1678. Goal Parser Interpretation

Question

You own a Goal Parser that can interpret a string command. The command consists of an alphabet of "G", "()", and/or "(al)" in some order.

The Goal Parser will interpret:

- "G" → "G"
- "()" → "o"
- "(al)" → "al"

The interpreted strings are concatenated in the original order.

Given the string command, return the Goal Parser's interpretation.

Constraints

- 1≤command.length≤100
- command consists of "G", "()", and/or "(al)" in some order.

Inputs

- command: A string consisting of "G", "()", and/or "(al)".

Outputs

- A string representing the interpreted result of the Goal Parser.

Example 1

Input:
command = "G()(al)"

Output:
"Goal"

Explanation:

- "G" → "G"
- "()" → "o"
- "(al)" → "al"
  Concatenated: "Goal".

Example 2

Input:
command = "G()()()()(al)"

Output:
"Gooooal"

Example 3

Input:
command = "(al)G(al)()()G"

Output:
"alGalooG"

Algorithm

1. Initialize an empty dynamically allocated string parsed to store the interpreted result.
2. Traverse the input string command character by character.
   - If the current character is 'G', append 'G' to parsed.
   - If the current character is '(' and the next character is ')', append 'o' to parsed and skip the next character.
   - Otherwise, append "al" to parsed and skip the next three characters (since "(al)" is four characters long).
3. Add the null terminator '\0' to mark the end of the string.
4. Return the dynamically allocated string parsed.

Code

```
#include <stdlib.h>
#include <string.h>

char* interpret(char* command) {
    char* parsed = (char*)malloc(sizeof(char) * 1); // Dynamically allocate
memory
    int index = 0;

    for (int i = 0; command[i] != '\0'; i++) {
        if (command[i] == 'G') {
            parsed = (char*)realloc(parsed, index + 2); // Allocate memory for 'G'
            parsed[index++] = 'G';
        }
        else if (command[i] == '(' && command[i + 1] == ')') {
            parsed = (char*)realloc(parsed, index + 2); // Allocate memory for 'o'
            parsed[index++] = 'o';
            i++; // Skip ')'
        }
        else { // It must be "(al)"
            parsed = (char*)realloc(parsed, index + 3); // Allocate memory for 'al'
            parsed[index++] = 'a';
            parsed[index++] = 'l';
            i += 3; // Skip "al)"
        }
    }
    parsed[index] = '\0'; // Null terminate the string

    return parsed;
}
```

Time Complexity

- O(n): The loop processes each character in command exactly once.

Space Complexity

- O(n): The dynamically allocated memory scales with the length of the output string.

Edge Cases

1. Minimal input size:
    - Example: command = "G" → Output: "G".
    - Example: command = "()" → Output: "o".
2. All components:
    - Example: command = "G(al)" → Output: "al".
3. Maximal input size: Ensure the algorithm handles strings of length 100 efficiently.