Topic 11.1: ADTs

Learning Goals:

- Differentiate between an abstract data type and a data structure.
- **Implement Stack and Queue abstract data types using a linked list data structure.**

Stack

- We've heard this term before (runtime stack?)
- Stack of pancakes!
- Newest pancakes (or method calls) on top
- Oldest on the bottom
- Gotta eat through the top pancakes to reach the bottom pancakes
  - Also known as (last in; first out, LIFO)
- Formally,
  - A **stack** of data is a sequence of values where only the most recently added value, of all the values currently in the sequence, can be accessed.
  - The value on "top" of the stack is the only one we can see or remove. The top is the only spot we can add a new value.

Stack

- We've heard this term before (runtime stack?)
- Stack of pancakes!
- Newest pancakes (or method calls) on top
- Oldest on the bottom
- Gotta eat through the top pancakes to reach the bottom pancakes
  - Also known as (last in; first out, LIFO)
- Formally,
  - A **stack** of data is a sequence of values where only the most recently added value, of all the values currently in the sequence, can be accessed.
  - The value on "top" of the stack is the only one we can see or remove. The top is the only spot we can add a new value.

- **Only the TOP you say...hm... should we implement this with**
  - **a LinkedList or an Array or an ArrayList?**

Stack

- You can place an item onto the top of the stack.
- You can examine the item on the top of the stack.
- You can remove the item from the top of the stack.
  - The item beneath it is now the top. It can be removed, or more items placed on it.
- We can’t remove items when none are left.
  - Until we add more.

Stack

- You can place an item onto the top of the stack.
- A **push** operation

- You can examine the item on the top of the stack.
- A **peek** operation

- You can remove the item from the top of the stack.
  - The item beneath it is now the top. It can be removed, or more items placed on it.
- A **pop** operation

- We can’t remove items when none are left.
  - Until we add more.
- if **isEmpty()** we can push more on

Stack ADT Interface

- An ADT Stack is defined by this restricted set of operations.
- Just like the Ordered List, where the ADT cannot allow operations that violate its ordering constraint.
- If you allow it to have (for example) a get for an arbitrary element, it’s not a Stack any more!

- We can define a stack using an interface (See StackInterface.zip).
- It can be a stack of Objects, or a more specific type.

Stack Implementation

- A stack can be implemented using a partially-filled array: we can easily push items to the end of the array, and pop them from the end.
    - The end of the array is the “top” of the stack.
    - We could also use an ArrayList, only allowing access at the end.
    - A stack can be implemented using a linked list, where we push/pop from the top of the linked list.
- LinkedList example is included in the example (See StackInterface.zip).

Stack Applications

- Some places where stacks are used:
    - To **reverse items**: push n items onto a stack, then pop them all to get them in the reverse order.
    - **Return path finding** (“backtracking”): go from a to b to c to d, push each location onto a stack, and pop them to go back.
        - We can try following different paths (branches) when we go back to an earlier location.
    - The **run-time stack** that makes recursion possible.

Replacing Recursion

- The last application is particularly interesting.

- The run-time stack is part of every program's execution environment, and allows functional independence and recursion.
  - By defining our own stack, we can **simulate** the use of the run-time stack in a recursive function.

- That means we can replace any recursive function with a non-recursive function and our own stack!
- Why? Just because! (and in some cases, it might be helpful too)

Queues

- and now the Queue ADT
- You may have heard this term too
- Sometimes a fancy word for a line (like a line at Tim Hortons or a virtual queue while waiting for T-Swift tickets online)
- It describes a First in; First out (FIFO) data relationship. First one to get in line is the first to be served. (My in-person office hours are a queue)

- Formally,
  - A **queue** of data is a sequence of values where only the least recently added value, of all the values currently in the sequence, can be accessed.
  - We add new values to one end of the queue, but we can only access values at the other end.

Queue

- You can place an item at the back/**tail** of the queue.

- You can examine the item at front/**head** of the queue.
  - Head and tail are at the opposite ends!

- You can remove the item from the head of the queue.

- We can’t remove items when none are left.
  - Until we add more.

Queue

- You can place an item at the back/**tail** of the queue.
- A **enqueue** operation

- You can examine the item at front/**head** of the queue.
  - Head and tail are at the opposite ends!
- A **peek** operation

- You can remove the item from the head of the queue.
- A **dequeue** operation

- We can’t remove items when none are left.
  - Until we add more.
- if **isEmpty()** we can push more on

Queue ADT Interface

- We can define a **queue** of Objects using the following interface. Like a Stack, we limit the operations.

- We can define a Queue using an interface (See QueueInterfaceExample.zip).
- It can be a queue of Objects, or a more specific type.

Queue Implementation

- A queue can be implemented using a **partially-filled array**: we could enqueue items at the end (easy) and dequeue them at the start (hard!).
  - It’s hard because we need to shift the remaining elements over when we dequeue the head. It’s easier but no more efficient with an ArrayList.
  - The start of the array is the **head**, end is **tail.**
- There is a better strategy using arrays, covered in later CS courses.
- **LinkedLists** are definitely easier for Queue implementation
  - if we keep a **top** and **tail** pointer!
  - Let’s implement it!
  - See QueueInterfaceExample.zip

Queue Applications

- Some places where queues are used:
    - When a program has to keep track of the work it needs to do, it can add the tasks to a queue and execute them in order.
    - As a buffer: to store a sequence of data values that need to be processed.
        - Such as sending data on a slow network, where new data has to wait until old data has been transmitted.
- Aside: Modified Queue types exist too
    - e.g. instead of First In First Out there are PriorityQueues which are kind of like OrderedLists maintained by Priority (more on this in future classes!)