3. (a) Write an array based implementation of the queue module described in Exercise 1. Use three integers to keep track of the queue's status, with one integer storing the position of the first empty slot in the array (used when an item is inserted), the second storing the position of the next item to be removed, and the third storing the number of items in the queue. An insertion or removal that would cause either of the first two integers to be incremented past the end of the array should instead reset the variable to zero, thus causing it to "wrap around" to the beginning of the array.

   (b) Write a linked-list implementation of the queue module described in Exercise 1. Use two pointers, one pointing to the first node in the list and the other pointing to the last node. When an item is inserted into the queue, add it to the end of the list. When an item is removed from the queue, delete the first node in the list.

Section 19.3

W 4. (a) Write an implementation of the `Stack` type, assuming that `Stack` is a structure containing a fixed-length array.

   (b) Redo the `Stack` type, this time using a linked-list representation instead of an array. (Show both `stack.h` and `stack.c`.)

5. Modify the `queue.h` header of Exercise 1 so that it defines a `Queue` type, where `Queue` is a structure containing a fixed-length array (see Exercise 3(a)). Modify the functions in `queue.h` to take a `Queue *` parameter.

Section 19.4

6. (a) Add a `peek` function to `stackADT.c`. This function will have a parameter of type `Stack`. When called, it returns the top item on the stack but doesn't modify the stack.

   (b) Repeat part (a), modifying `stackADT2.c` this time.

   (c) Repeat part (a), modifying `stackADT3.c` this time.

7. Modify `stackADT2.c` so that a stack automatically doubles in size when it becomes full. Have the `push` function dynamically allocate a new array that's twice as large as the old one and then copy the stack contents from the old array to the new one. Be sure to have `push` deallocate the old array once the data has been copied.

Programming Projects

1. Modify Programming Project 1 from Chapter 10 so that it uses the stack ADT described in Section 19.4. You may use any of the implementations of the ADT described in that section.

2. Modify Programming Project 6 from Chapter 10 so that it uses the stack ADT described in Section 19.4. You may use any of the implementations of the ADT described in that section.

3. Modify the `stackADT3.c` file of Section 19.4 by adding an `int` member named `len` to the `stack_type` structure. This member will keep track of how many items are currently stored in a stack. Add a new function named `length` that has a `Stack` parameter and returns the value of the `len` member. (Some of the existing functions in `stackADT3.c` will need to be modified as well.) Modify `stackclient.c` so that it calls the `length` function (and displays the value that it returns) after each operation that modifies a stack.

4. Modify the `stackADT.h` and `stackADT3.c` files of Section 19.4 so that a stack stores values of type `void *`, as described in Section 19.5; the `Item` type will no longer be used. Modify `stackclient.c` so that it stores pointers to strings in the `s1` and `s2` stacks.