UNIVERSITY OF DELHI

COLLEGE OF VOCATIONAL STUDIES

BSC (HONS) COMPUTER SCIENCE

SEMESTER - 3

DATA STRUCTURES USING C++

ANUJ KUMAR SEN

2K21/CS/15

Question 1 :

Given a list of N elements, which follows no particular arrangement, you are required to search an element x in the list. The list is stored using array data structure. If the search is successful, the output should be the index at which the element occurs, otherwise returns -1 to indicate that the element is not present in the list. Assume that the elements of the list are all distinct. Write a program to perform the desired task.

Solution 1 :

```
/*Note : Since datatype of the array is not mentioned, we will create a
Generic program using template function. Also, as no particular
arrangement(ascending or descending) has been followed in the array, we will
use
LINEAR SEARCH.*/

/*Created By - ANUJ KUMAR SEN */

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

template <class T>
class Practical1
{
public:
    int N;
    T *arr;
    void createArray()
    {
        cout << "Enter length of array" << endl;
        cin >> N;
        arr = new T[N];

        cout << "Enter array elements... " << endl;
        for (int i = 0; i < N; i++)
        {
            cout << "Element at index " << i << " : ";
            cin >> arr[i];
        }

        cout << "Printing the array..." << endl;
        for (int i = 0; i < N; i++)
        {
            cout << arr[i] << ' ';
```

```
        }
        cout << endl;
    }
    int LinearSearch()
    {
        T x;
        cout << "Enter element to be searched : ";
        cin >> x;

        for (int i = 0; i < N; i++)
        {
            if (arr[i] == x)
            {
                return i;
            }
        }
        return -1;
    }
};
int main()
{
    cout<<"Choose datatype of the elements you wish to enter"<<endl;
    cout << "Press 1. int" << endl;
    cout << "Press 2. char" << endl;
    cout << "Press 3. float" << endl;
    cout << "Press 4. double" << endl;
    cout << "Press 5. string" << endl;

    int ch, result;
    cin >> ch;
    switch (ch)
    {
    case 1:
    {
        Practical1<int> ob;
        ob.createArray();
        result = ob.LinearSearch();
        break;
    }
    case 2:
    {
        Practical1<char> ob;
        ob.createArray();
        result = ob.LinearSearch();
        break;
```

```cpp
    }
    case 3:
    {
        Practical1<float> ob;
        ob.createArray();
        result = ob.LinearSearch();
        break;
    }
    case 4:
    {
        Practical1<double> ob;
        ob.createArray();
        result = ob.LinearSearch();
        break;
    }
    case 5:
    {
        Practical1<string> ob;
        ob.createArray();
        result = ob.LinearSearch();
        break;
    }
    default:
    {
        cout << "Wrong choice" << endl;
        exit(0);
    }
    }

    if (result == -1)
    {
        cout << "Element is not present in the list" << endl;
    }
    else
    {
        cout << "Element is present in the list at index " << result << endl;
    }
    return 0;
}
```

Output 1 :

```
Choose datatype of the elements you wish to enter
Press 1. int
Press 2. char
Press 3. float
Press 4. double
Press 5. string
1
Enter length of array
7
Enter array elements...
Element at index 0 : 2
Element at index 1 : 0
Element at index 2 : 2
Element at index 3 : 1
Element at index 4 : 1
Element at index 5 : 0
Element at index 6 : 4
Printing the array...
2 0 2 1 1 0 4
Enter element to be searched : 7
Element is not present in the list
PS E:\Data Structures\Guidelines>
```

```
Choose datatype of the elements you wish to enter
Press 1. int
Press 2. char
Press 3. float
Press 4. double
Press 5. string
2
Enter length of array
3
Enter array elements...
Element at index 0 : f
Element at index 1 : d
Element at index 2 : n
Printing the array...
f d n
Enter element to be searched : d
Element is present in the list at index 1
```

Question 2 :

Given a list of N elements, which is sorted in ascending order, you are required to search an element x in the list. The list is stored using array data structure. If the search is successful, the output should be the index at which the element occurs, otherwise returns -1 to indicate that the element is not present in the list. Assume that the elements of the list are all distinct. Write a program to perform the desired task.

Solution 2 :

```
/*Note : Since datatype of the array is not mentioned, we will create a
Generic program using template function. Also, as the array is in ascending
order, we will use BINARY SEARCH.*/

/*Created By - ANUJ KUMAR SEN */

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

template <class T>
class Practical2
{
public:
    int N;
    T *arr;
    void createArray()
    {
        cout << "Enter length of array" << endl;
        cin >> N;
        arr = new T[N];

        cout << "Enter array elements... " << endl;
        for (int i = 0; i < N; i++)
        {
            cout << "Element at index " << i << " : ";
            cin >> arr[i];
        }

        cout << "Printing the array..." << endl;
        for (int i = 0; i < N; i++)
        {
            cout << arr[i] << ' ';
        }
        cout << endl;
```

```cpp
    }
    int BinarySearch()
    {
        T x;
        cout << "Enter element to be searched : ";
        cin >> 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;
    }
};
int main()
{
    cout << "Choose datatype of the elements you wish to enter" << endl;
    cout << "Press 1. int" << endl;
    cout << "Press 2. char" << endl;
    cout << "Press 3. float" << endl;
    cout << "Press 4. double" << endl;
    cout << "Press 5. string" << endl;

    int ch, result;
    cin >> ch;
    switch (ch)
    {
    case 1:
    {
        Practical2<int> ob;
        ob.createArray();
        result = ob.BinarySearch();
        break;
    }
    case 2:
    {
        Practical2<char> ob;
```

```
        ob.createArray();
        result = ob.BinarySearch();
        break;
    }
    case 3:
    {
        Practical2<float> ob;
        ob.createArray();
        result = ob.BinarySearch();
        break;
    }
    case 4:
    {
        Practical2<double> ob;
        ob.createArray();
        result = ob.BinarySearch();
        break;
    }
    case 5:
    {
        Practical2<string> ob;
        ob.createArray();
        result = ob.BinarySearch();
        break;
    }
    default:
    {
        cout << "Wrong choice" << endl;
        exit(0);
    }
    }

    if (result == -1)
    {
        cout << "Element is not present in the list" << endl;
    }
    else
    {
        cout << "Element is present in the list at index " << result << endl;
    }
    return 0;
}
```

Output 2 :

```
Choose datatype of the elements you wish to enter
Press 1. int
Press 2. char
Press 3. float
Press 4. double
Press 5. string
3
Enter length of array
4
Enter array elements...
Element at index 0 : 3.6
Element at index 1 : 2.8
Element at index 2 : 4.1
Element at index 3 : 9.66
Printing the array...
3.6 2.8 4.1 9.66
Enter element to be searched : 4.1
Element is present in the list at index 2
```

```
Choose datatype of the elements you wish to enter
Press 1. int
Press 2. char
Press 3. float
Press 4. double
Press 5. string
5
Enter length of array
4
Enter array elements...
Element at index 0 : Live
Element at index 1 : and
Element at index 2 : let
Element at index 3 : live
Printing the array...
Live and let live
Enter element to be searched : live
Element is present in the list at index 3
```

Question 3 :

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

Solution 3 :

```
/*Created By - ANUJ KUMAR SEN */

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

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

class Practical3
{
private:
    Node *head;

public:
    Node *flag = new Node();
    Practical3()
    {
        head = NULL;
    }

    void insertAtBeginning(int newElement)
    {
        Node *newNode = new Node();
        newNode->data = newElement;
        newNode->next = NULL;
        if (head == NULL)
        {
            head = newNode;
```

```
        }
        else
        {
            newNode->next = head;
            head = newNode;
        }
    }
    void insertAtEnd(int newElement)
    {
        Node *newNode = new Node();
        newNode->data = newElement;
        newNode->next = NULL;

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

        if (position < 1)
        {
            cout << "Position should be greater than 1" << endl;
        }
        else if (position == 1)
        {
            newNode->next = head;
            head = newNode;
        }
        else
        {
            Node *temp = head;
```

```
        for (int i = 1; i < position - 1; i++)
        {
            if (temp != NULL)
            {
                temp = temp->next;
            }
        }

        if (temp != NULL)
        {
            newNode->next = temp->next;
            temp->next = newNode;
        }
        else
        {
            cout << "The previous node is NULL" << endl;
        }
    }
}
void deleteAtBeginning()
{
    if (head != NULL)
    {
        Node *temp = head;
        head = head->next;
        free(temp);
    }
}
void deleteAtPosition(int position)
{
    if (position < 1)
    {
        cout << "Position should be >=1." << endl;
    }
    else if (position == 1 && head != NULL)
    {
        Node *temp = head;
        head = head->next;
        free(temp);
    }
    else
    {
        Node *temp = head;
        for (int i = 1; i < position - 1; i++)
        {
```

```
            if (temp != NULL)
            {
                temp = temp->next;
            }
        }
        if (temp != NULL && temp->next != NULL)
        {
            Node *nodeToDelete = temp->next;
            temp->next = temp->next->next;
            free(nodeToDelete);
        }
        else
        {
            cout << "The node is already full" << endl;
        }
    }
}
void deleteAtEnd()
{
    if (head != NULL)
    {
        if (head->next == NULL)
        {
            head = NULL;
        }
        else
        {
            Node *temp = head;
            while (temp->next->next != NULL)
            {
                temp = temp->next;
            }
            Node *lastNode = temp->next;
            temp->next = NULL;
            free(lastNode);
        }
    }
}
Node *search(int x)
{
    Node *temp = head;
    int pos = -1;
    if (temp != NULL)
    {
        while (temp != NULL)
```

```
            {
                pos++;
                if (temp->data == x)
                {
                    flag->data = pos;
                    return temp;
                }
                temp = temp->next;
            }
        }
        return flag;
    }
    void runSearch(int x)
    {
        Node *result = search(x);
        if (result == flag)
        {
            cout << "Element " << x << " is not found in the list" << endl;
        }
        else
        {
            cout << "Element " << x << " is found in the list at index " <<
flag->data << " in the list at address " << result << endl;
        }
    }

    Node *makeList()
    {
        Node *h = NULL;

        int elem;
        int run = 1;
        do
        {
            cout << "Enter element : ";
            cin >> elem;

            Node *node = new Node();
            node->data = elem;
            node->next = NULL;

            if (h == NULL)
            {
                h = node;
            }
```

```
            else
            {
                Node *temp = h;
                while (temp->next != NULL)
                {
                    temp = temp->next;
                }
                temp->next = node;
            }

            cout << "To continue... Enter 1 (To exit : Enter any
integer(only) except 1) : ";
            cin >> run;
            if (run != 1)
            {
                break;
            }
        } while (true);
        return h;
    }
    Node *concatenate(Node *node1, Node *node2)
    {
        if (node1->next == NULL)
        {
            node1->next = node2;
        }
        else
        {
            concatenate(node1->next, node2);
        }
        return node1;
    }

    // Function overloading
    void printList(Node *list)
    {
        Node *temp = list;
        if (temp != NULL)
        {
            while (temp != NULL)
            {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
```

```
        }
        else
        {
            cout << "The linked list is EMPTY" << endl;
        }
    }
    void runConcatenate()
    {
        cout << "\nMaking new lists..." << endl;
        cout << "Enter values for List 1" << endl;
        Node *list1 = makeList();
        cout << "\nEnter values for List 2" << endl;
        Node *list2 = makeList();

        cout << "List 1 : ";
        printList(list1);

        cout << "List 2 : ";
        printList(list2);

        Node *conctenatedList = concatenate(list1, list2);

        cout << "Concatenated list : ";
        printList(conctenatedList);
    }
    void printList()
    {
        Node *temp = head;
        if (temp != NULL)
        {
            cout << "The list contains : ";
            while (temp != NULL)
            {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
        else
        {
            cout << "The singly linked list is EMPTY" << endl;
        }
    }
};
int main()
```

```
{

    Practical3 LinkedList;
    int choice;
    cout << "\nEnter your choice" << endl;
    do
    {
        cout << "1 - Insert an element x at the beginning of the singly
linked list" << endl;
        cout << "2 - Insert an element x at position in the singly linked
list" << endl;
        cout << "3 - Remove an element from the beginning of the singly
linked list" << endl;
        cout << "4 - Remove an element from position in the singly linked
list." << endl;
        cout << "5 - Search for an element x in the singly linked list and
return its pointer" << endl;
        cout << "6 - Concatenate two singly linked lists" << endl;
        cout << "7 - Print the list" << endl;
        cout << "8 - Exit" << endl;

        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            int elem;
            cout << "Enter an element : ";
            cin >> elem;
            LinkedList.insertAtBeginning(elem);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 2:
        {
            int elem, pos;
            cout << "Enter an element : ";
            cin >> elem;
            cout << "Enter position at which " << elem << " is to be
inserted : ";
            cin >> pos;
            LinkedList.insertAtPosition(elem, pos);
            cout << "Updated list : ";
```

```
            LinkedList.printList();
            break;
        }
        case 3:
        {
            LinkedList.deleteAtBeginning();
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 4:
        {
            int pos;
            cout << "Enter position from which element has to be deleted :
";
            cin >> pos;
            LinkedList.deleteAtPosition(pos);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 5:
        {
            int elem;
            cout << "Enter element to be searched : ";
            cin >> elem;
            LinkedList.runSearch(elem);
            break;
        }
        case 6:
        {
            LinkedList.runConcatenate();
            break;
        }
        case 7:
        {
            LinkedList.printList();
            break;
        }
        case 8:
        {
            cout << "Exiting..." << endl;
            break;
        }
        default:
```

```
        {
            cout << "Wrong choice...\nTry again...";
            break;
        }
        }

    } while (choice != 8);

    return 0;
}
```

Output 3 :

```
Enter your choice
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
1
Enter an element : 4
Updated list : The list contains : 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
1
Enter an element : 2
Updated list : The list contains : 2 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
1
Enter an element : 6
Updated list : The list contains : 6 2 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
```

```
Enter an element : 0
Enter position at which 0 is to be inserted : 1
Updated list : The list contains : 0 6 2 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
2
Enter an element : 8
Enter position at which 8 is to be inserted : 6
The previous node is NULL
Updated list : The list contains : 0 6 2 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
3
Updated list : The list contains : 6 2 4
```

```
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
4
Enter position from which element has to be deleted : 2
Updated list : The list contains : 6 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
5
Enter element to be searched : 0
Element 0 is not found in the list
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
5
Enter element to be searched : 6
Element 6 is found in the list at index 0 in the list at address 0x1fa530
```

```
6

Making new lists...
Enter values for List 1
Enter element : 4
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 5
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 0
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 0

Enter values for List 2
Enter element : 6
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 7
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 9
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 8
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 0
List 1 : 4 5 0
List 2 : 6 7 9 8
Concatenated list : 4 5 0 6 7 9 8
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
7
The list contains : 6 4
1 - Insert an element x at the beginning of the singly linked list
2 - Insert an element x at position in the singly linked list
3 - Remove an element from the beginning of the singly linked list
4 - Remove an element from position in the singly linked list.
5 - Search for an element x in the singly linked list and return its pointer
6 - Concatenate two singly linked lists
7 - Print the list
8 - Exit
8
Exiting...
```

Question 4 :

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

Solution 4 :

```
/*Created By - ANUJ KUMAR SEN */

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

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

class Practical4
{
private:
    Node *head;

public:
    Node *flag = new Node();
    Practical4()
    {
        head = NULL;
    }

    void insertAtBeginning(int newElement)
    {
        Node *newNode = new Node();
        newNode->data = newElement;
        newNode->next = NULL;
        newNode->prev = NULL;
        if (head == NULL)
        {
```

```
        head = newNode;
    }
    else
    {
        head->prev = newNode;
        newNode->next = head;
        head = newNode;
    }
}
void insertAtEnd(int newElement)
{
    Node *newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    newNode->prev = NULL;

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

    if (position < 1)
    {
        cout << "Position should be greater than 1" << endl;
    }
    else if (position == 1)
    {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
```

```
        }
        else
        {
            Node *temp = head;
            for (int i = 1; i < position - 1; i++)
            {
                if (temp != NULL)
                {
                    temp = temp->next;
                }
            }

            if (temp != NULL)
            {
                newNode->next = temp->next;
                newNode->prev = temp;
                temp->next = newNode;
                if (newNode->next != NULL)
                {
                    newNode->next->prev = newNode;
                }
            }
            else
            {
                cout << "The previous node is NULL" << endl;
            }
        }
    }
    void deleteAtBeginning()
    {
        if (head != NULL)
        {
            Node *temp = head;
            head = head->next;
            free(temp);
            if (head != NULL)
            {
                head->prev = NULL;
            }
        }
    }
    void deleteAtPosition(int position)
    {
        if (position < 1)
        {
            cout << "Position should be >=1." << endl;
        }
```

```cpp
    else if (position == 1 && head != NULL)
    {
        Node *temp = head;
        head = head->next;
        free(temp);
        if (head != NULL)
        {
            head->prev = NULL;
        }
    }
    else
    {
        Node *temp = head;
        for (int i = 1; i < position - 1; i++)
        {
            if (temp != NULL)
            {
                temp = temp->next;
            }
        }
        if (temp != NULL && temp->next != NULL)
        {
            Node *nodeToDelete = temp->next;
            temp->next = temp->next->next;
            if (temp->next->next != NULL)
            {
                temp->next->next->prev = temp->next;
            }

            free(nodeToDelete);
        }
        else
        {
            cout << "The node is already full" << endl;
        }
    }
}
void deleteAtEnd()
{
    if (head != NULL)
    {
        if (head->next == NULL)
        {
            head = NULL;
        }
        else
        {
```

```
                Node *temp = head;
                while (temp->next->next != NULL)
                {
                    temp = temp->next;
                }
                Node *lastNode = temp->next;
                temp->next = NULL;
                free(lastNode);
            }
        }
    }
    Node *search(int x)
    {
        Node *temp = head;
        int pos = -1;
        if (temp != NULL)
        {
            while (temp != NULL)
            {
                pos++;
                if (temp->data == x)
                {
                    flag->data = pos;
                    return temp;
                }
                temp = temp->next;
            }
        }
        return flag;
    }
    void runSearch(int x)
    {
        Node *result = search(x);
        if (result == flag)
        {
            cout << "Element " << x << " is not found in the list" << endl;
        }
        else
        {
            cout << "Element " << x << " is found in the list at index " <<
flag->data << " in the list at address " << result << endl;
        }
    }

    Node *makeList()
    {
        Node *h = NULL;
```

```
        int elem;
        int run = 1;
        do
        {
            cout << "Enter element : ";
            cin >> elem;

            Node *node = new Node();
            node->data = elem;
            node->next = NULL;
            node->prev = NULL;

            if (h == NULL)
            {
                h = node;
            }
            else
            {
                Node *temp = h;
                while (temp->next != NULL)
                {
                    temp = temp->next;
                }
                node->prev = temp;
                temp->next = node;
            }

            cout << "To continue... Enter 1 (To exit : Enter any integer(only)
except 1) : ";
            cin >> run;
            if (run != 1)
            {
                break;
            }
        } while (run == 1);
        return h;
    }
    Node *concatenate(Node *node1, Node *node2)
    {
        if (node1->next == NULL)
        {
            node1->next = node2;
            node2->prev = node1;
        }
        else
        {
```

```
        concatenate(node1->next, node2);
    }
    return node1;
}

// Function overloading
void printList(Node *list)
{
    Node *temp = list;
    if (temp != NULL)
    {
        while (temp != NULL)
        {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
    else
    {
        cout << "The linked list is EMPTY" << endl;
    }
}
void runConcatenate()
{
    cout << "\nMaking new lists..." << endl;
    cout << "Enter values for List 1" << endl;
    Node *list1 = makeList();
    cout << "\nEnter values for List 2" << endl;
    Node *list2 = makeList();

    cout << "List 1 : ";
    printList(list1);

    cout << "List 2 : ";
    printList(list2);

    Node *conctenatedList = concatenate(list1, list2);

    cout << "Concatenated list : ";
    printList(conctenatedList);
}

// function overloading
void printList()
{
    Node *temp = head;
```

```
        if (temp != NULL)
        {
            cout << "The list contains : ";
            while (temp != NULL)
            {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
        else
        {
            cout << "The doubly linked list is EMPTY" << endl;
        }
    }
};
int main()
{

    Practical4 LinkedList;
    int choice;
    cout << "\nEnter your choice" << endl;
    do
    {
        cout << "1 - Insert an element x at the beginning of the doubly linked
list" << endl;
        cout << "2 - Insert an element x at position in the doubly linked list" <<
endl;
        cout << "3 - Insert an element x at the end of the doubly linked list" <<
endl;
        cout << "4 - Remove an element from the beginning of the doubly linked
list" << endl;
        cout << "5 - Remove an element from position in the doubly linked list." <<
endl;
        cout << "6 - Remove an element from the end of the doubly linked list." <<
endl;
        cout << "7 - Search for an element x in the doubly linked list and return
its pointer" << endl;
        cout << "8 - Concatenate two doubly linked lists" << endl;
        cout << "9 - Print the list" << endl;
        cout << "10 - Exit" << endl;

        cin >> choice;

        switch (choice)
        {
        case 1:
```

```
        {
            int elem;
            cout << "Enter an element : ";
            cin >> elem;
            LinkedList.insertAtBeginning(elem);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 2:
        {
            int elem, pos;
            cout << "Enter an element : ";
            cin >> elem;
            cout << "Enter position at which " << elem << " is to be inserted : ";
            cin >> pos;
            LinkedList.insertAtPosition(elem, pos);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 3:
        {
            int elem;
            cout << "Enter an element : ";
            cin >> elem;
            LinkedList.insertAtEnd(elem);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 4:
        {
            LinkedList.deleteAtBeginning();
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 5:
        {
            int pos;
            cout << "Enter position from which element has to be deleted : ";
            cin >> pos;
            LinkedList.deleteAtPosition(pos);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
```

```
        }
        case 6:
        {
            LinkedList.deleteAtEnd();
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 7:
        {
            int elem;
            cout << "Enter element to be searched : ";
            cin >> elem;
            LinkedList.runSearch(elem);
            break;
        }
        case 8:
        {
            LinkedList.runConcatenate();
            break;
        }
        case 9:
        {
            LinkedList.printList();
            break;
        }
        case 10:
        {
            cout << "Exiting..." << endl;
            break;
        }
        default:
        {
            cout << "Wrong choice...\nTry again...";
            break;
        }
        }

    } while (choice != 10);

    return 0;
}
```

Output 4 :

```
Enter your choice
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
1
Enter an element : 5
Updated list : The list contains : 5
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
1
Enter an element : 4
Updated list : The list contains : 4 5
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
2
```

```
Enter an element : 1
Enter position at which 1 is to be inserted : 5
The previous node is NULL
Updated list : The list contains : 4 5
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
2
Enter an element : 1
Enter position at which 1 is to be inserted : 0
Position should be greater than 1
Updated list : The list contains : 4 5
```

```
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
3
Enter an element : 7
Updated list : The list contains : 4 5 7
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
4
Updated list : The list contains : 5 7
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
5
Enter position from which element has to be deleted : 1
Updated list : The list contains : 7
```

```
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
6
Updated list : The list contains : 6 2 5 7
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
7
Enter element to be searched : 5
Element 5 is found in the list at index 2 in the list at address 0x101ab28
```

```
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
8

Making new lists...
Enter values for List 1
Enter element : 4
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 7
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 5
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 0

Enter values for List 2
Enter element : 2
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 5
List 1 : 4 7 5
List 2 : 2
Concatenated list : 4 7 5 2
```

```
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
9
The list contains : 6 2 5 7
1 - Insert an element x at the beginning of the doubly linked list
2 - Insert an element x at position in the doubly linked list
3 - Insert an element x at the end of the doubly linked list
4 - Remove an element from the beginning of the doubly linked list
5 - Remove an element from position in the doubly linked list.
6 - Remove an element from the end of the doubly linked list.
7 - Search for an element x in the doubly linked list and return its pointer
8 - Concatenate two doubly linked lists
9 - Print the list
10 - Exit
10
Exiting...
```

Question 5 :

Write a program to implement circular 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

Solution 5 :

```
/*Created By - ANUJ KUMAR SEN */

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

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

class Practical5
{
private:
    Node *head;

public:
    Node *flag = new Node();
    Practical5()
    {
        head = NULL;
    }

    void insertAtBeginning(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 insertAfterElement(int elementX, int elementY)
    {
        Node *result = search(elementY);
        if (result == flag)
        {
            cout << "Element " << elementX << " cannot be inserted as Element " <<
elementY << " is not found in the list" << endl;
        }
        else
        {
            Node *elemX = new Node();
            elemX->data = elementX;
```

```
            elemX->next = NULL;

            if (result->next == head)
            {
                result->next = elemX;
                elemX->next = head;
            }
            else
            {
                elemX->next = result->next;
                result->next = elemX;
            }
        }
    }
    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 << "\nInvalid position.";
        }
        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 deleteAtBeginning()
{
    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 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);
    }
}
void deleteAtEnd()
{
    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 deleteElement(int x)
{
    Node *result = search(x);
    if (result == flag)
    {
        cout << "Element " << x << " is not found in the list" << endl;
    }
    else
    {

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

            free(result);
        }
        else
        {
            Node *temp = head;
            while (temp->next != result)
            {
                temp = temp->next;
```

```
            }
            if (result->next != head)
            {
                temp->next = result->next;
                free(result);
            }
            else
            {
                temp->next = head;
                free(result);
            }
        }
    }
}
Node *search(int x)
{
    Node *temp = head;
    int pos = -1;
    if (temp != NULL)
    {
        while (true)
        {
            pos++;
            if (temp->data == x)
            {
                flag->data = pos;
                return temp;
            }
            temp = temp->next;
            if (temp == head)
            {
                break;
            }
        }
    }
    return flag;
}
void runSearch(int x)
{
    Node *result = search(x);
    if (result == flag)
    {
        cout << "Element " << x << " is not found in the list" << endl;
    }
    else
    {
```

```
            cout << "Element " << x << " is found in the list at index " <<
flag->data << " in the list at address " << result << endl;
        }
    }

    Node *makeList()
    {
        Node *h = NULL;

        int elem;
        int run = 1;
        do
        {
            cout << "Enter element : ";
            cin >> elem;

            Node *newNode = new Node();
            newNode->data = elem;
            newNode->next = NULL;
            if (h == NULL)
            {
                h = newNode;
                newNode->next = h;
            }
            else
            {
                Node *temp = h;
                while (temp->next != h)
                {
                    temp = temp->next;
                }

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

            cout << "To continue... Enter 1 (To exit : Enter any integer(only)
except 1) : ";
            cin >> run;
            // if (run != 1)
            // {
            //     break;
            // }
        } while (run == 1);

        return h;
    }
```

```cpp
Node *concatenate(Node *node1, Node *node2)
{
    Node *temp = node1;
    while (temp->next != node1)
    {
        temp = temp->next;
    }
    temp->next = node2;

    Node *temp2 = node2;
    while (temp2->next != node2)
    {
        temp2 = temp2->next;
    }
    temp2->next = node1;

    return node1;
}

// Function overloading
void printList(Node *list)
{

    Node *temp = list;
    if (temp != NULL)
    {
        cout << "The list contains: ";
        while (true)
        {
            cout << temp->data << " ";
            temp = temp->next;
            if (temp == list)
                break;
        }
        cout << endl;
    }
    else
    {
        cout << "The list is empty.\n";
    }
}
void runConcatenate()
{
    cout << "\nMaking new lists..." << endl;
    cout << "Enter values for List 1" << endl;
    Node *list1 = makeList();
    cout << "\nEnter values for List 2" << endl;
```

```
        Node *list2 = makeList();

        cout << "List 1 : ";
        printList(list1);

        cout << "List 2 : ";
        printList(list2);

        Node *conctenatedList = concatenate(list1, list2);

        cout << "Concatenated list : ";
        printList(conctenatedList);
    }
    void printList()
    {
        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";
        }
    }
};
int main()
{

    Practical5 LinkedList;
    int choice;
    cout << "\nEnter your choice" << endl;
    do
    {
        cout << "1 - Insert an element x at the beginning of the circular linked
list" << endl;
        cout << "2 - Insert an element x after an element y in the circular linked
list" << endl;
```

```
        cout << "3 - Insert an element x at the end of the circular linked list" <<
endl;
        cout << "4 - Remove an element from the beginning of the circular linked
list" << endl;
        cout << "5 - Remove an element from the end of the circular linked list."
<< endl;
        cout << "6 - Remove an element x from the circular linked list." << endl;
        cout << "7 - Search for an element x in the circular linked list and return
its pointer" << endl;
        cout << "8 - Concatenate two circular linked lists" << endl;
        cout << "9 - Print the list" << endl;
        cout << "10 - Exit" << endl;

        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            int elem;
            cout << "Enter an element : ";
            cin >> elem;
            LinkedList.insertAtBeginning(elem);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 2:
        {
            int elemX, elemY;
            cout << "Enter elementX : ";
            cin >> elemX;
            cout << "Enter elementY : ";
            cin >> elemY;
            LinkedList.insertAfterElement(elemX, elemY);
            cout << "Updated list : ";
            LinkedList.printList();
            break;
        }
        case 3:
        {
            int elem;
            cout << "Enter an element : ";
            cin >> elem;
            LinkedList.insertAtEnd(elem);
            cout << "Updated list : ";
            LinkedList.printList();
```

```
                break;
            }
            case 4:
            {
                LinkedList.deleteAtBeginning();
                cout << "Updated list : ";
                LinkedList.printList();
                break;
            }
            case 5:
            {
                LinkedList.deleteAtEnd();
                cout << "Updated list : ";
                LinkedList.printList();
                break;
            }
            case 6:
            {
                int x;
                cout << "Enter element x : ";
                cin >> x;
                LinkedList.deleteElement(x);
                cout << "Updated list : ";
                LinkedList.printList();
                break;
            }
            case 7:
            {
                int elem;
                cout << "Enter element to be searched : ";
                cin >> elem;
                LinkedList.runSearch(elem);
                break;
            }
            case 8:
            {
                LinkedList.runConcatenate();
                break;
            }
            case 9:
            {
                LinkedList.printList();
                break;
            }
            case 10:
            {
                cout << "Exiting..." << endl;
```

```
            break;
        }
        default:
        {
            cout << "Wrong choice...\nTry again...";
            break;
        }
        }

    } while (choice != 10);

    return 0;
}
```

Output 5 :

```
Enter your choice
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
1
Enter an element : 2
Updated list : The list contains: 2
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
3
Enter an element : 4
Updated list : The list contains: 2 4
```

```
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
2
Enter elementX : 7
Enter elementY : 4
Updated list : The list contains: 2 4 7
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
4
Updated list : The list contains: 4 7
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
3
Enter an element : 9
Updated list : The list contains: 4 7 9
```

```
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
6
Enter element x : 7
Updated list : The list contains: 4 9
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
7
Enter element to be searched : 8
Element 8 is not found in the list
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
7
Enter element to be searched : 4
Element 4 is found in the list at index 0 in the list at address 0xeda490
```

```
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
8

Making new lists...
Enter values for List 1
Enter element : 4
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 1
Enter element : 5
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 0

Enter values for List 2
Enter element : 2
To continue... Enter 1 (To exit : Enter any integer(only) except 1) : 0
List 1 : The list contains: 4 5
List 2 : The list contains: 2
Concatenated list : The list contains: 4 5 2
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
9
The list contains: 4 9
1 - Insert an element x at the beginning of the circular linked list
2 - Insert an element x after an element y in the circular linked list
3 - Insert an element x at the end of the circular linked list
4 - Remove an element from the beginning of the circular linked list
5 - Remove an element from the end of the circular linked list.
6 - Remove an element x from the circular linked list.
7 - Search for an element x in the circular linked list and return its pointer
8 - Concatenate two circular linked lists
9 - Print the list
10 - Exit
10
Exiting...
```

Question 6 :

Implement a stack using Array representation

Solution 6 :

```
/*Created By - ANUJ KUMAR SEN */

#include <iostream>
using namespace std;

class Stack
{
    int *stack;
    int n, top;

public:

    Stack(int maxSize = 100)
    {
        stack = new int[maxSize];
        n = maxSize;
        top = -1;
    }
    void push(int val)
    {
        if (top >= n - 1)
            cout << "Stack Overflow" << endl;
        else
        {
            top++;
            stack[top] = val;
            cout << "Element pushed in Stack : " << val << endl;
        }
    }
    void pop()
    {
        if (top <= -1)
            cout << "Stack Underflow" << endl;
        else
        {
            cout << "Element pushed in Stack : " << stack[top] << endl;
            top--;
        }
    }
    void display()
```

```
    {
        if (top >= 0)
        {
            cout << "Stack : ";
            for (int i = top; i >= 0; i--)
            {
                cout << stack[i] << " ";
            }
            cout << endl;
        }
        else
            cout << "The Stack is empty" << endl;
    }
};

int main()
{
    Stack stack;
    int ch, val;

    do
    {
        cout << "1 - Push element into the stack" << endl;
        cout << "2 - Pop an element from the stack" << endl;
        cout << "3 - Display the stack" << endl;
        cout << "4 - Exit" << endl;

        cout << "Enter choice : ";
        cin >> ch;
        switch (ch)
        {
        case 1:
        {
            cout << "Enter the value to be pushed:" << endl;
            cin >> val;
            stack.push(val);
            stack.display();
            break;
        }
        case 2:
        {
            stack.pop();
            stack.display();
            break;
        }
        case 3:
        {
```

```
            stack.display();
            break;
        }
        case 4:
        {
            cout << "Exiting..." << endl;
            break;
        }
        default:
        {
            cout << "Invalid Choice" << endl;
        }
        }
    } while (ch != 4);
    return 0;
}
```

Output 6 :

```
 1 - Push element into the stack
 2 - Pop an element from the stack
 3 - Display the stack
 4 - Exit
 Enter choice : 1
 Enter the value to be pushed:
 5
 Element pushed in Stack : 5
 Stack : 5
 1 - Push element into the stack
 2 - Pop an element from the stack
 3 - Display the stack
 4 - Exit
 Enter choice : 1
 Enter the value to be pushed:
 4
 Element pushed in Stack : 4
 Stack : 4 5
 1 - Push element into the stack
 2 - Pop an element from the stack
 3 - Display the stack
 4 - Exit
 Enter choice : 1
 Enter the value to be pushed:
 3
 Element pushed in Stack : 3
 Stack : 3 4 5
 1 - Push element into the stack
 2 - Pop an element from the stack
 3 - Display the stack
 4 - Exit
 Enter choice : 2
 Element popped from Stack : 3
 Stack : 4 5
```

```
1 - Push element into the stack
2 - Pop an element from the stack
3 - Display the stack
4 - Exit
Enter choice : 1
Enter the value to be pushed:
2
Element pushed in Stack : 2
Stack : 2 4 5
1 - Push element into the stack
2 - Pop an element from the stack
3 - Display the stack
4 - Exit
Enter choice : 3
Stack : 2 4 5
1 - Push element into the stack
2 - Pop an element from the stack
3 - Display the stack
4 - Exit
Enter choice : 2
Element popped from Stack : 2
Stack : 4 5
1 - Push element into the stack
2 - Pop an element from the stack
3 - Display the stack
4 - Exit
Enter choice : 3
Stack : 4 5
1 - Push element into the stack
2 - Pop an element from the stack
3 - Display the stack
4 - Exit
Enter choice : 4
Exiting...
```

Question 7 :

Implement a stack using Linked representation

Solution 7 :

```
/*Created By - ANUJ KUMAR SEN */

#include <iostream>
using namespace std;

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

class Stack
{
private:
    Node *head;

public:
    Stack()
    {
        head = NULL;
    }

    void push(int elem)
    {
        Node *node = new Node();
        node->data = elem;
        node->next = NULL;

        cout << "Element pushed in stack : " << elem << endl;

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

```
    }

    void pop()
    {
        if (head == NULL)
        {
            cout << "Underflow";
        }
        else
        {
            Node *temp = head;
            head = temp->next;

            cout << "Element popped from stack : " << temp->data << endl;
            free(temp);
        }
        print();
    }
    int top()
    {
        return head->data;
    }

    void print()
    {
        if (head != NULL)
        {
            Node *temp = head;
            cout << "Stack : ";
            while (temp != NULL)
            {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
        else
        {
            cout << "The list is Empty" << endl;
        }
    }
};

int main()
{
    Stack stack;
    int ch;
```

```cpp
    do
    {
        cout << "1 - Push element into the stack" << endl;
        cout << "2 - Pop element from the stack" << endl;
        cout << "3 - Find element at top of stack" << endl;
        cout << "4 - Print stack" << endl;
        cout << "5 - Exit the program" << endl;
        cin >> ch;
        switch (ch)
        {
        case 1:
        {
            int elem;
            cout << "Enter an element" << endl;
            cin >> elem;
            stack.push(elem);
            break;
        }
        case 2:
        {
            stack.pop();
            break;
        }
        case 3:
        {
            cout << "Element at the top of stack : " << stack.top() << endl;
            break;
        }
        case 4:
        {
            stack.print();
            break;
        }
        case 5:
        {
            cout << "Exiting..." << endl;
            break;
        }
        }

    } while (ch != 5);
    return 0;
}
```

Output 7 :

```
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
1
Enter an element
5
Element pushed in stack : 5
Stack : 5
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
1
Enter an element
4
Element pushed in stack : 4
Stack : 4 5
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
1
Enter an element
6
Element pushed in stack : 6
Stack : 6 4 5
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
1
Enter an element
3
Element pushed in stack : 3
Stack : 3 6 4 5
```

```
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
2
Element popped from stack : 3
Stack : 6 4 5
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
3
Element at the top of stack : 6
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
4
Stack : 6 4 5
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
2
Element popped from stack : 6
Stack : 4 5
```

```
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
2
Element popped from stack : 4
Stack : 5
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
2
Element popped from stack : 5
The list is Empty
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
4
The list is Empty
1 - Push element into the stack
2 - Pop element from the stack
3 - Find element at top of stack
4 - Print stack
5 - Exit the program
5
Exiting...
```

Question 8 :

Implement Queue using Circular Array representation

Solution 8 :

```
/*Created By - ANUJ KUMAR SEN */

#include <iostream>
using namespace std;

class Queue
{
public:
    int *arr;
    int front, rear, size;
    Queue(int arrSize = 10)
    {
        size = arrSize;
        arr = new int(size);
        front = -1;
        rear = -1;
    }

    bool isEmpty()
    {
        if (front == -1 && rear == -1)
            return true;
        else
            return false;
    }

    void enqueue(int value)
    {
        if ((rear + 1) % size == front)
        {
            cout << "Queue is full \n";
        }

        else
        {
            if (front == -1)
            {
                front = 0;
            }
```

```
            rear = (rear + 1) % size;
            arr[rear] = value;
        }
    }

    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Queue is empty\n";
        }
        else if (front == rear)
        {
            front = rear = -1;
        }

        else
        {
            front = (front + 1) % size;
        }
    }

    void showfront()
    {
        if (isEmpty())
            cout << "Queue is empty\n";
        else
            cout << "Element at front is : " << arr[front];
    }

    void displayQueue()
    {
        if (isEmpty())
        {
            cout << "Queue : ";
            cout << "The queue is empty\n";
        }

        else
        {
            cout << "Queue : ";
            int i;
            if (front <= rear)
            {
                for (i = front; i <= rear; i++)
                    cout << arr[i] << " ";
            }
```

```
            else
            {
                i = front;
                while (i < size)
                {
                    cout << arr[i] << " ";
                    i++;
                }
                i = 0;
                while (i <= rear)
                {
                    cout << arr[i] << " ";
                    i++;
                }
            }
        }
    }
};

int main()
{
    Queue queue(5);
    int choice, value;
    do
    {
        cout << "\n1. Enqueue\n2. Dequeue\n3. Show front element\n4. Display
Queue\n5. Exit\n";
        cin >> choice;
        switch (choice)
        {
        case 1:
        {
            cout << "Enter Value:\n";
            cin >> value;
            queue.enqueue(value);
            queue.displayQueue();
            break;
        }
        case 2:
        {
            queue.dequeue();
            queue.displayQueue();
            break;
        }
        case 3:
        {
            queue.showfront();
```

```
            break;
        }
        case 4:
        {
            queue.displayQueue();
            break;
        }
        case 5:
        {
            cout << "Exiting..." << endl;
            break;
        }

        default:
        {
            cout << "Invalid choice" << endl;
            break;
        }
        }
    } while (choice != 5);

    return 0;
}
```

Output 8 :

```
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
1
Enter Value:
7
Queue : 7
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
1
Enter Value:
9
Queue : 7 9
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
3
Element at front is : 7
```

```
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
4
Queue : 7 9
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
2
Queue : 9
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
2
Queue : The queue is empty

1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
4
Queue : The queue is empty

1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
5
Exiting...
```

Question 9 :

Implement Queue using Circular linked list representation

Solution 9 :

```
/*Created By - ANUJ KUMAR SEN */

#include <iostream>
using namespace std;

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

class Queue
{
public:
    Node *front = NULL;
    Node *rear = NULL;

    bool isEmpty()
    {
        if (front == NULL && rear == NULL)
        {
            return true;
        }

        else
        {
            return false;
        }
    }

    void enqueue(int value)
    {
        Node *ptr = new Node();
        ptr->data = value;
        ptr->link = NULL;

        if (front == NULL)
        {
            front = ptr;
            rear = ptr;
        }
```

```
    else
    {
        rear->link = ptr;
        rear = ptr;
    }
}
void dequeue()
{
    if (isEmpty())
    {
        cout << "Queue is empty\n";
    }

    else
    {
        if (front == rear)
        {
            free(front);
            front = rear = NULL;
        }
        else
        {
            Node *ptr = front;
            front = front->link;
            free(ptr);
        }
    }
}

void showFront()
{
    if (isEmpty())
    {
        cout << "Queue is empty\n";
    }
    else
    {
        cout << "Element at front is : " << front->data << endl;
    }
}

void displayQueue()
{
    if (isEmpty())
    {
        cout << "Queue : ";
        cout << "The queue is empty\n";
```

```
        }
        else
        {
            cout << "Queue : ";
            Node *ptr = front;
            while (ptr != NULL)
            {
                cout << ptr->data << " ";
                ptr = ptr->link;
            }
            cout << endl;
        }
    }
};

int main()
{
    Queue queue;
    int choice, value;
    do
    {
        cout << "1. Enqueue\n2. Dequeue\n3. Show front element\n4. Display
Queue\n5. Exit\n";
        cin >> choice;
        switch (choice)
        {
        case 1:
        {
            cout << "Enter Value:\n";
            cin >> value;
            queue.enqueue(value);
            queue.displayQueue();
            break;
        }
        case 2:
        {
            queue.dequeue();
            queue.displayQueue();
            break;
        }
        case 3:
        {
            queue.showFront();
            break;
        }
        case 4:
        {
```

```cpp
                queue.displayQueue();
                break;
            }
            case 5:
            {
                cout << "Exiting..." << endl;
                break;
            }
            default:
            {
                cout << "Invlid choice" << endl;
                break;
            }
            }
    } while (choice != 5);

    return 0;
}
```

Output 9 :

```
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
1
Enter Value:
4
Queue : 4
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
1
Enter Value:
7
Queue : 4 7
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
1
Enter Value:
2
Queue : 4 7 2
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
3
Element at front is : 4
```

```
Queue : 4 7 2
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
2
Queue : 7 2
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
2
Queue : 2
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
2
Queue : The queue is empty
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
4
Queue : The queue is empty
1. Enqueue
2. Dequeue
3. Show front element
4. Display Queue
5. Exit
5
Exiting...
```

Question 10 :

Implement Double-ended Queues using Linked list representation

Solution 10 :

```
/*Created By - ANUJ KUMAR SEN */

#include <iostream>
using namespace std;

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

class Deque
{

public:
    Node *front;
    Node *rear;
    int size;
    Deque()
    {
        front = rear = NULL;
        size = 0;
    }
    Node *getnode(int data)
    {
        Node *newNode = new Node();
        newNode->data = data;
        newNode->prev = newNode->next = NULL;
        return newNode;
    }

    bool isEmpty()
    {
        return (front == NULL);
    }

    int dequeSize()
    {
        return size;
    }
```

```cpp
void insertFront(int data)
{
    Node *newNode = getnode(data);
    if (newNode == NULL)
    {
        cout << "OverFlow" << endl;
    }
    else
    {
        if (front == NULL)
        {
            rear = front = newNode;
        }

        else
        {
            newNode->next = front;
            front->prev = newNode;
            front = newNode;
        }
        size++;
    }
}

void insertRear(int data)
{
    Node *newNode = getnode(data);
    if (newNode == NULL)
    {
        cout << "OverFlow" << endl;
    }
    else
    {
        if (rear == NULL)
        {
            front = rear = newNode;
        }
        else
        {
            newNode->prev = rear;
            rear->next = newNode;
            rear = newNode;
        }
        size++;
    }
}
```

```
void deleteFront()
{
    if (isEmpty())
    {
        cout << "UnderFlow" << endl;
    }

    else
    {
        Node *temp = front;
        front = front->next;

        if (front == NULL)
        {
            rear = NULL;
        }
        else
        {
            front->prev = NULL;
        }
        free(temp);

        size--;
    }
}

void deleteRear()
{
    if (isEmpty())
    {
        cout << "UnderFlow" << endl;
    }
    else
    {
        Node *temp = rear;
        rear = rear->prev;

        if (rear == NULL)
        {
            front = NULL;
        }
        else
        {
            rear->next = NULL;
        }
        free(temp);
```

```
            size--;
        }
    }

    int getFront()
    {
        if (isEmpty())
        {
            return -1;
        }
        return front->data;
    }

    int getRear()
    {
        if (isEmpty())
        {
            return -1;
        }
        return rear->data;
    }

    void displayDeque()
    {
        if (isEmpty())
        {
            cout << "Deque : ";
            cout << "The deque is Empty\n";
        }
        else
        {
            cout << "Deque : ";
            Node *ptr = front;
            while (ptr != NULL)
            {
                cout << ptr->data << " ";
                ptr = ptr->next;
            }
            cout << endl;
        }
    }
};

int main()
{
    Deque dq;
    int choice, elem;
```

```
do
{
    cout << "\n1 - Insert element at FRONT" << endl;
    cout << "2 - Insert element at REAR" << endl;
    cout << "3 - Delete element at FRONT" << endl;
    cout << "4 - Delete element at REAR" << endl;
    cout << "5 - Show element at FRONT" << endl;
    cout << "6 - Show element at REAR" << endl;
    cout << "7 - Size of Deque" << endl;
    cout << "8 - Display Deque" << endl;
    cout << "9 - Exit" << endl;

    cout << "\nEnter your choice : ";
    cin >> choice;

    switch (choice)
    {
    case 1:
    {
        cout << "Enter element to be inserted at FRONT end of Deque : ";
        cin >> elem;
        dq.insertFront(elem);
        cout << "Inserted at FRONT : " << elem << endl;
        dq.displayDeque();
        break;
    }
    case 2:
    {
        cout << "Enter element to be inserted at REAR end of Deque : ";
        cin >> elem;
        dq.insertRear(elem);
        cout << "Inserted at REAR : " << elem << endl;
        dq.displayDeque();
        break;
    }
    case 3:
    {

        dq.deleteFront();
        cout << "Deleted from FRONT : " << elem << endl;
        dq.displayDeque();
        break;
    }
    case 4:
    {
        dq.deleteRear();
```

```cpp
            cout << "Deleted from REAR : " << elem << endl;
            dq.displayDeque();
            break;
        }
        case 5:
        {
            cout << "Element at FRONT : " << dq.getFront() << endl;
            break;
        }
        case 6:
        {
            cout << "Element at REAR : " << dq.getRear() << endl;
            break;
        }
        case 7:
        {
            cout << "Size of Deque : " << dq.dequeSize() << endl;
            break;
        }
        case 8:
        {
            dq.displayDeque();
            break;
        }
        case 9:
        {
            cout << "Exiting...!" << endl;
            break;
        }
        default:
        {
            cout << "Invalid choice!" << endl;
            break;
        }
        }

    } while (choice != 9);
    return 0;
}
```

Output 10 :

```
1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 1
Enter element to be inserted at FRONT end of Deque : 2
Inserted at FRONT : 2
Deque : 2

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 2
Enter element to be inserted at REAR end of Deque : 4
Inserted at REAR : 4
Deque : 2 4

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 2
Enter element to be inserted at REAR end of Deque : 5
Inserted at REAR : 5
Deque : 2 4 5
```

```
1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 1
Enter element to be inserted at FRONT end of Deque : 8
Inserted at FRONT : 8
Deque : 8 2 4 5

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 3
Deleted from FRONT : 8
Deque : 2 4 5

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 4
Deleted from REAR : 8
Deque : 2 4
```

```
1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 5
Element at FRONT : 2

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 6
Element at REAR : 4

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 7
Size of Deque : 2
```

```
1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 8
Deque : 2 4

1 - Insert element at FRONT
2 - Insert element at REAR
3 - Delete element at FRONT
4 - Delete element at REAR
5 - Show element at FRONT
6 - Show element at REAR
7 - Size of Deque
8 - Display Deque
9 - Exit

Enter your choice : 9
Exiting...!
```

Question 11 :

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

Solution 11 :

```
/*Created By - ANUJ KUMAR SEN */

#include <iostream>
using namespace std;

template <class T>
struct Node
{
    T data;
    Node<T> *left;
    Node<T> *right;
};

template <class T>
class BST
{
private:
    Node<T> *root;

public:
    BST()
    {
        root = nullptr;
    }

    void insert(T x)
    {
        Node<T> *newNode = new Node<T>();
        newNode->data = x;
        newNode->right = nullptr;
        newNode->left = nullptr;
        Node<T> *temp, *ptr = root;
        if (root == nullptr)
        {
```

```
            root = newNode;
        }
        else
        {
            while (ptr != nullptr)
            {
                temp = ptr;
                if (x >= ptr->data)
                {
                    ptr = ptr->right;
                }
                else
                {
                    ptr = ptr->left;
                }
            }
            if (x >= temp->data)
            {
                temp->right = newNode;
            }
            else
            {
                temp->left = newNode;
            }
        }
        cout << "Inserted Node<" << x << ">" << endl;
        ;
    }

    void inorder(Node<T> *ptr)
    {
        if (ptr == nullptr)
            return;
        else
        {
            inorder(ptr->left);
            cout << ptr->data << " ";
            inorder(ptr->right);
        }
    }

    void preorder(Node<T> *ptr)
    {
        if (ptr == nullptr)
        {
            return;
        }
```

```
    else
    {
        cout << ptr->data << " ";
        preorder(ptr->left);
        preorder(ptr->right);
    }
}

void postorder(Node<T> *ptr)
{
    if (ptr == nullptr)
    {
        return;
    }
    else
    {
        postorder(ptr->left);
        postorder(ptr->right);
        cout << ptr->data << " ";
    }
}

Node<T> *get_root()
{
    return root;
}

void delc(Node<T> *&temp)
{
    Node<T> *prev, *tmp = temp;
    if (temp->right == NULL)
    {
        temp = temp->left;
    }
    else if (temp->left == NULL)
    {
        temp = temp->right;
    }
    else
    {
        tmp = temp->left;
        prev = temp;
        while (tmp->right != NULL)
        {
            prev = tmp;
            tmp = tmp->right;
        }
```

```
            temp->data = tmp->data;
            if (prev == temp)
            {
                prev->left = tmp->left;
            }
            else
            {
                prev->right = tmp->left;
            }
        }
        delete tmp;
    }

    void del_copy(T el)
    {
        Node<T> *prev;
        Node<T> *ptr = root;
        while (ptr != nullptr)
        {
            if (ptr->data == el)
                break;
            prev = ptr;
            if (ptr->data < el)
                ptr = ptr->right;
            else
                ptr = ptr->left;
        }
        if (ptr != nullptr && ptr->data == el)
        {
            if (ptr == root)
                delc(root);
            else if (prev->left == ptr)
                delc(prev->left);
            else
                delc(prev->right);
        }
        else if (root != 0)
            cout << "\nNode not found in the tree!";
        else
            cout << "\n\tThe tree is Empty!";
    }

    void search_change()
    {
        T key, newKey;
        cout << "\nEnter the key to be searched : ";
        cin >> key;
```

```
        Node<T> *ptr = root;
        if (ptr == nullptr)
        {
            cout << "The tree is Empty!" << endl;
        }
        else
        {
            cout << "Enter the new key: ";
            cin >> newKey;

            int flag = 0;
            while (ptr != nullptr)
            {
                if (key == ptr->data)
                {
                    flag = 1;
                    break;
                }
                else if (key > ptr->data)
                    ptr = ptr->right;
                else
                    ptr = ptr->left;
            }
            if (flag == 0)
                cout << "Node not found in the tree!" << endl;
            else
            {
                del_copy(key);
                insert(newKey);
            }
        }
    }

    int height_helper(Node<T> *temp)
    {
        int hleft = 0;
        int hright = 0;
        if (temp != nullptr)
        {
            hleft = height_helper(temp->left);
            hright = height_helper(temp->right);
            if (hleft > hright)
                return hleft + 1;
            else
                return hright + 1;
        }
        return -1;
```

```
    }

    int height()
    {
        return height_helper(root);
    }
    void printGivenLevel(Node<T> *rootNode, int level)
    {
        if (rootNode == NULL)
        {
            return;
        }
        if (level == 1)
        {
            cout << rootNode->data << " ";
        }
        else if (level > 1)
        {
            printGivenLevel(rootNode->left, level - 1);
            printGivenLevel(rootNode->right, level - 1);
        }
    }

    void printLevelOrder()
    {
        int h = height();
        for (int i = 1; i <= h + 1; i++)
        {
            cout << "Level " << i << " : ";
            printGivenLevel(root, i);
            cout << endl;
        }
    }
};

int main()
{
    BST<int> bst;

    int choice, temp;
    do
    {
        cout << "\n1 - Insert an element x" << endl;
        cout << "2 - Delete an element x" << endl;
        cout << "3 - Search for an element x - change its value to y - place node
at its appropriate position in the BST " << endl;
```

```
        cout << "4 - Display the elements of the BST in preorder, inorder, and
postorder traversal" << endl;
        cout << "5 - Display the elements of the BST in level-by-level traversal"
<< endl;
        cout << "6 - Display the height of the BST" << endl;
        cout << "7 - Exit" << endl;

        cout << "\nEnter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            cout << "Enter an element x : ";
            cin >> temp;
            bst.insert(temp);
            break;
        }
        case 2:
        {
            cout << "Enter an element you want to delete : ";
            cin >> temp;
            bst.del_copy(temp);
            cout << "Deleted Node<" << temp << ">" << endl;
            break;
        }
        case 3:
        {
            bst.search_change();
            break;
        }
        case 4:
        {
            cout << "Inorder : ";
            bst.inorder(bst.get_root());
            cout << endl;
            cout << "Preorder : ";
            bst.preorder(bst.get_root());
            cout << endl;
            cout << "Postorder : ";
            bst.postorder(bst.get_root());
            cout << endl;
            break;
        }
        case 5:
        {
```

```
            bst.printLevelOrder();
            break;
        }
        case 6:
        {
            cout << "Height of tree: " << bst.height();
            cout << endl;
            break;
        }
        case 7:
        {
            cout << "Exiting..." << endl;
            break;
        }

        default:
            cout << "Invalid Choice!" << endl;
            break;
        }

    } while (choice != 7);
    return 0;
}
```

Output 11 :

```
1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 1
Enter an element x : 8
Inserted Node<8>

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 1
Enter an element x : 9
Inserted Node<9>

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 1
Enter an element x : 1
Inserted Node<1>
```

```
1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 5
Level 1 : 8
Level 2 : 1 9

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 3

Enter the key to be searched : 5
Enter the new key: 4
Node not found in the tree!

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 3

Enter the key to be searched : 8
Enter the new key: 4
Inserted Node<4>
```

```
Enter your choice: 4
Inorder : 1 4 9
Preorder : 1 9 4
Postorder : 4 9 1

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 6
Height of tree: 2

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 1
Enter an element x : 2
Inserted Node<2>

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 5
Level 1 : 1
Level 2 : 9
Level 3 : 4
Level 4 : 2
```

```
1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 3

Enter the key to be searched : 2
Enter the new key: 10
Inserted Node<10>

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 4
Inorder : 1 4 9 10
Preorder : 1 9 4 10
Postorder : 4 10 9 1

1 - Insert an element x
2 - Delete an element x
3 - Search for an element x - change its value to y - place node at its appropriate position in the BST
4 - Display the elements of the BST in preorder, inorder, and postorder traversal
5 - Display the elements of the BST in level-by-level traversal
6 - Display the height of the BST
7 - Exit

Enter your choice: 7
Exiting...
```

END OF ASSIGNMENT