Stack - Take Home Problems Solutions

1.

```
function isParenthesisValid(validationString){
var stack = new Stack();
for (var pos=0;pos<validationString.length;pos++){
var currentChar validationString.charAt(pos);
if(currentChar=="("){
stack.push(currentChar);
}else if(currentChar==")"){
if(stack. isEmpty())
return false;
stack.pop();
}
}
return stack. isEmpty();
}
isParenthesisValid("(((()"); // false;
isParenthesisValid("()()"); // true;
```

2.

```
function prec(c) {
            if(c == '^')
                  return 3;
            else if(c == '/' || c=='*')
                  return 2;
            else if(c == '+' || c == '-')
                  return 1;
            else
                  return -1;}
      function infixToPostfix(s) {

            let st = [];
            let result = "";

            for(let i = 0; i < s.length; i++) {
                  let c = s[i];
                  // If the character is
                  // an operand, add it to output string.
                  if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c
>= '0' && c <= '9'))
                        result += c;

                  // If the character is an
                  // '(', push it to the stack.
                  else if(c == '(')
                        st.push('(');

                  // If the character is an ')',
                  // pop and to output string from the stack
```

```
                    // until an '(' is encountered.
                    else if(c == ')') {
                          while(st[st.length - 1] != '(')
                          {
                                result += st[st.length - 1];
                                st.pop();
                          }
                          st.pop();
                    }

                    //If an operator is there
                    else {
                          while(st.length != 0 && prec(s[i]) <=
prec(st[st.length - 1])) {
                                result += st[st.length - 1];
                                st.pop();
                          }
                          st.push(c);
                    }
              }

              // Pop all the remaining elements from the stack
              while(st.length != 0) {
                    result += st[st.length - 1];
                    st.pop();
              }

              document.write(result + "</br>");
        }

        let exp = "a+b*(c^e-f)";
        infixToPostfix(exp);
```

3.

```
function evaluatePostfix(exp)
{
        let stack=[];

        for(let i=0;i<exp.length;i++)
        {
            let c=exp[i];

            // If the character is an operand (number here),
            // push it to the stack.
            if(! isNaN( parseInt(c) ))
            stack.push(c.charCodeAt(0) - '0'.charCodeAt(0));

            //  If the character is an operator, pop two
            // elements from stack apply the operator
            else
            {
                let val1 = stack.pop();
                let val2 = stack.pop();
```

```
                switch(c)
                {
                    case '+':
                    stack.push(val2+val1);
                    break;

                    case '-':
                    stack.push(val2- val1);
                    break;

                    case '/':
                    stack.push(val2/val1);
                    break;

                    case '*':
                    stack.push(val2*val1);
                    break;
                }
            }
        }
        return stack.pop();
}

let exp="521+*7/";
document.write("postfix evaluation: "+evaluatePostfix(exp));
```

4.

```
function max_area_histogram(histogram){
    // Create an empty stack. The stack
    // holds indexes of histogram[] list.
    // The bars stored in the stack are
    // always in increasing order of
    // their heights.
    let stack = []

    let max_area = 0 // Initialize max area
    let index = 0
    while(index < histogram.length){

        // If this bar is higher
        // than the bar on top
        // stack, push it to stack

        if(stack.length == 0 || histogram[stack[stack.length-1]] <=
histogram[index]){
            stack.push(index)
            index += 1
        }

        // If this bar is lower than top of stack,
        // then calculate area of rectangle with
        // stack top as the smallest (or minimum
```

```
        // height) bar.'i' is 'right index' for
        // the top and element before top in stack
        // is 'left index'
        else{
            // pop the top
            let top_of_stack = stack.pop()

            // Calculate the area with
            // histogram[top_of_stack] stack
            // as smallest bar
            let area = (histogram[top_of_stack] *
                (stack.length > 0 ? (index - stack[stack.length-1] - 1) :
index))

            // update max area, if needed
            max_area = Math.max(max_area, area)
        }
    }
    // Now pop the remaining bars from
    // stack and calculate area with
    // every popped bar as the smallest bar
    while(stack.length > 0){

        // pop the top
        let top_of_stack = stack.pop()

        // Calculate the area with
        // histogram[top_of_stack]
        // stack as smallest bar
        let area = (histogram[top_of_stack] *
            (stack.length > 0 ? (index - stack[stack.length-1] - 1) :
index))

        // update max area, if needed
        max_area = Math.max(max_area, area)
    }

    // Return maximum area under the given histogram
    return max_area
}
let hist = [9, 5, 2, 1, 3, 4, 5]
document.write("Maximum area is", max_area_histogram(hist))
```