Module 2 - Discussion Prompt Questions
Adam Rich
Section EN.605.202.87
Spring 2018

1. **Identify some key characteristics of stacks or stack operations.**
   Stacks are LIFO structures, the only element that is meant to be accessible is the one last inserted.
   The ADT has no limits on number or nature of items (implementation will, though).
   Stacks are used a lot, by operating systems to facilitate switching between process, or to push arguments for function calls, etc.
   Stacks also support recursion.

2. **What methods are *required* to implement a stack?**
   Push (insert)
   Pop (delete and return value)
   Is_Empty (test if the stack has any elements)

3. **What methods might be helpful in managing a stack, but are not required? You might consider methods that could be implemented as a combination of methods identified in the last question.**
   Peek – could also be a combination of Pop and Push
   Display – create a visual representation of a stack
   Copy or Deep Copy
   Clear Stack – pop 'till you drop. Pop it like it's hot? Just remove everything in the stack and start afresh.

4. **Name an example (or two) of an application for which stacks would be helpful, and explain why. Consider real-life applications, not just classic algorithms.**
   How about getting off and on the elevator or the subway? If you have a lot of people involved it is so much more efficient to have the people who got on last, i.e. those closest to the door, get off first.

   My desk at work . . . I just keep piling important stuff around me (in different stacks). The more recent stuff goes on the top. It would be silly to put new stuff on the bottom, even though it would be fairer for me to deal with older requests first (why should the be penalized just because I got those requests right after the last time I cleaned my desk?). Likewise, when I go through everything (about once a day) I start with the stuff on top. A good example of a LIFO process. To keep it fair, though, I go through everything and come up with a plan before starting work. So, processing inputs is LIFO but actioning them depends on the actual priority.

5. **Name an example (or two) of an application for which stacks would NOT be helpful, and explain why. Consider real-life applications, not just classic algorithms.**
   Serving people at a restaurant . . . if you served the most recently arrived customers first you would have some very angry and hungry (hangry?) people sitting around all day.

   The DMV. If the most recent arrivals were attended to first, everyone would wait till the end of the day to show up!

6. **Have you used prefix or postfix notation in the past? In what context? Did you find it useful?**
   I have never used prefix or postfix notation. However, I have worked with more experienced actuaries, usually pension actuaries, that have old HP financial calculators that use Reverse Polish Notation. They think about arithmetic problems differently, RPN is unambiguous when it comes to order of operations. Generally, these actuaries are faster at using their calculators than I am with mine . . . but that could be for lots of different reasons.

7. **What are some pros and cons of choosing to implement a stack using an array versus a list implementation?**

   Array Pros: can use random access, i.e. can read more than just the top element. No part of the stack ADT but could be useful anyway
   Array Cons: you have to define the length of the array up front so there is a limit to the number of items in the stack, bounded by more than just the memory constraints of the system. Also, depending on how the array is implemented, popping might just move the "top" pointer and leave the data in the array. If so, the programmer just needs to be careful to not treat the stack like a true array. Or implement it to clear the data when popping.
   List Pros: can have as many elements in the stack as memory allows, you do not need to know an upper bound on the stack size up front
   List Cons: no sequential access, but this is not part of the stack definition anyway