Linked List Operations: Traverse, Insert and Delete

There are various linked list operations that allow us to perform different actions on linked lists. For example, the insertion operation adds a new element to the linked list.

Here's a list of basic linked list operations that we will cover in this article.

* [Traversal](https://www.programiz.com/dsa/linked-list-operations#traverse) - access each element of the linked list
* [Insertion](https://www.programiz.com/dsa/linked-list-operations#add) - adds a new element to the linked list
* [Deletion](https://www.programiz.com/dsa/linked-list-operations#delete) - removes the existing elements
* [Search](https://www.programiz.com/dsa/linked-list-operations#search) - find a node in the linked list
* [Sort](https://www.programiz.com/dsa/linked-list-operations#sort) - sort the nodes of the linked list

**Things to Remember about Linked List**

* *head* points to the first node of the linked list
* *next* pointer of the last node is NULL, so if the next current node is NULL, we have reached the end of the linked list.

In all of the examples, we will assume that the linked list has three nodes *1 --->2 --->3*with node structure as below:

struct node {

int data;

struct node \*next;

};

**Traverse a Linked List**

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

When temp is NULL, we know that we have reached the end of the linked list so we get out of the while loop.

while (tnode != null) {

System.out.print(tnode.data + " ");

tnode = tnode.next;

}

The output of this program will be:

List elements are -

1 --->2 --->3 --->

**Insert Elements to a Linked List**

You can add elements to either the beginning, middle or end of the linked list.

**1. Insert at the beginning**

* Allocate memory for new node
* Store data
* Change next of new node to point to head
* Change head to point to recently created node

// Insert at the beginning

public void insertAtBeginning(int new\_data) {

// insert the data

Node new\_node = new Node(new\_data);

new\_node.next = head;

head = new\_node;

}

**2. Insert at the End**

* Allocate memory for new node
* Store data
* Traverse to last node
* Change next of last node to recently created node

// Insert at the end

public void insertAtEnd(int new\_data) {

Node new\_node = new Node(new\_data);

if (head == null) {

head = new Node(new\_data);

return;

}

new\_node.next = null;

Node last = head;

while (last.next != null)

last = last.next;

last.next = new\_node;

return;

}

**3. Insert at the Middle**

* Allocate memory and store data for new node
* Traverse to node just before the required position of new node
* Change next pointers to include new node in between

// Insert after a node

public void insertAfter(Node prev\_node, int new\_data) {

if (prev\_node == null) {

System.out.println("The given previous node cannot be null");

return;

}

Node new\_node = new Node(new\_data);

new\_node.next = prev\_node.next;

prev\_node.next = new\_node;

}

**Delete from a Linked List**

You can delete either from the beginning, end or from a particular position.

**1. Delete from beginning**

* Point head to the second node

head = head->next;

**2. Delete from end**

* Traverse to second last element
* Change its next pointer to null

**Search an Element on a Linked List**

You can search an element on a linked list using a loop using the following steps. We are finding item on a linked list.

* Make head as the current node.
* Run a loop until the current node is NULL because the last element points to NULL.
* In each iteration, check if the key of the node is equal to item. If it the key matches the item, return true otherwise return false.

// Search a node

boolean search(Node head, int key) {

Node current = head;

while (current != null) {

if (current.data == key)

return true;

current = current.next;

}

return false;

}

**Sort Elements of a Linked List**

We will use a simple sorting algorithm, [Bubble Sort](https://www.programiz.com/dsa/bubble-sort), to sort the elements of a linked list in ascending order below.

1. Make the head as the current node and create another node index for later use.
2. If head is null, return.
3. Else, run a loop till the last node (i.e. NULL).
4. In each iteration, follow the following step 5-6.
5. Store the next node of current in index.
6. Check if the data of the current node is greater than the next node. If it is greater, swap current and index.

// Sort the linked list

void sortLinkedList(Node head) {

Node current = head;

Node index = null;

int temp;

if (head == null) {

return;

} else {

while (current != null) {

// index points to the node next to current

index = current.next;

while (index != null) {

if (current.data > index.data) {

temp = current.data;

current.data = index.data;

index.data = temp;

}

index = index.next;

}

current = current.next;

}

}

}