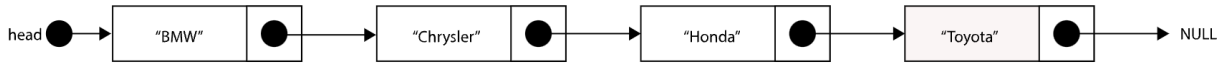
Linked Lists



A **Singly Linked List** is a linear data structure composed of **nodes**. Each node stores data and also a pointer to the next node in the list.

The first pointer is an *external pointer* pointing to the first node in the list.

The pointer from the last node will point to null (or equivalent).

An array is also a linear data structure.

One difference between linked lists and arrays is that linked lists are dynamic - the length of the list can change as nodes are inserted and removed. ***Arrays are static*** - array lengths are fixed.

Another difference between linked lists and arrays is that if we know the index of an array element, we can target it directly eg

```
arr[3] = 6;
```

However, when processing a linked list, we always start at the external pointer (pointing to the first node) and traverse the list.

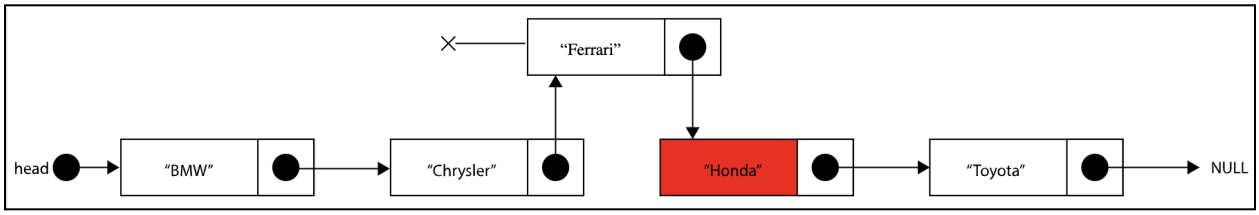
If the head pointer points to `null`, we can say that the list is empty.

Here is a pseudocode algorithm traversing a singly linked list:

```
//create a search pointer pointing to the head of the list
searchPointer = head
loop while searchPointer.next != null
    output searchPointer.data //output data of current node
    searchPointer = searchPointer.next //update searchPointer
end loop
```

Look how the search pointer updates - nice!

Inserting a node into a singly linked list

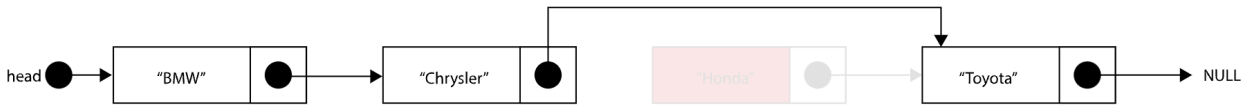


To insert a node into a list, follow these steps:

1. Create the new node eg "Ferrari"
2. Search for the insertion point from the head of the list eg "Honda"
3. Update the next pointer of the new node to point to the insertion point
4. Update the next pointer of the node before the insertion point ("Chrysler") to point to the new node

Removing a node from a singly linked list

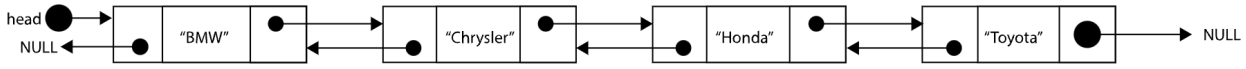
To remove a node from a singly linked list, follow these steps:



1. From the head of the list, search for the node to be removed eg "Honda"
2. Set the next pointer of the previous node ("Chrysler") to point to the same node as the node-to-be-removed's next pointer

Note: removing the first node will mean updating the external pointer, setting it to the first node's next pointer.

Doubly linked lists

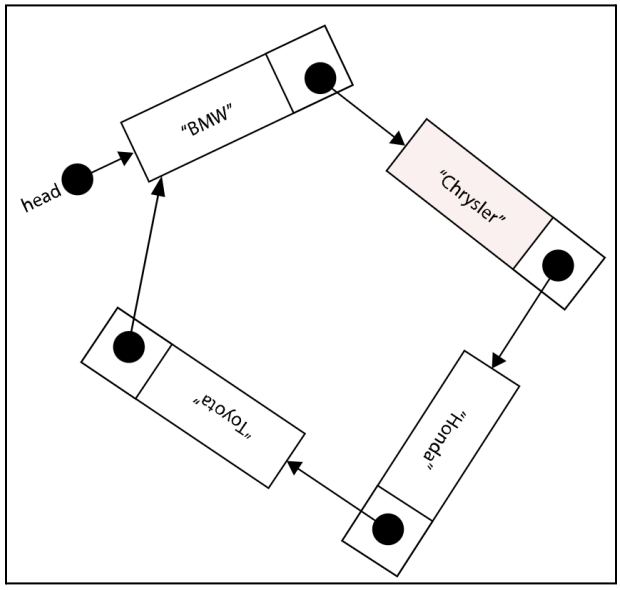


A doubly linked list is a list of nodes with an external pointer. Each node stores data and 2 pointers: one to the next node in the list and one to the previous node in the list.

The ***previous pointer*** in the first node points to `null` and the ***next pointer*** in the last node points to `null`.

So, a doubly linked list can be traversed in both directions.

Circular linked lists



A circular linked list is a list of nodes with an external pointer. Each node stores data and a pointer to the next node in the list. Unlike singly linked lists, a circular linked list's last node's next pointer does not point to null. It points to the head of the list. This allows for easy processing when nodes need to be constantly revisited. A classic example of its use is when an operating system schedules cpu-time to different processes (round robin scheduling). Each process is stored in a circular linked list and is revisited until the process is complete.

Advantages of Linked Lists

Dynamic memory allocation: Linked lists allow for dynamic memory allocation, meaning that the size of the list can change as elements are added or removed. This is different from static arrays whose size/length must be pre-determined and cannot be changed.

Space-efficient: linked lists are space-efficient, as they only need to store a reference to the previous/next node in each element, rather than a large block of contiguous memory.

Inserting nodes into a doubly linked list is relatively easy because after finding the insertion point, we have the address of the previous and next node to allow an easy insert. Singly linked lists can be a little more complicated in this regard.

It is complicated to insert items into a static array (requires shuffling) wheres linked lists simply require the pointer(s) to be updated when inserting or deleting nodes.

Disadvantages of Linked Lists

Poor random or direct access performance: Accessing an element in a linked list requires traversing the list from the head to the desired node, making it slow for random or direct access operations compared to arrays.

Increased memory overhead: Singly linked lists require additional memory for storing the pointers to the next node in each element, resulting in increased memory overhead compared to arrays.

Vulnerability to data loss: Singly linked lists are vulnerable to data loss if a node's next pointer is lost or corrupted, as there is no way to traverse the list and access other elements.

In singly linked lists, backward traversing not possible.

Linked lists cannot take advantage of Binary Search, a fast searching algorithm.

Past Paper Questions

9. Identify the components of a node in a doubly linked list. [3]

9. Compare the use of a linked list with an array to store and process the daily sales in a business. [3]

11. The names of vegetables must be always held in **alphabetical** order in a list in the main memory.

The application program should allow insertion and deletion of the names of vegetables from this list.

(a) Compare the use of a dynamic linked list for holding these names of vegetables with a static one-dimensional array. [3]

Given the following vegetables: potato, asparagus, lettuce, radish.

(b) Sketch a single linked list holding these vegetables. [2]

(c) (i) List the steps required to insert cabbage into the linked list. [4]

(ii) Explain why deleting the first node in this list is different to deleting other nodes. [2]

(d) State the dynamic data structure suitable for maintaining this list of vegetables which will allow faster searching for a given vegetable name. [1]

The vegetable names are input in the following order

potato, asparagus, lettuce, radish.

(e) Sketch the data structure suggested in part (d) containing the vegetable names sorted in **alphabetical** order. [3]

9. Data;
A pointer/reference to the previous node;
A pointer/reference to the next node; **[3]**

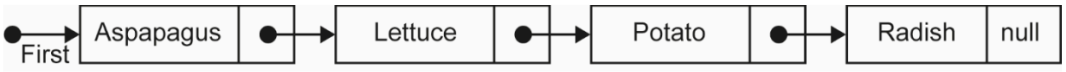
9. *Award **[3]**.*
*Award **[1]** for size comparison.*
*Award **[1]** for access comparison.*
*Award **[1]** for relation to the given scenario.*

Example:
Size/length of a static array is predetermined/fixed / size of a linked list can be changed / size of a linked list depends only on memory available;
Linked list can be expanded to suit the daily sales;
Each item in a static array can be directly accessed whilst access to the items in a linked list is sequential;
And searching for the given item in the linked list is slower /an item in the linked list cannot be searched for using binary search; **[3]**

11. (a) *Award **[1]** for each comparison, up to **[3 max]**.*

The size of the dynamic list does not have to be predetermined as in an array;
The size of the dynamic list is not fixed whilst the size of an array is always fixed;
If names are sorted they can be added/deleted (more easily) by changing the pointers without having to shuffle the names;
As records can be dynamically added/deleted the memory is better used because there are no wasted / missing spaces as in an array; **[3]**

(b) *Award **[1]** for a node containing two fields / data(name) and link (pointer) to the next node and **[1]** for showing an external pointer pointing to the first vegetable on the list, and null pointer in the last node, and pointers which link the nodes in alphabetical order up to **[2 max]**.*



[2]

(c) (i) *Award **[1]** for each step identified up to **[4 max]**.*

Create a new node containing cabbage;
Traverse the list (from the beginning) to find the place to insert a new node;
Cabbage should be inserted before lettuce and after asparagus;
The pointer in new (cabbage) node should be set to point to the node that is before the insertion point / lettuce;
The pointer in node before insertion point / asparagus / should now point to the new node / cabbage;

***Note**: award marks for clearly labelled diagrams in candidates' answers.* **[4]**

(ii) *Award **[1]** for identifying a reason why deleting the first node is different to deleting other nodes and **[1]** for an expansion up to **[2 max]**.*

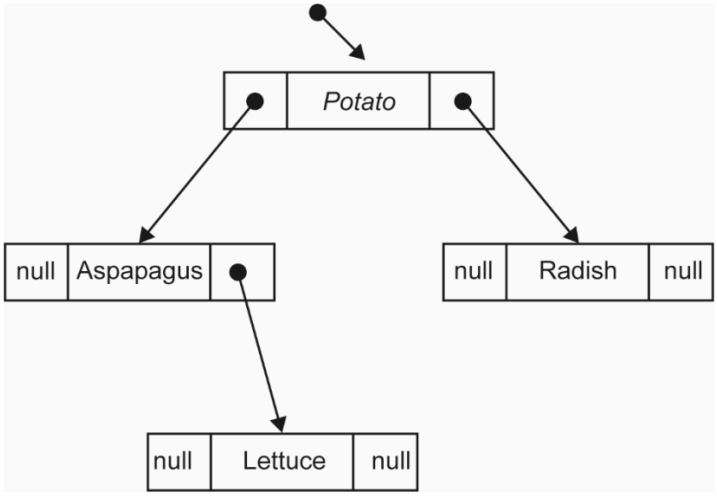
External pointer (First) must be changed/only in situation when deleting the beginning node the external pointer must be changed;
And set to the pointer in the link field of the first node (Asparagus) which points to the second node/Lettuce; **[2]**

(d) *Award **[1 max]**.*

Binary tree; **[1]**

(e) *Award **[3 max]**.*

*Award **[1]** for clearly a binary tree and the root is Potato.*
*Award **[1]** for the correct left subtree.*
*Award **[1]** for the correct right subtree.*



[3]