Data Structures and Algorithms
Lab Assignment 5

SUBMITTED BY:

Hasaan Ahmad SP22-BSE-017

SUBMITTED TO: Sir Syed Ahmad Qasim

Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node(int data)
    {
        this->data = data;
        this->next = NULL;
    }
};
class CircularLinkedList
{
public:
    Node *head;
    Node *tail;
    CircularLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }
    void insert(int data)
    {
        Node *newNode = new Node(data);
        if (head == nullptr)
        {
            head = newNode;
            tail = newNode;
            newNode->next = head;
        }
        else
        {
            tail->next = newNode;
            tail = newNode;
            tail->next = head;
        }
    }
    void insertAtHead(int data)
    {
        Node *newNode = new Node(data);
        if (head == nullptr)
        {
```

```
            head = newNode;
            tail = newNode;
            newNode->next = head;
        }
        else
        {
            newNode->next = head;
            head = newNode;
            tail->next = head;
        }
    }
    void print()
    {
        Node *temp = head;
        while (temp->next != head)
        {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << temp->data << " ";
    }
    void returnIndex(int key)
    {
        Node *temp = head;
        int index = 0;
        while (temp->next != head)
        {
            if (temp->data == key)
            {
                cout << index << endl;
                return;
            }
            temp = temp->next;
            index++;
        }
        if (temp->data == key)
        {
            cout << index << endl;
            return;
        }
        cout << "Not Found" << endl;
    }
    void deleteNode(int key)
    {
        Node *temp = head;
```

```
        if (head->data == key)
        {
            head = head->next;
            tail->next = head;
            delete temp;
            return;
        }
        while (temp->next != head)
        {
            if (temp->next->data == key)
            {
                Node *toDelete = temp->next;
                temp->next = temp->next->next;
                delete toDelete;
                return;
            }
            temp = temp->next;
        }
        cout << "Not Found" << endl;
    }
    void deleteComplete()
    {
        Node *temp = head;
        while (temp->next != head)
        {
            Node *toDelete = temp;
            temp = temp->next;
            delete toDelete;
        }
        delete temp;
        head = nullptr;
        tail = nullptr;
    }
    // A function which can delete all even number values nodes from the linked
list
    void deleteEven()
    {
        Node *temp = head;
        while (temp->next != head)
        {
            if (temp->next->data % 2 == 0)
            {
                Node *toDelete = temp->next;
                temp->next = temp->next->next;
                delete toDelete;
```

```
            }
            else
            {
                temp = temp->next;
            }
        }
        if (head->data % 2 == 0)
        {
            Node *toDelete = head;
            head = head->next;
            tail->next = head;
            delete toDelete;
        }
    }
    // A function which can delete all odd number values nodes from the linked
list
    void deleteOdd()
    {
        Node *temp = head;
        while (temp->next != head)
        {
            if (temp->next->data % 2 != 0)
            {
                Node *toDelete = temp->next;
                temp->next = temp->next->next;
                delete toDelete;
            }
            else
            {
                temp = temp->next;
            }
        }
        if (head->data % 2 != 0)
        {
            Node *toDelete = head;
            head = head->next;
            tail->next = head;
            delete toDelete;
        }
    }
    // Josephus Problem
    void josephus(int k)
    {
        Node *temp = head;
        while (temp->next != head)
```

```
        {
            for (int i = 0; i < k - 2; i++)
            {
                temp = temp->next;
            }
            Node *toDelete = temp->next;
            temp->next = temp->next->next;
            temp = temp->next;
            delete toDelete;
        }
        cout << temp->data << endl;
    }

    // A function which can delete all even positioned nodes
    void deleteEvenPosition()
    {
        Node *temp = head;
        int index = 0;
        while (temp->next != head)
        {
            if (index % 2 == 0)
            {
                Node *toDelete = temp->next;
                temp->next = temp->next->next;
                delete toDelete;
            }
            else
            {
                temp = temp->next;
            }
            index++;
        }
        if (index % 2 == 0)
        {
            Node *toDelete = head;
            head = head->next;
            tail->next = head;
            delete toDelete;
        }
    }
};

int main()
{
    // Testing Circular Linked List
```

```
    CircularLinkedList cll;
    cll.insert(1);
    cll.insert(2);
    cll.insert(3);
    cll.insert(4);
    cll.insert(5);
    cll.insert(6);
    cll.insert(7);
    cll.insert(8);
    cll.insert(9);
    cll.insert(10);
    cll.insertAtHead(0);
    cll.print();
    cout << endl;
    cout << "Testing delete Methods" << endl;
    cll.deleteNode(0);
    cll.deleteNode(10);
    cll.print();
    cout << endl;
    cout << "testing return index" << endl;
    cll.returnIndex(5);
    // Removed
    cll.returnIndex(10);
    // testing delete even
    cout << "before deleting even" << endl;
    cll.print();
    cout << endl;
    cout << "testing delete even" << endl;
    cll.deleteEven();
    cll.print();
    cout << endl;
    //    Making a new linked List
    CircularLinkedList cll2;
    cll2.insert(1);
    cll2.insert(3);
    cll2.insert(2);
    cll2.insert(4);
    cll2.insert(5);
    cll2.insert(6);
    cll2.insertAtHead(12);
    cll2.print();
    cout << endl;
    // testing delete odd
    cout << "testing delete odd";
    cll2.deleteOdd();
```

```
    cout << endl;
    cll2.print();
    cout << endl;
    cout<<"Testing Josephus Problem"<<endl;
    // testing josephus problem
    CircularLinkedList cll3;
    cll3.insert(1);
    cll3.insert(2);
    cll3.insert(3);
    cll3.insert(4);
    cll3.insert(5);
    cll3.insert(6);
    cll3.insert(7);
    cll3.insert(8);
    cll3.insert(9);
    cll3.insert(10);
    cll3.josephus(2);
    // testing delete even positions
  CircularLinkedList* l1 = new CircularLinkedList();
    l1->insert(1);
    l1->insert(2);
    l1->insert(3);
    l1->insert(4);
    l1->insert(5);
    l1->print();
    cout<<endl;
    // Deleting Even  Positions
    l1->deleteEvenPosition();
    l1->print();

    return 0;
}
```

Output:

```
3 5
PS D:\Ishtudy Material\3rd Sem\DSA\LAB\LAB 05> cd "d:\Ishtudy Material\3rd Sem\DSA\LAB\LAB 05\" ;
0 1 2 3 4 5 6 7 8 9 10
Testing delete Methods
1 2 3 4 5 6 7 8 9
testing return index
4
Not Found
before deleting even
1 2 3 4 5 6 7 8 9
testing delete even
1 3 5 7 9
12 1 3 2 4 5 6
testing delete odd
12 2 4 6
Testing Josephus Problem
9
1 2 3 4 5
Deleting Even Positions
3 5
```