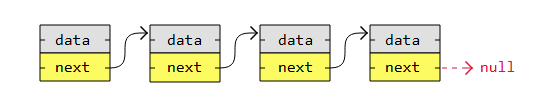
A **Linked List** is, as the word implies, a list where the nodes are linked together. Each node contains

data and a pointer. The way they are linked together is that each node points to where in the

memory the next node is placed.

Linked Lists

A linked list consists of nodes with some sort of data, and a pointer, or link, to the next node.



|  |  |  |
| --- | --- | --- |
|  | **Arrays** | **Linked Lists** |
| *An existing data structure in the programming language* | Yes | No |
| *Fixed size in memory* | Yes | No |
| *Elements, or nodes, are stored right after each other in memory (contiguously)* | Yes | No |
| *Memory usage is low (each node only contains data, no links to other nodes)* | Yes | No |
| *Elements, or nodes, can be accessed directly (random access)* | Yes | No |
| *Elements, or nodes, can be inserted or deleted in constant time, no shifting operations in memory needed.* | No | Yes |

These are some key linked list properties, compared to arrays:

* Linked lists are not allocated to a fixed size in memory like arrays are, so linked lists do not require to move the whole list into a larger memory space when the fixed memory space fills up, like arrays must.
* Linked list nodes are not laid out one right after the other in memory (contiguously), so linked list nodes do not have to be shifted up or down in memory when nodes are inserted or deleted.
* Linked list nodes require more memory to store one or more links to other nodes. Array elements do not require that much memory, because array elements do not contain links to other elements.
* Linked list operations are usually harder to program and require more lines than similar array operations, because programming languages have better built in support for arrays.
* We must traverse a linked list to find a node at a specific position, but with arrays we can access an element directly by writing myArray[5].

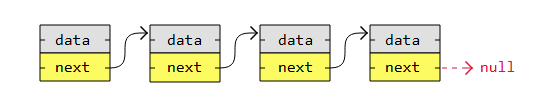
Types of Linked Lists

There are three basic forms of linked lists:

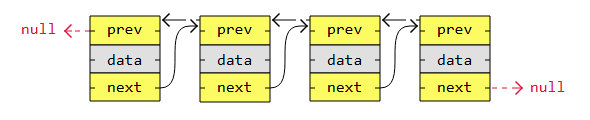
1. Singly linked lists
2. Doubly linked lists
3. Circular linked lists

A **singly linked list** is the simplest kind of linked lists. It takes up less space in memory because each

node has only one address to the next node, like in the image below.



A **doubly linked list** has nodes with addresses to both the previous and the next node, like in the image below, and therefore takes up more memory. But doubly linked lists are good if you want to be able to move both up and down in the list.

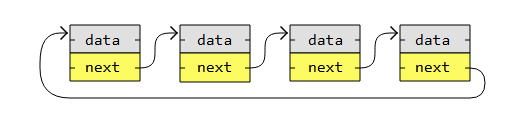


A **circular linked list** is like a singly or doubly linked list with the first node, the "head", and the last node, the "tail", connected.

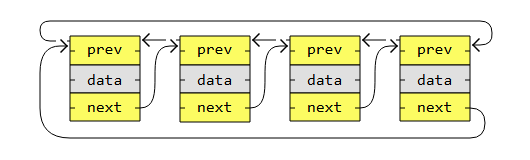
In singly or doubly linked lists, we can find the start and end of a list by just checking if the links are null. But for circular linked lists, more complex code is needed to explicitly check for start and end nodes in certain applications.

Circular linked lists are good for lists you need to cycle through continuously.

The image below is an example of a singly circular linked list:



he image below is an example of a doubly circular linked list:



Linked List Operations

Basic things we can do with linked lists are:

1. Traversal
2. Remove a node
3. Insert a node
4. Sort