Data Structure Lab

Lab # 04

Singly Circular Linked List

Instructor: Muhammad Saad Khan

Email: saad.khan@nu.edu.pk

Course Code: CL2001

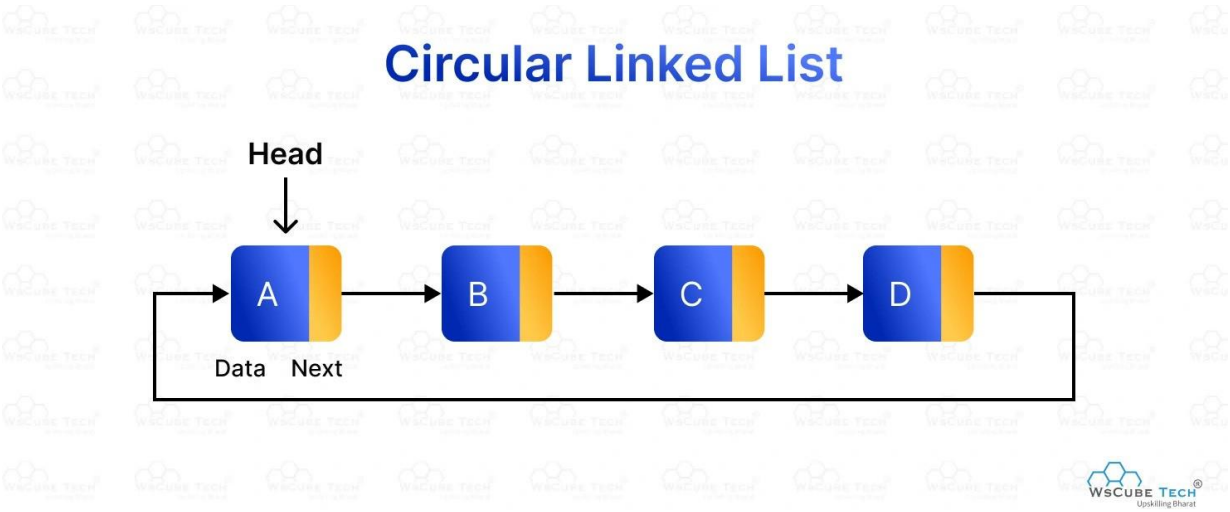
National University of Computer and Emerging Sciences FAST Peshawar Campus

Singly Circular Linked List:

A Singly Circular Linked List is a variation of a singly linked list in which the last node points back to the first node, forming a circle. Unlike a normal singly linked list, there is no NULL pointer at the end.

Characteristics:

- Each node contains data and a pointer to the next node.
- The last node points to the head, creating a circular structure.
- Traversal can start from any node and eventually return to the starting node.
- The list can be traversed infinitely if a stopping condition is not applied.



Deletion of All Nodes through Destructor:

- When the linked list object is destroyed, the destructor runs automatically.
- It traverses the linked list, deletes each node using delete, and releases the memory back to the system.
- This prevents memory leaks, which occur if dynamically allocated memory is not freed.

Code Example:

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

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

    Node(int val) {
        data = val;
        next = NULL;
    }
};

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = NULL;
    }

    // Insert at end
    void insert(int val) {
        Node* newNode = new Node(val);
```

```
    if (head == NULL) {
        head = newNode;
        head->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }

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

// Display linked list
void display() {
    Node* temp = head;
    do {
      cout << temp->data << " -> ";
        temp = temp->next;
                }
    while (temp != head);
    cout << "NULL" << endl;
}

// Destructor: delete all nodes
~ LinkedList() {
      Node * temp = head;
      while(temp!=head) {
```

```
            Node * temp2 = temp->next;
            cout<<"The deleting node is "<< temp->data<<endl;
            delete temp;
            temp = temp2;
            }
        }

};

int main() {
   {
      LinkedList list;

      list.insert(10);
      list.insert(20);
      list.insert(30);
      list.insert(40);
      list.insert(50);

      cout << "Linked List: ";
      list.display();
   } // destructor automatically called here when list goes out of scope

   cout << "All nodes deleted automatically!" << endl;
   return 0;
}
```

Deletion in Singly Linked List:

Deletion in a Singly Linked List means removing a node from the list and updating pointers so that the list remains connected.

Cases of Deletion:

1. **Deleting the first node (head):**

   - Update the head pointer to point to the second node.
   - Free memory of the old head node.

2. **Deleting a node in the middle:**

   - Traverse the list to find the node before the target node.
   - Change its next pointer to skip the target node.
   - Free memory of the target node.

3. **Deleting the last node:**

   - Traverse until the second-last node.
   - Set its next pointer to NULL.
   - Free memory of the last node.

4. **Deleting a node by value (general case):**

   - Search for the node with the given value.
   - Adjust pointers to remove it from the chain.
   - If value not found, no deletion occurs.

Delete Target Node Code Example:

```
// Delete target node
void deleteNode(int target) {
    if (head == NULL) {
        cout << "List is empty, nothing to delete." << endl;
        return;
    }

    Node* curr = head;
    Node* prev = NULL;

    // Case 1: head node is the target
    if (head->data == target) {
        // Only one node
        if (head->next == head) {
            cout << "Deleting node with value " << head->data << endl;
            delete head;
            head = NULL;
            return;
        }

        // Find last node to fix circular link
        Node* last = head;
        while (last->next != head) {
            last = last->next;
        }

        cout << "Deleting node with value " << head->data << endl;
        Node* temp = head;
```

```
        head = head->next;    // move head forward
        last->next = head;    // fix circle
        delete temp;
        return;
    }

    // Case 2: find target elsewhere
    do {
        prev = curr;
        curr = curr->next;

        if (curr->data == target) {
            prev->next = curr->next;
            cout << "Deleting node with value " << curr->data << endl;
            delete curr;
            return;
        }

} while (curr != head);

    cout << "Value " << target << " not found in list." << endl;
}
```