Lesson 5-6

Objectives

- The disadvantages of arrays are...
- What is Linked List?
- Why to use Linked List instead of arrays?
- Defining Linked List
- Implementation of Linked List

The disadvantages of arrays are...

- The size of the array is fixed — 100 elements in this case. Most often this size is specified at compile time with a simple declaration such as in the example above. With a little extra effort, the size of the array can be deferred until the array is created at runtime, but after that it remains fixed. (extra for experts) You can go to the trouble of dynamically allocating an array in the heap and then dynamically resizing it, but that requires some real programmer effort.
- Because of (1), the most convenient thing for programmers to do is to allocate arrays which seem "large enough" (e.g. the 100 in the scores example). Although convenient, this strategy has two disadvantages: (a) most of the time there are just 20 or 30 elements in the array and 70% of the space in the array really is wasted. (b) If the program ever needs to process more than 100 scores, the code breaks. A surprising amount of commercial code has this sort of naive array allocation which wastes space.
- (minor) Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room. Linked lists have their own strengths and weaknesses, but they happen to be strong where arrays are weak. The array's features all follow from its strategy of allocating the memory for all its elements in one block of memory. Linked lists use an entirely different strategy.
- **As we will see, linked lists allocate memory for each element separately and only when necessary**.
    - Linked lists have their own strengths and weaknesses, but they happen to be strong where arrays are weak. The array's features all follow from its strategy of allocating the memory for all its elements in one block of memory. Linked lists use an entirely different strategy. As we will see, linked lists allocate memory for each element separately and only when necessary.

What is Linked List?

Linked lists are probably the second most commonly used general purpose storage structures after arrays. The linked list is a versatile mechanism suitable for use in many kinds of general-purpose databases. It can also replace an array as the basis for other storage structures such as stacks and queues. In fact, you can use a linked list in many cases in which you use an array, unless you need frequent random access to individual items using an index. Linked lists aren’t the solution to all data storage problems, but they are surprisingly versatile and conceptually simpler than some other popular structures such as trees.

Why to use Linked List instead of arrays?

- An array is the data structure that contains a collection of similar type data elements whereas the Linked list is considered as non-primitive data structure contains a collection of unordered linked elements known as nodes.
- In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location within the square bracket.
- In a linked list though, you have to start from the head and work your way through until you get to the fourth element.
- Accessing an element in an array is fast, while Linked list takes linear time, so it is quite a bit slower.
- Operations like insertion and deletion in arrays consume a lot of time. On the other hand, the performance of these operations in Linked lists is fast.
- Arrays are of fixed size. In contrast, Linked lists are dynamic and flexible and can expand and contract its size.
- In an array, memory is assigned during compile time while in a Linked list it is allocated during execution or runtime.
- Elements are stored consecutively in arrays whereas it is stored randomly in Linked lists.
- The requirement of memory is less due to actual data being stored within the index in the array. As against, there is a need for more memory in Linked Lists due to storage of additional next and previous referencing elements.
- In addition memory utilization is inefficient in the array. Conversely, memory utilization is efficient in the linked list.

Following are the points in favor of Linked Lists.

- The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, the upper limit is rarely reached.
- Inserting a new element in an array of elements is expensive because a room has to be created for the new elements and to create room existing elements have to be shifted.

  For example, suppose we maintain a sorted list of IDs in an array id[].

  id[] = [1000, 1010, 1050, 2000, 2040, …..].

  And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

  Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

- So Linked list provides the following two advantages over arrays
  Dynamic size
- Ease of insertion/deletion

Defining Linked List

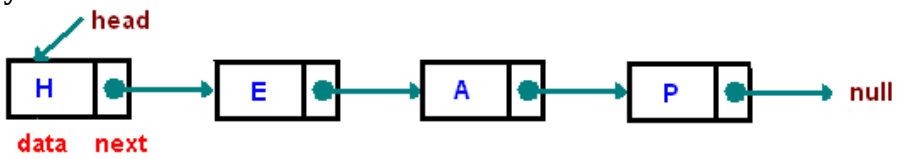
- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list. Each element (we will call it a node) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate

node, but the reference to the first node. If the list is empty then the head is a null reference.

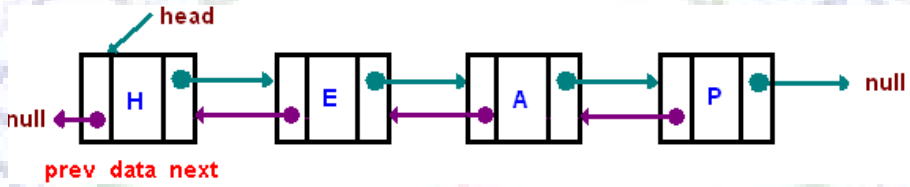
- A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.
- One disadvantage of a linked list against an array is that it does not allow direct access to the individual elements. If you want to access a particular item then you have to start at the head and follow the references until you get to that item.

Types of Linked Lists

- A **Singly Linked List** is described above



- A **Doubly Linked List** is a list that has two references, one to the next node and another to previous node.



Basic Operations

Following are the basic operations supported by a list.

- Insertion − Adds an element at the beginning of the list.
- Deletion − Deletes an element at the beginning of the list.
- Display − Displays the complete list.
- Search − Searches an element using the given key.
- Delete − Deletes an element using the given key.