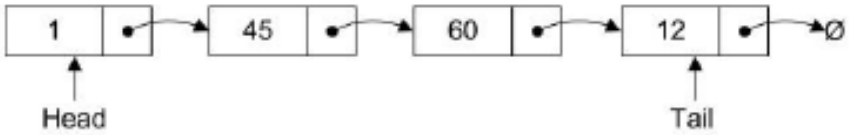
LINKED LISTS

What are Linked Lists?

- In object oriented programming,linked lists can be thought of as a series of nodes
- Each node has a pointer to the next node
- Has a head node pointing to the first node on the list
- Could have a tail node pointing to the last node on the list
- Tail node's next property would always point to null



Why Linked Lists?
*One major drawback of **arrays** is that a **fixed amount of storage** remains allocated even when only a small fraction of this is being used. Moreover, the amount of storage is **fixed by the declaration**, thus the possibility of an **overflow** (a case of more data need to be stored in a fixed number of storage in the array). An alternative data structure that specifically addresses these two problems is the **linked list**. (CMSC 204 Module)*

Discuss:
What are some programs you've written which have made use of linked lists? What were the usual errors encountered? Do you prefer arrays or linked lists?

Linked List in C#

Take Note:
After finishing this module, you should be able to:

1. *Design algorithms for searching, deleting and inserting new elements into the linked list structure; and*
2. *Solve programming problems involving the use of linked lists.*

Watch These:
Video 1 (5:48):
http://youtu.be/lcNL_HLpcEs
Video 2 (12:03):
http://youtu.be/3svB0kM6f10

Explore:
https://msdn.microsoft.com/en-us/library/he2s3bh7(v=vs.110).aspx

- Getting Started

```
//Create a node class to represent a node on our list

public class Node {
public Student Data { get; set; }
public Node Next; //default to null
}

//For the data, we implemented a Student class which could store
student information:

public class Student {
public string Name { get; set; }
public int Age { get; set; }
public string Nickname { get; set; }
public string MobileNumber { get; set; }
}
```

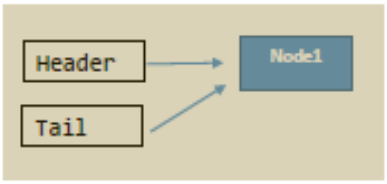
```
//Now that we have the basic classes in place, let's define our
LinkedList class

public class LinkedList {
protected Node Header { get; set; }
protected Node Tail { get; set; }
     //Class Constructor
     public LinkedList() {
     Header = null;
     Tail = null;
     }
}

//The header and the tail should initially be null, we'll add
content later
```

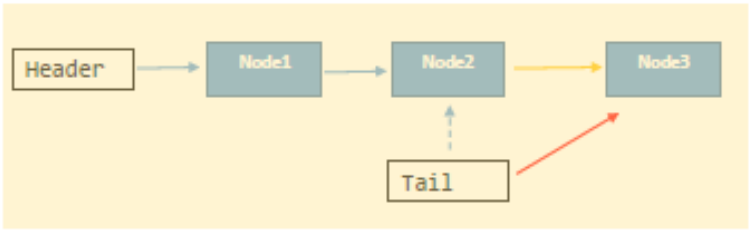
- Inserting
  - Can insert either at the head, midway, or end of the list
  - In our implementation here, we can easily insert at the end because we have **Tail node** pointing to the last element on the list

Question:
Can we remove Tail node and still have user add a new node at the end of the list? What are the implications of this tail-less pointer approach?

  - To insert a node provided we have a head and tail pointers, consider the following:
    - If inserting for first time, head and tail pointers should both refer to the same new node object
    - Otherwise, add new node to the end of the tail node and make tail pointer point to the newly added node

```
public void InsertEnd(Node newNode)
{
    //If inserting for first time, head and tail pointers
    //should both refer to the same new node object
    if(Header == null)
    {
        Header = newNode;
        Tail = newNode;
    }
    //add new node to the end of the tail node and make
    //tail pointer point to the newly added node
    else
    {
        Tail.Next = newNode; //yellow arrow
        Tail = newNode; //red arrow
    }
}
```





- Searching
  - Have to go through each nodes in the list until current node is the one being searched
  - If the Next pointer of current node is null, or if current node is the same as the tail node, this means we have reached the end of our list and traversal should stop!
  - Return null if node being searched is not found
  - **Best case** – node being searched is the head node
  - **Worst case** – node being searched is tail node, or, node being searched is not on list

```
public Node Search(string studentName) {
Node current = Header;
while((current != null) && (current.Data.Name != studentName)) {
current = current.Next;
}
//this would return null if node being searched is not found,
// otherwise, return found node
return current;
}
```

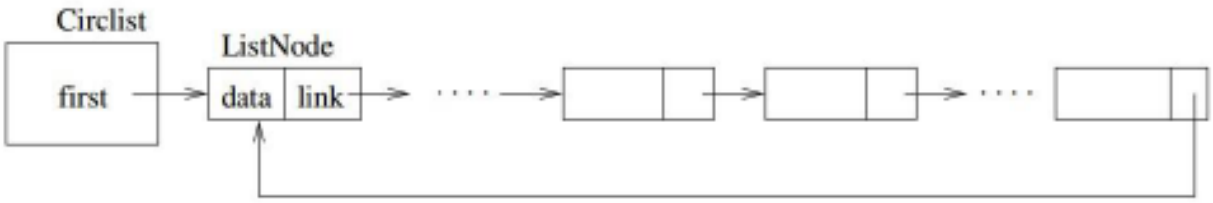
- Deleting
  - Change the Next property of the node that points to the removed Node to point to the node after the removed node.
  - There are cases that you have to account for when deleting:
    - The list is empty;
    - The node to remove is the only node in the linked list;
    - We are removing the head node;
    - We are removing the tail node;
    - The node to remove is somewhere in between the head and tail;
    - The item to remove doesn't exist in the linked list

Can you implement your own Delete() method provided the points above?

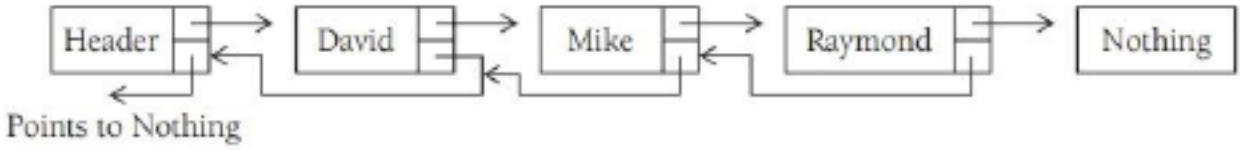
Linked Lists Visualization
Check this out: http://visualgo.net/list.html
Click "Linked List" and "Doubly Linked List". See how the different operations are performed.

Flavors of Linked Lists

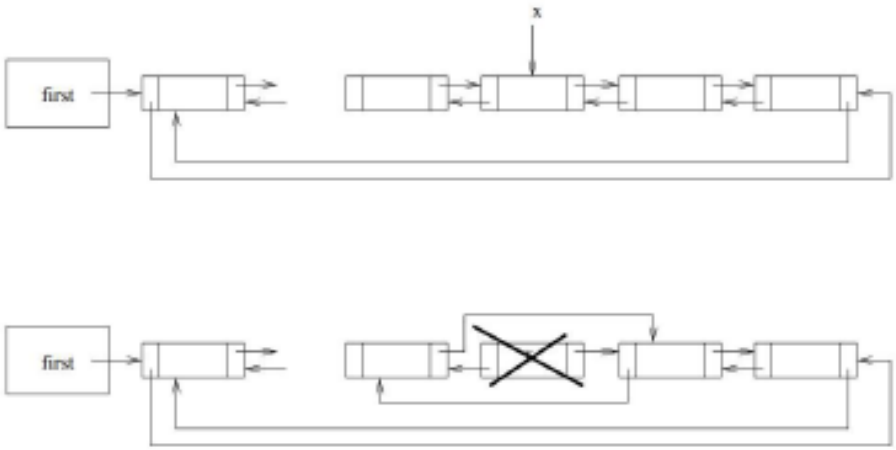
- **Circular Linked List**
  - Almost the same as regular linear linked list, only difference is last node on the list (tail) is pointing to the first element on the list (head)



- **Doubly Linked List**
  - Almost the same as singly linked list, only difference is each node has a reference to both the next and previous nodes in the list
  - We achieve this by adding a new pointer property on the LinkedList class to point to the previous node
  - Doubly linked list makes backward traversal very easy
  - *Can you think of other benefits of doubly linked list vs single implementation?*



- **Circular Doubly Linked List**
  - Circular linked list + doubly linked list, i.e., each node has pointers to previous and next nodes on the list and the last (tail) and first (head) nodes point to each other.
  - Consider the following problem: we are provided a pointer x to a node in a list and are required to delete it as shown in the figure below:



  - To accomplish this, one needs to have a pointer to the previous node. In a chain or a circular list, an expensive list traversal is required to gain access to this previous node. However, this can be done in O(1) time in a doubly linked circular list.

.NET Implementation of Linked List

- In the .NET framework, array implementation is abstracted by the LinkedList class
- The .NET library provides methods for adding, searching, and deleting arrays so you don't have to manually implement them!
- Check out the MSDN documentation on the LinkedList class to see the methods and properties exposed by .NET's LinkedList class

Takeaway Thoughts and Questions:

1. *What new knowledge about linked lists is the most interesting for you?*
2. *Give an example of a computer program/problem which will make use of a linked list. What kind of linked list is needed for this program?*