EX:NO.: 4
DATE:

CIRCULAR DOUBLE LINKED LIST

AIM:

To write a C++ program that performs various circular double linked list such as insertion at the beginning, insertion at the end, insertion at the given position, deletion at the beginning, deletion at the end, deletion at the given position, forward traversal, Reverse traversal, search an element using linear search, intersection of two lists.

ALGORITHM:

STEP 1: Start the porgram.
STEP 2: Insertion at the beginning,

- Create a new node with the given value.
- Set its next to current head and prev to NULL.
- Update head and fix previous pointer of old head (if it exists).

STEP 3: Insertion at the end,

- Create a new node with the given value,prev and next as NULL.
- Traverse the list to reach the last node.
- Set the last node’s next to the new node and new node’s prev to last node.

STEP 4: Insertion at the given position,.

- Traverse to the (pos-1)th node in the list.
- Create a new node and adjust next and prev links.
- Handle edge cases: insert at beginning if position ≤ 1 or at end if beyond size.

STEP 5: Deletion at the beginning,

- Check if the list is empty.
- Move head to next node.
- Set new head’s prev to NULL and delete old head.

STEP 6: Deletion at the end,

- Check for empty or single-node list.
- Traverse to the second-last node.
- Delete the last node and set second-last’s next to NULL.

STEP 7: Deletion at the given position,

- Traverse to the specified position.
- Adjust next and prev of neighboring nodes.
- Delete the target node.

STEP 8: Search an element in the list,

- Traverse the list from head to end.
- Compare each node's data with the search key.
- Return true if found, false otherwise.

STEP 9: Display the elements in the list,

- Check if the list is empty.
- Traverse each node from head to end.
- Print node data followed by an arrow (<->).

STEP 10: Display the list in reverse,

- Initialize three pointers: prev, curr, and next.
- Iterate through the list, reversing each link.
- Update head to the last node (now the new head).
- Print the data of each node.

STEP 11: Intersection of two lists,

- Traverse the first list.
- For each node, check if it exists in the second list using search.
- If found and not already in result list, insert it into the result.

STEP 12: STOP.

<u>CODE:</u>

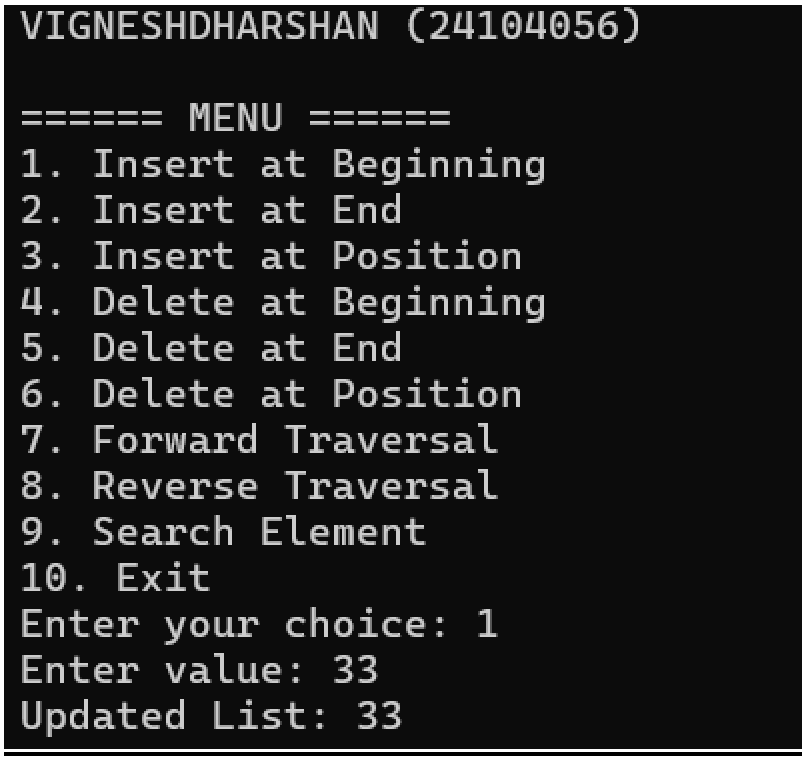
```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;

    Node(int value) {
        data = value;
        prev = next = NULL;
    }
};

class CircularDoublyLinkedList {
```

```
private:
    Node* head;
public:
    CircularDoublyLinkedList() { head = NULL; }

    void insertAtBeginning(int data) {
        Node* newNode = new Node(data);
        if (!head) {
            newNode->next = newNode->prev = newNode;
            head = newNode;
            return;
        }
        Node* last = head->prev;
        newNode->next = head;
        newNode->prev = last;
        last->next = newNode;
        head->prev = newNode;
        head = newNode;
    }

    void insertAtEnd(int data) {
        if (!head) { insertAtBeginning(data); return; }
        Node* newNode = new Node(data);
        Node* last = head->prev;
        newNode->next = head;
        newNode->prev = last;
        last->next = newNode;
        head->prev = newNode;
    }

    void insertAtPosition(int data, int position) {
        if (!head || position <= 1) { insertAtBeginning(data); return; }
        Node* temp = head;
        for (int i = 1; i < position - 1 && temp->next != head; i++)
            temp = temp->next;
        if (temp->next == head) { insertAtEnd(data); return; }
        Node* newNode = new Node(data);
        newNode->next = temp->next;
        newNode->prev = temp;
        temp->next->prev = newNode;
        temp->next = newNode;
    }

    void deleteAtBeginning() {
        if (!head) return;
        if (head->next == head) { delete head; head = NULL; return; }
        Node* temp = head;
        Node* last = head->prev;
        head = head->next;
        head->prev = last;
        last->next = head;
        delete temp;
    }

    void deleteAtEnd() {
        if (!head) return;
        if (head->next == head) { delete head; head = NULL; return; }
        Node* last = head->prev;
        last->prev->next = head;
        head->prev = last->prev;
        delete last;
    }

    void deleteAtPosition(int position) {
        if (!head || position <= 1) { deleteAtBeginning(); return; }
        Node* temp = head;
        for (int i = 1; i < position && temp->next != head; i++)
            temp = temp->next;
        if (temp == head) { deleteAtEnd(); return; }
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        delete temp;
    }

    void forwardTraversal() {
        if (!head) { cout << "List is empty.\n"; return; }
        Node* temp = head;
        do {
            cout << temp->data;
            temp = temp->next;
```

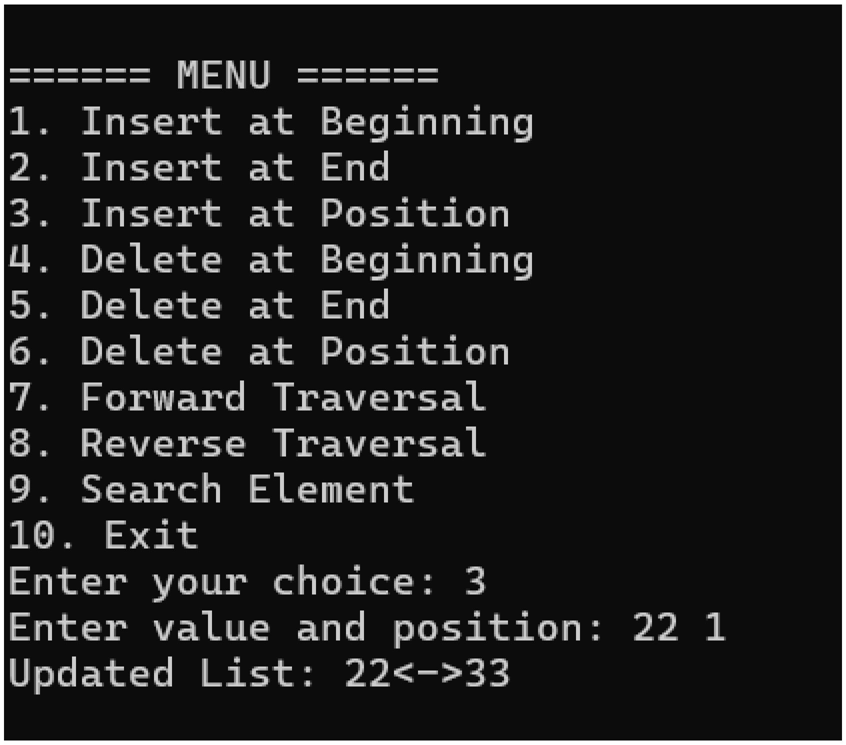
```
            if (temp != head) cout << "<->";
        } while (temp != head);
        cout << endl;
    }

    void reverseTraversal() {
        if (!head) { cout << "List is empty.\n"; return; }
        Node* temp = head->prev;
        Node* start = temp;
        do {
            cout << temp->data;
            temp = temp->prev;
            if (temp != start) cout << "<->";
        } while (temp != start);
        cout << endl;
    }

    void search(int key) {
        if (!head) { cout << "List is empty.\n"; return; }
        Node* temp = head; int pos = 1;
        do {
            if (temp->data == key) {
                cout << "Element " << key << " found at position " << pos << endl;
                return;
            }
            pos++;
            temp = temp->next;
        } while (temp != head);
        cout << "Element " << key << " not found.\n";
    }
};

int main() {
    CircularDoublyLinkedList list1;
    int choice, value, pos;

    cout << "VIGNESHDHARSHAN (24104056)\n";

    while (true) {
        cout << "\n====== MENU ======\n";
cout << "1. Insert at Beginning\n";
cout << "2. Insert at End\n";
cout << "3. Insert at Position\n";
cout << "4. Delete at Beginning\n";
cout << "5. Delete at End\n";
cout << "6. Delete at Position\n";
cout << "7. Forward Traversal\n";
cout << "8. Reverse Traversal\n";
cout << "9. Search Element\n";
cout << "10. Exit\n";
cout << "Enter your choice: ";
cin >> choice;

switch (choice) {
    case 1:
        cout << "Enter value: ";
        cin >> value;
        list1.insertAtBeginning(value);
        cout << "Updated List: ";
        list1.forwardTraversal();
        break;

    case 2:
        cout << "Enter value: ";
        cin >> value;
        list1.insertAtEnd(value);
        cout << "Updated List: ";
        list1.forwardTraversal();
        break;

    case 3:
        cout << "Enter value and position: ";
        cin >> value >> pos;
        list1.insertAtPosition(value, pos);
        cout << "Updated List: ";
        list1.forwardTraversal();
        break;

    case 4:
        list1.deleteAtBeginning();
        cout << "Updated List: ";
        list1.forwardTraversal();
        break;

    case 5:
        list1.deleteAtEnd();
        cout << "Updated List: ";
        list1.forwardTraversal();
        break;

    case 6:
        cout << "Enter position to delete: ";
        cin >> pos;
        list1.deleteAtPosition(pos);
        cout << "Updated List: ";
        list1.forwardTraversal();
        break;

    case 7:
        cout << "Current List (Forward): ";
        list1.forwardTraversal();
        break;

    case 8:
        cout << "Current List (Reverse): ";
        list1.reverseTraversal();
        break;

    case 9:
        cout << "Enter element to search: ";
        cin >> value;
        list1.search(value);
        break;

    case 10:
        cout << "Exiting program.\n";
        return 0;

    default:
        cout << "Invalid choice! Try again.\n";
```

```
        }
    }
}
```

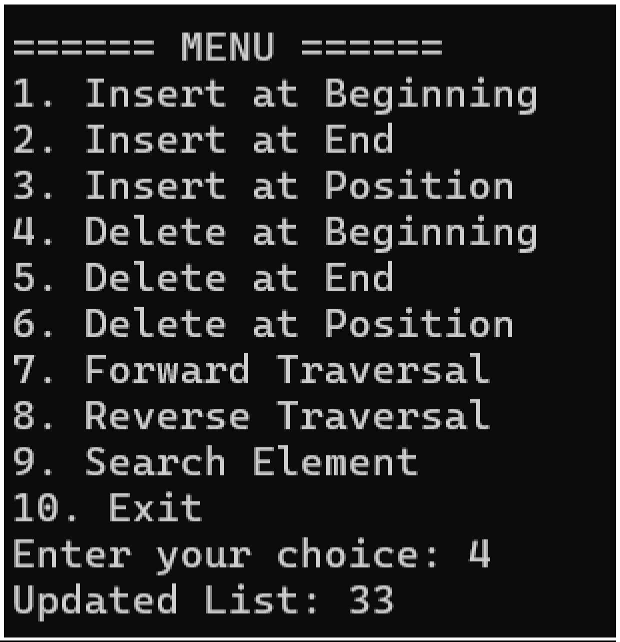
<u>OUTPUT SCREEN:</u>



```
VIGNESHDHARSHAN (24104056)

====== MENU ======
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Forward Traversal
8. Reverse Traversal
9. Search Element
10. Exit
Enter your choice: 1
Enter value: 33
Updated List: 33
```



```
====== MENU ======
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Forward Traversal
8. Reverse Traversal
9. Search Element
10. Exit
Enter your choice: 3
Enter value and position: 22 1
Updated List: 22<->33
```



```
====== MENU ======
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Forward Traversal
8. Reverse Traversal
9. Search Element
10. Exit
Enter your choice: 4
Updated List: 33
```

RUBRICS:

Problem Identification (5)	Implementation (10)	Time Management (2)	Viva (3)	Total (20)

RESULT:

Thus the CPP program executes the circular double linked list operations like inserting, deleting, searching, display, reverse display of a list.