Read Me:

1. Connor Pham
2. Project Structure

Class PrintItem{}:

In this class there are two variables: string fileName and float fileSize. They are used as parameters for the constructors in this class and are used to initialize an object of the PrintItem class. There also contains a print() method that prints what is stored in the variables. It has a deconstructor method.

template <typename T> class Node{}:

This is a class of typename T, made to be generic for code reusability. There are two variables T *Data and Node<T> *nextNode. Data is what is loaded into the node and nextNode acts as a pointer. The constructor takes in *data as an argument to set the Node's data. There is a deconstructor that deletes the two variables and there is a print method to print the data of the node.

template <typename T> class LLStack{}:

This is a class of typename T, made to be generic for code reusability. There are 3 private variables int stackSize, const int SMAXITEMS, and Node<T> *top. They function as they are described. The default constructor initializes the variables, while the other constructor takes in the argument T *value to initialize the top node and stackSize becomes 1. It has a valid deconstructor ~LLStack(){} that deletes all nodes and data, if called. Both bool IsFull() and bool

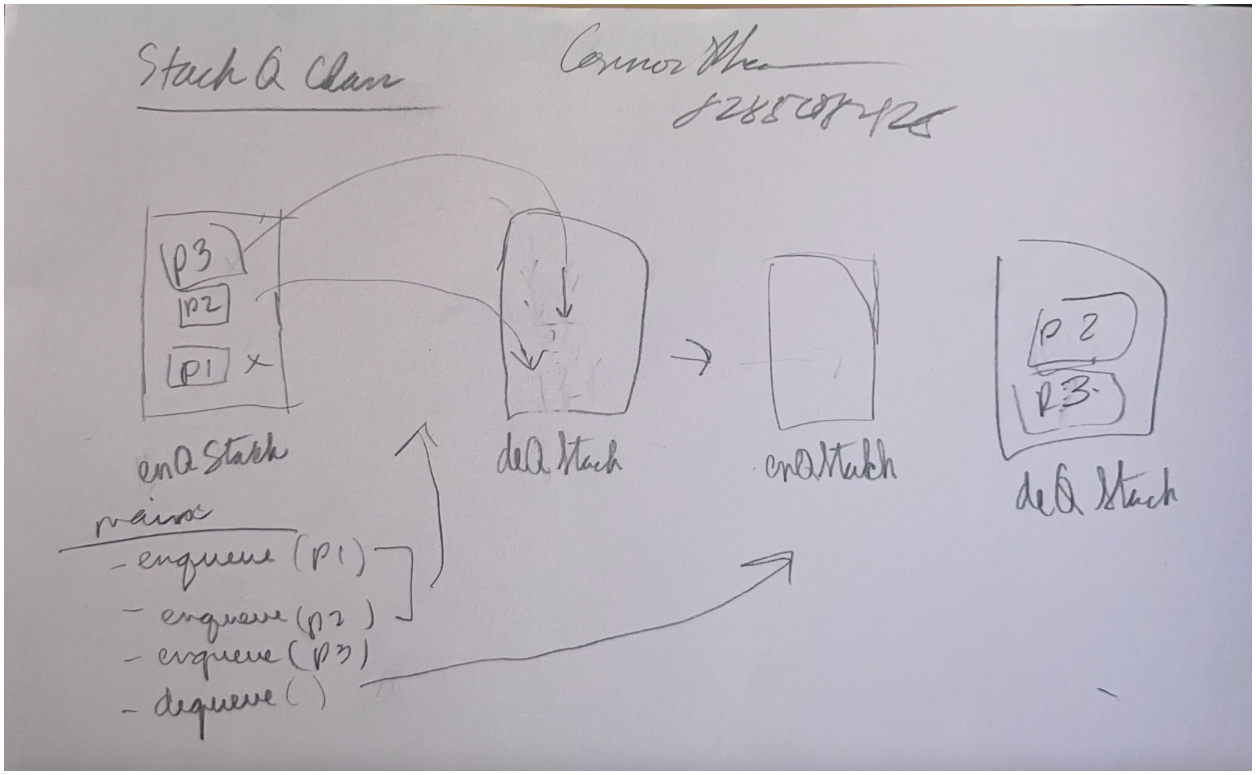
IsEmpty() are methods that return true or false depending if the stack is full or if it is empty. The void push(T *item){} method pushes T *item to the top of the stack and increments stackSize. The void pop(){} method pops the topmost item in the stack and decrements stackSize. T *peek() {} method returns the node's data of the top item within the stack. The void print() method prints all items of the stack. Lastly, Node<T> *getTop(){} is to return the top item of the stack, it is used in the StackQ class as a way to access Node<T> *top, since it is a private member, I had to create a public method as a way to access it.

template <typename T> class StackQ{}

This is a class of typename T, made to be generic for code reusability. There are 4 private variables LLStack<T> enQStack, LLStack<T> deQStack, int queueSize, and const int QMAXITEMS. Both enQStack and deQStack are objects of the LLStack class and they are two separate stacks meant to operate like a queue. QueueSize functions as it sounds and so does QMAXITEMS. The default constructor StackQ(){} initializes the variables. The overloaded constructor StackQ(T *item){} enqueues T *item into enQStack while setting queueSize to 1 since an item was added. There is a functional deconstructor ~StackQ(){} that deletes enQStack and deQStack. Both bool IsFull() and bool IsEmpty() are methods that return true or false depending if the stack is full or if it is empty. The method void enqueue(T *item){} enqueues or adds T *item into the enQStack so long as the queue is not already full, adds the new node to the top of the enQStack, and its Empty condition for the enQStack is only to maintain the top reference. Conversely, the void dequeue(){} method: pops the topmost element from the deQStack, if the deQStackis empty it pop all the elements from the enQStack and pushes them one by one into the deQStack then pop from deQStack to perform dequeue(). T* peek(){} method returns the T* item with the highest priority in the queue. The int queueSize(){} returns queueSize. Both void

printQ(){} and void printEQandDQ(){} function the same to print the items in queue however, printEQandDQ() has a clear print statement differentiating the items of enQStack and deQStack while the printQ() prints the same list yet to the user it only seems like there is a queue, not multiple stacks.

3.



This is a loose sketch of what enQ() and deQ() would look like given the context of the functions created for this programming assignment. To explain the functions and this diagram, enQ() is called 3 times to load some data that doesn't really matter but I used blocks to make the picture clear. They are pushed into the stack from the top down into enQStack. Then to show how deQ() works, it pops all items and pushes them into deQStack, if deQStack is empty. In this case it is empty so all items from enQStack are popped and pushed while having the last element "p1"

discarded and deleted since the two stacks are to replicate the functionality of a queue. Stacks are LIFO so “p3” gets popped and pushed first into deQStack then “p2.” Since we want to deQ() and remove an item “p1” is the first item if we look at it as a queue so it has the highest priority, therefore, we would discard this item.