Slot 05 - Linked List

Presenter:

Dr. LE Thanh Tung

Content

1. Linked List
2. Linked List Operations
3. Other Types of Linked List

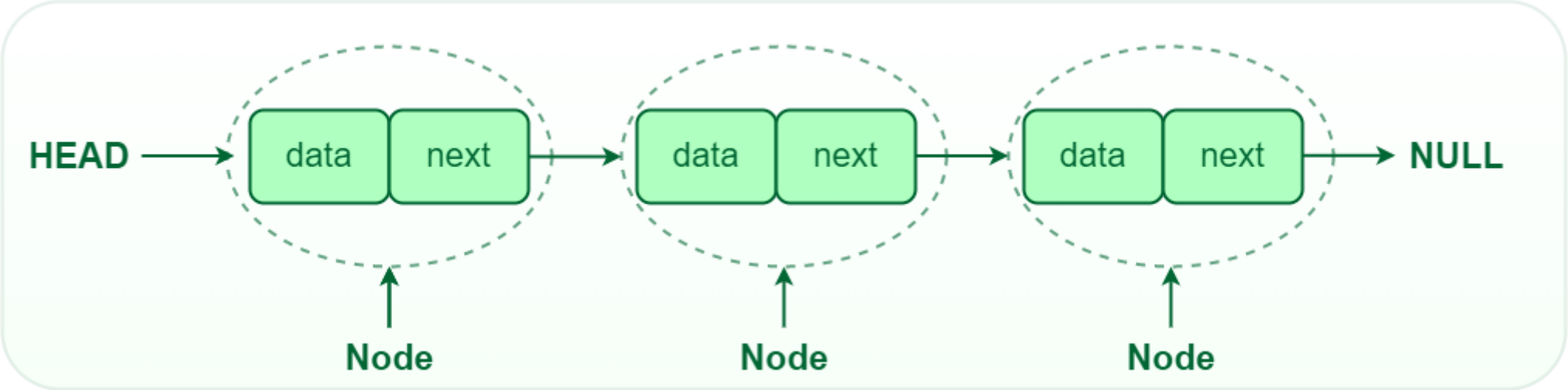
Linked List

- A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the **data** and the **address** of the next node

- Unlike Arrays, Linked List elements are not stored at a contiguous location

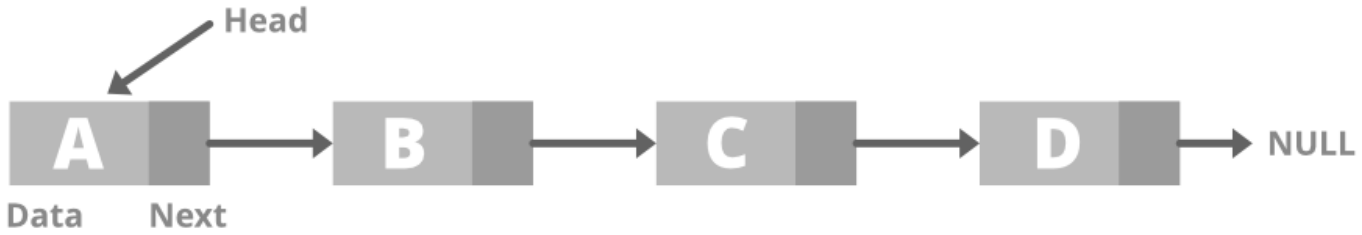
Advantages of Linked List

- Dynamic Data structure: based on the operation insertion or deletion
- Ease of Insertion/Deletion
- Efficient Memory Utilization: avoids the wastage of memory
- Implementation: Various advanced data structures can be implemented using a linked list like a stack, queue, graph, hash maps, etc

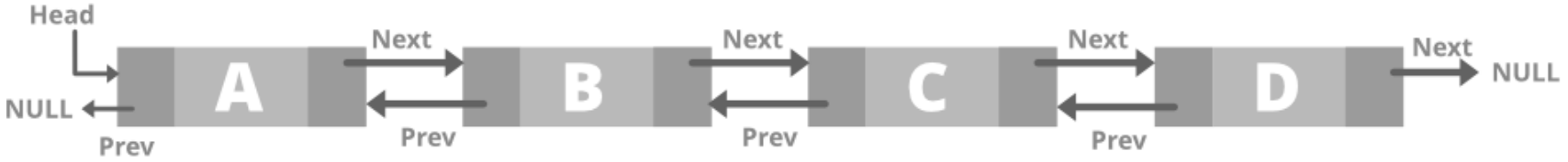


Types of Linked List

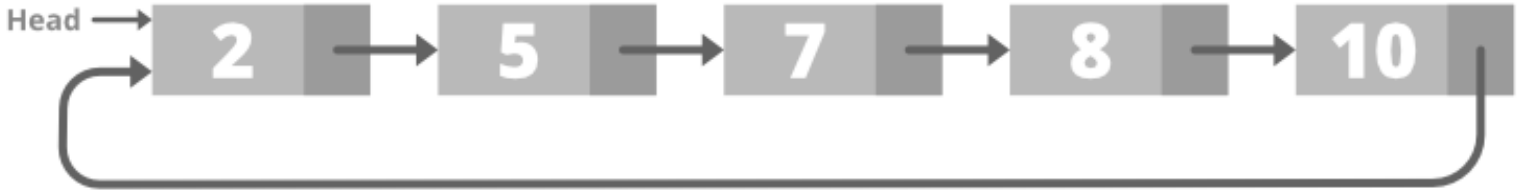
- Singly Linked List:



- Doubly Linked List:



- Circular Linked List



Node in Linked List

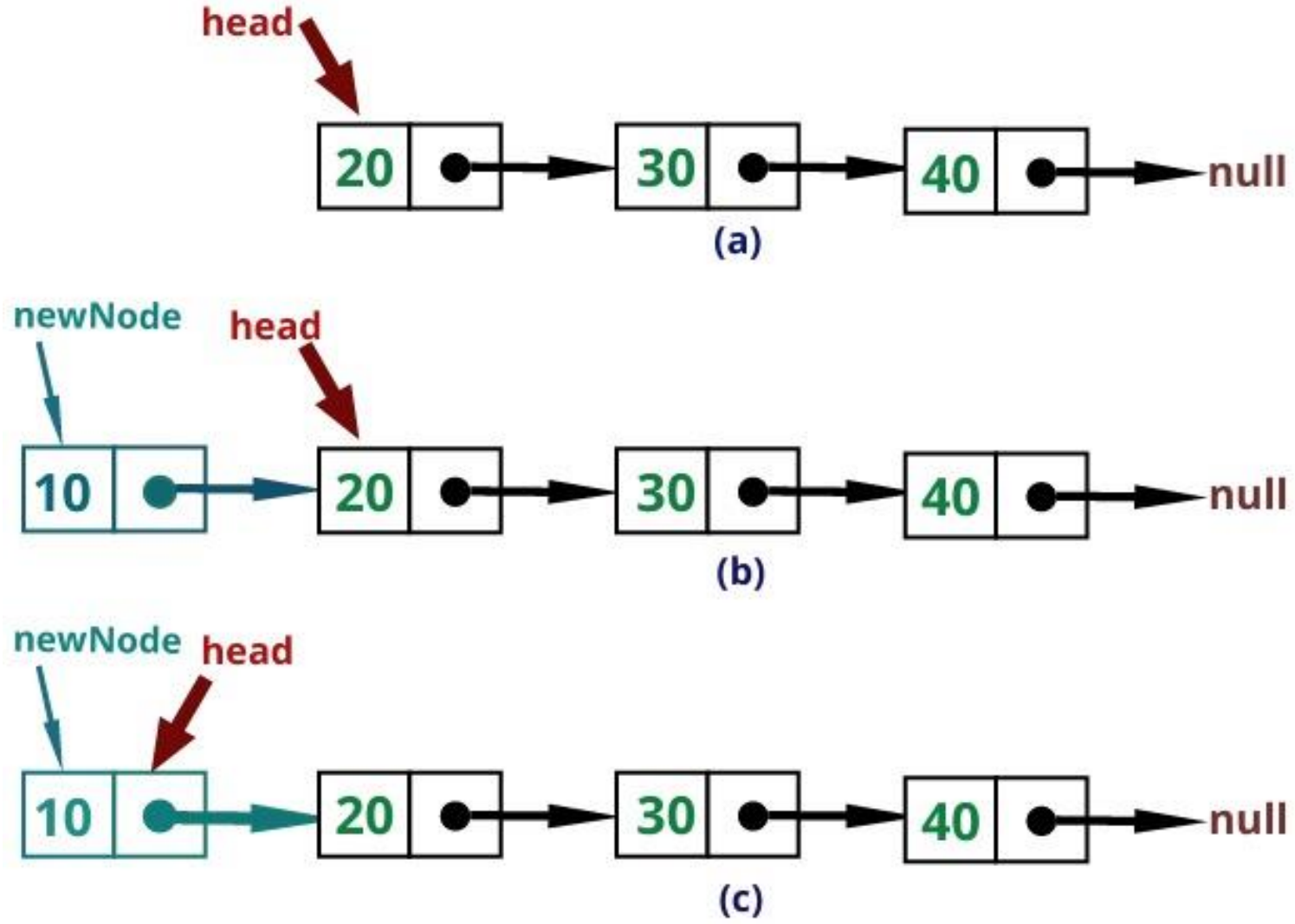
- A node in a linked list contains one or more members that represent data
- Each node also contains (at least) a link to another node

```
int main(){
    Node n, c;
    c.key = 2;
    n.key = 1;
    n.next = &c;
    cout << n.key << "->" << n.next->key;
    return 0;
}
```

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

Add Node to Head

- Assume that we will control a linked list by only pointer **head**



Add Node to Head

- Assume that we will control a linked list by only pointer **head**

```
void addHead(Node* &pHead, int val){
    // step 1: create new node
    Node* newNode = createNode(val);
    // step 2: add into head
    if (pHead == NULL) pHead = newNode;
    else{
        newNode->next = pHead;
        pHead = newNode;
    }
}
```

Traverse a Linked List

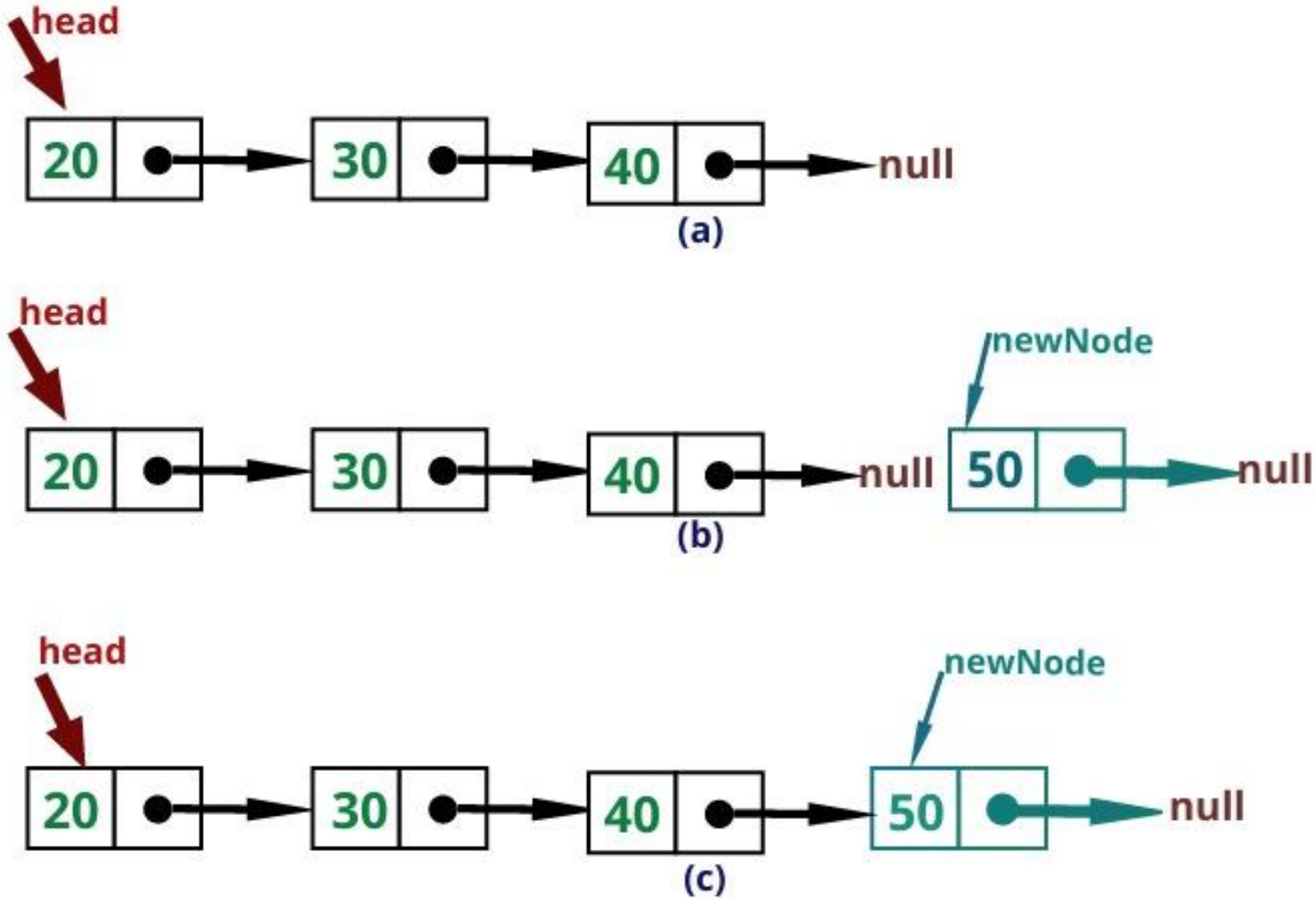
- Print all value of data in a linked list with pointer head

```
void printLL(Node* pHead){
    Node* curNode = pHead;
    while (curNode != NULL){
        cout << curNode->key << " ";
        curNode = curNode->next;
    }
}
```

- **Note: Must not change the value of pHead**

Add Node to Tail

- Assume that we will control a linked list by only pointer **pHead**

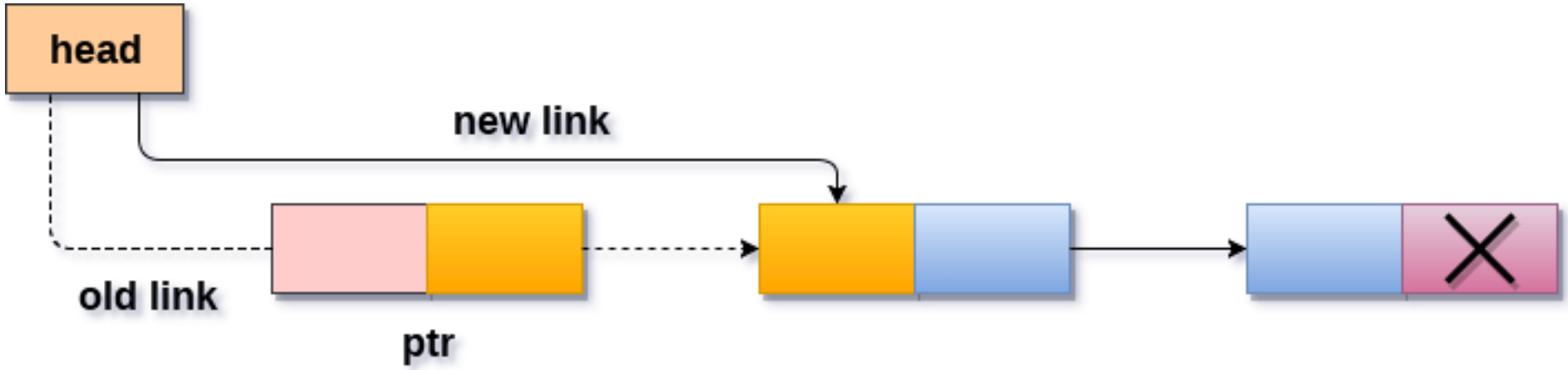


Add Node to Tail

- Assume that we will control a linked list by only pointer **phead**

```
void addTail(Node* &pHead, int val){
    // step 1: create new node
    Node* newNode = createNode(val);
    // step 2: find the node at tail
    if (pHead == NULL) pHead = newNode;
    else{
        Node* curNode = pHead;
        while (curNode->next != NULL){
            curNode = curNode->next;
        }
        // step: add newNode to tail
        curNode->next = newNode;
    }
}
```

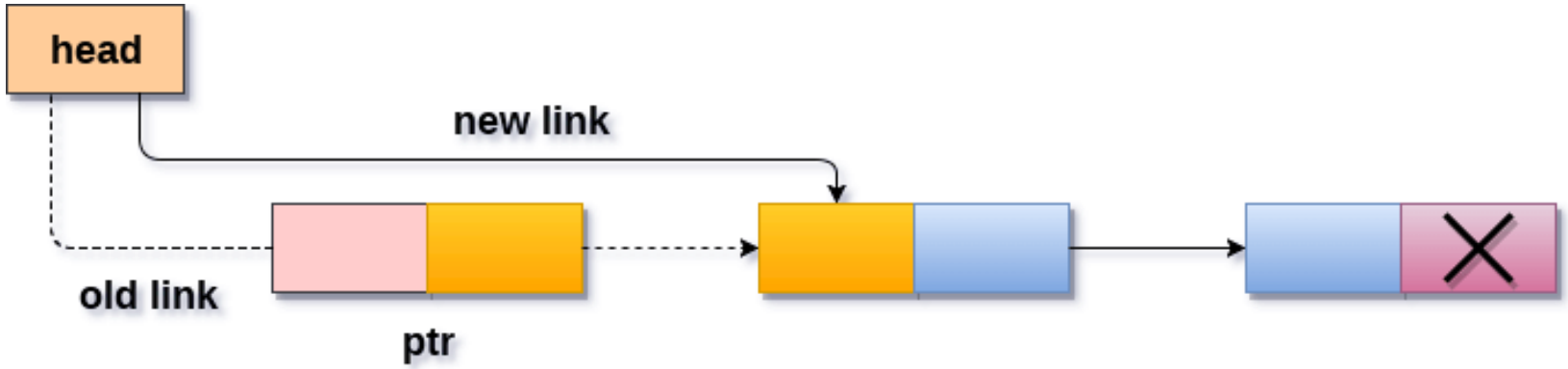
Remove First Node




```
ptr = head
head = ptr -> next
free(ptr)
```

Deleting a node from the beginning

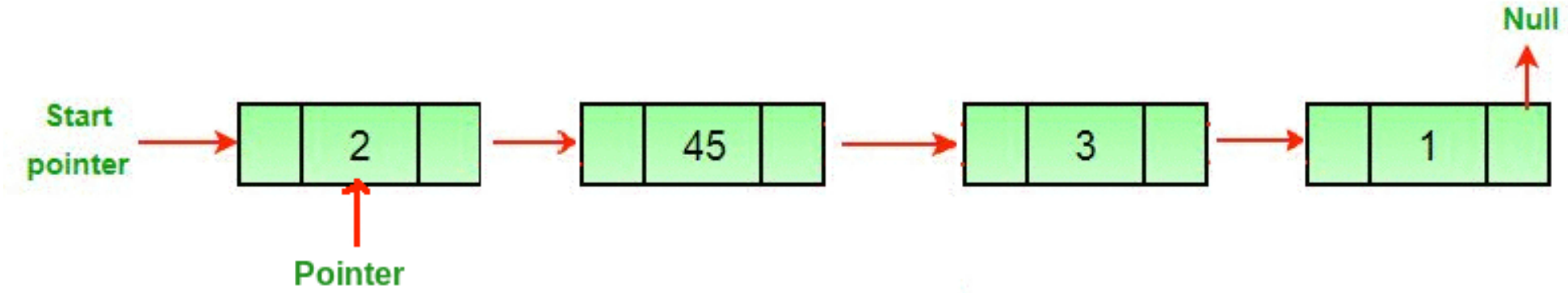
Remove First Node



ptr = head
head = ptr -> next
free(ptr)

```
void removeHead(Node* &pHead){
    Node* tmp = pHead;
    pHead = pHead->next;
    delete tmp;
}
```

Remove Last Node



Remove Last Node

```
void removeLast(Node*& head)
{
    if (head == NULL)
        return;

    if (head->next == NULL) {
        delete head;
        head = NULL;
    }
```

```
// Find the second last node
Node* curNode = head;
while (curNode->next->next != NULL)
    curNode = curNode->next;

// Delete last node
delete (curNode->next);

// Change next of second last
curNode->next = NULL;
```

Remove All Nodes

```
void clearLL(Node* &pHead){
    Node* curNode;
    while (pHead != NULL){
        curNode = pHead->next;
        delete pHead;
        pHead = curNode;
    }
}
```

Remove Node by Value

- Remove an integer before a value of a given linked list

```
void removeBefore(Node* &pHead, int val)
```

- Remove an integer after a value of a given linked list

```
void romveAfter(Node* &pHead, int val)
```

Insert Node with Position

Example:

- Input: insert 10 into position 2

$$3 \rightarrow 5 \rightarrow 8 \rightarrow 4$$

- Out:

$$3 \rightarrow 5 \rightarrow 10 \rightarrow 8 \rightarrow 4$$

$$0 \quad 1 \quad 2 \quad 3 \quad 4$$

Solution:

- Traverse the Linked list up to position -1 nodes.
- Once all the position -1 nodes are traversed, create new Node
- Point the next pointer of the new node to the next of current node.
- Point the next pointer of current node to the new node.

Insert Node with Position

```
bool addPos(Node* &pHead, int data, int pos){
    if (pos < 1) return false;
    Node* curNode = pHead;
    while (pos > 1){
        if (curNode->next == NULL) return false;
        pos = pos - 1;
        curNode = curNode->next;
    }
    Node* newNode = createNode(data);
    newNode->next = curNode->next;
    curNode->next = newNode;
    return true;
}
```

Exercise

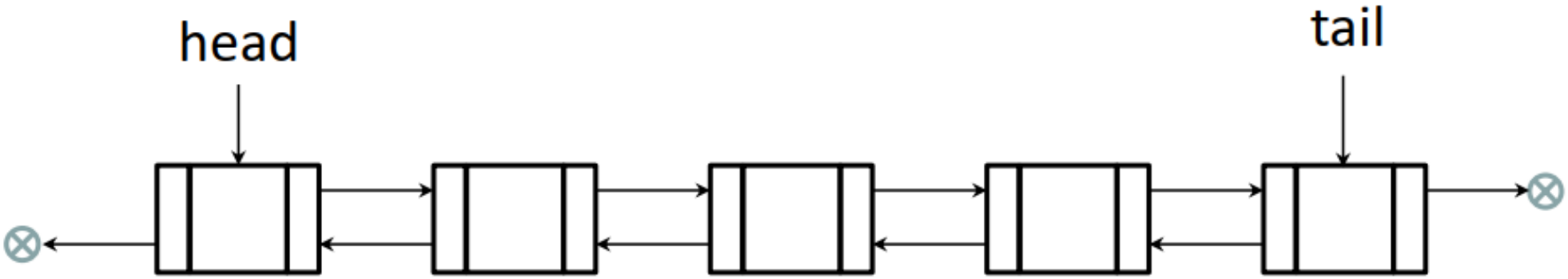
- Count the prime number in a linked list
- Write a function to get the size of linked list

Linked List with Tail

- Rewrite the function:
  - Remove Last Node
  - Add Node to Last
- Advantages of Tail in Linked List:
  - Easy to control
  - Less complexity in add and remove Last node

Doubly Linked List

- Each node points to the previous and the next node in the list




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

```
Node* createNode(int keyVal){
    Node* n = new Node;
    n->key = keyVal;
    n->next = NULL;
    n->prev = NULL;
    return n;
}
```

Doubly Linked List

- Implement the following functions for the doubly linked list
  - inserting at the beginning of the linked list
  - inserting at the end of the linked list
  - inserting after another node in a linked list
  - inserting before another node in a linked list
  - removing at the beginning of a linked list
  - removing a node X from the linked list

THANK YOU
for YOUR ATTENTION