CSE 332: Summer 18
Section 01 Quick Check.

1 $\mathcal{O}$dd Jobs

For each of the following scenarios, choose

(1) an ADT: `Stack` or `Queue`, and
(2) a data structure: `Array` or `LinkedList` with `front` or `LinkedList` with `front` and `back`

Then, explain why your choice works better than the other options.

`WorkList` Situations

You're designing a tool that checks code to verify that all opening brackets, braces, parentheses, ... have closing counterparts.

Solution: We'd use the `Stack` ADT, because we want to match the *most recent* bracket we've seen first.
Since `Stack`s push and pop on the same end, there is no reason to use an implementation with two pointers. (We don't need access to the "back" ever.)
Asymptotically, there is no difference between the `LinkedList` with a `front` pointer and the `Array` implementation, but *cache locality* will likely be a problem with the `LinkedList`. (Remember, arrays are contiguous in memory, but linked lists are stored using arbitrary pointers.)

Disneyland has hired you to find a way to improve the processing efficiency of their long lines at attractions. There is no way to forecast how long the lines will be.

Solution: We'd use the `Queue` ADT here, because we're dealing with... a line. The important thing to note here is that if we try to use the implementation of a `LinkedList` with *only a* ***`front`*** *pointer*, either *add* or *next* will be very slow. That is clearly not a good choice.
Arguably, the `LinkedList` implementation with both pointers is better than the array implementation because we will never have to resize it.

A sandwich shop wants to serve customers in the order that they arrived, but also wants to look ahead to know what people have ordered and how many times to maximize efficiency in the kitchen.

Solution: This is still clearly the `Queue` ADT, but it's unclear that any of these implementations are a good choice!
One of the cool things about data structures is that if only one isn't good enough, you can use *two*. If we only care about the "normal queue features", then we would probably use the `LinkedList` implementation with one pointer. However, we can **ALSO** simultaneously use a *Map* to store the "number of times a food item appears in the queue".
This is still actually a `WorkList`! It has the same interface!! But, in the implementation, we update both the map and the queue whenever something changes.