DOUBLY AND CIRCULAR LINKED LISTS
ASSIGNMENT – 6

AADITA GARG
1024030461 – 2C32

Develop a menu driven program for the following operations of on a Circular as well as a Doubly Linked List.

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

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

class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() { head = nullptr; }

    void insertAtBeginning(int val) {
        Node* newNode = new Node{val, nullptr, nullptr};
        if (!head) {
            head = newNode;
            return;
        }
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
```

```
}

void insertAtEnd(int val) {
    Node* newNode = new Node{val, nullptr, nullptr};
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
}

void insertAfter(int key, int val) {
    Node* temp = head;
    while (temp && temp->data != key) temp = temp->next;
    if (!temp) {
        cout << "Node " << key << " not found!\n";
        return;
    }
    Node* newNode = new Node{val, temp->next, temp};
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
}

void insertBefore(int key, int val) {
    if (!head) return;
    if (head->data == key) {
        insertAtBeginning(val);
        return;
    }
    Node* temp = head;
    while (temp && temp->data != key) temp = temp->next;
    if (!temp) {
```

```
        cout << "Node " << key << " not found!\n";
        return;
    }
    Node* newNode = new Node{val, temp, temp->prev};
    temp->prev->next = newNode;
    temp->prev = newNode;
}

void deleteNode(int key) {
    if (!head) return;
    Node* temp = head;

    if (head->data == key) {
        head = head->next;
        if (head) head->prev = nullptr;
        delete temp;
        return;
    }

    while (temp && temp->data != key) temp = temp->next;
    if (!temp) {
        cout << "Node not found!\n";
        return;
    }

    if (temp->next) temp->next->prev = temp->prev;
    if (temp->prev) temp->prev->next = temp->next;
    delete temp;
}

bool searchNode(int key) {
    Node* temp = head;
    while (temp) {
        if (temp->data == key) return true;
        temp = temp->next;
```

```
        }
        return false;
    }

    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << "\n";
    }
};

// Circular Doubly Linked List
class CircularDoublyLinkedList {
private:
    Node* head;

public:
    CircularDoublyLinkedList() { head = nullptr; }

    void insertAtEnd(int val) {
        Node* newNode = new Node{val, nullptr, nullptr};
        if (!head) {
            head = newNode;
            head->next = head;
            head->prev = head;
            return;
        }
        Node* last = head->prev;
        newNode->next = head;
        newNode->prev = last;
        last->next = newNode;
        head->prev = newNode;
```

```
}

void deleteNode(int key) {
    if (!head) return;
    Node* temp = head;
    Node* toDelete = nullptr;
    do {
        if (temp->data == key) {
            toDelete = temp;
            break;
        }
        temp = temp->next;
    } while (temp != head);

    if (!toDelete) {
        cout << "Node not found!\n";
        return;
    }

    if (toDelete->next == toDelete) {
        head = nullptr;
        delete toDelete;
        return;
    }

    Node* prevNode = toDelete->prev;
    Node* nextNode = toDelete->next;
    prevNode->next = nextNode;
    nextNode->prev = prevNode;
    if (toDelete == head) head = nextNode;
    delete toDelete;
}

void display() {
    if (!head) {
```

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

int main() {
    DoublyLinkedList dll;
    CircularDoublyLinkedList cdll;
    int choice, val, key;

    while (true) {
        cout << "\n MENU \n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value: ";
                cin >> val;
                dll.insertAtBeginning(val);
                break;
            case 2:
                cout << "Enter value: ";
                cin >> val;
                dll.insertAtEnd(val);
                break;
            case 3:
                cout << "Enter key and value: ";
```

```
            cin >> key >> val;
            dll.insertAfter(key, val);
            break;
        case 4:
            cout << "Enter key and value: ";
            cin >> key >> val;
            dll.insertBefore(key, val);
            break;
        case 5:
            cout << "Enter value to delete: ";
            cin >> val;
            dll.deleteNode(val);
            break;
        case 6:
            cout << "Enter value to search: ";
            cin >> val;
            cout << (dll.searchNode(val) ? "Found!\n" : "Not found!\n");
            break;
        case 7:
            dll.display();
            break;
        case 8:
            cout << "Enter value: ";
            cin >> val;
            cdll.insertAtEnd(val);
            break;
        case 9:
            cout << "Enter value to delete: ";
            cin >> val;
            cdll.deleteNode(val);
            break;
        case 10:
            cdll.display();
            break;
        case 0:
```

```
            return 0;
        default:
            cout << "Invalid choice!\n";
    }
  }
}
```

```
 MENU
Enter choice: 1
Enter value: 2

 MENU
Enter choice: 1
Enter value: 4

 MENU
Enter choice: 1
Enter value: 5

 MENU
Enter choice: 0


=== Code Execution Successful ===
```

Display all the node values in a circular linked list, repeating value of head node at the end too.

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

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

class CircularLinkedList {
private:
   Node* head;

public:
   CircularLinkedList() { head = nullptr; }

   void insertAtEnd(int val) {
      Node* newNode = new Node{val, nullptr};
      if (!head) {
         head = newNode;
         newNode->next = head;
         return;
      }
      Node* temp = head;
      while (temp->next != head)
         temp = temp->next;
      temp->next = newNode;
      newNode->next = head;
   }

   void displayWithHeadRepeat() {
      if (!head) {
         cout << "List is empty.\n";
         return;
      }
```

```
        Node* temp = head;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        cout << head->data << endl;
    }
};

int main() {
    CircularLinkedList cll;

    // Sample input
    cll.insertAtEnd(20);
    cll.insertAtEnd(100);
    cll.insertAtEnd(40);
    cll.insertAtEnd(80);
    cll.insertAtEnd(60);

    cout << "Circular Linked List: ";
    cll.displayWithHeadRepeat();

    return 0;
}
```

```
Circular Linked List: 20 100 40 80 60 20


=== Code Execution Successful ===
```

Write a program to find size of
i. Doubly Linked List.

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

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

class DoublyLinkedList {
private:
   Node* head;

public:
   DoublyLinkedList() { head = nullptr; }

   void insertAtEnd(int val) {
      Node* newNode = new Node{val, nullptr, nullptr};
      if (!head) {
         head = newNode;
         return;
      }
      Node* temp = head;
      while (temp->next)
         temp = temp->next;
      temp->next = newNode;
      newNode->prev = temp;
   }

   int size() {
      int count = 0;
      Node* temp = head;
      while (temp) {
         count++;
```

```
            temp = temp->next;
        }
        return count;
    }

    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList dll;
    dll.insertAtEnd(10);
    dll.insertAtEnd(20);
    dll.insertAtEnd(30);
    dll.insertAtEnd(40);

    cout << "Doubly Linked List: ";
    dll.display();
    cout << "Size of DLL = " << dll.size() << endl;

    return 0;
}
```

```
Doubly Linked List: 10 20 30 40
Size of DLL = 4
```

Write a program to find size of
ii. Circular Linked List.

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

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

class CircularLinkedList {
private:
   Node* head;

public:
   CircularLinkedList() { head = nullptr; }

   void insertAtEnd(int val) {
      Node* newNode = new Node{val, nullptr};
      if (!head) {
         head = newNode;
         newNode->next = head;
         return;
      }
      Node* temp = head;
      while (temp->next != head)
         temp = temp->next;
      temp->next = newNode;
      newNode->next = head;
   }

   int size() {
      if (!head) return 0;
      int count = 0;
      Node* temp = head;
     do {
```

```
            count++;
            temp = temp->next;
        } while (temp != head);
        return count;
    }
    void display() {
        if (!head) {
            cout << "List is empty\n";
            return; }
        Node* temp = head;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        cout << endl;
    }
};
int main() {
    CircularLinkedList cll;
    cll.insertAtEnd(20);
    cll.insertAtEnd(100);
    cll.insertAtEnd(40);
    cll.insertAtEnd(80);
    cll.insertAtEnd(60);
    cout << "Circular Linked List: ";
    cll.display();
    cout << "Size of CLL = " << cll.size() << endl;
    return 0;
}
```

```
Circular Linked List: 20 100 40 80 60
Size of CLL = 5
```

Write a program to check if a doubly linked list of characters is palindrome or not.

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

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

class DoublyLinkedList {
private:
   Node* head;
   Node* tail;

public:
   DoublyLinkedList() {
      head = nullptr;
      tail = nullptr;
   }

   void insertAtEnd(char ch) {
      Node* newNode = new Node{ch, nullptr, nullptr};
      if (!head) {
         head = tail = newNode;
      } else {
         tail->next = newNode;
         newNode->prev = tail;
         tail = newNode;
      }
   }

   void display() {
      Node* temp = head;
      while (temp) {
```

```
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    bool isPalindrome() {
        if (!head || !head->next)
            return true;

        Node* left = head;
        Node* right = tail;

        while (left != right && right->next != left) {
            if (left->data != right->data)
                return false;
            left = left->next;
            right = right->prev;
        }
        return true;
    }
};

int main() {
    DoublyLinkedList dll;
    string str;

    cout << "Enter a string: ";
    cin >> str;

    for (char c : str)
        dll.insertAtEnd(c);

    cout << "Doubly Linked List: ";
    dll.display();
```

```
    if (dll.isPalindrome())
        cout << "The list is a palindrome." << endl;
    else
        cout << "The list is NOT a palindrome." << endl;

    return 0;
}
```

```
Enter a string: aadita
Doubly Linked List: a a d i t a
The list is NOT a palindrome.
```

```
Enter a string: helloolleh
Doubly Linked List: h e l l o o l l e h
The list is a palindrome.
```