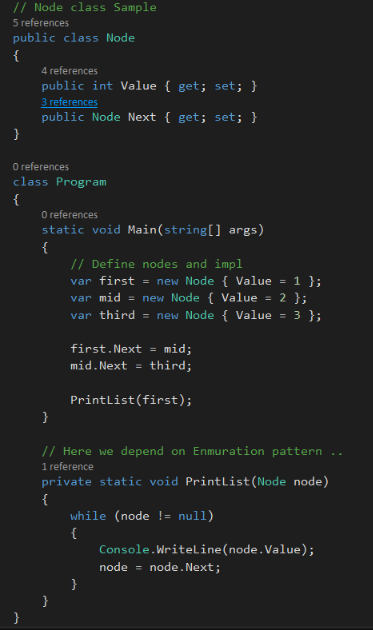
# Linked List:

## Node:

* Node: Is the most basic building block for many common data structures.
* Node fulfil 2 functions:
  + Provide a mechanism to contain piece of data.
  + It provides a means of connecting itself to other nodes via object reference pointer (called Next Pointer)
  + 
* Chain of Nodes: Nodes linked together.
* Example on chain of nodes:



## Linked List:

### It’s a single chain of nodes contains:

#### Head Pointer (Well defined starting point)

#### Tail Pointer (Well defined ending point)

#### Operations (This operations make the list truly useful as a collection):

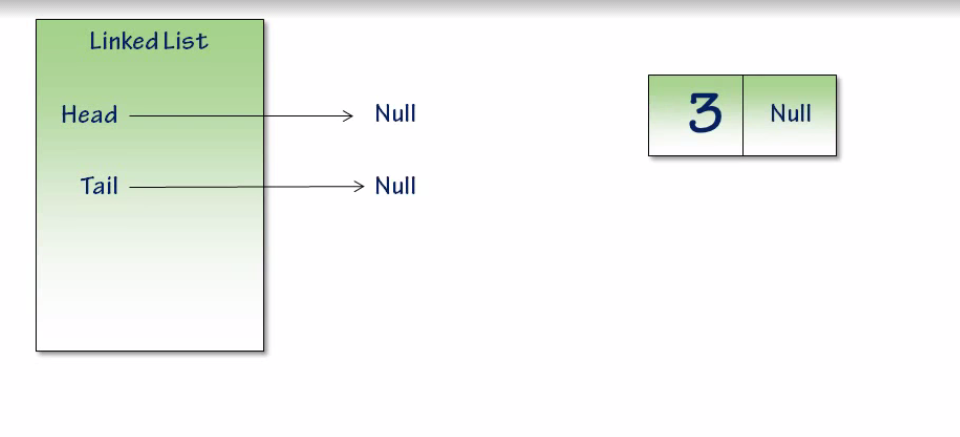
* Add
* Remove
* Find
* Enumerate

### Add Operation:

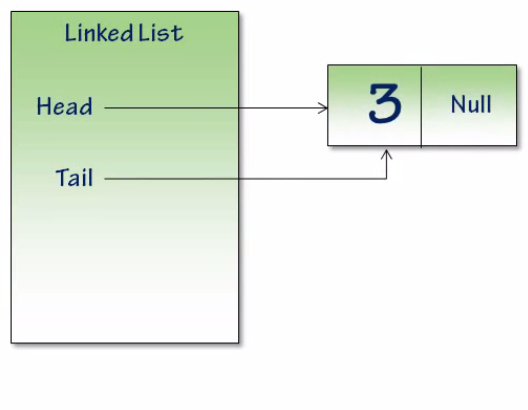
Adding an item to list is the most basic operation to perform.

#### **Add item to front:**

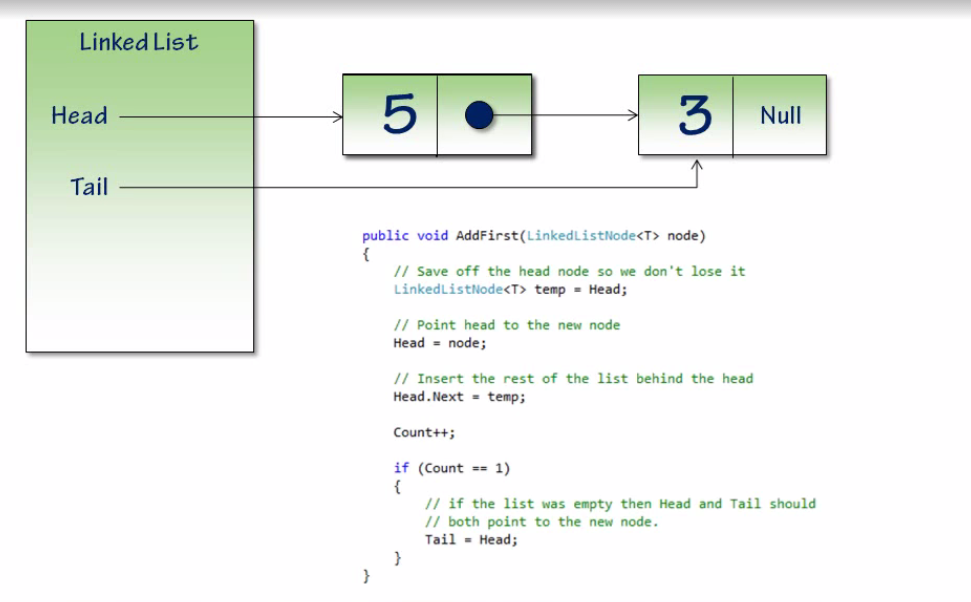
1- If the linked list is empty it will look like the below and say we want to add a node illustrated below.



2- So we will be pointing the head and tail of the linked list to this node.



3- And adding another node to front will be also a straightforward way like below

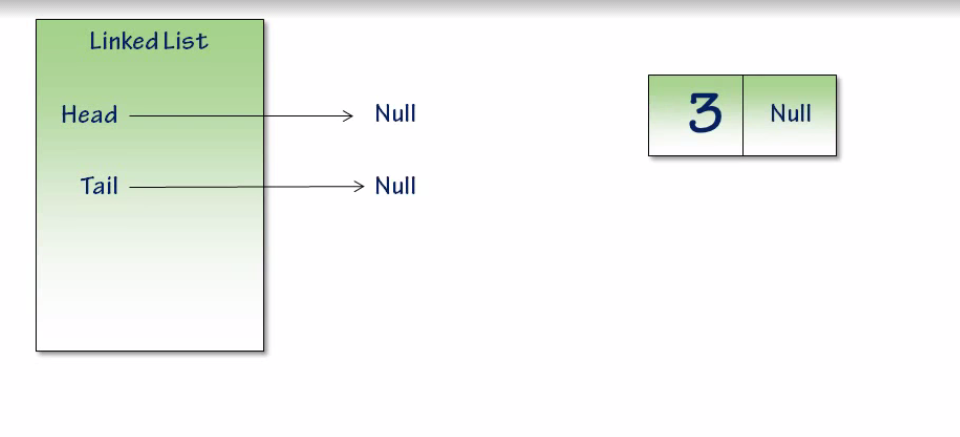


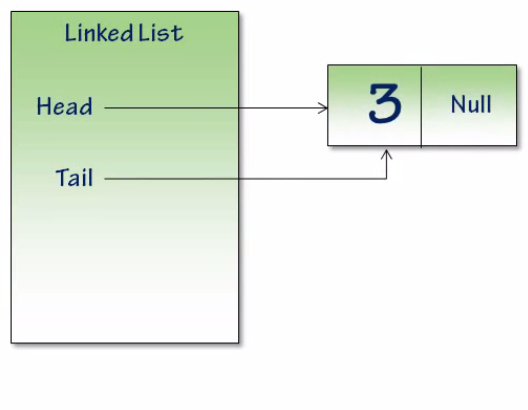
4- Insertion to front is a very efficient process with O(1) efficiency.

5- Note: If we are using an array instead of linked list, adding to the front will involve shifting all the existing data to the right and if the array were full it will involving allocating an entirely new larger array, and copying all of the data from the smaller array to larger one.

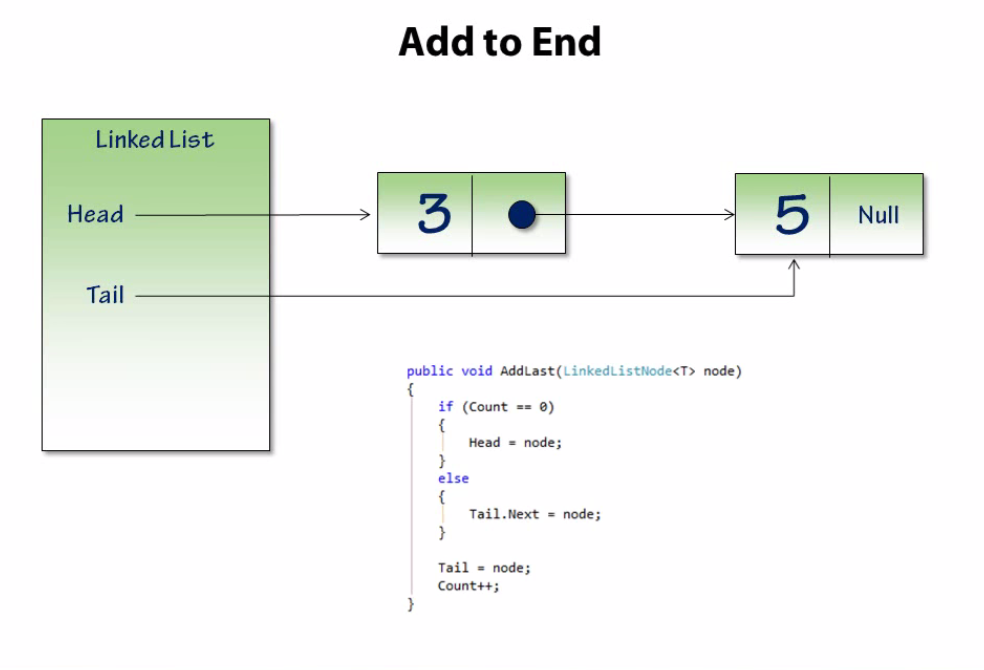
#### **Add item to last:**

1- If the linked list is empty it will look like adding first node as shown in adding to front.





2- So Add to the end will be simple as below

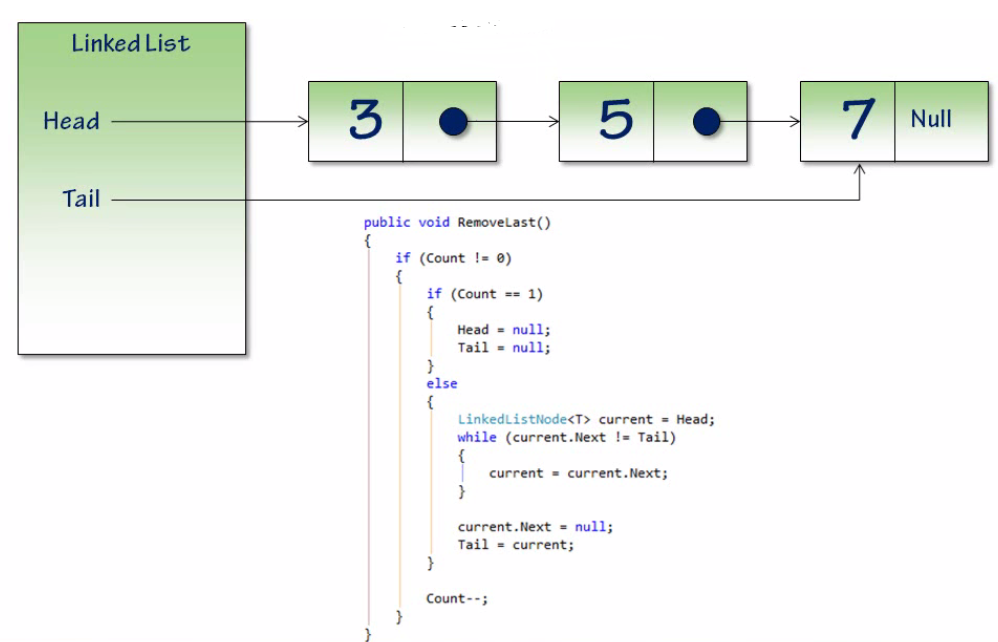


### Remove Operation:

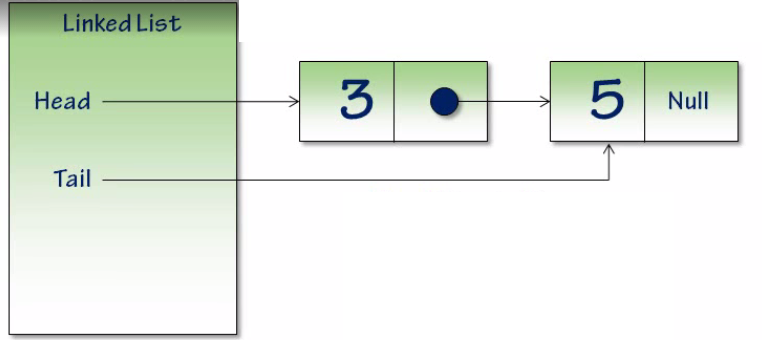
Like add, remove can be performed from the front or end of the list.

#### **Remove item from last:**

1. Removing item from last is an uncomplicated way but it a little bit difficult than removing item from the front.
2. As shown below, removing the tail node [7] requires updating the tail pointer and setting the next value for 5 to null.
3. The only weakness of removing node from the last when working with linked list is that the operation requires enumeration over the list to reach the second to last node to update it by setting it’s Next to null and made Tail reference it. this enumeration is because we only storing references to the tail and head nodes (Doubly linked list solved this issue)



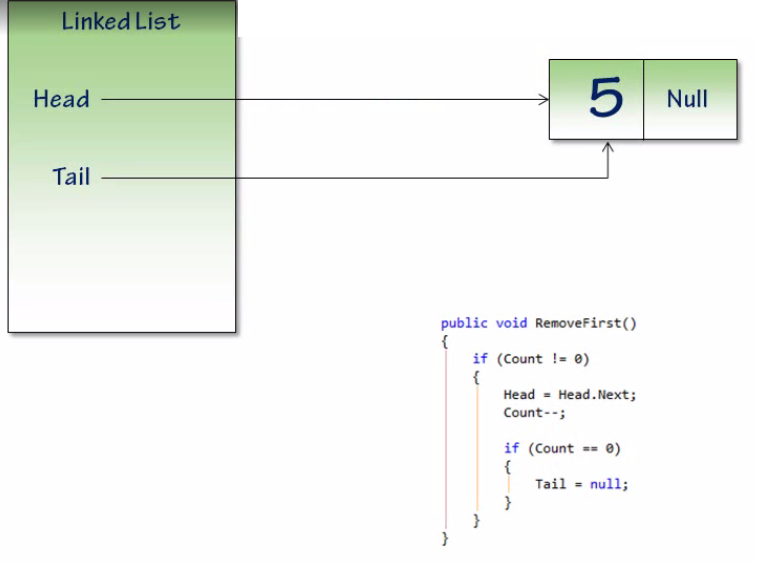
1. And after removing the node the figure will look like the below:



1. So, removing a one node from the last of a linked list is an operation that can impact badly on your app performance if the linked list contains a lot of nodes, because it requires enumerating over Count -2 of the list with O(n) complexity.

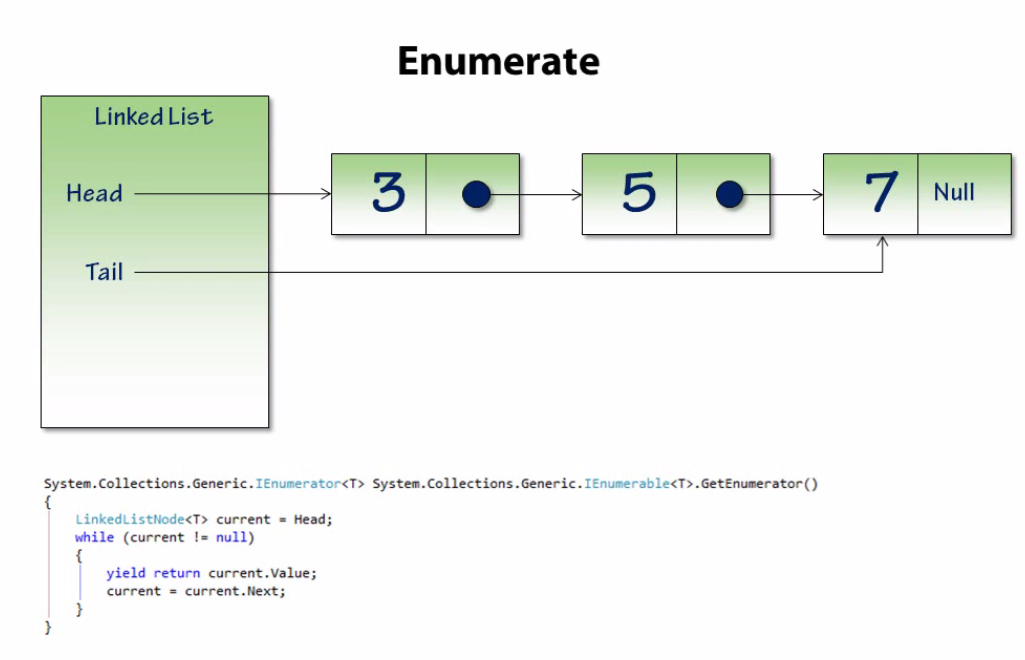
#### **Remove item from front:**

1. It’s a straightforward way like the below:



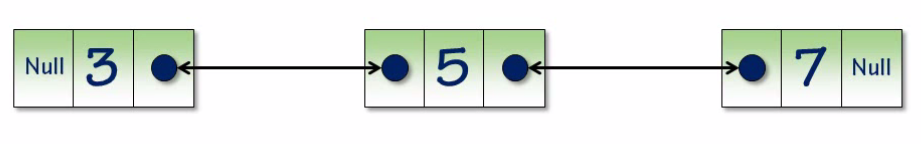
#### **Enumeration over linked list:**

* We want the caller to write foreach loop that enumerate over each of this values so the best implementation will be like below:



## Doubly Linked List:

### It’s a specialization from linked list where each node contains 2 pointers one to the next node and other to the previous node:



### It’s contains of:

#### Head Pointer (Well defined starting point)

#### Tail Pointer (Well defined ending point)

#### Operations (This operations make the list truly useful as a collection):

* Add
* Remove
* Find
* Enumerate

## Single linked list vs. Doubly linked list:

It all comes down to usage. There's a tradeoff here.

Singly linked list is simpler in terms of implementation, and typically has a smaller memory requirement as it only needs to keep the forward member referencing in place.

Doubly linked list has more efficient iteration, especially if you need to ever iterate in reverse (which is horribly inefficient with a single linked list), and more efficient deletion of specific nodes.

That being said - since you have this tagged .NET, double linked lists also have the advantage of being directly in the framework in the form of the [LinkedList<T>](http://msdn.microsoft.com/en-us/library/he2s3bh7.aspx) class. This provides a huge advantage in that you don't have to implement, test, and maintain your own collection class.