You are given a string `s` and an integer `k`, a `k` **duplicate removal** consists of choosing `k` adjacent and equal letters from `s` and removing them, causing the left and the right side of the deleted substring to concatenate together.

We repeatedly make `k` **duplicate removals** on `s` until we no longer can.

Return the final string after all such duplicate removals have been made. It is guaranteed that the answer is unique.

Example 1:

```
Input: s = "abcd", k = 2
Output: "abcd"
Explanation: There's nothing to delete.
```

Example 2:

```
Input: s = "deeedbbcccbdaa", k = 3
Output: "aa"
Explanation:
First delete "eee" and "ccc", get "ddbbbdaa"
Then delete "bbb", get "dddaa"
Finally delete "ddd", get "aa"
```

1209. Remove All Adjacent Duplicates

```
/* Idea:
    keep putting pair of char with count into stack
    if stack top is not equal to new char then start the count from 1
    else the count should be count of top+1
    if the count becomes k, remove element from stack
 */

    string removeDuplicates(string s, int k) {

        stack<pair<char,int>>st;

        for(auto val:s)
        {
            if(!st.empty())
            {
                if(st.top().first != val)
                {
                    st.push({val,1});
                }
                else
                {
                    int count=st.top().second;
                    st.push({val,count+1});

                    if(count+1 == k)
                    {
                        while(!st.empty() && st.top().first==val)
                        {
                            st.pop();
                        }
                    }
                }
            }
            else
            {
                st.push({val,1});
            }
        }

        //making the result string
        string str="";
        while(!st.empty())
        {
            str+=st.top().first;
            st.pop();
        }
        reverse(str.begin(),str.end());
        return str;

    }
```