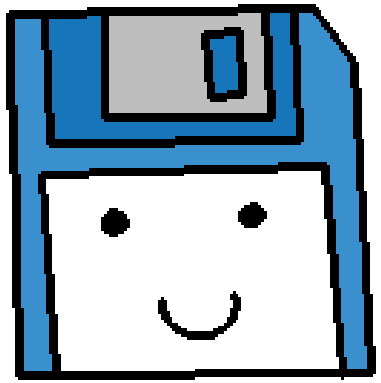
Implementing Data Structures

Written by Rachel J. Morris, last updated 2016-04-24

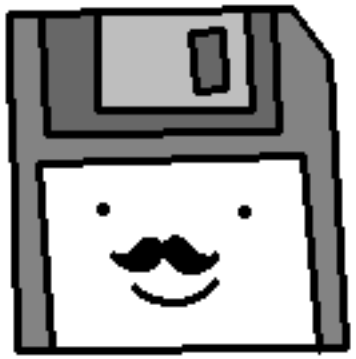
Guess we'll reinvent the wheel



Data Structures

Let's cover how to implement some basic data structures!

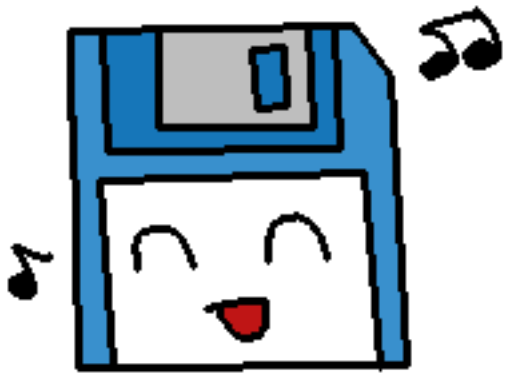
You'll end up doing this and more in a Data Structures class, but it is good to get acquainted with it!



Storing Data

Up until now, we've worked with static arrays, whose sizes must be known at compile time, and dynamic arrays, which can be any size, but are difficult to resize.

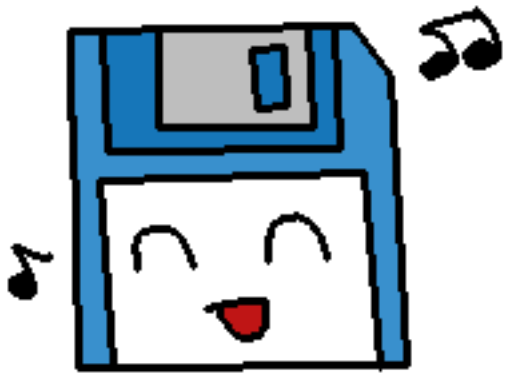
Working with dynamic arrays means that any time we want to resize it, we have to make a new array and copy the data over, which could be slow if you have a lot of complex objects in the array.



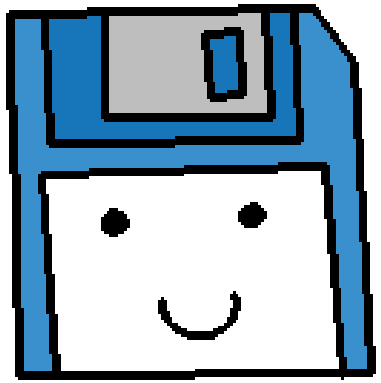
Storing Data

We can write data structures that allow us to add new elements to our “list”, without a complex resize process.

Through utilizing pointers and dynamic variables, we can use a pointer to point to the next element of a list, so we no longer need to rely on a contiguous sequence of memory “blocks”.



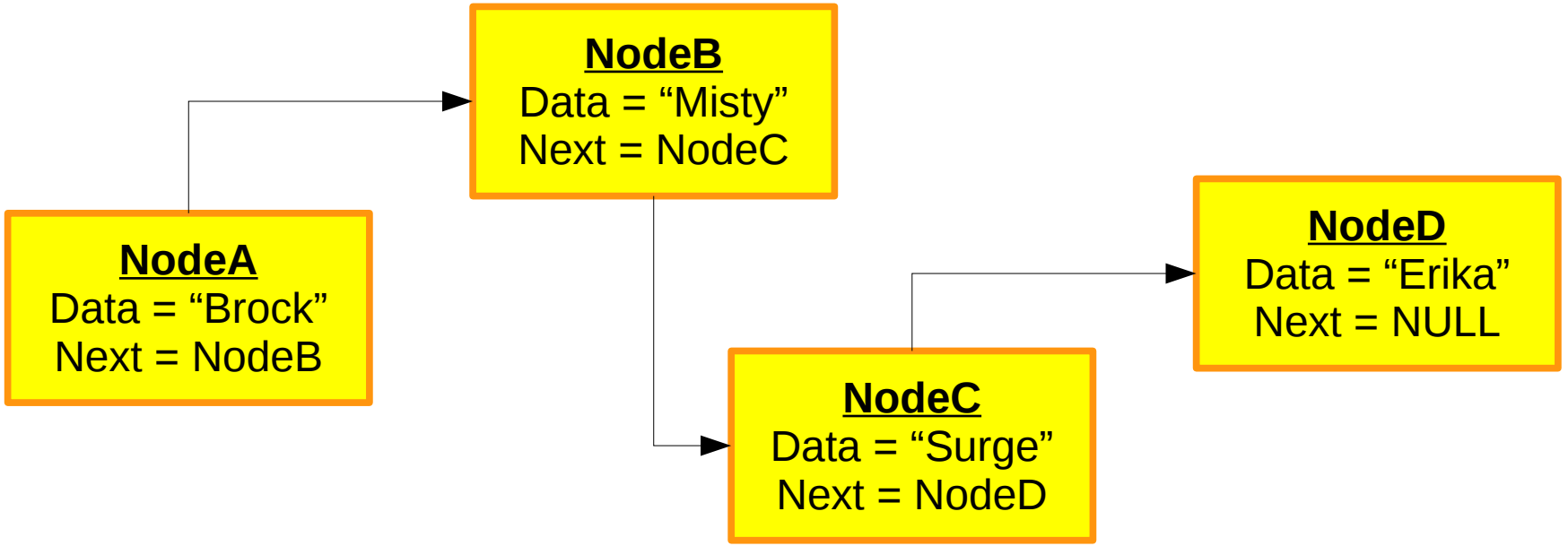
Linked List



Linked List

With a Linked List, we will need a List class, and a Node class.

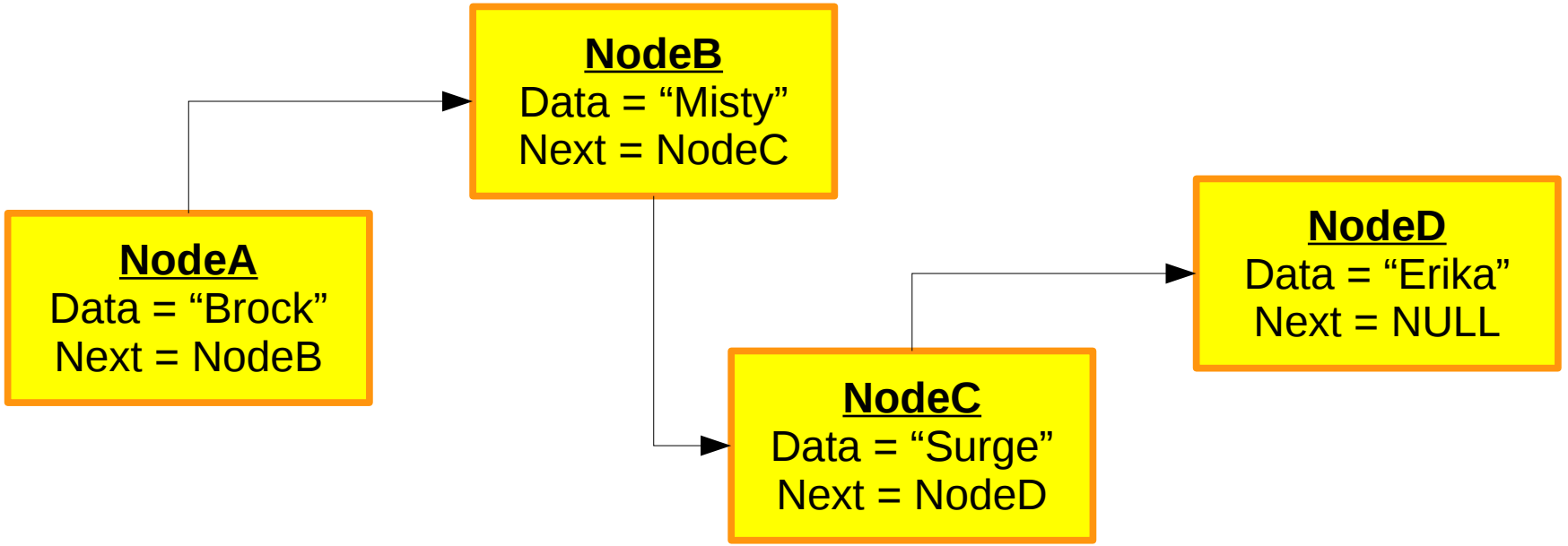
The Node class will be each element of the list – it will store the data, as well as the pointer to the next object.



Linked List

Any time we need a new Node, we simply create one, and point to it.

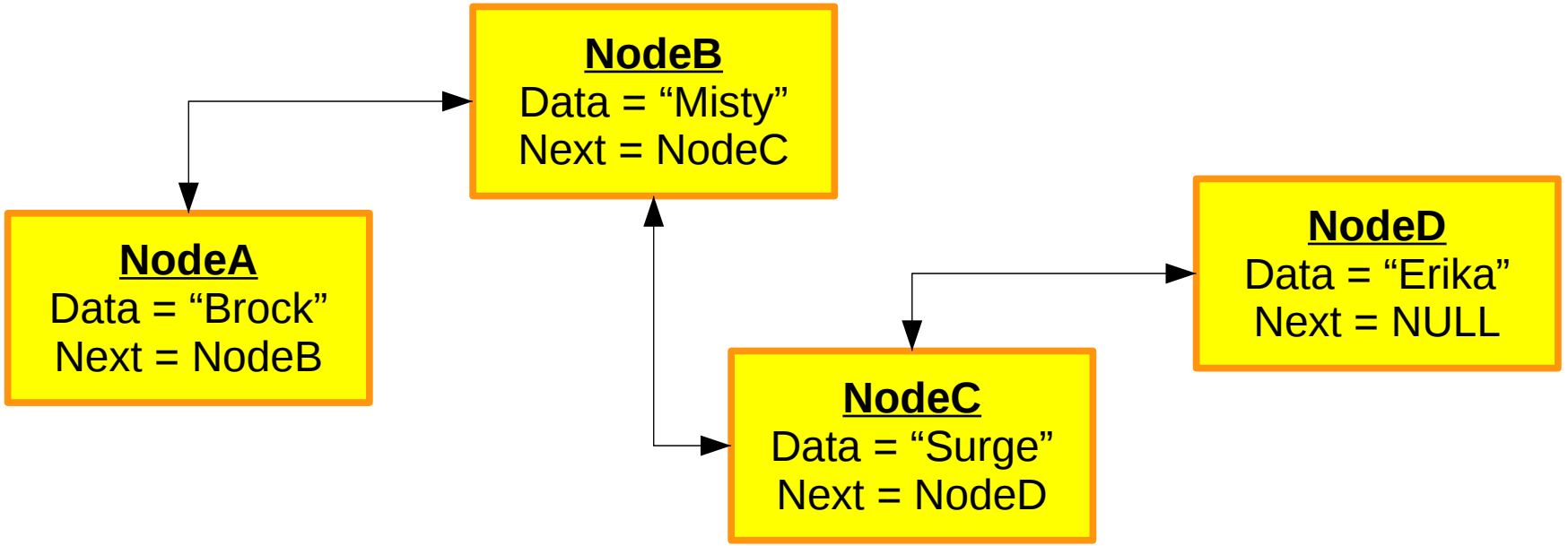
```
Node* newNode = new Node;
newNode->data = "Misty";
lastNode->next = newNode;
```



Linked List

You could also design the nodes to point to the **previous** Node as well

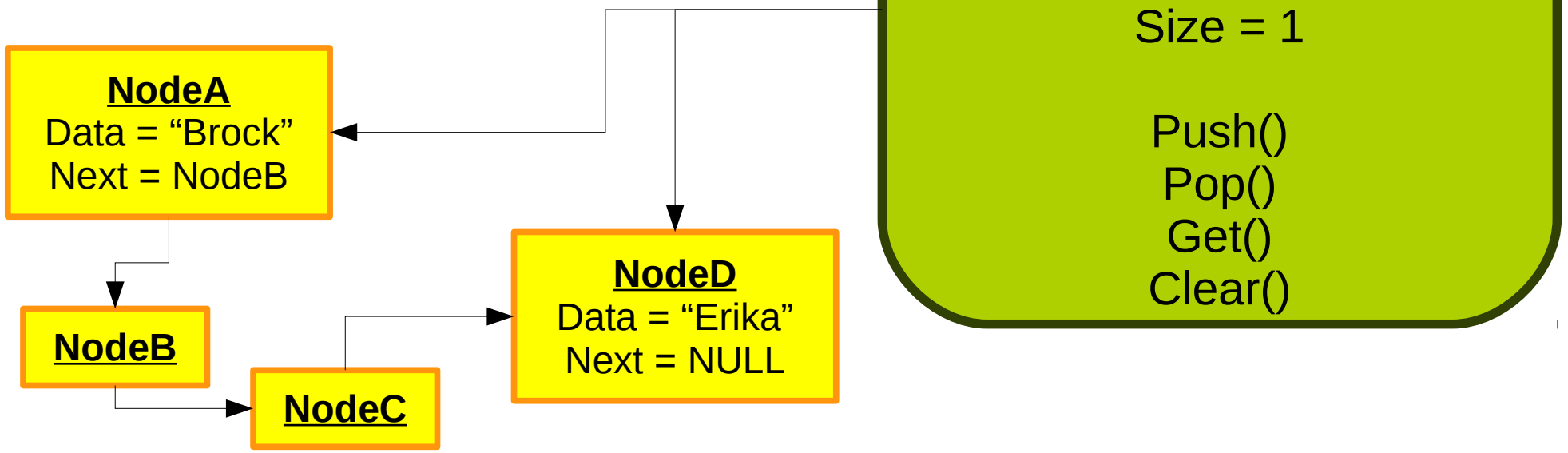
```
Node* newNode = new Node;
newNode->data = "Misty";
newNode->previous = lastNode;
lastNode->next = newNode;
```



Linked List

The **List** object will contain a pointer to the first node, and perhaps the last node.

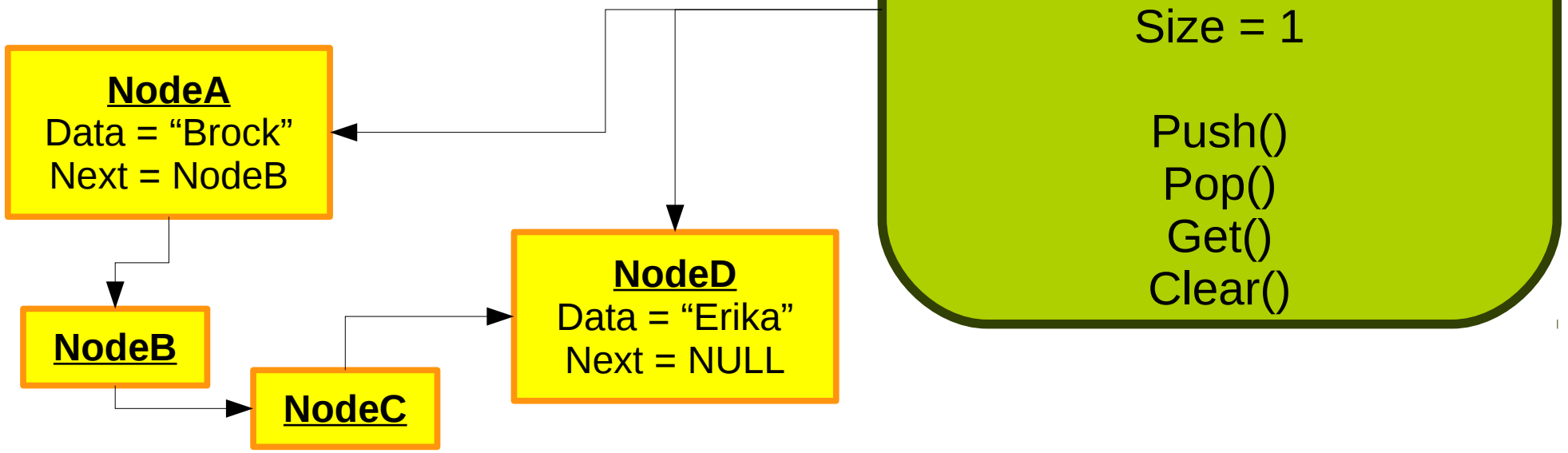
As well as any functions needed to add items to the list, remove items, and access items.



Linked List

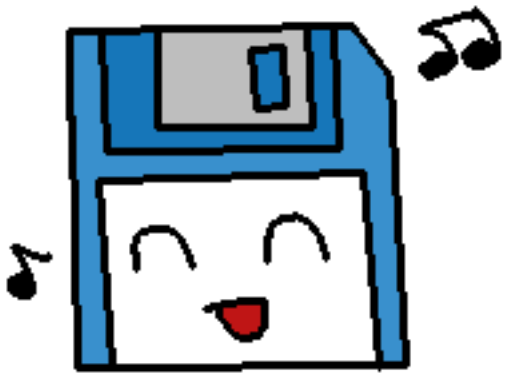
The list does not keep track of every element – at minimum, just the first one, but perhaps also the last one.

To get to other elements, you would have to **traverse** the list, starting at the first Node and going to each Node's **next**.

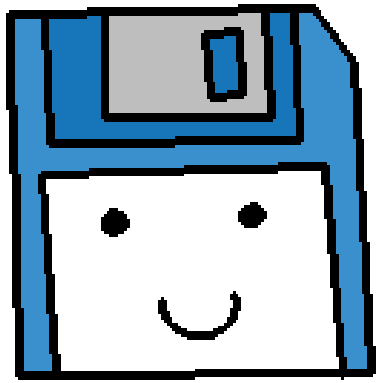


Linked List

Let's implement a Linked List!



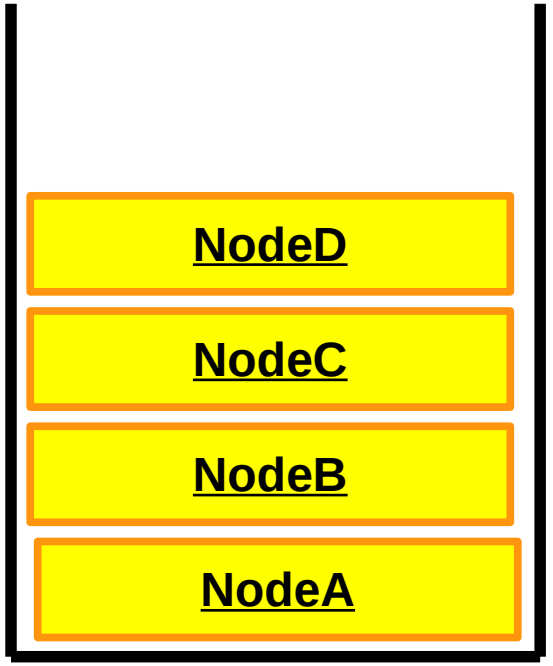
Stack



Stacks

A stack is a type of data structure that restricts how new items are added to it, and how items are accessed and removed from it.

It is known as **last-in, first-out (LIFO)**, because the first item pushed into a stack will be the last item to be removed. Likewise, the most recent item added to the stack will be the first item to be removed.



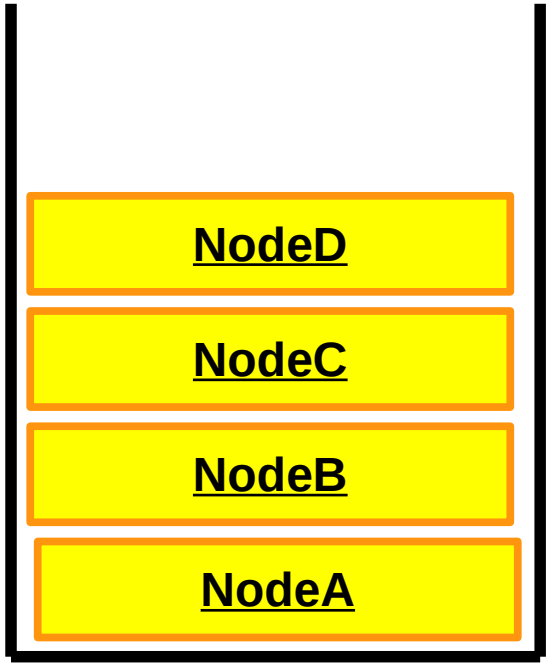
Stacks

You can implement a stack with a simple array,
or on top of a Linked List.

To add a new item to the Stack, we will have a function called **Push**.

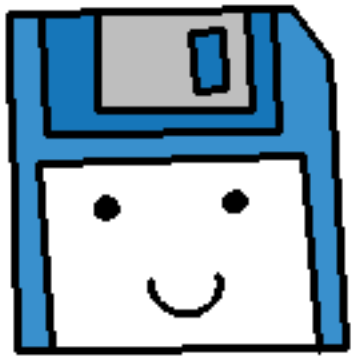
To access the top-most item from the Stack, we will have a function called **Top**.

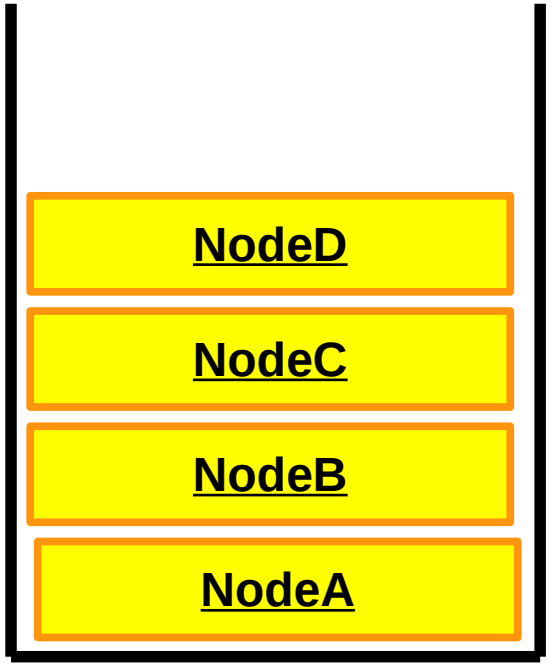
To remove the top-most item from the Stack, we will have a function called **Pop**.



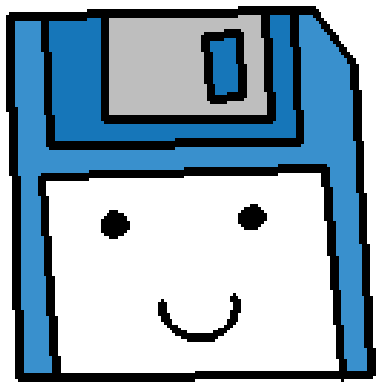
Stacks

Let's modify our Linked List
and turn it into a Stack!



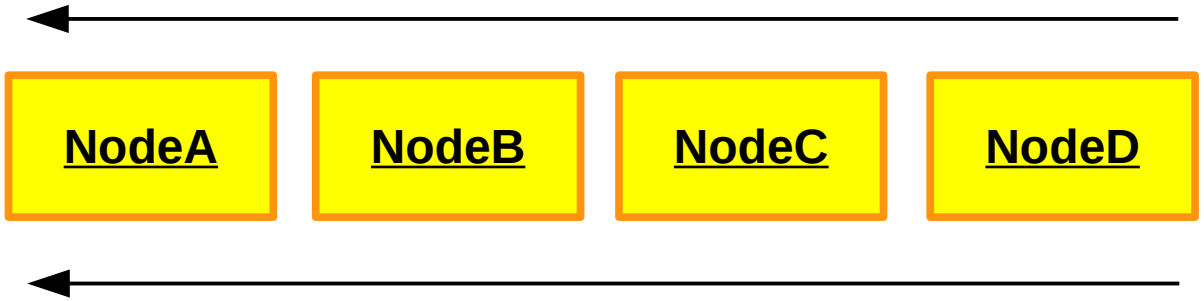


Queue



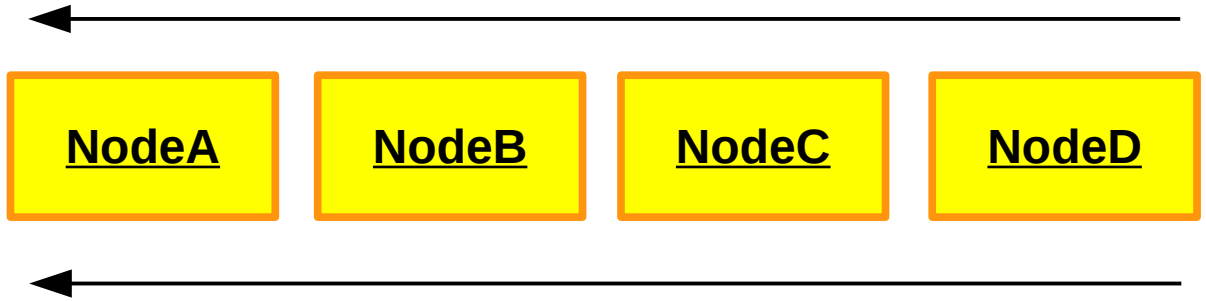
Queues

A queue is another type of data structure that restricts how new items are added to it, how items are accessed, and how items are removed from it.



It is known as **first-in, first-out (FIFO)**. It is like queuing up in a line at the store – the first item into the list is also the first to be removed from the list.

Queues

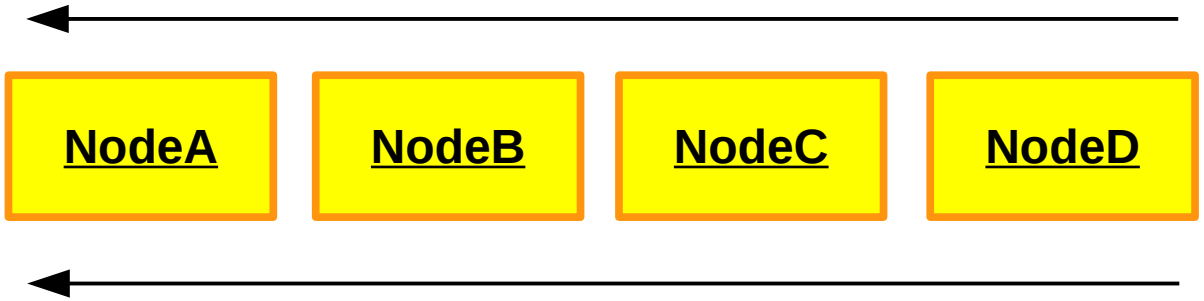


We will have a **push** function to add a new item to the end of the structure,

To get the item at the front of the list, we use the function **front**.

And to remove the item at the front of the list, we use the function **pop**.

Queues



Let's modify the Linked List to make a Queue!

Data Structures

It is good to know how these data structures work.

However, the Standard Template Library has implementations of a lot of the structures that you would need to use.

It is good to practice writing your own data structures, but if you're working on a project, I'd suggest just going with the STL.