The Fundamentals: LabPM

Assume that the data stored in each element of a list/stack/queue is a character or integer (you may choose whichever you think is more convenient)

All assignments should implement tests (**not hard coded tests**).

All JAVA implementations should implement a method returning a string representation of the list/queue where each element is placed between brackets "[x]" and adjacent elements are separated by a comma ","

You are not allowed to use Java library implementations of the data structures (queues, lists etc.) The idea is that you should learn to implement the internals yourself!

**Programming assignments**:

1. In C implement a recursive and an iterative version of a function which reads characters from stdin until a newline character is read and then prints them on stdout in reverse order. Hint: use getchar(), putchar() (or getc(), putc()). For the iterative version you may assume a fixed max length of the input.
2. Implement the above program in JAVA (both iterative and recursive) using one of the ADTs suggested in Algorithms ch. 1.3 for the iterative version.
3. Implement a generic iterable FIFO-queue based on a double linked circular list (see the pdf in the module Course litterature for a description of double linked circular lists). You should print the content of the list after each insertion/deletion of an element.
4. Implement a generic iterable circular linked list which allows the user to insert and remove elements to/from the front and back end of the queue. Be careful when designing the API. You should print the content of the list after each insertion/deletion of an element.
5. Implement a generalized queue which allows the user to remove the *k*th element from the queue. Assume the most recently added element has index 1. You should print the content of the list after each insertion/deletion of an element.
6. Implement an ordered queue based on one of the implementations above. The elements stored in the queue should be integer values. The elements should be ordered at insertion so that all elements are stored in ascending order starting from when you insert the first element and in all following insertions. You should print the content of the list after each insertion/deletion of an element.
7. Implement a program which takes as input a series of parentheses , that is a series of the characters: '(', ')', '[', ']', '{', '}'. The program should check if the parentheses are "balanced" or not. Also show the time and memory complexity of the algorithm.

Upload code, tests and a text motivating your choice of algorithm and how you have calculated the complexities.

**Questions to answer in the seminar, one point per question**:

In the following "*Show*" means that you should be able to project the code/files you have uploaded in Canvas from Canvas from your own computer or the teachers. I.e. you cannot use another file than the one you have uploaded in Canvas. You should be able to explain your code in detail and your design choices.

1. Show and explain the code from assignment 1. Is it easier to implement this function recursively or iteratively? Are there other pros/cons of the two approaches?
2. Show and explain the code from assignment 2, how your tests work and why your tests are sufficient to validate the correctness of your implementation.
3. Show and explain the code from assignment 3, how your tests work and why your tests are sufficient to validate the correctness of your implementation.
4. Show and explain the code from assignment 4, how your tests work and why your tests are sufficient to validate the correctness of your implementation.
5. Show and explain the code from assignment 5, how your tests work and why your tests are sufficient to validate the correctness of your implementation.
6. Show and explain the code from assignment 6, how your tests work and why your tests are sufficient to validate the correctness of your implementation.
7. Show and explain the code from assignment 7, how your tests work and why your tests are sufficient to validate the correctness of your implementation.