جامـــــعة المـــســتـقـبـل
AL MUSTAQBAL UNIVERSITY

قســـــم المـــــــــــن الـــــــــــــــسيبرانـــــــــــــــــي

Department of Cyber Security

Subject: Data Structure

Class: Second

Lecturer: Msc :Muntather AL-mussawee

Lecture: (8)

Implementation of Linked List

Study Year: 2025-2026

Implementation of Linked List in C++

To implement a linked list in C++ we can follow the below approach:

Approach:

- *Define a structure **Node** having two members **data** and a **next** pointer. The data will store the value of the node and the next pointer will store the address of the next node in the sequence. For doubly linked list you will have to add an addition pointer **prev** that will store the address of the prev node in the sequence.*
- *Define a class **LinkedList** consisting of all the member functions for the **LinkedList** and a **head** pointer that will store the reference of a particular linked list.*
- *Initialize the **head** to **NULL** as the linked list is empty initially.*
- *Implement basic functions like **insertAtBeginnin**g, **insertAtEnd**, **deleteFromBeginning**, **deleteFromEnd** that will manipulate the elements of the linked list.*

Representation of a Node in the Linked List

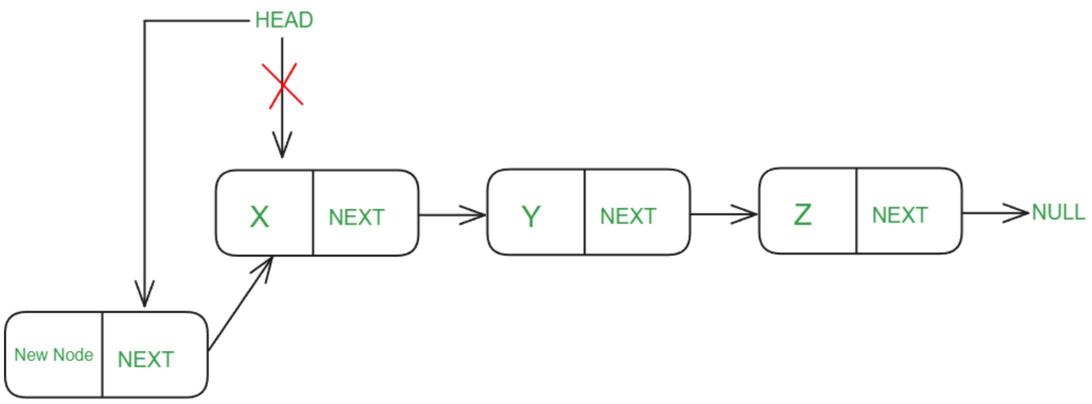
Each node of the linked list will be represented as a structure having two members **data** and **next** where:

- **Data:** Represents the value stored in the node.
- **Next Pointer:** Stores the reference to the next node in the sequence.

```
struct Node {
          int data;
          Node* next;
};
```
Note: Add another data member **Node * prev** in the structure for **doubly linked lists**.

Algorithm for insertAtBeginning Implementation

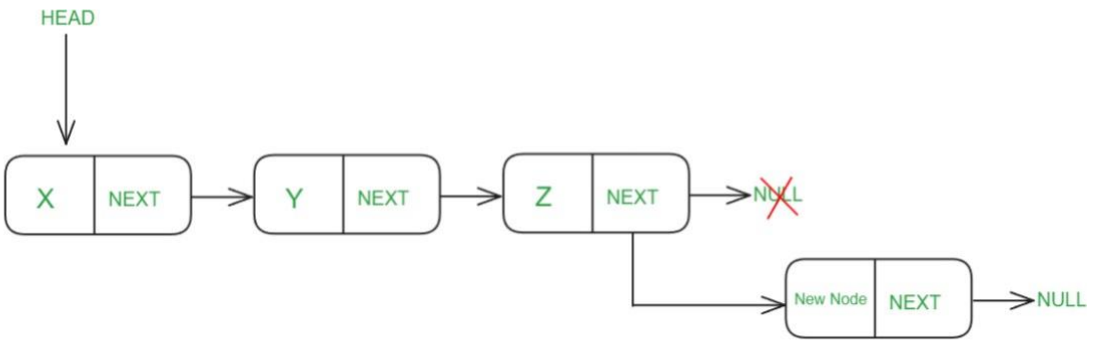
1. *Create a new Node*
2. *Point the new Node's next pointer to the current head.*
3. *Update the head of the linked list as the new node.*



Inserting a node at the first of the linked list

Algorithm for insertAtEnd Implementation

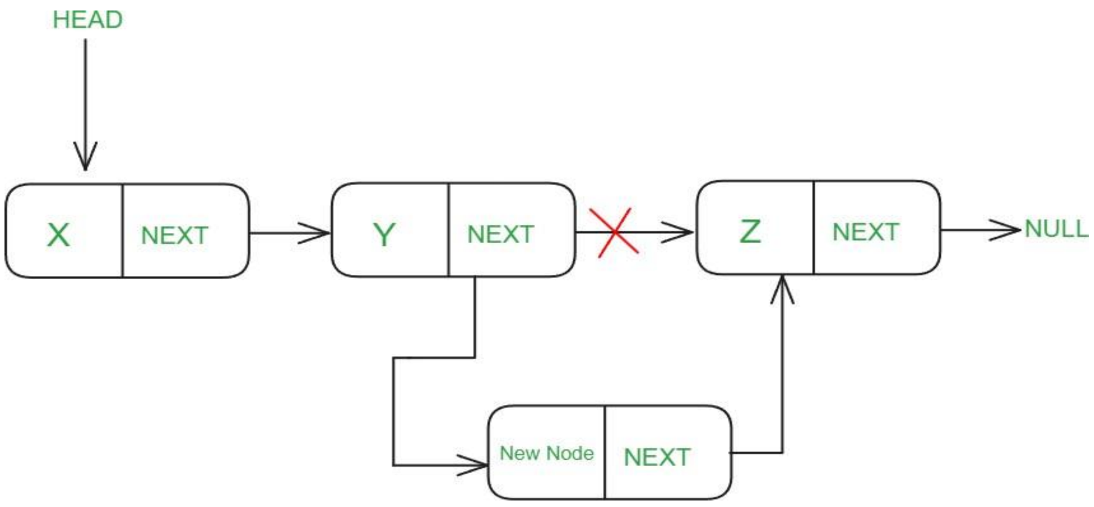
1. *Create a new Node*
2. *If the linked list is empty, update the head as the new node.*
3. *Otherwise traverse till the last node of the linked list.*
4. *Update the next pointer of the last node from NULL to new node.*



Inserting a node at the end of the linked list

Algorithm for insertAtPosition Implementation

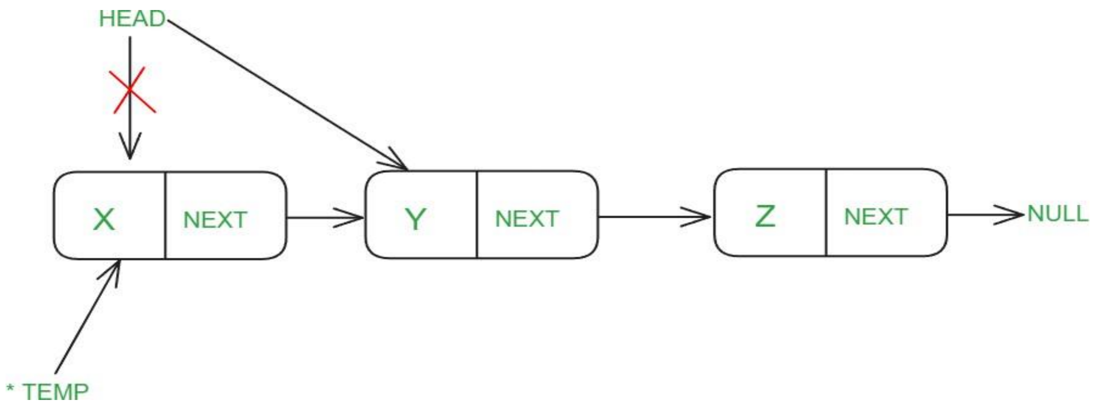
1. *Check if the provided position by the user is a valid poistion.*
2. *Create a new node.*
3. *Find the node at position -1.*
4. *Update the next pointer of the new node to the next pointer of the current node.*
5. *Update the next pointer of the current node to new node.*



Inserting a node after the second position in the linked list

Algorithm for deleteFromBeginning Implementation

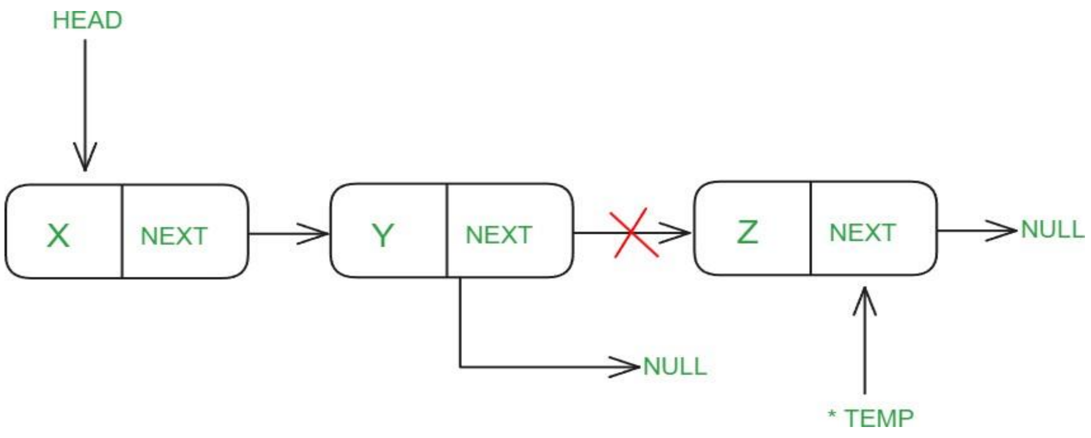
1. *Check whether the* ***Head*** *of the linked list is not NULL. If Head is equal to NULL return as the linked list is empty, there is no node present for deletion.*
2. *Store the head of the linked list in a temp pointer.*
3. *Update the head of the linked list to next node.*
4. *Delete the temporary node stored in the temp pointer.*



Deleting the first node of the linked list

Algorithm for deleteFromEnd Implementation

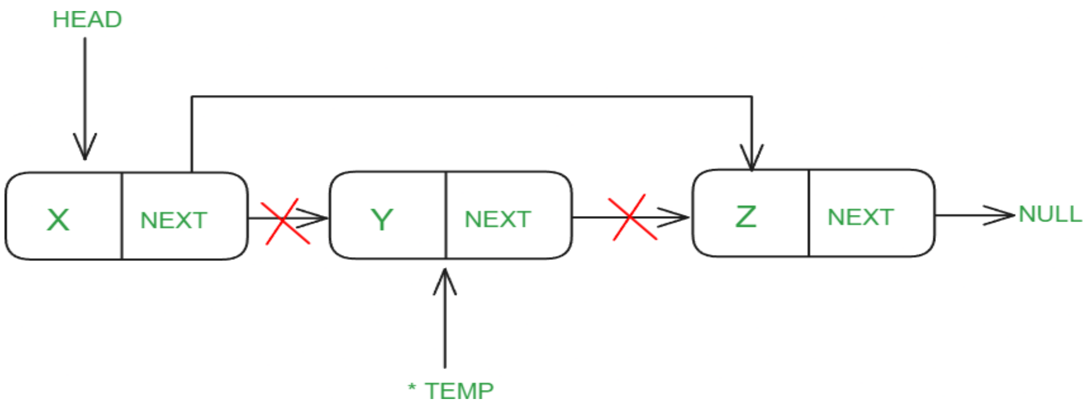
1. *Verify whether the linked is empty or not before deletion.*
2. *If the linked list has only one node, delete head and set head to NULL.*
3. *Traverse till the second last node of the linked list.*
4. *Store the last node of the linked list in a temp pointer.*
5. *Pointer the next pointer of the second last node to NULL.*
6. *Delete the node represented by the temp pointer.*



Deleting the last node of the linked list

Algorithm for deleteFromPosition Implementation

1. *Check if the provided postion by the users is a valid position in the linked list or not.*
2. *Find the node at position -1.*
3. *Save node to be deleted in a temp pointer.*
4. *Set the next pointer of the current node to the next pointer of the node to be deleted.*
5. *Set the next pointer of temp to NULL.*
6. *Delete the node represented by temp pointer.*



Deleting the node present at the second position in the linked list

Algorithm for Display Implementation

```
1. Check if the Head pointer of the linked list is not equal to
   NULL.
2. Set a temp pointer to the Head of the linked list.
3. Until temp becomes null:
    1. Print temp->data
    2. Move temp to the next node.
```

C++ Program for Implementation of Linked List

The following program illustrates how we can implement a singly linked list data structure in C++:

```
#include <iostream>
using namespace std;
// Structure for a node in the linked list
struct Node {
   int data;
   Node* next;
};
// Function to insert a new node at the beginning of the list
void insertAtBeginning(Node*& head, int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->next = head;
   head = newNode;
}
// Function to insert a new node at the end of the list
void insertAtEnd(Node*& head, int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->next = NULL;
   if (!head) {
       head = newNode;
       return;
   }
   Node* temp = head;
   while (temp->next) {
       temp = temp->next;
   }
   temp->next = newNode;
}
// Function to insert a new node at a specific position in the list
void insertAtPosition(Node*& head, int value, int position) {
   if (position < 1) {
       cout << "Position should be >= 1." << endl;
       return;
   }
   if (position == 1) {
       insertAtBeginning(head, value);
       return;
   }
   Node* newNode = new Node();
   newNode->data = value;
   Node* temp = head;
   for (int i = 1; i < position - 1 && temp; ++i) {
       temp = temp->next;
   }
   if (!temp) {
       cout << "Position out of range." << endl;
       delete newNode;
       return;
   }
   newNode->next = temp->next;
   temp->next = newNode;
}
// Function to delete the first node of the list
void deleteFromBeginning(Node*& head) {
   if (!head) {
       cout << "List is empty." << endl;
       return;
   }
   Node* temp = head;
   head = head->next;
   delete temp;
}
// Function to delete the last node of the list
void deleteFromEnd(Node*& head) {
   if (!head) {
       cout << "List is empty." << endl;
       return;
   }
   if (!head->next) {
       delete head;
       head = NULL;
       return;
   }

   Node* temp = head;
   while (temp->next->next) {
       temp = temp->next;
   }
   delete temp->next;
   temp->next = NULL;
}
// Function to delete a node at a specific position in the list
void deleteFromPosition(Node*& head, int position) {
   if (position < 1) {
       cout << "Position should be >= 1." << endl;
       return;
   }
   if (position == 1) {
       deleteFromBeginning(head);
       return;
   }
   Node* temp = head;
   for (int i = 1; i < position - 1 && temp; ++i) {
       temp = temp->next;
   }
   if (!temp || !temp->next) {
       cout << "Position out of range." << endl;
       return;
   }
   Node* nodeToDelete = temp->next;
   temp->next = temp->next->next;
   delete nodeToDelete;
}
// Function to display the nodes of the linked list
void display(Node* head) {
   if (!head) {
       cout << "List is empty." << endl;
       return;
   }
   Node* temp = head;
   while (temp) {
       cout << temp->data << " -> ";
       temp = temp->next;
   }
   cout << "NULL" << endl;
```

```
}
int main() {
    Node* head = NULL;
    // Insert elements at the end
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);

    // Insert element at the beginning
    insertAtBeginning(head, 5);

    // Insert element at a specific position
    insertAtPosition(head, 15, 3);
    cout << "Linked list after insertions: ";
    display(head);

    // Delete element from the beginning
    deleteFromBeginning(head);
    cout << "Linked list after deleting from beginning: ";
    display(head);

    // Delete element from the end
    deleteFromEnd(head);
    cout << "Linked list after deleting from end: ";
    display(head);
    // Delete element from a specific position
    deleteFromPosition(head, 2);
    cout << "Linked list after deleting from position 2: ";
    display(head);
    return 0;
}
```