3174. Clear Digits

You are given a string `s`.

Your task is to remove **all** digits by doing this operation repeatedly:

- Delete the *first* digit and the **closest non-digit** character to its *left*.

Return the resulting string after removing all digits.

Example 1:

Input: s = "abc"

Output: "abc"

Explanation:

There is no digit in the string.

Example 2:

Input: s = "cb34"

Output: ""

Explanation:

First, we apply the operation on `s[2]`, and `s` becomes `"c4"`.

Then we apply the operation on `s[1]`, and `s` becomes `""`.

Constraints:

- `1 <= s.length <= 100`
- `s` consists only of lowercase English letters and digits.
- The input is generated such that it is possible to delete all digits.

Overview

We are given a string `s` containing letters and digits. Our task is to perform the following operations on every digit of the string:

1. Remove the digit.
2. If there exists any non-digit character to the left of the digit, remove the one closest to it.

As we iterate through each digit in the string and apply these operations, we end up removing all digits along with some non-digit characters. In the end, we will return the final string, after processing and removing all digits.

3174. Clear Digits


```
/*
   Stack
   Time complexity: O(3n)
   Space complexity: O(n)
*/
class Solution {
   public:
      std::string clearDigits(std::string& s) {
         int n=s.size();

         std::stack<char> st;

         // For every character
         for(int i=0;i<n;++i){
            // If there is some characters before it (stack not empty) and
            // it is a digit
            // then remove the non-digit character (pop it from the stack) and pass the next character
            if(!st.empty() && std::isdigit(s[i])) st.pop();

            // If there is no characters before the current character
            // or it is not digit
            // then push it ot the stack and pass the next character
            else st.push(s[i]);
         }

         // Get element from the stack
         // ans will be in reverse order
         std::string ans;
         while(!st.empty()){
            ans.push_back(st.top());
            st.pop();
         }

         // Reverse the answer
         std::reverse(ans.begin(),ans.end());

         return ans;
      }
};
```

3174. Clear Digits

```
/*
   Space optimization: String Stack-Like
   Time complexity: O(n)
   Space complexity: O(1)
*/
```


```
class Solution {
   public:
      std::string clearDigits(std::string& s) {
         int n=s.size();

         // ans string will behave as a stack
         std::string ans;

         // For every character
         for(int i=0;i<n;++i){
            // If there is some characters before it (stack not empty) and
            // it is a digit
            // then remove the non-digit character (pop it from the stack) and pass the next character
            if(!ans.empty() && std::isdigit(s[i])) ans.pop_back();

            // If there is no characters before the current character
            // or it is not digit
            // then push it ot the stack and pass the next character
            else ans.push_back(s[i]);
         }
         return ans;
      }
};
```

3174. Clear Digits

```
/*
   More space optimization: In place: two pointers
   Time complexity: O(2n)
   Space complexity: O(1)
*/
```


```
class Solution {
   public:
      std::string clearDigits(std::string& s) {
         int n=s.size();

         int w=0; // Pointer to write characters
         // For every character
         // r: pointer to read characters
         for(int r=0;r<n;++r){
            // If current character is a digit, reduce the write pointer by 1
            // because the character to the left should be removed
            if(std::isdigit(s[r])) w=std::max(w-1,0);

            // Otherwise, write the character pointer by the read
            // pointer in its correct place
            else s[w++]=s[r];
         }

         // Return all overwritten characters
         return s.substr(0,w);
      }
};
```