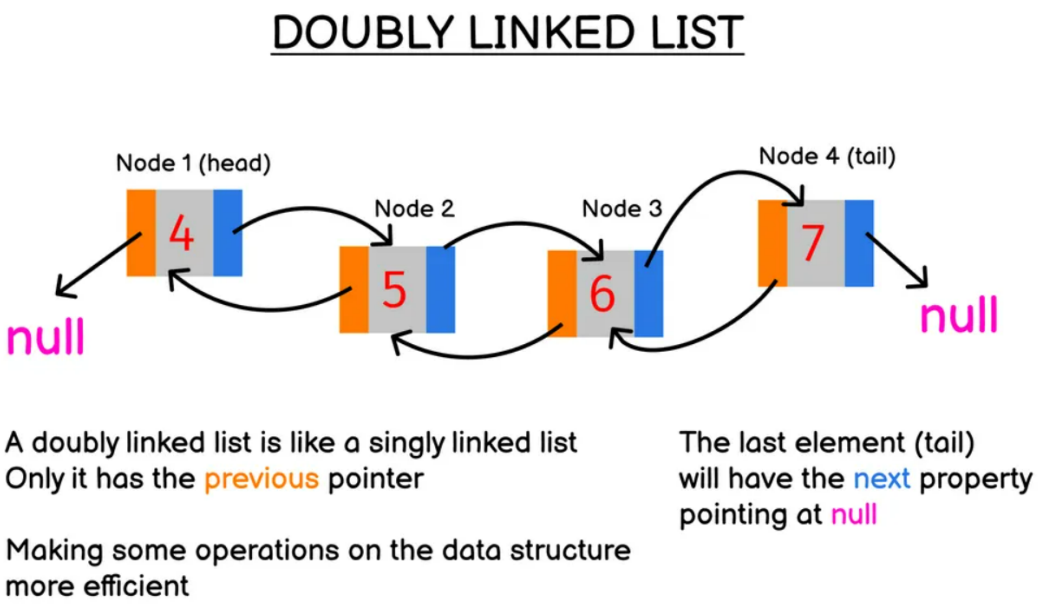
Agenda

- **What is Doubly Linked List**
- **Doubly Linked List Methods and Examples**

What is a Doubly Linked List?

- the Doubly Linked List consists of nodes
- each node has a value
- each node has a pointer to the previous node (or null at the beginning of the list)
- each node has a pointer to the next node (or null at the end of the list)
- the List has a head (= beginning)
- the List has a tail (= end)
- the List has a length (= how many nodes are in the List)
- the List has no index like an Array
- "doubly" means every node has two connections (one to the previous node and one to the next node)

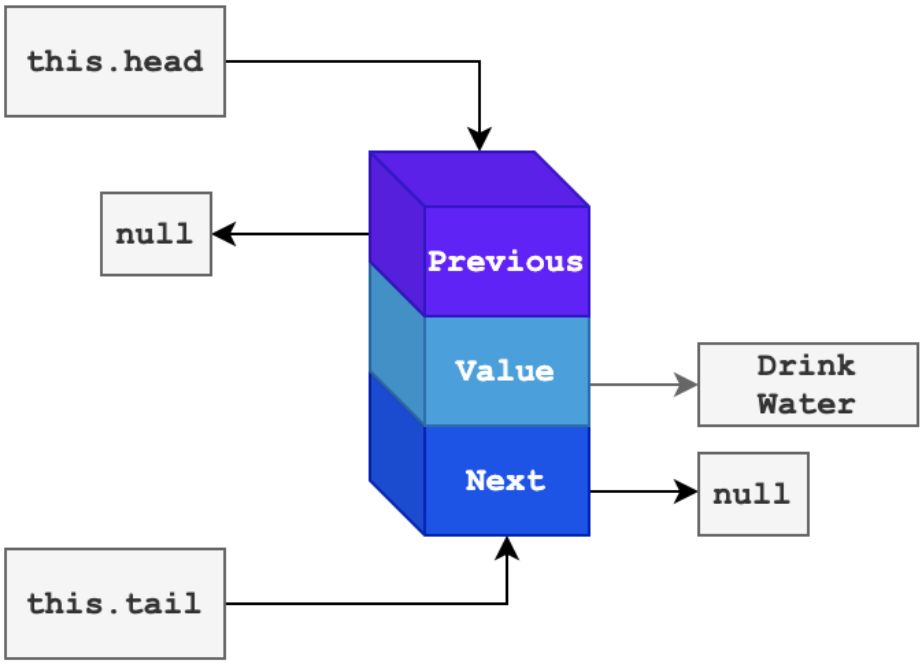


Implementing doubly linked list data structure in JavaScript

A **doubly linked list** (sometimes also called double linked list) is a type of linked list data structure that has two references in each node:

- The next reference that points to the next node
- The previous reference that points to the previous node

Just like a singly linked list, a doubly linked list also has the head and tail references that point to the first and last node in the list respectively.



Implementing a doubly linked list in javascript

We will use a `Node` object which will be storing the element and the reference to the next and the previous element.

```
function doubleLinkedList() {
  //Node
  let Node = function(element) {
    this.element = element;
    this.next = null;
    this.prev = null;
  }

  let length = 0;
  let head = null;
  let tail = null;

  //Other methods go here
}
```

Adding an item in the doubly linked list

If the `head` is empty then assign the current `Node` to the head else add the `Node` as the next element and mark the exisiting as previous element.

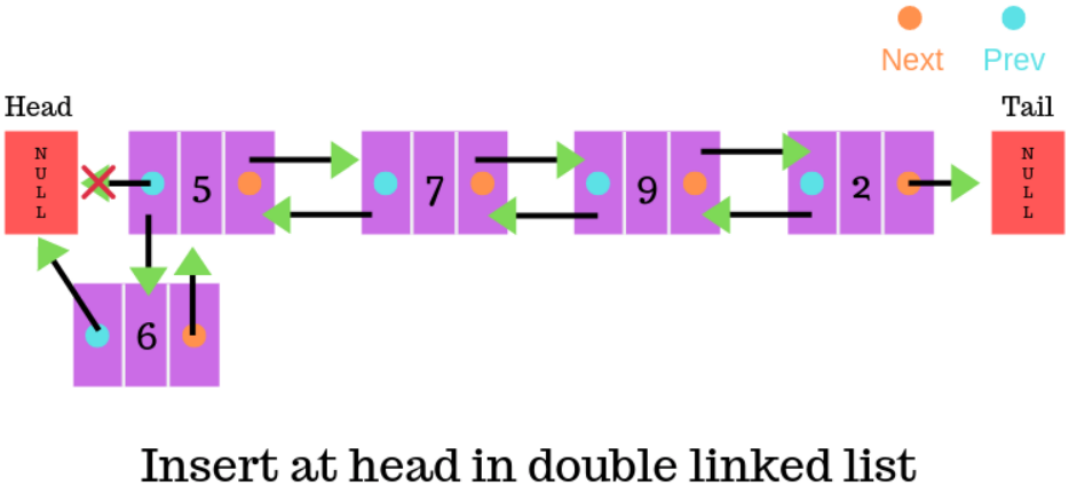
```
//Append a new element
this.append = function(element) {
    let node = new Node(element),
          current = head,
          previous;

    if(!head){
          head = node;
          tail = node;
     }else{
       node.prev = tail;
       tail.next = node;
       tail = node;
     }
```

```
    length++;
  }
```

Insert an element at the given position in doubly linked list

Insert at the beginning

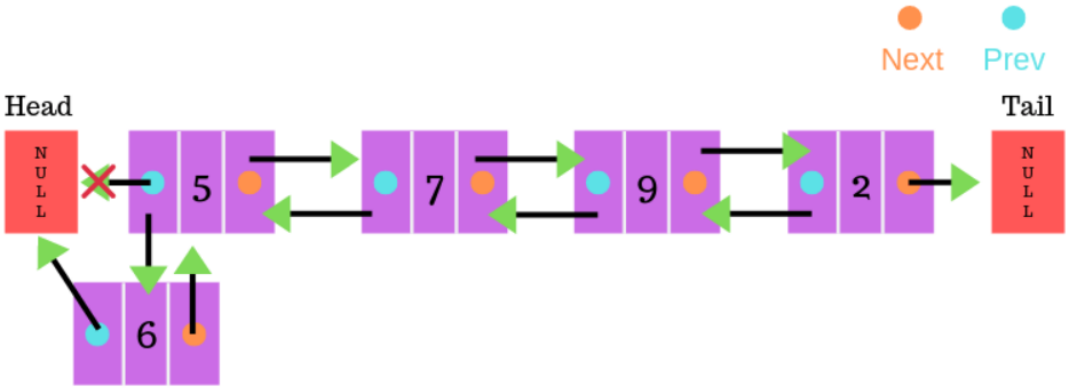
If the current element is to be added at the start then just make all the existing elements as the next element of the current element.



Insert at head in double linked list

Insert in the middle

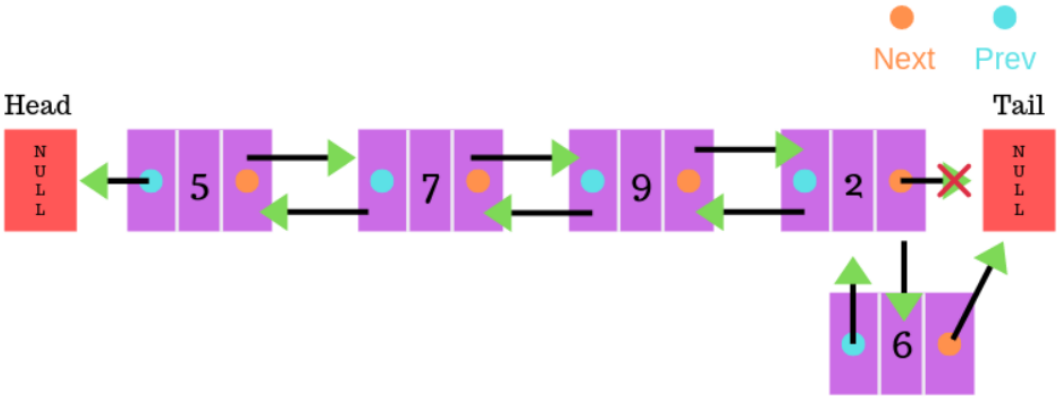
It is a little complicated step.If we are going to insert an element in the middle then we will have to make the previous element to point to the curre element and current element to point to the next element of the current element.



Insert at head in double linked list

Insert at the end

To add the element at the last then we will just have to make sure that last element points to the current element and the current points to `null` .



Insert at tail in double linked list

Code:

```
//Add element at any position
  this.insert = function(position, element) {

    //Check of out-of-bound values
    if(position >= 0 && position <= length){
      let node = new Node(element),
          current = head,
          previous,
          index = 0;

      if(position === 0){
        if(!head){
          head = node;
          tail = node;
        }else{
          node.next = current;
          current.prev = node;
          head = node;
        }
      }else if(position === length){
        current = tail;
        current.next = node;
        node.prev = current;
        tail = node;
      }else{
        while(index++ < position){
          previous = current;
          current = current.next;
        }

        node.next = current;
        previous.next = node;

        //New
        current.prev = node;
        node.prev = previous;
      }

      length++;
      return true;
    }else{
      return false;
```

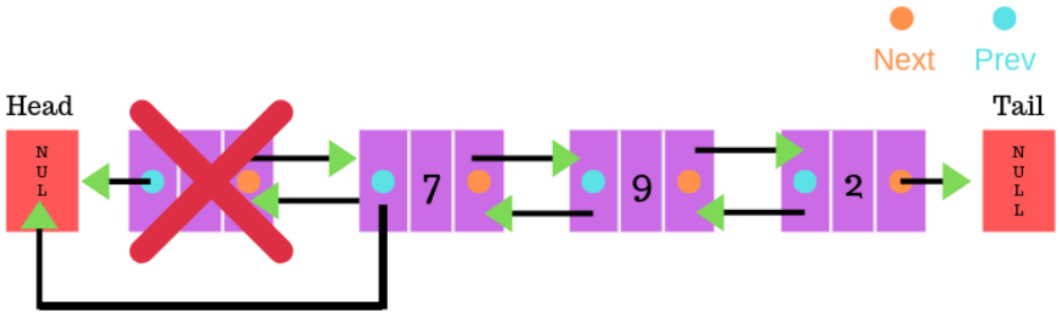
```
    }
}
```

Remove an element from the given position in doubly linked list

There are three different cases while removing an element from the doubly linked list.

Remove the first element

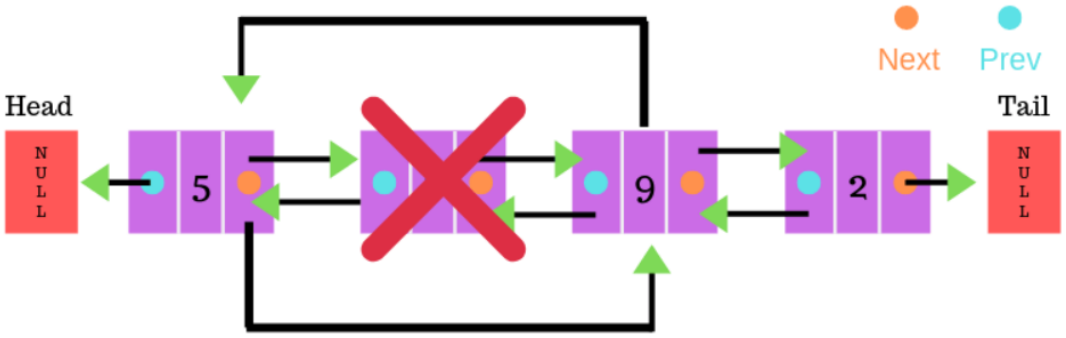
If the current element is to removed then just move the head to the very next element.



Remove head in double linked list

Remove the middle element

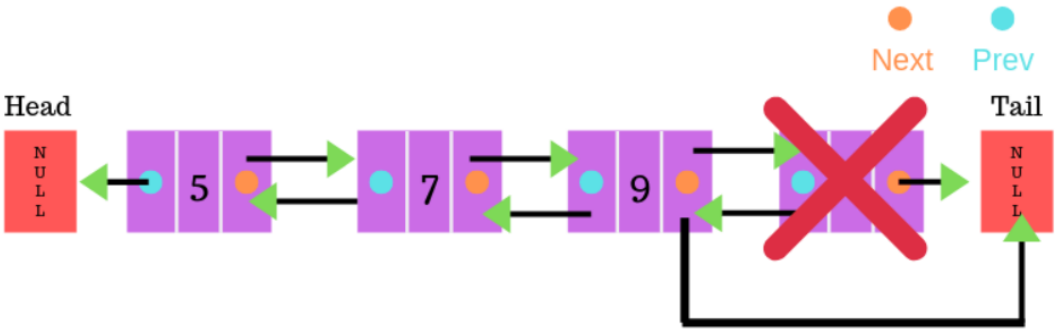
This is a little complicated process so we need to implement this carefully.If we are going to remove an element from the middle then we will have make the previous element to point to the next element and next element to point to the previous element of the current element.



Remove from middle in double linked list

Remove the last element

To remove the last element will just have to make sure previous element point to the `null` .



Remove tail in double linked list

Code:

```
//Remove element at any position
  this.removeAt = function(position){
    //look for out-of-bounds value
    if(position > -1 && position < length){
      let current = head, previous, index = 0;

      //Removing first item
      if(position === 0){
        head = current.next;

        //if there is only one item, update tail //NEW
        if(length === 1){
          tail = null;
        }else{
          head.prev = null;
        }
      }else if(position === length - 1){
        current = tail;
        tail = current.prev;
        tail.next = null;
      }else{
        while(index++ < position){
          previous = current;
          current = current.next;
        }

        //link previous with current's next - skip it
        previous.next = current.next;
        current.next.prev = previous;
      }

      length--;
      return current.element;
    }else{
      return null;
    }
  }
```

Getting index of the given element

We will be returning the zero based index for linked list elements just like arrays, If element is not present then we will return `-1` .

```
//Get the indexOf item
  this.indexOf = function(elm){
    let current = head,
    index = -1;

    //If element found then return its position
    while(current){
      if(elm === current.element){
         return ++index;
      }

       index++;
       current = current.next;
     }

    //Else return -1
    return -1;
  };
```

Check if element is present in the doubly linked list

We can store any type of data in a doubly-linked list in javascript but here we are only finding the data for String and Numeric type.

```
//Find the item in the list
  this.isPresent = (elm) => {
    return this.indexOf(elm) !== -1;
  };
```

Delete an element from the doubly linked list

To delete an element from the list we will find the index of the element and remove it.

```
//Delete an item from the list
this.delete = (elm) => {
   return this.removeAt(this.indexOf(elm));
};
```

Delete the head from the doubly linked list

To remove the head from the doubly linked list we will just have to remove the first element.

```
//Delete the first item from the list
  this.deleteHead = function(){
    this.removeAt(0);
  }
```

Delete the tail from the doubly linked list

To remove the tail from the doubly linked list we will just have to remove the last element.

```
//Delete the last item from the list
  this.deleteTail = function(){
    this.removeAt(length-1);
  }
```

Complete Code:

```
function doubleLinkedList() {
  let Node = function(element) {
    this.element = element;
```

```
  this.next = null;
  this.prev = null;
}

let length = 0;
let head = null;
let tail = null;

//Add new element
this.append = function(element) {
  let node = new Node(element),
        current = head,
        previous;

  if(!head){
        head = node;
        tail = node;
   }else{
     node.prev = tail;
     tail.next = node;
     tail = node;
   }

  length++;
}


//Add element
this.insert = function(position, element) {

  //Check of out-of-bound values
  if(position >= 0 && position <= length){
    let node = new Node(element),
        current = head,
        previous,
        index = 0;

    if(position === 0){
      if(!head){
        head = node;
        tail = node;
      }else{
        node.next = current;
        current.prev = node;
        head = node;
      }
    }else if(position === length){
      current = tail;
      current.next = node;
      node.prev = current;
      tail = node;
    }else{
      while(index++ < position){
        previous = current;
        current = current.next;
      }

      node.next = current;
      previous.next = node;

      //New
      current.prev = node;
      node.prev = previous;
    }
```

```
    length++;
    return true;
  }else{
    return false;
  }
}

//Remove element at any position
this.removeAt = function(position){
  //look for out-of-bounds value
  if(position > -1 && position < length){
    let current = head, previous, index = 0;

    //Removing first item
    if(position === 0){
      head = current.next;

      //if there is only one item, update tail //NEW
      if(length === 1){
        tail = null;
      }else{
        head.prev = null;
      }
    }else if(position === length - 1){
      current = tail;
      tail = current.prev;
      tail.next = null;
    }else{
      while(index++ < position){
        previous = current;
        current = current.next;
      }

      //link previous with current's next - skip it
      previous.next = current.next;
      current.next.prev = previous;
    }

    length--;
    return current.element;
  }else{
    return null;
  }
}

//Get the indexOf item
this.indexOf = function(elm){
  let current = head,
  index = -1;

  //If element found then return its position
  while(current){
    if(elm === current.element){
       return ++index;
    }

     index++;
     current = current.next;
   }

  //Else return -1
  return -1;
};

//Find the item in the list
```

```
this.isPresent = (elm) => {
  return this.indexOf(elm) !== -1;
};

//Delete an item from the list
this.delete = (elm) => {
   return this.removeAt(this.indexOf(elm));
};

//Delete first item from the list
this.deleteHead = function(){
  this.removeAt(0);
}

//Delete last item from the list
this.deleteTail = function(){
  this.removeAt(length-1);
}

//Print item of the string
this.toString = function(){
  let current = head,
  string = '';

  while(current){
    string += current.element + (current.next ? '\n' : '');
    current = current.next;
  }

  return string;
};

//Convert list to array
this.toArray = function(){
  let arr = [],
  current = head;

  while(current){
    arr.push(current.element);
    current = current.next;
  }

  return arr;
};

//Check if list is empty
this.isEmpty = function(){
  return length === 0;
};

//Get the size of the list
this.size = function(){
  return length;
}

//Get the head
this.getHead = function() {
  return head;
}

//Get the tail
this.getTail = function() {
  return tail;
```

```
  }
}
```

Lets have a look into another example

```
function createNode(value) {
  return {
    value: value,
    next: null,
    previous: null,
  };
}
```

Next, create the DoublyLinkedListclass as follows:

```
class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
    this.length = 0;
  }
}
```

With that, you can start writing code that will insert and print the DoublyLinkedList instance content.

Adding insert method to the doubly linked list

The code for the insert() method is as follows:

```
insert(value) {
  this.length++;
  let newNode = createNode(value);

  if (this.tail) {
    // list is not empty
    this.tail.next = newNode;
    newNode.previous = this.tail;
    this.tail = newNode;
    return newNode;
  }

  this.head = this.tail = newNode;
  return newNode;
}
```

The insert()method of the list will accept one parameter: the valuethat will be stored in the node.

Each time a new node is inserted into the list, the length property of the list will go up by one and a new node is created using the createNode() function

If the list is not empty, then the list needs to be adjusted with the following steps:

- The current tail next reference points to the newNode
- The newNode.previous refers to the current this.tail
- Then this.tail will point to the newNode

When the list is empty, then the head and tail reference points to the newNode

Next, let's create a print() method to print each node value.

Doubly linked list print method

The code for the print()method is as follows:

```
print() {
  let current = this.head;
  while (current) {
    console.log(
      `${current.previous?.value} ${current.value} ${current.next?.value}`
    );
    current = current.next;
  }
}
```

The method above will print the current node's value, along with the value of the previous and next pointer when they are available.

You can test the list implementation with the following code:

```
const dLinkedList = new DoublyLinkedList();

dLinkedList.insert(7);
dLinkedList.insert(8);
dLinkedList.insert(9);
dLinkedList.print();
```

The console output should be like this:

```
undefined 7 8
7 8 9
8 9 undefined
```

It means, the insert()and print()methods are working correctly.

Doubly Linked List Remove Method

The code for the remove()method of the list is as follows:

```
remove() {
  if (this.tail) {
    this.length--;

    const removedTail = this.tail;

    this.tail = this.tail.previous;
    if(this.tail){
      this.tail.next = null;
    } else {
      this.head = null;
    }

    return removedTail;
  }
  return undefined;
}
```

First, you need to check that the tail reference is not null. After that, you can start by decrementing the length property by one.

Next, keep a reference to the current tail so you can return it later. Then adjust the this.tail reference to point to the previous node.

Finally, check if the new tail is null or not. If the new tail is not null, cut off the previous tail by setting the this.tail.next property to null.

If the new tail is null, then adjust the this.head property to refer to null as well.

Return the removedTail once the execution is completed.

Now you have the insert() and remove() methods working, you can test to remove a node from the list instance.

Insert and remove a node from the head

The insertHead() method is a method that allows you to insert a node from the head.

The code for the method is as shown below:

```
insertHead(value) {
  this.length++;
  let newNode = createNode(value);

  if (this.head) {
    this.head.previous = newNode;
    newNode.next = this.head;
    this.head = newNode;
    return newNode;
  }

  this.head = this.tail = newNode;
  return newNode;
}
```

It's very similar to the insert() method, but instead of adjusting the tail reference, you adjust the head reference of the list.

The same goes for the removeHead() method:

```
removeHead() {
  if (this.head) {
    this.length--;
    const removedHead = this.head;
    this.head = this.head.next;

    if(this.head){
      this.head.previous = null;
    } else {
      this.tail = null;
    }

    return removedHead;
  }
  return undefined;
}
```

Now your doubly linked list implementation should be able to insert and remove a node from the head.

Insert and remove at specific index

One difference between a linked list and an array is that a linked list doesn't store the index of the nodes.

To insert and remove a node at a specific index, you need to traverse the node until you reach the node at that index.

Here's the code for inserting and removing a node at a specific index:

```
insertIndex(value, index) {
  if (index >= this.length) {
    throw new Error("Insert index out of bounds");
  }

  if (index === 0) {
    return this.insertHead(value);
  }

  this.length++;
  let currentNode = this.head;
  for (let i = 0; i < index; i++) {
    currentNode = currentNode.next;
  }
  const previousNode = currentNode.previous;
  const newNode = createNode(value);
  newNode.next = currentNode;
  newNode.previous = previousNode;
  previousNode.next = newNode;
```

```
    currentNode.previous = newNode;
    return newNode;
  }

  // remove at specific index

  removeIndex(index) {
    if (index >= this.length) {
      throw new Error("Remove index out of bounds");
    }

    if (index === 0) {
      return this.removeHead();
    }

    this.length--;
    let currentNode = this.head;
    for (let i = 0; i < index; i++) {
      currentNode = currentNode.next;
    }
    const previousNode = currentNode.previous;
    const nextNode = currentNode.next;
    previousNode.next = nextNode;
    nextNode.previous = previousNode;
    return currentNode;
  }
```

Conclusion

- A doubly linked list is just a linked list with an extra reference in each node that points to the previous value.
- Because each node knows about the previous node, you can remove nodes from the list more efficiently than a singly linked list.

Thank You !