7.2 - Week 7 - Applied Practical

Objectives of this Applied Practical Session

The objectives of this class are as follows:

- The Stack and Queue ADT and how to use it
- Big O time complexity of simple algorithms

and we know how to both implement and follow code that combines these concepts.

Simple Implementation of StackADT

Using the given code for StackADT, implement an array-based version of Stacks that will be used in the future.

HINT - You can use the implementation covered in the lecture, but make sure to go through each method and understand what it's doing. What methods should be overridden? What methods need to be further defined?

Generating Printing Combinations

Consider the ArrayStack ADT defined in the lectures, where the pop method has been modified to also print out the return value. Given the stack **my_stack** = ArrayStack(5), your task is to add to the code:

```
my_stack.push(0)
my_stack.push(1)
my_stack.push(2)
my_stack.push(3)
my_stack.push(4)
```

5 calls to **my stack.pop()** so that the resulting code prints a particular sequence of numbers. For example, by adding it like:

```
my_stack.push(0)
my_stack.push(1)
my_stack.pop()
my_stack.push(2)
my_stack.push(3)
my_stack.pop()
my_stack.pop()
my_stack.pop()
my_stack.push(4)
my_stack.pop()
```

the code will print sequence 1 3 2 0 4.

In `Exercise 1.txt`, for each of the following sequences of numbers, give the code resulting from adding 5 **my stack.pop()** calls to the 5 **push** instructions above that would print that sequences:

1. 2 1 0 3 4
2. 2 4 3 1 0
3. 2 1 0 4 3
4. 3 4 1 2 0

Without reordering the push operations.

Simple Implementation of Circular Queues

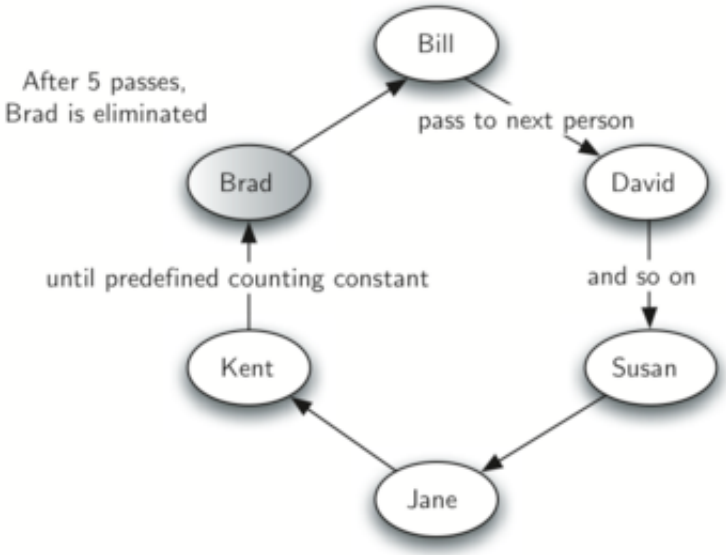
Now, let's do something similar for Queues. You've been given implementation of QueueADT. You need to create a class for a Circular Queue that encompasses the properties of a Circular Queue. Again, you can reuse the code but you need to understand how it works because otherwise, it won't be of any use to you in your assignment + interviews.

The Hot Potato Game!

HOT POTATOOOOOO! Pass it along or get burrrnttttt

Alright, so we are going to use our implementation of Circular Queue to design a game of hot potato. Here are the rules:

One of the typical applications for showing a queue in action is to simulate a real situation that requires data to be managed in a FIFO manner. To begin, let's consider the children's game Hot Potato. In this game (see diagram) children line up in a circle and pass an item from neighbor to neighbor as fast as they can. At a certain point in the game, the action is stopped and the child who has the item (the potato) is removed from the circle. Play continues until only one child is left.



This game is a modern-day equivalent of the famous Josephus problem. Based on a legend about the famous first-century historian Flavius Josephus, the story is told that in the Jewish revolt against Rome, Josephus and 39 of his comrades held out against the Romans in a cave. With defeat imminent, they decided that they would rather die than be slaves to the Romans. They arranged themselves in a circle. One man was designated as number one, and proceeding clockwise they killed

every seventh man. Josephus, according to the legend, was among other things an accomplished mathematician. He instantly figured out where he ought to sit in order to be the last to go. When the time came, instead of killing himself, he joined the Roman side. You can find many different versions of this story. Some count every third man and some allow the last man to escape on a horse. In any case, the idea is the same.

We will implement a general **simulation** of Hot Potato. Our program will input a list of names and a constant, call it "num," to be used for counting. It will return the name of the last person remaining after repetitive counting by `num`. What happens at that point is up to you.

Define a method `hot_potato(queue: CircularQueue, num: int)` that takes in a Circular Queue of variable length containing strings of names and an integer which contains the number of moves before eliminating someone (in the real game this should be randomized but let's just code it for something fixed right now) and returns the name of the person that is left in the queue after everyone else has been eliminated.

SPOILER (Massive Hint Ahead)

▶ Expand

QUEUE | EUEUQ

This exercise is about *circular queue* implementation. Consider the implementation of circular queues we saw in the lectures. Write a Python method `print_reverse_queue(self) -> None` within our `CircularQueue` class that prints all the items in the queue from rear to front.

Make sure you either do not change the queue, or (if you cannot think how to do it without modifying it) the queue is at least left as you found it.