Riphah International University
I-14 Main Campus
Faculty of Computing

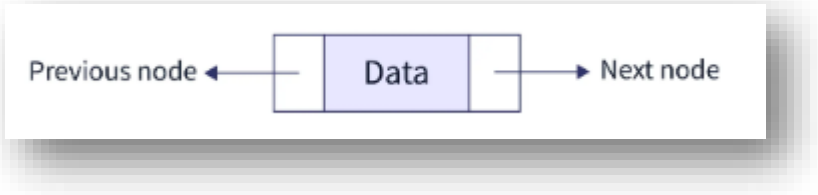
Class:	Fall-2024	**Subject:**	Data Structures & Algorithms
Course Code:	CS 2124	**Lab Instructor:**	Zeeshan Ali

Learning Objective:

- Double linked list
- Double linked list operations
- Implementation of Double linked list in C++
- Operations on Double linked list
- Insert
- Delete
- Print
- Lab Tasks

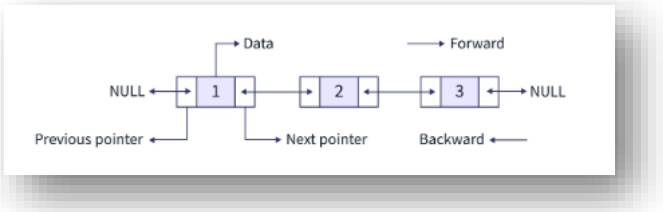
Double linked list:

Double linked list is a sequence of elements in which every element has **links** to its **previous element** and **next element** in the sequence.



Double linked list is a **two-way list** because one can move in either from **left to right** or from **right to left**.

Every node has **link** to its **previous node** and **next node**. So, we can traverse **forward** by using **next field** and can **traverse backward** by using **previous field**. **Every node** in double linked list contains **three fields**.



Double linked list operations:

- Insertion
- Deletion
- Display

Double linked list operations:

Insertion

Operation can be performed in three ways:

1. Insertion at beginning of the list
2. Insertion at end of the list
3. Insertion at specific location in the list

Deletion

1. Deletion at beginning of the list
2. Deletion at end of the list
3. Deletion at specific location in the list

Display

1. Forward
2. Backward

Implementation of Double linked list in C++

DoubleLinkedList.cpp

```cpp
#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;
    }

};
```

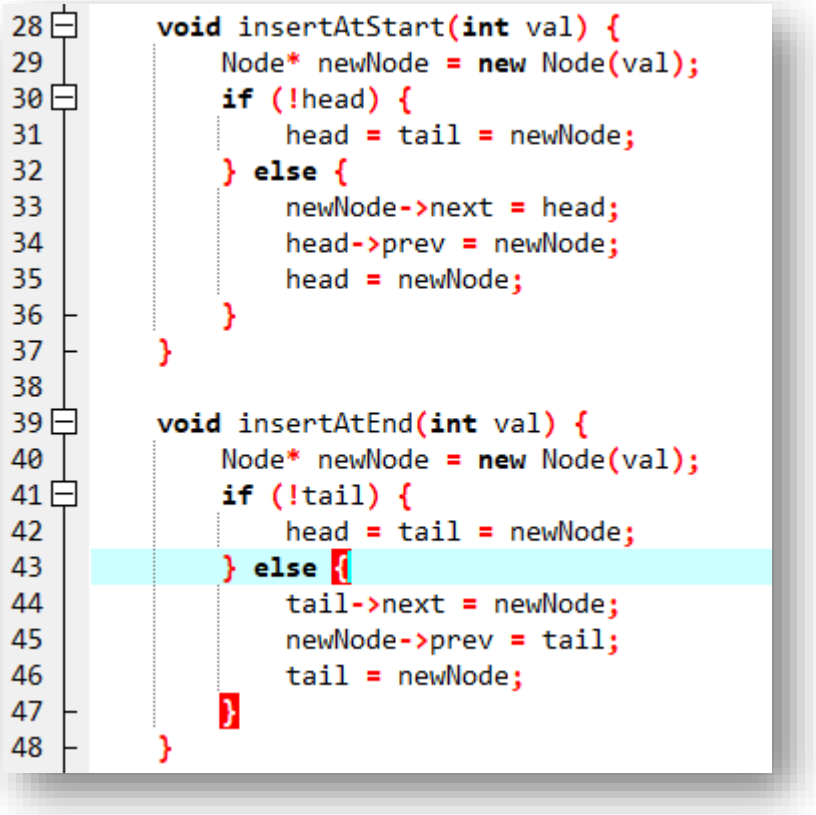
```cpp

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList()
    { head = NULL;
      tail = NULL;
    }

```

Insert



```
28      void insertAtStart(int val) {
29          Node* newNode = new Node(val);
30          if (!head) {
31              head = tail = newNode;
32          } else {
33              newNode->next = head;
34              head->prev = newNode;
35              head = newNode;
36          }
37      }
38
39      void insertAtEnd(int val) {
40          Node* newNode = new Node(val);
41          if (!tail) {
42              head = tail = newNode;
43          } else {
44              tail->next = newNode;
45              newNode->prev = tail;
46              tail = newNode;
47          }
48      }
```

```
50  void insertAtPosition(int val, int position) {
51      if (position < 1) {
52          cout << "Position Invalid." <<endl;
53          return;
54      }
55      Node* newNode = new Node(val);
56      if (position == 1) {
57          insertAtStart(val);
58      } else {
59          Node* current = head;
60          int currentPosition = 1;
61
62          while (current && currentPosition < position - 1) {
63              current = current->next;
64              currentPosition++;
65          }
66          if (!current) {
67              cout << "Invalid Position." <<endl;
68              delete newNode;
69              return;
70          }
71          newNode->next = current->next;
72          newNode->prev = current;
73          if (current->next) {
74              current->next->prev = newNode;
75          }
76          current->next = newNode;
77      }
78  }
```

Delete

```
80      void deleteFromStart() {
81          if (!head) {
82              cout << "List is empty." <<endl;
83              return;
84          }
85
86          Node* temp = head;
87          head = head->next;
88          if (head) {
89              head->prev = NULL;
90          } else {
91              tail = NULL;
92          }
93          delete temp;
94      }
```

```
96      void deleteFromEnd() {
97          if (!tail) {
98              cout << "List is empty." <<endl;
99              return;
00          }
01
02          Node* temp = tail;
03          tail = tail->prev;
04          if (tail) {
05              tail->next = NULL;
06          } else {
07              head = NULL;
08          }
09          delete temp;
10      }
```

Print

```
112     void printList() {
113         Node* current = head;
114         while (current) {
115             cout << current->data << " ";
116             current = current->next;
117         }
118         cout <<endl;
119     }
120 };
121
122 int main() {
```

```
21
22  int main() {
23      DoublyLinkedList Dlist;
24
25      Dlist.insertAtStart(10);
26      Dlist.insertAtEnd(11);
27      Dlist.insertAtStart(5);
28      Dlist.insertAtPosition(12, 2);
29      Dlist.printList();
30      Dlist.deleteFromStart();
31      Dlist.deleteFromEnd();
32      Dlist.printList();
33      Dlist.insertAtEnd(15);
```

```
E:\00 Ripha Uni\CS3 Data Structures & Algorithms(Male)\CS3-2 Data Structures & Al
5 12 10 11
12 10
12 14 10 15

--------------------------------
Process exited after 1.547 seconds with return value 0
Press any key to continue . . .
```

Lab Task.

1. Add a new function insertAtMiddle() that will insert a node at the middle of the list (i.e., after the middle node for odd-sized lists, before the middle node for even-sized lists)
2. Add a new function deleteByValue() that will delete a node by searching for the first occurrence of a given value. Make sure to handle cases where the value is not found in the list.
3. Add a new function countNodes() that returns the total number of nodes in the list.
4. Implement a function mergeLists() that merges two doubly linked lists into one.
5. Write a program to get input [NAME, SEMESTER and SAP_ID of student] from user at least 7 inputs.

Get input from user to:

I. Insert input at any location.
II. Delete input from start and end.
III. Display record in presentable form.