CSCI 2270 – Data Structures

Recitation 3, Sept 2018

Linked Lists

Objectives

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

1. Linked List

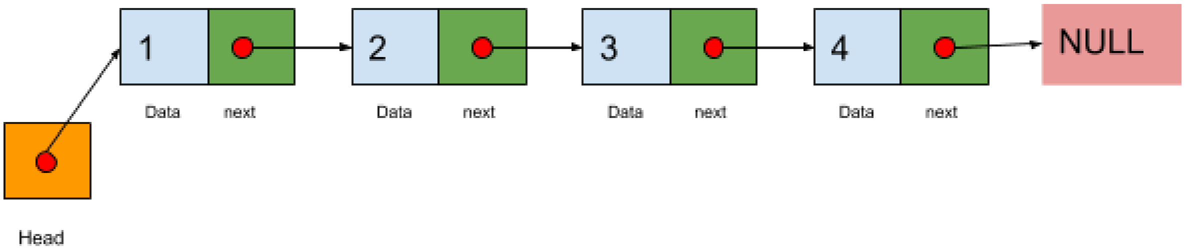
A linked list is a data structure that stores a list, where each element in the list points to the next element.

Let’s 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 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.

Linked Lists



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

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

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

2. Insertion in a linked list

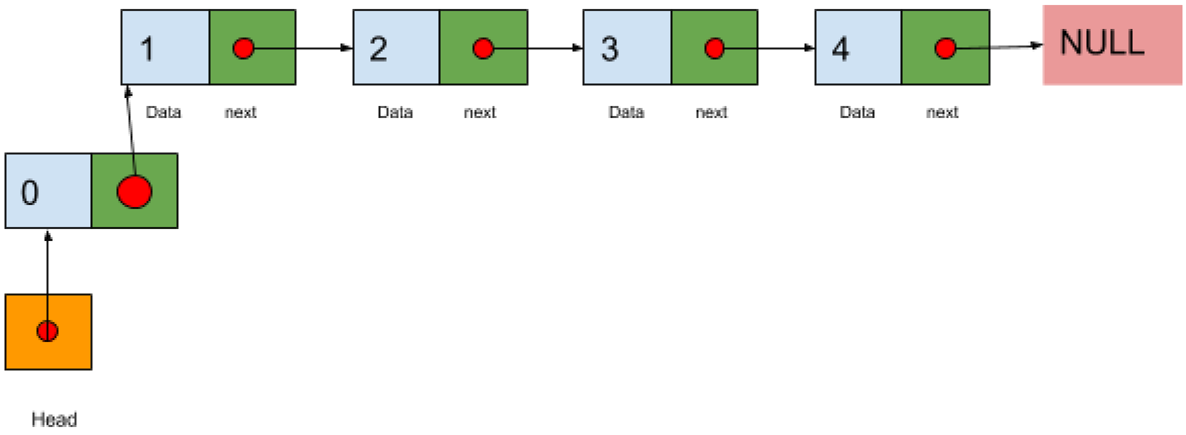
Adding a new node in a linked list is a multi-step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it must be inserted.

Scenarios in insertion

1. Insertion at the start.
2. Insertion at a given position.
3. Insertion at the end.

1. Inserting at the start of the list.

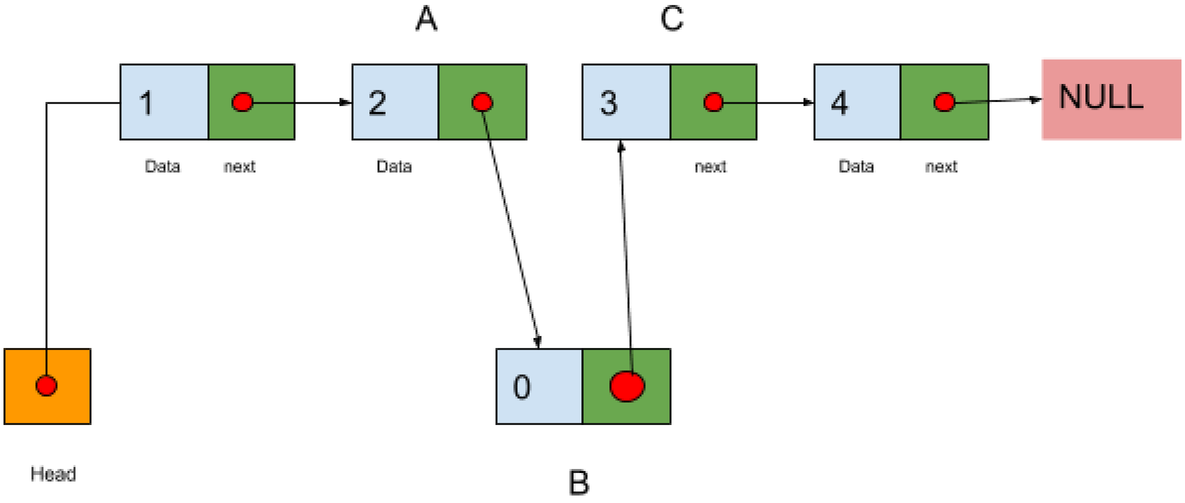
Now we will insert an element at the start of the list.



a. Create a new node,
b. Update the next pointer of that node to start of the list. (Value of the head pointer)
c. Update head pointer to new node.

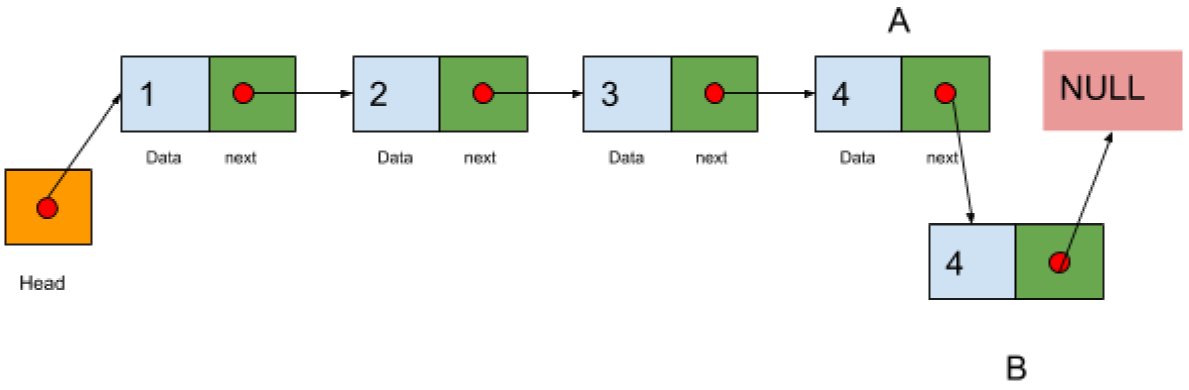
2. Insertion at a given position

For example let us insert a new node at position 2.



a. Create a new node (B).
b. Count and traverse until the node previous to given position i.e A.
c. Store the A’s next pointer value in a temporary variable.
d. Update the next pointer of A to the address of new node.
e. Copy the temporary variable’s value to B’s next pointer.
f. Now B’s next pointer points to the address of the node C.

3. Insertion at the end



a. Create a new node B.
b. Traverse till the node whose next pointer points to NULL. (A)
c. Update the next pointer of A to B’s address.
d. Point B’s next pointer to NULL.

3. Traversal and printing

To print a list, we need to traverse through all the nodes in the list until we encounter the last node. When a node points to NULL we know that it is the last node.

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

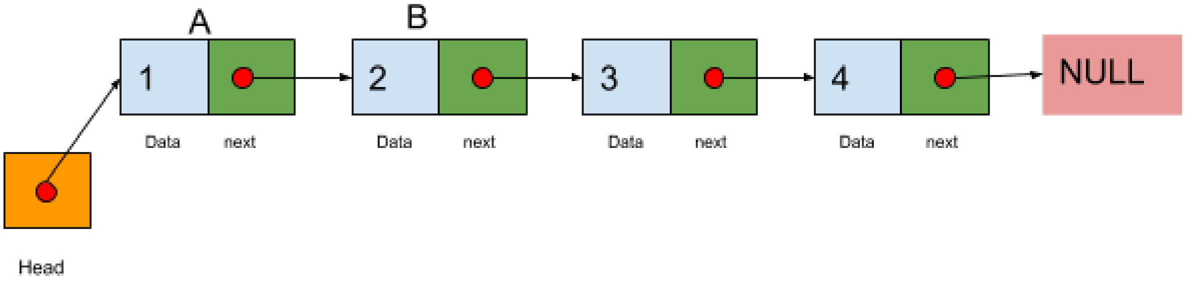
4. Deletion

Deleting a node in the linked list is a multi-step activity. Let's call the node to be deleted as 'A' and the node 'A' is pointing to as 'B'.

- First, the position of the node to be deleted ('A') must be found.
- The next step is to point the node pointing to 'A', to point to 'B'.
- The last step is to free the memory held by 'A'.

1. Deletion of the first node

a. Given below is the linked list representation before the deletion of the first node



b. Steps followed to delete the first node ('A') having value '1'.

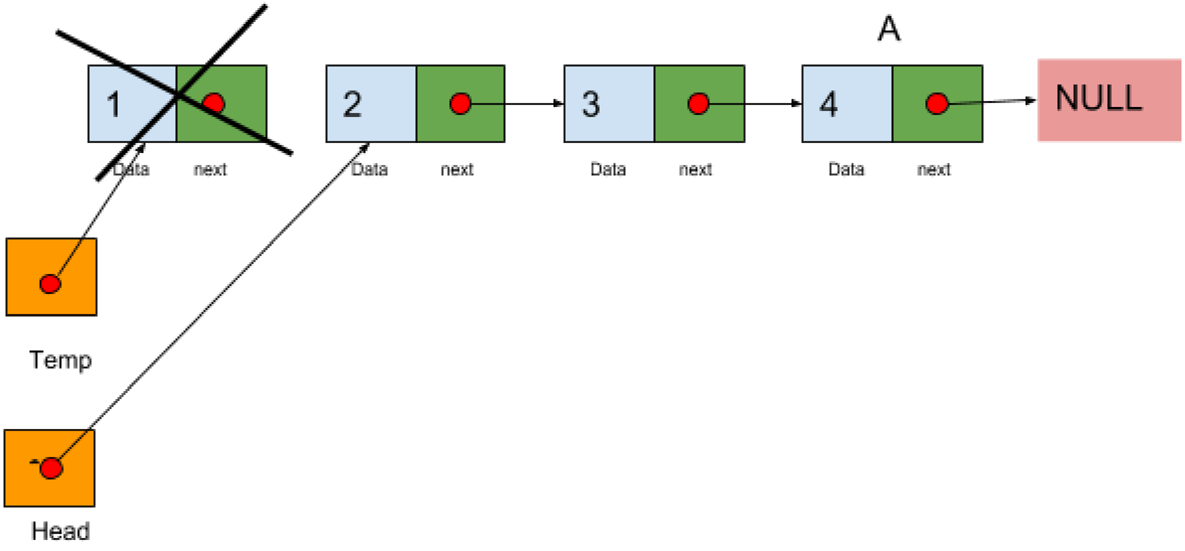
i. Create a variable **temp** having a reference copy of the head node.

ii. Point the head node from 'A' to 'B'

iii. Head is now pointing to 'B'. So, The Linked List's first element now is 'B' with the value '2'

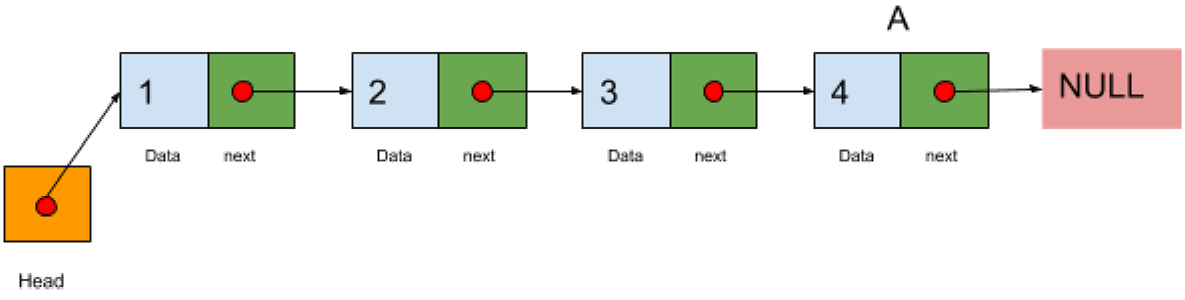
iv. Free the node 'A' pointed to by **temp**.

c. Linked list representation after deletion.



2. Deletion of the last node

a. Given below is a linked list representation before deletion of the last node



b. Steps followed to delete the last node ('A') having value '4'.

i. Create a variable **prev** having a reference copy of the head node.

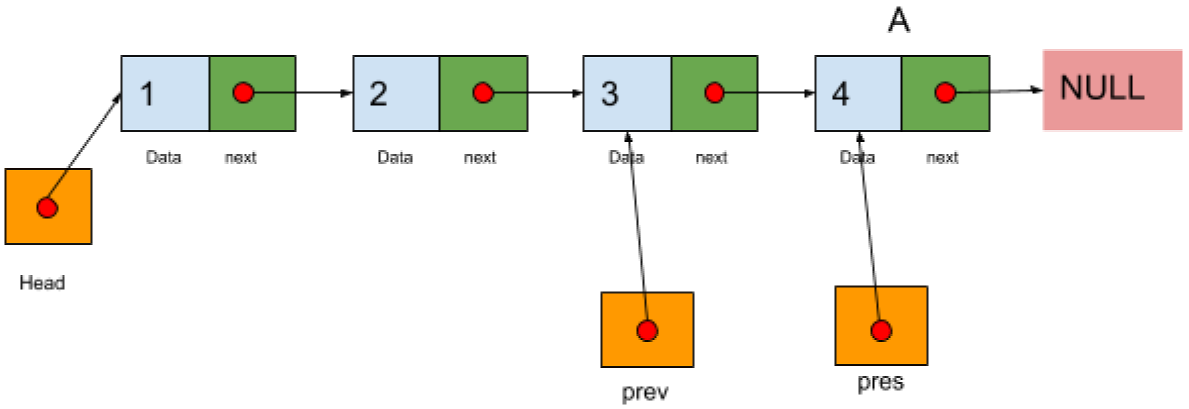
ii. Create a variable **pres** having a reference copy of the next node after the head.

iii. Traverse the list until **pres** is pointing to the last node ‘A’. **prev** will be pointing to the second last node now.

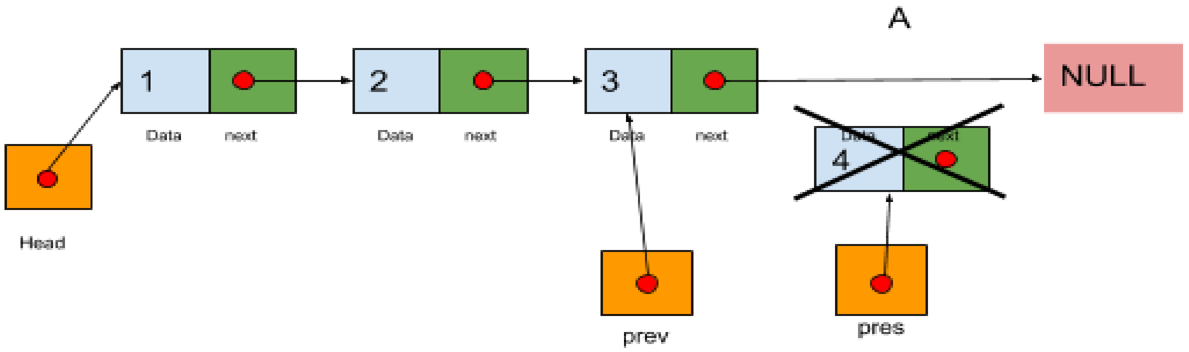
iv. Make **prev** point to **NULL.**

v. Free the node ‘A’ pointed to by **pres.**

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



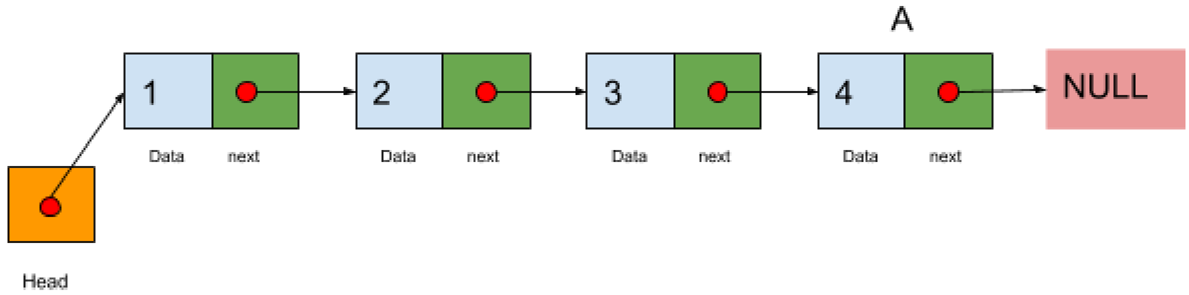
d. Linked list representation after deletion of the node ‘A’ and pointing prev to NULL.



3. Deletion of a Linked List

The deletion of a linked list involves iteration over the complete linked list and deleting (freeing) every node in the linked list.

a. Given below is a linked list representation before deletion of the last node



b. Steps followed to delete every node in the linked list.

i. Create a variable **prev** having a reference copy of the head node.

ii. Create a variable **pres** having a reference copy of the next node after the head.

iv. While traversing the list, at each step delete/free the memory pointed to by **prev.**

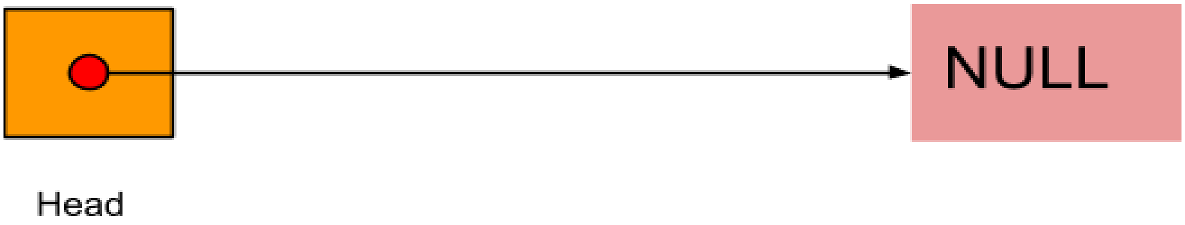
v. Now, point **prev** to the **pres** and point **pres** to the next node after **pres (ie. pres->next)**.

vi. Traverse the list until **pres** is pointing to the **NULL**.

prev will be pointing to the second last node now.

iv. Free the memory pointed to by **prev.** Now every element in the linked list is deleted/freed.

c. Linked list representation after deletion of all the nodes.



Exercise

Download the linkedlist.cpp file from moodle. This file consists of all the node insertion and traversal methods of a linked list.

Your task is to implement deletion for a single case i.e

1. **Given a position in the linked list, delete the node at that position**.

Refer to the steps given in this manual to implement node deletion.