CSCI 2270 – Data Structures

Recitation 4, Fall 2021

Linked List Operations

Objectives

1. Linked Lists
2. Insertion
3. Traversal
4. Deletion
5. Exercise

1. Linked Lists

A linked list is an abstract data structure that stores a list, where each element in the list points to the next element.

Let us elaborate:

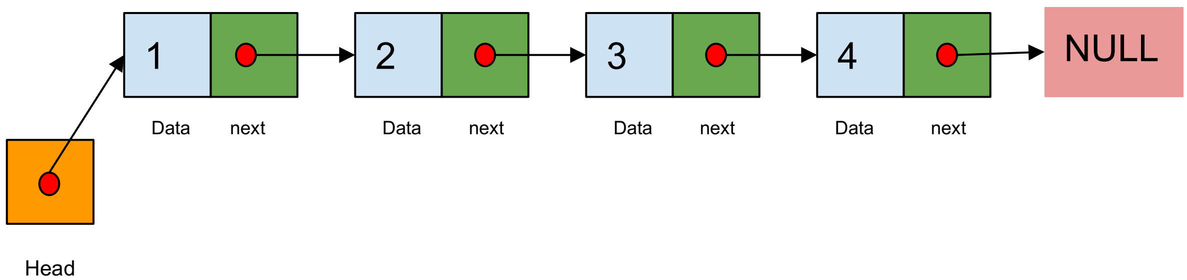
Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after the array. Following are the important terms to understand the concept of Linked List.

- **Node** − Each Node of a linked list can store data and a link.
- **Link** − Each Node of a linked list contains a link to the next Node (unit of Linked List), often called 'next' in code.
- **Linked List** − A Linked List contains a connection link from the first link called Head. Every Link after the head points to another Node (unit of Linked List). The last node points to NULL. Sometimes, the first node of the Linked List is treated as the head.

CSCI 2270 – Data Structures

Recitation 4, Fall 2021

Linked List Operations



In code, each node of the linked list will be represented by a class or struct. These contain, at a minimum, two pieces of information - the data that node contains, and a pointer to the next node. Example code to create these is below:

You can base your Linked List on a **class** or a **struct**, we prefer you stick to **class.**

```
class Node
{
public:
    int data;
    Node *next;
};
```

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

1.a Types of Linked Lists

The one seen above is called a Singly Linked List. You will also work with Doubly Linked Lists where each Node has a pointer to its next as well as its previous Node, and Circular Linked Lists, where the last Node points to the head of the Linked List.

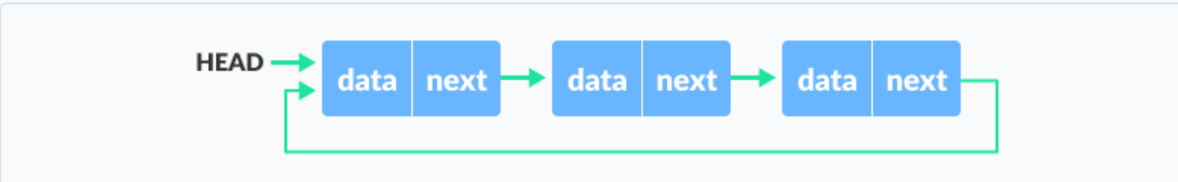
The Two Linked Lists are shown below.

CSCI 2270 – Data Structures

Recitation 4, Fall 2021

Linked List Operations

Doubly linked list



Circular linked list

2. Insertion

Adding a new node to a Linked List is a multi-step activity. For your understanding, we have visualizations. First, you must create a new node that is of the SAME datatype as the other nodes of the Linked List. Then, find the location where it must be inserted.

Different scenarios of insertion

A. Inserting at the beginning of the Linked List.
B. Inserting at the end of the Linked List.
C. Inserting at any given position.

A. Inserting at the beginning.

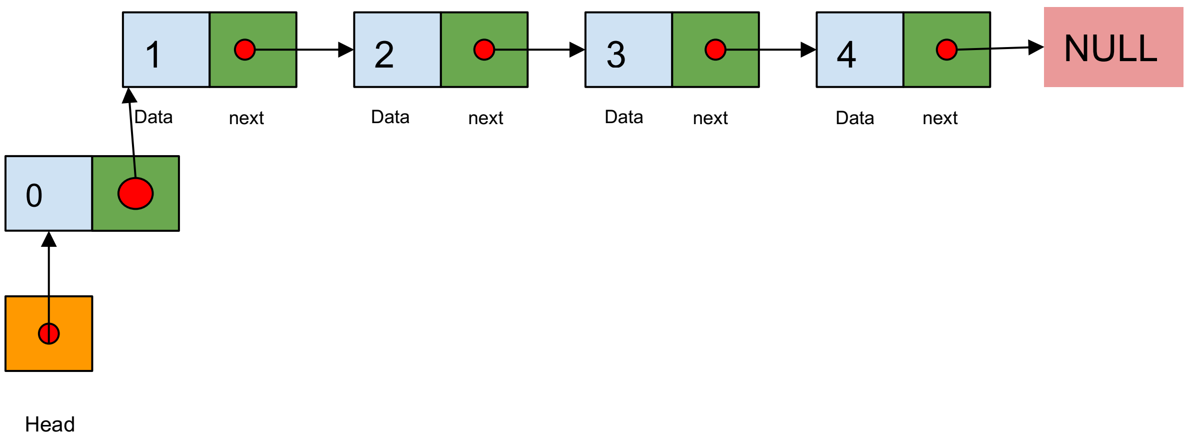
The following diagram shows a Linked List with 4 nodes (1, 2, 3, and 4). And a node with data 0 is inserted at the beginning of the Linked List. The steps are as follows.

a. Create a new node with data 0.
b. Update its *next* pointer to point to the beginning of the Linked List. This is done by making it point to the *head*.
c. Now, update the *head* pointer to point to the new node.

CSCI 2270 – Data Structures

Recitation 4, Fall 2021

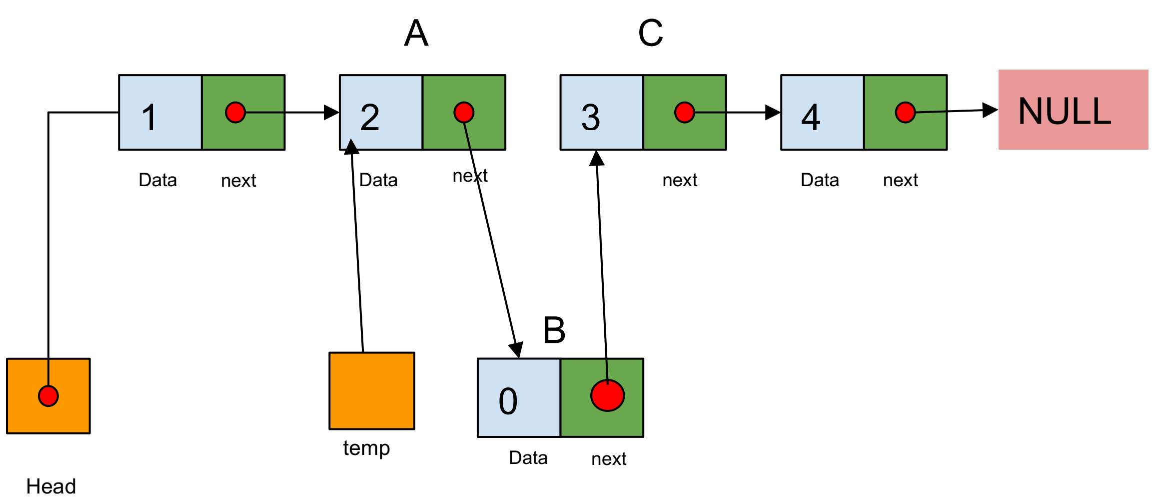
Linked List Operations



B. Inserting at any given position.

For the same Linked List (4 nodes – 1, 2, 3, 4), we would like to insert the Node **B** with data 0, after **A** and before **C**. For this, we follow the steps below.

a. Create the new node **B** with data 0 in it, and make its *next* pointer point to NULL.
b. By getting the location of the first node (Head), traverse the Nodes of the Linked List until the Node with data 2 is found.
c. We must not lose the address of Node **C** which is stored in the *next* pointer of Node **A**. Therefore, we store this address in a temporary variable.
d. We can safely update Node **A**'s *next* pointer to point to the address of Node **B**.
e. Finally, we make the **B**'s *next* pointer to point to the address of Node **C** that is stored in the temporary variable.



CSCI 2270 – Data Structures

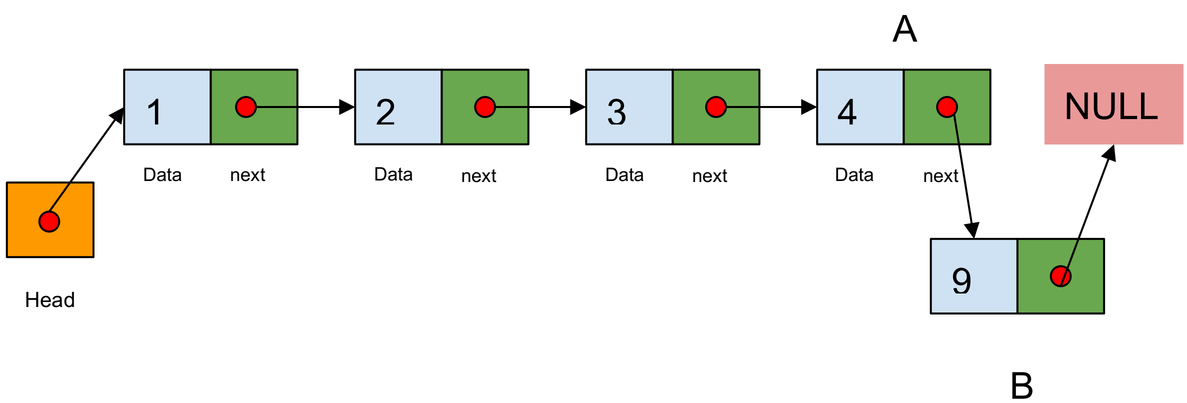
Recitation 4, Fall 2021

Linked List Operations

C. Inserting at the end

Yet again, for the same Linked List (4 Nodes with 1, 2, 3, 4), we would like to insert Node **B**, at the end of the Linked List. For this, we follow the steps below.

a. Create the new node B with the data 9 in it, and make its *next* pointer point to NULL.
b. Beginning at the Head node, traverse the Linked List till you find a node with its *next* pointer pointing to NULL.
c. Update this pointer to point to the address of node B.



Note: It is always a good practice to make a new node's *next* pointer to initially point to NULL.

2. Traversal and printing

To print the elements of the Linked List, we start from its *head* and traverse every node. The traversal is done by using the *next* pointer which points to the next element of the Linked List. This traversal is done until the *next* pointer no longer points to a node (i.e., NULL). In a Circular Linked List, this traversal is done until the *next* pointer points to the **Head.**

```
Node *node = root;
while(node != nullptr) {
    cout << node->value << endl;
    node = node->next;
}
```

CSCI 2270 – Data Structures

Recitation 4, Fall 2021

Linked List Operations

3. Deletion

Like insertion, deleting a node from a Linked List is a multi-step activity. Remember, as we are deleting, we do not need to create a new Node.

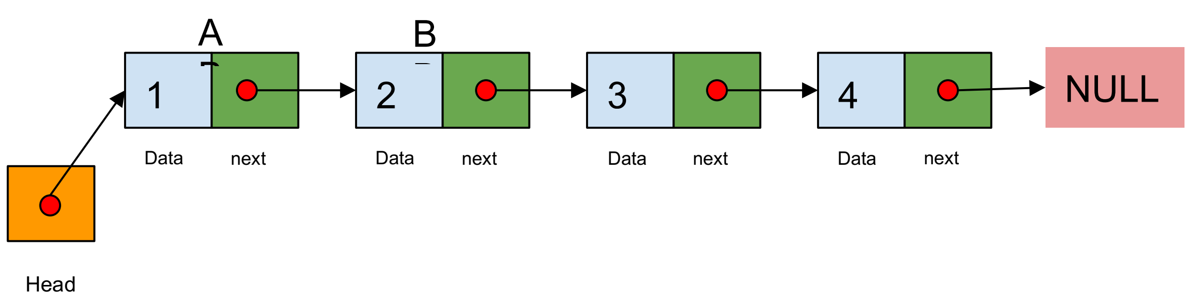
We are going to cover the following scenarios of Deletion.

A. Deleting the head (first node) of the Linked List.
B. Deleting the last node of the Linked List.
C. Deleting the entire Linked List.

1. Deletion of the first node

Here, we are deleting the head Node **A**. Upon careful consideration, we notice that, in-order to delete this node, and maintain the integrity of the Linked List, we must make **B** as the new Head node and free the memory allotted to Node **A**. This is done as follows.

a. Create a *temp* variable and point it to **A.**
b. Using the *next* pointer of the head node **A,** point the Head pointer to **B.**
c. We now have the new Linked List with **B** as its head.
d. Finally, we free the memory of the node **A** by using the *temp* variable we previously created.

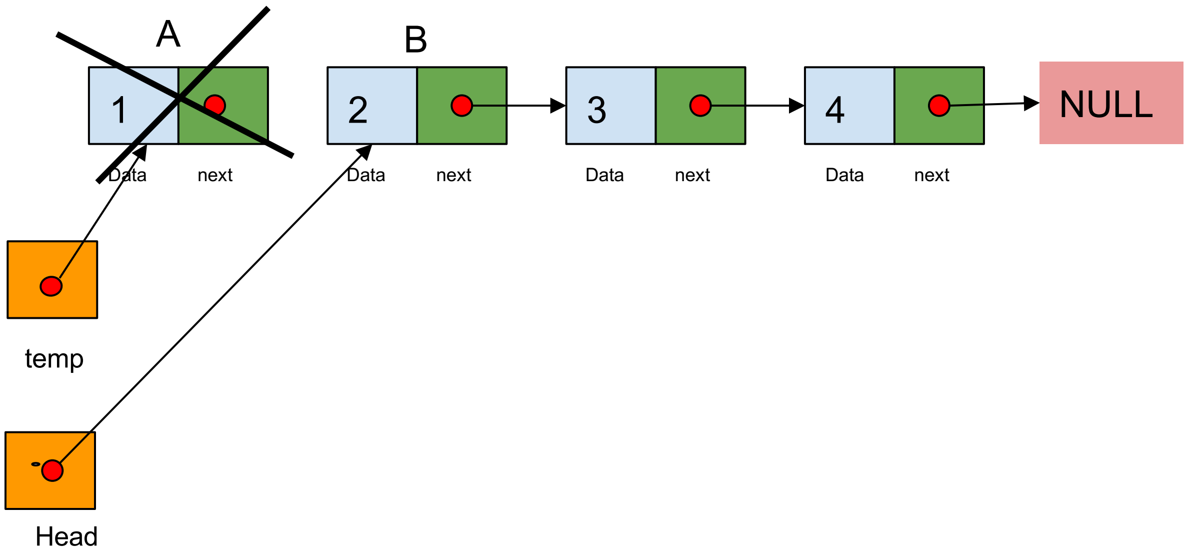


Linked list representation after deletion is given below.

CSCI 2270 – Data Structures

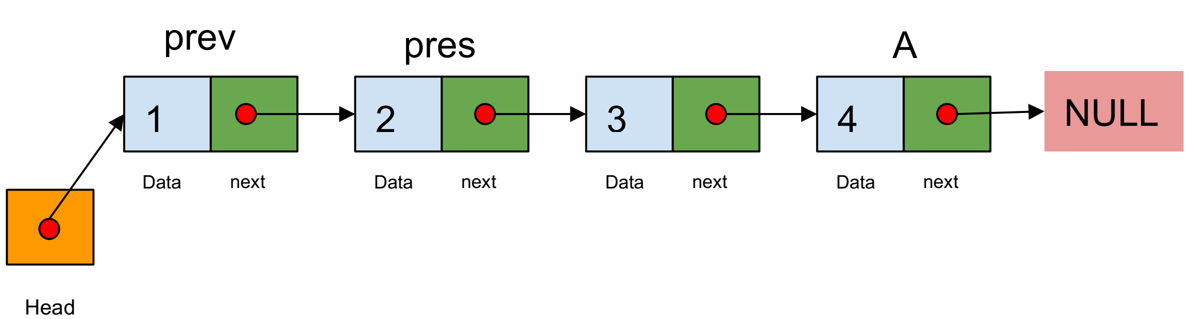
Recitation 4, Fall 2021

Linked List Operations



2. Deletion of the last node

Here, we are deleting the final node (A) of the Linked List. We can see that, in order to delete the final node of the Linked List, we must be able to make its previous node point to NULL and free the memory of the last node **A.** This is achieved by utilizing two pointers – which always points to the previous node traversed and the current node being traversed. As you might have already realized by now, we traverse the Linked List using these two pointers till the final node is reached. We free the memory associated with the node *pres* and make the node *prev* point to NULL. The steps are as follows.



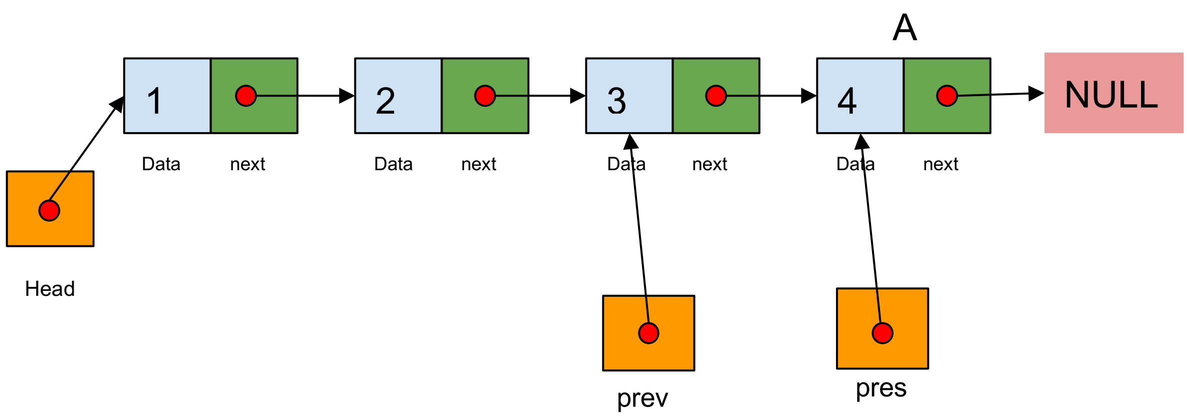
CSCI 2270 – Data Structures

Recitation 4, Fall 2021

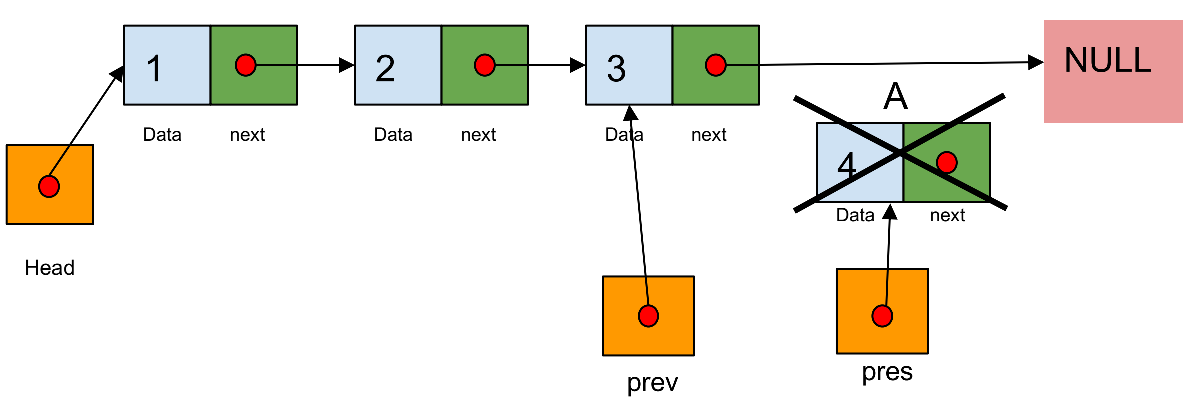
Linked List Operations

a. Start by making the *prev* pointer point to the Head node, and *pres* pointer point to the Head's *next* node.
b. Traverse the Linked List until the *next* pointer of *pres* points to NULL, updating both the pointers *prev* & *pres* point to the nodes as referenced by their respective *next* pointers.
c. Once the *next* pointer of *pres* points to null, free the memory of node *pres* and make *prev* point to null.

Linked list representation after **prev** and **pres** have completed traversing.



Linked list representation after deletion of the node **A** and pointing *prev* to NULL.



CSCI 2270 – Data Structures

Recitation 4, Fall 2021

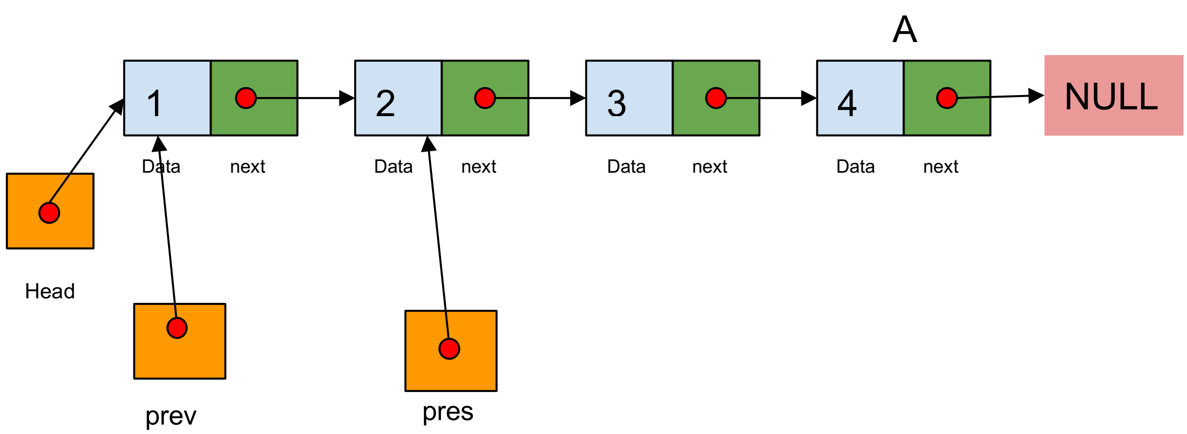
Linked List Operations

3. Deletion of a linked list

The deletion of a linked list involves iteration over all nodes of the Linked List and deleting (freeing) them one by one. This is achieved by maintaining two pointers *prev* and *pres* which are contractions for previous and present. At every traversal, we free the memory associated with *prev* and move the pointers one step forward. The steps are as follows.

e. Make *prev* and *pres* point to the Head node and the node pointed by head's *next* pointer respectively.
f. Free the memory associated with *prev* and move both the pointers one step forward by making *prev* point to *pres* and *pres* point to the node referenced by its *next* pointer.
g. Stop until *pres* is NULL and memory of the node pointed by *prev* is freed.

Given below is a linked list representation before deletion of all nodes of the Linked List.



Linked list representation after deletion of all the nodes.

CSCI 2270 – Data Structures

Recitation 4, Fall 2021

Linked List Operations

4. Quizzes

1. Why do we need to use Linked list instead of Array?
2. What kind of memory allocation do we need to do for Linked lists?
    a. Compile time allocation
    b. Run time allocation
3. What does “Head” mean in a Linked list?
4. Any practical examples of Linked lists that we use in our daily life?

5. Exercise

Download the Recitation 4 folder from canvas. There are LinkedList header, implementation, and main files.

Your task is to complete the following function/functions:

1. **Given a position in the linked list, delete the node at that position.**(Silver problem - Mandatory)
2. Swap the first and last nodes in a linked list (Gold problem)

Submission: Once you’ve completed the exercises, zip all the files up and submit on the Canvas link. Name the submission in the following format: recitation4_firstname_lastname.zip