Remove Spaces (Easy)

Given a string, remove all leading/trailing/duplicated empty spaces.

Assumptions:

- The given string is not null.

Examples:

- “ a” --> “a”
- “ I   love MTV ” --> “I love MTV”

```
void RemoveSpace(string& s){
    int slow = 0, fast = 0;
    int wordCount = 0;
    while(1){
        while(fast < s.size() && s[fast] == ' ')
            fast++;
        if(fast == s.size())
            break;
        if(wordCount > 0)
            s[slow++] = ' ';
        while(fast < s.size() && s[fast] != ' ')
            s[slow++] = s[fast++];
        wordCount++;
    }
    s.resize(slow);
    return;
}
```

Time Complexity: O(n), since we traverse the string once.
//Ok, great!

Right Shift By N Characters (Easy)

Right shift a given string by n characters.

Assumptions

- The given string is not null.
- n >= 0.

Examples

- "abc", 4 -> “cab"

```
string rightShift(string s, int k){
    string result;
    int mid = s.length() - k % s.length();
    // result = s[mid, :] + s[:mid]
    return string(s.begin() + mid, s.end()) + string(s.begin(), s.begin() + mid);
}
```

Time Complexity: O(n), it takes O(n) to return the string.
//Ok, no problem!

Reverse Words In A Sentence I (Medium)

Reverse the words in a sentence.

Assumptions

- Words are separated by single space
- There are no heading or tailing white spaces

Examples

- "I love Google" →" Google love I"

Corner Cases

- If the given string is null, we do not need to do anything.

```
void reverse(string& s, int left, int right){
    char temp;
    while(left < right){
        temp = s[left];
        s[left++] = s[right];
        s[right--] = temp;
    }
}

void reverseWords(string& s){
    int slow = 0, fast = 0;
    if(s.length() == 0)
         return;
    while(fast <= s.length()) {
        if(s[fast] == '\0') {
            reverse(s, slow, fast - 1);
            // reverse the whole string after reversing every words
            reverse(s, 0, fast - 1);
            return;
        }

        if(s[fast] == ' ') {
            reverse(s, slow, fast - 1);
            slow = fast + 1;
        }
        fast++;
    }
    return;
}
```

Time Complexity: O(n), since we traverse the string twice.

//Ok, pretty good!

Reverse Words In A Sentence II (Medium)

Reverse the words in a sentence and truncate all heading/trailing/duplicate space characters.

Examples

- " I love Google " →" Google love I"

Corner Cases

- If the given string is null, we do not need to do anything.

We could use `RemoveSpace()` in problem 1 to remove the useless spaces, and then use `reverseWords()` in problem 3 to get the final result.
Time Complexity: O(n)
//Great! That's awesome!

Add Binary (Medium)

Given two binary strings, return their sum (also a binary string).

Input: a = “11”
b = “1”

Output: “100”

```
string addBinary(string a, string b){
    string result = "";
    int i = a.length() - 1, j = b.length() - 1;
    int sum = 0, carry = 0;
    while(i >= 0 || j >= 0 || carry == 1){
        sum = 0;
        sum += (i >= 0) ? a[i] - '0' : 0;
        sum += (j >= 0) ? b[j] - '0' : 0;
        sum += carry;

        result = char(sum % 2 + '0') + result;
        // compute new carry
        carry = sum / 2;
        i--;
        j--;
    }
    return result;
}
```

Time Complexity: O(m + n), where m is the length of a, n is the length of b.
//Ok, good!

In-order Traversal Of Binary Tree (Medium) **(homework 5)**

Implement an iterative, in-order traversal of a given binary tree, return the list of keys of each node in the tree as it is in-order traversed.

Examples
5
/ \

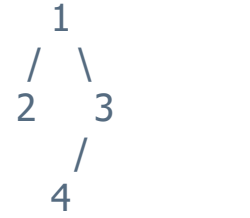

In-order traversal is [1, 3, 4, 5, 8, 11]

Corner Cases

- What if the given binary tree is null? Return an empty list in this case.

How is the binary tree represented?
We use the level order traversal sequence with a special symbol "#" denoting the null node.

For Example:
The sequence [1, 2, 3, #, #, 4] represents the following binary tree:



```
// convert string to int in the vector
vector<int> binaryTreeConvert(vector<string> binaryTreeString){
    vector<int> binaryTree;
    for(auto it : binaryTreeString){
        if(it[0] != '#')
            binaryTree.push_back(stoi(it));
        else
            binaryTree.push_back(-1);
    }
    return binaryTree;
}

// iterative, in-order traversal of binary tree.
vector<string> inOrder(vector<string> binaryTreeString){
    // convert string to int
    vector<int>binaryTree = binaryTreeConvert(binaryTreeString);
    vector<int> resultIdx;
    vector<string> result;
    stack<int> s;
    int n = binaryTree.size();
    int curr = 0;
    while(binaryTree[curr] != -1 || !s.empty()){
        while(binaryTree[curr] != -1 && curr <= n - 1){
            s.push(curr);
            curr = 2 * curr + 1;
        }
        curr = s.top();
        s.pop();
        resultIdx.push_back(curr);
        curr = 2 * curr + 2;
    }
    for(int i = 0; i < resultIdx.size(); i++)
        result.push_back(binaryTreeString[resultIdx[i]]);
    return result;
}
```

//OK, very good!