**AQ1:**

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node \*next, \*prev;

};

Node\* CLLinsertAtPosition(Node \*last, int data, int pos) {

    if (last == NULL) {

        if (pos != 1) {

            cout << "Invalid position!" << endl;

            return last;

        }

        Node \*newNode = new Node();

        newNode->data = data;

        newNode->next = newNode;

        return newNode;

    }

    Node \*newNode = new Node();

    newNode->data = data;

    Node \*curr = last->next;

    if (pos == 1) {

        newNode->next = curr;

        last->next = newNode;

        return last;

    }

    for (int i = 1; i < pos - 1; ++i) {

        curr = curr->next;

        if (curr == last->next) {

            cout << "Invalid position!" << endl;

            delete newNode;

            return last;

        }

    }

    newNode->next = curr->next;

    curr->next = newNode;

    if (curr == last) last = newNode;

    return last;

}

Node\* CLLdeleteByValue(Node \*last, int value) {

    if (last == NULL) {

        cout << "List empty" << endl;

        return last;

    }

    Node \*curr = last->next, \*prev = last;

    do {

        if (curr->data == value) {

            if (curr == last && curr->next == last) {

                delete curr;

                return NULL;

            }

            if (curr == last->next) {

                prev->next = curr->next;

                delete curr;

                return last;

            }

            if (curr == last) {

                prev->next = curr->next;

                delete curr;

                last = prev;

                return last;

            }

            prev->next = curr->next;

            delete curr;

            return last;

        }

        prev = curr;

        curr = curr->next;

    } while (curr != last->next);

    cout << "Value not found" << endl;

    return last;

}

void CLLdisplay(Node \*last) {

    if (last == NULL) {

        cout << "List empty" << endl;

        return;

    }

    Node \*head = last->next;

    while (true) {

        cout << head->data << " ";

        head = head->next;

        if (head == last->next) break;

    }

    cout << endl;

}

void CLLsearch(Node \*last, int value) {

    if (last == NULL) {

        cout << "List empty" << endl;

        return;

    }

    Node \*curr = last->next;

    int pos = 1;

    bool found = false;

    do {

        if (curr->data == value) {

            cout << "Value found at position " << pos << endl;

            found = true;

            break;

        }

        curr = curr->next;

        pos++;

    } while (curr != last->next);

    if (!found) cout << "Value not found" << endl;

}

Node\* DLLinsertAtPosition(Node \*head, Node \*&tail, int data, int pos) {

    Node \*newNode = new Node();

    newNode->data = data;

    newNode->next = newNode->prev = NULL;

    if (head == NULL) {

        if (pos != 1) {

            cout << "Invalid position!" << endl;

            delete newNode;

            return head;

        }

        head = tail = newNode;

        return head;

    }

    if (pos == 1) {

        newNode->next = head;

        head->prev = newNode;

        head = newNode;

        return head;

    }

    Node \*curr = head;

    for (int i = 1; i < pos - 1 && curr->next != NULL; ++i) curr = curr->next;

    if (pos > 1 && curr == tail) {

        curr->next = newNode;

        newNode->prev = curr;

        tail = newNode;

        return head;

    }

    if (curr == NULL) {

        cout << "Invalid position!" << endl;

        delete newNode;

        return head;

    }

    newNode->next = curr->next;

    newNode->prev = curr;

    curr->next->prev = newNode;

    curr->next = newNode;

    return head;

}

Node\* DLLdeleteByValue(Node \*head, Node \*&tail, int value) {

    if (head == NULL) {

        cout << "List empty" << endl;

        return head;

    }

    Node \*curr = head;

    while (curr!=NULL && curr->data != value) curr = curr->next;

    if (curr==NULL) {

        cout << "Value not found" << endl;

        return head;

    }

    if (curr == head && curr == tail) {

        delete curr;

        head = tail = NULL;

        return head;

    }

    if (curr == head) {

        head = head->next;

        head->prev = NULL;

        delete curr;

        return head;

    }

    if (curr == tail) {

        tail = tail->prev;

        tail->next = NULL;

        delete curr;

        return head;

    }

    curr->prev->next = curr->next;

    curr->next->prev = curr->prev;

    delete curr;

    return head;

}

void DLLdisplay(Node \*head) {

    if (head == NULL) {

        cout << "List empty" << endl;

        return;

    }

    Node \*temp = head;

    while (temp!=NULL) {

        cout << temp->data << " ";

        temp = temp->next;

    }

    cout << endl;

}

void DLLsearch(Node \*head, int value) {

    if (head == NULL) {

        cout << "List empty" << endl;

        return;

    }

    Node \*curr = head;

    int pos = 1;

    bool found = false;

    while (curr!=NULL) {

        if (curr->data == value) {

            cout << "Value found at position " << pos << endl;

            found = true;

            break;

        }

        curr = curr->next;

        pos++;

    }

    if (!found) cout << "Value not found" << endl;

}

int main() {

    int mainChoice;

    while (true) {

        cout << "\n1.Circular Linked List  2.Doubly Linked List  3.Exit\nEnter: ";

        cin >> mainChoice;

        if (mainChoice == 3) return 0;

        if (mainChoice == 1) {

            Node \*last = NULL;

            int choice, data, pos;

            while (true) {

                cout << "\n1.Insert 2.Display 3.Delete 4.Search 5.Exit\nEnter: ";

                cin >> choice;

                if (choice == 5) return 0; // Exit program

                if (choice == 1) {

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

                    cout << "Enter position: "; cin >> pos;

                    last = CLLinsertAtPosition(last, data, pos);

                } else if (choice == 2) {

                    CLLdisplay(last);

                } else if (choice == 3) {

                    cout << "Enter value to delete: "; cin >> data;

                    last = CLLdeleteByValue(last, data);

                } else if (choice == 4) {

                    cout << "Enter value to search: "; cin >> data;

                    CLLsearch(last, data);

                }

            }

        }

        else if (mainChoice == 2) {

            Node \*head = NULL, \*tail = NULL;

            int choice, data, pos;

            while (true) {

                cout << "\n1.Insert 2.Display 3.Delete 4.Search 5.Exit\nEnter: ";

                cin >> choice;

                if (choice == 5) return 0; // Exit program

                if (choice == 1) {

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

                    cout << "Enter position: "; cin >> pos;

                    head = DLLinsertAtPosition(head, tail, data, pos);

                } else if (choice == 2) {

                    DLLdisplay(head);

                } else if (choice == 3) {

                    cout << "Enter value to delete: "; cin >> data;

                    head = DLLdeleteByValue(head, tail, data);

                } else if (choice == 4) {

                    cout << "Enter value to search: "; cin >> data;

                    DLLsearch(head, data);

                }

            }

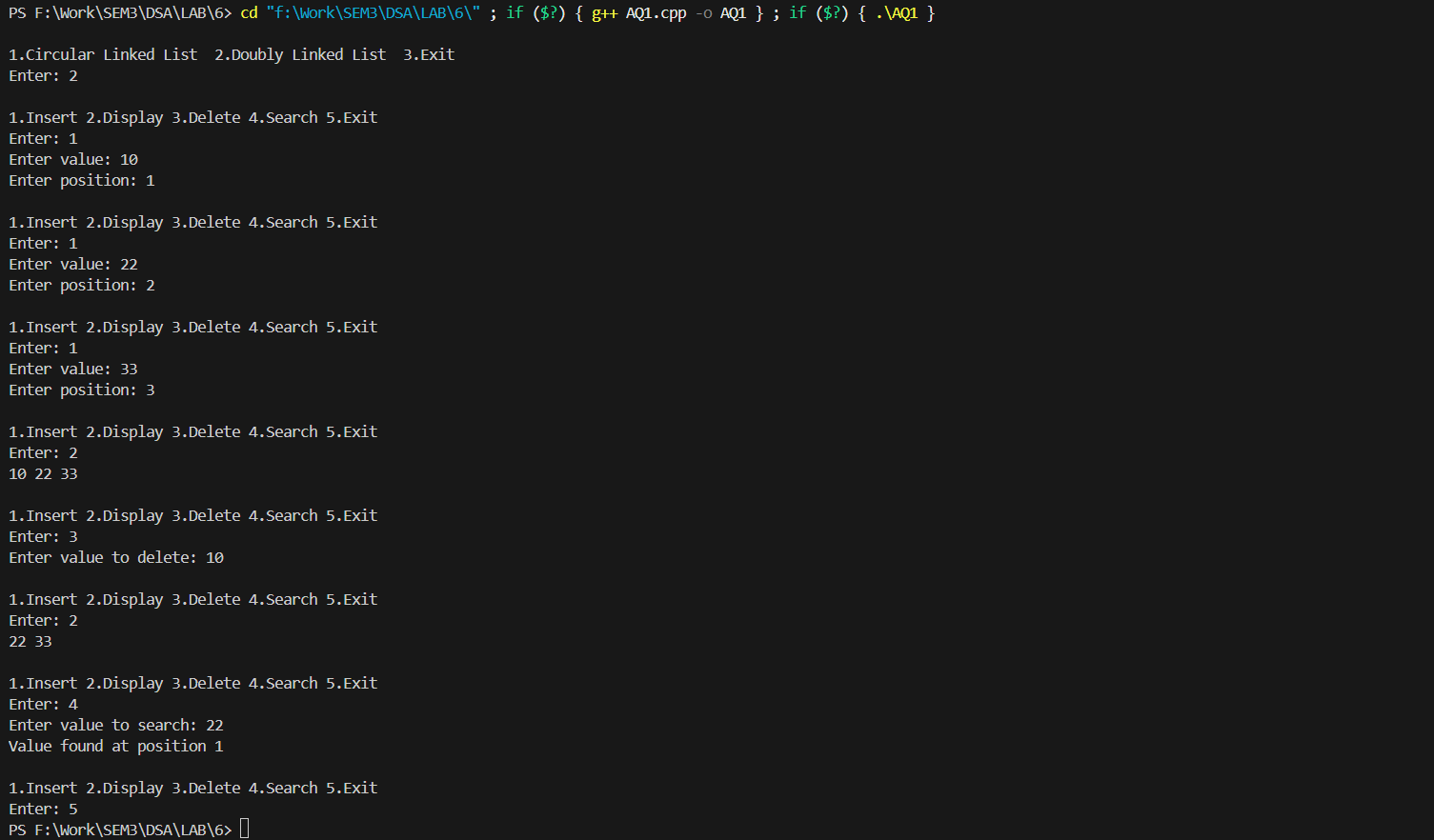
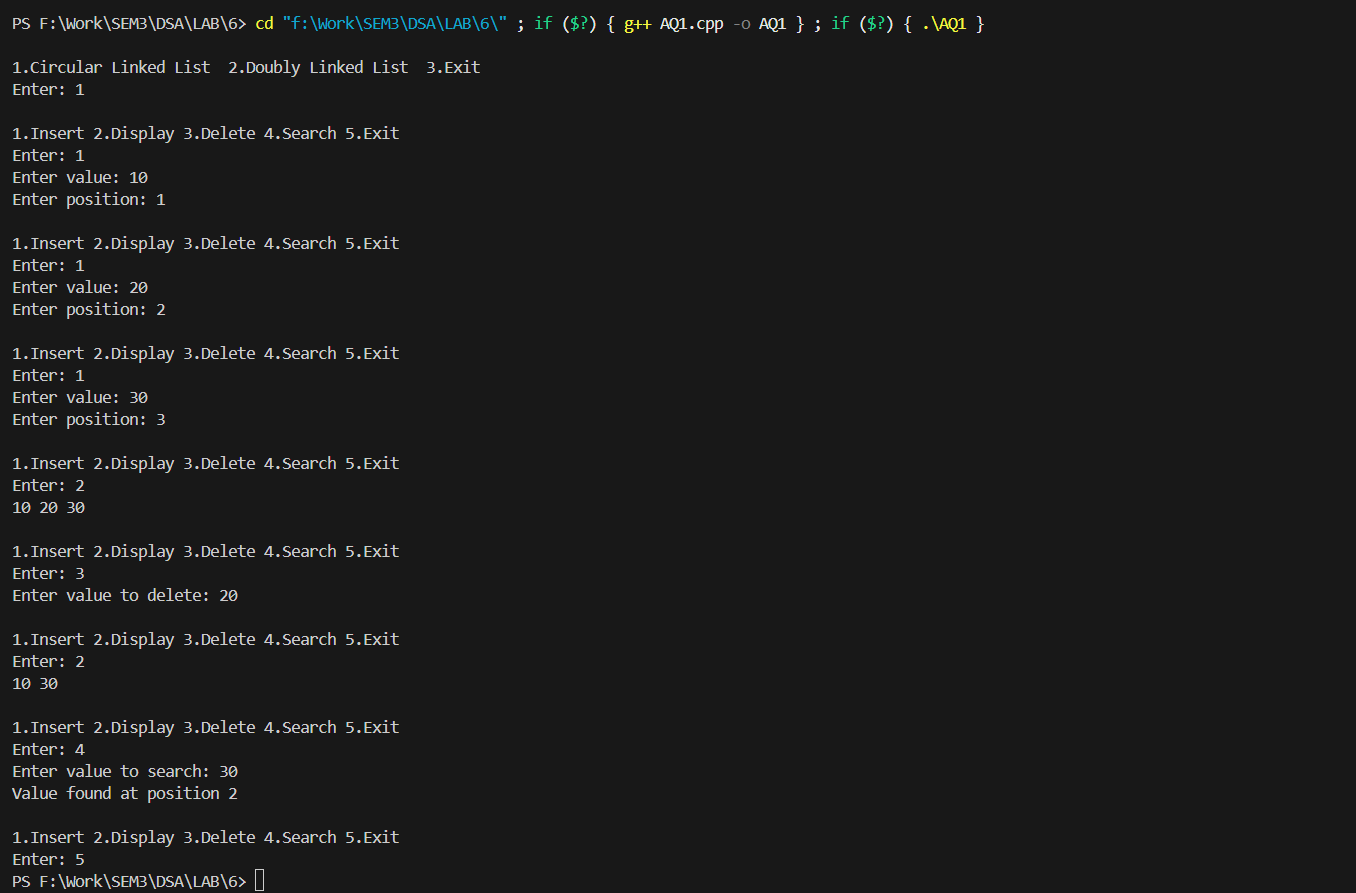
        }

    }

    return 0;

}

**Output:**



**AQ2:**

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int x) {

        data = x;

        next = NULL;

    }

};

void DisplayCLL(Node\* head) {

    if (head == NULL)

    return;

    Node\* curr = head;

    int result = 0;

    do {

        cout<<curr->data<<" ";

        curr = curr->next;

        result++;

    } while (curr != head);

    cout<<curr->data<<" ";

}

int main() {

    Node\* head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(3);

    head->next->next->next = new Node(4);

    head->next->next->next->next = new Node(5);

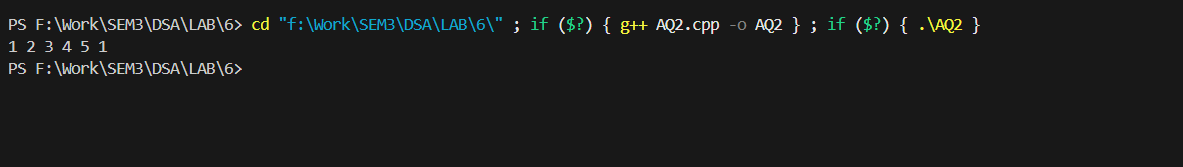
    head->next->next->next->next->next = head;

    DisplayCLL(head);

    return 0;

}

**Output:**



**AQ3:**

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node \*next;

    Node \*prev;

    Node(int val) {

        data = val;

        next = NULL;

        prev = NULL;

    }

};

int sizeDLL(Node \*head) {

    int count = 0;

    Node \*curr = head;

    while (curr != NULL) {

        count++;

        curr = curr->next;

    }

    return count;

}

int sizeCLL(Node \*last) {

    if (last==NULL)

    return 0;

    int count = 0;

    Node \*curr = last->next;

    Node \*head = last->next;

    do {

        count++;

        curr = curr->next;

    } while (curr != head);

    return count;

}

int main() {

    Node \*dllHead = new Node(1);

    dllHead->next = new Node(2);

    dllHead->next->prev = dllHead;

    dllHead->next->next = new Node(3);

    dllHead->next->next->prev = dllHead->next;

    dllHead->next->next->next = new Node(4);

    dllHead->next->next->next->prev = dllHead->next->next;

    cout << "Size of Doubly Linked List: " << sizeDLL(dllHead) << endl;

    Node \*cll1 = new Node(10);

    Node \*cll2 = new Node(20);

    Node \*cll3 = new Node(30);

    cll1->next = cll2;

    cll2->next = cll3;

    cll3->next = cll1;

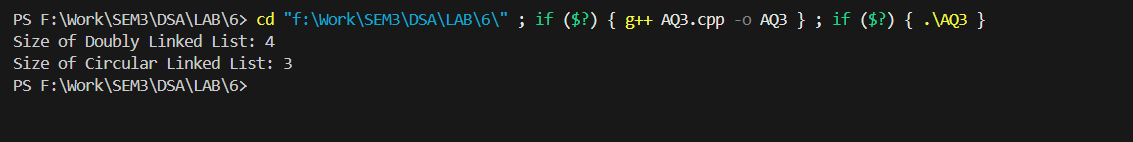
    Node \*cllLast = cll3;

    cout << "Size of Circular Linked List: " << sizeCLL(cllLast) << endl;

    return 0;

}

**Output:**



**AQ4:**

#include <iostream>

using namespace std;

struct Node {

    char data;

    Node \*next;

    Node \*prev;

    Node(char val) {

        data = val;

        next = prev = NULL;

    }

};

bool isPalindrome(Node \*head) {

    if (head==NULL) return true;

    Node \*tail = head;

    while (tail->next) tail = tail->next;

    Node \*front = head;

    while (front != tail && front->prev != tail) {

        if (front->data != tail->data)

            return false;

        front = front->next;

        tail = tail->prev;

    }

    return true;

}

int main() {

    Node \*head = new Node('L');

    head->next = new Node('E');

    head->next->prev = head;

    head->next->next = new Node('V');

    head->next->next->prev = head->next;

    head->next->next->next = new Node('E');

    head->next->next->next->prev = head->next->next;

    head->next->next->next->next = new Node('L');

    head->next->next->next->next->prev = head->next->next->next;

    if (isPalindrome(head))

        cout << "True" << endl;

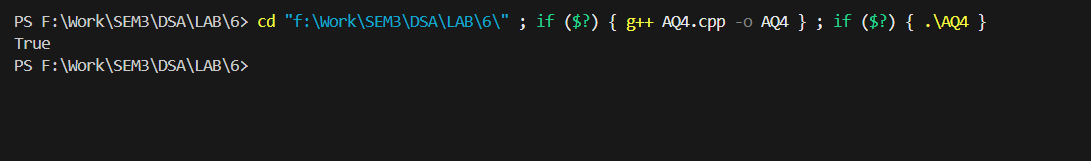
    else

        cout << "False" << endl;

    return 0;

}

**Output:**



**AQ5:**

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int x)

    {

        data = x;

        next = NULL;

    }

};

bool isCircular(Node \*head)

{

    if (head==NULL)

        return true;

    Node \*curr = head;

    while (curr!=NULL && curr->next != head)

        curr = curr->next;

    if (curr==NULL)

        return false;

    return true;

}

int main()

{

    Node \*head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(3);

    head->next->next->next = new Node(4);

    head->next->next->next->next = head;

    if (isCircular(head))

        cout << "Yes\n";

    else

        cout << "No\n";

    return 0;

}

**Output:**

