Assignment 6

AQ1

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


class Node {
public:
    int data;
    Node* next;
    Node* prev;
    Node(int data) {
        this->data = data;
        this->next = NULL;
        this->prev = NULL;
    }
    ~Node() {
        cout << "Memory freed for node with data " << data <<
endl;
    }
};
```

```
class CircularList {
public:
    Node* head;
    CircularList() { head = NULL; }

    void insertAtBeginning(int d) {
        Node* n = new Node(d);
        if (!head) {
            head = n;
            n->next = head;
            return;
        }
        Node* temp = head;
        while (temp->next != head) temp = temp->next;
        n->next = head;
        temp->next = n;
        head = n;
    }

    void insertAtEnd(int d) {
```

```
    Node* n = new Node(d);
    if (!head) {
        head = n;
        n->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) temp = temp->next;
    temp->next = n;
    n->next = head;
}

void insertAfter(int key, int d) {
    if (!head) return;
    Node* temp = head;
    do {
        if (temp->data == key) {
            Node* n = new Node(d);
            n->next = temp->next;
            temp->next = n;
            return;
```

```
        }
        temp = temp->next;
    } while (temp != head);
    cout << "Node not found" << endl;
}

void deleteNode(int key) {
    if (!head) return;
    Node* temp = head;
    Node* prev = NULL;
    do {
        if (temp->data == key) {
            if (temp == head) {
                Node* last = head;
                while (last->next != head) last = last->next;
                if (head->next == head) {
                    delete head;
                    head = NULL;
                    return;
                }
                last->next = head->next;
```

```
            Node* del = head;
            head = head->next;
            del->next = NULL;
            delete del;
            return;
        } else {
            prev->next = temp->next;
            temp->next = NULL;
            delete temp;
            return;
        }
    }
    prev = temp;
    temp = temp->next;
  } while (temp != head);
  cout << "Node not found" << endl;
}

void search(int key) {
  if (!head) {
    cout << "List is empty" << endl;
```

```
        return;
    }
    Node* temp = head;
    int pos = 1;
    do {
        if (temp->data == key) {
            cout << "Found " << key << " at position " << pos <<
endl;
            return;
        }
        temp = temp->next;
        pos++;
    } while (temp != head);
    cout << "Node not found" << endl;
  }

  void print() {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }
```

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

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

    void insertAtBeginning(int d) {
        Node* n = new Node(d);
        if (head) {
            n->next = head;
            head->prev = n;
        }
        head = n;
```

```
}

void insertAtEnd(int d) {
    Node* n = new Node(d);
    if (!head) {
        head = n;
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = n;
    n->prev = temp;
}

void insertAfter(int key, int d) {
    Node* temp = head;
    while (temp) {
        if (temp->data == key) {
            Node* n = new Node(d);
            n->next = temp->next;
            if (temp->next) temp->next->prev = n;
```

```
            n->prev = temp;
            temp->next = n;
            return;
        }
        temp = temp->next;
    }
    cout << "Node not found" << endl;
}

void insertBefore(int key, int d) {
    Node* temp = head;
    while (temp) {
        if (temp->data == key) {
            Node* n = new Node(d);
            n->next = temp;
            n->prev = temp->prev;
            if (temp->prev) temp->prev->next = n;
            else head = n;
            temp->prev = n;
            return;
        }
```

```
        temp = temp->next;
    }
    cout << "Node not found" << endl;
}

void deleteNode(int key) {
    Node* temp = head;
    while (temp) {
        if (temp->data == key) {
            if (temp->prev) temp->prev->next = temp->next;
            else head = temp->next;
            if (temp->next) temp->next->prev = temp->prev;
            temp->next = NULL;
            temp->prev = NULL;
            delete temp;
            return;
        }
        temp = temp->next;
    }
    cout << "Node not found" << endl;
}
```

```
    void search(int key) {
        Node* temp = head;
        int pos = 1;
        while (temp) {
            if (temp->data == key) {
                cout << "Found " << key << " at position " << pos <<
endl;
                return;
            }
            temp = temp->next;
            pos++;
        }
        cout << "Node not found" << endl;
    }

    void print() {
        if (!head) {
            cout << "List is empty" << endl;
            return;
        }
```

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

int main() {
    CircularList cll;
    DoublyList dll;
    int choice, ch, data, key;
    while (true) {
        cout << "\n1. Circular Linked List\n2. Doubly Linked
List\n3. Exit\nEnter choice: ";
        cin >> choice;
        if (choice == 1) {
            while (true) {
```

```
            cout << "\n--- Circular Linked List ---\n1. Insert at
Beginning\n2. Insert at End\n3. Insert After\n4. Delete
Node\n5. Search\n6. Display\n7. Back\nEnter choice: ";
            cin >> ch;
            if (ch == 1) { cout << "Enter data: "; cin >> data;
cll.insertAtBeginning(data); }
            else if (ch == 2) { cout << "Enter data: "; cin >> data;
cll.insertAtEnd(data); }
            else if (ch == 3) { cout << "Enter key: "; cin >> key;
cout << "Enter data: "; cin >> data; cll.insertAfter(key, data); }
            else if (ch == 4) { cout << "Enter data to delete: "; cin
>> key; cll.deleteNode(key); }
            else if (ch == 5) { cout << "Enter data to search: "; cin
>> key; cll.search(key); }
            else if (ch == 6) cll.print();
            else if (ch == 7) break;
            else cout << "Invalid choice" << endl;
        }
    } else if (choice == 2) {
        while (true) {
            cout << "\n--- Doubly Linked List ---\n1. Insert at
Beginning\n2. Insert at End\n3. Insert After\n4. Insert
```

```
Before\n5. Delete Node\n6. Search\n7. Display\n8. Back\nEnter choice: ";
            cin >> ch;
            if (ch == 1) { cout << "Enter data: "; cin >> data; dll.insertAtBeginning(data); }
            else if (ch == 2) { cout << "Enter data: "; cin >> data; dll.insertAtEnd(data); }
            else if (ch == 3) { cout << "Enter key: "; cin >> key; cout << "Enter data: "; cin >> data; dll.insertAfter(key, data); }
            else if (ch == 4) { cout << "Enter key: "; cin >> key; cout << "Enter data: "; cin >> data; dll.insertBefore(key, data); }
            else if (ch == 5) { cout << "Enter data to delete: "; cin >> key; dll.deleteNode(key); }
            else if (ch == 6) { cout << "Enter data to search: "; cin >> key; dll.search(key); }
            else if (ch == 7) dll.print();
            else if (ch == 8) break;
            else cout << "Invalid choice" << endl;
        }
    } else if (choice == 3) break;
    else cout << "Invalid choice" << endl;
  }
```

```
    return 0;
}
```

```
1. Circular Linked List
2. Doubly Linked List
3. Exit
Enter choice: 2

--- Doubly Linked List ---
1. Insert at Beginning
2. Insert at End
3. Insert After
4. Insert Before
5. Delete Node
6. Search
7. Display
8. Back
Enter choice: 1
Enter data: 4

--- Doubly Linked List ---
1. Insert at Beginning
2. Insert at End
3. Insert After
4. Insert Before
5. Delete Node
6. Search
7. Display
8. Back
Enter choice: 2
Enter data: 68

--- Doubly Linked List ---
1. Insert at Beginning
2. Insert at End
3. Insert After
4. Insert Before
5. Delete Node
6. Search
7. Display
8. Back
Enter choice: 7
```

```
Enter choice: 7
4 68

--- Doubly Linked List ---
1. Insert at Beginning
2. Insert at End
3. Insert After
4. Insert Before
5. Delete Node
6. Search
7. Display
8. Back
Enter choice: 8

1. Circular Linked List
2. Doubly Linked List
3. Exit
Enter choice: 3
```

AQ2

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

class Node {
public:
    int data;
    Node* next;
    Node(int data) {
        this->data = data;
        this->next = NULL;
    }
    ~Node() {
        cout << "Memory freed for node with data " << data <<
endl;
    }
};

void insertAtEnd(Node*& head, int d) {
    Node* n = new Node(d);
    if (!head) {
```

```
        head = n;
        n->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) temp = temp->next;
    temp->next = n;
    n->next = head;
}

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

int main() {
```

```
    Node* head = NULL;
    insertAtEnd(head, 20);
    insertAtEnd(head, 100);
    insertAtEnd(head, 40);
    insertAtEnd(head, 80);
    insertAtEnd(head, 60);
    display(head);
    return 0;
}
```

```
20 100 40 80 60 20
```

AQ3

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

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

```
    Node(int data) {
        this->data = data;
        this->next = NULL;
        this->prev = NULL;
    }
    ~Node() {
        cout << "Memory freed for node with data " << data <<
endl;
    }
};

void insertAtEndDoubly(Node*& head, int d) {
    Node* n = new Node(d);
    if (!head) {
        head = n;
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = n;
    n->prev = temp;
```

```
}

void insertAtEndCircular(Node*& head, int d) {
    Node* n = new Node(d);
    if (!head) {
        head = n;
        n->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) temp = temp->next;
    temp->next = n;
    n->next = head;
}

int sizeOfDoubly(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp) {
        count++;
        temp = temp->next;
```

```
    }
    return count;
}

int sizeOfCircular(Node* head) {
    if (!head) return 0;
    int count = 0;
    Node* temp = head;
    do {
        count++;
        temp = temp->next;
    } while (temp != head);
    return count;
}

int main() {
    Node* doubly = NULL;
    Node* circular = NULL;

    insertAtEndDoubly(doubly, 10);
    insertAtEndDoubly(doubly, 20);
```

```
    insertAtEndDoubly(doubly, 30);
    insertAtEndDoubly(doubly, 40);

    insertAtEndCircular(circular, 5);
    insertAtEndCircular(circular, 15);
    insertAtEndCircular(circular, 25);

    cout << "Size of Doubly Linked List: " <<
sizeOfDoubly(doubly) << endl;
    cout << "Size of Circular Linked List: " <<
sizeOfCircular(circular) << endl;

    return 0;
}
```

```
Size of Doubly Linked List: 4
Size of Circular Linked List: 3
```

AQ4

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

```
class Node {
public:
    char data;
    Node* next;
    Node* prev;
    Node(char data) {
        this->data = data;
        this->next = NULL;
        this->prev = NULL;
    }
    ~Node() {
        cout << "Memory freed for node with data " << data <<
endl;
    }
};

void insertAtEnd(Node*& head, char ch) {
    Node* n = new Node(ch);
    if (!head) {
        head = n;
        return;
```

```
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = n;
    n->prev = temp;
}

bool isPalindrome(Node* head) {
    if (!head || !head->next) return true;
    Node* left = head;
    Node* right = head;
    while (right->next) right = right->next;
    while (left != right && right->next != left) {
        if (left->data != right->data)
            return false;
        left = left->next;
        right = right->prev;
    }
    return true;
}
```

```
int main() {
    Node* head = NULL;
    string s = "RADAR";
    for (char c : s) insertAtEnd(head, c);

    if (isPalindrome(head))
        cout << "Palindrome" << endl;
    else
        cout << "Not Palindrome" << endl;

    return 0;
}
```

```
Palindrome
```

AQ5

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

class Node {
public:
    int data;
```

```
    Node* next;
    Node(int data) {
        this->data = data;
        this->next = NULL;
    }
    ~Node() {
        cout << "Memory freed for node with data " << data <<
endl;
    }
};

void insertAtEnd(Node*& head, int d) {
    Node* n = new Node(d);
    if (!head) {
        head = n;
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = n;
}
```

```
bool isCircular(Node* head) {
    if (!head) return false;
    Node* temp = head->next;
    while (temp && temp != head)
        temp = temp->next;
    return (temp == head);
}

int main() {
    Node* head = NULL;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);
    insertAtEnd(head, 40);

    if (isCircular(head))
        cout << "Circular Linked List" << endl;
    else
        cout << "Not Circular Linked List" << endl;
```

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

    if (isCircular(head))
        cout << "Circular Linked List" << endl;
    else
        cout << "Not Circular Linked List" << endl;

    return 0;
}
```

```
Not Circular Linked List
Circular Linked List
```