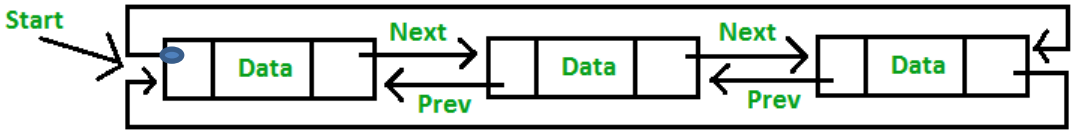
Circular Linked Lists



Circular lists have no ends so the biggest problem in traversal is not traversing endlessly. They can be single or double.

Notice too there are no NULL pointers

Practical use:

Say you are designing a game with many players. A circular linked list can easily keep tabs on player turns

Also, in time sharing operating system applications. If many users want to use the same resource, a circular linked list can keep track of who is next

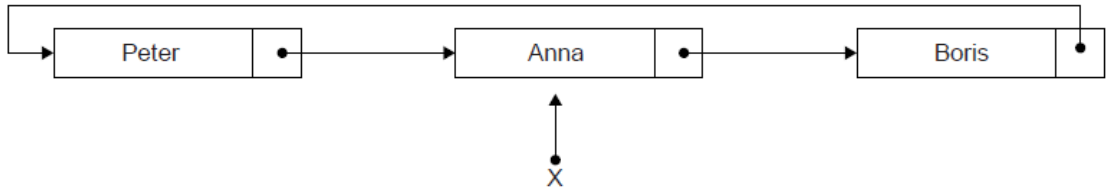
A circular list can be used to create a STACK

We could do this by :

1. Initialise an empty stack
2. Traverse the list pushing each data value from the list onto the stack
3. We can pop an element from the stack .

Adding Elements

11. The diagram shows a list of names held in a circular linked list. The end of the list is pointed to by an external pointer, X.



(a) State the first name in this circular list. [1]

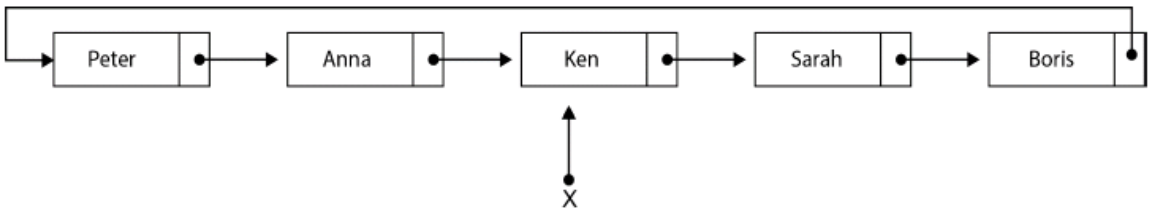
Two operations are performed on the list in the following order:
1. A node containing the name Sarah is inserted at the beginning of the list.
2. A node containing the name Ken is inserted at the end of the list.

(b) Sketch a diagram showing the resulting circular linked list. [3]

(c) Describe how the number of names held in this list could be determined. [4]

Boris is the start as it is a circular list and Anna is at the end of the list

Like the doubly linked example if the question hadn't suggested it, draw the end result first AND then explain it.



1. Start at the START (seems logical!) and follow along unit we get to Anna (End of list)
2. Put its .next pointer (Boris) in a temporary variable
3. Make anna' s **.next** pointer to point to Ken
4. Mark Ken as end of the list X
5. Ken's .next points to Sarah
6. Move one node along
7. Make Ken's **.next the** temp value.
8. Sarah's .next points to the temp one ie Boris

Note that removal will be a similar process. Keep it logical!