# Data Structures and Algorithms

Mr. Tahir Iqbal

*tahir.iqbal@bahria.edu.pk*

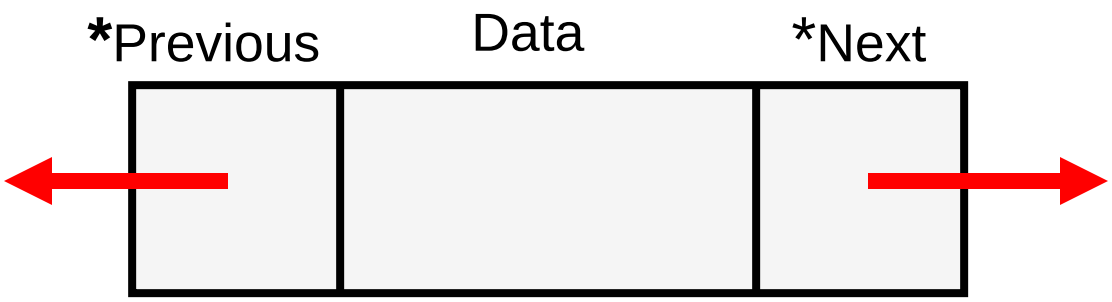
Lecture 04: Doubly Linked List

# Introduction

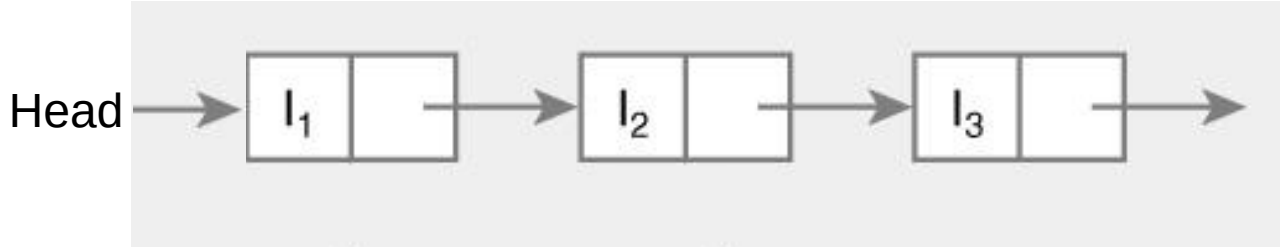
- The singly linked list contains only one pointer field i.e. every node holds an address of next node.
- The singly linked list is uni-directional i.e. we can only move from one node to its successor.
- This limitation can be overcome by **Doubly linked list.**

# Doubly Linked List

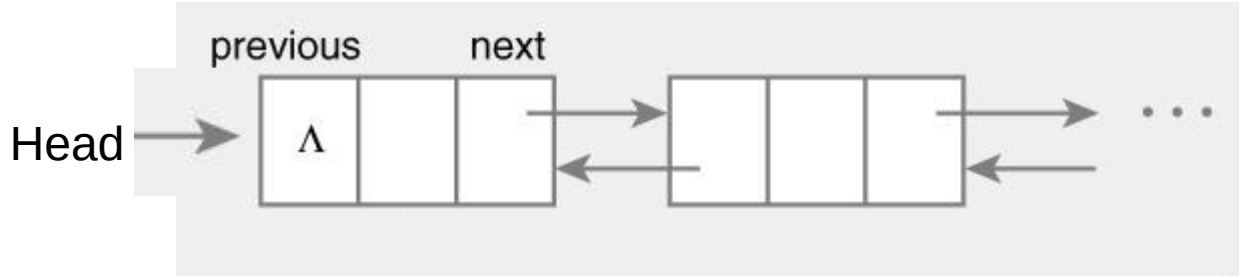
- In Doubly linked list, each node has two pointers.
- One pointer to its successor (NULL if there is none) and one pointer to its predecessor (NULL if there is none).
- These pointers enable bi-directional traversing.



# A Singly Linked List



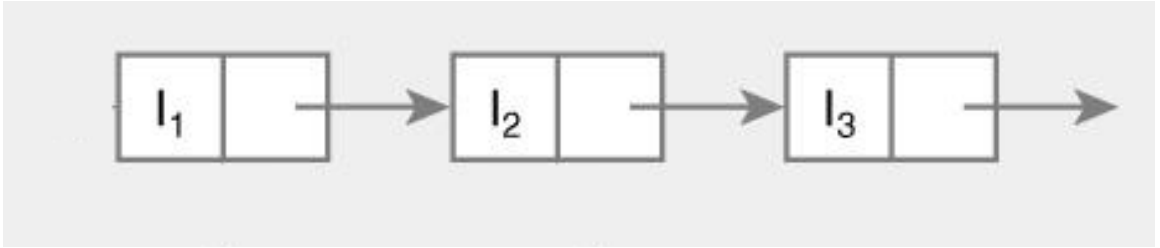

# A Doubly Linked List



# Comparison of Linked Lists

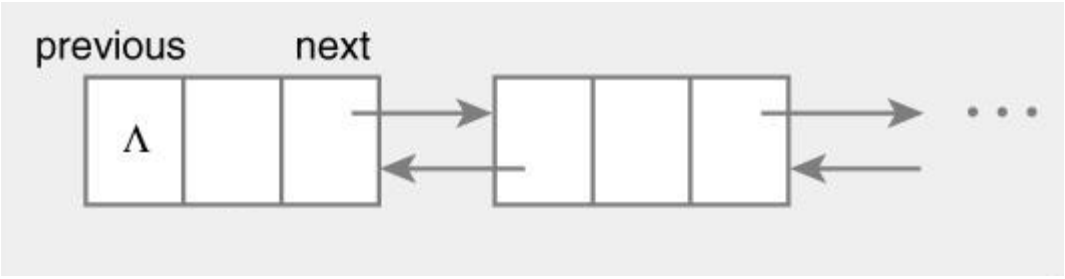
- Linked list

```
struct Node {
        int data;
        Node* next;
};
```




- Doubly linked list

```
struct Node {
        Node *previous;
        int data;
        Node *next;
};
```

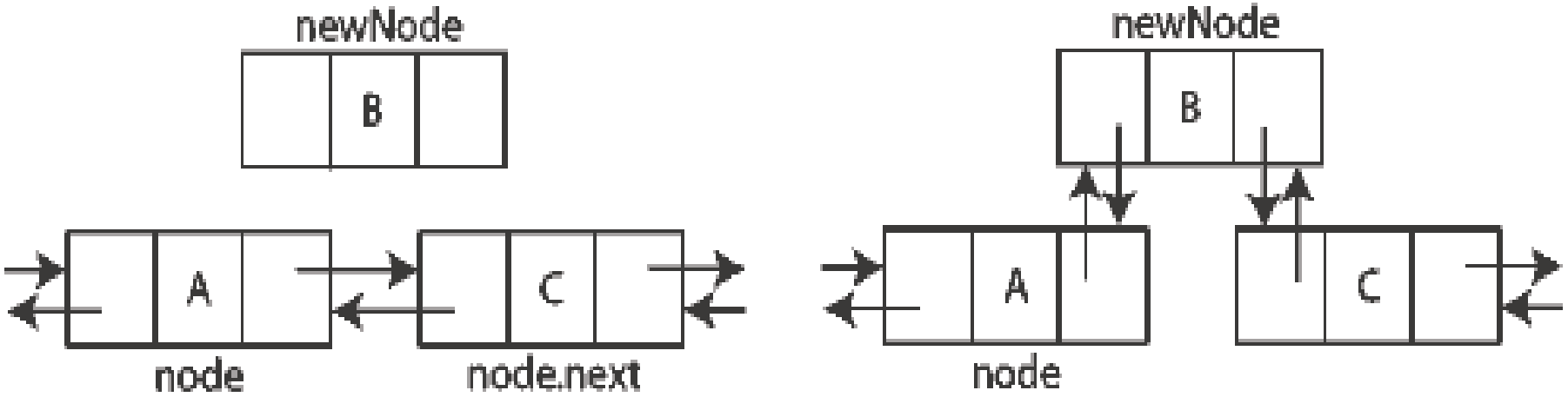


# Insertion

- In insertion process, element can be inserted in three different places
  - At the beginning of the list
  - At the end of the list
  - At the specified position.

- To insert a node in doubly linked list, you must update pointers in both predecessor and successor nodes.

# Insertion



# Doubly Linked List

```
struct DoublyList{
  int data;
  DoublyList * prev;
  DoublyList * next;
};
DoublyList * first = NULL;
DoublyList * last = NULL;
void main(){
  //switch statement
  insert(10);
  insert(20);
  delete();
  traverse();
}
```

```
void insert(){
  DoublyList *  newNode = new
    DoublyList;
  if (last == NULL){
    newNode->prev =
      newNode->next = NULL;
    first = last = newNode;
  }
  else{
    newNode->next = NULL;
    newNode->prev = last;
    last->next = newNode;
    last = newNode;
  }
}
```

# Doubly Linked List

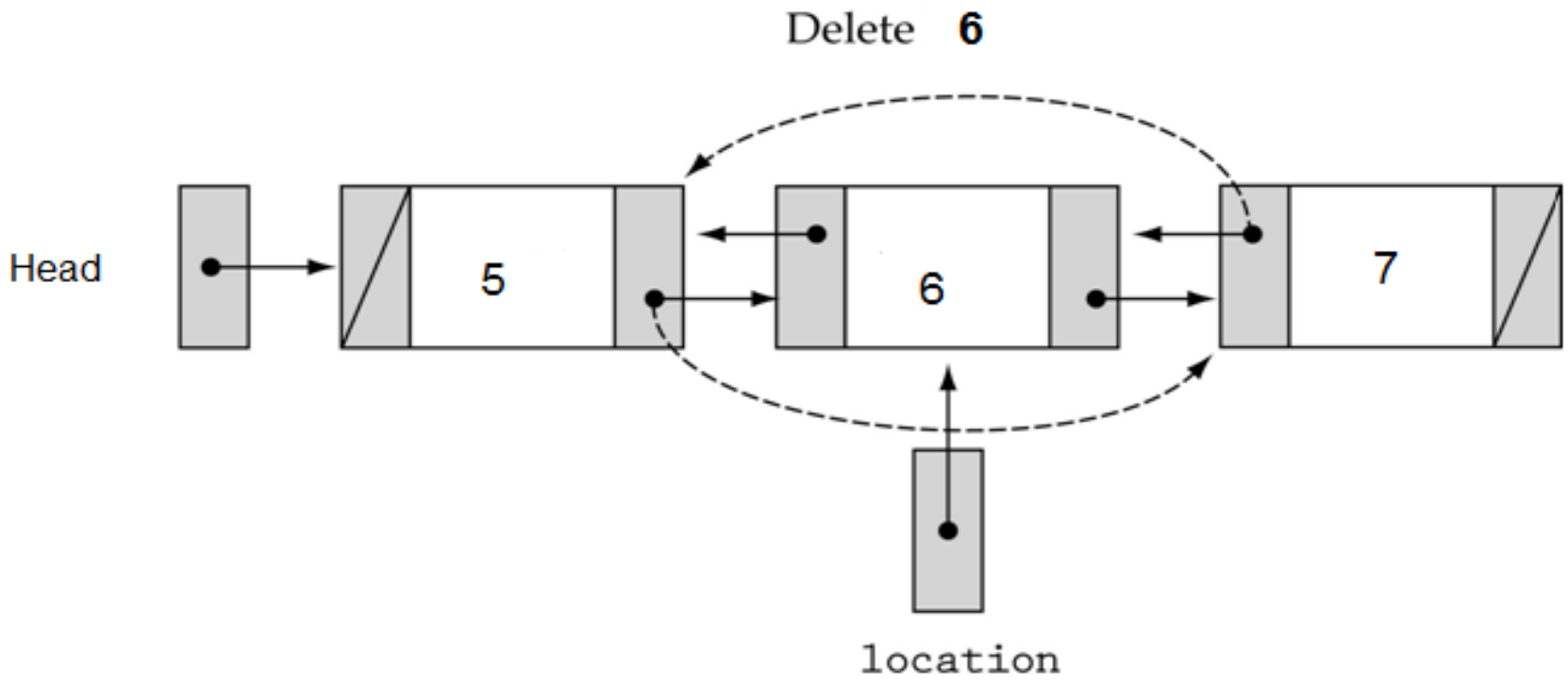
```
void traverse(){
  DoublyList * temp = front;
  while(temp!=NULL){
    cout<<temp->data;
    temp = temp->next;
  }
}
void insertAT(int x){
// assuming that list is in ascending
   order
  DoublyList * temp = front;
  DoublyList * newNode = new
   DoublyList;
newNode->data = x;
  while(temp!=NULL){
    if(temp->data > x){
      newNode->next = temp;
      newNode->prev = temp->prev;
      temp->prev->next = newNode;
      temp->prev = newNode;
    }
    temp = temp->next;
  }

}
```

# Deletion

- In deletion process, element can be deleted from three different places
  - From the beginning of the list
  - From the end of the list
  - From the specified position in the list.

- When the node is deleted, the memory allocated to that node is released and the previous and next nodes of that node are linked

# Deletion



# Doubly Linked List

```
void delete(int x){
  DoublyList * temp = front;
  DoublyList * toDelete = NULL;
  while(temp!=NULL){
    if(temp->data == x){
      temp->prev->next = temp-next;
      temp->next->prev = temp->prev;
      toDelete = temp;
      delete toDelete;
    }
    temp = temp->next;
  }
}
```

# Advantages of Doubly Linked List

1. The doubly linked list is bi-directional, i.e. it can be traversed in both backward and forward direction.
2. The operations such as insertion, deletion and searching can be done from both ends.
3. Previous and Next  records of any element can be searched quickly

# Disadvantages

1. It consume more memory space.
2. More pointer adjustments during insertion and deletion of element.