Data Structure Lab

Lab # 05

Doubly And Circular Linked List

Instructor: Muhammad Saad Khan

Email: saad.khan@nu.edu.pk

Course Code: CL2001

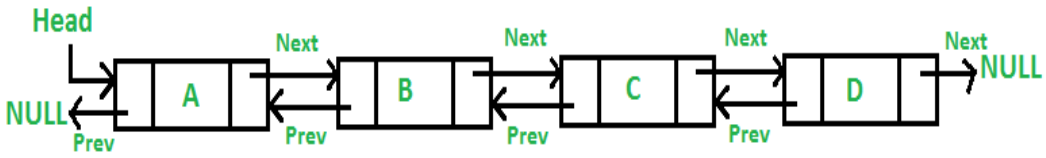
National University of Computer and Emerging Sciences FAST Peshawar Campus

Prepared By: M Saad Khan

Doubly Linked List:

Doubly Linked List in C++ is very similar to a linked list, except that each node also contains a pointer to the node previous to the current one. This means that in a doubly linked list in C++ we can travel not only in the forward direction, but also in the backward direction, which was not possible with a palin linked list in C++.

Here, we have the same node class, the only difference is that now we have a prev pointer which points to the previous node.




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

		Node(int val)
		{
			this->val=val;
			next=NULL;
			prev=NULL;
		}
};
```

```
~LinkedList()
{
        Node *curr=head;
        while(head!=NULL)
        {
                curr=head;
                head=head->next;

         delete curr;
        }
}
```

Doubly Linked Code Example:

1. Node Class

- The Node class represents a single node of the doubly linked list.
- Each node has three parts:
    - `data` → stores the actual value.
    - `prev` → pointer to the previous node.
    - `next` → pointer to the next node.
- The constructor initializes the `data` with the given value and sets `prev` and `next` to `nullptr`.

---

2. DoublyLinkedList Class

This class manages the entire linked list. It contains a `head` pointer which always points to the first node of the list.

a. Constructor

- Initializes the list with `head = nullptr`, meaning the list is empty at the start.

b. insertAtEnd(int value)

- Creates a new node with the given value.
- If the list is empty, the new node becomes the `head`.
- Otherwise, it traverses the list until the last node and attaches the new node at the end.
- The `prev` pointer of the new node is updated to point to the last node.

c. insertAtBeginning(int value)

- Creates a new node with the given value.
- If the list is empty, the new node becomes the `head`.
- Otherwise, it sets the `next` pointer of the new node to the current `head`, and updates the `prev` of the old head.
- Finally, updates `head` to point to the new node.

d. deleteNode(int value)

- Searches for the node containing the given value.
- If not found, prints a message.
- If found:
    - Adjusts the `next` pointer of the previous node and the `prev` pointer of the next node to bypass the node being deleted.
    - Special case: If the node to delete is the `head`, then `head` is updated to the next node.
    - Deletes the node to free memory.

e. displayForward()

- Traverses the list from `head` to the last node using `next` pointers.
- Prints all the data values in forward order.

f. displayBackward()

- Traverses the list to reach the last node first.
- Then moves backward using `prev` pointers and prints the data values in reverse order.

---

3. Main Function

- Creates a `DoublyLinkedList` object.
- Inserts values: `10, 20, 30` at the end and `5` at the beginning.
- Displays the list forward and backward.
- Deletes the node with value `20` and again displays the list in both directions.

Code Example:

```
#include <iostream>

using namespace std;


// Node class for doubly linked list

class Node {

public:

    int data;

    Node* prev;

    Node* next;
```

```
    Node(int value) {
        data = value;
        prev = nullptr;
        next = nullptr;
    }
};

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

public:
    DoublyLinkedList() {
        head = nullptr;
    }

    // Insert at the end
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
```

```
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

// Insert at the beginning
void insertAtBeginning(int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
        return;
    }
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
}
```

```
// Delete a node by value
void deleteNode(int value) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        cout << "Value not found in the list.\n";
        return;
    }
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next; // Deleting the head node
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }
    delete temp;
```

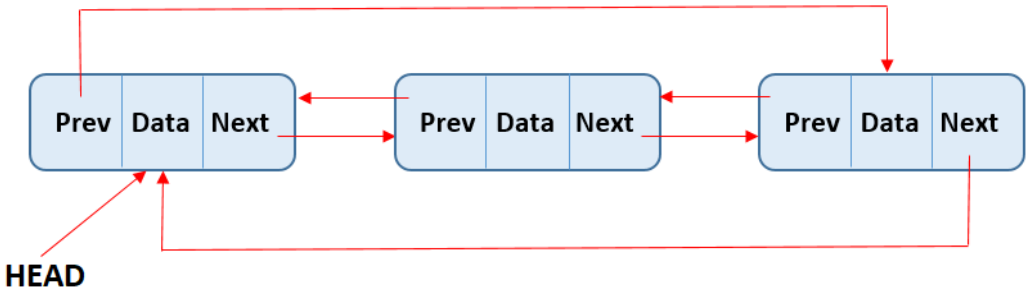
```
}

// Display list forward
void displayForward() {
    Node* temp = head;
    cout << "List (forward): ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Display list backward
void displayBackward() {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next; // go to last node
    }
```

```
        cout << "List (backward): ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->prev;
        }
        cout << endl;
    }
};

// Main function
int main() {
    DoublyLinkedList dll;
    dll.insertAtEnd(10);
    dll.insertAtEnd(20);
    dll.insertAtEnd(30);
    dll.insertAtBeginning(5);

    dll.displayForward();
    dll.displayBackward();

    dll.deleteNode(20);
    dll.displayForward();
    dll.displayBackward();
```

```
    return 0;
}
```

Doubly Circular Linked List:

- Each node has three fields: data, a pointer to the next node, and a pointer to the previous node.
- The last node's next pointer points back to the first node, and the first node's previous pointer points to the last node, forming a circular structure.
- This allows traversal in both directions (forward and backward) starting from any node.



Doubly Circular Linked List Code Example:

Short Description of the Code:

1. **Node Class**
    - Stores data, a pointer to the next node, and a pointer to the previous node.
2. **DoublyCircularLinkedList Class**
    - insert(int val): Inserts a new node at the end. If the list is empty, it links the node to itself (circular).

- deleteNode(int target): Deletes the node containing the target value. Adjusts both next and prev pointers to maintain circular structure. Special handling for deleting head or the last remaining node.

- display(): Traverses the circular list and prints all node values.

3. **Main Function**

   - Demonstrates insertion of nodes, deletion by target value, and displaying the list after each operation.

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

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

    Node(int val) {
        data = val;
        next = nullptr;
        prev = nullptr;
    }
};

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

```
public:
    DoublyCircularLinkedList() {
        head = nullptr;
    }

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

        if (head == nullptr) {
            head = newNode;
            head->next = head;
            head->prev = head;
        } else {
            Node* last = head->prev;

            last->next = newNode;
            newNode->prev = last;

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

    // Delete by target value
    void deleteNode(int target) {
        if (head == nullptr) {
            cout << "List is empty, cannot delete.\n";
            return;
        }

        Node* curr = head;
        Node* prevNode = nullptr;

        // Search target
        do {
            if (curr->data == target) {
                if (curr->next == curr && curr->prev == curr) {
                    // Only one node
                    delete curr;
```

```
            head = nullptr;
            return;
        }

        if (curr == head) {
            head = head->next;
        }

        curr->prev->next = curr->next;
        curr->next->prev = curr->prev;

        delete curr;
        return;
      }
      prevNode = curr;
      curr = curr->next;
    } while (curr != head);

    cout << "Target not found in the list.\n";
  }

  // Display the list
  void display() {
    if (head == nullptr) {
      cout << "List is empty.\n";
      return;
    }

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

// Main function
int main() {
```

```
    DoublyCircularLinkedList  list;

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

    list.display();

    list.deleteNode(20);
    list.display();

    list.deleteNode(10);
    list.display();

    list.deleteNode(50);  // Target not in list

    return 0;
}
```