Linked Lists

What is a linked list?

A **linked list** is a **dynamic data structure** that store item in **nodes**.

- Can dynamically add new items

Arrays can mimic this but they essentially create copies of themselves with new dimensions when resizing

- This has implications for both memory and processor usage

Each **node** has a **pointer** which points to the **address** of the **next node**

Linked List Operations

- **Adding a new node at the end**
  The pointer of the last node will have to point to the newly created node

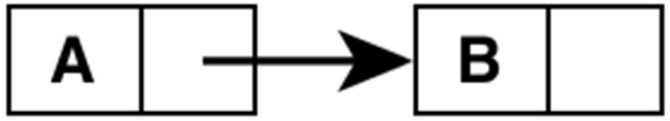
- **Inserting between existing nodes**
  When a new node is added then the pointer of the previous node has to be amended to point to the new node and the new node has to point to the node that was previously next in the list

- **Deleting nodes**
  The pointer of the previous node before the item to be deleted will be amended to point to the next item in the list

- **Traversing lists**
  Each node has to be visited and the pointer followed to the address of the next node

Examples of Linked List operations

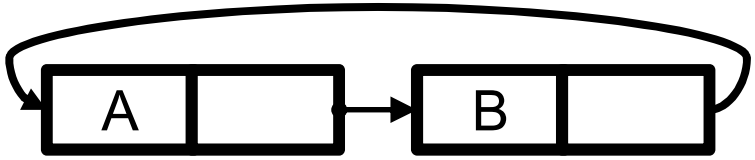
Add new Node (C) to End of List:
Set pointer for Node B to location of Node C



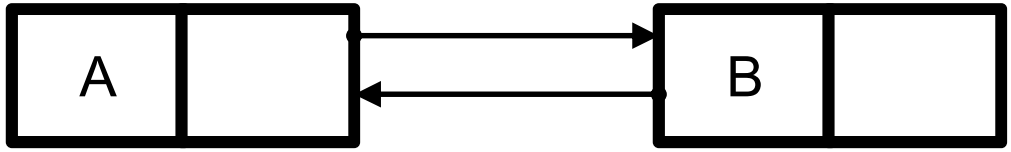
Types of Linked Lists

Single - Nodes have a data item as well as a pointer to the next field. The last node may contain a NULL pointer

Circular- This is when the last node will point to the first node in the list



Double/Doubly - This is when each node will have a pointer to the next AND previous nodes.



Advantages and Disadvantages vs Arrays

Advantages

- ✓ **Linked lists** are much more **efficient** in terms of **memory** and **processor usage** particularly when adding new items (as they don't need to resize)
- ✓ Inserting, deleting and re-ordering only requires pointers to be changed

Disadvantages

- ✗ Cannot **index** a **particular node** as in an array, you have to **traverse the linked list** in order to reach a particular node (sequential vs direct)
- ✗ Requires storing of additional pointers
- ✗ Nodes may not be stored contiguously