K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

# Data Structure

ENCS205

*School of Engineering & Technology (SOET)*
*K.R. Mangalam University*

## UNIT-2
## Session 17: SINGLY LINKED LIST Operations I

# Recap

**Definition**: Linked list is a linear data structure with nodes containing data and pointers to the next node.

**Components**: Nodes, head pointer (points to first node), optional tail pointer (points to last node).

**Representation**: Nodes linked via pointers, dynamic memory allocation for nodes, head pointer for access.

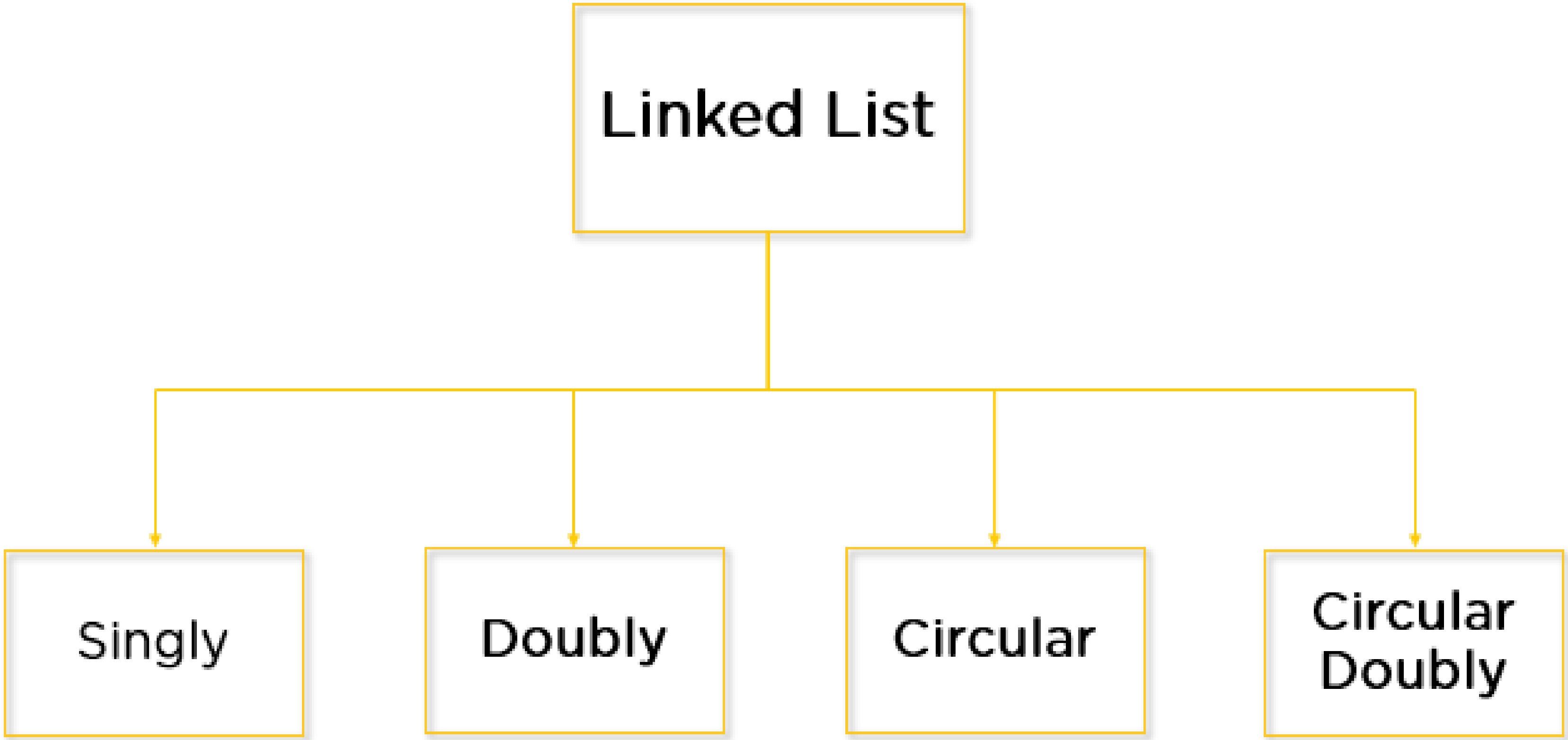
**Advantages**: Dynamic memory allocation, efficient insertion/deletion.

**Disadvantages**: Higher memory overhead, no direct access to elements, potentially slower access compared to arrays.

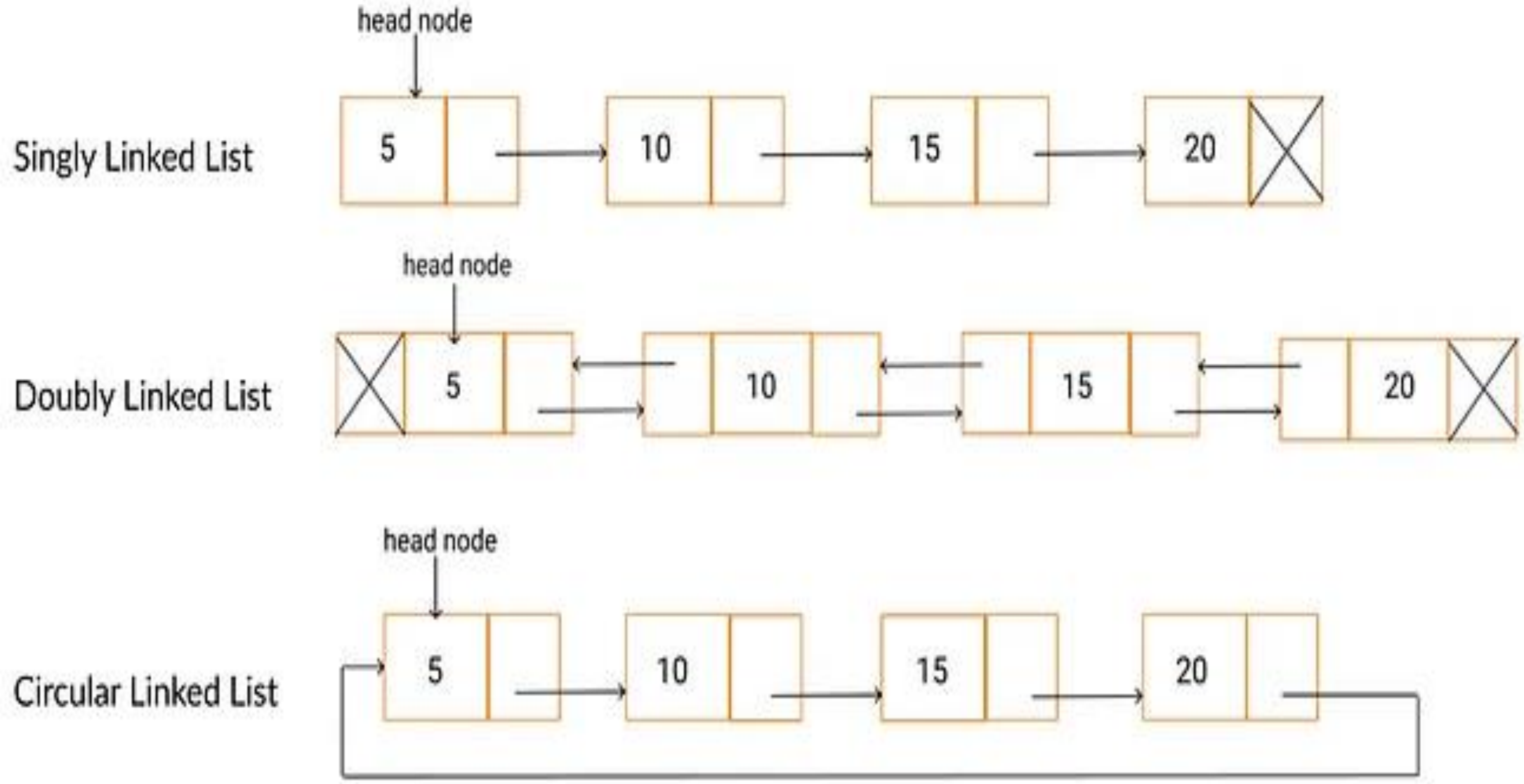
# Sessions 17 Outlook

- Types of linked list
- Singly linked list
- operations
- Inserting and Deleting a Node

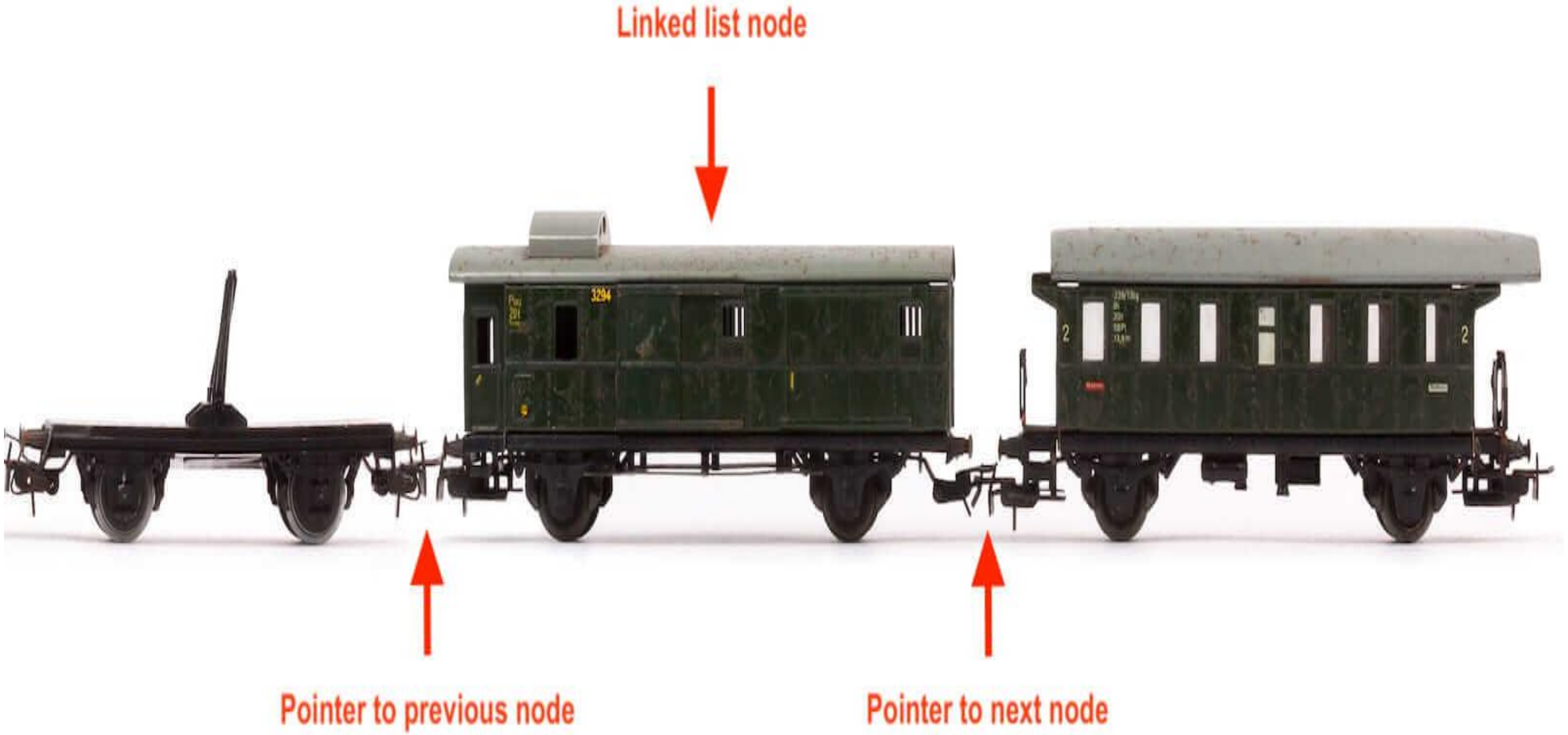
# Types of linked list



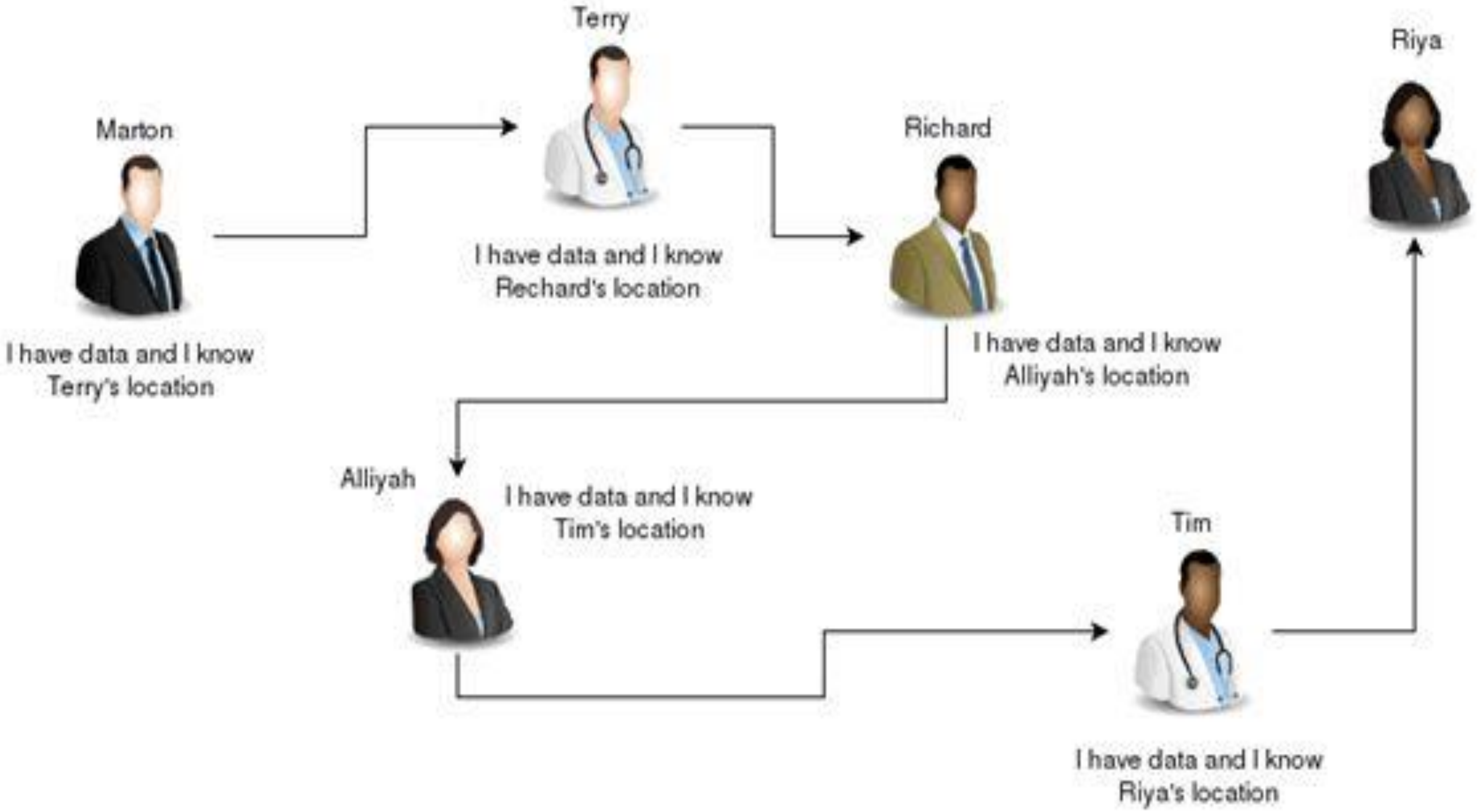
# Types of linked list



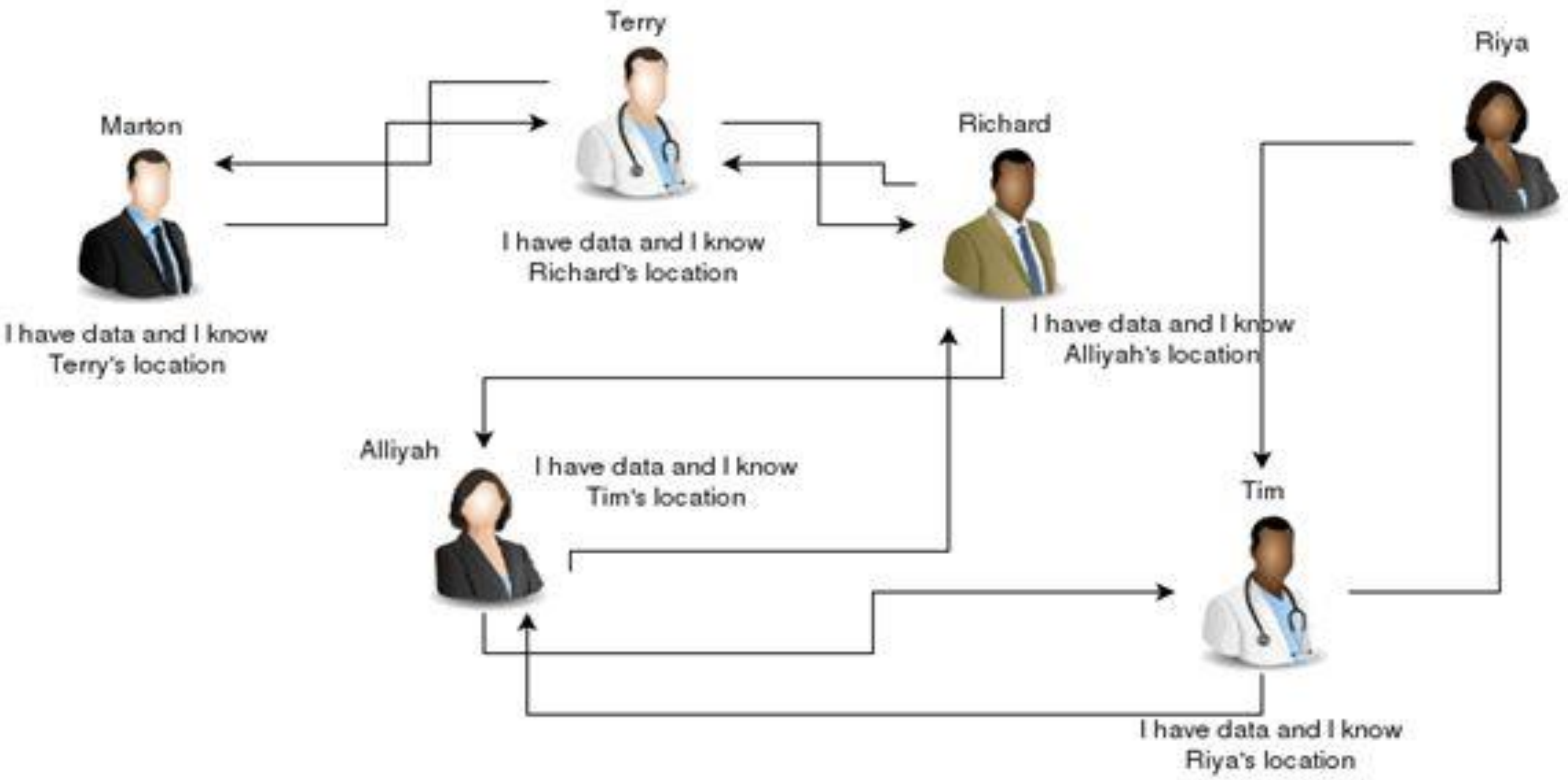
# Identify


Linked list node
Pointer to previous node
Pointer to next node

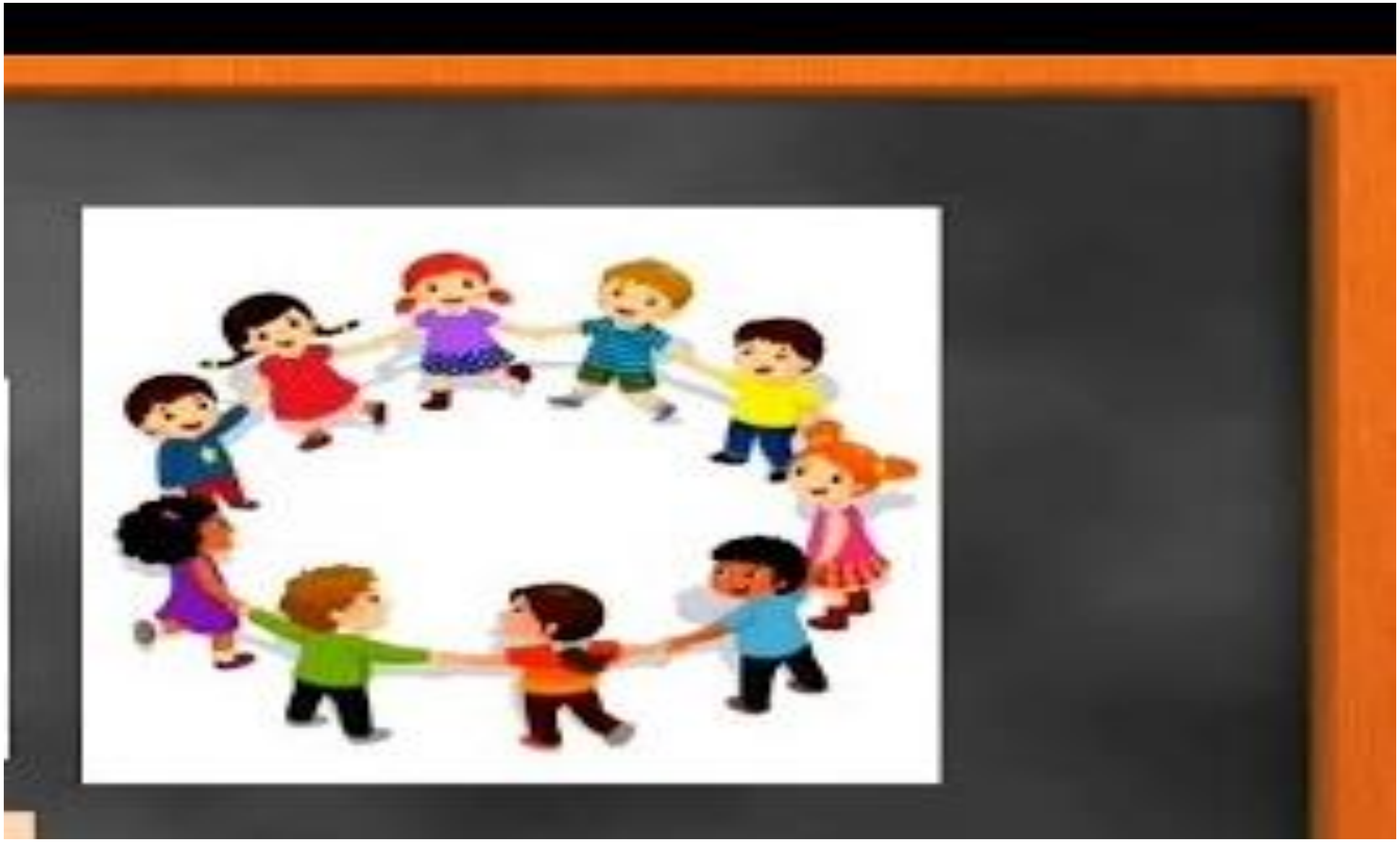
# Identify type of link list


Marton
I have data and I know Terry's location
Terry
I have data and I know Rechard's location
Richard
I have data and I know Alliyah's location
Riya
Alliyah
I have data and I know Tim's location
Tim
I have data and I know Riya's location

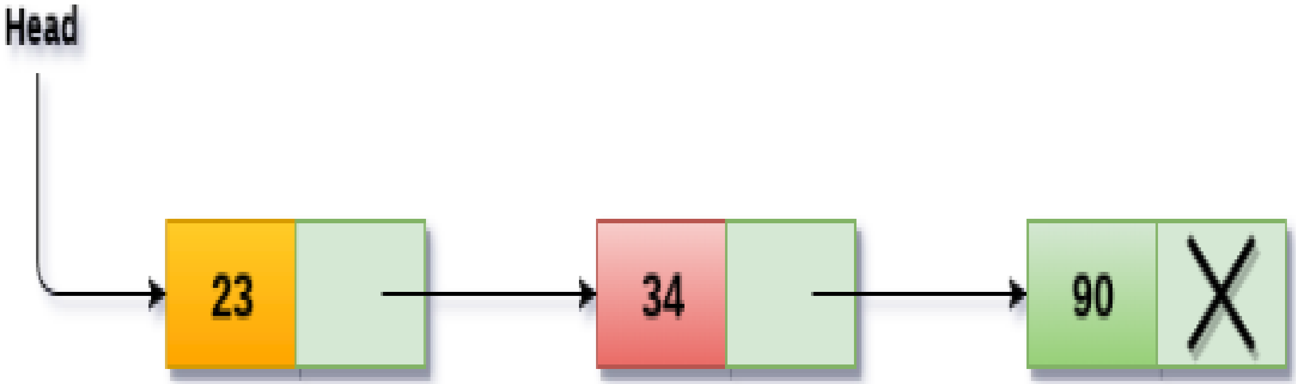
# Identify type of link list


Terry
Riya
Marton
Richard
I have data and I know Richard's location
I have data and I know Terry's location
I have data and I know Alliyah's location
Alliyah
I have data and I know Tim's location
Tim
I have data and I know Riya's location

# Identify type of link list

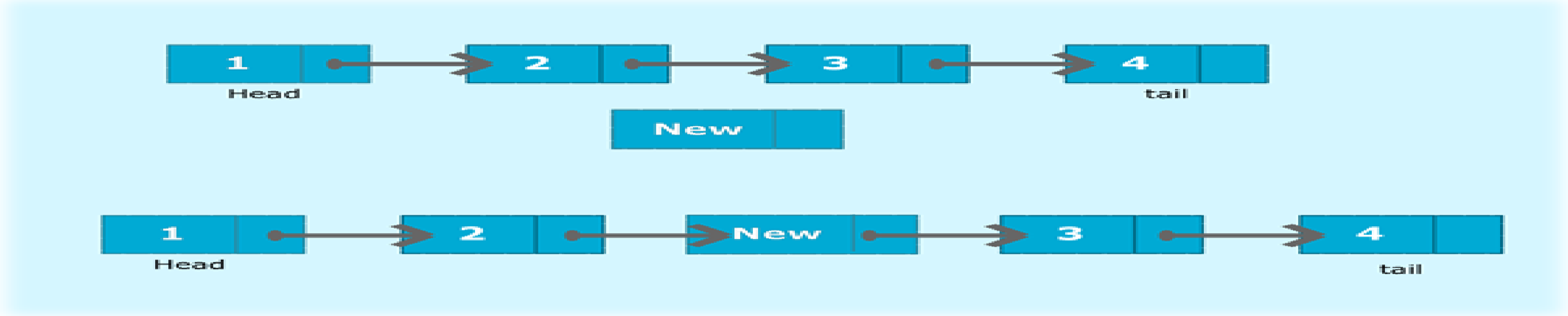


# Singly linked list

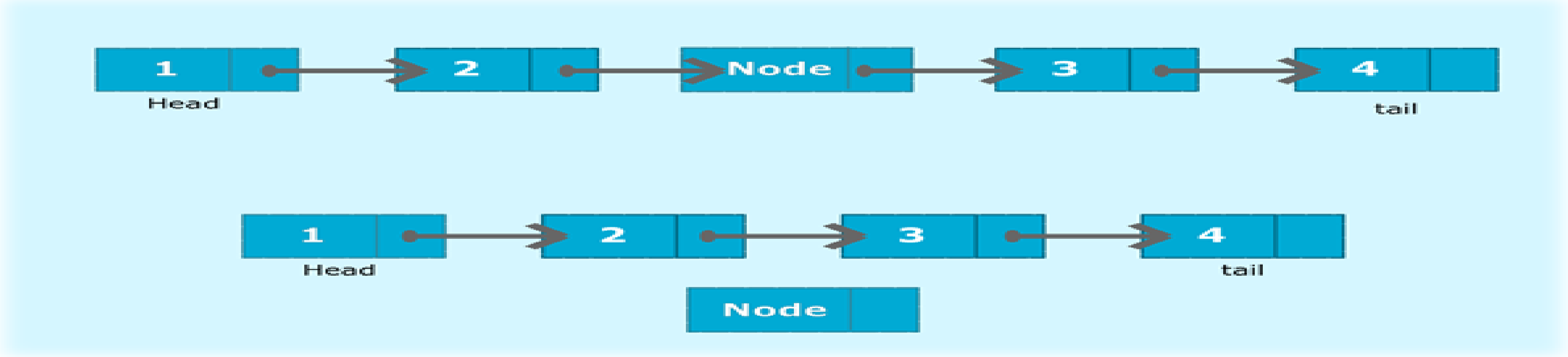


# Operations on Singly Linked List

## Insertion



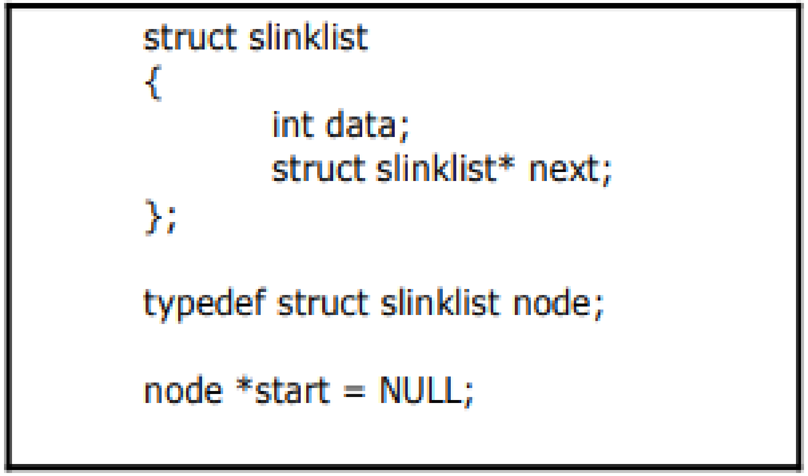
## Deletion



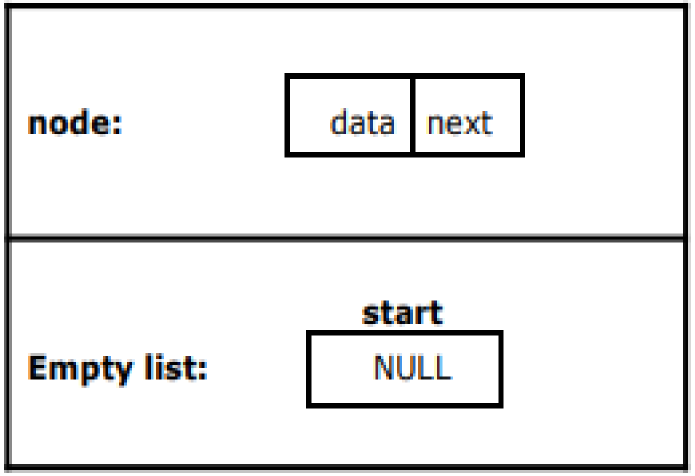
# Implementation of Single Linked List

we need to create a start node, used to create and access other nodes in the linked list. The following structure definition will do

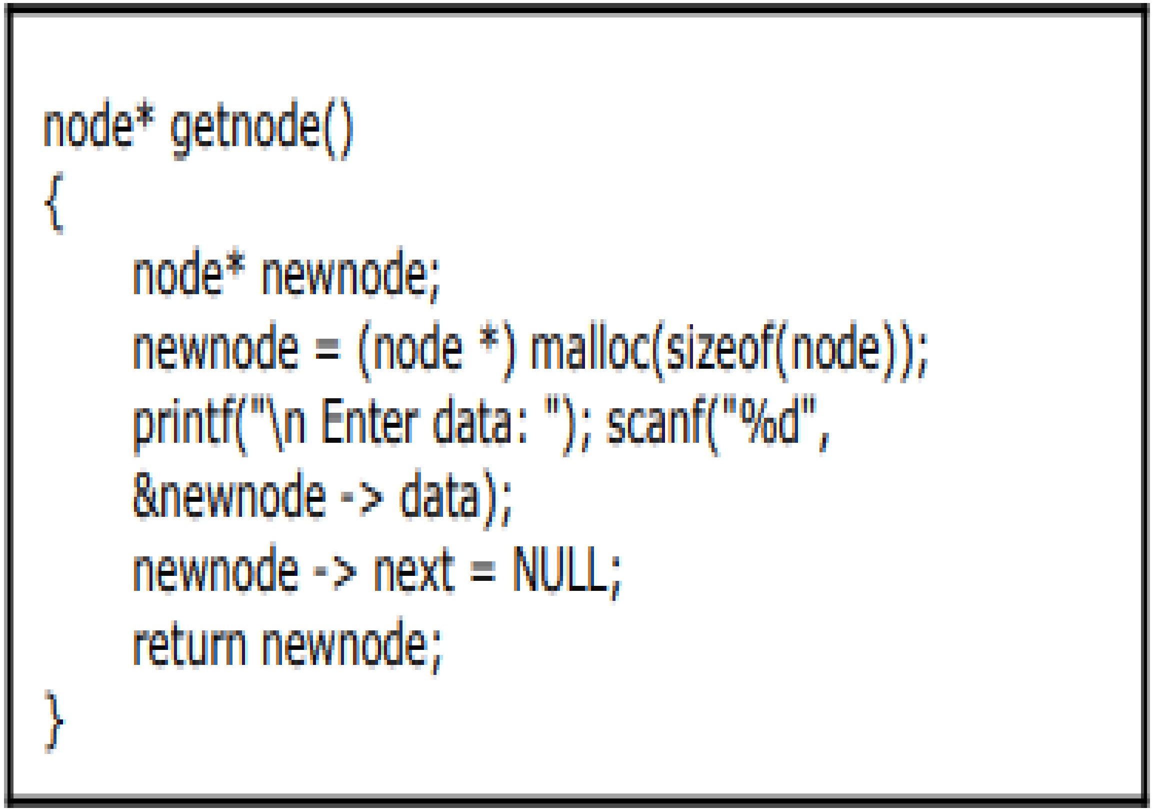
- Creating a structure with one data item and a next pointer, which will be pointing to next node of the list. This is called as self-referential structure.
- Initialise the start pointer to be NULL



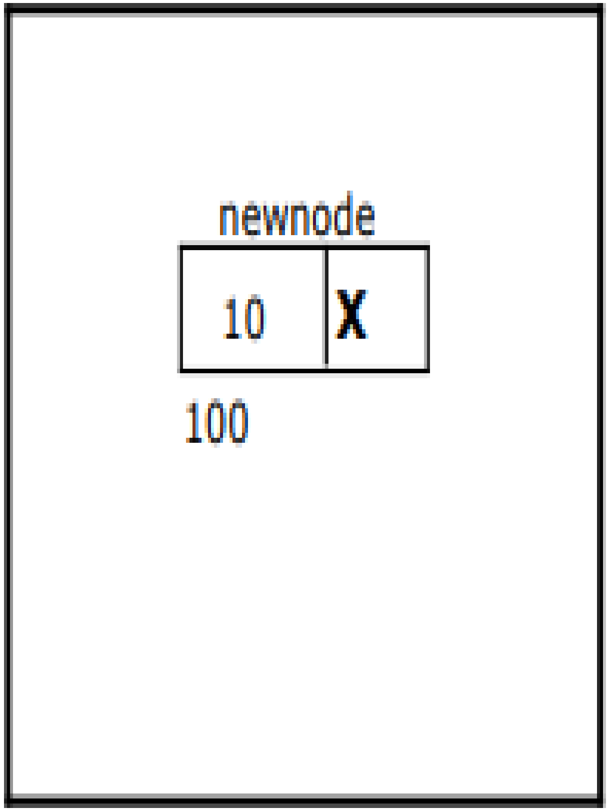
```
struct slinklist
{
        int data;
        struct slinklist* next;
};

typedef struct slinklist node;

node *start = NULL;
```



# Creating a node for Single Linked List



```
node* getnode()
{
        node* newnode;
        newnode = (node *) malloc(sizeof(node));
        printf("\n Enter data: "); scanf("%d",
        &newnode -> data);
        newnode -> next = NULL;
        return newnode;
}
```



# Singly Linked List

| Data Structure | Time Complexity | | | | | | | | Space Complexity |
|---|---|---|---|---|---|---|---|---|---|
| | Average | | | | Worst | | | | Worst |
| | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion | |
| Singly Linked List | θ(n) | θ(n) | θ(1) | θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |

# Representation

Node Linking: Nodes are linked sequentially through pointers. Each node's pointer points to the next node in the sequence, forming the linkage.

Memory Allocation: Typically, dynamic memory allocation is used for node creation, allowing nodes to be created and destroyed as needed.

Access: Access to elements in a singly linked list starts from the head pointer. To access or manipulate data in a specific node, traversal through the list from the head pointer is necessary.

# Advantages

**Efficient Insertion and Deletion**: Adding or removing elements at the beginning or middle of the list is efficient, requiring only adjustments to pointers.

**Dynamic Size**: Singly linked lists can grow or shrink dynamically, as nodes can be added or removed without requiring contiguous memory allocation.

**Flexibility**: Singly linked lists have no fixed size limitations, offering flexibility in managing varying amounts of data

# Disadvantages

**No Direct Access**: Access to specific elements in the list requires traversal from the head pointer, making direct access inefficient.

**Extra Memory Overhead**: Each node in the list requires additional memory for storing pointers, leading to higher memory overhead compared to arrays.

**Inefficient Reverse Traversal**: Singly linked lists are inefficient for operations requiring access to nodes in reverse order, as they don't have pointers to previous nodes.

# Pseudo-code for insertion

```
typedef struct nd {
   struct item data;
   struct nd * next;
   } node;

void insert(node *curr)
{
node * tmp;

tmp=(node *) malloc(sizeof(node));
tmp->next=curr->next;
curr->next=tmp;
}
```

# Pseudo-code for deletion

.

```
typedef struct nd {
   struct item data;
   struct nd * next;
   } node;

void delete(node *curr)
{
node * tmp;
 tmp=curr->next;
curr->next=tmp->next;
free(tmp);
}
```

# In essence ...

For insertion:

- A record is created holding the new item.
- The next pointer of the new record is set to link it to the item which is to follow it in the list.
- The next pointer of the item which is to precede it must be modified to point to the new item.

For deletion:

- The next pointer of the item immediately preceding the one to be deleted is altered, and made to point to the item following the deleted item.

# Insertion and deletion in liked list

```
/ Function to insert a new node.
void Linkedlist::insertNode(int data)
{
    // Create the new Node.
    Node* newNode = new Node(data);

    // Assign to head
    if (head == NULL) {
        head = newNode;
        return;
    }
```

Complete code here

# Test Your self

**1. As we have the memory address of Nodes can we traverse backwards in a linked list ?**

a: Yes, we can.

b: No, we can’t.

**2. What are the advantages of a linked list?**

a: Dynamic Memory Allocation

b: They require less memory than an array to store the same data.

c: We can easily traverse back to previous elements.

d: None of the above.

**3. What is the time complexity for insertion of an element into linked list?**

a: O(1)

b: O(log n)

c: O(n)

d: O(n^2)

# Quiz Answers

1. **Answer: b**
2. **Answer :a**
3. **Answer : c**

https://questions.examside.com/past-years/gate/question/pconsider-the-problem-of-reversing-a-singly-linked-list-t-gate-cse-theory-of-computation-finite-automata-and-regular-language-zpfpkf4re1g8xuff

# Review

**Definition:**

Singly linked list: linear structure with nodes containing data and pointers.

**Components:**

Nodes: data and next pointer.

Head Pointer: points to first node.

Tail Pointer: optional, points to last node.

**Representation:**

Nodes linked via pointers, dynamic memory allocation.

Access via head pointer, traversal for manipulation.

# Review

**Advantages:**

Efficient insertion/deletion.

Dynamic size, flexibility.

**Disadvantages:**

No direct access, traversal needed.

Higher memory overhead.

Inefficient reverse traversal.

THANK
YOU