**Linked Lists: (A-Level)**

A linked list is a **dynamic data structure**, which is used **to store an ordered sequence**, as described below:

* The items which form the sequence are **not necessarily in contiguous** (next or together in sequence) data locations, **or in the order in which they occur in the sequence** (chronological).
* A linked list can **provide a foundation** upon which other data structures can be built such as a stacks, queues, graphs & trees.
* A linked list is constructed from **nodes & pointers:**
1. Each node stores two values. It contains a ‘**data field**’ and a ‘**next address field**’.
2. The ‘data field’ holds the **actual data associated** with the list item.
3. The ‘next address field’ contains the **address of the next item** in the sequence. This is often known as a **pointer**. This is always an integer.
4. The ‘next address field’ can also been known as the ‘link’. This is shown by a ‘**null pointer**’. This means that **there are no further items** in the list.

**Visualise a Linked List:** (Traversing)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| Data: | A | H | C | E | L | L | B | D | O | Null |
| Pointer: | 6 | 3 | 7 | 4 | 5 | 8 | 2 | Null | Null | Null |

Take a look at this 2D array above. Let’s just go over something really quick:

Print(array[2]) will output [“C”,7]

Print(array[2] [0]) will output “C”

Print(array[2] [1]) will output 7

A 2D array can be used to store a linked list. Each data location in a linked list is called a ‘node’



The box highlighted in **red** is one of the many examples of a ‘node’ in the linked list. Each memory location is called a node.

* C is the data field. This is the actual data associated with the list item
* 7 is the pointer. This points towards the next item in the list to build the sequence.

This linked list begins at location 0: (meaning that the pointer = 0)



So first bit of data for our list is “A”.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** |  |  |  |

The pointer is 6, so we go to the sixth item in the list:



The second data item in our sequenced list is ‘B’.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** |  |  |

The pointer is now 2, so we go to the second item in the list.



The third data item in our sequenced list is ‘C’

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** |  |

The pointer is now 7, so we go to the seventh item in the list:



The fourth data item in our sequenced list is ‘D’

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **D** |

The pointer value is either ‘empty’ or ‘Null’. This concludes the end of the sequenced list as there is no value that points us to another position in the array. This means that there are no further items in the list.

This means that the linked list is [‘A’, ‘B’, ‘C’, ‘D’], starting at location 0 and following the pointer.

However, if we started at location 1, and followed that pointer. We would end up with another linked list.

**Traversing a linked list in python:**

First, you need the data you will be traversing. We are going to be using the 2D array presented above.

 

*The Traverse Script | Walkthrough:*

* **Pointer = 0** | The is the starting location for our linked list. Remember, it doesn’t necessarily have to be 0, there are other sequenced lists in the 2D array.
* **While pointer != None** |None basically meaning ‘Null’. The loop will stop if the pointer is equal to ‘ None ‘. This will conclude the end of linked list.
* **Node = array[pointer]** | the first node in the linked list is in the location array[pointer]. Remember that a node is a list of two items. The node at location 0 is: [“A”, 6]
* **Data = node[0]** | The first item in the node is the data value. It is the letter ‘A’. Print this value.
* **Pointer = node[1]** | The second item in the node is a pointer, it is the number 6. This becomes the new pointer.

*The result:*



Here is the result of the linked list if the starting location was 1 (pointer = 1):



**Appending to a Linked List:**





In order to use a 2D array as a linked list, the computer needs to remember two values.

**The header:** tells the computer where the first node of the linked list

**Free pointer:** This stores the location of the free empty cell. This tells the computer where there is spare space to add new values.

*The Appending Script | Walkthrough:*



This takes the ‘free pointer’ as a global variable. It creates a new node at that position of the free pointer and inserts the new data in.



This moves the pointer to the final node of the linked list.



This puts the new node onto the linked list by pointing to it and updates the free pointer.

