Recursion - II

Print String in Reverse

Objective: To print the string in reverse.

Base Case: If the string is empty:

                              return;

Reverse print the remaining string using recursion, and print then print the current character.

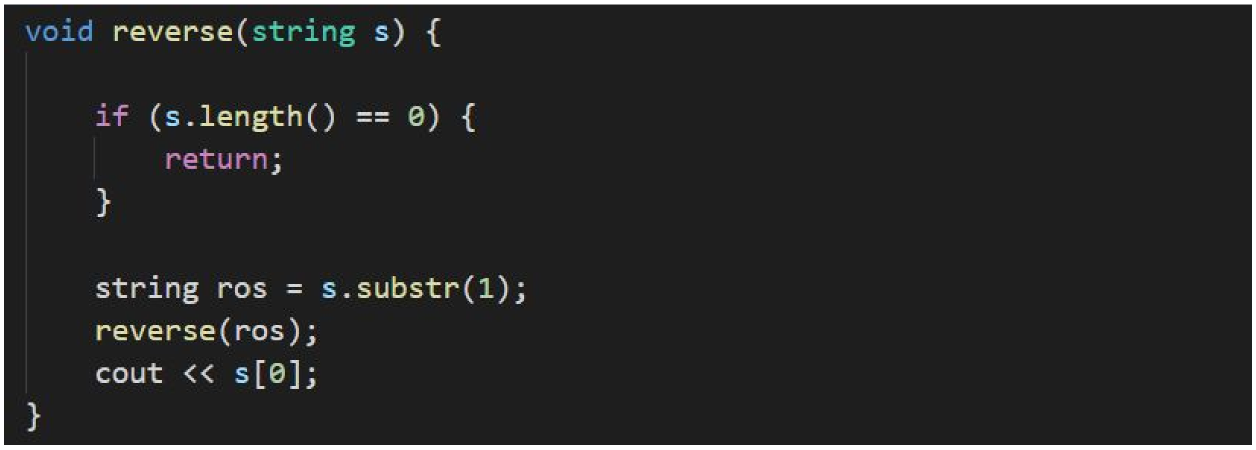
Time Complexity: O(N^2) **[IMP]**

Space Complexity: O(N^2) **[IMP]**

Time complexity will be O(N^2) because s.substr(i) takes O(N) times and it is called O(N) times.

Space complexity will be O(N^2) because s.substr(i) gives a string of O(N) size and it is called O(N) times.

FollowUp: Try to do this in O(N) time and space. Hint: Pass by reference and indices.



```
void reverse(string s) {

    if (s.length() == 0) {
        return;
    }

    string ros = s.substr(1);
    reverse(ros);
    cout << s[0];
}
```

Move all ‘x’ to the end of the string

Base Case: If the string is empty:

return “ “;

If the current character ch is ‘x’, we add the resultant string + ch,

Else we return ch + resultant string

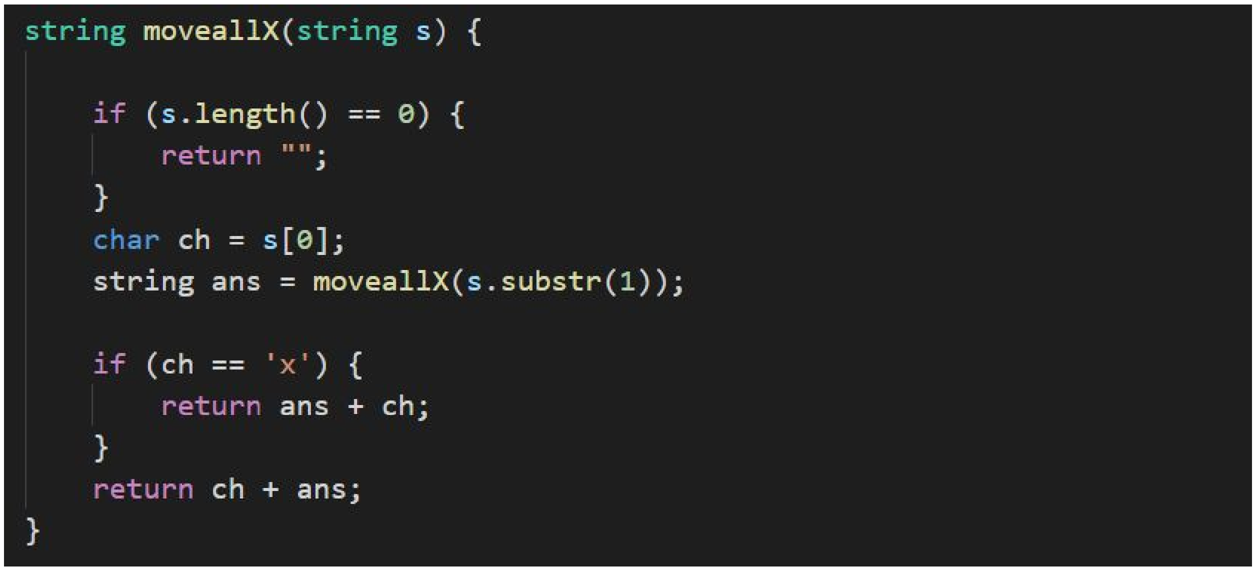
Time Complexity: O(N^2) **[IMP]**

Space Complexity: O(N^2) **[IMP]**

Time complexity will be O(N^2) because s.substr(i) takes O(N) times and it is called O(N) times.

Space complexity will be O(N^2) because s.substr(i) gives a string of O(N) size and it is called O(N) times.

FollowUp: Try to do this in O(N) time and space. Hint: Pass by reference and indices.



```
string moveallX(string s) {

    if (s.length() == 0) {
        return "";
    }
    char ch = s[0];
    string ans = moveallX(s.substr(1));

    if (ch == 'x') {
        return ans + ch;
    }
    return ch + ans;
}
```

Remove Duplicates

Base Case: If the string is empty: //no duplicates

return “”;

If the current character ch is ‘x’, we return resultant string + ch,

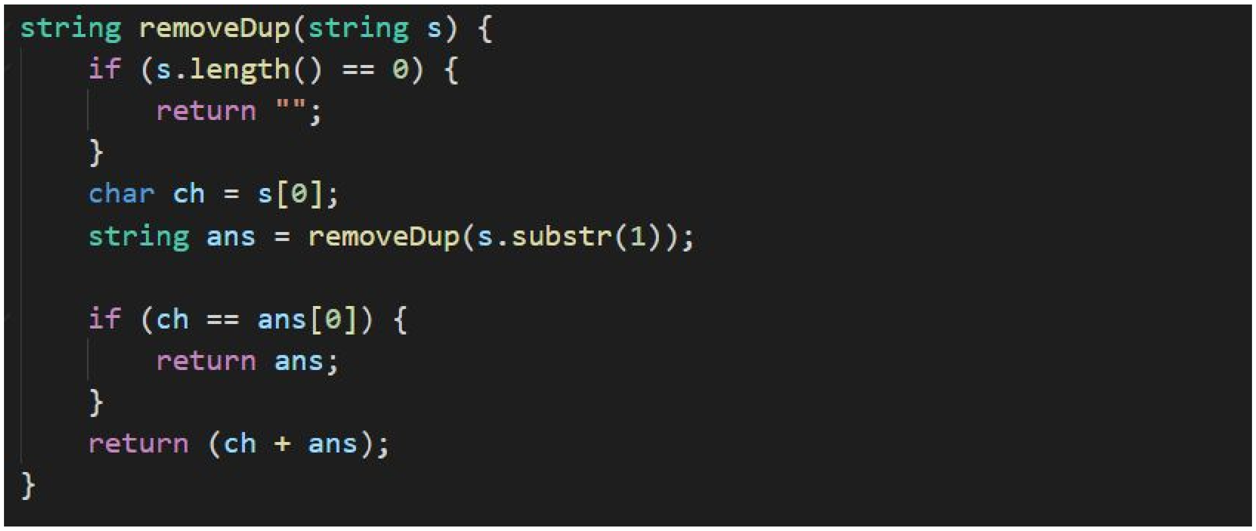
Else we return ch + resultant string

Time Complexity: O(N^2)

Space Complexity: O(N^2)

Time complexity will be O(N^2) because s.substr(i) takes O(N) times and it is called O(N) times.

Space complexity will be O(N^2) because s.substr(i) gives a string of O(N) size and it is called O(N) times.



```
string removeDup(string s) {
    if (s.length() == 0) {
        return "";
    }
    char ch = s[0];
    string ans = removeDup(s.substr(1));

    if (ch == ans[0]) {
        return ans;
    }
    return (ch + ans);
}
```

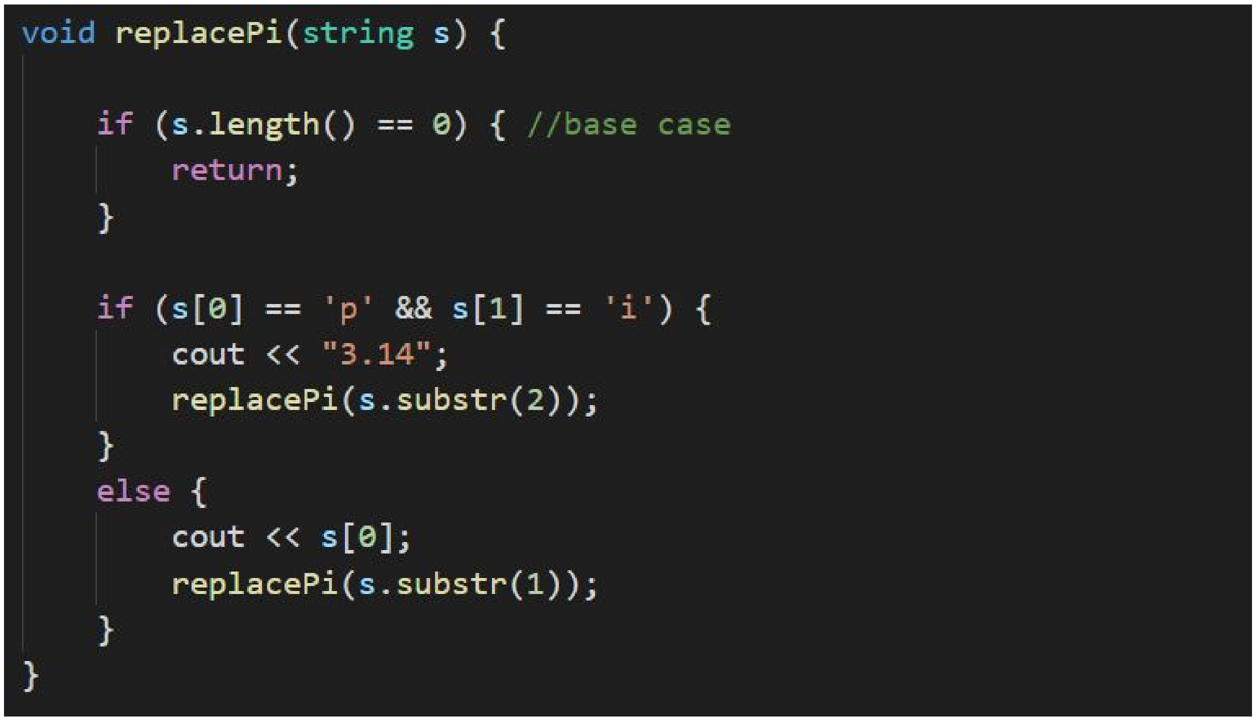
Replace Pi

```
Base Case:    If the string is empty:
                      return "";
If s[0] == 'p' and s[1] == 'i' :
        print("3.14")
else:
        print(s[0])
```

Time Complexity: O(N^2)

Space Complexity: O(N^2)

FollowUp: Try to do this in O(N) time and space. Hint: Pass by reference and indices.



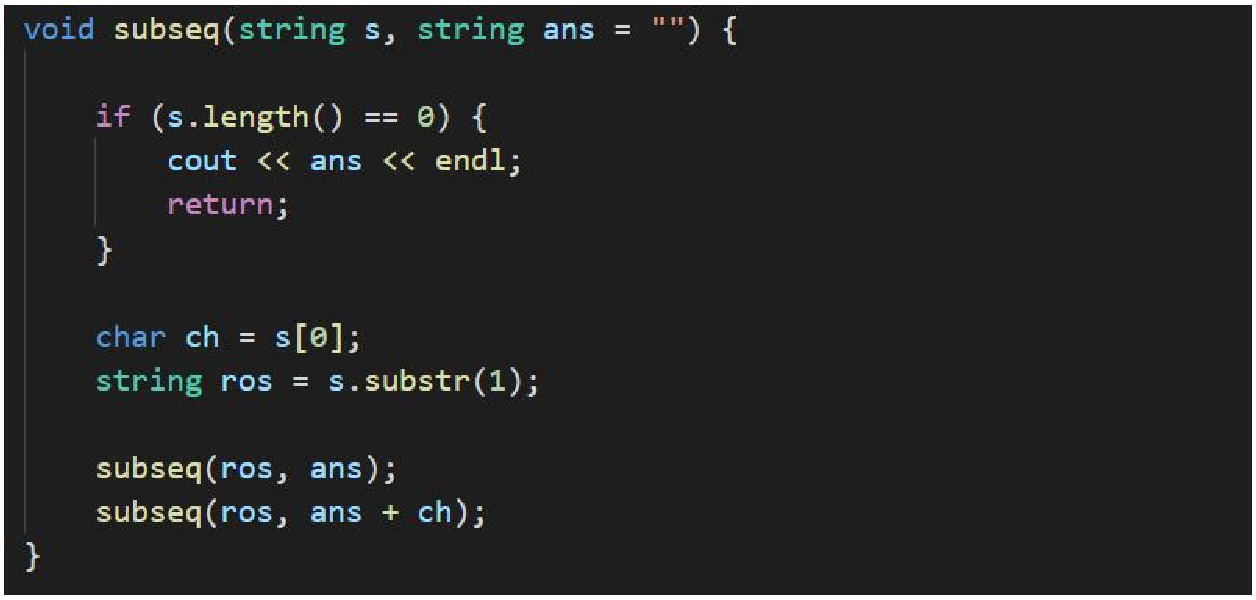
```cpp
void replacePi(string s) {

    if (s.length() == 0) { //base case
        return;
    }

    if (s[0] == 'p' && s[1] == 'i') {
        cout << "3.14";
        replacePi(s.substr(2));
    }
    else {
        cout << s[0];
        replacePi(s.substr(1));
    }
}
```

Print all the subsequences

Objective: For each character, we have two choices, either we include it or not.

Time Complexity: O(2^n)

Space Complexity: O(2^n)

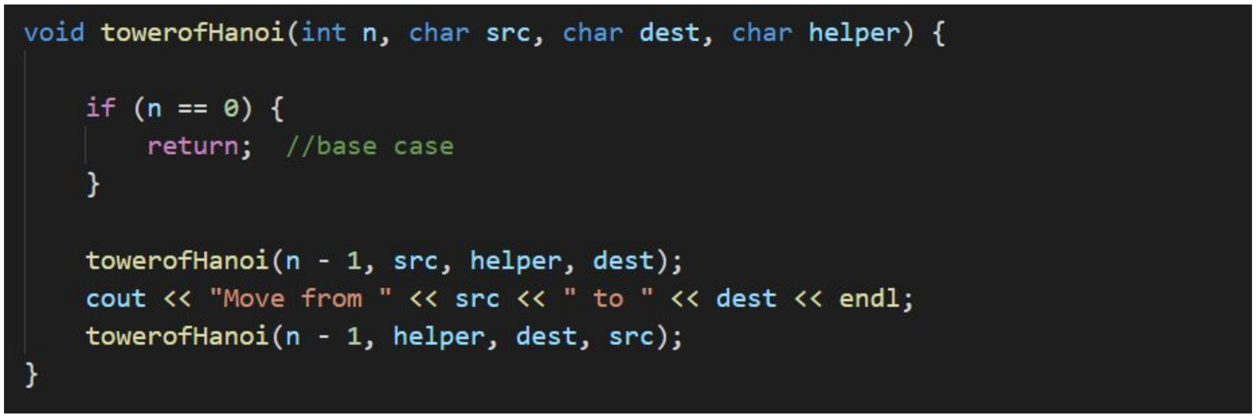


```cpp
void subseq(string s, string ans = "") {

    if (s.length() == 0) {
        cout << ans << endl;
        return;
    }

    char ch = s[0];
    string ros = s.substr(1);

    subseq(ros, ans);
    subseq(ros, ans + ch);
}
```

Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

Idea: move all the n-1 tiles to helper, and then place the remaining tile to dest and then place those n-1 tiles back from helper to dest.
Time Complexity: O(2^n)
Space Complexity: O(1)



```
void towerofHanoi(int n, char src, char dest, char helper) {

    if (n == 0) {
        return;  //base case
    }

    towerofHanoi(n - 1, src, helper, dest);
    cout << "Move from " << src << " to " << dest << endl;
    towerofHanoi(n - 1, helper, dest, src);
}
```

Code for follow-ups:

1. Reverse
2. Replace PI