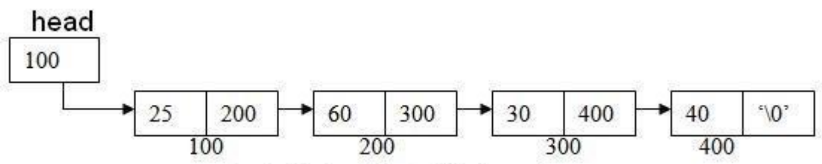
AIM: Implementation of all types of linked List Insert, Display, Delete, Search, Count, Reverse operation on Singly Linked Lists , Circular Linked List & Doubly Linked Lists

THEORY:

A) Singly Linked Lists:

Single linked list is a sequence of elements in which every element has link to its next element in the sequence. In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data field, and the next field.



singly linked list with 4 nodes

Advantages over arrays:
1) Dynamic size
2) Ease of insertion/deletion

Drawbacks:
1) Random access is not allowed. We have to access elements sequentially starting from the first node. So, we cannot do binary search with linked lists efficiently with its default implementation.
2) Extra memory space for a pointer is required with each element of the list.
3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Representation:
A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.
Each node in a list consists of at least two parts:
1) data
2) Pointer (Or Reference) to the next node

In CPP, we can represent a node using structures. Below is an example of a linked list node with integer data.
In Java or C#, LinkedList can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.

B) Circular Linked List:

Doubly linked list is a type of linked list in which each node apart from storing its data has two links. The first link points to the previous node in the list and the second link points to the next node in the list.

Advantages over singly linked list:

1) A DLL can be traversed in both forward and backward direction.

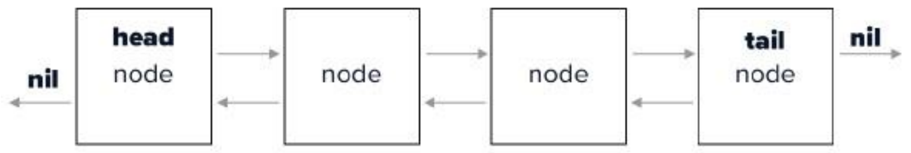
2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

3) We can quickly insert a new node before a given node. In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list:

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though.

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.



C) Doubly Linked Lists:

A circular linked list is a sequence of elements in which every element has a link to its next element in the sequence and the last element has a link to the first element. That means circular linked list is similar to the single linked list except that the last node points to the first node in the list.

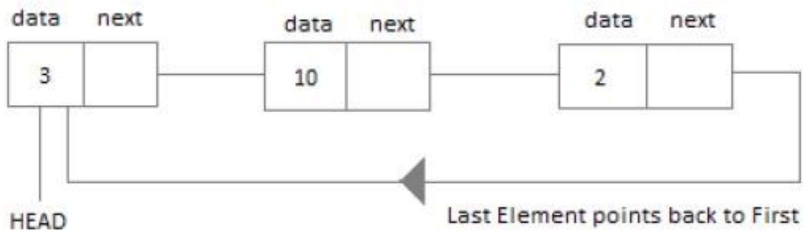
Advantages of Circular Linked Lists:

1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2) Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.

3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

4) Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.



A) SINGLY LINKED LISTS:

SOURCE CODE:

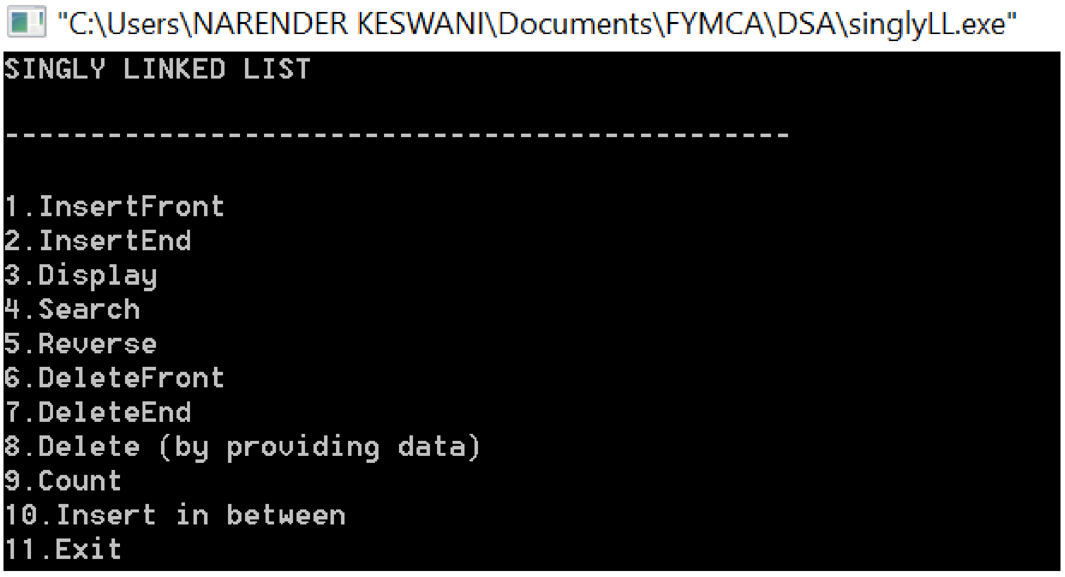
```
#include<iostream>
using namespace std;
class node
{
public:
    int data;
    node* next;

    node(int val)
    {
        data=val;
        next=NULL;
    }
};

int nodeCount = 0;

void insertStart(node*&head, int val)
{
    nodeCount+=1;
    node* n= new node(val);
    n->next= head;
    head= n;
}

void insertEnd(node* &head, int val)
{
    node* n = new node(val);
    node* temp = head;

    nodeCount+=1;
    if(head==NULL)
    {
        head=n;
        return;
    }

    while(temp->next!=NULL)
    {

        temp=temp->next;
    }
    temp->next=n;
    return;

}

void insertInBetween(node* &head, int data, int n)
```

```
{
   node* temp1 = new node(data);
   temp1->next = NULL;
   if(n == 1)
   {
      nodeCount+=1;
      temp1->next = head;
      head = temp1;
      return;
   }

   node* temp2 = head;

   for(int i=0; i<n-2; i++)
   {
      temp2 = temp2->next;
   }

   nodeCount+=1;
   temp1->next = temp2->next;
   temp2->next = temp1;
}

void search(node* head, int data)
{
   node *curr = head;
   while(curr!= NULL)
   {
      if(curr->data == data)
      {
         cout<<"Found in List\n";
         return;
      }
      curr = curr->next;
   }
   cout<<"Not Found\n";
}

void deleteNode(node **head, int key)
{
   node *temp = *head;
   node *prev = NULL;
   nodeCount-=1;
   if(temp != NULL && temp->data == key)
   {
      *head = temp->next;
      delete temp;
      return;
   }
   else
   {
        while(temp != NULL && temp->data != key)
        {
            prev = temp;
            temp = temp->next;
        }
        if(temp == NULL)
        {
            return;
        }
        prev->next = temp->next;
        delete temp;
    }
}

void deleteFront(node* &head)
{
    if(head == NULL)
    {
        cout<<"Underflow"<<endl;
        return;
    }
    else
    {
        nodeCount-=1;
        node* temp = head;
        head = head->next;
        delete temp;
        return;
    }
}

void deleteEnd(node* &head)
{
    if (head == NULL)
    {
        cout<<"UnderFlow"<<endl;
        return;
    }

    else if (head->next == NULL)
    {
        nodeCount-=1;
        delete head;
        return;
    }
    else
    {
        node* temp = head;
        nodeCount-=1;
        while (temp->next->next != NULL)
            temp = temp->next;
        delete (temp->next);

        temp->next = NULL;
        return;
    }


}

void reverseList(node** head)
{
    node* prev = NULL;
    node* curr = *head;
    node* next = NULL;
    while(curr != NULL)
    {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}

void display(node *head)
{
    if(head == NULL)
    {
        cout<<"EMPTY LINKED LIST"<<endl;
    }
    while(head != NULL)
    {
        cout<<head->data<<" -> ";
        head = head->next;
    }
    cout<<"\n";
}


int main()
{
    node *head = NULL;
    int choice, data;
    cout<<"SINGLY LINKED LIST"<<endl;
    cout<<"\n---------------------------------------------"<<endl;
    cout<<"\n1.InsertFront \n2.InsertEnd \n3.Display \n4.Search \n5.Reverse
\n6.DeleteFront \n7.DeleteEnd \n8.Delete (by providing data) \n9.Count \n10.Insert
in between \n11.Exit"<<endl;
    do
    {
```

```
cout<<"\nEnter the Choice: \n"<<endl;
cin>>choice;
switch(choice)
{
case 1:
   cout<<"Enter data to append (start): ";
   cin>>data;
   insertStart(head,data);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 2:
   cout<<"Enter data to append (end): ";
   cin>>data;
   insertEnd(head,data);
   break;
case 3:
   display(head);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 4:
   cout<<"Enter data to Search : ";
   cin>>data;
   search(head,data);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 5:
   reverseList(&head);
   display(head);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 6:
   deleteFront(head);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 7:
   deleteEnd(head);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 8:
   cout<<"Enter data to Delete :- ";
   cin>>data;
   deleteNode(&head,data);
   cout<<"\n---------------------------------------------"<<endl;
case 9:
   cout<<nodeCount<<"\n";
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 10:
   int data,pos;
   cout<<"Enter data : ";
   cin>>data;
```

```
            cout<<"Enter position : ";
            cin>>pos;
            insertInBetween(head, data, pos);
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 11:
            exit(0);
            break;
        default:
            cout << "\nSelect Proper Option (1/2/3/4/5/6/7/8/9/10/11)" << endl;
        }
    }
    while(choice!=11);
    return 0;
}
```

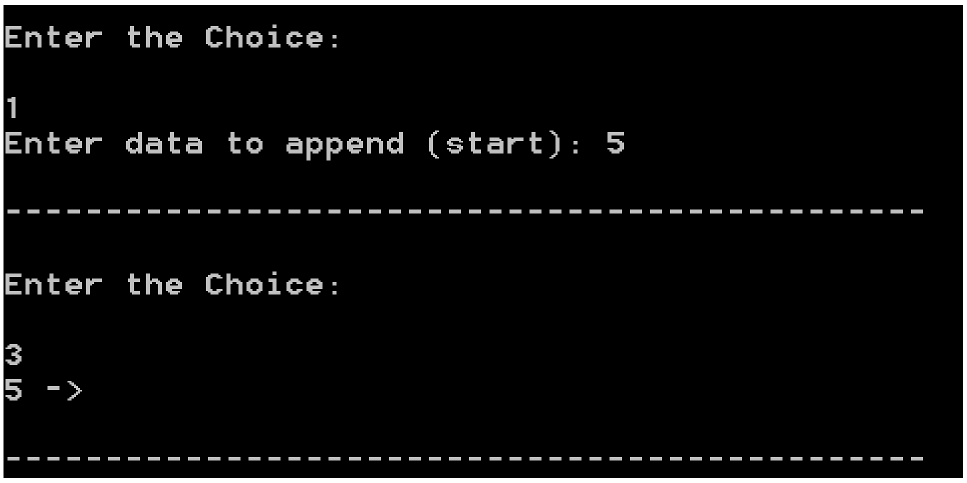
OUTPUT:



"C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\singlyLL.exe"

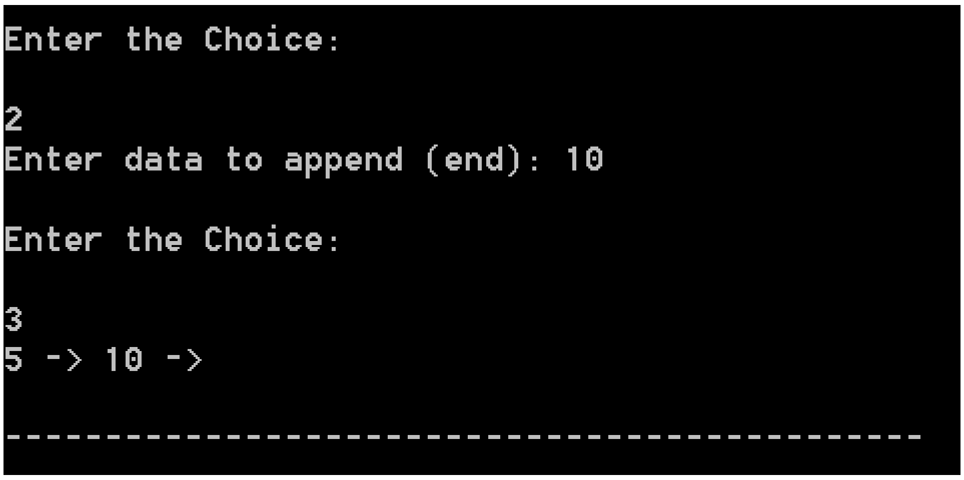
```
SINGLY LINKED LIST

---------------------------------------------

1.InsertFront
2.InsertEnd
3.Display
4.Search
5.Reverse
6.DeleteFront
7.DeleteEnd
8.Delete (by providing data)
9.Count
10.Insert in between
11.Exit
```

INSERT AT START:



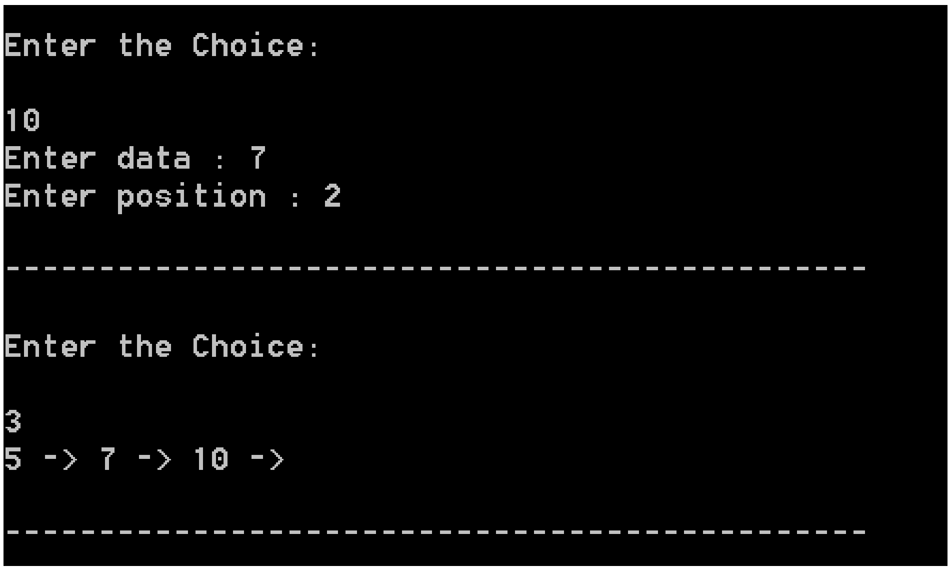
```
Enter the Choice:

1
Enter data to append (start): 5

---------------------------------------------

Enter the Choice:

3
5 ->

---------------------------------------------
```

INSERT AT END:



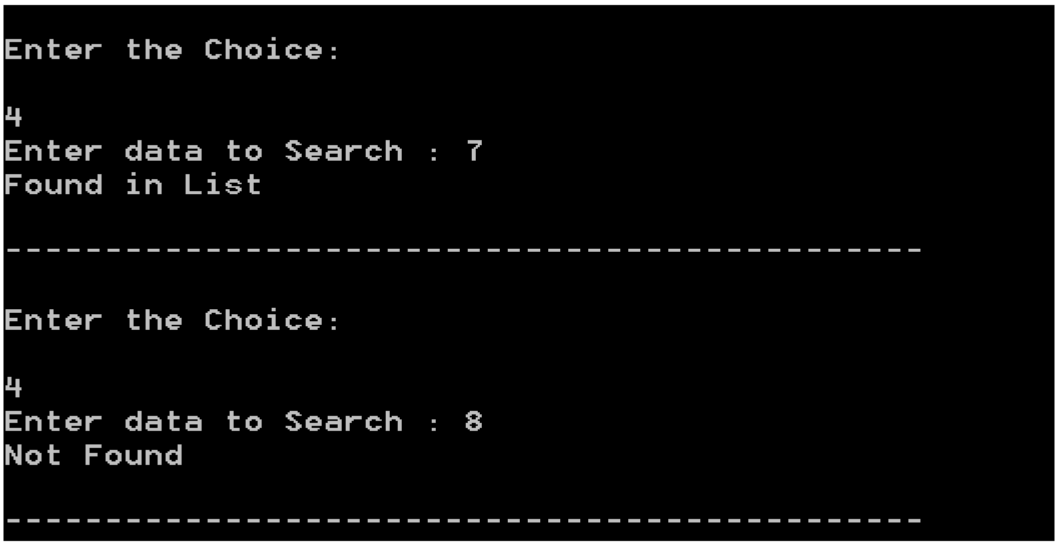
```
Enter the Choice:

2
Enter data to append (end): 10

Enter the Choice:

3
5 -> 10 ->

-----------------------------------------------
```

INSERT AT MIDDLE / SPECFIC POSITION:



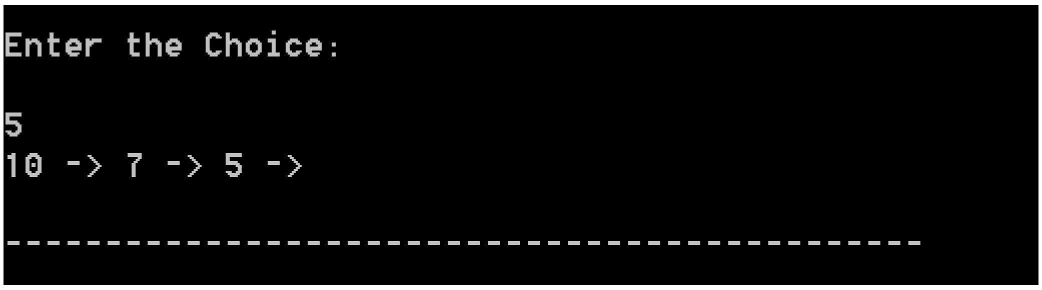
```
Enter the Choice:

10
Enter data : 7
Enter position : 2

-----------------------------------------------

Enter the Choice:

3
5 -> 7 -> 10 ->

-----------------------------------------------
```

SEARCH:



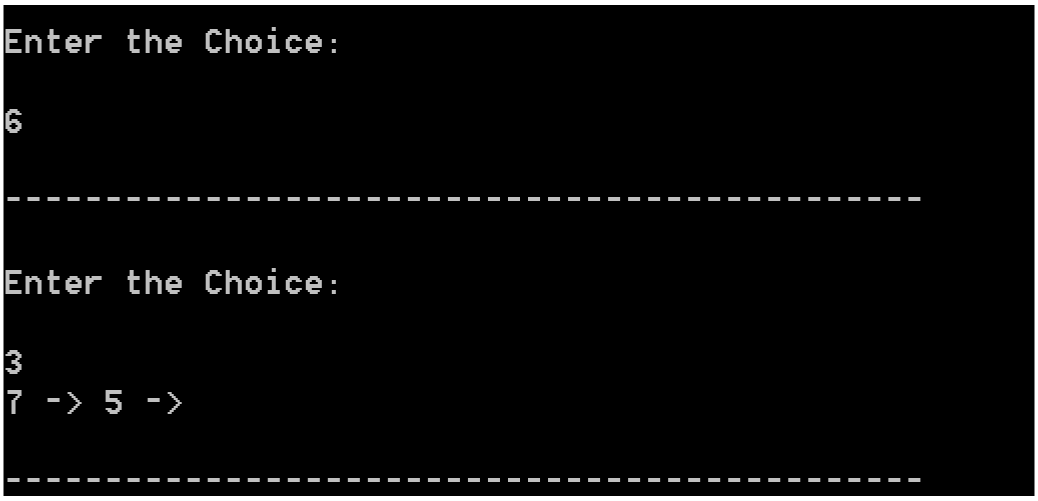
```
Enter the Choice:

4
Enter data to Search : 7
Found in List

-----------------------------------------------

Enter the Choice:

4
Enter data to Search : 8
Not Found

-----------------------------------------------
```

REVERSE:



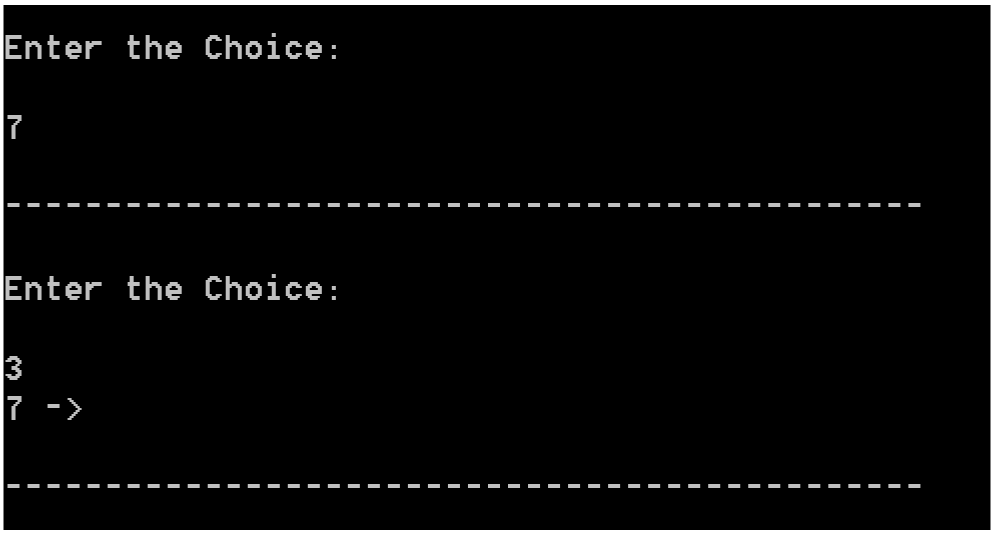
```
Enter the Choice:

5
10 -> 7 -> 5 ->

-----------------------------------------------
```

DELETE FROM FRONT:



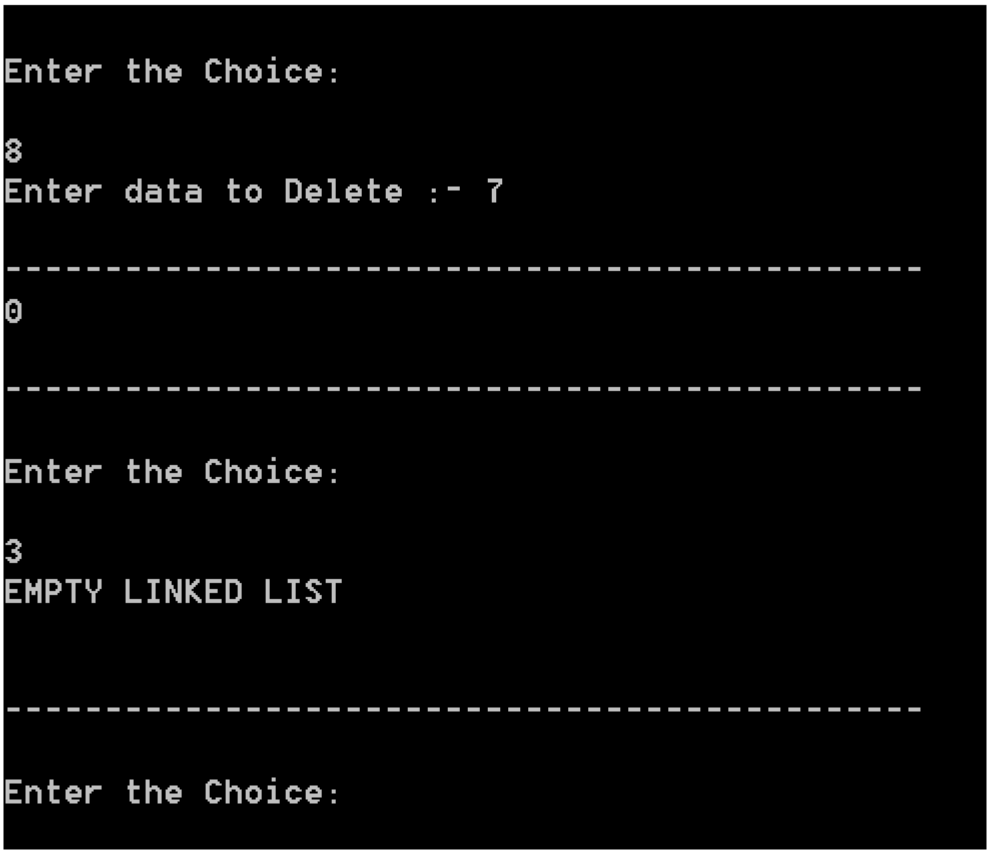
```
Enter the Choice:

6

-----------------------------------------------

Enter the Choice:

3
7 -> 5 ->

-----------------------------------------------
```

DELETE FROM END:



```
Enter the Choice:

7

-----------------------------------------------

Enter the Choice:

3
7 ->

-----------------------------------------------
```

DELETE SPECFIC DATA:



```
Enter the Choice:

8
Enter data to Delete :- 7

----------------------------------------------
0

----------------------------------------------

Enter the Choice:

3
EMPTY LINKED LIST


----------------------------------------------

Enter the Choice:
```

B) CIRCULAR LINKED LIST:

SOURCE CODE:

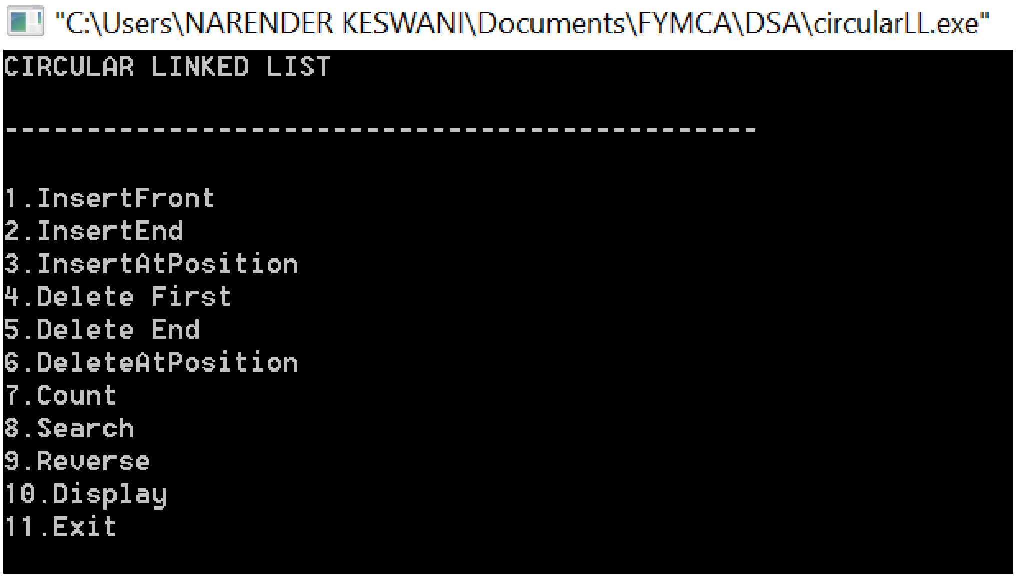
```
#include <iostream>
using namespace std;

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

class LinkedList
{
private:
   Node* head;
public:
   LinkedList()
   {
      head = NULL;
   }

   void InsertAtHead(int newElement)
   {
      Node* newNode = new Node();
      newNode->data = newElement;
      newNode->next = NULL;
      if(head == NULL)
      {
         head = newNode;
         newNode->next = head;
      }
      else
      {
         Node* temp = head;
         while(temp->next != head)
            temp = temp->next;
         temp->next = newNode;
         newNode->next = head;
         head = newNode;
      }
   }


   void InsertAtEnd(int newElement)
   {

      Node* newNode = new Node();

      newNode->data = newElement;

      newNode->next = NULL;
```

```
    if(head == NULL)
    {
      head = newNode;
      newNode->next = head;
    }
    else
    {

      Node* temp = head;
      while(temp->next != head)
        temp = temp->next;

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

  void InsertAtPosition(int newElement, int position)
  {

    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    Node* temp = head;
    int NoOfElements = 0;

    if(temp != NULL)
    {
      NoOfElements++;
      temp = temp->next;
    }
    while(temp != head)
    {
      NoOfElements++;
      temp = temp->next;
    }

    if(position < 1 || position > (NoOfElements+1))
    {
      cout<<"\nInavalid position."<<endl;
    }
    else if (position == 1)
    {

      if(head == NULL)
      {
        head = newNode;
        head->next = head;
      }
      else
      {
        while(temp->next != head)
        {
          temp = temp->next;
```

```
            }
            newNode->next = head;
            head = newNode;
            temp->next = head;
        }
    }
    else
    {


        temp = head;
        for(int i = 1; i < position-1; i++)
            temp = temp->next;
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

void display()
{
    Node* temp = head;
    if(temp != NULL)
    {
        cout<<"The list contains: ";
        while(true)
        {
            cout<<temp->data<<" ";
            temp = temp->next;
            if(temp == head)
                break;
        }
        cout<<endl;
    }
    else
    {
        cout<<"The list is empty.\n";
    }
}


void deleteFirst()
{
    if(head != NULL)
    {


        if(head->next == head)
        {
            head = NULL;
        }
        else
        {
            Node* temp = head;
            Node* firstNode = head;

            while(temp->next != head)
            {
              temp = temp->next;
            }


            head = head->next;
            temp->next = head;
            free(firstNode);
          }
        }
      }


      void deleteEnd()
      {
        if(head != NULL)
        {

          if(head->next == head)
          {
            head = NULL;
          }
          else
          {

            Node* temp = head;
            while(temp->next->next != head)
              temp = temp->next;


            Node* lastNode = temp->next;
            temp->next = head;
            free(lastNode);
          }
        }
      }

      void DeleteAtPosition(int position)
      {


        Node* nodeToDelete = head;
        Node* temp = head;
        int NoOfElements = 0;

        if(temp != NULL)
        {
          NoOfElements++;
          temp = temp->next;
        }
```

```
    while(temp != head)
    {
      NoOfElements++;
      temp = temp->next;
    }


    if(position < 1 || position > NoOfElements)
    {
      cout<<"\nInavalid position.";
    }
    else if (position == 1)
    {


      if(head->next == head)
      {
        head = NULL;
      }
      else
      {
        while(temp->next != head)
          temp = temp->next;
        head = head->next;
        temp->next = head;
        free(nodeToDelete);
      }
    }
    else
    {

      temp = head;
      for(int i = 1; i < position-1; i++)
        temp = temp->next;
      nodeToDelete = temp->next;
      temp->next = temp->next->next;
      free(nodeToDelete);
    }
  }

  int countNodes()
  {

    Node* temp = head;

    int i = 0;


    if(temp != NULL)
    {
      i++;
      temp = temp->next;
    }
    while(temp != head)
    {
        i++;
        temp = temp->next;
    }

    return i;
}

void SearchElement(int searchValue)
{

    Node* temp = head;
    int found = 0;
    int i = 0;


    if(temp != NULL)
    {
        while(true)
        {
            i++;
            if(temp->data == searchValue)
            {
                found++;
                break;
            }
            temp = temp->next;
            if(temp == head)
            {
                break;
            }
        }
        if (found == 1)
        {
            cout<<searchValue<<" is found at index = "<<i<<".\n";
        }
        else
        {
            cout<<searchValue<<" is not found in the list.\n";
        }
    }
    else
    {


        cout<<"The list is empty.\n";
    }
}

void reverseList()
{

    if(head != NULL)
    {
            Node* prevNode = head;
            Node* tempNode = head;
            Node* curNode = head->next;


            prevNode->next = prevNode;

            while(curNode != head)
            {

                tempNode = curNode->next;
                curNode->next = prevNode;
                head->next = curNode;
                prevNode = curNode;
                curNode = tempNode;
            }

            head = prevNode;
        }
    }

};


int main()
{
    int choice, data, location;
    LinkedList cll;

    cout<<"CIRCULAR LINKED LIST"<<endl;
    cout<<"\n---------------------------------------------"<<endl;
    cout<<"\n1.InsertFront\n2.InsertEnd\n3.InsertAtPosition\n4.Delete First\n5.Delete
End\n6.DeleteAtPosition\n7.Count\n8.Search\n9.Reverse\n10.Display\n11.Exit\n"<<endl;
    do
    {
        cout<<"\nEnter the Choice: \n"<<endl;
        cin>>choice;
        switch(choice)
        {
        case 1:
            cout<<"Enter data to append :- (start)";
            cin>>data;
            cll.InsertAtHead(data);
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 2:
            cout<<"Enter data to append :- (end)";
            cin>>data;
            cll.InsertAtEnd(data);
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 3:
            cout << "Enter data to be inserted: ";
            cin >> data;
            cout << "Enter location to be inserted into: ";
            cin >> location;
            cll.InsertAtPosition(data,location);
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 4:
            cll.deleteFirst();
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 5:
            cll.deleteEnd();
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 6:
            cout << "Enter location to be inserted into: ";
            cin >> location;
            cll.DeleteAtPosition(location);
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 7:
            cout<<cll.countNodes()<<endl;
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 8:
            cout<<"Enter data to Search :- ";
            cin>>data;
            cll.SearchElement(data);
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 9:
            cll.reverseList();
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 10:
            cll.display();
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 11:
            exit(1);
            break;
        default:
            cout << "\nSelect Proper Option (1/2/3/4/5/6/7/8/9/10/11)" << endl;
        }
    }
    while(choice!=11);
    return 0;
}
```

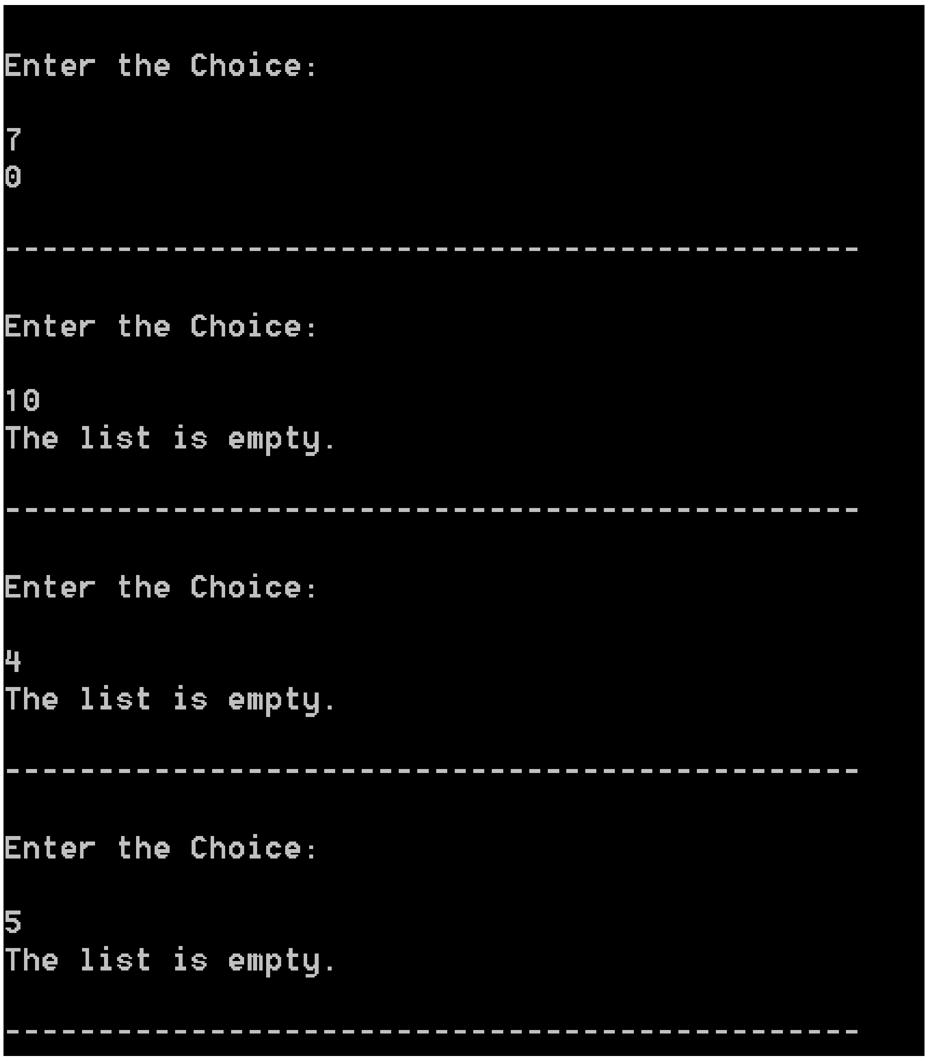
OUTPUT:



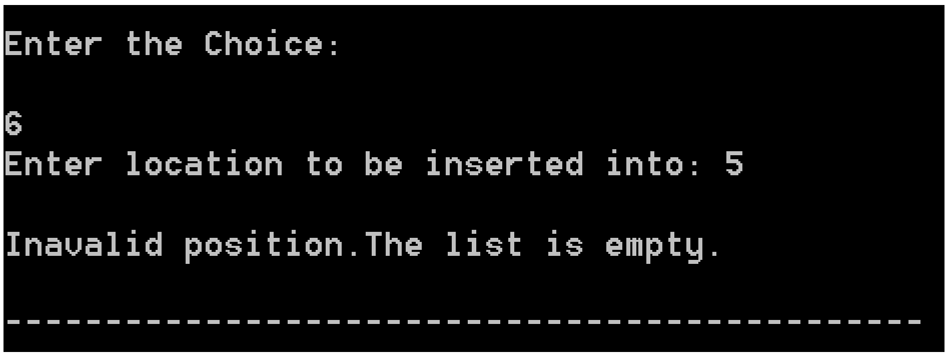
"C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\circularLL.exe"

```
CIRCULAR LINKED LIST

----------------------------------------------

1.InsertFront
2.InsertEnd
3.InsertAtPosition
4.Delete First
5.Delete End
6.DeleteAtPosition
7.Count
8.Search
9.Reverse
10.Display
11.Exit
```

NULL VALIDATION:

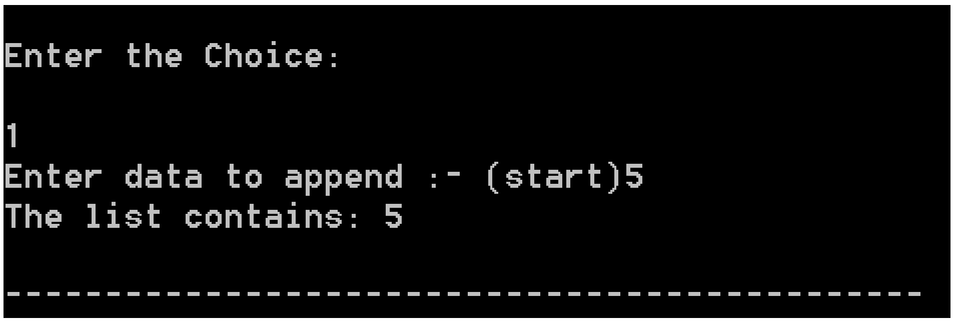


```

Enter the Choice:

7
0

----------------------------------------------

Enter the Choice:

10
The list is empty.

----------------------------------------------

Enter the Choice:

4
The list is empty.

----------------------------------------------

Enter the Choice:

5
The list is empty.

----------------------------------------------
```



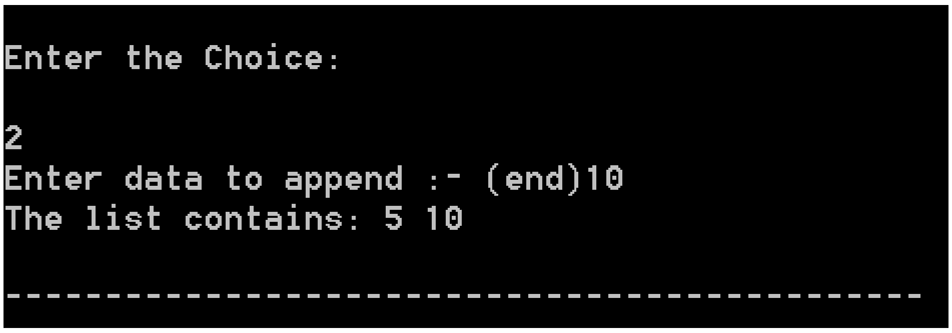
```
Enter the Choice:

6
Enter location to be inserted into: 5

Inavalid position.The list is empty.

---------------------------------------------
```

INSERT AT START:



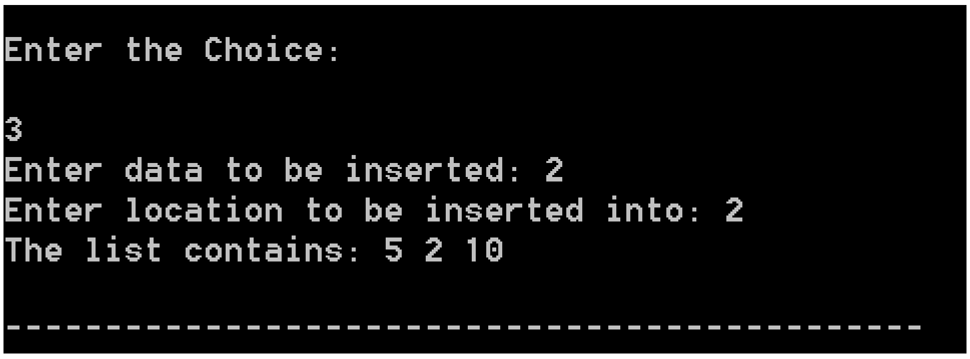
```
Enter the Choice:

1
Enter data to append :- (start)5
The list contains: 5

---------------------------------------------
```

INSERT AT END:



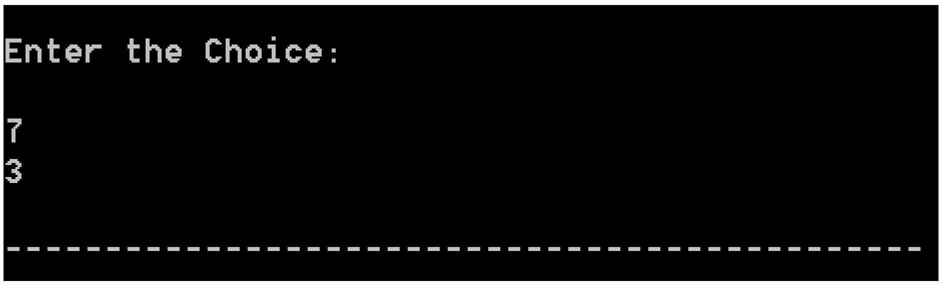
```
Enter the Choice:

2
Enter data to append :- (end)10
The list contains: 5 10

---------------------------------------------
```

INSERT AT POSITION:



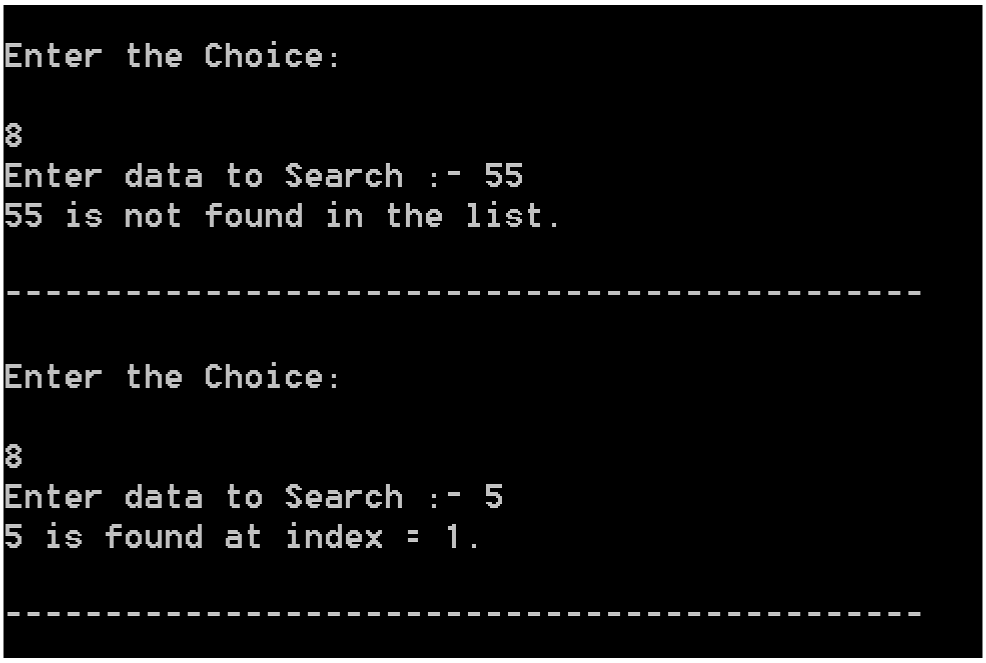
```
Enter the Choice:

3
Enter data to be inserted: 2
Enter location to be inserted into: 2
The list contains: 5 2 10

---------------------------------------------
```

COUNT:



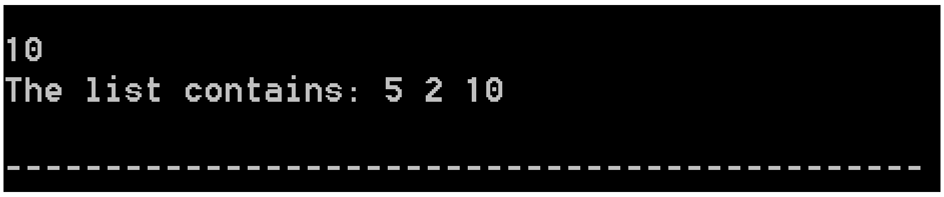
```
Enter the Choice:

7
3

---------------------------------------------
```

SEARCH:



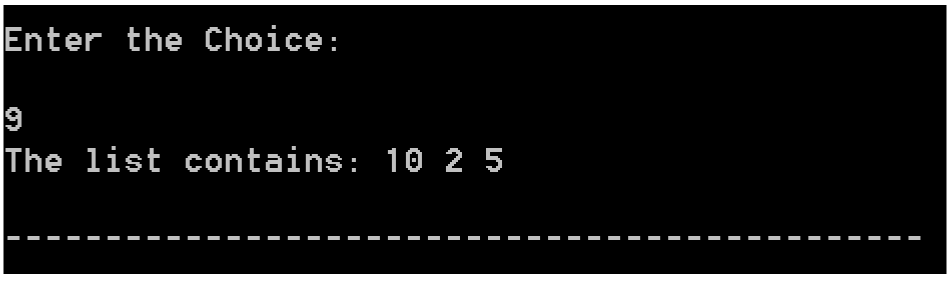
```
Enter the Choice:

8
Enter data to Search :- 55
55 is not found in the list.

---------------------------------------------

Enter the Choice:

8
Enter data to Search :- 5
5 is found at index = 1.

---------------------------------------------
```

PRINT:



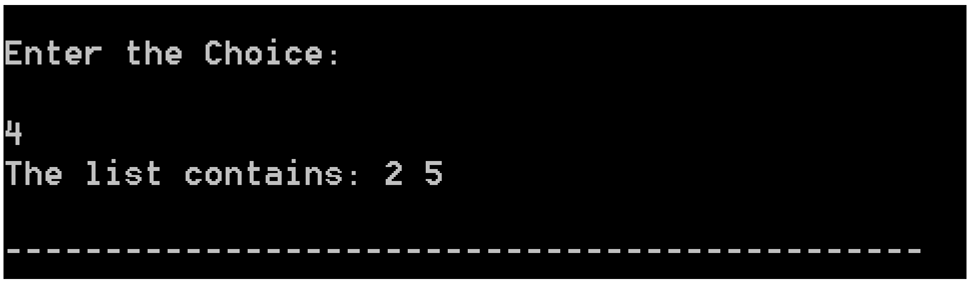
```
10
The list contains: 5 2 10

---------------------------------------------
```

REVERSE:

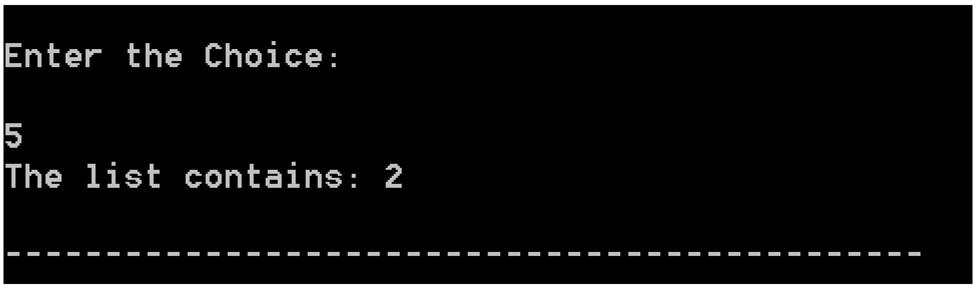


```
Enter the Choice:

9
The list contains: 10 2 5

---------------------------------------------
```

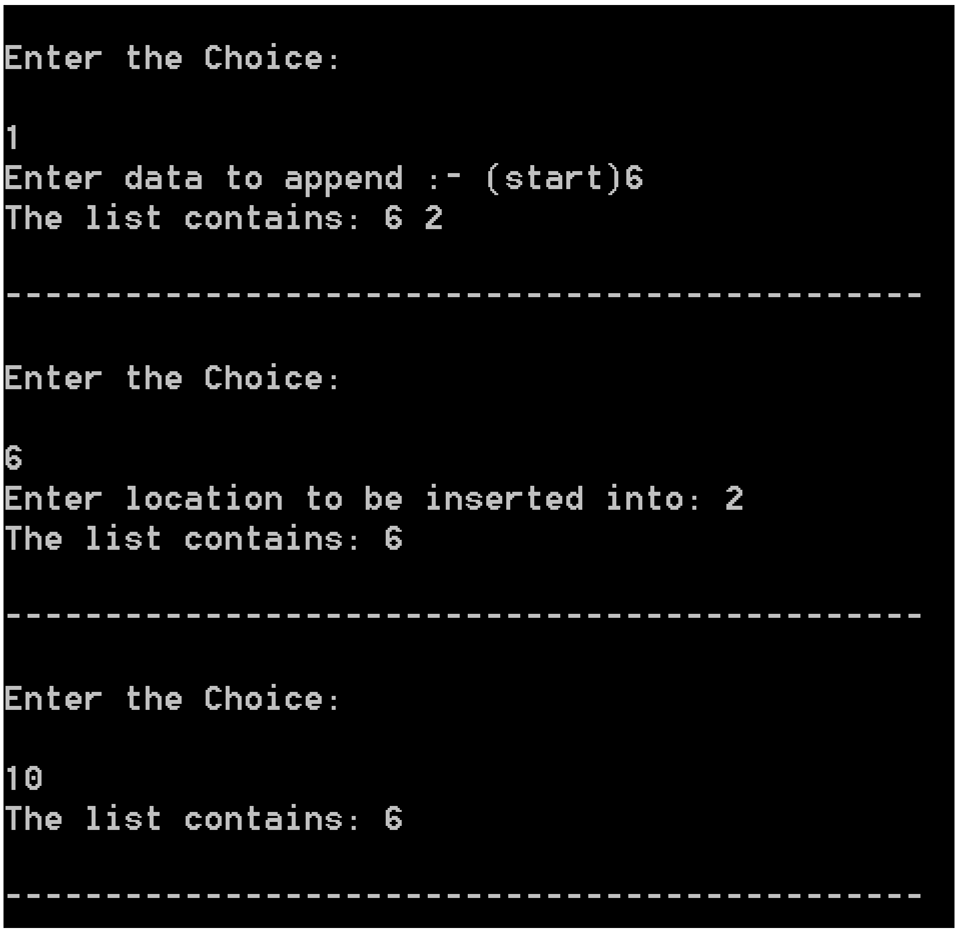
DELETE FRONT:


```
Enter the Choice:

4
The list contains: 2 5

----------------------------------------------
```

DELETE END:


```
Enter the Choice:

5
The list contains: 2

----------------------------------------------
```

DELETE AT POINT:


```
Enter the Choice:

1
Enter data to append :- (start)6
The list contains: 6 2

----------------------------------------------

Enter the Choice:

6
Enter location to be inserted into: 2
The list contains: 6

----------------------------------------------

Enter the Choice:

10
The list contains: 6

----------------------------------------------
```

C) DOUBLY LINKED LIST:

SOURCE CODE:

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
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;
```

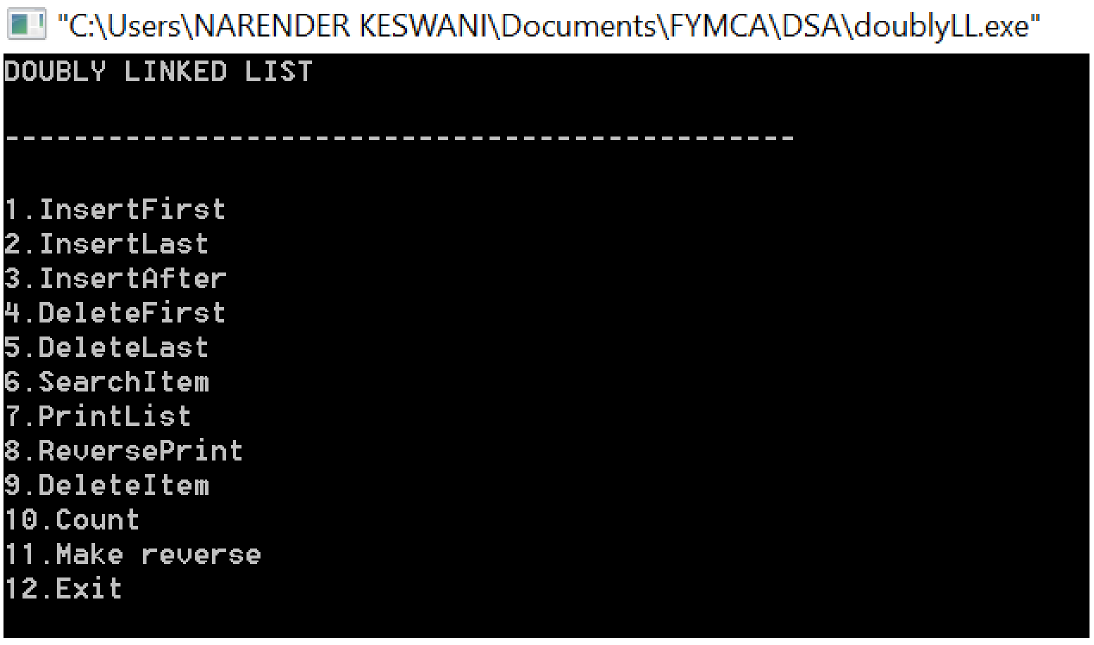
```
        newItem->prev=NULL;
        newItem->next=NULL;
        tail=newItem;
    }
    else
    {
        newItem->prev=tail;
        tail->next=newItem;
        newItem->next=NULL;
        tail=newItem;
    }
}

void insertAfter(int old, int element)
{
    struct node* newItem;
    newItem=new node;
    struct node* temp;
    temp=head;
    if(head==NULL)
    {
        return;
    }
    if(head==tail)
    {
        if(head->value!=old)
        {
            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)
    {
        cout<<temp->value<<"->";
        temp=temp->next;
    }
    puts("");
}

void printReverse()
{
    struct node* temp;
    temp=tail;
    while(temp!=NULL)
    {
        cout<<temp->value<<"->";
        temp=temp->prev;
    }
    cout<<endl;
}

void makereverse()
{
    struct node* prv=NULL;
    struct node* cur=head;
    struct node* nxt;
    while(cur!=NULL)
    {
        nxt=cur->next;
        cur->next=prv;
        prv=cur;
        cur=nxt;
    }
    head=prv;
}

int countNodes()
{
    struct node* temp=head;
    int i = 0;
    while(temp != NULL)
    {
        i++;
        temp = temp->next;
    }
    return i;
}

int dltfrst()
{
    if(head==NULL)
    {
        return 0;
    }
    int prev;
    prev=head->value;
    if(head==tail)
    {
        struct node* cur;
        cur=head;
        head=NULL;
        tail=NULL;
        delete cur;
        return prev;
    }
    else
    {
        struct node* cur;
        cur=head;
        head=head->next;
        head->prev=NULL;
        delete cur;
        return prev;
    }
}
int dltlast()
{
    if(head==NULL) return 0;
    int prev;
    prev=tail->value;
    if(head==tail)
    {
        struct node* cur;
        cur=head;
        head=NULL;
        tail=NULL;
        delete cur;
        return prev;
    }
    else
    {
        struct node* cur;
        cur=tail;
        tail=tail->prev;
        tail->next=NULL;
        delete cur;
        return prev;
    }
}

int main()
{
    init();
    int choice;

    cout<<"DOUBLY LINKED LIST"<<endl;
    cout<<"\n---------------------------------------------"<<endl;

cout<<"\n1.InsertFirst\n2.InsertLast\n3.InsertAfter\n4.DeleteFirst\n5.DeleteLast\n6.Sear
chItem\n7.PrintList\n8.ReversePrint\n9.DeleteItem\n10.Count\n11.Make
reverse\n12.Exit\n"<<endl;
    do
    {
        cout<<"\nEnter the Choice: \n"<<endl;
        cin>>choice;
        switch(choice)
        {
        case 1:
            {
            int elementStart;
            cout<<"Enter data to append :- (start)";
            cin>>elementStart;
            insertFirst(elementStart);
            printList();
            cout<<"\n---------------------------------------------"<<endl;
            break;
            }
        case 2:
            {
            int elementEnd;
            cout<<"Enter data to append :- (end)";
            cin>>elementEnd;
            insertLast(elementEnd);
            printList();
            cout<<"\n---------------------------------------------"<<endl;
            break;
            }
        case 3:
            {
            int old,newitem;
            cout << "Enter data to be inserted: ";
            cout<<"Enter Old Item_";
            cin>>old;
            cout<<"Enter new Item_";
            cin>>newitem;
            insertAfter(old,newitem);
            printList();
            cout<<"\n---------------------------------------------"<<endl;
            break;
            }
        case 4:
```

```
  {
  deleteFirst();
  printList();
  cout<<"\n---------------------------------------------"<<endl;
  break;
  }
case 5:
  {
  deleteLast();
  printList();
  cout<<"\n---------------------------------------------"<<endl;
  break;
  }
case 6:
  {
  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;
  }
  cout<<"\n---------------------------------------------"<<endl;
  break;
  }
case 7:
  {
  printList();
  cout<<"\n---------------------------------------------"<<endl;
  break;
  }
case 8:
  {
  printReverse();
  printList();
  cout<<"\n---------------------------------------------"<<endl;
  break;
  }
case 9:
  {
  int element;
  cout<<"Enter element to delete "<<endl;
  cin>>element;
  deleteItem(element);
  printList();
  cout<<"\n---------------------------------------------"<<endl;
  break;
  }
case 10:
```

```
            {
            cout<<countNodes()<<endl;
            cout<<"\n---------------------------------------------"<<endl;
            break;
            }
        case 11:
            {
            makereverse();
            printList();
            cout<<"\n---------------------------------------------"<<endl;
            break;
            }
        case 12:
            {
            exit(1);
            break;
            }
        default:
            cout << "\nSelect Proper Option (1/2/3/4/5/6/7/8/9/10/11/12)" << endl;
        }
    }
    while(choice!=12);
    return 0;
}
```

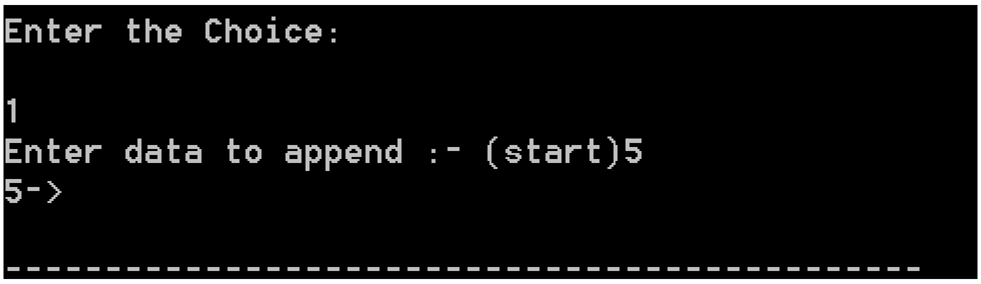
OUTPUT:



"C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\doublyLL.exe"

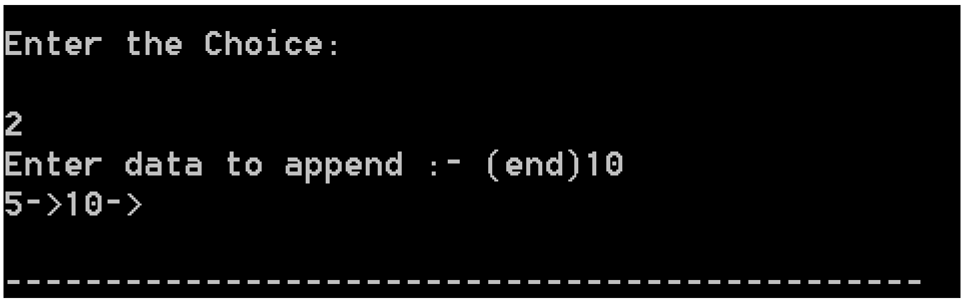
```
DOUBLY LINKED LIST

---------------------------------------------

1.InsertFirst
2.InsertLast
3.InsertAfter
4.DeleteFirst
5.DeleteLast
6.SearchItem
7.PrintList
8.ReversePrint
9.DeleteItem
10.Count
11.Make reverse
12.Exit
```

INSERT FIRST:



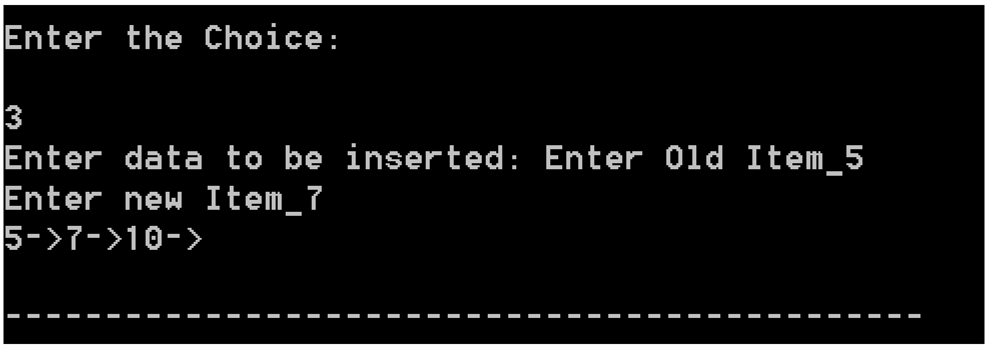
```
Enter the Choice:

1
Enter data to append :- (start)5
5->

-------------------------------------------------
```

INSERT LAST:



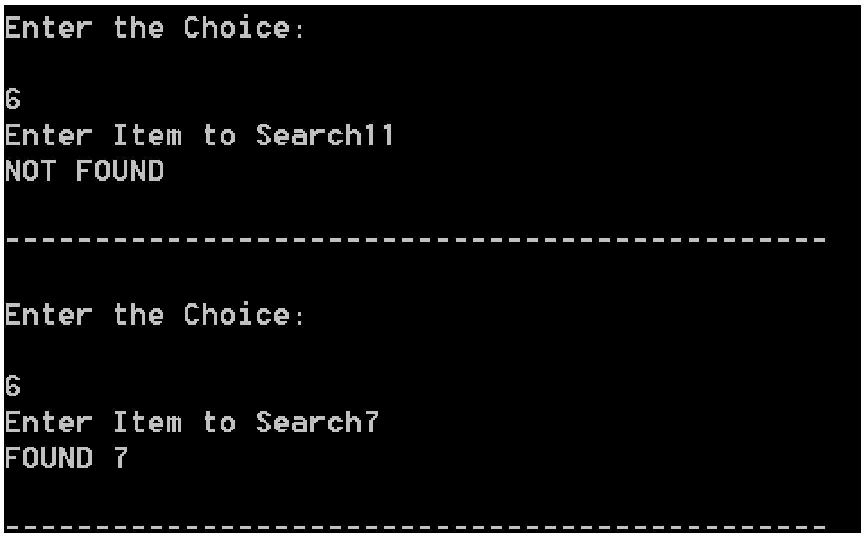
```
Enter the Choice:

2
Enter data to append :- (end)10
5->10->

-------------------------------------------------
```

INSERT AFTER SPECFIC VALUE:



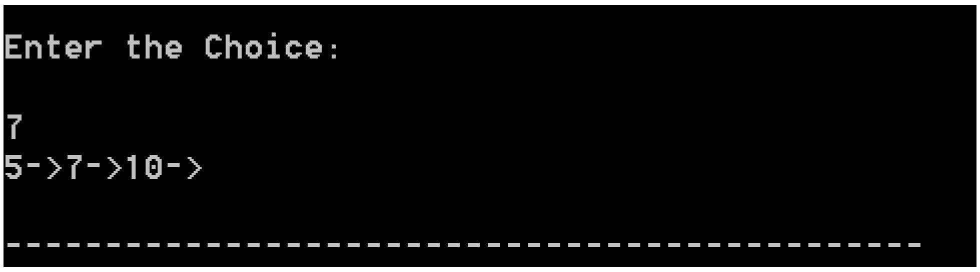
```
Enter the Choice:

3
Enter data to be inserted: Enter Old Item_5
Enter new Item_7
5->7->10->

-------------------------------------------------
```

SEARCH:



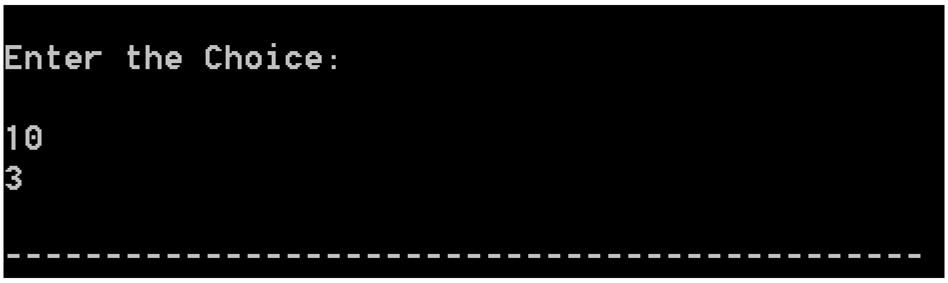
```
Enter the Choice:

6
Enter Item to Search11
NOT FOUND

-----------------------------------------------

Enter the Choice:

6
Enter Item to Search7
FOUND 7

-----------------------------------------------
```

PRINT / DISPLAY:



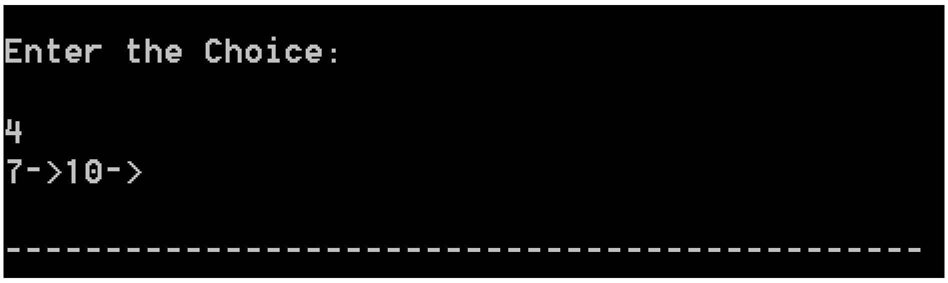
```
Enter the Choice:

7
5->7->10->

-------------------------------------------------
```

COUNT:



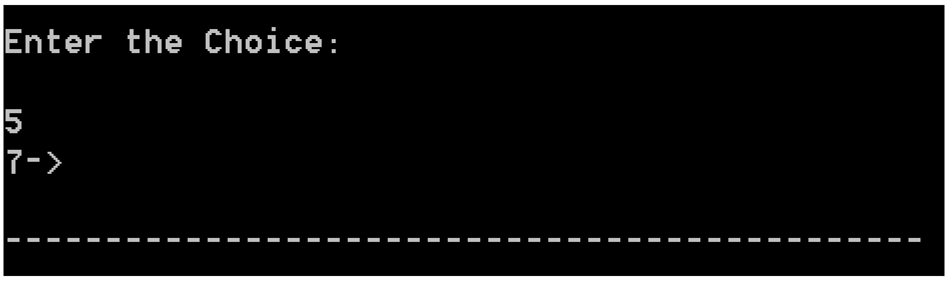
```
Enter the Choice:

10
3

-------------------------------------------------
```

DELETE FIRST:



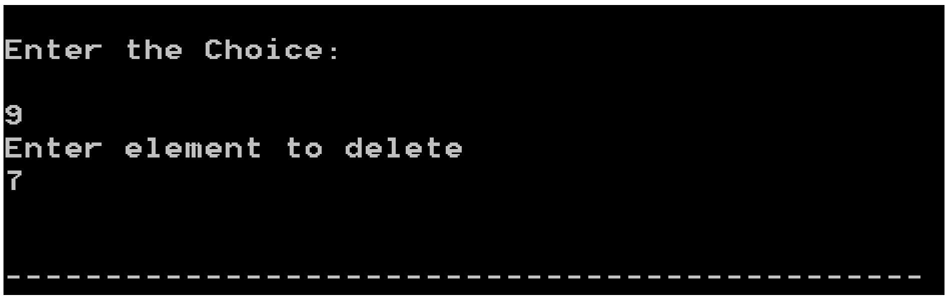
```
Enter the Choice:

4
7->10->

-------------------------------------------------
```

DELETE LAST:



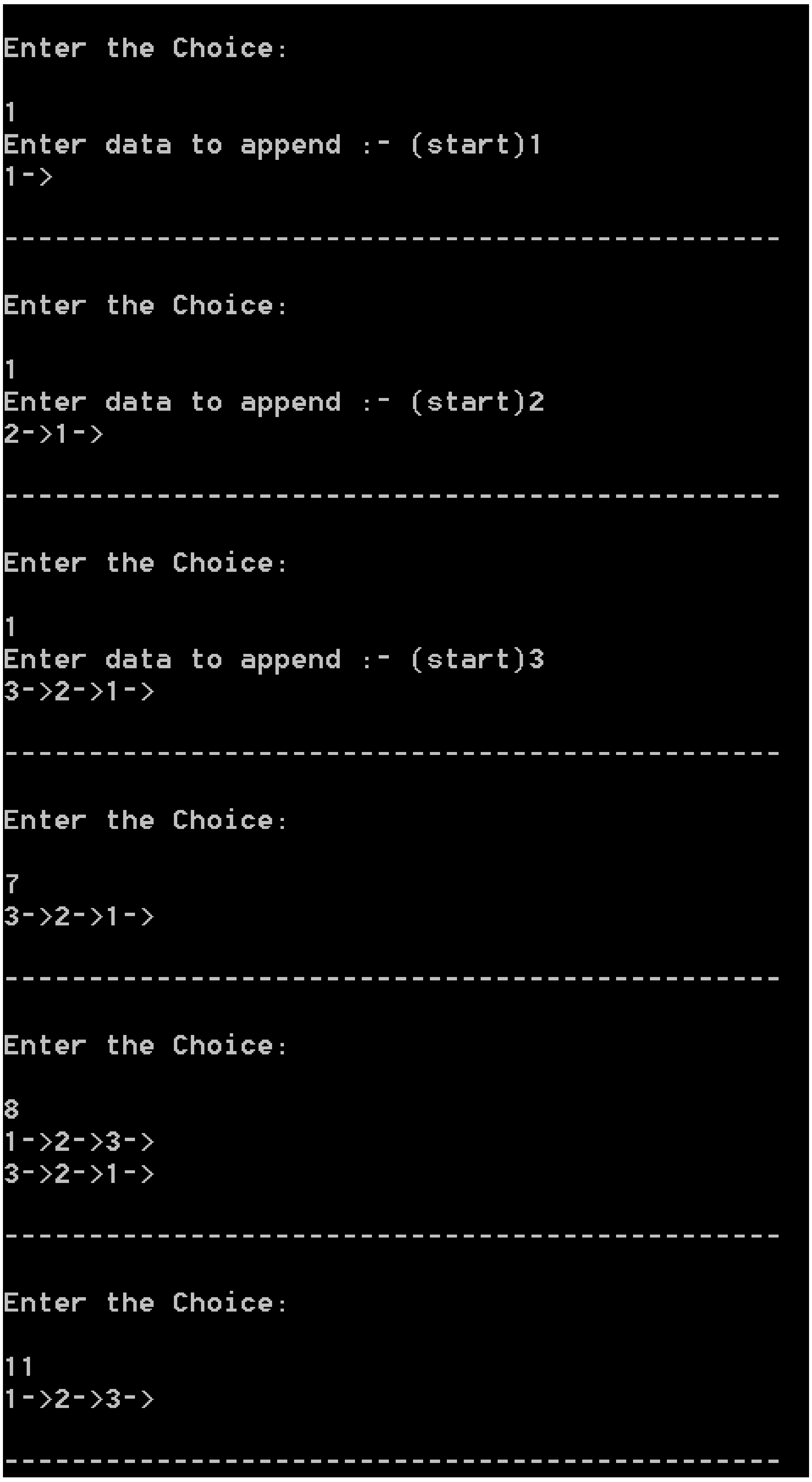
```
Enter the Choice:

5
7->

-------------------------------------------------
```

DELETE SPECFIC VALUE:



```
Enter the Choice:

9
Enter element to delete
7


-------------------------------------------------
```

INSERT & REVERSE:



```
Enter the Choice:

1
Enter data to append :- (start)1
1->

----------------------------------------------

Enter the Choice:

1
Enter data to append :- (start)2
2->1->

----------------------------------------------

Enter the Choice:

1
Enter data to append :- (start)3
3->2->1->

----------------------------------------------

Enter the Choice:

7
3->2->1->

----------------------------------------------

Enter the Choice:

8
1->2->3->
3->2->1->

----------------------------------------------

Enter the Choice:

11
1->2->3->

----------------------------------------------
```

CONCLUSION:

From this practical, I have learned how to implement singly, doubly, circular linked list.