CIRCULAR LINKED LIST CODE IMPLEMENTATION

```
#include <iostream>


using namespace std;


// Node class definition
class Node {
public:
    int data;
    Node* next;


    Node(int data) : data(data), next(nullptr) {}
};


// CircularLinkedList class definition
class CircularLinkedList {
private:
    Node* head;


public:
    CircularLinkedList() : head(nullptr) {}


    // Function to create a new node
    Node* createNode(int data) {
        return new Node(data);
    }
```

```
// Function to insert a node at the start
void insertNodeAtStart(int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        head = newNode;
        newNode->next = head; // Point to itself
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        newNode->next = head;
        temp->next = newNode;
        head = newNode;
    }
}

// Function to insert a node at the end
void insertNodeAtEnd(int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        head = newNode;
        newNode->next = head; // Point to itself
    } else {
        Node* temp = head;
        while (temp->next != head) {
```

```
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}

// Function to insert a node at any position
void insertNodeAtAny(int data, int position) {
    if (position == 0) {
        insertNodeAtStart(data);
        return;
    }

    Node* newNode = createNode(data);
    Node* temp = head;
    for (int i = 0; i < position - 1 && temp != nullptr; ++i) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Position out of bounds" << endl;
        delete newNode;
        return;
    }

    newNode->next = temp->next;
```

```
    temp->next = newNode;
}

// Function to delete a node at the start
void deleteNodeAtStart() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        Node* temp = head;
        Node* prev = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        head = head->next;
        temp->next = head;
        delete prev;
    }
}

// Function to delete a node at the end
void deleteNodeAtEnd() {
```

```
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        Node* temp = head;
        Node* prev = nullptr;
        while (temp->next != head) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = head;
        delete temp;
    }
}

// Function to delete a node at any position
void deleteNodeAtAny(int position) {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
```

```
    if (position == 0) {
        deleteNodeAtStart();
        return;
    }

    Node* temp = head;
    Node* prev = nullptr;
    for (int i = 0; i < position && temp->next != head; ++i) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == nullptr || (prev == nullptr && position != 0)) {
        cout << "Position out of bounds" << endl;
        return;
    }

    if (temp == head) {
        deleteNodeAtStart();
    } else {
        prev->next = temp->next;
        delete temp;
    }
}

// Function to traverse and print the list
void traversal() {
```

```
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }

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

// Function to search for a node with a specific value
bool search(int data) {
    if (head == nullptr) {
        return false;
    }

    Node* temp = head;
    do {
        if (temp->data == data) {
            return true;
        }
        temp = temp->next;
    } while (temp != head);
    return false;
```

```
    }
};

// Main function to test the circular linked list
int main() {
    CircularLinkedList cll;

    cll.insertNodeAtStart(100);
    cll.insertNodeAtEnd(200);
    cll.insertNodeAtEnd(300);
    cll.insertNodeAtAny(250, 3);

    cout << "List after insertion: ";
    cll.traversal();

    cll.deleteNodeAtStart();
    cout << "List after deleting node at start: ";
    cll.traversal();

    cll.deleteNodeAtEnd();
    cout << "List after deleting node at end: ";
    cll.traversal();

    cll.deleteNodeAtAny(2);
    cout << "List after deleting node at position 1: ";
    cll.traversal();
```

```
    if (cll.search(20)) {
        cout << "20 is in the list" << endl;
    } else {
        cout << "20 is not in the list" << endl;
    }

    return 0;
}
```

OUTPUT

```
C:\Users\studentuser\Desktop\DSA\Lectures\WEEK 02\Circular LL.exe
List after insertion: 100 200 300 250
List after deleting node at start: 200 300 250
List after deleting node at end: 200 300
List after deleting node at position 1: 200
20 is not in the list

--------------------------------
Process exited after 0.101 seconds with return value 0
Press any key to continue . . .
```