Queue Implementation

- The operations we need to implement a Queue are **ENQUEUE** (add to the end or *tail* of the queue) and **DEQUEUE** (take from the front or *head* of the queue).

- We haven’t seen a data structure that supports O(1) insert-at-end (or Last) and O(1) delete-at-start. So let’s invent one!

- Using singly-linked lists, we know how to do insert-at-front in O(1) time, but computing Last (L) is O(n).

- Solution 1: keep another pointer, which always points to the last element in the linked list:
Queuing Method 1

- Queue Structure:
  - A linked list - the first element is the head of the queue, and the last element is the tail of the queue.
  - A "header" structure, which has 2 pointers, one to the head of the queue and one to the tail of the queue.
Queuing Method 1: Dequeue

1. Create a pointer and set it equal to header->head

```
header
    ^
   / \             \      new_ptr
  /   \           \        data
  \    \          \       data
   \   \         \      ...
    \ /          \     
     \           \      data
      \          \     
       \         \    
        \     head
         \   tail
          \ data
```

2. header->head = header->head->next;
   new_ptr->next = NULL;

```
header
    ^
   / \             \      new_ptr
  /   \           \        data
  \    \          \       data
   \   \         \      ...
    \ /          \     
     \           \      data
      \          \     
       \         \    
        \     head
         \   tail
          \ data
```
Queuing Method 1: Enqueue

1. Create new Linked List element (we’ll call it new_q), fill in the data element, set the 'next' pointer to null

2. header->tail->next = new_q

3. header->tail = new_q
Queuing Method 2

- Reduces the header structure to a single cell pointer, by using a **circularly** linked-list

- In a circularly-linked list, there is no NULL pointer designating the end of the list - it points to the first element. From a single pointer, you can get to any element of the list.
Queuing Method 2

- Queue Structure:
  - A circularly linked list which instead of ending at a NULL pointer, wraps around to point to the start of the list;
  - A pointer (tail_ptr) which points to the tail of the queue, so the head of the queue is tail_ptr->next

```
    data
    data
    data
    data
    data

    tail_ptr
```

(first element of the queue)
Queuing Method 2: Dequeue

1. Create a pointer and set it equal to `tail_ptr->next` (the head of the queue)

2. (a) `tail_ptr->next = tail_ptr->next->next;`
   (b) `new_ptr->next = NULL;`
Queuing Method 2: Enqueue

1. Create new linked list element (we’ll call it `new_q`) and fill in the data element

2. (a) `new_q->next = tail_ptr->next;`
   (b) `tail_ptr->next = new_q;`

3. `tail_ptr = new_q;`