Author: Connor Symons

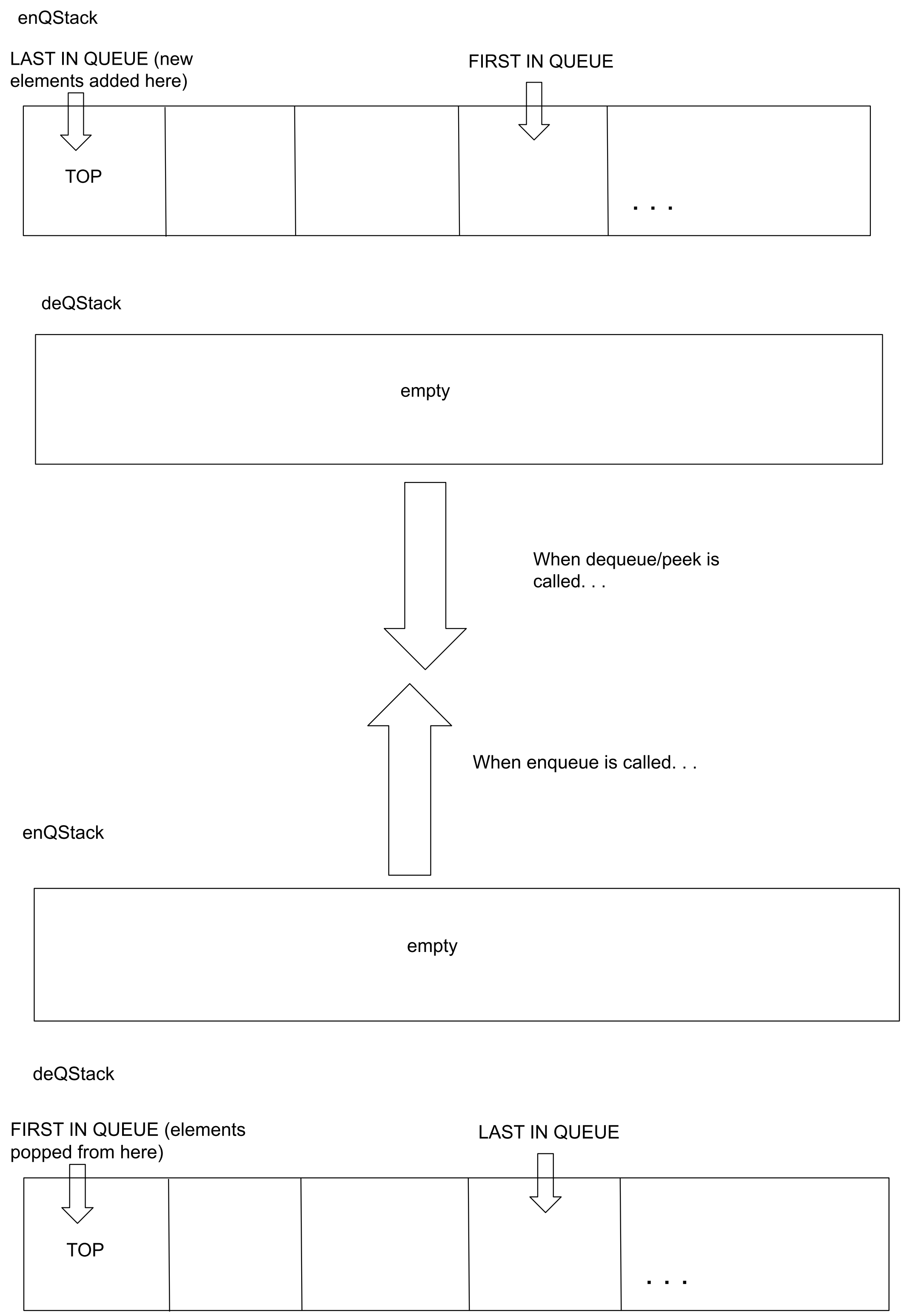
Project Structure:

- PrintItem
  - float fileSize, string fileName
    - Variables acting as the data to be stored/printed
  - PrintItem(float fileSize, string fileName)
    - Constructor initializes the two data variables above with the passed values.
  - Utilizes default destructor, as it has no extraneous data/data structures to delete
  - void print()
    - Displays the file size and file name for the PrintItem, in that order.
- Node<T>
  - T* data
    - Pointer for data of generic type is stored/associated with the node
  - Node<T>* nextNode
    - Since these nodes are used to form a List data structure, it is important to have a pointer to the next node in the list (nullptr if it is the last node in the list)
  - Node(T* data)
    - Constructor assigns pointer to the generic data passed to it and initializes nextNode with nullptr.
  - void print()
    - Calls the print method of the data associated with the node
- LLStack<T>
  - Node<T>* top
    - Pointer to the node that is considered to be on the top of the Stack ADT (head of List data structure underneath)
  - int stackSize
    - Running count of current size of the stack
  - const int MAXITEMS
    - Variable to control the max amount of items to be stored in the stack (set to 10 for easy testing of overflow conditions)
  - LLStack()
    - Constructor with no arguments creates an empty stack, sets top to nullptr and stackSize to 0
  - LLStack(T* data)
    - Constructor creates a stack with 1 node in it. Sets top to a new node initialized with the passed data and sets stackSize to 1.
  - ~LLStack()
    - Destructor uses a while loop to call the pop() method on itself until the stack is empty. This deletes all the nodes within the stack.
  - bool isFull()
    - Returns true if stackSize is equal to MAXITEMS

    - bool isEmpty()
        - Returns true if stackSize is equal to 0
    - void push(T* value)
        - If the stack is not full, create a new node with the passed value and place it on the top of the stack. Will assign the new node's nextNode pointer to be the old top if the stack isn't empty. Adds 1 to stackSize
    - void pop()
        - If the stack is not empty, pops the node that is currently on top off of the stack. Will delete the node that was popped to clear space in memory. Subtracts 1 from stackSize
    - T* peek()
        - Returns the generic data that is in the node on the top of the stack.
    - void print()
        - Creates a new temporary stack to transfer nodes as they are being popped. Will print the node on the top of the original stack, push its value to the temporary stack, then pop the node off of the original stack until the original stack is empty. Afterwards, it returns all nodes back to the original stack and deletes the temporary stack.
- StackQ<T>
    - LLStack<T>* enQStack
        - This stack holds the all of the nodes for the Queue ADT while the user is enqueueing values
    - LLStack<T>* deQStack
        - This stack holds the all of the nodes for the Queue ADT while the user needs access to the first value in the queue (dequeue, peek, print)
    - int queueSize
        - Running count of the current size of the queue
    - const int QMAXITEMS
        - Variable to control the max amount of items to be stored in the queue (set to 10 for easy testing of overflow conditions)
    - StackQ()
        - Constructor initializes the two stacks with their default (empty) constructors and sets the queueSize to 0
    - StackQ(T* value)
        - Constructor initializes the two stacks and calls the default constructor on deQStack while calling the enQStack constructor with the passed generic value. Sets the queueSize to 1
    - ~StackQ()
        - Destructor calls the destructor of both the enQStack and deQStack, which will result in all nodes within being deleted
    - bool isFull()
        - Returns true if queueSize is equal to QMAXITEMS
    - bool isEmpty()
        - Returns true if queueSize is equal to 0

- void stackTransferDeQ
    - Pushes all nodes from the enQStack to the deQStack
- void stackTransferEnQ
    - Pushes all nodes from the deQStack to the enQStack
- void enqueue(T* value)
    - If the queue isn't full, calls stackTransferEnQ(), then pushes a new node initialized with the passed generic value to the top of enQStack. Adds 1 to queueSize
- void dequeue()
    - If the queue isn't empty, calls stackTransferDeQ(), then pops the node on the top of deQStack. Subtracts 1 from queueSize.
- T* peek()
    - Calls stackTransferDeQ(), then prints and returns the generic data that is in the node on the top of the deQStack (first in the queue ADT).
- void print()
    - Calls stackTransferDeQ(), then calls print() on deQStack
- void printStacks()
    - Prints the contents of both enQStack and deQStack
- int getSize()
    - Returns the current value of queueSize member variable

Problem Definition:
As items are enqueued into the queue ADT, they are put on top of the enQStack. The first item enqueued will be on the bottom of the enQStack while the last item enqueued will sit on the top. The deQStack remains empty while items are being enqueued. When an item needs to be dequeued from the queue, stackTransferDeQ() is called, which will pop elements from the top of the enQStack and push them to the deQStack. This action inverts the order of the items in the stack, placing the first queue element on the top of the stack while having the last element sit on the bottom of the stack. Items can be dequeued from the queue while the deQStack still has items, and the first value in the queue can be peeked at since it sits on top of the stack. All of the items remain in the deQStack until an enqueue is called, then stackTransferEnQ() is used to transfer all of the items back to the enQStack, inverting the items back to where they were originally.


enQStack
LAST IN QUEUE (new elements added here)
FIRST IN QUEUE
TOP
. . .
deQStack
empty
When dequeue/peek is called. . .
When enqueue is called. . .
enQStack
empty
deQStack
FIRST IN QUEUE (elements popped from here)
LAST IN QUEUE
TOP
. . .