Problem 1

Write a program to insert data at the bottom of a stack using recursion. You are not allowed to use loop constructs like while, for..etc, and you can only use the following ADT functions on Stack S:

- isEmpty(S)
- push(S)
- pop(S)

Examples

- If `3 -> 2 -> 1` is the stack with 3 at the top, then the output when inserting `4` at the bottom should set the stack to `3 -> 2 -> 1 -> 4` with 3 still at the top.

Hint: The idea of the solution is to write a function that pops a value, and calls itself recursively while passing on both the stack and the data to be inserted as arguments. When the stack is empty the passed data is inserted (base case). After the recursive call, the value popped is inserted back into the stack.

References

- https://www.techiedelight.com/reverse-stack-using-recursion/
- https://afteracademy.com/blog/reverse-a-stack-using-recursion/

Problem 2

Write a program that checks if an expression with different types of brackets is well-formed or not.

Examples

- `{ 2 * ( 3 + 4 + [ 5 / 6 ] ) }` is well-formed
- `{ 2 * ( 3 + 4 + [ 5 / 6 ] ) } * ( 7 + 8 )` is well-formed
- `{ 2 * ( 3 + 4 + [ 5 / 6 ] ) } * ( 7 + 8 ] + 9 )` is **NOT** well-formed

Hints:

1. Go through the characters one-by-one. If the character is an opening bracket (`{` or `(` or `[`) push it to the stack. If a closing bracket is encountered, then the top of the stack must be the **matching** opening bracket in order for the expression to be well-formed.
2. Further pop the opening bracket when a matching closing bracket is found. If the given expression is well-formed, the stack will be empty when the end of the string is reached.

References

- https://www.techiedelight.com/check-given-expression-balanced-expression-not/