FIT1045 Introduction to Algorithms and Programming – Workshop 9

Objectives

The **objectives of this workshop** are:

- To implement recursive algorithms.
- To explore recursive sorting algorithms.

Useful Links:

For this workshop, you may find it useful to review some of the following concepts:

- Function decomposition (Lecture 11)
- Divide and conquer (Lecture 13)
- Recursion (Lecture 14)

Task 0 (To be completed before class):

Write a recursive function `printToN` that takes as input a positive integer *n* and prints the numbers 1 to n.
Once you have done this, create a similar function `printSquares` so that it now prints the first n square numbers.
Finally, create a recursive function `sumOfSquares` that returns (not prints) the *sum* of the first n square numbers.

Note: You should not be using any loops for this task. This task should be solved using recursion only.

Output of printToN (to the console) on input n=4:

```
1
2
3
4
```

Output of printSquares (to the console) on input n=4:

```
1
4
9
16
```

Output of sumOfSquares (as a return value) on input n=4:

```
30
```

Note: A Python tutorial on recursion is available at `http://www.python-course.eu/recursive_functions.php`.

Task 1:

Write a recursive function that takes as input a positive integer *n* and returns a list *L* that contains the factorial values, 1!, 2!, . . . , n!.

For example: If the input is n=5, then the output should be:

```
[1, 2, 6, 24, 120]
```

Task 2:

Modify your recursive function in Task 1 so that given a positive integer *n* as input, it prints (via recursive calls) a line of *k* factorial stars for each integer k in the range 1 to n, and also (as before) returns a list *L* that contains the factorial values, 1!, 2!, . . . , n!. Define a function `printStars` to use within your main function. It should accept an integer k as input and print k stars on a single line.

For example: If the input is n=4, then the following should be printed to the console:

```
*
**
******
************************
```

And the following should be output from the function:

```
[1, 2, 6, 24]
```

Task 3:

A bracket is considered to be any one of the following characters: (,), {, }, [, or]. Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or) occurs to the left of a closing bracket (i.e.,),], or) of the exact same type. There are three types of matched pairs of brackets: [], , and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, [(]) is not balanced because the contents in between and are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket,].

By this logic, we say a sequence of brackets is considered to be balanced if the following conditions are met:

- It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, print YES on a new line; otherwise, print NO on a new line. **Hint:** You can use a stack for this task. That is:

- If the current character is a starting bracket (or { or [then push it to stack.
- If the current character is a closing bracket) or } or] then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.
- After complete traversal, if there is some starting bracket left in stack then "not balanced"

For example: your program may do the following:

```
Enter string of brackets: {[()]}
Output: YES
Enter string of brackets: {[(])}
Output: NO
Enter string of brackets: {{[[(())]]}}
Output: YES
```

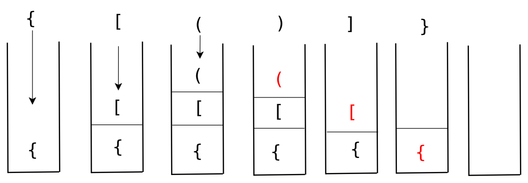

Figure 1: Bracket matching of the string {[()]}

Task 4:

You are required to design a order service system for a resturant. To give equal priority to each customer, the management decided that the food orders will be served in the order they arrive. That is the orders will be served in the order they arrive. Note that at any instance of time orders will be served and new orders will be placed. Your task is to print the orders list after every operation (serve or append).

NOTE: Consider a queue for this implementation.

For example: your program may do the following:

```
Is any order served: No
Please enter the order number:57
The order list is: [57]
Is any order served: No
Please enter the order number:77
The order list is: [57,77]
Is any order served: No
Please enter the order number:24
The order list is: [57,77,24]
Is any order served: Yes
The order list is: [77,24]
Is any order served: No
Please enter the order number:3
The order list is: [77,24,3]
Is any order served: Yes
The order list is: [24,3]
```

Task 5: Optional

Stack is LIFO (last in - first out) data structure, in which elements are added and removed from the same end, called top. In general stack is implemented using array or linked list, but in the current article we will review a different approach for implementing stack using queues. In contrast queue is FIFO (first in - first out) data structure, in which elements are added only from the one side - rear and removed from the other - front. In order to implement stack using queues, we need to maintain two queues A and B. Implement a stack using two queues.

Hint: Refer Figure 2 .

note F,R and Top represent the indices of the queue front and rear and stack top respectively

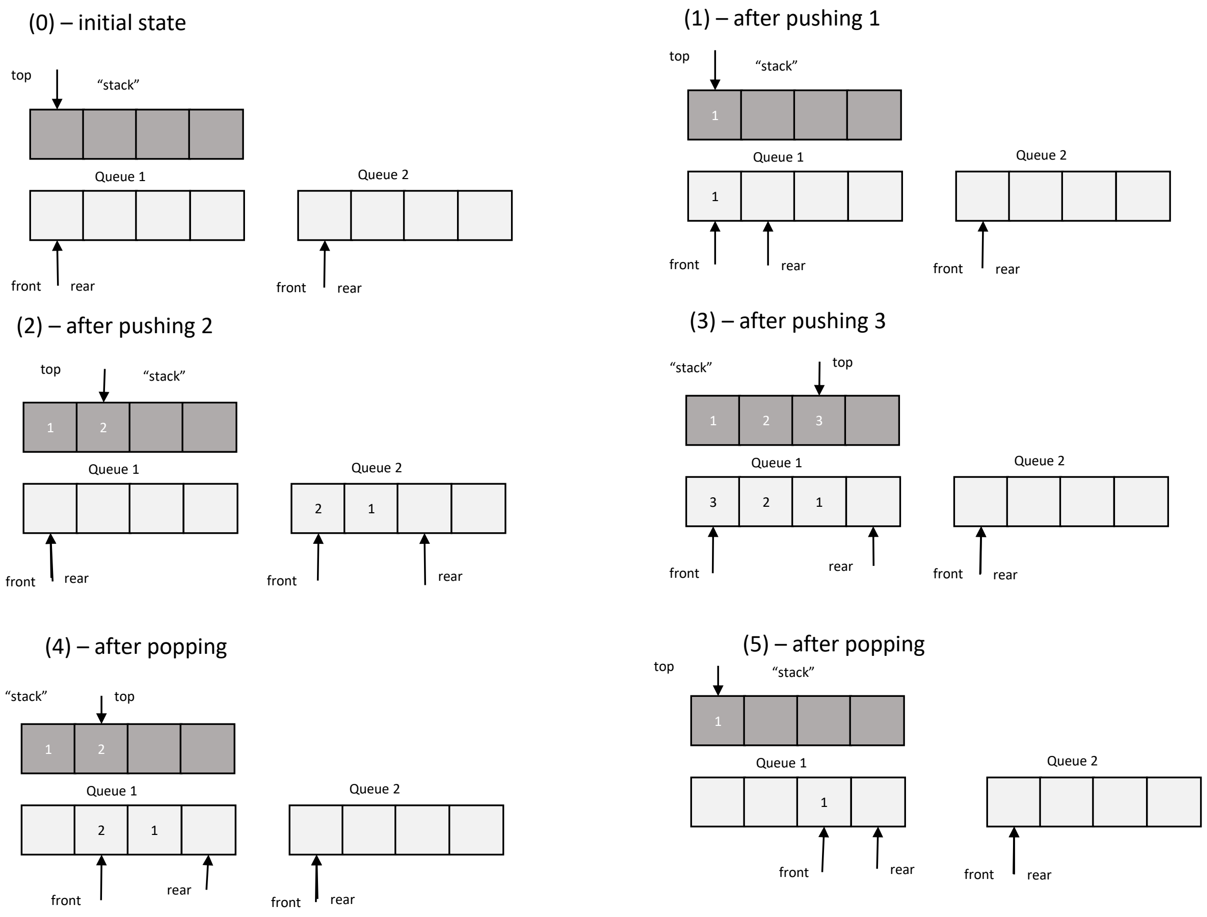

Figure 2: Stack implementation using 2 queues