INDEX

Q1) Write a program to search an element from a list. Give user the option to perform Linear or Binary search. Use Template functions.

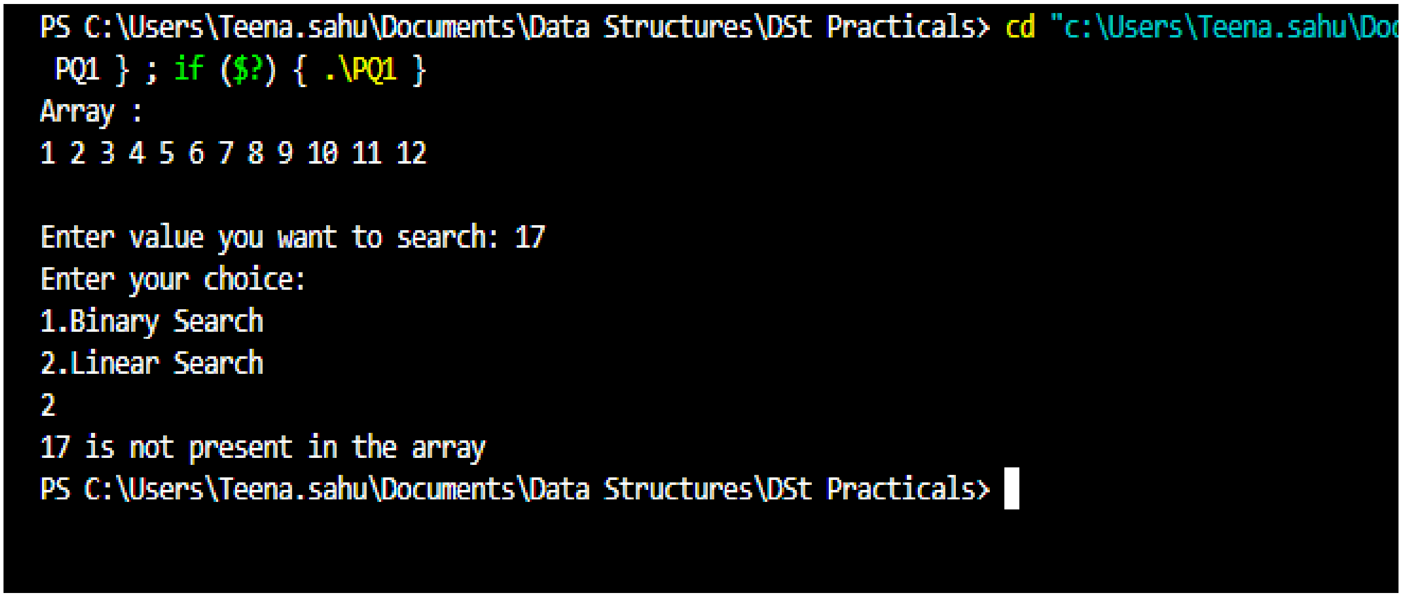
Code:

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

// binary search function using template
// n: size of arr
// x: value to search
// the fucntion returns -1 if x is not found in arr
// otherwise it returns index of x
template<typename T>
int LinearSearch(T arr[], int n, T x) {

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

                if (arr[i] == x)
                        return i;

        }

        return -1;

}
template<typename T1>
int binary_search(T1 arr[],int n,T1 x)
{
        int start = 0;
        int end = n-1;
        while(start<=end)
        {
                int mid = (start+end)/2;
                if(arr[mid]==x)
                        return mid;
                else if(arr[mid]<x)
```

```
                                start = mid + 1;
                        else
                                end = mid - 1;
                }
                return -1;
}

// Template function to print array
// n: size of arr
template<typename T>
void PrintArray(T arr[], int n)
{
   for (int i = 0; i < n; ++i)
      cout << arr[i] << " ";
   cout << "\n\n";
}

int main()
{

   int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ,12 };
   int n = sizeof(arr) / sizeof(int);

   cout << "Array : " << endl;
   PrintArray(arr, n);

   int x, index;
   cout<<"Enter value you want to search: ";
   cin>>x;

   int ch;
   cout<<"Enter your choice: "<<endl<<"1.Binary Search"<<endl<<"2.Linear Search"<<endl;
   cin>>ch;

   if (ch==1)
      index = binary_search(arr, n, x);
```

```
    else if(ch==2)
      index = LinearSearch(arr, n, x);
    else
      cout<<"Wrong Choice"<<endl;



    if(index==-1)
          cout<<x<<" is not present in the array"<<endl;
    else
          cout<<x<<" is present in the array at position "<<index<<endl;

}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd "c:\Users\Teena.sahu\Doc
 PQ1 } ; if ($?) { .\PQ1 }
Array :
1 2 3 4 5 6 7 8 9 10 11 12

Enter value you want to search: 17
Enter your choice:
1.Binary Search
2.Linear Search
2
17 is not present in the array
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

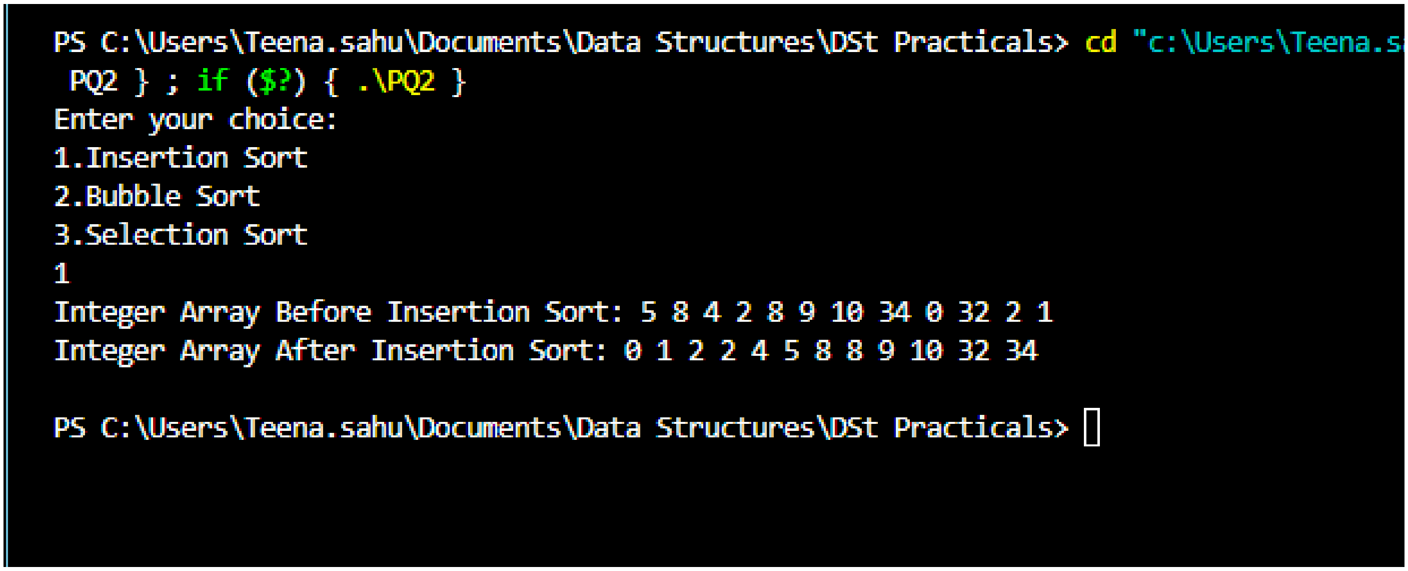
Q2) WAP using templates to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

Code:

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

// insertion sort template function
// to sort array in ascending order
// n is the size of array
template <class T>
void InsertionSort(T arr[], int n)
{
    int i, j;
    T temp;

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

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

        while (j >= 0 && arr[j] > temp)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }

        arr[j + 1] = temp;

    }
}
template<typename T>
void BubbleSort(T arr[], int n)
{
    for(int i=0;i<n-1;++i){
        for(int j=0;j<n-i-1;++j){
            if(arr[j]>arr[j+1]){
                T temp = arr[j+1];
                arr[j+1] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

```
template<typename T>
void SelectionSort(T arr[], int n)
{
    T temp;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(arr[i]>arr[j]){
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
                }
            }
        }
    }



// template function to print array
// n: size of array
template <class T>
void PrintArray(T arr[], int n)
{
    for (int i = 0; i < n; ++i)
    {
        cout << arr[i] << ' ';
    }
    cout << endl;
}


int main()
{
    int intArray[] = { 5,8,4,2,8,9,10,34,0,32,2,1 };
    int n = sizeof(intArray) / sizeof(int);

    int ch;
    cout<<"Enter your choice: "<<endl<<"1.Insertion Sort"<<endl<<"2.Bubble Sort"<<endl<<"3.Selection
Sort"<<endl;
    cin>>ch;

    if (ch==1){
        cout << "Integer Array Before Insertion Sort: ";
        PrintArray(intArray, n);
        InsertionSort(intArray, n);
        cout << "Integer Array After Insertion Sort: ";
        PrintArray(intArray, n);
    }
    else if (ch==2){
        cout << "Integer Array Before Bubble Sort: ";
        PrintArray(intArray, n);
        BubbleSort(intArray, n);
        cout << "Integer Array After Bubble Sort: ";
        PrintArray(intArray, n);
    }
    else if (ch==3){
        cout << "Integer Array Before Selection Sort: ";
        PrintArray(intArray, n);
        SelectionSort(intArray, n);
        cout << "Integer Array After Selection Sort: ";
        PrintArray(intArray, n);
    }
    else{
        cout<<"Wrong Choice!";
    }


    cout << "\n";



}
```

<u>Output:</u>



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd "c:\Users\Teena.s
 PQ2 } ; if ($?) { .\PQ2 }
Enter your choice:
1.Insertion Sort
2.Bubble Sort
3.Selection Sort
1
Integer Array Before Insertion Sort: 5 8 4 2 8 9 10 34 0 32 2 1
Integer Array After Insertion Sort: 0 1 2 2 4 5 8 8 9 10 32 34

PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> 
```

Q3) Write a program to implement singly linked list which supports the following operations:

(i) Insert an element x at the beginning of the singly linked list

(ii) Insert an element x at position in the singly linked list

(iii)Remove an element from the beginning of the singly linked list

(iv) Remove an element from position in the singly linked list.

(v) Search for an element x in the singly linked list and return its pointer

(vi) Concatenate two singly linked lists.

Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
   int data;
   Node *next;
};
void creat(Node *&head, int data)
{
   Node *p = new Node();
   p->data = data;
   p->next = NULL;
   head = p;
}
void display(Node *temp)
{
   cout << "The LINKED LIST is  : ";
   while (temp != NULL)
   {
      cout << temp->data << " ";
```

```
        temp = temp->next;
    }
    cout << "\n";
}
void insertatfront(Node **headrefrence, int data1)
{
    Node *new_node = new Node();
    new_node->data = data1;
    new_node->next = (*headrefrence);
    *headrefrence = new_node;
}
void addend(Node **headre, int data1)
{
    Node *nnode = new Node();
    Node *last = *headre;
    nnode->data = data1;
    nnode->next = NULL;
    if (*headre == NULL)
    {
        *headre = nnode;
    }
    while (last->next != NULL)
    {

        last = last->next;
    }
     last->next = nnode;
}
void insertatbet(Node *&head, int data1, int position)
{
```

```
Node *p2 = new Node();
p2 = head;
if (head == NULL)
{
    cout << "The given previous node cannot be NULL\n";
    return;
}
else
{

    int j = 1;
    {
    while (j < position && p2 != NULL)
        p2 = p2->next;
        j++;
    }
}
Node *p = new Node();
Node *p1 = new Node();

p = head;

Node *new_node = new Node();
new_node->data = data1;

if (position == 1)
{
    insertatfront(&head, data1);
    return;
}
```

```
    else if (p2 == NULL)
    {
        addend(&head, data1);
        return;
    }
    int i = 1;
    while (i < position - 1)
    {
        p = p->next;
        i++;
    }
    p1 = p->next;
    p->next = new_node;
    new_node->next = p1;
}
void deletefront(Node *&head)
{
    Node *p = new Node();
    Node *p1 = new Node();
    p = head;
    p1 = p;
    p1 = p1->next;
    head = p1;
    free(p);
}
void concat(Node *first, Node **second)
{
    Node *firstRef = first;

    // finding the lat node of first linked list
    while (firstRef->next != NULL)
    {
        firstRef = firstRef->next;
    }

    firstRef->next = *second;
}

// This function prints contents of
// linked list starting from head

void printList(Node *node)
{
    while (node != NULL)
    {
        cout << " " << node->data;
        node = node->next;
    }
}
void deleteend(Node *&head)
{
    Node *p = new Node();
    Node *p1 = new Node();
    if (head == NULL)
    {
        cout << "\nLinked list can not be empty\n";
        return;
    }
    p = head;
    while (p->next != NULL)
    {

        p1 = p;
        p = p->next;
    }
    free(p);
    p1->next = NULL;
}
void append(Node **head_ref, int new_data)
{

    // 1. allocate node
    Node *new_node = new Node();

    // used in step 5
    Node *last = *head_ref;

    // 2. put in the data
    new_node->data = new_data;

    // 3. This new node is going to be
    // the last node, so make next of
    // it as NULL
    new_node->next = NULL;

    // 4. If the Linked List is empty,
    // then make the new node as head
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }

    // 5. Else traverse till the last node
    while (last->next != NULL)
    {
        last = last->next;
    }

    // 6. Change the next of last node
    last->next = new_node;
    return;
}
void deletbyposition(Node *&head, int position)
{
    Node *p3 = new Node();
    p3 = head;
    if (head == NULL)
    {
        cout << "\nLinked list can not be empty\n";
        return;
    }
    // else if(position==1){
    //     deletefront(head);
    //     return;
    // }
    else
    {
        int j = 0;
        while (j < position && p3 != NULL)
        {
            p3 = p3->next;
            j++;
        }
    }
    if (position == 1)
    {
        deletefront(head);
        return;
    }
    else if (p3 == NULL)
    {
        deleteend(head);
        return;
    }

    Node *p = new Node();
    Node *p2 = new Node();
    Node *p1 = new Node();
    p = head;
    int i = 0;
    while (i < position - 1)
    {

        p1 = p;
        p = p->next;
        i++;
    }
    p2 = p->next;
```

```
    free(p);
    p1->next = p2;
}
void search(Node *&head, int data)
{
    Node *p = new Node();
    p = head;
    bool f = true;
    while (p != NULL && f)
    {
        if (p->data == data)
        {
            f = false;
        }
        else
        {
            p = p->next;
        }
    }
    if (f != true)
    {
        cout << "\n"
             << data << " is PRESENT at ADDRESS " << p << endl;
        ;
    }
    else
    {
        cout << "\n"
             << data << " is NOT PRESENT in this linked list\n";
    }
```

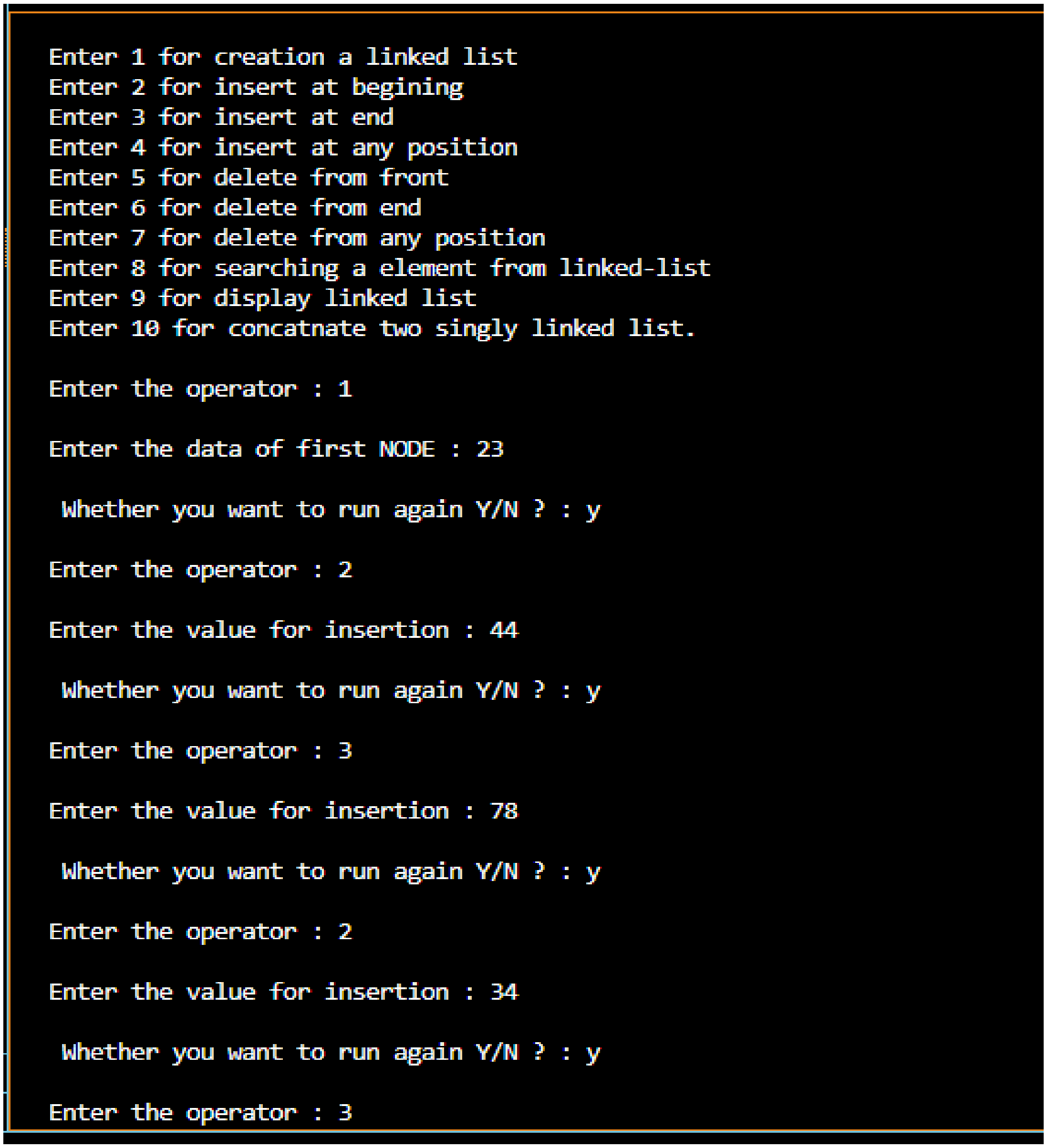
```
}
int main()
{
    Node *head;

    cout << "\nEnter 1 for creation a linked list "
         << "\nEnter 2 for insert at begining "
         << "\nEnter 3 for insert at end "
         << "\nEnter 4 for insert at any position "
         << "\nEnter 5 for delete from front "
         << "\nEnter 6 for delete from end "
         << "\nEnter 7 for delete from any position "
         << "\nEnter 8 for searching a element from linked-list "
         << "\nEnter 9 for display linked list "
         << "\nEnter 10 for concatnate two singly linked list." << endl;


    string s = "y";
    int op, x, y;
    while (s == "y" || s == "Y")
    {
        cout << "\nEnter the operator : ";
        cin >> op;
        if (op == 1)
        {
            cout << "\nEnter the data of first NODE : ";
            cin >> x;
            creat(head, x);
        }
        else if (op == 2)
```

```
{
   cout << "\nEnter the value for insertion : ";
   cin >> x;
   insertatfront(&head, x);
}
else if (op == 3)
{
   cout << "\nEnter the value for insertion : ";
   cin >> x;
   addend(&head, x);
}
else if (op == 4)
{
   cout << "\nEnter the value for insertion : ";
   cin >> x;
   cout << "\nEnter the position where you want to insert the value : ";
   cin >> y;
   insertatbet(head, x, y);
}
else if (op == 5)
{
   deletefront(head);
}
else if (op == 6)
{
   deleteend(head);
}
else if (op == 7)
{
   cout << "\nEnter the position of Node: ";
```

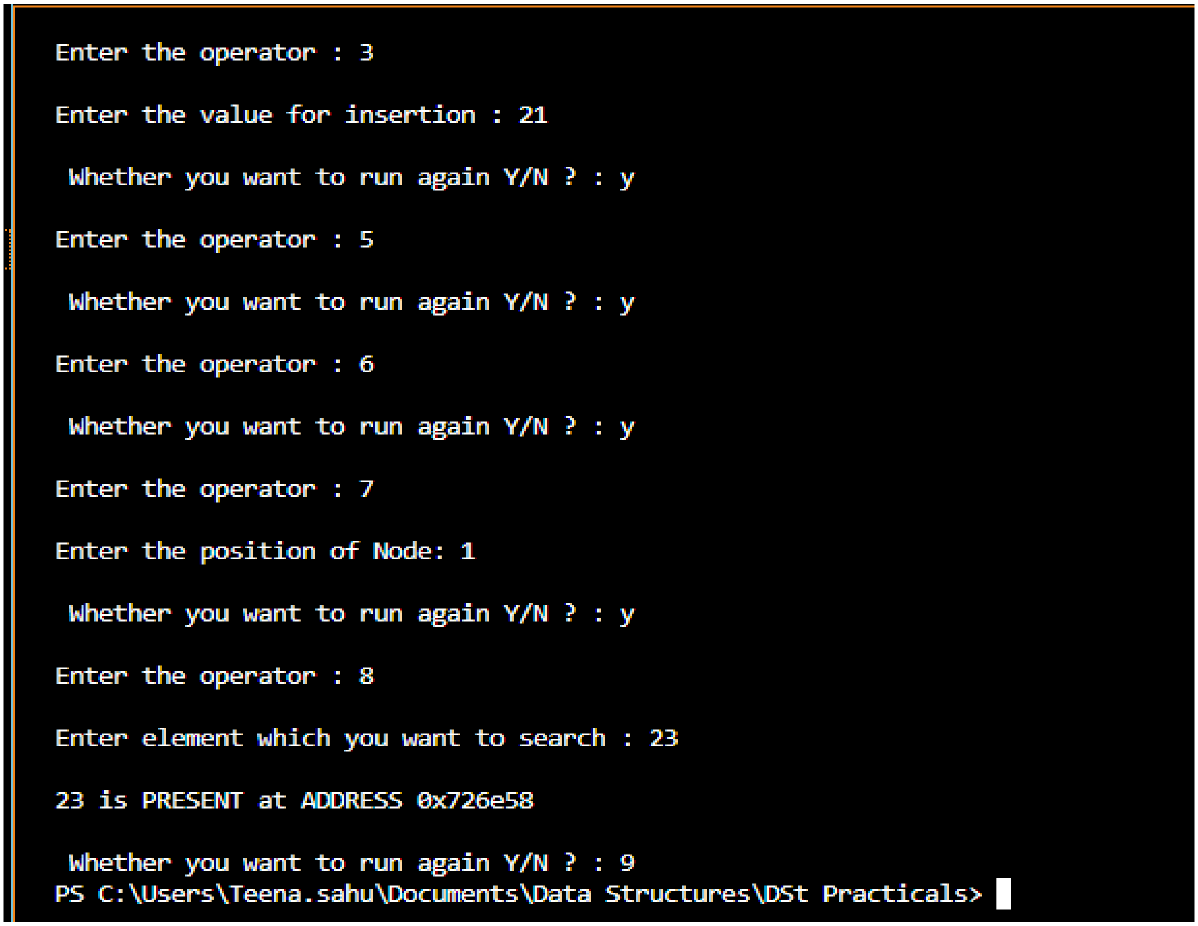
```
      cin >> y;

      deletbyposition(head, y);

   }

   else if (op == 8)

   {

      cout << "\nEnter element which you want to search : ";

      cin >> x;

      search(head, x);

   }

   else if (op == 9)

   {

      display(head);

   }

   else if (op==10)

   {

      Node *first = NULL;


// Insert 6. So linked list becomes 6->NULL

int n;int k;

cout<<"How many nodes you want in 1st linked list?  : ";

cin>>n;

cout<<"Enter node data :";

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

   cin>>k;

   append(&first, k);

}

cout<<"First linked list is : ";

printList(first);

cout<<endl;
```

```
  Node *second = NULL;
  int n1;int k1;
  cout<<"\nHow many nodes you want in 2nd linked list?  : ";
  cin>>n;
  cout<<"Enter node data : ";
  for(int i=0;i<n;i++){
    cin>>k;
    append(&second, k);
  }

  cout << "\nCreated Second Linked list is: ";
  printList(second);
  cout << "\nConcatinated list is: ";
  concat(first, &second);
  printList(first);
    }
    else
    {
      cout << "\n Invalid choice";
    }
    cout << "\n Whether you want to run again Y/N ? : ";
    cin >> s;
  }
  return 0;
}
```

Output:



```
Enter 1 for creation a linked list
Enter 2 for insert at begining
Enter 3 for insert at end
Enter 4 for insert at any position
Enter 5 for delete from front
Enter 6 for delete from end
Enter 7 for delete from any position
Enter 8 for searching a element from linked-list
Enter 9 for display linked list
Enter 10 for concatnate two singly linked list.

Enter the operator : 1

Enter the data of first NODE : 23

 Whether you want to run again Y/N ? : y

Enter the operator : 2

Enter the value for insertion : 44

 Whether you want to run again Y/N ? : y

Enter the operator : 3

Enter the value for insertion : 78

 Whether you want to run again Y/N ? : y

Enter the operator : 2

Enter the value for insertion : 34

 Whether you want to run again Y/N ? : y

Enter the operator : 3
```



```
Enter the operator : 3

Enter the value for insertion : 21

 Whether you want to run again Y/N ? : y

Enter the operator : 5

 Whether you want to run again Y/N ? : y

Enter the operator : 6

 Whether you want to run again Y/N ? : y

Enter the operator : 7

Enter the position of Node: 1

 Whether you want to run again Y/N ? : y

Enter the operator : 8

Enter element which you want to search : 23

23 is PRESENT at ADDRESS 0x726e58

 Whether you want to run again Y/N ? : 9
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

Q4) Write a program to implement doubly linked list which supports the following operations:

(i) Insert an element x at the beginning of the doubly linked list

(ii) Insert an element x at position in the doubly linked list

(iii)Insert an element x at the end of the doubly linked list

(iv) Remove an element from the beginning of the doubly linked list

(v) Remove an element from position in the doubly linked list.

(vi) Remove an element from the end of the doubly linked list

(vii) Search for an element x in the doubly linked list and return its pointer

(viii) Concatenate two doubly linked lists .

Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
   int data;
   Node *next;
   Node*pre;
};
void creat(Node *&head,int data){
   Node *p=new Node();
   p->data=data;
   p->next=NULL;
   p->pre=NULL;
   head=p;

}
void display(Node *temp)
```

```
{cout<<"The doubly linked list is  : ";
Node*p=new Node();
   while (temp != NULL)
   {
      cout << temp->data << " ";
      p=temp;
      temp = temp->next;
   }

   cout<<"\n";
   cout<<"\nLinked list using pre pointer is :  ";
      while (p != NULL)
   {
      cout << p->data << " ";
      p = p->pre;
   }
    cout<<"\n";
}

void insertatfront(Node**headrefrence,int data1){
   Node*p=new Node();
   p->data=data1;
   p->pre=NULL;
   p->next=(*headrefrence);
   (*headrefrence)->pre=p;
   *headrefrence=p;
}
void addend(Node** headre,int data1){
   Node* nnode=new Node();
   Node*last=*headre;
```

```
  nnode->data=data1;
  nnode->next=NULL;
  if(*headre==NULL){
    nnode->pre=NULL;
    *headre=nnode;
  }
  while(last->next!=NULL){


        last=last->next;

  }
  last->next=nnode;
  nnode->pre=last;

}
void insertatbet(Node*&head,int data1,int position){
  Node*p2=new Node();
  p2=head;
 if(head==NULL){
  cout<<"the given previous node cannot be NULL\n";
  return;
 }
 else{

  int j=1;
  while(j<position &&p2!=NULL){
    p2=p2->next;
    j++;
  }
  }



if(position==1){
    insertatfront(&head,data1);
    return;
}
else if(p2==NULL){
    addend(&head,data1);
    return;
}
  Node*p=new Node();

  Node*p1=new Node();

  p=head;

  Node* new_node=new Node();
  new_node->data=data1;
int i=0;
while(i<position-1){
    p1=p;
p=p->next;
i++;
}
p1->next=new_node;
new_node->pre=p1;
p->pre=new_node;
new_node->next=p;
```

```
}

void deletefront(Node *&head){
   Node *p=new Node();
   Node*p1=new Node();
   p=head;
   p1=p;
   p1=p1->next;
   p1->pre=NULL;
   head=p1;
   free(p);
}
void append(Node **head_ref, int new_data)
{
   // 1. allocate node
   Node *new_node = new Node();

   Node *last = *head_ref; // used in step 5

   // 2. put in the data
   new_node->data = new_data;

   // 3. This new node is going to be the last node, so
   // make next of it as NULL
   new_node->next = NULL;

   // 4. If the Linked List is empty, then make the new
   //  node as head
   if (*head_ref == NULL)
```

```
    {
        new_node->pre = NULL;
        *head_ref = new_node;
        return;
    }

    // 5. Else traverse till the last node
    while (last->next != NULL)
        last = last->next;

    // 6. Change the next of last node
    last->next = new_node;

    // 7. Make last node as previous of new node
    new_node->pre = last;

    return;
}
void printList(Node *node)
{
    cout << "\nTraversal in forward direction \n";
    while (node != NULL)
    {
        std::cout << " " << node->data << " ";
        node = node->next;
    }
}
void concat(Node *first, Node **second)
{
    Node *firstRef = first;
    // finding the lat node of first linked list
    while (firstRef->next != NULL)
    {
        firstRef = firstRef->next;
    }

    firstRef->next = *second;
}
void deleteend(Node *&head){
    Node *p=new Node();
    Node*p1=new Node();
if(head==NULL){
    cout<<"\nLinked list can not be empty\n";
    return;
}
p=head;

while(p->next!=NULL){

    p1=p;
    p=p->next;
}
free(p);
p1->next=NULL;

}
void deletbyposition(Node *&head,int position){
   Node *p3=new Node();
   p3=head;
```

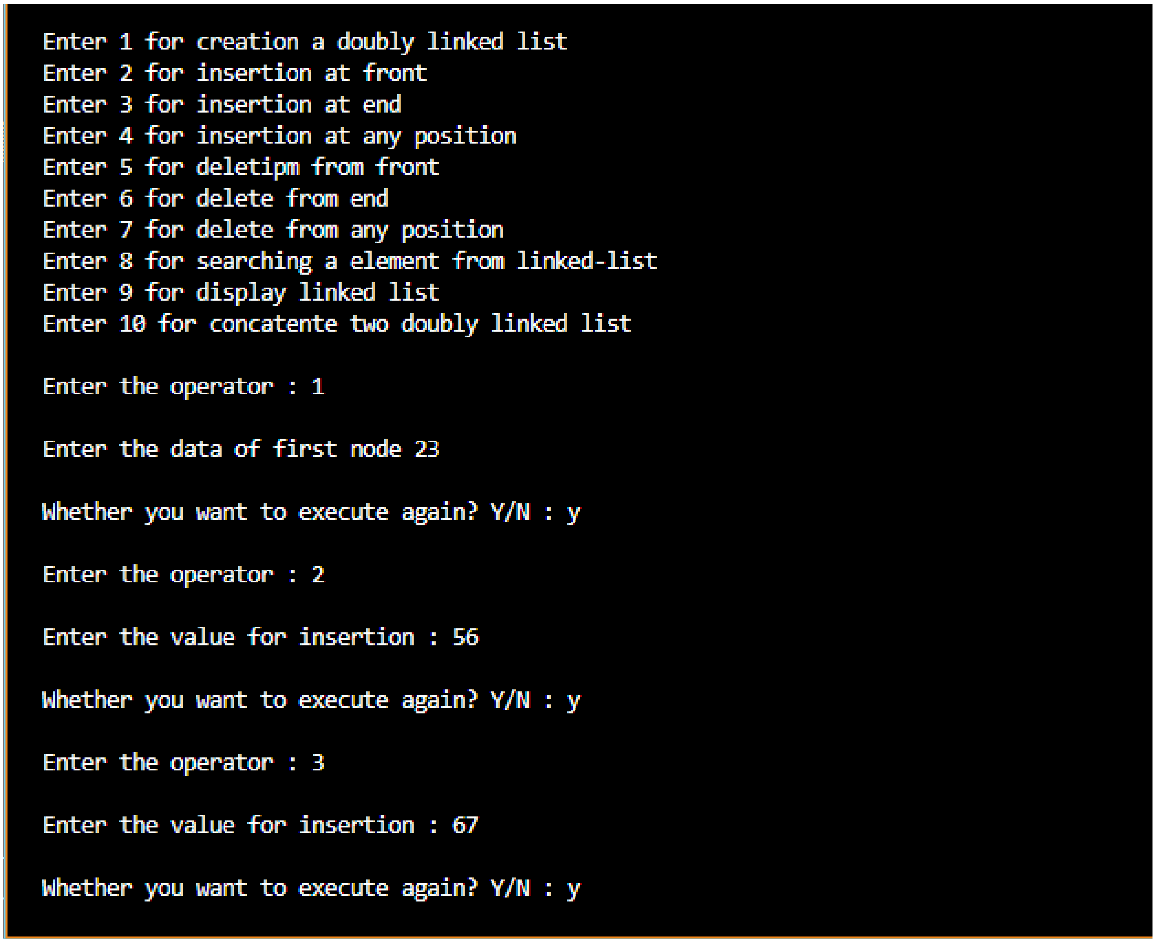
```
if(head==NULL){
   cout<<"\nLinked list can not be empty\n";
   return;
}
// else if(position==1){
//     deletefront(head);
//     return;
// }
else {
   int j=0;
   while(j<position && p3!=NULL){
      p3=p3->next;
      j++;
   }
}
if(position==1){
   deletefront(head);
   return;
}
else if(p3==NULL){
   deleteend(head);
   return;
}



Node *p=new Node();
Node *p2=new Node();
Node*p1=new Node();


p=head;
```

```
int i=0;
while(i<position-1){


    p1=p;
    p=p->next;
    i++;
}
p1->next=p->next;
p2=p->next;
p2->pre=p1;

free(p);



}
void search(Node *&head,int data ){
    Node *p=new Node();
    p=head;
    bool f=true;
    while(p!=NULL &&f){
        if(p->data==data){
            f=false;
        }
        else{
        p=p->next;}
    }
    if(f!=true){
        cout<<"\n"<<data<<" present at address "<<p<<endl;;
    }
    else{
        cout<<"\n"<<data <<" is not present in this linked list\n";
    }
}
int main()
{
    //Node *head;
    Node *head = NULL;
    Node *second = NULL;

cout<<"\nEnter 1 for creation a doubly linked list "
 <<"\nEnter 2 for insertion at front "
    <<"\nEnter 3 for insertion at end "
    <<"\nEnter 4 for insertion at any position "
    <<"\nEnter 5 for deletipm from front "
    <<"\nEnter 6 for delete from end "
    <<"\nEnter 7 for delete from any position "
    <<"\nEnter 8 for searching a element from linked-list "
    <<"\nEnter 9 for display linked list "
    <<"\nEnter 10 for concatente two doubly linked list"<<endl;

string s="y";
int op,x,y;
while(s=="y" || s=="Y"){
    cout<<"\nEnter the operator : ";
    cin>>op;
    if(op==1){
        cout<<"\nEnter the data of first node ";
        cin>>x;
        creat(head,x);
    }
```

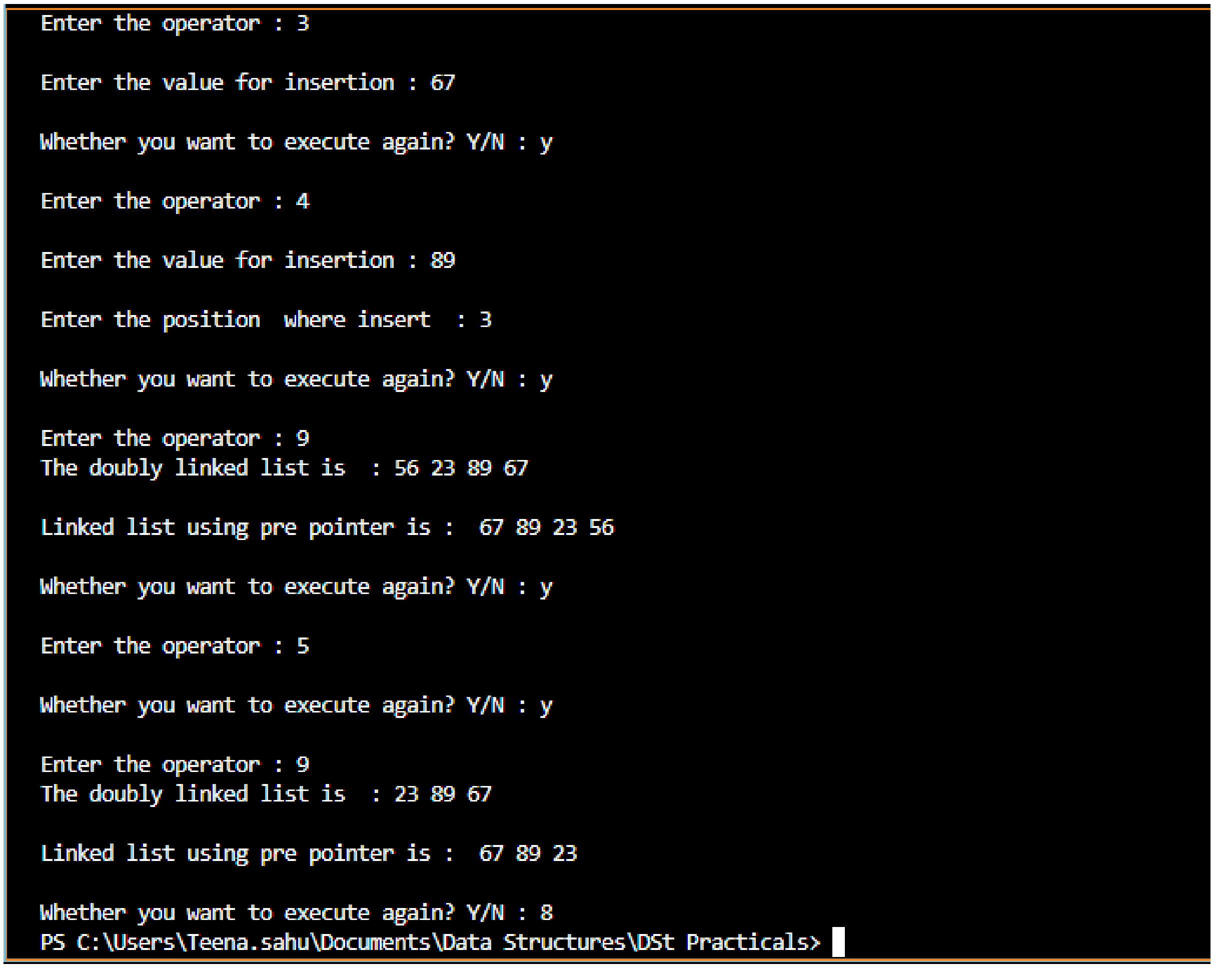
```
else if(op==2){
    cout<<"\nEnter the value for insertion : ";
    cin>>x;
    insertatfront(&head,x);
}
else if(op==3){
     cout<<"\nEnter the value for insertion : ";
    cin>>x;
    addend(&head,x);
}
else if(op==4){
       cout<<"\nEnter the value for insertion : ";
    cin>>x;
       cout<<"\nEnter the position  where insert  : ";
    cin>>y;
    insertatbet(head,x,y);
}
else if(op==5){
    deletefront(head);
}
else if(op==6){
    deleteend(head);
}
else if(op==7){
    cout<<"\nEnter the position of Node: ";
    cin>>y;
    deletbyposition(head,y);
}
else if (op==8){
    cout<<"\nEnter the node data which  you want to serach: ";
```

```
   cin>>x;
   search(head,x);
}
else if(op==9){
display(head);
}
else if (op==10)
 {
   int n;
   int k;
   cout<<endl;
    std::cout << "-------CONCATINATE TWO LIST--------";
    cout<<endl;
   std::cout << "\nHow many nodes you want in doubly linked list? : ";
   cin >> n;
   std::cout << "\nEnter the node data :";
   for (int i = 0; i < n; i++)
   {
      cin >> k;
      append(&head, k);
   }
  // cout<<"Concatenation of doubly linked list is : ";
    printList(head);
   // int n12;
   // int k2;
   // std::cout << "\nHow many nodes you want in doubly linked list?";
   // cin >> n12;
   // std::cout << "\nEnter the node data :";
   // for (int i = 0; i < n; i++)
   // {
        //     cin >> k2;
        //     append(&second, k2);
        // }
        // printList(second);
        // concat(head, &second);
        // printList(head);
     }
    else{
        cout<<"\n Invalid choice";
    }
cout<<"\nWhether you want to execute again? Y/N : ";
cin>>s; }
    return 0;}
```

Output:



```
Enter 1 for creation a doubly linked list
Enter 2 for insertion at front
Enter 3 for insertion at end
Enter 4 for insertion at any position
Enter 5 for deletipm from front
Enter 6 for delete from end
Enter 7 for delete from any position
Enter 8 for searching a element from linked-list
Enter 9 for display linked list
Enter 10 for concatente two doubly linked list

Enter the operator : 1

Enter the data of first node 23

Whether you want to execute again? Y/N : y

Enter the operator : 2

Enter the value for insertion : 56

Whether you want to execute again? Y/N : y

Enter the operator : 3

Enter the value for insertion : 67

Whether you want to execute again? Y/N : y
```



```
Enter the operator : 3

Enter the value for insertion : 67

Whether you want to execute again? Y/N : y

Enter the operator : 4

Enter the value for insertion : 89

Enter the position  where insert  : 3

Whether you want to execute again? Y/N : y

Enter the operator : 9
The doubly linked list is  : 56 23 89 67

Linked list using pre pointer is :  67 89 23 56

Whether you want to execute again? Y/N : y

Enter the operator : 5

Whether you want to execute again? Y/N : y

Enter the operator : 9
The doubly linked list is  : 23 89 67

Linked list using pre pointer is :  67 89 23

Whether you want to execute again? Y/N : 8
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

Q5) Write a program to implement circularly linked list which supports the following operations:

(i) Insert an element x at the front of the circularly linked list

(ii) Insert an element x after an element y in the circularly linked list

(iii)Insert an element x at the back of the circularly linked list

(iv) Remove an element from the back of the circularly linked list

(v) Remove an element from the front of the circularly linked list

(vi) remove the element x from the circularly linked list

(vii)Search for an element x in the circularly linked list and return its pointer

(viii) Concatenate two circularly linked lists

Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
   int data;
   Node *next;
};
void creat(Node *&head,int data){
head->data=data;

head->next=head;



}
void display(Node *&head)
{cout<<"The circular linked list is : ";
Node *p=new Node();
```

```
p=head;
   do
   {
      cout << p->data << " ";
      p = p->next;
   }while (p != head);
   cout<<"\n";
}
void insertatfront(Node *&head,int data1){
   Node*p=new Node();
   Node*p1=new Node();

   p->data=data1;
     p1=head->next;
     while(p1->next!=head){
      p1=p1->next;
     }
     p1->next=p;
     p->next=head;
     head=p;

}
void addend(Node*&head ,int data1){
    Node*p=new Node();
   Node*p1=new Node();

   p->data=data1;
     p1=head->next;
     while(p1->next!=head){
      p1=p1->next;
     }
     p1->next=p;
     p->next=head;




}
void insertatbet(Node*&head,int data1,int position){
   Node*p=new Node();
   Node*p1=new Node();
   Node*p2=new Node();
   Node *p3=new Node();
   p3=head->next;
   p=head;
   p2->data=data1;
  if(p3==head){
   cout<<"\n the linked list have only one node\n";
   return;
  }
  if(position==1){
   insertatfront(head,data1);
   return;

  }
  else {
   int j=1;


   while(p3!=head && j<position-1){
        p3=p3->next;
        j++;
    }
  }
  if(p3==head){
    addend(head,data1);
    return;
  }
int i=1;
while(i<position-1){
p=p->next;
i++;
}
p1=p->next;
p->next=p2;
p2->next=p1;


}


void deletefront(Node *&head){
    Node *p=new Node();
    Node*p1=new Node();
    Node*p2=new Node();
    p=head;
    p1=p->next;
       p2=p->next;
  while(p1->next!=head){
    p1=p1->next;
  }
  p1->next=p2;
```

```
    head=p2;
    free(p);
}
void deleteend(Node *&head){
    Node *p=new Node();
    Node*p1=new Node();
    Node*p2=new Node();
    p2=head->next;
if(p2==head){
    cout<<"\nLinked list can not be empty\n";
    return;
}

p=head;
while(p->next!=head){


    p1=p;
    p=p->next;
}
p1->next=head;
free(p);


}
void deletbyposition(Node *&head,int position){
   Node *p3=new Node();
   p3=head->next;
if(p3==head){
    cout<<"\nLinked list can not be empty\n";
    return;
}
```

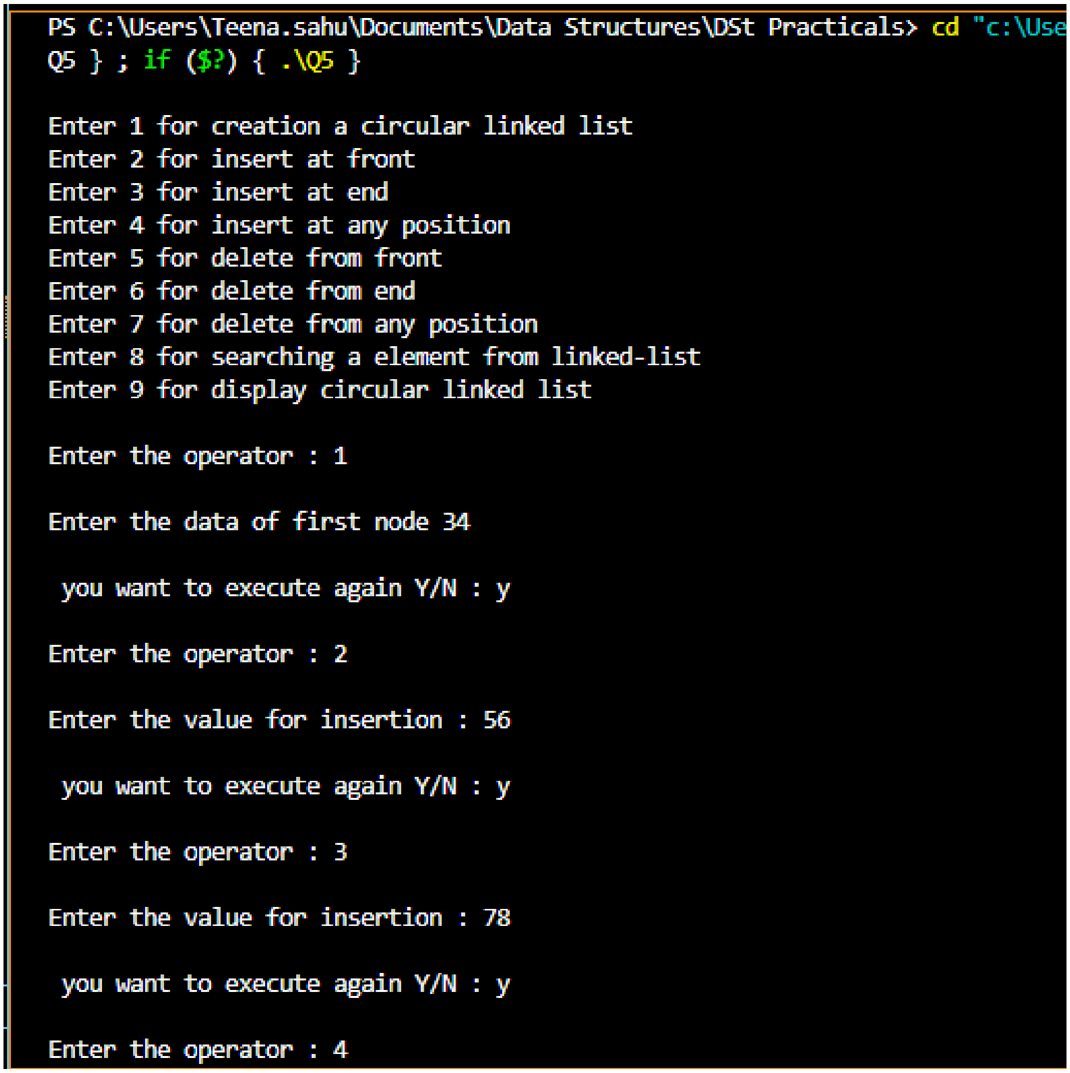
```
// else if(position==1){
//     deletefront(head);
//     return;
// }
else {
   int j=1;
   while(j<position-1 && p3!=head){
      p3=p3->next;
      j++;
   }
}
if(position==1){
   deletefront(head);
   return;
}
else if(p3==head){
   deleteend(head);
   return;
}



Node *p=new Node();
Node *p2=new Node();
Node*p1=new Node();
p=head;
int i=0;
while(i<position-1){

   p1=p;
   p=p->next;
```

```
    i++;
}
p2=p->next;
free(p);
p1->next=p2;

}
void search(Node *&head,int data ){
    Node *p=new Node();
    p=head;
    bool f=true;
    do{
        if(p->data==data){
            f=false;
        }
        else{
        p=p->next;}
    }while(p!=head&&f);
    if(f!=true){
        cout<<"\n"<<data<<" present at address "<<p<<endl;;
    }
    else{
        cout<<"\n"<<data <<" is not present in this linked list\n";
    }
}
int main()
{
    Node *head;
    head->next=NULL;
```

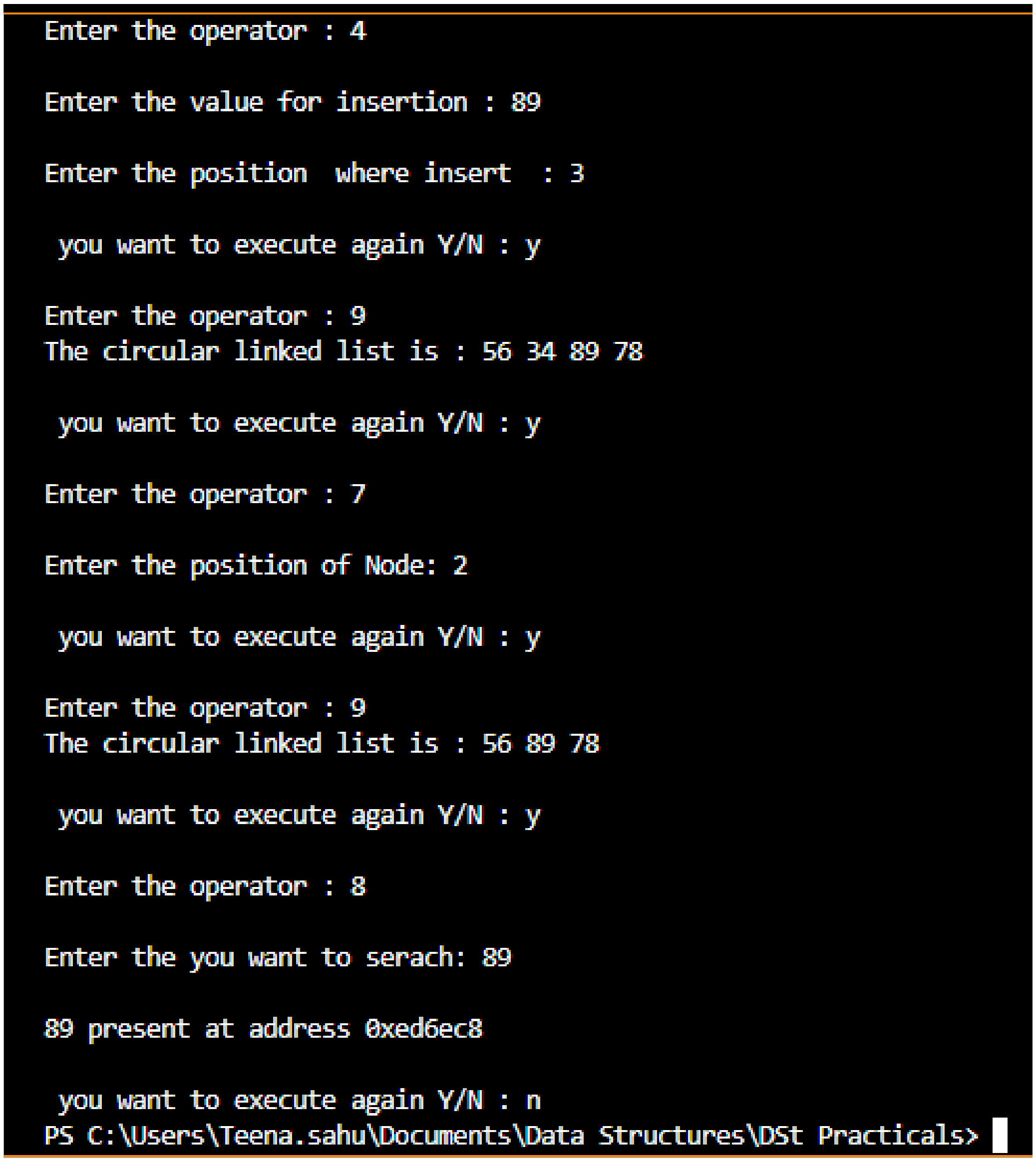
```
cout<<"\nEnter 1 for creation a circular linked list "
    <<"\nEnter 2 for insert at front "
    <<"\nEnter 3 for insert at end "
    <<"\nEnter 4 for insert at any position "
    <<"\nEnter 5 for delete from front "
    <<"\nEnter 6 for delete from end "
    <<"\nEnter 7 for delete from any position "
    <<"\nEnter 8 for searching a element from linked-list "
    <<"\nEnter 9 for display circular linked list "<<endl;

string s="y";
int op,x,y;
while(s=="y" || s=="Y"){
    cout<<"\nEnter the operator : ";
    cin>>op;
    if(op==1){
        cout<<"\nEnter the data of first node ";
        cin>>x;
        creat(head,x);
    }
    else if(op==2){
        cout<<"\nEnter the value for insertion : ";
        cin>>x;
        insertatfront(head,x);
    }
    else if(op==3){
         cout<<"\nEnter the value for insertion : ";
        cin>>x;
        addend(head,x);
    }
    else if(op==4){
          cout<<"\nEnter the value for insertion : ";
        cin>>x;
          cout<<"\nEnter the position  where insert  : ";
        cin>>y;
        insertatbet(head,x,y);
    }
    else if(op==5){
        deletefront(head);
    }
    else if(op==6){
        deleteend(head);
    }
    else if(op==7){
        cout<<"\nEnter the position of Node: ";
        cin>>y;
        deletbyposition(head,y);
    }
    else if (op==8){
        cout<<"\nEnter the you want to serach: ";
        cin>>x;
        search(head,x);
    }
    else if(op==9){
    display(head);
    }
    else{
        cout<<"\n Invalid choice";
    }
  cout<<"\n you want to execute again Y/N : ";
```

```
cin>>s; }
   return 0;}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd "c:\Use
Q5 } ; if ($?) { .\Q5 }

Enter 1 for creation a circular linked list
Enter 2 for insert at front
Enter 3 for insert at end
Enter 4 for insert at any position
Enter 5 for delete from front
Enter 6 for delete from end
Enter 7 for delete from any position
Enter 8 for searching a element from linked-list
Enter 9 for display circular linked list

Enter the operator : 1

Enter the data of first node 34

 you want to execute again Y/N : y

Enter the operator : 2

Enter the value for insertion : 56

 you want to execute again Y/N : y

Enter the operator : 3

Enter the value for insertion : 78

 you want to execute again Y/N : y

Enter the operator : 4
```



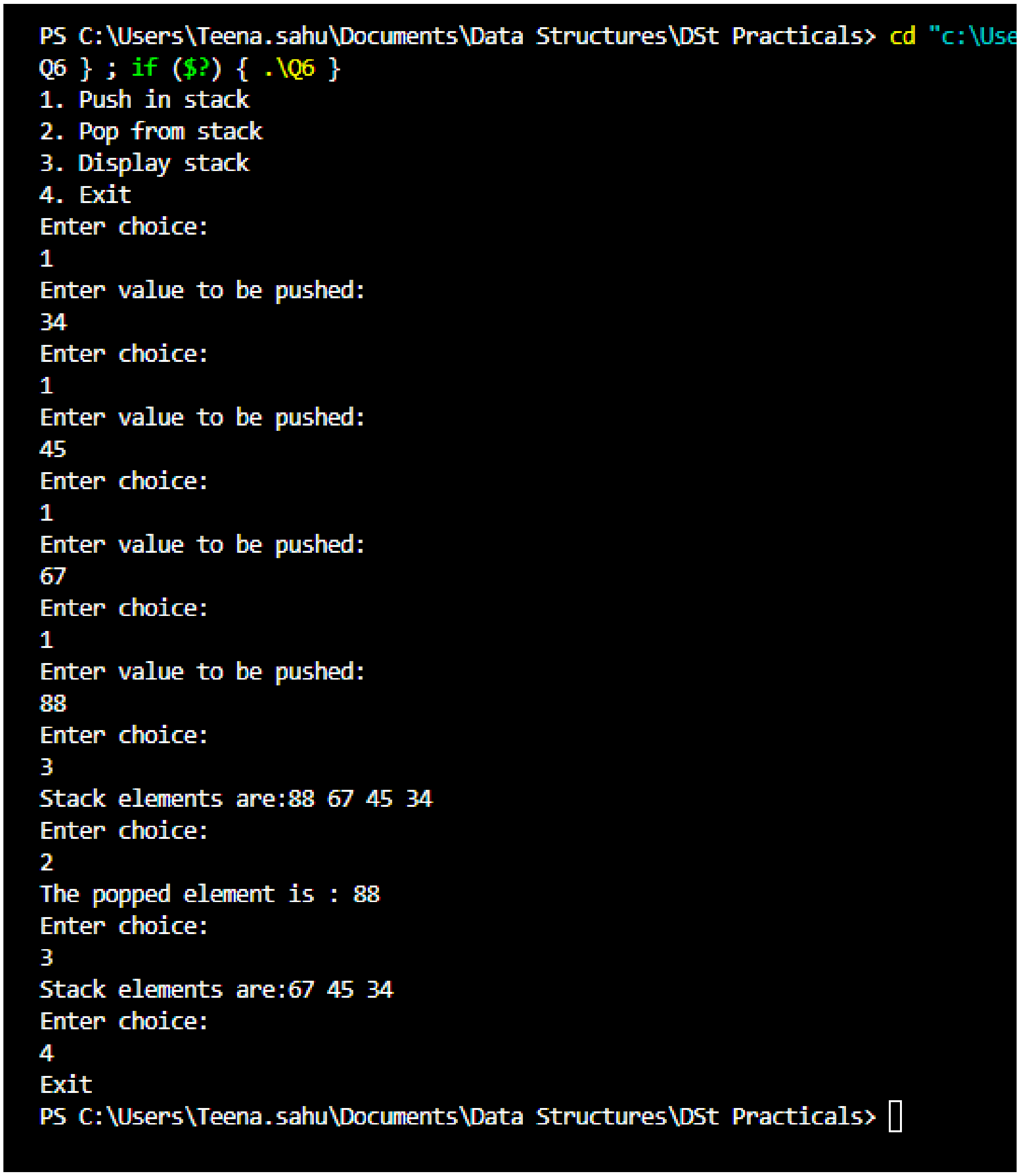
```
Enter the operator : 4

Enter the value for insertion : 89

Enter the position  where insert  : 3

 you want to execute again Y/N : y

Enter the operator : 9
The circular linked list is : 56 34 89 78

 you want to execute again Y/N : y

Enter the operator : 7

Enter the position of Node: 2

 you want to execute again Y/N : y

Enter the operator : 9
The circular linked list is : 56 89 78

 you want to execute again Y/N : y

Enter the operator : 8

Enter the you want to serach: 89

89 present at address 0xed6ec8

 you want to execute again Y/N : n
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

Q6) Implement a stack using Array representation.

Code:

```
#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"<<endl;
}
int main() {
  int ch, val;
  cout<<"1. Push in stack"<<endl;
  cout<<"2. Pop from stack"<<endl;
  cout<<"3. Display stack"<<endl;
  cout<<"4. Exit"<<endl;
  do {
    cout<<"Enter choice: "<<endl;
    cin>>ch;
    switch(ch) {
      case 1: {
        cout<<"Enter 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:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd "c:\Use
Q6 } ; if ($?) { .\Q6 }
1. Push in stack
2. Pop from stack
3. Display stack
4. Exit
Enter choice:
1
Enter value to be pushed:
34
Enter choice:
1
Enter value to be pushed:
45
Enter choice:
1
Enter value to be pushed:
67
Enter choice:
1
Enter value to be pushed:
88
Enter choice:
3
Stack elements are:88 67 45 34
Enter choice:
2
The popped element is : 88
Enter choice:
3
Stack elements are:67 45 34
Enter choice:
4
Exit
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> 
```

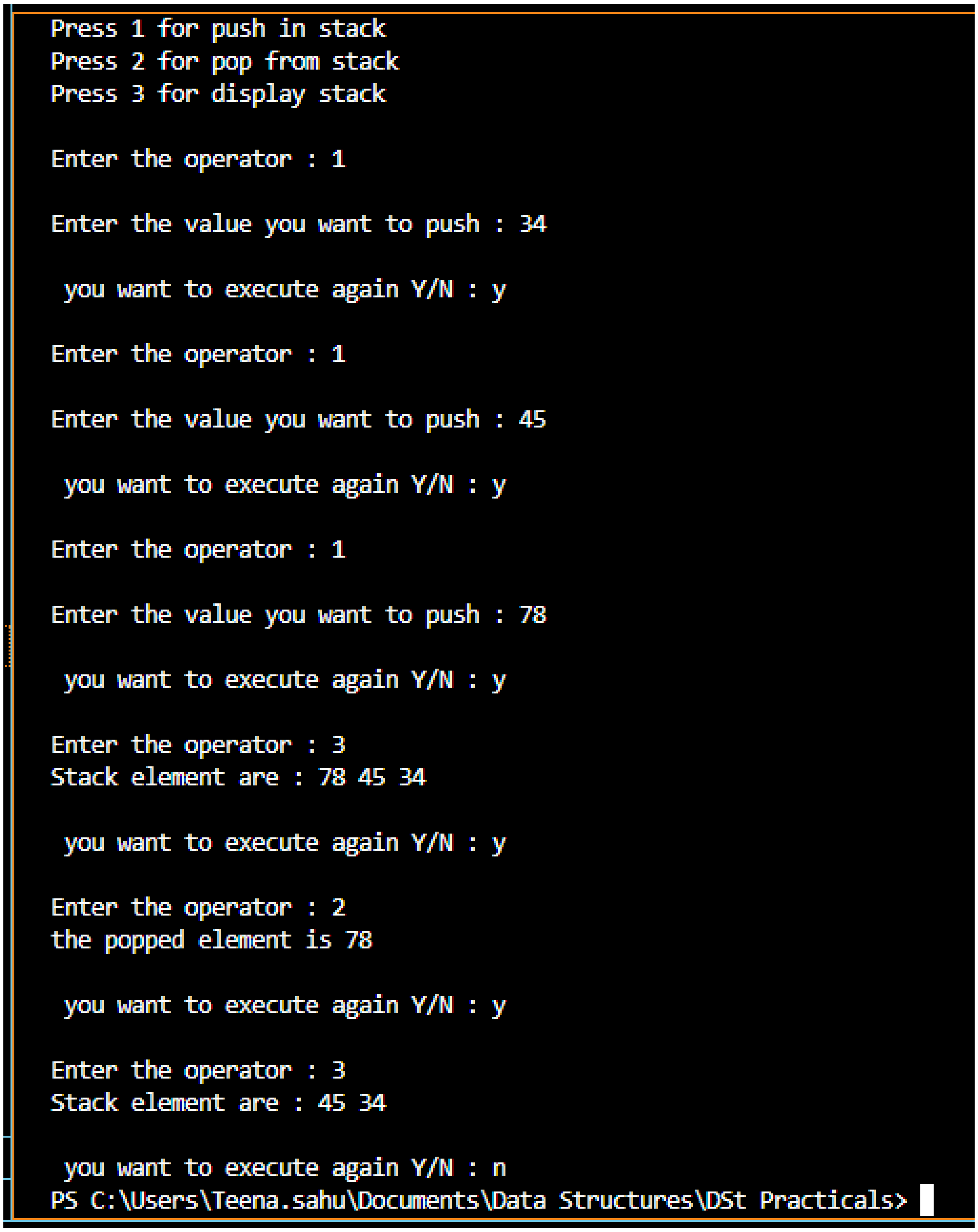
Q7) Implement a stack using Linked representation.

Code:

```
#include <iostream>

using namespace std;

template<class T>

class Node


{private:


 public:

 int size=-1;

    T data;

    Node *next;


    void display(Node*top){




cout<<"Stack element are : ";

   Node<T>*top1=top;

    while (top1 != NULL)

    {

        cout << top1->data << " ";

        top1 = top1->next;

    }

    cout<<"\n";

}

void push(Node * &top,T val){

    Node*n=new Node();
    if(size>=9){
        cout<<"\nstack is overflow\n";
        return;

    }
    if(size<=-1){
        n->data=val;
        n->next=NULL;
        top=n;

        size++;
    }
    else{
n->data=val;
n->next=top;
top=n;
size=size+1;
    }


}
T pop(Node* &top){
    if(size<=-1){
        cout<<"\nstack is underflow\n";
        return -1;
    }
    else{
    Node*p=top;
    T x=p->data;
    top=top->next;
    free(p);
    size=size-1;
    return x;}
}
};
Node<int >*top;
int main()
{
    Node <int >head;
    Node<int>*h;

string s="y";
int op;
int  x;
cout<<"\nPress 1 for push in stack "<<"\nPress 2 for pop from stack "<<"\nPress 3 for display stack "<<endl;
while(s=="y" || s=="Y"){
    cout<<"\nEnter the operator : ";
    cin>>op;
    if(op==1){
        cout<<"\nEnter the value you want to push : ";
        cin>>x;
        head.push(top,x);
    }
    else if(op==2){
     int b= head.pop(top);
     if(b!=-1){
     cout<<"the popped element is "<<b<<endl;}
    }
    else if(op==3){
    head.display(top);
```

```
   }
       else{
       cout<<"\n Invalid choice";
   }
cout<<"\n you want to execute again Y/N : ";
cin>>s;
   }
// head.push(h,78);
// head.push(h,718);

// head.display(h);
// int b=head.pop(h);
// cout<<"the popped element is "<<b<<endl;
// head.display(h);

   return 0;}
```

Output:



```
Press 1 for push in stack
Press 2 for pop from stack
Press 3 for display stack

Enter the operator : 1

Enter the value you want to push : 34

 you want to execute again Y/N : y

Enter the operator : 1

Enter the value you want to push : 45

 you want to execute again Y/N : y

Enter the operator : 1

Enter the value you want to push : 78

 you want to execute again Y/N : y

Enter the operator : 3
Stack element are : 78 45 34

 you want to execute again Y/N : y

Enter the operator : 2
the popped element is 78

 you want to execute again Y/N : y

Enter the operator : 3
Stack element are : 45 34

 you want to execute again Y/N : n
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

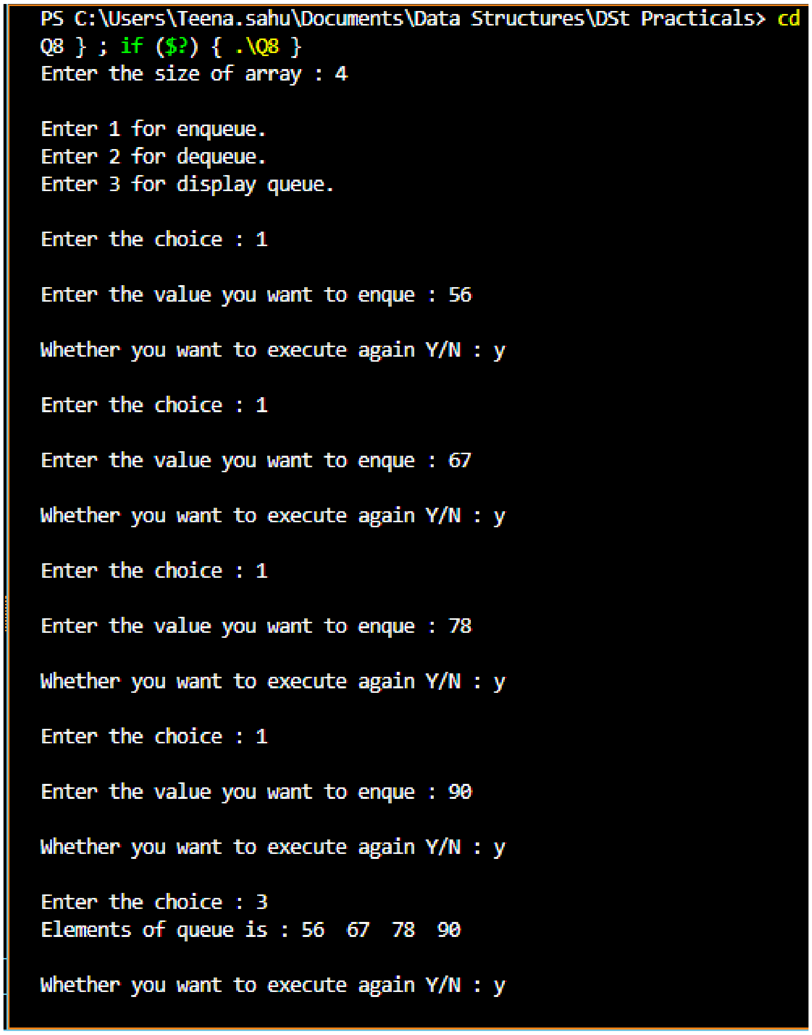
Q8) Implement Queue using Circular Array representation.

Code:

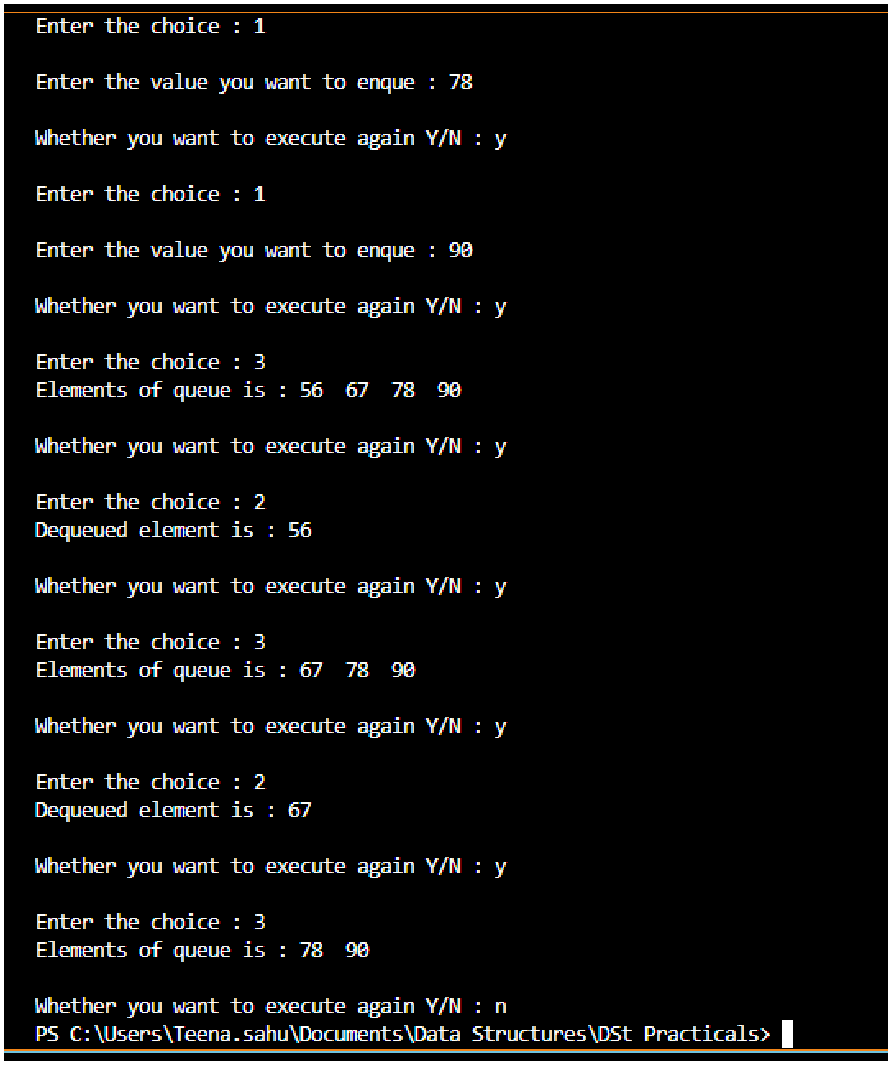
```
#include <bits/stdc++.h>
using namespace std;
template <class T>

class quque1{
    int f=-1,r=-1;
    int size;
    T arr[];
    public:
    quque1(int a){
        size=a;
        arr[size];
    }
    void push(T data){
        if(f==((r+1)%size)) {
            cout<<"\nQueue is overflow\n";
        }
        else if(f==-1)   {
            f=(f+1)%size;
            r=(r+1)%size;
            arr[r]=data;
        }
        else{
         r=(r+1)%size;
            arr[r]=data;
          }
    }
```

```
T pop(){
    if(r==-1 && f==-1 ){
        cout<<"Queue is underflow\n";
        return -1;
    }
    else if(f==r && f!=-1){
      T v=  arr[f];
      r=-1;
      f=-1;
      return v;
    }



    else{
        T v=arr[f];
        f=(f+1)%size;
        return v;
    }
}
void display(){
    cout<<"Elements of queue is : ";
    // cout<<s;
    for(int i=f;i>-1;i=(i+1)%size){
        cout<<arr[i]<<"  ";
        if(i==r)
        {
            break;
        }
    }
    cout<<endl;
    }


};
int main(){
    //stack <int> s(6) ;
    int b;
    string str;
    str="y";


    cout<<"Enter the size of array : ";
    cin>>b;
    quque1 <int>s(b);
string s1="y";
int op;
int  x;
cout<<"\nEnter 1 for enqueue. "
    <<"\nEnter 2 for dequeue. "
    <<"\nEnter 3 for display queue. "<<endl;
while(s1=="y" || s1=="Y"){
    cout<<"\nEnter the choice : ";
    cin>>op;
    if(op==1){
        cout<<"\nEnter the value you want to enque : ";
        cin>>x;
        s.push(x);
    }
    else if(op==2){
     int b1= s.pop();
     if(b1!=-1){
     cout<<"Dequeued element is : "<<b1<<endl;}
    }
    else if(op==3){
   s.display();
    }
        else{
        cout<<"\n Invalid choice";
    }
cout<<"\nWhether you want to execute again Y/N : ";
cin>>s1;
    }
       return 0;
}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd
Q8 } ; if ($?) { .\Q8 }
Enter the size of array : 4

Enter 1 for enqueue.
Enter 2 for dequeue.
Enter 3 for display queue.

Enter the choice : 1

Enter the value you want to enque : 56

Whether you want to execute again Y/N : y

Enter the choice : 1

Enter the value you want to enque : 67

Whether you want to execute again Y/N : y

Enter the choice : 1

Enter the value you want to enque : 78

Whether you want to execute again Y/N : y

Enter the choice : 1

Enter the value you want to enque : 90

Whether you want to execute again Y/N : y

Enter the choice : 3
Elements of queue is : 56   67   78   90

Whether you want to execute again Y/N : y
```



```
Enter the choice : 1

Enter the value you want to enque : 78

Whether you want to execute again Y/N : y

Enter the choice : 1

Enter the value you want to enque : 90

Whether you want to execute again Y/N : y

Enter the choice : 3
Elements of queue is : 56  67  78  90

Whether you want to execute again Y/N : y

Enter the choice : 2
Dequeued element is : 56

Whether you want to execute again Y/N : y

Enter the choice : 3
Elements of queue is : 67  78  90

Whether you want to execute again Y/N : y

Enter the choice : 2
Dequeued element is : 67

Whether you want to execute again Y/N : y

Enter the choice : 3
Elements of queue is : 78  90

Whether you want to execute again Y/N : n
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

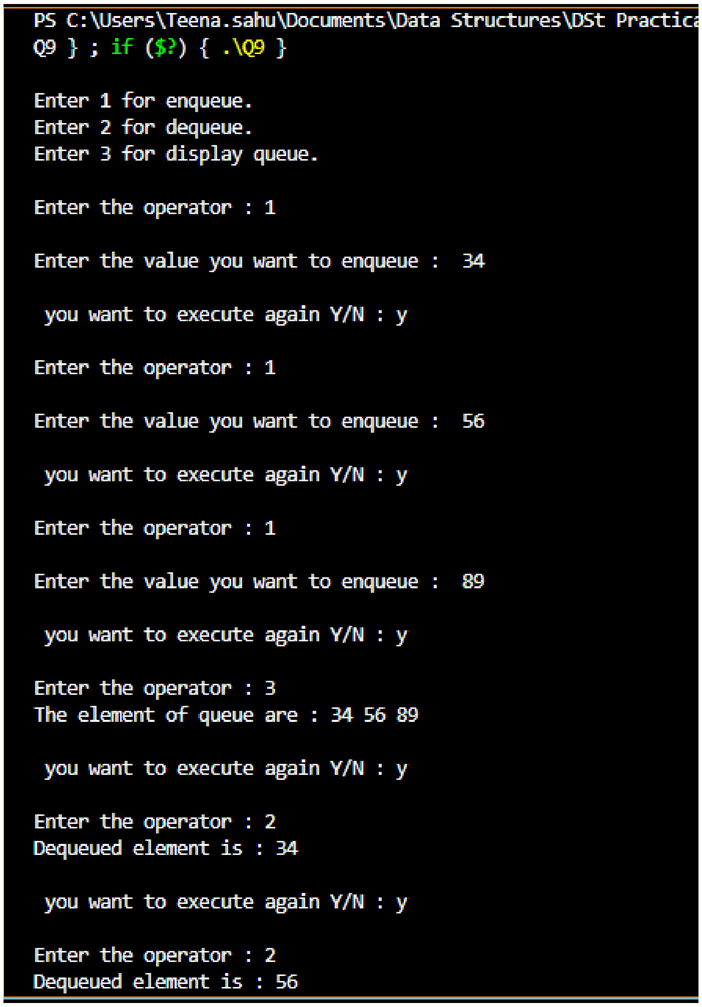
Q9) Implement Queue using Circular linked list representation.

Code:

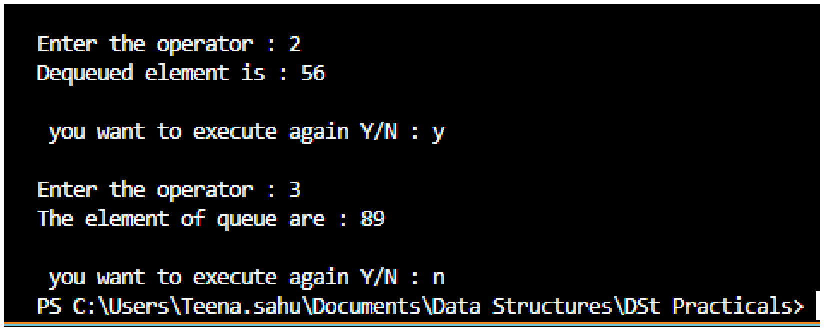
```
#include <iostream>
using namespace std;
template<class T>
class Node

{private:

 public:
 int size=-1;
    T data;
    Node *next;

    void display(Node*top){
     if(size==-1){
        cout<<" \nQueue is empty ";
        return;
     }

cout<<"The element of queue are : ";
   Node<T>*top1=top;
   do{
      cout << top1->data << " ";
        top1 = top1->next;
   }
    while (top1 != top);

    cout<<"\n";
}
void enqueue(Node * &top,T val){
    Node*n=new Node();

    if(size>=9){
        cout<<"\nQueue is overflow\n";
        return;

    }
    if(size<=-1){
        n->data=val;
        n->next=n;
        top=n;
        size++;
    }
    else{
Node*p=new Node();
Node*p1=new Node();
Node*p2=new Node();
p1->data=val;

p=top;
while(p->next!=top){
    p=p->next;
}
p2=p->next;
p1->next=p2;
p->next=p1;
size=size+1;
    }

}
T  denqueue(Node* &top){
    if(size<=-1){
        cout<<"\nQueue is underflow\n";
        return -1;
    }
    else{
    Node*p=top;
    Node*p1=p->next;
    Node *p2=p->next;
    while(p2->next!=top){
      p2=p2->next;
    }
    p2->next=p1;
    top=p1;
    T x=p->data;

    free(p);
    size=size-1;
    return x;}
}


};


Node<int >*top;
```

```
int main()
{
   Node <int >head;
   Node<int>*h;



string s="y";
int op;
int  x;
cout<<"\nEnter 1 for enqueue. "<<"\nEnter 2 for dequeue. "<<"\nEnter 3 for display queue.  "<<endl;
while(s=="y" || s=="Y"){
   cout<<"\nEnter the operator : ";
   cin>>op;
   if(op==1){

      cout<<"\nEnter the value you want to enqueue :  ";
      cin>>x;
      head.enqueue(top,x);
   }
   else if(op==2){
    int b= head.denqueue(top);
    if(b!=-1){
    cout<<"Dequeued element is : "<<b<<endl;}
   }
   else if(op==3){
   head.display(top);
   }
      else{
      cout<<"\n Invalid choice";
   }
cout<<"\n you want to execute again Y/N : ";
cin>>s;
   }
// head.push(h,78);
// head.push(h,718);

// head.display(h);
// int b=head.pop(h);
// cout<<"the popped element is "<<b<<endl;
// head.display(h);
return 0;}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practica
Q9 } ; if ($?) { .\Q9 }

Enter 1 for enqueue.
Enter 2 for dequeue.
Enter 3 for display queue.

Enter the operator : 1

Enter the value you want to enqueue :  34

 you want to execute again Y/N : y

Enter the operator : 1

Enter the value you want to enqueue :  56

 you want to execute again Y/N : y

Enter the operator : 1

Enter the value you want to enqueue :  89

 you want to execute again Y/N : y

Enter the operator : 3
The element of queue are : 34 56 89

 you want to execute again Y/N : y

Enter the operator : 2
Dequeued element is : 34

 you want to execute again Y/N : y

Enter the operator : 2
Dequeued element is : 56
```



```
Enter the operator : 2
Dequeued element is : 56

 you want to execute again Y/N : y

Enter the operator : 3
The element of queue are : 89

 you want to execute again Y/N : n
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

Q10) Implement Double-ended Queues using Linked list representation.

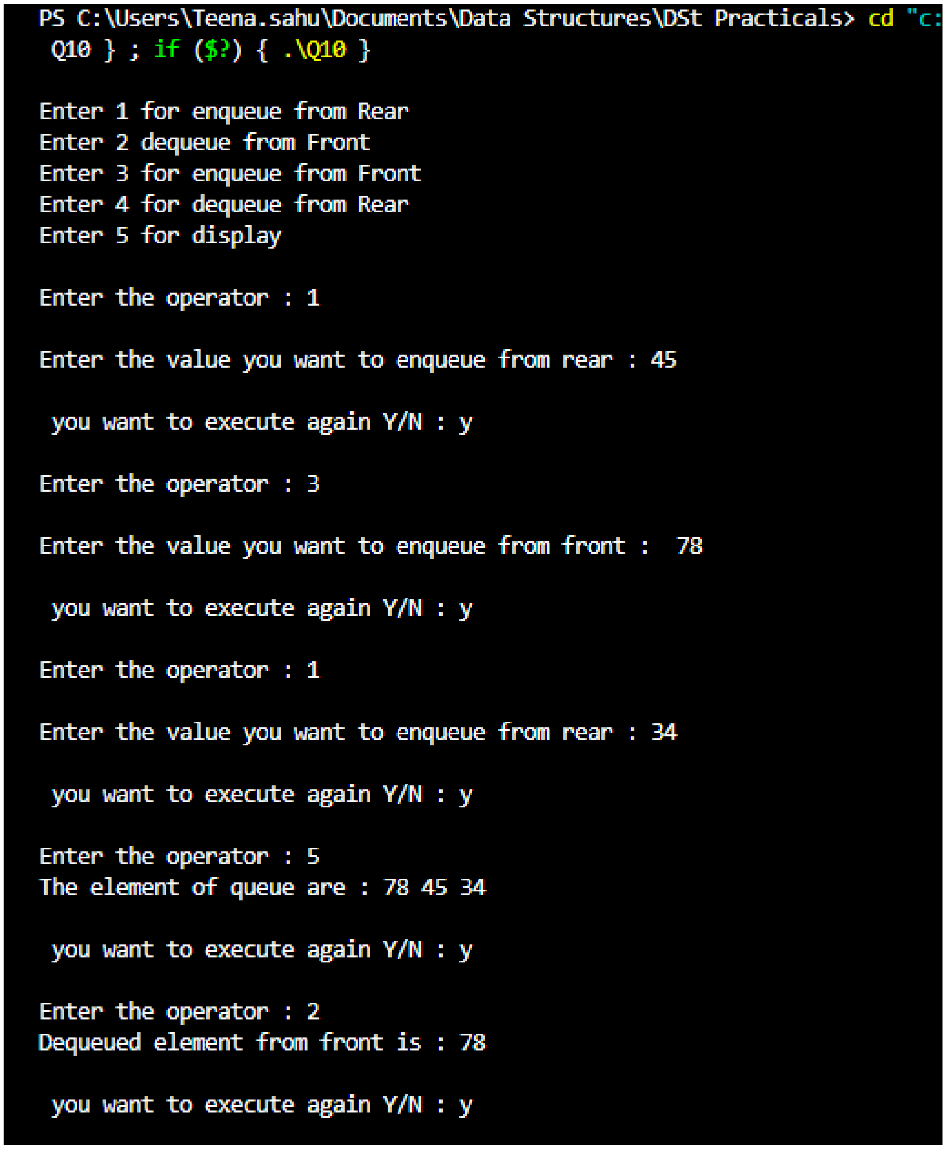
Code:

```
#include <iostream>
using namespace std;
template <class T>
class Node

{
private:
public:
    int size = -1;
    T data;
    Node *next;

    void display(Node *top)
    {

        cout << "The element of queue are : ";
        Node<T> *top1 = top;
        while (top1 != NULL)
        {
            cout << top1->data << " ";
            top1 = top1->next;
        }
        cout << "\n";
    }
    void enqueuer(Node *&top, T val)
    {
        Node *n = new Node();
    if (size >= 9)
    {
        cout << "\nQueue is overflow\n";
        return;
    }
    if (size <= -1)
    {
        n->data = val;
        n->next = NULL;
        top = n;

        size++;
    }
    else
    {
        Node *p = new Node();
        Node *p1 = new Node();
        p1->data = val;
        p1->next = NULL;
        p = top;
        while (p->next != NULL)
        {
            p = p->next;
        }
        p->next = p1;
        size = size + 1;
    }
}
T dequeuef(Node *&top)
```

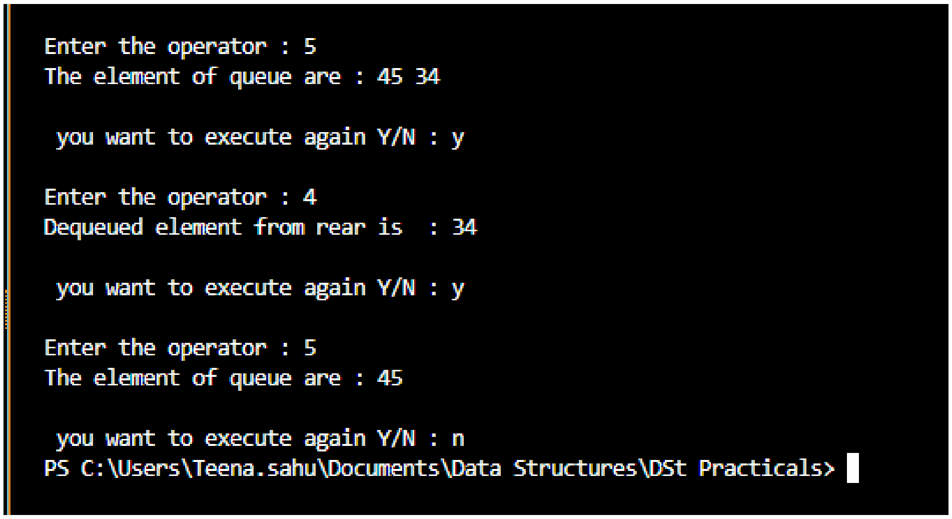
```
{
    if (size <= -1)
    {
        cout << "\nQueue is underflow\n";
        return -1;
    }
    else
    {
        Node *p = top;
        T x = p->data;
        top = top->next;
        free(p);
        size = size - 1;
        return x;
    }
}
void enqueuef(Node * &top,T val){
Node*n=new Node();

if(size>=9){
    cout<<"\nQueue is overflow\n";
    return;

}
if(size<=-1){
    n->data=val;
    n->next=NULL;
    top=n;

    size++;
    }
    else{
n->data=val;
n->next=top;
top=n;
size=size+1;
    }

}
T dequeuer(Node* &top){
    if(size<=-1){
        cout<<"\nQueue is underflow\n";
        return -1;
    }
    else{
    Node*p=top;
    Node*p1=new Node();
    Node*p2=new Node();

    while(p->next!=NULL){
        p1=p;
        p=p->next;
    }
    T x=p->data;
    p1->next=NULL;
    free(p);
    size=size-1;
    return x;}
}
};
```

```
Node<int> *top;

int main()
{
    Node<int> head;
    Node<int> *h;

    string s = "y";
    int op;
    int x;
    cout << "\nEnter 1 for enqueue from Rear "
         << "\nEnter 2 dequeue from Front  "
         << "\nEnter 3 for enqueue from Front  "
         << "\nEnter 4 for dequeue from Rear  "
         << "\nEnter 5 for display  " << endl;
    while (s == "y" || s == "Y")
    {
        cout << "\nEnter the operator : ";
        cin >> op;
        if (op == 1)
        {
            cout << "\nEnter the value you want to enqueue from rear : ";
            cin >> x;
            head.enqueuer(top, x);
        }
        else if (op == 2)
        {
            int b = head.dequeuef(top);
            if (b != -1)
            {
                cout << "Dequeued element from front is : " << b << endl;
            }
        }
         else if (op == 3)
        {
            cout << "\nEnter the value you want to enqueue from front :  ";
            cin >> x;
            head.enqueuef(top, x);
        }
          else if (op == 4)
        {
            int b = head.dequeuer(top);
            if (b != -1)
            {
                cout << "Dequeued element from rear is  : " << b << endl;
            }
        }
        else if (op == 5)
        {
            head.display(top);
        }
        else
        {
            cout << "\n Invalid choice";
        }
        cout << "\n you want to execute again Y/N : ";
        cin >> s;
    }
    // head.push(h,78);
    // head.push(h,718);


    // head.display(h);
    // int b=head.pop(h);
    // cout<<"the popped element is "<<b<<endl;
    // head.display(h);


    return 0;
}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd "c:
 Q10 } ; if ($?) { .\Q10 }

Enter 1 for enqueue from Rear
Enter 2 dequeue from Front
Enter 3 for enqueue from Front
Enter 4 for dequeue from Rear
Enter 5 for display

Enter the operator : 1

Enter the value you want to enqueue from rear : 45

 you want to execute again Y/N : y

Enter the operator : 3

Enter the value you want to enqueue from front :  78

 you want to execute again Y/N : y

Enter the operator : 1

Enter the value you want to enqueue from rear : 34

 you want to execute again Y/N : y

Enter the operator : 5
The element of queue are : 78 45 34

 you want to execute again Y/N : y

Enter the operator : 2
Dequeued element from front is : 78

 you want to execute again Y/N : y
```



```
Enter the operator : 5
The element of queue are : 45 34

 you want to execute again Y/N : y

Enter the operator : 4
Dequeued element from rear is  : 34

 you want to execute again Y/N : y

Enter the operator : 5
The element of queue are : 45

 you want to execute again Y/N : n
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> █
```

Q11) Write a program to implement Binary Search Tree which supports the following operations:

(i) Insert an element x

(ii) Delete an element x

(iii) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position in the BST

(iv) Display the elements of the BST in preorder, inorder, and postorder traversal

(v) Display the elements of the BST in level-by-level traversal

(vi) Display the height of the BST

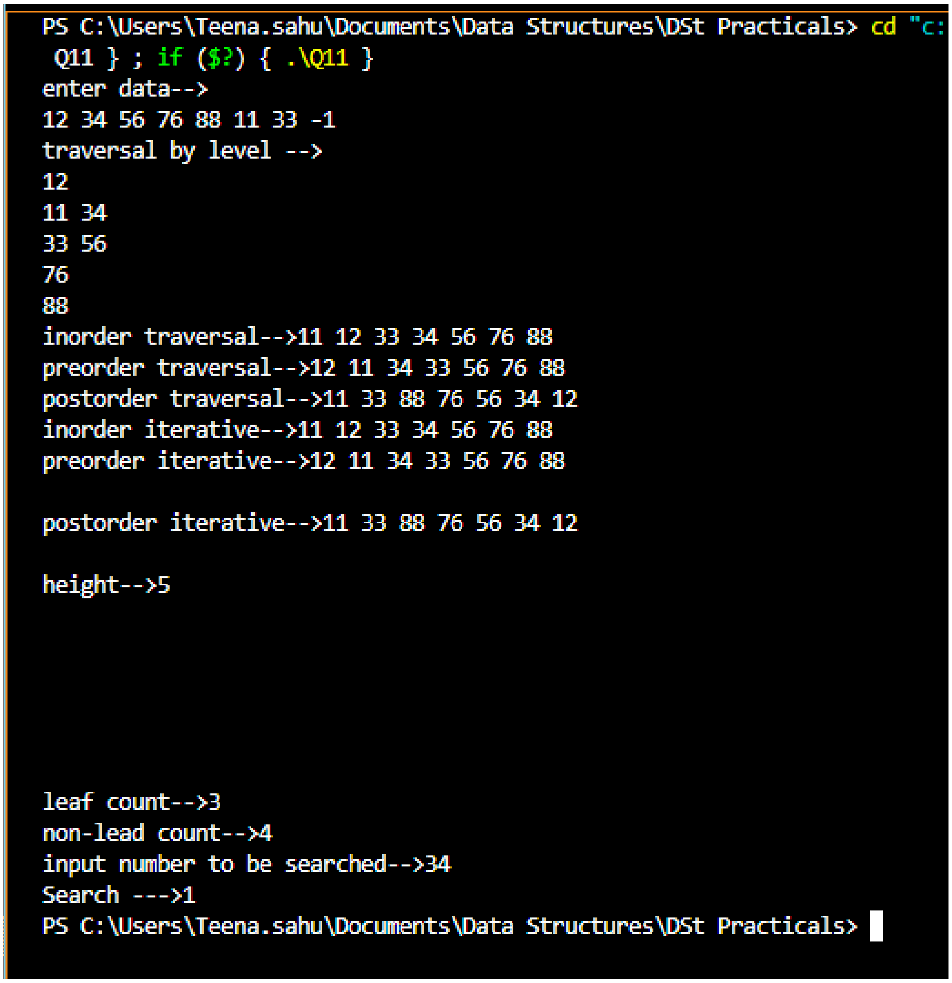
Code:

```
#include <bits/stdc++.h>
using namespace std;
class node
{
public:
   int data;
   node *left;
   node *right;
   node(int d)
   {
      data = d;
      left = NULL;
      right = NULL;
   }
};

void treversalByLevel(node *root)
{
   // cout<<"root-data--> "<<root->data<<endl;
   queue<node *> q;
   q.push(root);
   q.push(NULL);
   while (!q.empty())
   {
      node *temp = q.front();
      // cout<<"front-->"<<temp->data<<"\n";
```

```
            q.pop();
            if (temp == NULL)
            {
                cout << endl;
                if (!q.empty())
                    q.push(NULL);
            }
            else
            {
                cout << temp->data << " ";
                if (temp->left)
                    q.push(temp->left);
                if (temp->right)
                {
                    q.push(temp->right);
                }
            }
        }
    }
    node *insertInBst(node *&root, int d)
    {

        if (root == NULL)
        {
            // cout<<"ok1\n";
            root = new node(d);
            return root;
        }
        if (d > root->data)
        {
            // cout<<"ok2\n";

            root->right = insertInBst(root->right, d);
        }
        else
        {
            // cout<<"ok3\n";
            root->left = insertInBst(root->left, d);
    }
    // treversalByLevel(root);
    return root;
}
void takeinput(node *&root)
{
    int data;
    cout<<"enter data-->\n";
    cin >> data;
    while (data != -1)
    {
        // cout<<"ok\n
        root = insertInBst(root, data);
        cin >> data;
    }
}
void inorder(node *&root)
{
    if (root == NULL)
        return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

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

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

void leafs(node *root, int &cnt, int &leafcnt)
{
    if (root == NULL)
        return;
    leafs(root->left, cnt, leafcnt);
    if (root->left == NULL && root->right == NULL)
    {
        cnt++;
        // cout<<root->data<< " ";
    }
    else
    {
        leafcnt++;
    }
    leafs(root->right, cnt, leafcnt);
    // cout<<root->data<<" ";
    cout << endl;
}
void inorderIterative(node *&root)
{
    stack<node *> s;
    node *curr = root;
    while (curr != NULL || s.empty() == false)
    {
        while (curr != NULL)
        {
            s.push(curr);
            curr = curr->left;
        }
        // cout<<"s.top=-->"<<s.top()->data<<endl;
        curr = s.top();
        s.pop();
        cout << curr->data << " ";
        curr = curr->right;
    }
}
void preorderIterative(node *&root)
{
    stack<node *> s;
    node *curr = root;
    while (curr != NULL || s.empty() == false)
    {
        while (curr != NULL)
        {
            s.push(curr);
            cout << curr->data << " ";
            curr = curr->left;
        }
        curr = s.top();
        s.pop();
        curr = curr->right;
    }
    cout << endl;
}

void postorderIterative(node *&root)
{
    stack<node *> s;
    node *curr = root;
    while (curr != NULL || s.empty() == false)
    {
        if (curr != NULL)
        {
            s.push(curr);
            curr = curr->left;
        }
        else
        {
            node *temp = s.top()->right;
            if (temp == NULL)
            {
                temp=s.top();
                s.pop();
                cout<<temp->data<<" ";
                while(s.empty()==false && s.top()->right==temp)
                {
                    temp=s.top();
                    s.pop();
                    cout<<temp->data<<" ";
                }
            }
            else
            {
                curr=temp;
            }
        }
    }
}

int height(node *&root)
{
    if(root==NULL) return 0;
    int left=height(root->left);
    int right=height(root->right);
    int ans=max(left,right)+1;

    return ans;
}

bool search(node *root,int x)
{
    if(root==NULL) return false;
    if(root->data==x) return true;

    if(root->data>x)
    {
        search(root->left,x);
    }
    else
    {
        search(root->right,x);
    }
}
int main()
{
    node *root = NULL;
    takeinput(root);


    cout << "traversal by level -->\n";
    treversalByLevel(root);


    cout << "inorder traversal-->\n";
    inorder(root);


    cout << "preorder traversal-->\n";
    preorder(root);


    cout << "postorder traversal-->\n";
    postorder(root);


    cout << "inorder iterative-->";
    inorderIterative(root);


    cout << "preorder iterative-->";
    preorderIterative(root);


    cout << "postorder iterative-->";
    postorderIterative(root);

cout<<endl;
```

```
    cout<<"height-->"<<height(root);

    int cnt = 0;
    int leafcnt = 0;

    leafs(root, cnt, leafcnt);
    cout << "leaf count-->" << cnt << endl;
    cout << "non-lead count-->" << leafcnt << endl;



    int x;
    cout<<"input number to be searched-->";
    cin>>x;
    cout<<"Search --->"<<search(root,x)<<endl;
}
```

Output:



```
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals> cd "c:
 Q11 } ; if ($?) { .\Q11 }
enter data-->
12 34 56 76 88 11 33 -1
traversal by level -->
12
11 34
33 56
76
88
inorder traversal-->11 12 33 34 56 76 88
preorder traversal-->12 11 34 33 56 76 88
postorder traversal-->11 33 88 76 56 34 12
inorder iterative-->11 12 33 34 56 76 88
preorder iterative-->12 11 34 33 56 76 88

postorder iterative-->11 33 88 76 56 34 12

height-->5






leaf count-->3
non-lead count-->4
input number to be searched-->34
Search --->1
PS C:\Users\Teena.sahu\Documents\Data Structures\DSt Practicals>
```

THANK YOU