1. When we save data in array like data structure then the first element is positioned at first in that array(index 0) then the second element is positioned at second(index 1), third element is positioned at third(index 2) and all other elements are positioned at sequentially. Now in this article we will learn about a new data structure name linked list. In linked list, data are not stored in sequentially but in different memory locations. So we can not get data by using index, like array. In here data are linked through pointers. In details, a linked list is a data structure that consists of a sequence of nodes where each node stores a reference to an object and a reference to the next node in the sequence. It is commonly used for their efficient insertion and deletion operations.
2. In array,there are a length. But,in linked list,there is no length.
3. Constructor called at a one time
4. Object is a combination of collective data(key-value) and method
5. Array is a contigious block of memory. It stores data in one line. It’s store a data in 1 dimension. In RAM normally data stored 2 dimension
6. In linked list,memory generated dynamically. It's not contigious. There is an address here.

A linked list contains a nodes. Inside the node, there are a current value and next value.The next value contains the address. In this case object will be needed. Because there are key-value pairs. In linked list ,there is no meaning of index. There is only position

1. Object is a reference type data. Reference means provides data not directly. It’s provide address
2. There are three common types of Linked List:

* **Singly Linked List(1 way)**
* **Doubly Linked List(2 way)**
* **Circular Linked List**

1. **Singly Linked List**

A singly linked list is the most basic type of linked list. Each node in the list stores a pointer to the next node in the sequence, except for the last node which stores a null reference.

**Implementation**

To implement a linked list in javascript we will follow the object-oriented way. So that we need a ***node class*** and a ***linked list class.***

**Class Node**

The purpose of the ***Node class*** is to represent the individual elements (nodes) that made up the linked list. To represent a node class, we need to have two properties. First is ***value***that will represent the data to be stored and the ***next,*** the property represents a pointer to the next item in the list.

class Node {  
 constructor(value, next = null) {  
 this.value = value;   
 this.next = next;   
 }  
}

**Class LinkedList**

The **LinkedList class** represents the linked list data structure, which is a collection of nodes that are linked together through their next properties. The purpose of the **LinkedList class** is to provide methods for manipulating the linked list such as adding, updating and deleting nodes. The **LinkedList class** also has properties such as **head**, **tail**and **length**to keep track of the first and last nodes in the list and the number of nodes in the list respectively.

Initially the first node will become **head**and **tail**and **length**is set to 1 of the **LinkedList class**, then in different type of methods we will update these properties periodically.

class LinkedList {  
 constructor(data) {  
 let node = new Node(data);   
 this.head = this.tail = node;   
 this.length = 1;   
 }  
}

**Node append method**

append(value) {  
 let node = new Node(value);   
 this.tail.next = node;   
 this.tail = node;   
 this.length++;   
}

The **append**method is used for adding a new node at the end of the linked list. In this code snippet:

* The method takes a **value**parameter, which represents the value to be stored in the new node.
* Now we create a new **Node object** using this value.
* The next property of the current **tail**node (i.e. the last node in the list) is set to the new node.
* Then the tail property of the **LinkedList object** is updated to the new node.
* Update the **length**property by 1.

**Node prepend method**

prepend(value) {  
 let node = new Node(value);   
 node.next = this.head;   
 this.head = node;   
 this.length++;   
}

The **prepend**method is used for adding a new node at the beginning of the linked list. Here are the below steps of how it works:

* This method also takes a **value**parameter and a new **Node** object is created with this value parameter.
* The next property of the new node is set to the current **head**node of the linked list.
* The **head**property of the **LinkedList** object is updated to the new node.
* Finally the **length**property of the object is incremented by 1.

**Adding a node at specific position**

appendAtPosition(value, n) {   
 if(n === 1) {  
 this.prepend(value);   
 return;   
 }   
  
 if(n > this.length) {  
 this.append(value);   
 return;   
 }  
  
 let node = new Node(value);   
 let prevNode = this.findNode(n-1);   
 const temp = prevNode.next;   
 prevNode.next = node;   
 node.next = temp;   
 this.length++;   
}   
  
findNode(n) {  
 let data = this.head;   
 let count = 0;   
  
 while(data) {  
 count++;   
 if(count === n) {  
 break;   
 }   
  
 data = data.next;   
 }   
  
 return data;   
}

In this code snippet:

* The **appendAtPosition**method takes two arguments: **value**, which is the value to be added to the list and **n**, which is the position in the list where the new value should be inserted.
* Now, first check if the new node can be added at the beginning of the list (i.e. if **n**is 1). If so, it calls the **prepend**method to add the new node at the beginning of the list.
* Then, check the new node can be added at the end of the list (i.e. if **n**is greater than the length of the list). In this situation we call the **append**method.
* If neither of these cases are true, then we create a new **Node**instance with the given value. Then call the **findNode**method to locate the previous node so that the new node can be inserted at the given position. Then assign this node to the variable **prevNode**.
* Now, assign the next node of **prevNode**to a temporary variable **temp**. Then update the next property of **prevNode**to point to the new node. Finally, update the next property of node to point to the **temp**variable, which represents the original next node of **prevNode**.

**Print values of linked list**

print() {  
 let data = this.head;   
 while(data) {  
 console.log(data.value);  
 data = data.next;   
 }  
}

In this code snippet:

We traverse the list element and printing the value of the property.

**Update node at specific position**

update(value, n) {  
 let node = this.findNode(n);   
 node.value = value;   
}

In this code snippet:

First, we use the **findNode**method to locate the node at position **n**in the linked list. Then simply update the value of node with the new value.