1. Compare Version Numbers

Compare two version numbers *version1* and *version2*. If \*version1\* > \*version2\* return 1; if \*version1\* < \*version2\* return -1;otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character.

The . character does not represent a decimal point and is used to separate number sequences.

For instance, 2.5 is not “two and a half” or “half way to version three”, it is the fifth second-level revision of the second first-level revision.

You may assume the default revision number for each level of a version number to be 0. For example, version number 3.4 has a revision number of 3 and 4 for its first and second level revision number. Its third and fourth level revision number are both 0.

**Example 1:**

Input: version1 = "0.1", version2 = "1.1"  
Output: -1

**Example 2:**

Input: version1 = "1.0.1", version2 = "1"  
Output: 1

**Example 3:**

Input: version1 = "7.5.2.4", version2 = "7.5.3"  
Output: -1

**Example 4:**

Input: version1 = "1.01", version2 = "1.001"  
Output: 0  
Explanation: Ignoring leading zeroes, both “01” and “001" represent the same number “1”

**Example 5:**

Input: version1 = "1.0", version2 = "1.0.0"  
Output: 0  
Explanation: The first version number does not have a third level revision number, which means its third level revision number is default to "0"

**Note:**

1. Version strings are composed of numeric strings separated by dots . and this numeric strings **may** have leading zeroes.
2. Version strings do not start or end with dots, and they will not be two consecutive dots.

**解** 解析出每一段版本号进行比较，假设解析出来的数字分别为n1, n2

* n1 > n2：版本1>版本2
* n1 < n2：版本1 < 版本2
* n1 == n2 ：比较下一段

在比较到末尾时，长的版本号剩余部分如果全部为0，则版本1 == 版本2，否则长的号对应的版本高

为了方便，在每个版本号后面增加了’#’作为结束符

class Solution {  
public:  
 int compareVersion(string version1, string version2) {  
 version1.push\_back('#');  
 version2.push\_back('#');  
 int pos1 = 0, pos2 = 0;  
 int num1 = 0, num2 = 0;  
 while(version1[pos1] != '#' && version2[pos2] != '#'){  
 num1 = 0; num2 = 0;  
 if(version1[pos1] == '.')pos1++;  
 while(version1[pos1] != '#' && version1[pos1] != '.'){  
 num1 = num1 \* 10 + (version1[pos1++] - '0');   
 }  
 if(version2[pos2] == '.')pos2++;  
 while(version2[pos2] != '#' && version2[pos2] != '.'){  
 num2 = num2 \* 10 + (version2[pos2++] - '0');   
 }  
 if(num1 < num2)return -1;  
 else if(num1 > num2)return 1;  
 }  
 if(version1[pos1] == '#' && version2[pos2] == '#'){  
 return 0;  
 }else if(version1[pos1] == '#'){  
 if(allZeros(version2.substr(pos2)))return 0;  
 else return -1;  
 }else{  
 if(allZeros(version1.substr(pos1)))return 0;  
 else return 1;  
 }  
 }  
 bool allZeros(const string &s){  
 int test = 0, pos = 0;  
 while(s[pos] != '#'){  
 if(s[pos] == '.')pos++;  
 else test += s[pos++] - '0';  
 }  
 if(test == 0)return true;  
 else return false;  
 }  
};