**Data Structure**

**Data Structure**

* A data structure is a storage that is used to to store and organize data.
* It is a way of arranging data on a computer so that iit can be accessed and updated efficiently.
* A data structure is not only used for organizing the data it is also used for processing retrieving and storing data.

**Classificaton of Data Structure**

There are two type of data structure.

1. Linear data structure

1.1 Static Data Structure

1.1.1 Array

1.2 Dynamic Data Structure

1.2.1 Queue

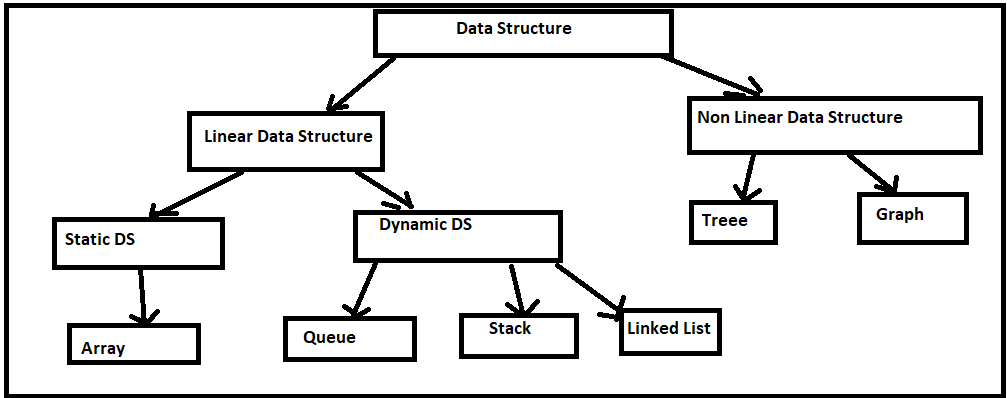
1.2.2 Stack

1.2.3 Linked List

2. Non Linear Data Structue

2.1 Tree

2.2 Graph



**Linear Data Structure**

* Data Structure in which data elements are arranged sequentially or linearly, where each elements is attached to its previous and next adjucent elements is called a linear data structure.

**Example**=> Array, Stack, Queue, Linked List etc.

**Static Data Structure**

* Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.

**Example**=> Array

**Dynamic Data Structure**

* In dynamic data structure the size is not defined. It can be randomly updated during the runtime which may be considered efficient concerning the memory(space) complexity of the code.

**Example**=> Queue, Stack, Linked List

**Non Linear Data Structure**

* Data Structure where data elements are not placed sequentially are called Non-Linear Data Structure. We can't traverse all the elements in a single run only.

**Example**=> Tree and Graph

**LINKED LIST**

A Linked List is a**linear data structure** which looks like a chain of nodes, where each node is a different element. Unlike Arrays, Linked List elements are not stored at a contiguous location.

**Why linked list data structure needed?**

Here are a few advantages of a linked list that is listed below, it will help you understand why it is necessary to know.

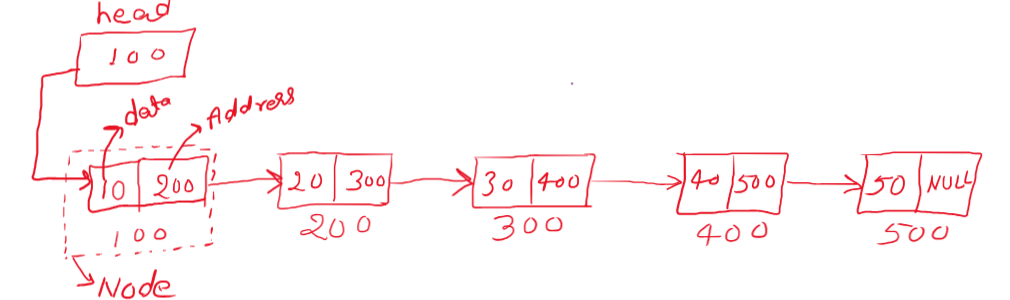
* **Dynamic Data structure:** The size of memory can be allocated or de-allocated at run time based on the operation insertion or deletion.
* **Ease of Insertion/Deletion:** The insertion and deletion of elements are simpler than arrays since no elements need to be shifted after insertion and deletion, Just the address needed to be updated.
* **Efficient Memory Utilization:**As we know Linked List is a dynamic data structure the size increases or decreases as per the requirement so this avoids the wastage of memory.
* **Implementation:**Various advanced data structures can be implemented using a linked list like a stack, queue, graph, hash maps, etc.

**There are three type of linked list**

* Singly Linked List
* Doubly Linked List
* Circular Linked List

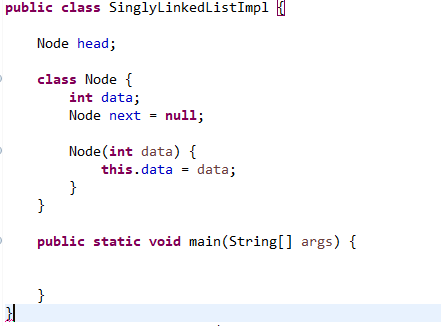
**Singly Linked List**

A **singly linked list** is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.



**How to create list**

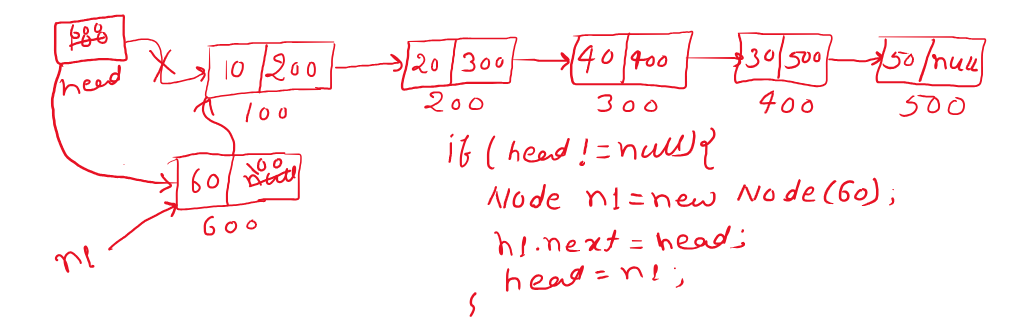
**Node Structure**

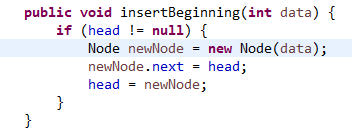
****

**Insertion of the Singly Linked List**

* At the front of the linked list
* After a given node.
* At the end of the linked list

## [How to Insert a Node at the Front/Beginning of Linked List](https://www.geeksforgeeks.org/insert-a-node-at-front-beginning-of-a-linked-list/)

* Make the first node of Linked List linked to the new node
* Remove the head from the original first node of Linked List
* Make the new node as the Head of the Linked List.

****

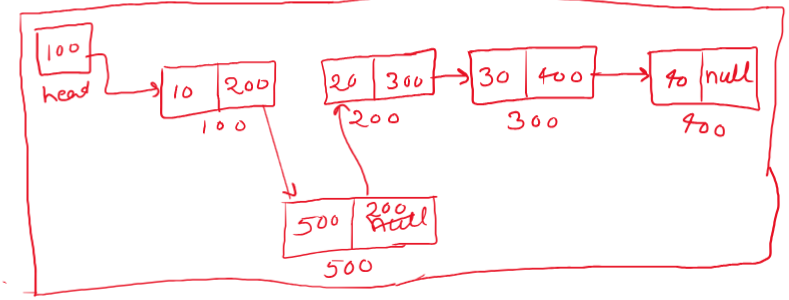
**Complexity Analysis:**

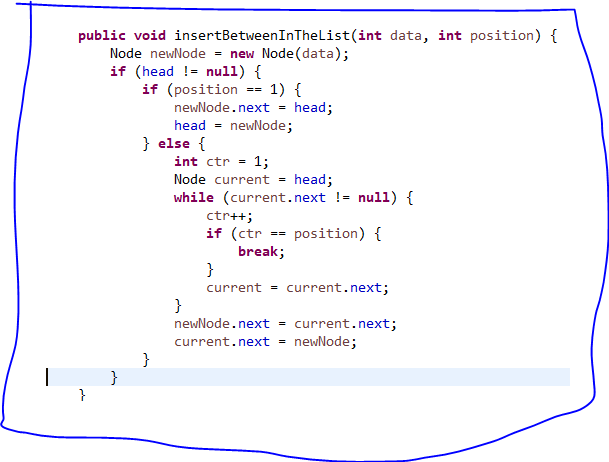
* **Time Complexity:** O(1), We have a pointer to the head and we can directly attach a node and change the pointer. So the Time complexity of inserting a node at the head position is O(1) as it does a constant amount of work.
* **Auxiliary Space:**O(1)

## How to Insert a Node Between of the List

In this list need to add new node between of the list

* create the current node from head.
* Do the while loop based on current node. If while loop iteration is equal of position the break the while loop
* Assign current into current of next node.
* Create the new node
* Current.next address assign into the new node of next.
* Assign new node address into the current.next





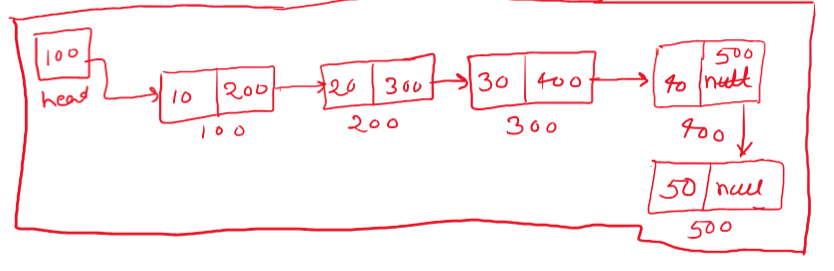
[How to Insert a Node at the End of Linked List](https://www.geeksforgeeks.org/insert-node-at-the-end-of-a-linked-list/)

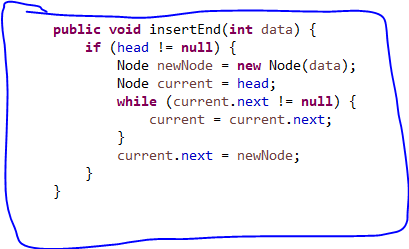
**Approach:**

To insert a node at the end of a Linked List, we need to:

* Go to the last node of the Linked List
* Change the next pointer of last node from NULL to the new node
* Make the next pointer of new node as NULL to show the end of Linked List

After insertion of new node in the last of the list





**Full Code of Linked List Operation**

**public** **class** **SinglyLinkedListImpl** {

Node head;

**class** Node {

**int** data;

Node next = **null**;

Node(**int** data) {

**this**.data = data;

}

}

**public void createList**(**int** data) {

Node newNode = **new** Node(data);

**if** (head == **null**) {

head = newNode;

} **else** {

Node current = head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newNode;

}

}

**public** **void** **displayList**() {

**if** (head != **null**) {

Node current = head;

**while** (current != **null**) {

System.***out***.print(current.data + " ");

current = current.next;

}

System.***out***.println();

}

}

**public** **void** **insertBeginning**(**int** data) {

**if** (head != **null**) {

Node newNode = **new** Node(data);

newNode.next = head;

head = newNode;

}

}

**public** **void** **insertEnd**(**int** data) {

**if** (head != **null**) {

Node newNode = **new** Node(data);

Node current = head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newNode;

}

}

**public** **void** **insertBetweenInTheList**(**int** data, **int** position) {

Node newNode = **new** Node(data);

**if** (head != **null**) {

**if** (position == 1) {

newNode.next = head;

head = newNode;

} **else** {

**int** ctr = 1;

Node current = head;

**while** (current.next != **null**) {

ctr++;

**if** (ctr == position) {

**break**;

}

current = current.next;

}

newNode.next = current.next;

current.next = newNode;

}

}

}

**public** **void** **reverseList**() {

Node prev = **null**;

Node current = head;

Node next = **null**;

**while** (current != **null**) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

**if**(prev != **null**) {

head = prev;

}

displayList();

}

**public** **static** **void** main(String[] args) {

SinglyLinkedListImpl list = **new** SinglyLinkedListImpl();

**for** (**int** i = 1; i <= 10; i++) {

list.createList(i);

}

list.displayList();

System.***out***.println("insert beginning of the list");

list.insertBeginning(1 );

list.displayList();

System.***out***.println("insert end of the list");

list.insertEnd(12);

list.displayList();

System.***out***.println("insert between of the list");

list.insertBetweenInTheList(13, 1);

list.displayList();

System.***out***.println("reverse of the list");

list.reverseList();

}

}