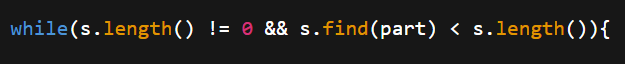
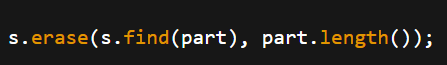


* This defines a class Solution with a public method removeOccurrences.
* The method takes two strings:
  + s — the main string from which you want to remove all occurrences of part.
  + part — the substring that you want to remove repeatedly from s.

This is the **while loop condition**. It keeps running as long as:

* 1. s.length() != 0: the string s is **not empty**.
  2. s.find(part) < s.length(): the substring part **exists somewhere inside s**.
* How does s.find(part) work?
  1. It returns the **starting index of the first occurrence** of part in s.
  2. If part is not found, it returns a special value called string::npos, which is **a very large number** (usually larger than any possible string length).
* So, checking s.find(part) < s.length() means: “**If the substring part is found somewhere inside s, then continue the loop.**”



Inside the loop, this line does the actual removal:

* + s.find(part) returns the starting index of the **first occurrence** of part.
  + part.length() gives the number of characters in the substring part.
  + s.erase(position, length) removes length characters from s, starting at position.
* So this removes **the first occurrence** of part from s.

}

return s;

* After the loop finishes (meaning there are no more occurrences of part in s), the function returns the modified string s.

**Summary of what your code does:**

* It keeps looking for the **first occurrence** of the substring part inside s.
* Removes it immediately when found.
* Keeps repeating this process until part no longer exists in s.
* Then returns the final string with all occurrences of part removed.

**Important note:**

Your code works fine, but calling s.find(part) **twice per loop iteration** (once in the condition and once in erase) is inefficient. You can store the result of s.find(part) in a variable to avoid doing the same search twice. Also, checking for s.length() != 0 is not necessary because if s is empty, find() will return npos anyway.