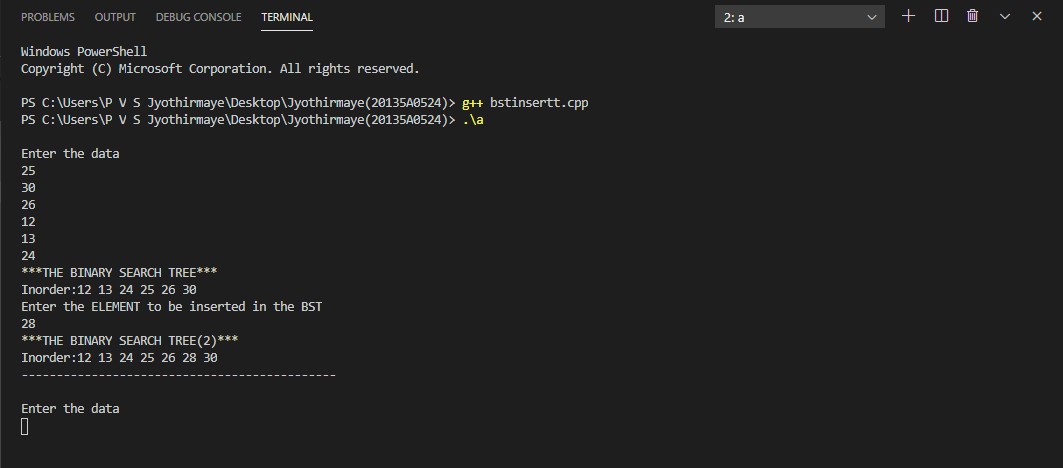
bst<int>a;

cout<<endl<<" "<<endl; bst<char>b;

return 0;

}

### OUTPUT :



### b.)C++ program to Delete an element from Binary Search Tree . PROGRAM :

#include<iostream> #include<stdio.h> #include<stdlib.h> using namespace std; struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

struct node \*temp = (struct node \*)malloc(sizeof(struct node)); temp->key = item;

temp->left = temp->right = NULL; return temp;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left); cout<< root->key; inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key)

{

if (node == NULL) return newNode(key); if (key < node->key)

node->left = insert(node->left, key); else

node->right = insert(node->right, key);

return node;

}

struct node \* minValueNode(struct node\* node)

{

struct node\* current = node;

while (current && current->left != NULL) current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key)

{

if (root == NULL) return root; if (key < root->key)

root->left = deleteNode(root->left, key); else if (key > root->key)

root->right = deleteNode(root->right, key); else

{

if (root->left == NULL)

{

struct node \*temp = root->right; free(root);

return temp;

}

else if (root->right == NULL)

{

struct node \*temp = root->left; free(root);

return temp;

}

struct node\* temp = minValueNode(root->right); root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main()

{

struct node \*root = NULL; root = insert(root, 50); root = insert(root, 30); root = insert(root, 20); root = insert(root, 40); root = insert(root, 70); root = insert(root, 60); root = insert(root, 80);

cout << "Inorder traversal of the given tree \n"; inorder(root);

cout<<"\nDelete 20\n";

root = deleteNode(root, 20);

cout<<"Inorder traversal of the modified tree \n"; inorder(root);

cout<<"\nDelete 30\n";

root = deleteNode(root, 30);

cout<<"Inorder traversal of the modified tree \n"; inorder(root);

cout<<"\nDelete 50\n";

root = deleteNode(root, 50);

cout<<"Inorder traversal of the modified tree \n"; inorder(root);

return 0;

}

### OUTPUT :



### c.)C++ program for Searching a key element in a Binary Search Tree . PROGRAM :

#include<iostream> using namespace std; struct node {

int d;

node \*left; node \*right;

};

node\* CreateNode(int d) { node \*newnode = new node; newnode->d = d;

newnode->left = NULL; newnode->right = NULL; return newnode;

}

node\* InsertIntoTree(node\* root, int d) { node \*temp = CreateNode(d);

node \*t = new node; t = root;

if(root == NULL) root = temp;

else {

while(t != NULL) {

if(t->d < d) {

if(t->right == NULL) { t->right = temp; break;

}

t = t->right;

} else if(t->d > d) { if(t->left == NULL) {

t->left = temp; break;

}

t = t->left;

}

}

}

return root;

}

void Search(node \*root, int d) { int depth = 0;

node \*temp = new node; temp = root;

while(temp != NULL) { depth++;

if(temp->d == d) {

cout<<"\nitem found at depth: "<<depth; return;

} else if(temp->d > d) temp = temp->left; else

temp = temp->right;

}

cout<<"\n item not found"; return;

}

int main() { char ch;

int n, i, a[10] = {93, 53, 45, 2, 7, 67, 32, 26, 71, 76};

node \*root = new node; root = NULL;

for (i = 0; i < 10; i++)

root = InsertIntoTree(root, a[i]); up:

cout<<"\nEnter the Element to be searched: "; cin>>n;

Search(root, n);

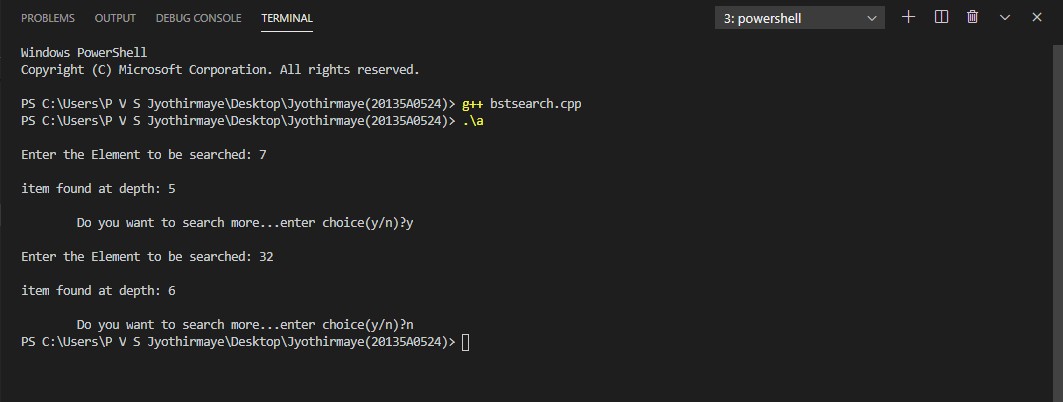
cout<<"\n\n\tDo you want to search more...enter choice(y/n)?"; cin>>ch;

if(ch == 'y' || ch == 'Y')

goto up; return 0;

}

### OUTPUT :



# WEEK - 11

## Ex. No. 11 : Programs on AVL Tree . Date :

### AIM : Write a C++ program to implement : a.)Insert an element into AVL Tree . b.)Delete an element from AVL Tree .

**a.)C++ program to Insert an element into AVL Tree . PROGRAM :**

#include<iostream> #include<cstdio> #include<sstream> #include<algorithm> #define pow2(n) (1 << (n)) using namespace std; struct avl\_node

{

int data;

struct avl\_node \*left; struct avl\_node \*right;

}\*root;

class avlTree

{

public:

int height(avl\_node \*);

int diff(avl\_node \*);

avl\_node \*rr\_rotation(avl\_node \*); avl\_node \*ll\_rotation(avl\_node \*); avl\_node \*lr\_rotation(avl\_node \*); avl\_node \*rl\_rotation(avl\_node \*); avl\_node\* balance(avl\_node \*); avl\_node\* insert(avl\_node \*, int ); void display(avl\_node \*, int); avlTree()

{

root = NULL;

}

};

int main()

{

int choice, item; avlTree avl; while (1)

{

cout<<"\n "<<endl;

cout<<"AVL Tree Implementation"<<endl; cout<<"\n "<<endl;

cout<<"1.Insert Element into the tree"<<endl; cout<<"2.Display Balanced AVL Tree"<<endl;

cout<<"3.Exit"<<endl; cout<<"Enter your Choice: "; cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter value to be inserted: "; cin>>item;

root = avl.insert(root, item); break;

case 2:

if (root == NULL)

{

cout<<"Tree is Empty"<<endl; continue;

}

cout<<"Balanced AVL Tree:"<<endl; avl.display(root, 1);

break; case 3:

exit(1); break;

default:

cout<<"Wrong Choice"<<endl;

}

}

return 0;

}

int avlTree::height(avl\_node \*temp)

{

int h = 0;

if (temp != NULL)

{

int l\_height = height (temp->left); int r\_height = height (temp->right);

int max\_height = max (l\_height, r\_height); h = max\_height + 1;

}

return h;

}

int avlTree::diff(avl\_node \*temp)

{

int l\_height = height (temp->left); int r\_height = height (temp->right); int b\_factor= l\_height - r\_height; return b\_factor;

}

avl\_node \*avlTree::rr\_rotation(avl\_node \*parent)

{

avl\_node \*temp; temp = parent->right;

parent->right = temp->left; temp->left = parent; return temp;

}

avl\_node \*avlTree::ll\_rotation(avl\_node \*parent)

{

avl\_node \*temp; temp = parent->left;

parent->left = temp->right; temp->right = parent; return temp;

}

avl\_node \*avlTree::lr\_rotation(avl\_node \*parent)

{

avl\_node \*temp; temp = parent->left;

parent->left = rr\_rotation (temp); return ll\_rotation (parent);

}

avl\_node \*avlTree::rl\_rotation(avl\_node \*parent)

{

avl\_node \*temp; temp = parent->right;

parent->right = ll\_rotation (temp); return rr\_rotation (parent);

}

avl\_node \*avlTree::balance(avl\_node \*temp)

{

int bal\_factor = diff (temp); if (bal\_factor > 1)

{

if (diff (temp->left) > 0) temp = ll\_rotation (temp);

else

temp = lr\_rotation (temp);

}

else if (bal\_factor < -1)

{

if (diff (temp->right) > 0) temp = rl\_rotation (temp);

else

temp = rr\_rotation (temp);

}

return temp;

}

avl\_node \*avlTree::insert(avl\_node \*root, int value)

{

if (root == NULL)

{

root = new avl\_node; root->data = value; root->left = NULL; root->right = NULL; return root;

}

else if (value < root->data)

{

root->left = insert(root->left, value); root = balance (root);

}

else if (value >= root->data)

{

root->right = insert(root->right, value); root = balance (root);

}

return root;

}

void avlTree::display(avl\_node \*ptr, int level)

{

int i;

if (ptr!=NULL)

{

display(ptr->right, level + 1); printf("\n");

if (ptr == root) cout<<"Root -> ";

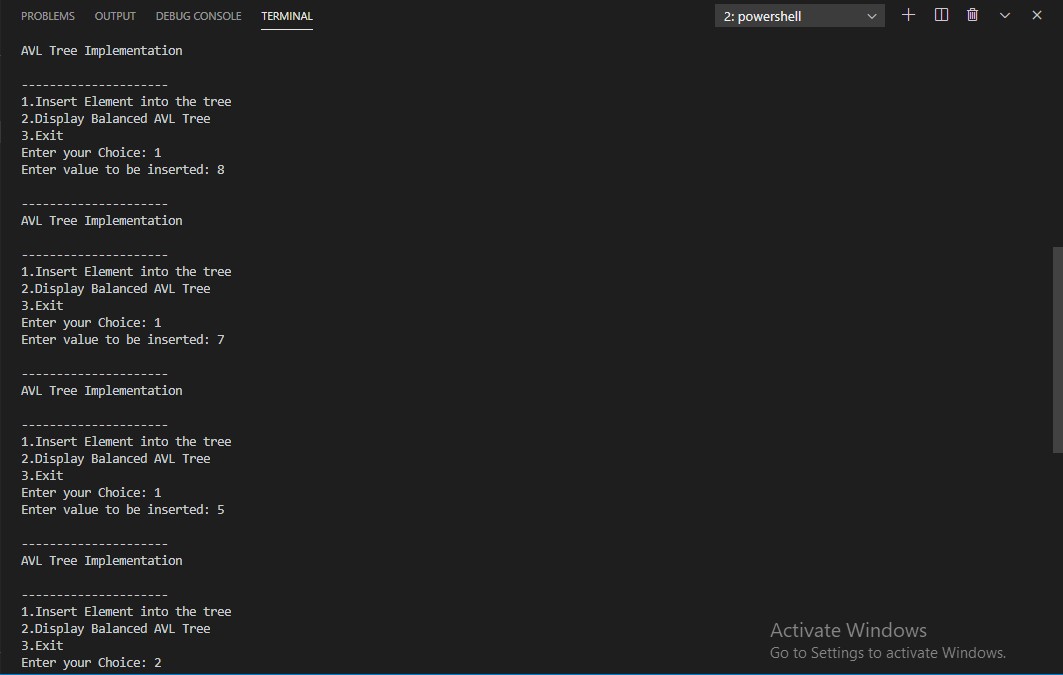
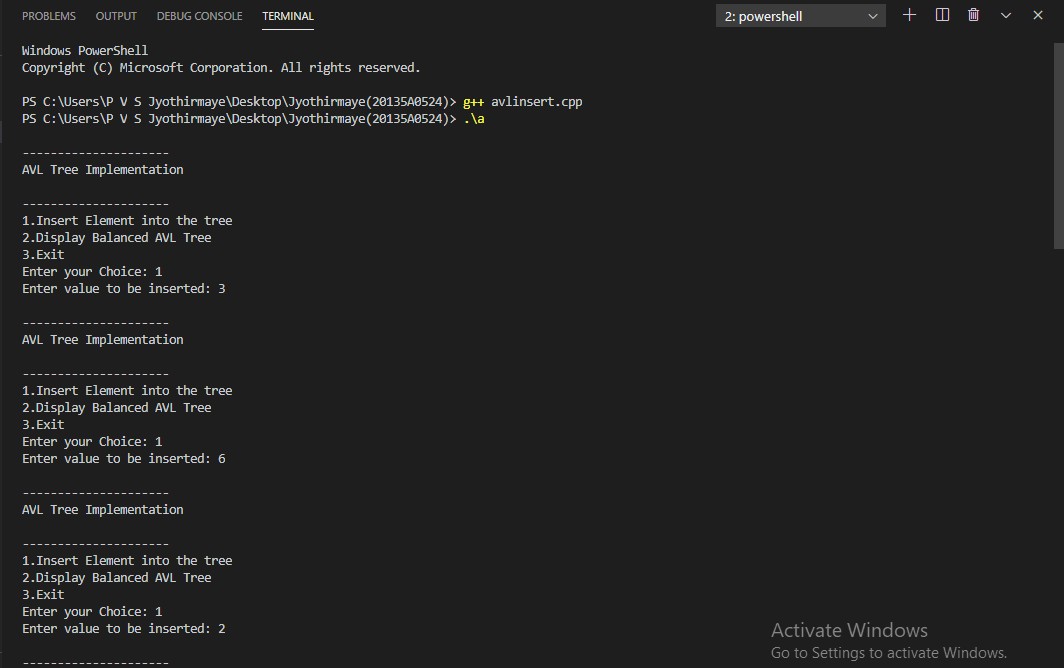
for (i = 0; i < level && ptr != root; i++) cout<<" ";

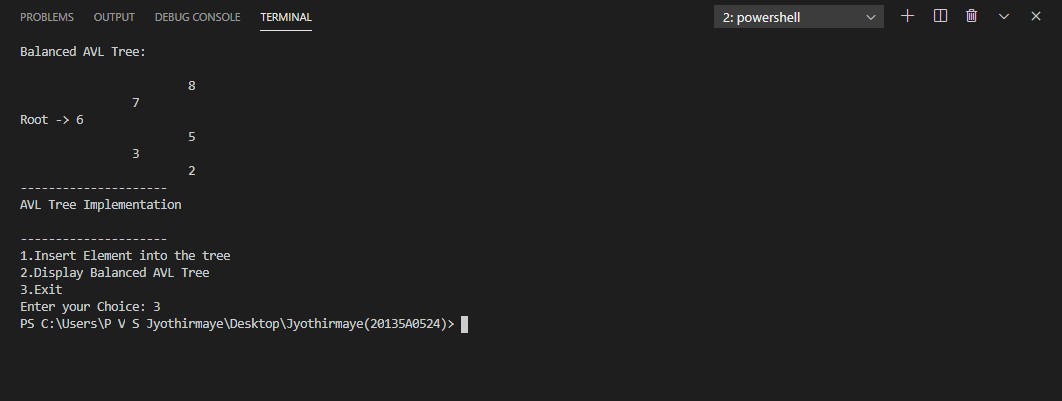
cout<<ptr->data; display(ptr->left, level + 1);

}

}

### OUTPUT :





**RESULT :** The above program is executed successfully and verified .

### b.)C++ program to Delete an element into AVL Tree . PROGRAM :

#include <iostream> #include <queue>

#include <unordered\_map> using namespace std; struct node {

struct node \*left; int data;

int height;

struct node \*right;

};

class AVL

{

private:

public:

struct node \* root; AVL(){

this->root = NULL;

}

int calheight(struct node \*p){

if(p->left && p->right){

if (p->left->height < p->right->height) return p->right->height + 1;

else return p->left->height + 1;

}

else if(p->left && p->right == NULL){ return p->left->height + 1;

}

else if(p->left ==NULL && p->right){ return p->right->height + 1;

}

return 0;

}

int bf(struct node \*n){

if(n->left && n->right){

return n->left->height - n->right->height;

}

else if(n->left && n->right == NULL){ return n->left->height;

}

else if(n->left== NULL && n->right ){ return -n->right->height;

}

return 0;

}

struct node \* llrotation(struct node \*n){ struct node \*p;

struct node \*tp; p = n;

tp = p->left;

p->left = tp->right; tp->right = p; return tp;

}

struct node \* rrrotation(struct node \*n){ struct node \*p;

struct node \*tp; p = n;

tp = p->right;

p->right = tp->left; tp->left = p;

return tp;

}

struct node \* rlrotation(struct node \*n){ struct node \*p;

struct node \*tp; struct node \*tp2; p = n;

tp = p->right;

tp2 =p->right->left; p -> right = tp2->left;

tp ->left = tp2->right; tp2 ->left = p;

tp2->right = tp; return tp2;

}

struct node \* lrrotation(struct node \*n){

struct node \*p; struct node \*tp; struct node \*tp2; p = n;

tp = p->left;

tp2 =p->left->right; p -> left = tp2->right;

tp ->right = tp2->left; tp2 ->right = p;

tp2->left = tp; return tp2;

}

struct node\* insert(struct node \*r,int data){

if(r==NULL){

struct node \*n;

n = new struct node; n->data = data;

r = n;

r->left = r->right = NULL; r->height = 1;

return r;

}

else{

if(data < r->data)

r->left = insert(r->left,data); else

r->right = insert(r->right,data);

}

r->height = calheight(r);

if(bf(r)==2 && bf(r->left)==1){ r = llrotation(r);

}

else if(bf(r)==-2 && bf(r->right)==-1){ r = rrrotation(r);

}

else if(bf(r)==-2 && bf(r->right)==1){ r = rlrotation(r);

}

else if(bf(r)==2 && bf(r->left)==-1){ r = lrrotation(r);

}

return r;

}

void levelorder\_newline(){ if (this->root == NULL){

cout<<"\n"<<"Empty tree"<<"\n"; return;

}

levelorder\_newline(this->root);

}

void levelorder\_newline(struct node \*v){ queue <struct node \*> q;

struct node \*cur; q.push(v);

q.push(NULL); while(!q.empty()){

cur = q.front(); q.pop();

if(cur == NULL && q.size()!=0){ cout<<"\n";

q.push(NULL); continue;

}

if(cur!=NULL){

cout<<" "<<cur->data;

if (cur->left!=NULL){ q.push(cur->left);

}

if (cur->right!=NULL){ q.push(cur->right);

}

}

}

}

struct node \* deleteNode(struct node \*p,int data){

if(p->left == NULL && p->right == NULL){ if(p==this->root)

this->root = NULL; delete p;

return NULL;

}

struct node \*t; struct node \*q; if(p->data < data){

p->right = deleteNode(p->right,data);

}

else if(p->data > data){

p->left = deleteNode(p->left,data);

}

else{

if(p->left != NULL){ q = inpre(p->left);

p->data = q->data;

p->left=deleteNode(p->left,q->data);

}

else{

q = insuc(p->right); p->data = q->data;

p->right = deleteNode(p->right,q->data);

}

}

if(bf(p)==2 && bf(p->left)==1){ p = llrotation(p); }

else if(bf(p)==2 && bf(p->left)==-1){ p = lrrotation(p); } else if(bf(p)==2 && bf(p->left)==0){ p = llrotation(p); } else if(bf(p)==-2 && bf(p->right)==-1){ p = rrrotation(p); } else if(bf(p)==-2 && bf(p->right)==1){ p = rlrotation(p); } else if(bf(p)==-2 && bf(p->right)==0){ p = llrotation(p); } return p;

}

struct node\* inpre(struct node\* p){ while(p->right!=NULL)

p = p->right; return p;

}

struct node\* insuc(struct node\* p){ while(p->left!=NULL)

p = p->left; return p;

}

~AVL(){

}

};

int main(){ AVL b;

int c,x; do{

cout<<"\n1.Display levelorder on newline"; cout<<"\n2.Insert";

cout<<"\n3.Delete\n"; cout<<"\n0.Exit\n"; cout<<"\nChoice: "; cin>>c;

switch (c)

{

case 1:

b.levelorder\_newline(); break;

case 2:

cout<<"\nEnter no. "; cin>>x;

b.root = b.insert(b.root,x); break;

case 3:

cout<<"\nWhat to delete? "; cin>>x;

b.root = b.deleteNode(b.root,x); break;

case 0:

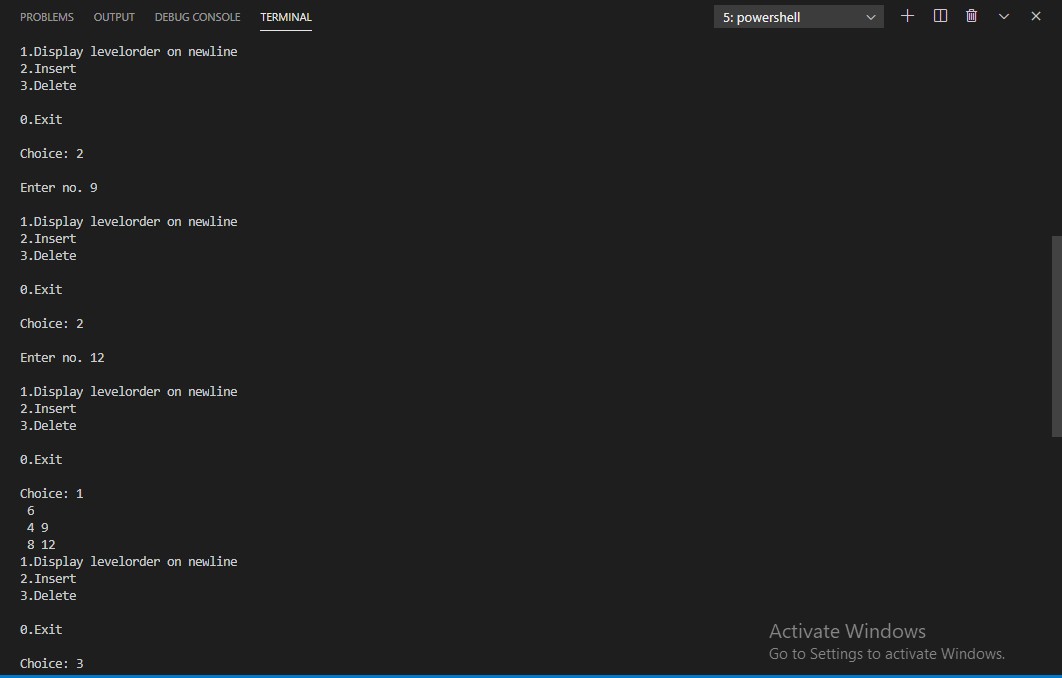
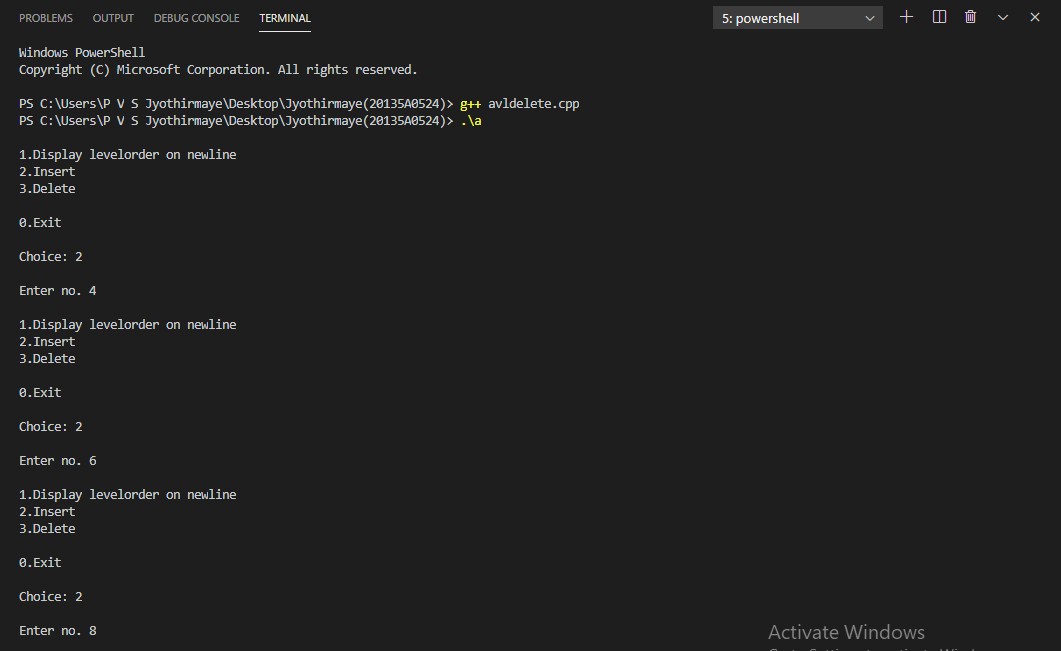
break;

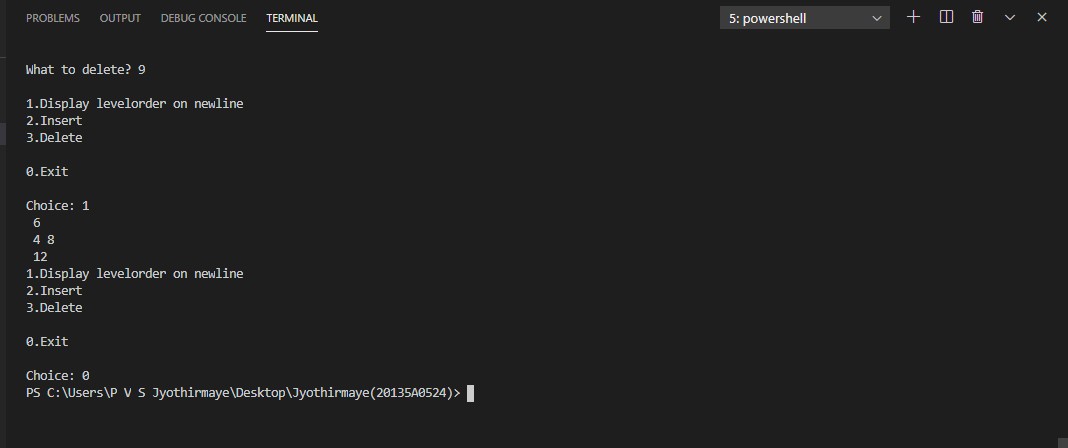
}

} while(c!=0);

}

### OUTPUT :





# WEEK - 12

## Ex. No. 12 : Programs on Graph . Date :

### AIM : Write a C++ program to implement : a.)Breath First Search.

**b.)Depth First Search .**

**a.)C++ program to implement Breath First Search in Graph. PROGRAM :**

//Program on BFS #include <iostream> #include <list>

using namespace std; class Graph {

int numVertices; list<int>\* adjLists; bool\* visited; public:

Graph(int vertices);

void addEdge(int src, int dest); void BFS(int startVertex);

};

Graph::Graph(int vertices) { numVertices = vertices;

adjLists = new list<int>[vertices];

}

void Graph::addEdge(int src, int dest) { adjLists[src].push\_back(dest); adjLists[dest].push\_back(src);

}

void Graph::BFS(int startVertex) { visited = new bool[numVertices]; for (int i = 0; i < numVertices; i++) visited[i] = false;

list<int> queue;

visited[startVertex] = true; queue.push\_back(startVertex);

list<int>::iterator i;

while (!queue.empty()) {

int currVertex = queue.front();

cout << "Visited " << currVertex << " "; queue.pop\_front();

for (i = adjLists[currVertex].begin(); i != adjLists[currVertex].end(); ++i) { int adjVertex = \*i;

if (!visited[adjVertex]) { visited[adjVertex] = true; queue.push\_back(adjVertex);

}

}

}

}

int main() { Graph g(4); g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

g.BFS(2);

return 0;

}

### C:\Users\P V S Jyothirmaye\Desktop\bfs11111.PNGOUTPUT :

### b.)C++ program to implement Depth First Search in Graph. PROGRAM :

//Program on DFS #include <iostream> #include <list>

using namespace std; class Graph {

int numVertices; list<int> \*adjLists; bool \*visited; public:

Graph(int V);

void addEdge(int src, int dest); void DFS(int vertex);

};

Graph::Graph(int vertices) {

numVertices = vertices;

adjLists = new list<int>[vertices]; visited = new bool[vertices];

}

void Graph::addEdge(int src, int dest) { adjLists[src].push\_front(dest);

}

void Graph::DFS(int vertex) { visited[vertex] = true;

list<int> adjList = adjLists[vertex]; cout << vertex << " "; list<int>::iterator i;

for (i = adjList.begin(); i != adjList.end(); ++i) if (!visited[\*i])

DFS(\*i);

}

int main() { Graph g(4); g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 3);

g.DFS(2);

return 0;

}

### OUTPUT :

