VTC Academy

Think Ahead, Beyond Boundaries

DATA STRUCTURES AND ALGORITHMS

LESSON 2

List Abstract Data Types and Linked Lists

CONTENT

1. Introduction to List Abstract Data Type
2. Generic Operation on the List
3. Introduction to Linked List
4. Implement Basic Operation of Linked List
    - Insert
    - Display
    - Delete
5. Summary

List Abstract Data Type (List ADT)

- List ADT is a dynamic ordered tuple of homogenous elements
  - A0, A1, A2, … , AN-1
  - where Ai is the i-th element of list
- The position of Ai is i ; positions range from 0 to N-1 inclusive
- The size of list is N (a list with no elements is called an “empty list”)

Generic Operations on a List

insert(e, position)	//insert e into the list at the specified position
remove(e)	//remove e from the list if present
find(e)	//return position of the first occurrence of e
findKth(int k)	//return the element in the specified position
isEmpty()	//return true if the list has no elements

Array Implementation of a List ADT

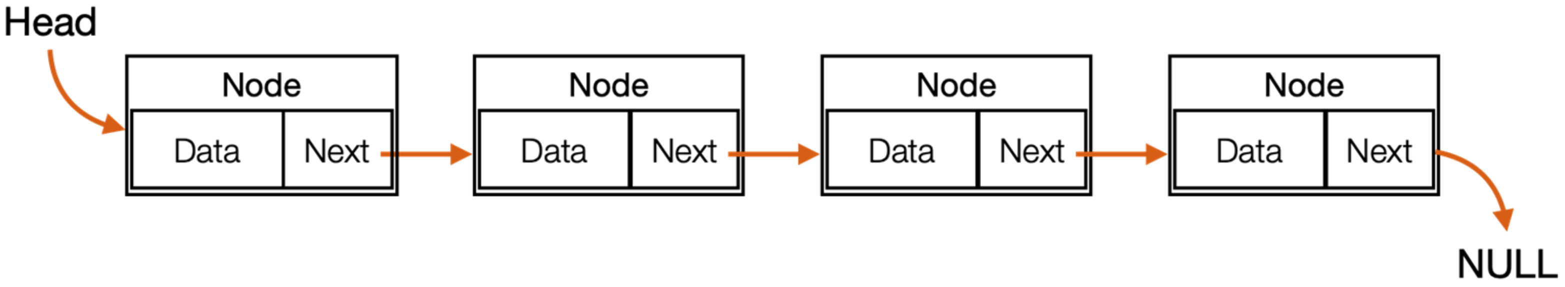
- Use an array to store the element of the list.
- Also, array have a fixed capacity, but can fix with implementation.
- You must implement the actual code of above methods:
    - insert(e, position) //insert e into the list at the specified position
    - remove(e) //remove e from the list if present
    - find(e) //return position of the first occurrence of e
    - findKth(int k) //return the element in the specified position
    - isEmpty() //return true if the list has no elements

Linked List

- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array
- The important terms to understand the concept of Linked List:
  - Link – Each link of a linked list can store a data called an element
  - Next – Each link of a linked list contains a link to the next link called Next
  - LinkedList – A Linked List contains the connection link to the first link called First

Linked List Representation

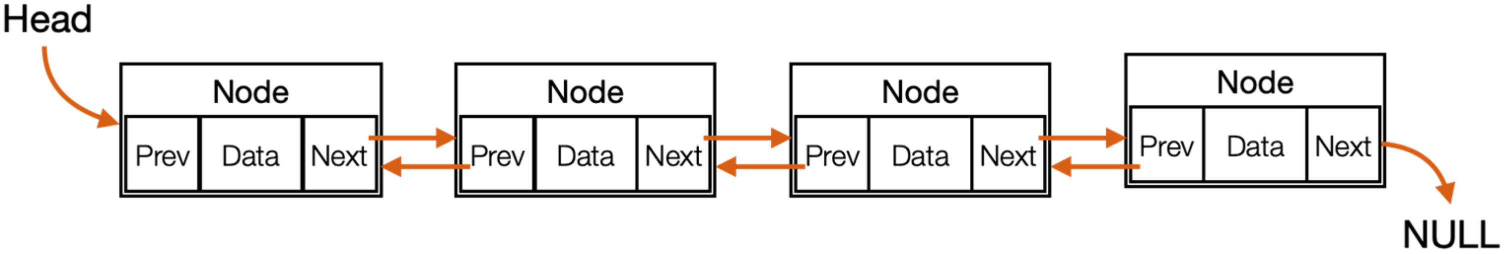
- Linked list can be visualized as a chain of nodes, where every node points to the next node



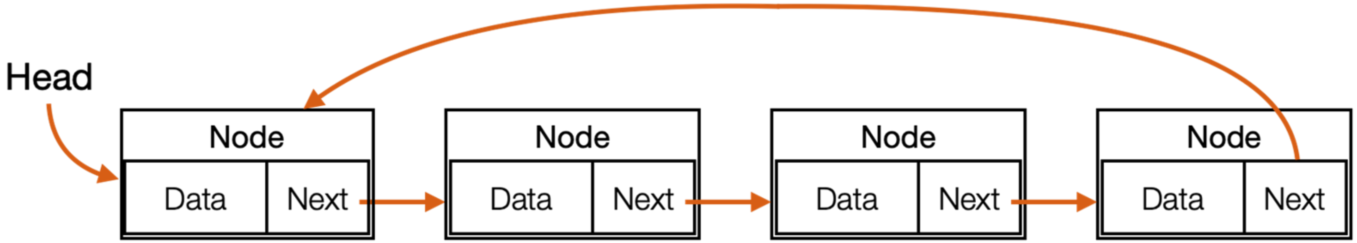
- As per the above illustration, following are the important points to be considered.
  - Linked List contains a link element called first
  - Each link carries a data field(s) and a link field called next
  - Each link is linked with its next link using its next link
  - Last link carries a link as null to mark the end of the list

Types of Linked List

- Simple Linked List – Item navigation is forward only (See above figure).
- Doubly Linked List – Items can be navigated forward and backward.



- Circular Linked List – Last item contains link of the first element as next and the first element has a link to the last element as previous.



Basic Operations

- Insertion – Adds an element at the beginning of the list
- Display – Displays the complete list
- Deletion – Deletes an element at the beginning of the list
- Delete – Deletes an element using the given key
- Search – Searches an element using the given key

Linked List Implementation

- Singly Linked List

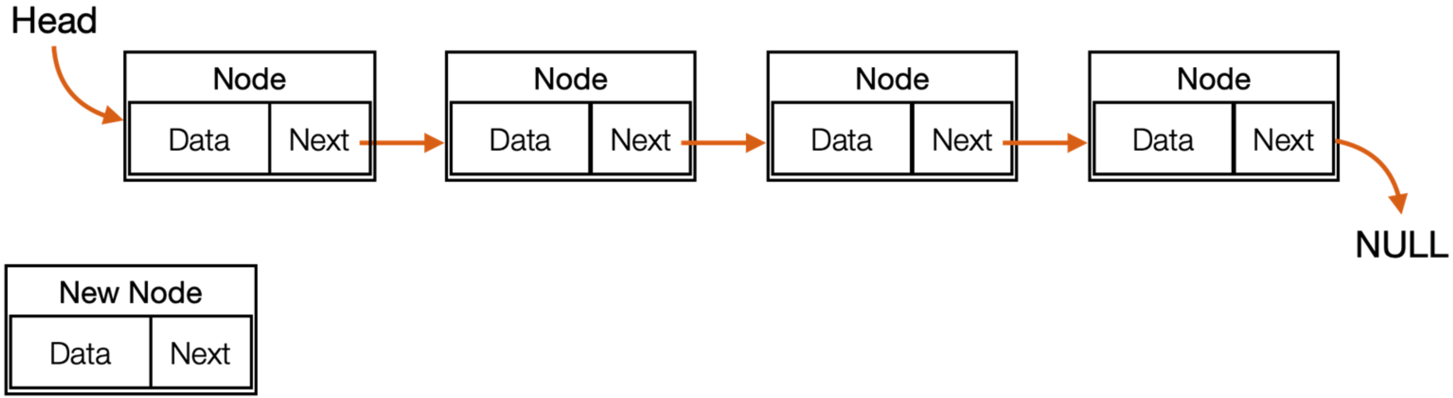
```
struct node {
        int data;
        struct node *next;
}
```

- Doubly Linked List

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

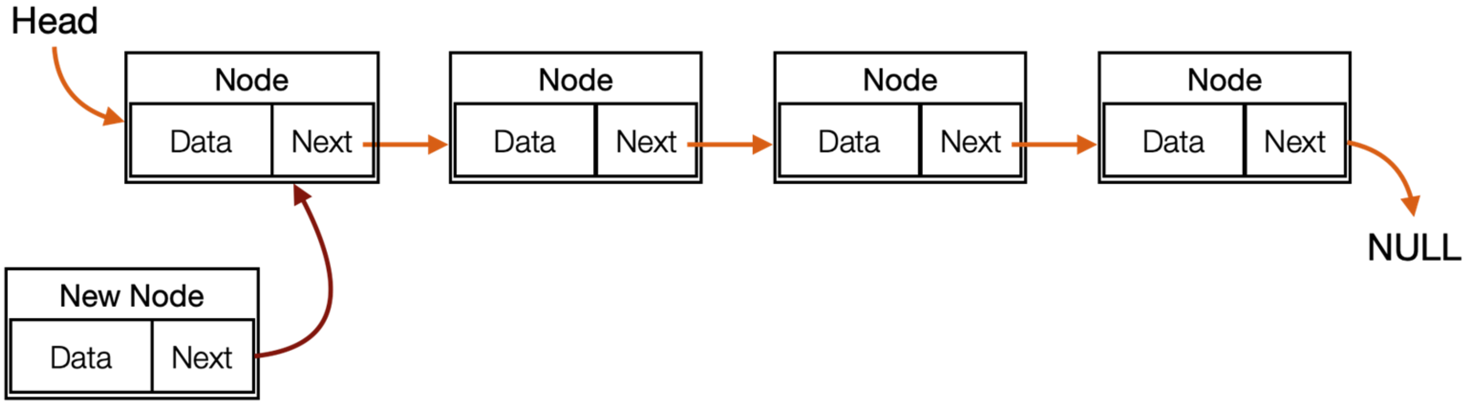
Insert - Step 1

- Adds an element at the beginning of the list



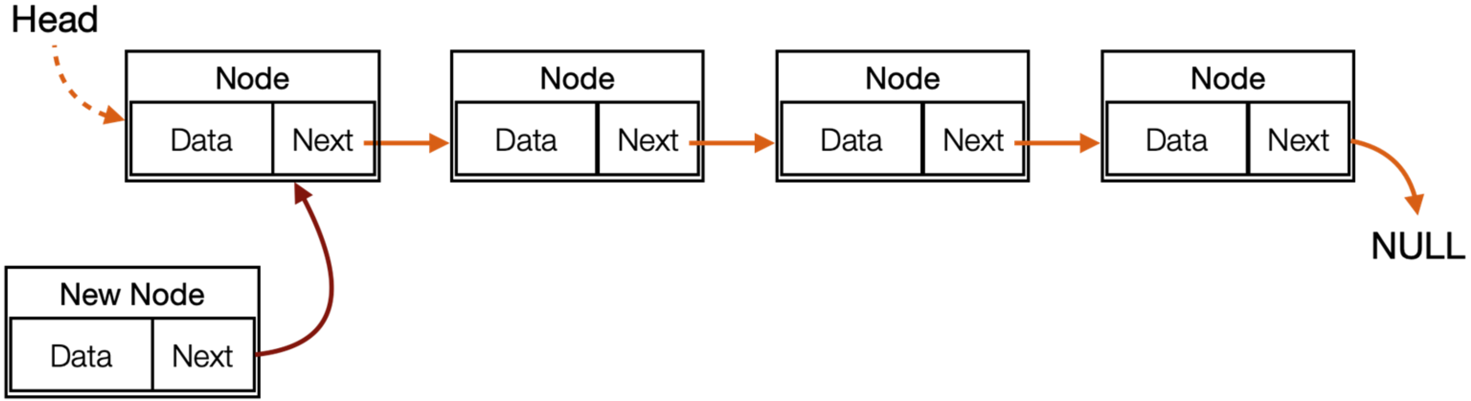
Insert – Step 2

- Adds an element at the beginning of the list



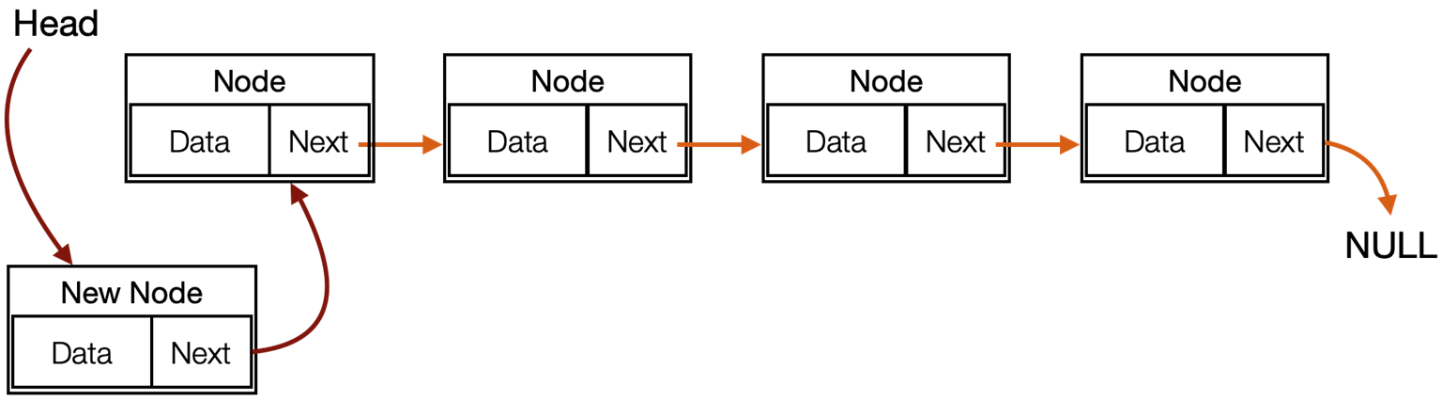
Insert – Step 3

- Adds an element at the beginning of the list

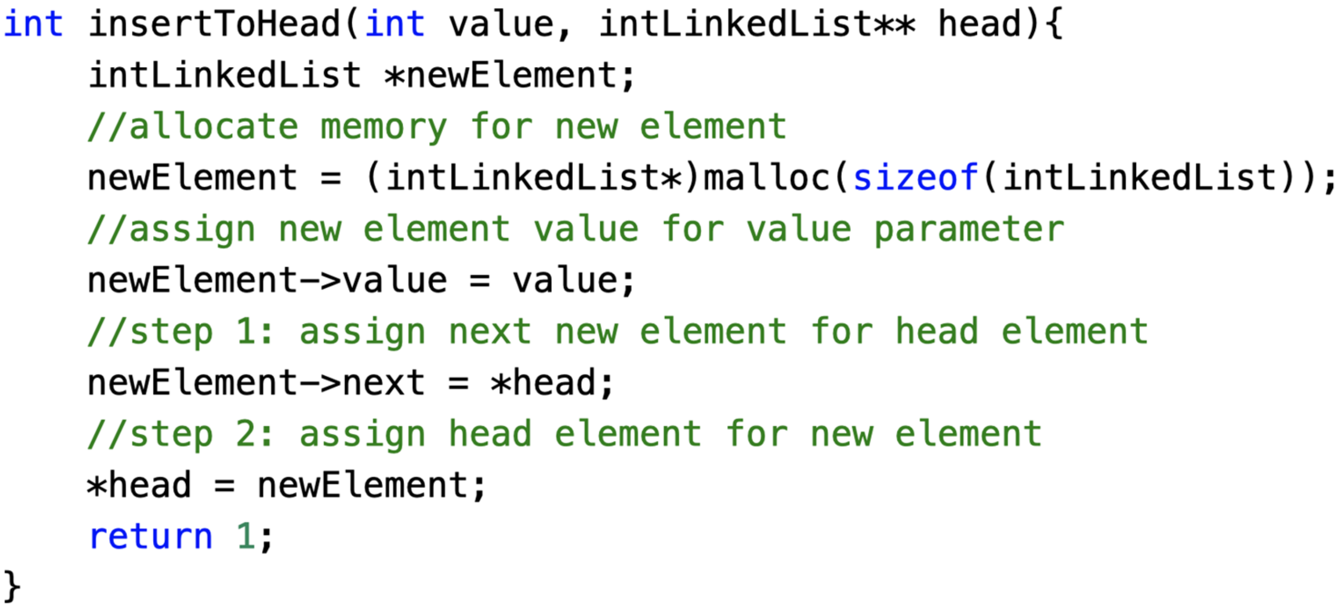


Insert – Step 4

- Adds an element at the beginning of the list

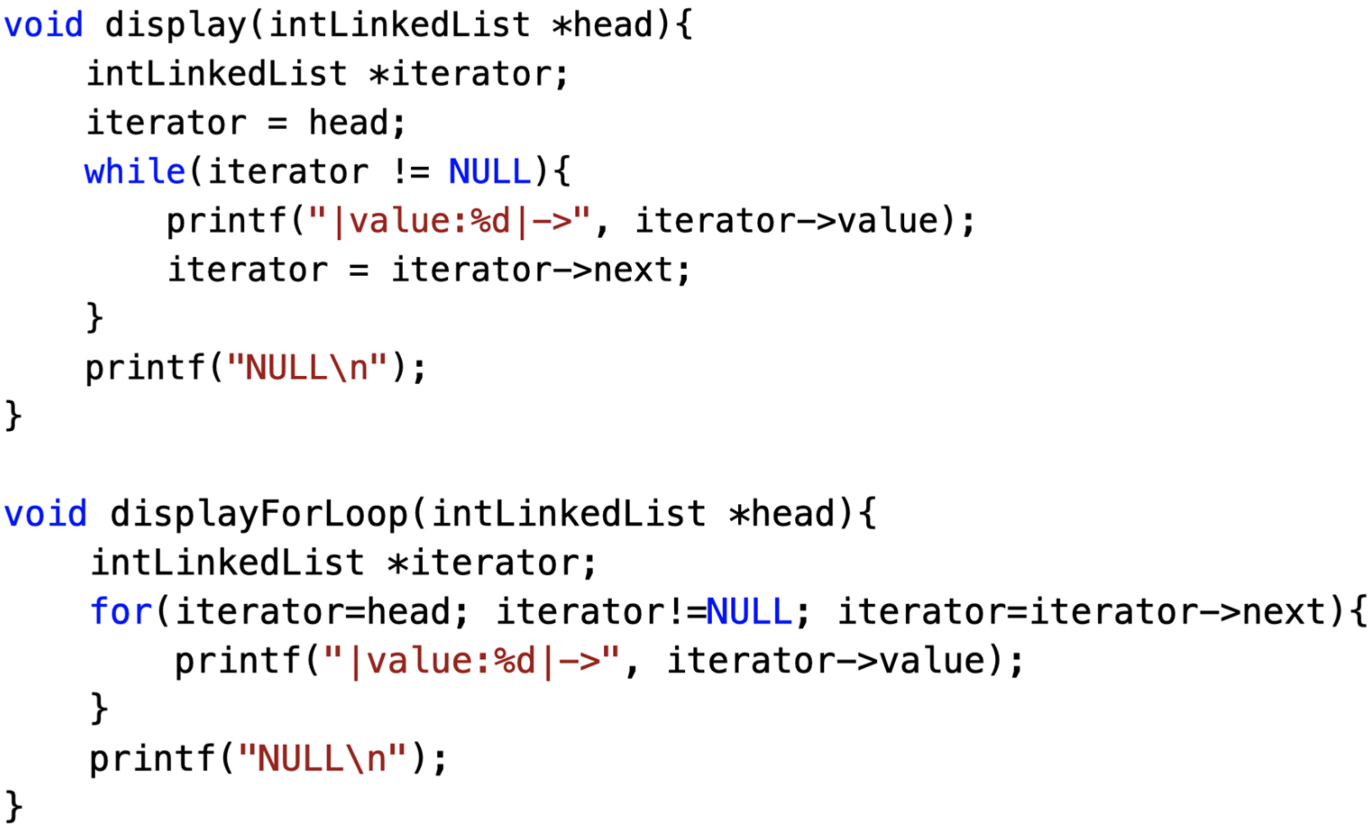


Insert Example



```c
int insertToHead(int value, intLinkedList** head){
    intLinkedList *newElement;
    //allocate memory for new element
    newElement = (intLinkedList*)malloc(sizeof(intLinkedList));
    //assign new element value for value parameter
    newElement->value = value;
    //step 1: assign next new element for head element
    newElement->next = *head;
    //step 2: assign head element for new element
    *head = newElement;
    return 1;
}
```

Display Linked List

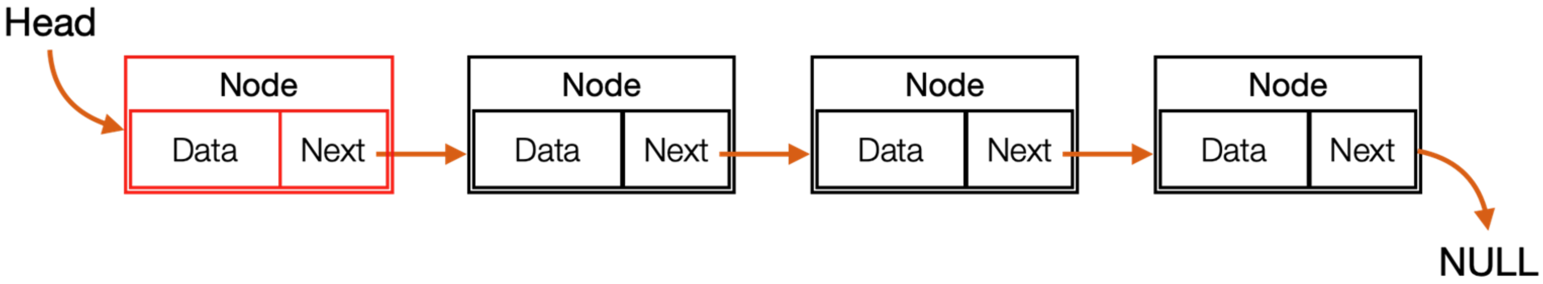


```c
void display(intLinkedList *head){
    intLinkedList *iterator;
    iterator = head;
    while(iterator != NULL){
        printf("|value:%d|->", iterator->value);
        iterator = iterator->next;
    }
    printf("NULL\n");
}

void displayForLoop(intLinkedList *head){
    intLinkedList *iterator;
    for(iterator=head; iterator!=NULL; iterator=iterator->next){
        printf("|value:%d|->", iterator->value);
    }
    printf("NULL\n");
}
```

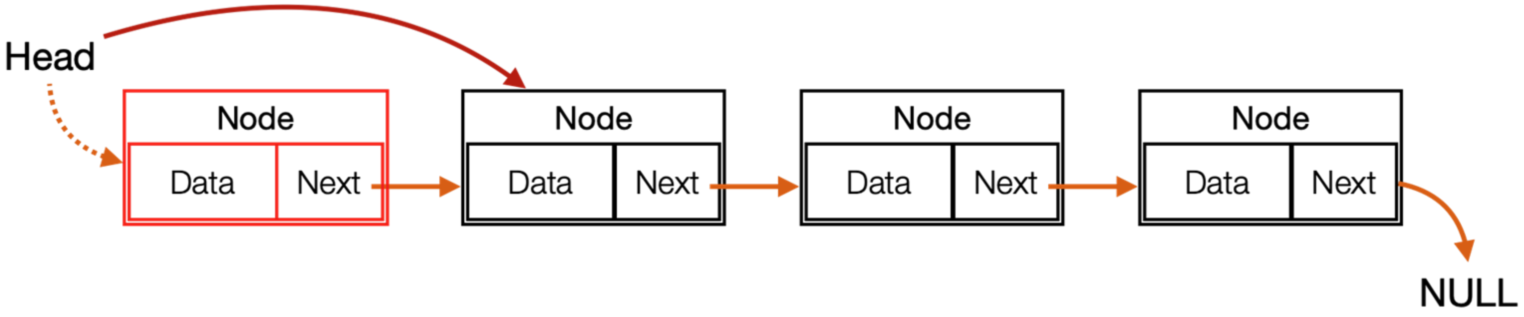
Delete First Element – Step 1

- Delete an element at the beginning of the list



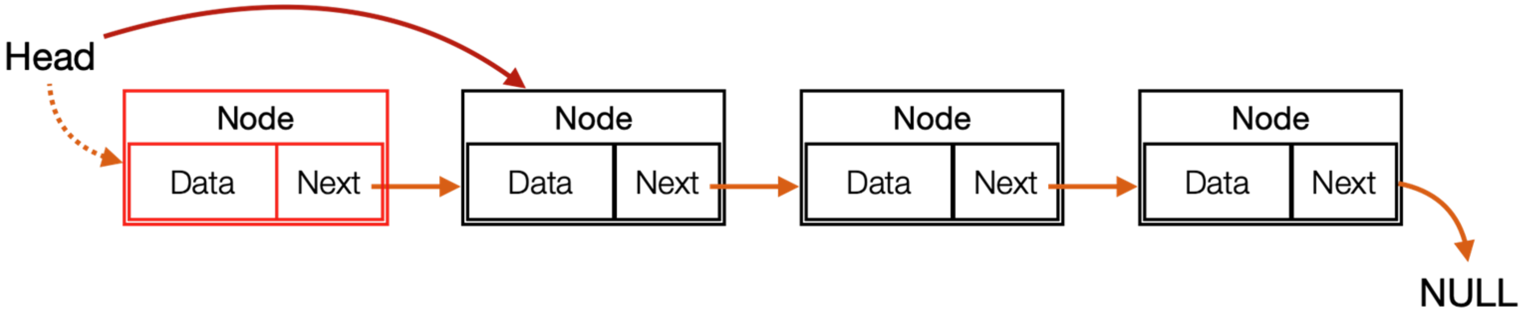
Delete First Element – Step 2

- Delete an element at the beginning of the list



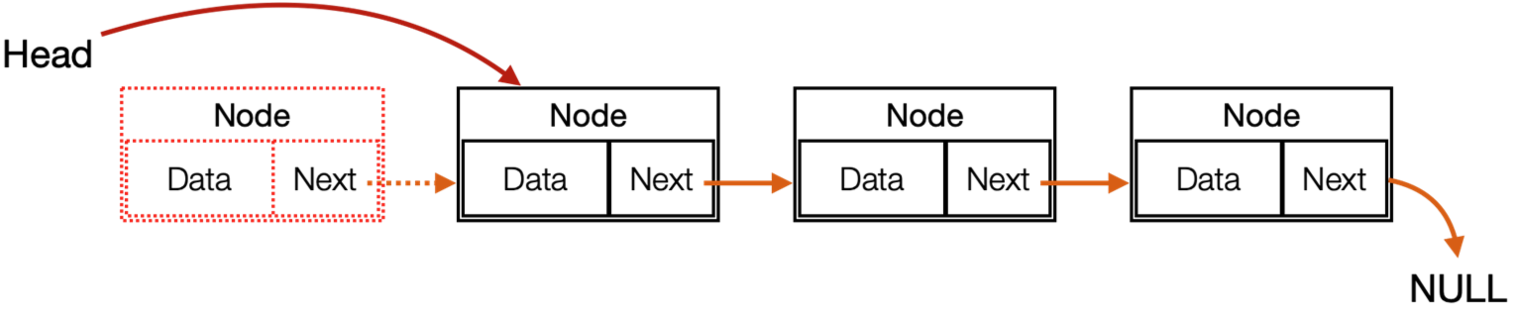
Delete First Element – Step 3

- Delete an element at the beginning of the list

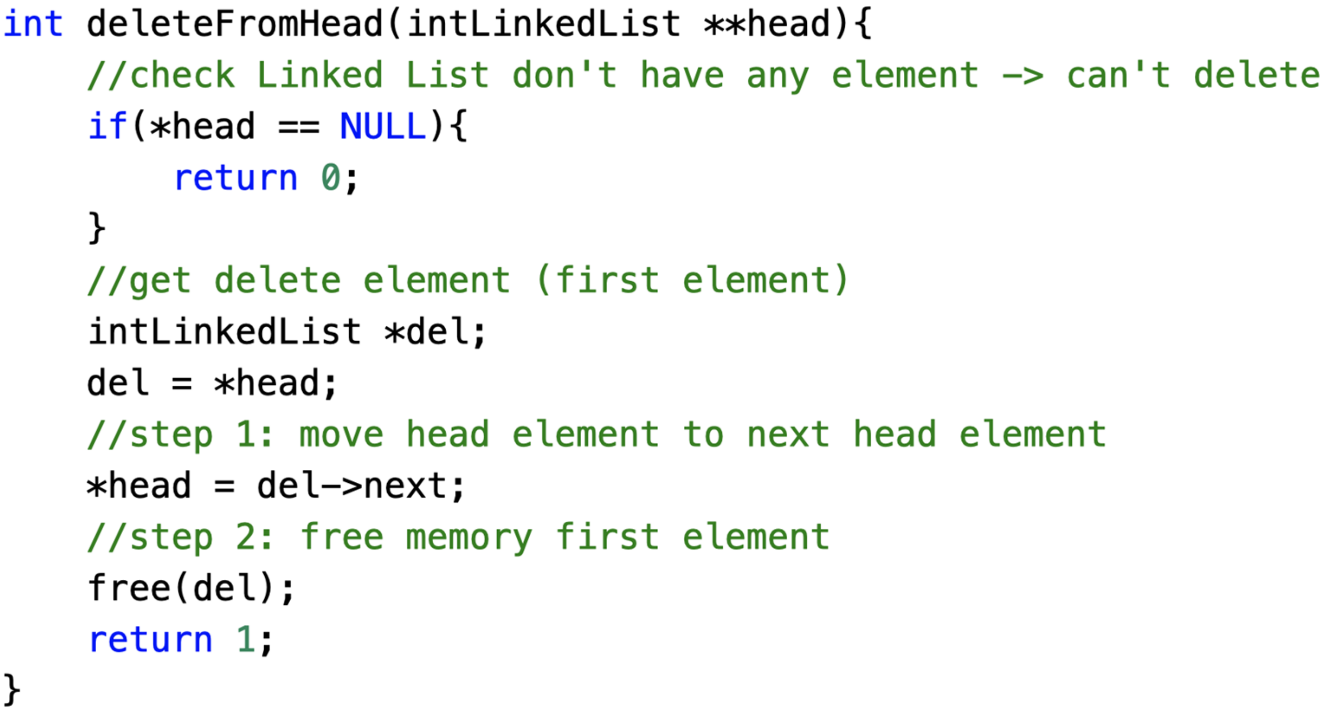


Delete First Element – Step 4

- Delete an element at the beginning of the list



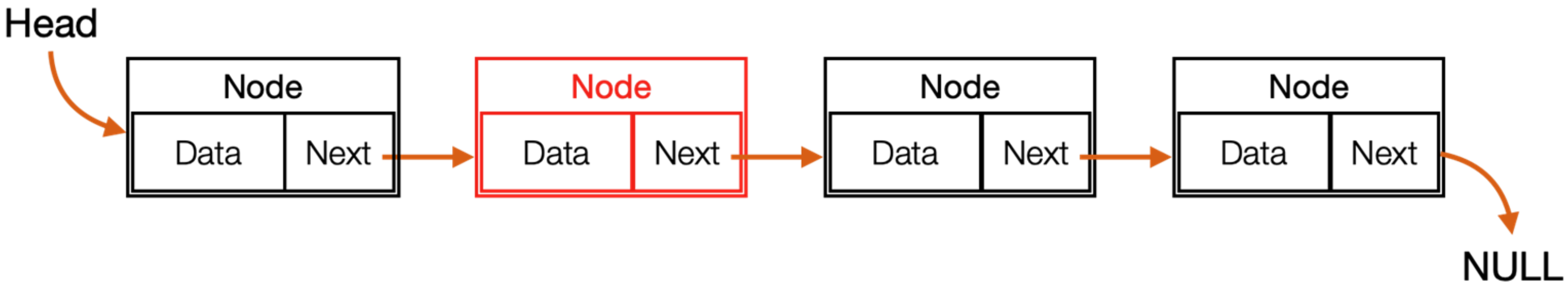
Delete First Element Example



```
int deleteFromHead(intLinkedList **head){
    //check Linked List don't have any element -> can't delete
    if(*head == NULL){
        return 0;
    }
    //get delete element (first element)
    intLinkedList *del;
    del = *head;
    //step 1: move head element to next head element
    *head = del->next;
    //step 2: free memory first element
    free(del);
    return 1;
}
```

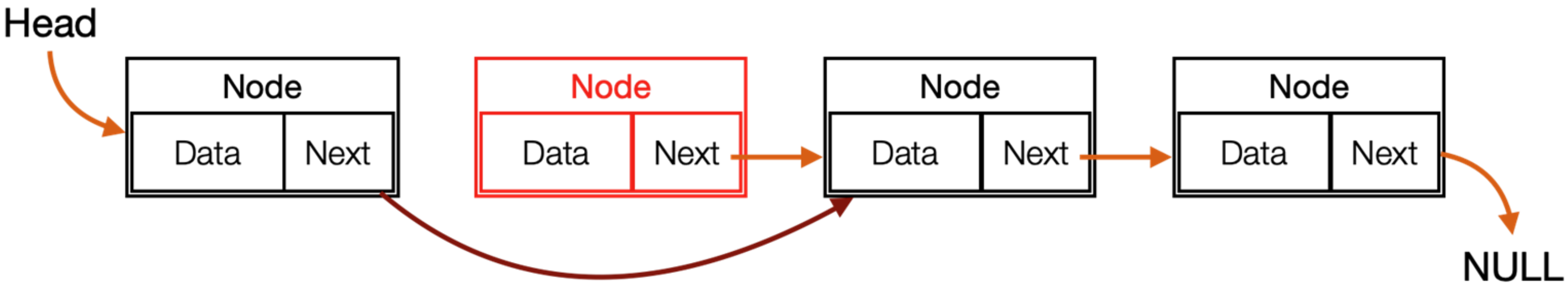
Delete Specific Element – Step 1

- Delete an element at the specific position.
- Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



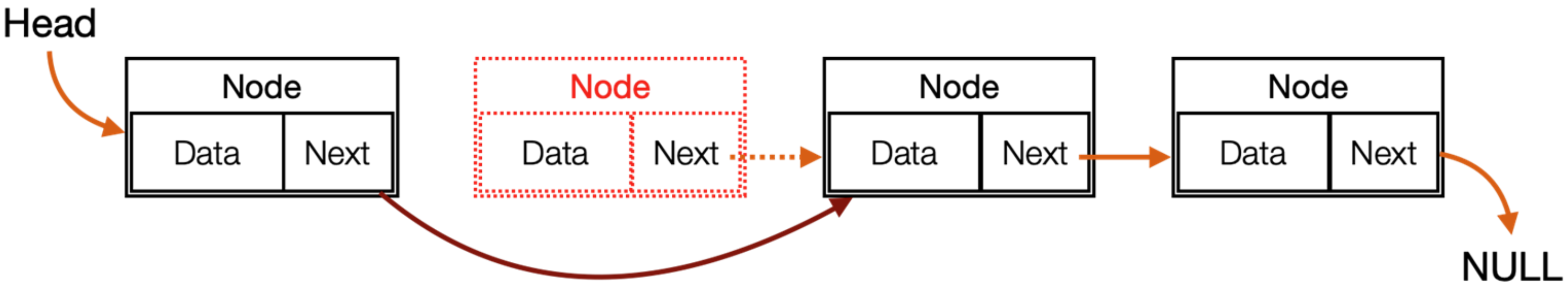
Delete Specific Element – Step 2

- The left (previous) node of the target node now should point to the next node of the target node –



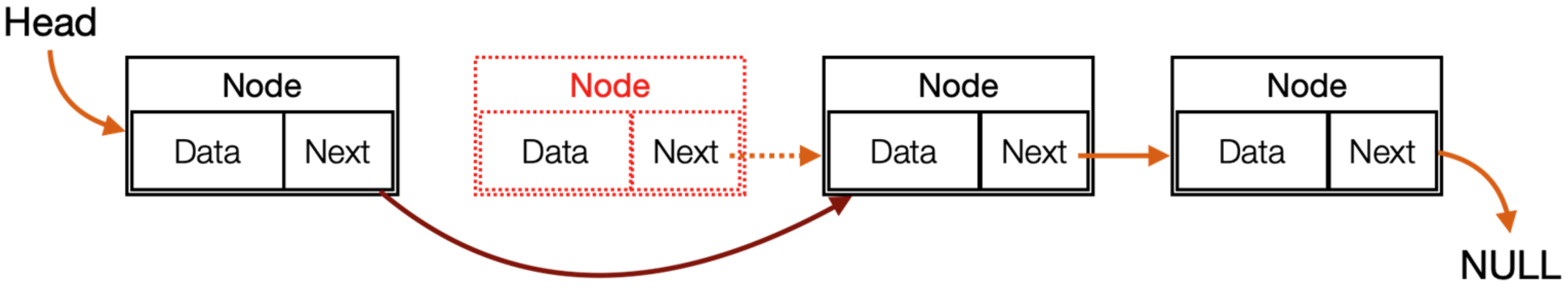
Delete Specific Element – Step 3

- This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

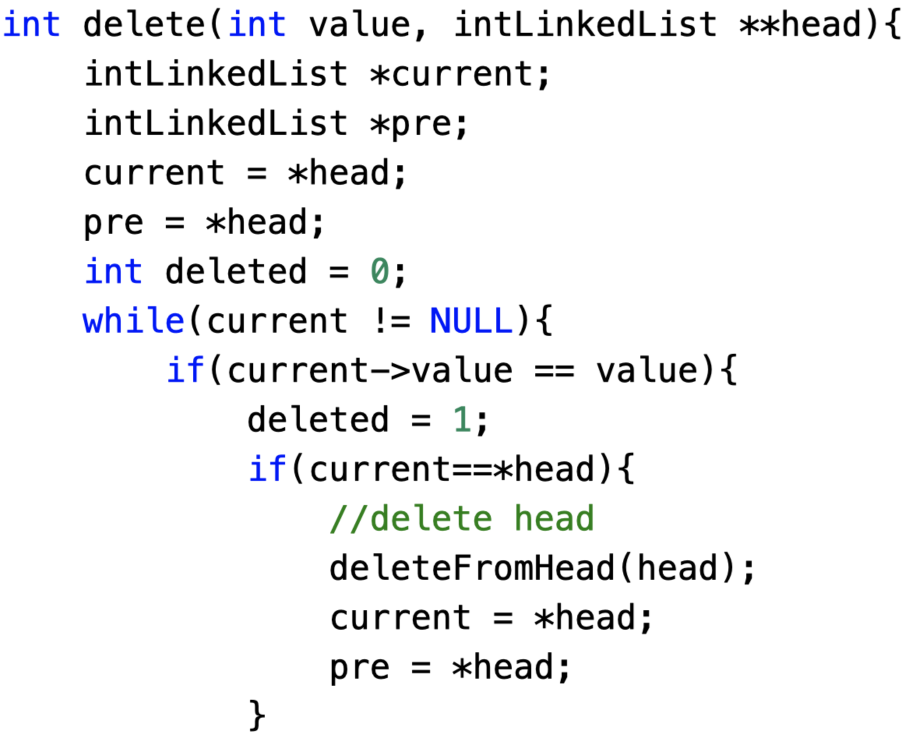


Delete Specific Element – Step 4

- We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.

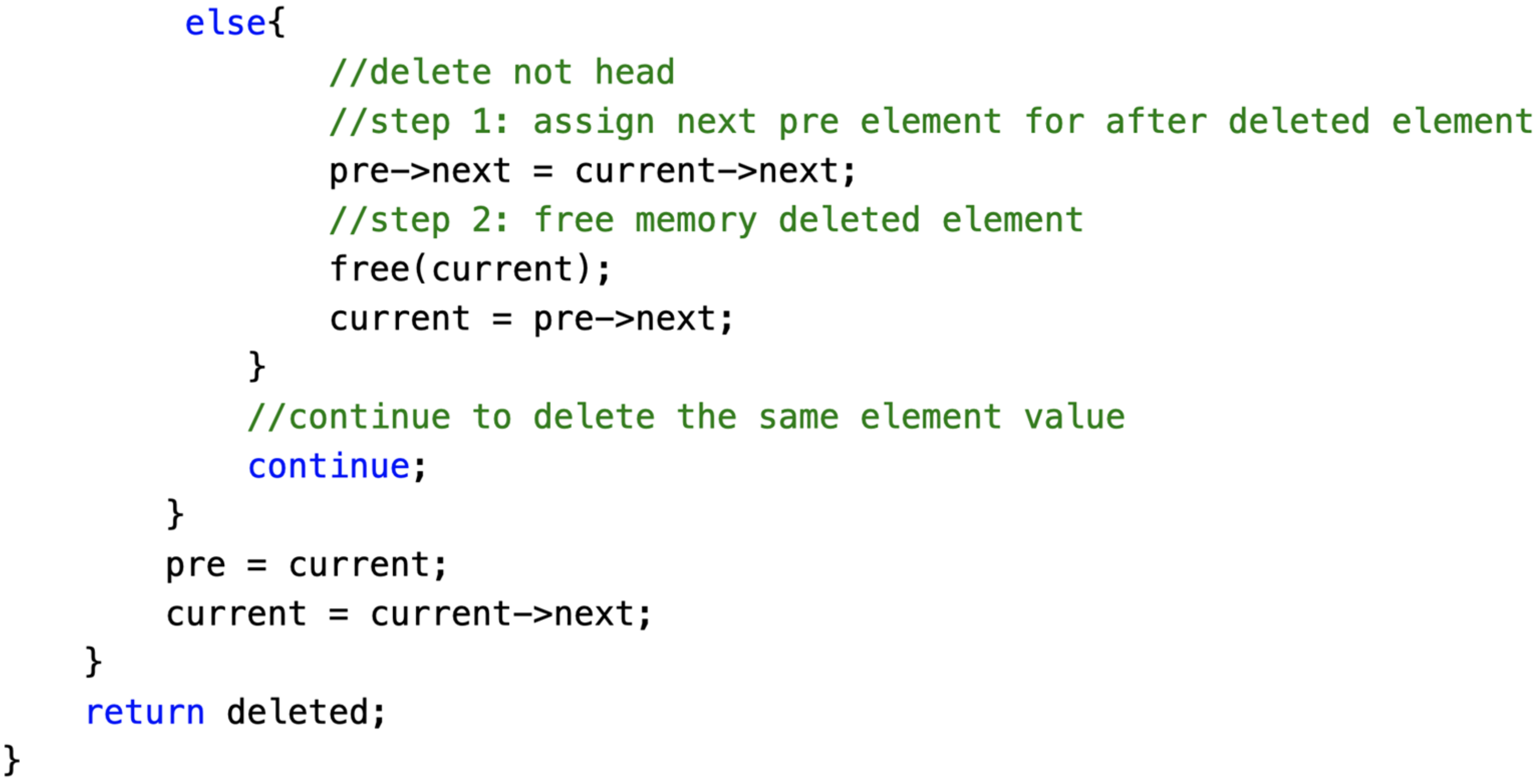


Delete Specific Element – Example



```
int delete(int value, intLinkedList **head){
    intLinkedList *current;
    intLinkedList *pre;
    current = *head;
    pre = *head;
    int deleted = 0;
    while(current != NULL){
        if(current->value == value){
            deleted = 1;
            if(current==*head){
                //delete head
                deleteFromHead(head);
                current = *head;
                pre = *head;
            }
```

Delete Specific Element – Example



```
         else{
                //delete not head
                //step 1: assign next pre element for after deleted element
                pre->next = current->next;
                //step 2: free memory deleted element
                free(current);
                current = pre->next;
            }
            //continue to delete the same element value
            continue;
        }
        pre = current;
        current = current->next;
    }
    return deleted;
}
```

Bookstore Linked List Example

```
// create a book struct
struct book
{
        int isbn;
        char title[100];
};

// create a node in singly linked list
struct node
{
        struct book book;
        struct node *next;
};
```

Summary

- List ADT is a dynamic ordered tuple of homogenous elements.
- The position of Ai is i ; positions range from 0 to N-1 inclusive.
- The size of list is N (a list with no elements is called an “empty list”).
- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Thank you!