## LC: 31: Next Permutation

#### Q:

This code solves the "Next Permutation" problem, which asks you to rearrange the numbers into the lexicographically next greater permutation of numbers.

For example:

Input: [1,2,3] → Output: [1,3,2]

Input: [3,2,1] → Output: [1,2,3]

Input: [1,1,5] → Output: [1,5,1]

Algorithm steps:

* Find pivot: Scan right to left, find first number that's smaller than next number
* Find next larger: From right of pivot, find first number larger than pivot
* Swap pivot with that number
* Reverse everything after original pivot position

### PSEUDOCODE:

A screenshot of a computer program

AI-generated content may be incorrect.

### EXAMPLE:

Let me walk through the [1,2,3,6,5,4] example step by step:

**Finding Pivot (First Step)**

Scan from right to left: 4←5←6←3←2←1

Compare each pair:

4 < 5 ✓

5 < 6 ✓

6 > 3 ⚡ (Found pivot!)

Here, 3 is our pivot because it's the first number from right that's smaller than its next number (3 < 6)

Pivot index = 2

**Finding Next Larger Number & Swapping (Second Step)**

Look at numbers right of pivot (6,5,4)

Need smallest number that's larger than pivot (3)

4 > 3 ✓

So swap 3 and 4:

Array becomes: [1,2,4,6,5,3]

**Reversing (Third Step)**

Reverse all numbers after pivot position

Reverse [6,5,3]

Final array: [1,2,4,3,5,6]

### CODE:

class Solution {

public:

    void nextPermutation(vector<int>& A) {

        //find pivot

        int pivot = -1, n=A.size();

        for(int i = n-2; i>=0; i--){

            if(A[i] < A[i+1]){

                pivot = i;

                break;

            }

        }

        if(pivot ==-1){ //if no pivot then return reverse

            reverse(A.begin(), A.end()); //in place changes

            return;

        }

        //2nd step: next larger number/element

        for(int i = n-1; i>pivot; i--){

            if(A[i] > A[pivot]){

                swap(A[i], A[pivot]);

                break;

            }

        }

        //3rd step: reverse(pivot+1 to n-1)

        //reverse(A.begin() + pivot + 1, A.end());

        int i = pivot +1, j=n-1;

        while(i <= j){

            swap(A[i++], A[j--]);

        }

    }

};

### 

## LC:125: Valid Palindrome

#### Q:

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward.

 Converting all letters to lowercase

 Removing all non-alphanumeric characters (keeping only letters and numbers)

 Reading it forward and backward

### PSEUDOCODE:

A screenshot of a computer program

AI-generated content may be incorrect.

### EXAMPLE:

Original: "A man, a plan, a canal: Panama"

Step 1: start=0 ('A'), end=29 ('a')

- both alphanumeric

- 'a' == 'a' ✓

- start=1, end=28

Step 2: start=1 (' '), end=28 ('m')

- skip space at start

- start=2 ('m'), end=28 ('m')

- 'm' == 'm' ✓

- start=3, end=27

And so on...

After cleaning: "amanaplanacanalpanama"

### CODE:

class Solution {

public:

    bool isPalindrome(string s) {

        int start = 0, end = s.length() - 1;

        while(start < end){

            if(!isalnum(s[start])){

                start++;

                continue;

            }

            if(!isalnum(s[end])){

                end--;

                continue;

            }

            if(tolower(s[start]) != tolower(s[end])){

                return false;

            }

            start++;

            end--;

        }

        return true;

    }

};

## 567. Permutation in String

Q

Given two strings s1 and s2, return true if s2 contains a

permutation of s1, or false otherwise.

In other words, return true if one of s1's permutations is the substring of s2

Example 1:

Input: s1 = "ab", s2 = "eidbaooo"

Output: true

Explanation: s2 contains one permutation of s1 ("ba").

### PSEUDOCODE:

A screenshot of a computer program

AI-generated content may be incorrect.

### EXAMPLE:

**Given Inputs:**

* s1 = "ab"
* s2 = "eidbaooo"

We need to check if s2 contains a permutation of s1.

**Step 1: Compute Frequency of s1**

We first count the frequency of each character in "ab":

| **Char** | **Frequency** |
| --- | --- |
| 'a' | 1 |
| 'b' | 1 |

So, the frequency array (freq) for s1 is:

freq = [1, 1, 0, 0, 0, ..., 0] // 'a' at index 0, 'b' at index 1

**Iteration 1: (i = 0, Window = "ei")**

**Step 1: Initialize WindFreq**

We start with an empty frequency array (WindFreq initialized to all zeros).

**Step 2: Build Frequency for Window "ei"**

* First character 'e':
* WindFreq['e' - 'a'] = WindFreq[4] += 1

Now WindFreq = [0, 0, 0, 0, 1, 0, ..., 0]

* Second character 'i':
* WindFreq['i' - 'a'] = WindFreq[8] += 1

Now WindFreq = [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, ..., 0]

**Step 3: Compare WindFreq with freq**

* freq (expected): [1, 1, 0, 0, 0, ..., 0]
* WindFreq (current): [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, ..., 0]
* They **do not match** → Move to next window.

**Iteration 2: (i = 1, Window = "id")**

**Step 1: Initialize WindFreq**

Reset WindFreq to zeros.

**Step 2: Build Frequency for Window "id"**

* First character 'i':
* WindFreq['i' - 'a'] = WindFreq[8] += 1

Now WindFreq = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ..., 0]

* Second character 'd':
* WindFreq['d' - 'a'] = WindFreq[3] += 1

Now WindFreq = [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, ..., 0]

**Step 3: Compare WindFreq with freq**

* freq (expected): [1, 1, 0, 0, 0, ..., 0]
* WindFreq (current): [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, ..., 0]
* They **do not match** → Move to next window.

**Conclusion**

In the first two iterations, the windows "ei" and "id" do not match the required frequency of "ab", so the algorithm moves forward to check the next window.

Would you like me to continue with further iterations? 🚀

### CODE:

    class Solution {

    public:

        bool isFreqSame(int freq1[], int freq2[]){

            for(int i = 0; i<26; i++){

                if(freq1[i] != freq2[i]){

                    return false;

                }

            }

            return true;

        }

        bool checkInclusion(string s1, string s2) {

            int freq[26] = {0};                 //calculate freq

            for(int i=0; i<s1.length(); i++){

                freq[s1[i] - 'a']++;

            }

            int WindSize = s1.length();

            for(int i =0; i<s2.length(); i++){

                int WindIdx = 0, idx = i;

                int WindFreq[26] = {0};

                while(WindIdx < WindSize && idx < s2.length()){

                    WindFreq[s2[idx] - 'a']++;

                    WindIdx++; idx++;

                }

                if(isFreqSame(freq, WindFreq)){

                    return true;

                }

            }

            return false;

        }

    };

## LC: 151. Reverse Words in a String

#### Q:

Given an input string s, reverse the order of the **words**.  
The **words** in s will be separated by **at least one space.**Return *a string of the words in reverse order concatenated by a single space*

**Note** that s may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

#### PSEUDOCODE:

A screenshot of a computer program

AI-generated content may be incorrect.

#### Example

Let's go step by step through the execution of the function with the example **s = "the pen"**.

**Step 1: Initial Input**

We have the string:

s = "the pen"

Length (n) = **7**

**Step 2: Reverse the Entire String**

The function first reverses the entire string s:

Original: "the pen"

Reversed: "nep eht"

Now, s = "nep eht"

**Step 3: Process Each Word**

We iterate over s, extract words, reverse them again, and construct the final output.

**Iteration 1 (i = 0, Extracting "nep")**

* word = ""
* Read characters until a space:
  + word = "n"
  + word = "ne"
  + word = "nep"
* Now word = "nep" and i = 3 (points to the space)
* Reverse "nep" → "pen"
* Append " pen" to ans
* ans = " pen"

**Iteration 2 (i = 4, Extracting "eht")**

* word = ""
* Read characters until a space:
  + word = "e"
  + word = "eh"
  + word = "eht"
* Now word = "eht" and i = 7 (end of string)
* Reverse "eht" → "the"
* Append " the" to ans
* ans = " pen the"

**Step 4: Return the Final Output**

The result currently has an extra space at the start:

" pen the"

To remove this, we use substr(1) to remove the leading space:

Final Output: "pen the"

**Final Output**

reverseWords("the pen") → "pen the"

**Summary of Transformations**

Original: "the pen"

Reversed: "nep eht"

Word Extraction and Reversal:

"nep" → "pen"

"eht" → "the"

Final Output: "pen the"

Let me know if you need further clarification! 🚀

#### Code

class Solution {

public:

    string reverseWords(string s) {

        int n = s.length();

        string ans = "";

        reverse(s.begin(), s.end());

        for(int i = 0; i < n; i++){

            string word = "";

            while(i<n && s[i] != ' '){

                word += s[i];

                i++;

            }

            reverse(word.begin(), word.end());

            if(word.length() > 0){

                ans += " " + word;

            }

        }

        return ans.substr(1);

    }

};