CSCI 2270 – Data Structures

Recitation 5, Fall 2022

Linked List Operations Contd.

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.

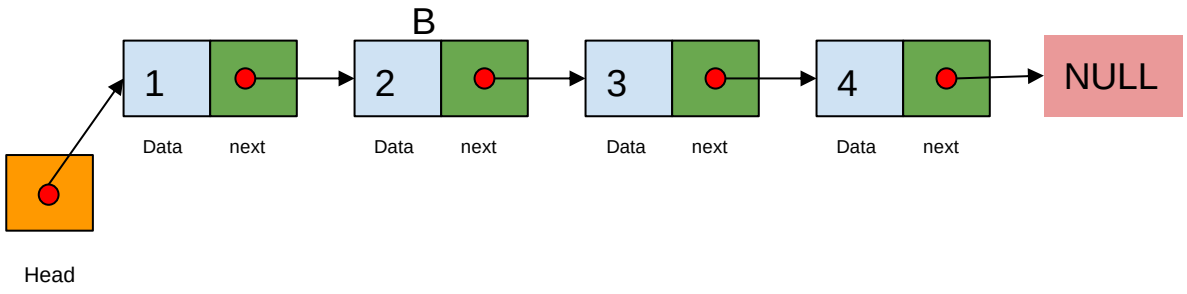
We are going to cover the following scenarios of Deletion.

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.

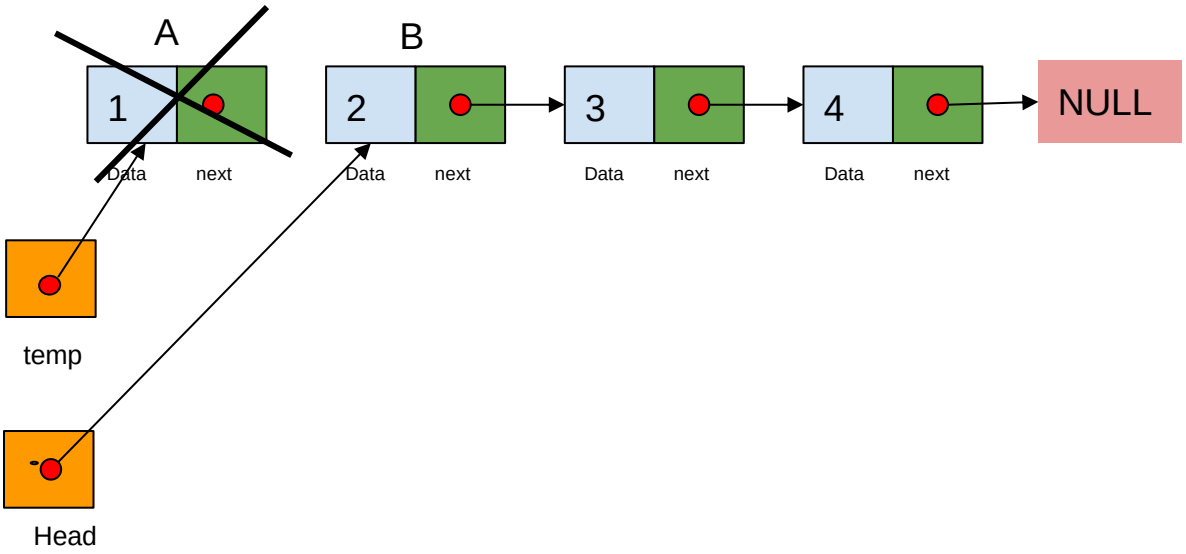


Linked list representation after deletion is given below.

CSCI 2270 – Data Structures

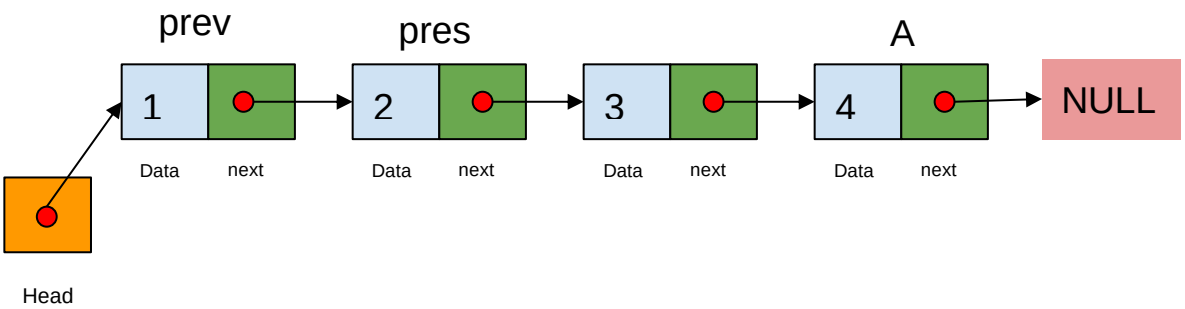
Recitation 5, Fall 2022

Linked List Operations Contd.



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.



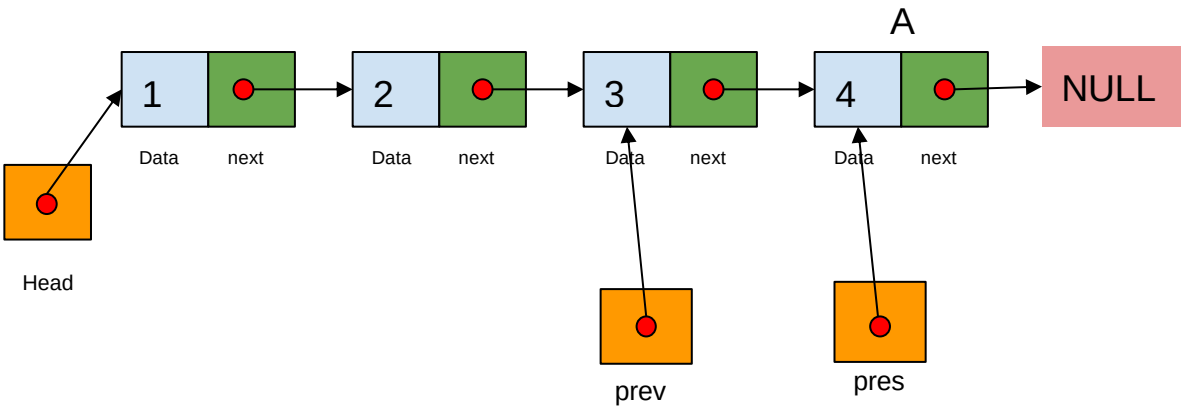
CSCI 2270 – Data Structures

Recitation 5, Fall 2022

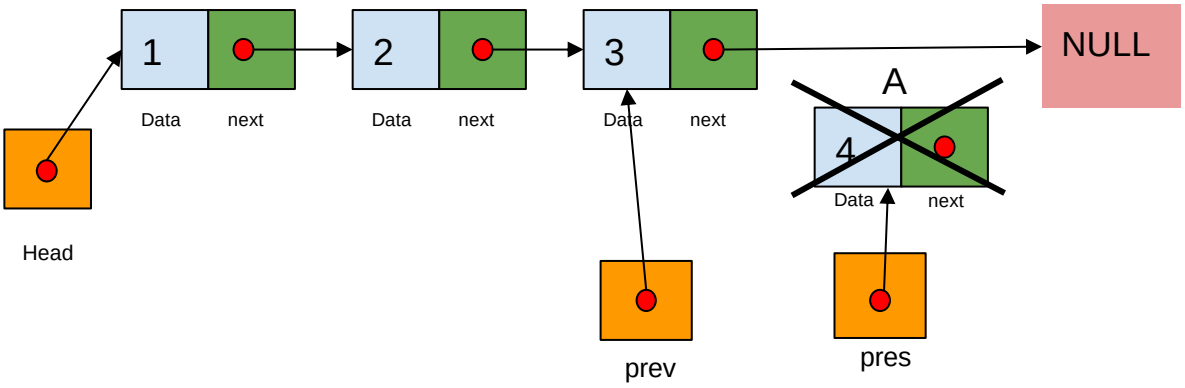
Linked List Operations Contd.

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.



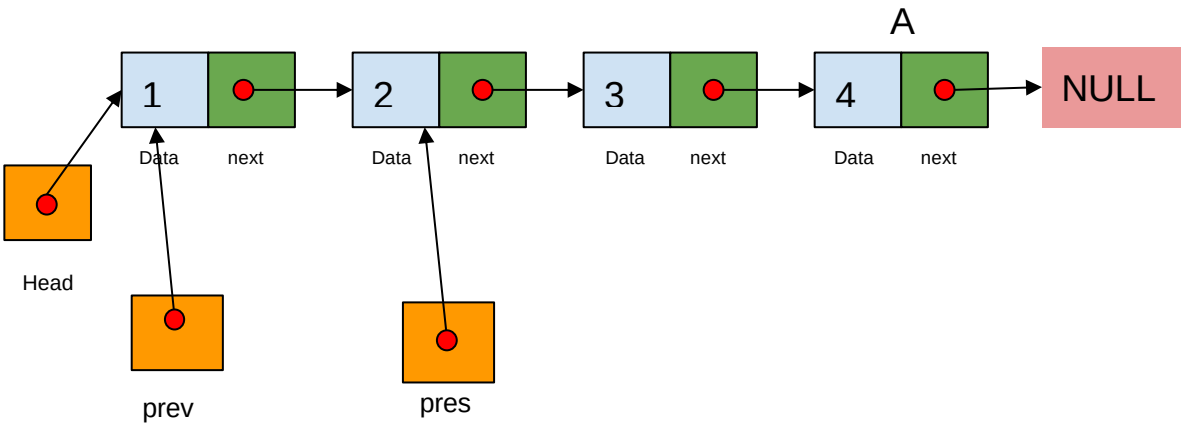
3. Deletion of a linked list

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.

Destructors

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

The thing is to be noted here, if the object is created by using new or the constructor uses new to allocate memory which resides in the heap memory or the free store, the destructor should use delete to free the memory.

Properties of Destructor:

1. Destructor function is automatically invoked when the objects are destroyed.

CSCI 2270 – Data Structures

Recitation 5, Fall 2022

Linked List Operations Contd.

2. It cannot be declared static or const.
3. The destructor does not have arguments.
4. It has no return type not even void.
5. An object of a class with a Destructor cannot become a member of the union.
6. A destructor should be declared in the public section of the class.
7. The programmer cannot access the address of destructor.

When is destructor called?

A destructor function is called automatically when the object goes out of scope:

1. the function ends
2. the program ends
3. a block containing local variables ends
4. a delete operator is called

(source: https://www.geeksforgeeks.org/destructors-c/)

Syntax

```
class LinkedList
{
  private:
    Node *head;

  public:
    LinkedList(){
    ——»head = NULL;
    }
    ~LinkedList();
    void insert(Node *prev, int newKey);
    Node* searchList(int key);
    void printList();
    bool deleteAtIndex(int n);
};
```

Assume the name of the class is LinkedList. The destructor will have the following syntax

~LinkedList()

The definition of the destructor will have no return type. You should free all the memory that has been acquired during the course of execution. For our example we should delete all the node in the destructor.

CSCI 2270 – Data Structures

Recitation 5, Fall 2022

Linked List Operations Contd.

Download the Recitation 5 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. Write the destructor for the linked list (Gold problem)

Submission: Once you've completed the exercises, push the code to github, zip all the files up and submit on the Canvas link.