Iley Alvarez Funcke

The Fundamentals Lab Assignment Text

Question 1

The recursive function I made was very short and concise. The way it works is that every time the function is called it lets the user input a char and if that char isn't a return key click the function gets called again. When the user finally presses enter the function continues and writes out the entered char, since it's recursive it will do this backwards since it will go back one "function iteration" at a time and write out what was input that time.

The iterative function is much simpler and just allows an input that is 5 chars long or shorter (since it was ok to have a max). What it does is simply to let the user input 5 or less chars that are put in an array and this array is later printed out in reverse order.

The test for this program was simply to run both functions since they let the user input anything.

Are there other pros/cons of the two approaches?

- I made the iterative one not be able to have any size of input, of course this is a negative.
- The recursive one is shorter and arguably easier to understand because of that.
- The iterative one went faster to make and wasn't as hard to come up with

These are the main differences and so I'd say the recursive one is better if you want code that looks good and in this case also can take any size inputs. However if you want to write code that's easy to come up with and fast to write out and you can have a length limit on the input then the iterative version works great.

Question 2

The recursive method I made first got the user's input. After this it used an int counter x and the length of the input to check if it was on the last char of the inputted string, if not the char it was on was printed out. The correct char was printed out using the int counter since for every call of the method it was incremented and we could use the charAt method to write out the correct char using this int counter.

The iterative version I made using a stack (a stack class I made myself). It simply lets the user input a String and this string is later pushed into the stack one char at a time using a for-loop. When this is done we later write up every element we pop out one at a time from the stack, we pop the elements out using a for-loop that loops the same amount of times as the inputted String is long. The stack was made very similarly to the queues

that I will soon explain, however of course a stack is last in first out so I had to change a bit (even though this was very similar to what is done and explained in question 4). However what was added was an isEmpty method that simply checks if there are no nodes in the stack's linked list.

The test for this program was simply to run both functions since they let the user input anything.

Question 3

The most special thing with a doubly linked circular list queue compared to the other queues I would say is that it has a next and previous pointer on every node that is in the queue. The field variables that the queue has are a firstNode and a lastNode. The queue class also contains the Node class which is a class with the field variables item, previous and next. In summary how the queue works is that it every Node points to its next and previous Node so to find the Nodes in the middle of the queue if you have several you either have to use the first or last node and use the next and previous node information to go forward and backwards in the node queue. The methods in the queue class are: toString, enqueue, dequeue and printQueue. Printqueue is simply a method to print out all node objects in the queue using the toString method. The toString method goes through every node starting from firstNode and going forward using next and every time it goes through a node it prints it out, if only one node is in the queue it directly prints that one out since it would cause an error looking for a next node in the queue if it doesn't exist. The enqueue method first adds a node to the queue if the queue is empty, if the queue isn't empty it adds a new Node element with the item variable set to the input parameter of the enqueue method, this Node is added to the end of the queue of course. The dequeue first checks if the queue is empty since then there isn't any node to remove. After this it checks if there is only one node in the queue and in this case it simply sets that node to null. If none of these are true the dequeue method removes the lastNode.

The test for this question was to let the user choose if they want to enqueue or dequeue a thing from the queue and they can also choose to print out the queue.

Question 4

This question was very similar to the last one so I will explain the main differences. It wasn't a double linked circular list queue and this means I simply removed the previous pointer since this was the main difference between the two. I let the user choose if they wanted to enqueue or dequeue to the start or the end of the queue by adding a parameter to the enqueue and dequeue methods. These parameters are later used in the

methods where the methods use an if-statement to do different things depending on what this parameter says.

The test for this question was to let the user choose if they want to enqueue or dequeue a thing from the queue and they can also choose to print out the queue. The user also got to enter if they want to enqueue/dequeue the object to the start or the end.

Question 5

The difference between this queue and the previous ones is that in this one the user could remove an element on any given position of the queue. I made this possible by adding a field variable to the queue class that had the length of the queue. Using this length I could easily use a for-loop that looped the queue length minus the index the user wanted to remove minus 1. This element was the one that had to be removed which was done by removing that node and making sure the next pointers were still correct after this. The length of the queue was calculated by adding 1 to the length every time a node was added to the queue and removing 1 from the length every time a node was removed from the queue.

The test for this question was to let the user choose if they want to enqueue or dequeue a thing from the queue and they can also choose to print out the queue. The user also got to enter what index they wanted to remove when choosing to dequeue an object from the queue.

Question 6

For this queue I used the one from question 5 as a base. The difference here was the enqueue method. Every time I added an element I had to use the next pointers and item field variables of the nodes in the queue to make sure the newly added queue got set in the correct place so every node's item variable was in ascending order. The first time a node got added I didn't have to check anything and the node simply got put in the queue. If there was only one node I had two checks, and they were for if the new node item variable was larger or smaller than the firstNode, depending on this I put the new node either in front or after the firstNode. However, in the most normal case, there are several nodes in the queue. I had to go forward one node at a time and use their item variable value stop when I found out that the new Nodes item variable value was smaller than the nextNode that I was checking. And when this happened I put the newNode in front of the bigger one, of course her eI also needed to make sure I was not checking the item variable value for the next node if the next node is null since this would cause an error.

The test for this question was to let the user choose if they want to enqueue or dequeue a thing from the queue and they can also choose to print out the queue. The user also

got to enter what index they wanted to remove when choosing to dequeue an object from the queue.

Question 7

I solved this problem using a stack that I had created. I went through the String of parentheses, that I let the user input, one at a time and if I found an opening parentheses I put it in the stack. If I found a closing parenthesis I popped from the stack to make sure that the popped parenthese was an opening to this kind of parentheses otherwise I knew that the parentheses weren't balanced. In the end I also made sure to check if the stack was empty since otherwise it means that there were still unclosed opening parentheses. The stack was made in a very similar way to the 4th queue since that queue could add and remove items from both sides of it. However what was added was an isEmpty method that simply checks if there are no nodes in the stack's linked list.