Tutorial 4: Stacks

ELEC 278: Fundamentals of Information Structures

The learning goals for Tutorial 4 are:

- Practice stacks and see stacks in-application by building a string decoder.

Problem 1. Replicated from `https://leetcode.com/problems/decode-string/`

Given an encoded string, return its decoded string.

The encoding rule is: `k[encoded_string]`, where the `encoded_string` inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4].

The test cases are generated so that the length of the output will never exceed 10^3.

Example 1:
Input: s = "3[a]2[bc]"
Output: "aaabcbc"

Example 2:
Input: s = "3[a2[c]]"
Output: "accaccacc"

Example 3:
Input: s = "2[abc]3[cd]ef"
Output: "abcabccdcdcdef"

Constraints:

- 1 <= s.length <= 30
- s consists of lowercase English letters, digits, and square brackets '[]'.
- s is guaranteed to be a valid input.
- All the integers in s are in the range [1, 300].

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX 1000
// Function to decode the string
char* decodeString(char* s) {
    // to do: complete
}

int main() {
    char s1[] = "3[a]2[bc]";
    char s2[] = "3[a2[c]]";
    char s3[] = "2[abc]3[cd]ef";

    char* s = decodeString(s1);
    printf("Decoded string [%s]: %s\n", s1, s);
    free(s);

    s = decodeString(s2);
    printf("Decoded string [%s]: %s\n", s2, s);
    free(s);

    s = decodeString(s3);
    printf("Decoded string [%s]: %s\n", s3, s);
    free(s);

    return 0;
}
```

Solution.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX 1000
// Function to decode the string
char* decodeString(char* s) {
    int len = strlen(s); // length of the input encoded string
    char temp[MAX]; // string we will use later in decoding
    int tempIndex = 0; // index to use to 'index' temp string above

    int j = 0; // we will use j as a counter in for loops
    int base = 1; // we will use base to make multi-digit integers from strings
    int k = 0; // we will use k to hold the number of times we need to repeat a
   string when decoding

    // We are going to build the stack inside decodeString
    char* stack = (char*)malloc(MAX * sizeof(char)); // assume the max length of
   stack is 1000
    int top = -1; // index to keep track of the top of the stack

    for (int i = 0; i < len; i++) {
        if (s[i] != ']') { // keep adding elements onto the stack until we
   encounter a close bracket
            stack[++top] = s[i]; // this is equivalent to pushing an item to the
   stack
```

```
        } else { // once we encounter a close bracket, we will extract the encoded
    string and decode it

            // 1. Pop from the stack the string within brackets [] onto temp
    string
            tempIndex = 0;
            while (stack[top] != '[') {
                temp[tempIndex++] = stack[top--];
            }
            top--; // Pop the '['; we do not need it anymore
            temp[tempIndex] = '\0'; // terminate temp; temp now holds the reverse
    (because we popped off the stack) of the
                                    // encoded string we need to decode

            // 2. Reverse the extracted string
            for (j = 0; j < tempIndex / 2; j++) {
                char t = temp[j];
                temp[j] = temp[tempIndex - j - 1];
                temp[tempIndex - j - 1] = t;
            }

            // 3. Extract the number k (of repetions) from the stack
            base = 1; k = 0;
            while (top >= 0 && isdigit(stack[top])) {
                k = k + (stack[top--] - '0') * base; // notice how we build a
    multi-digit integer from the characters
                base *= 10;
            }

            // 4. Repeat the string k times and push back to stack
            for (j = 0; j < k; j++) {
                for (int l = 0; l < tempIndex; l++) {
                    stack[++top] = temp[l];
                }
            }
        }
    }
    stack[++top] = '\0';

    return stack;
}

int main() {
    char s1[] = "3[a]2[bc]";
    char s2[] = "3[a2[c]]";
    char s3[] = "2[abc]3[cd]ef";

    char* s = decodeString(s1);
    printf("Decoded string [%s]: %s\n", s1, s);
    free(s);

    s = decodeString(s2);
    printf("Decoded string [%s]: %s\n", s2, s);
    free(s);

    s = decodeString(s3);
    printf("Decoded string [%s]: %s\n", s3, s);
    free(s);
```

```
    return 0;
}
```