Permutation of string in C++

```
#include <iostream>
#include <unordered_map>
using namespace std;

void generate(int cs, int ts, unordered_map<char,
int>& fmap, string asf) {
    if (cs > ts) {
        cout << asf << endl;
        return;
    }

    for (auto entry : fmap) {
        char ch = entry.first;
        int count = entry.second;

        if (count > 0) {
            fmap[ch]--;
            generate(cs + 1, ts, fmap, asf + ch);
            fmap[ch]++;
        }
    }
}

int main() {
    string str = "abc";
    unordered_map<char, int> fmap;

    for (char ch : str) {
        fmap[ch]++;
    }

    generate(1, str.length(), fmap, "");

    return 0;
}
```

Initial Setup:

1. We create an unordered map fmap to store the frequency of each character in the string.
    - fmap = {'a': 1, 'b': 1, 'c': 1}.
2. Call generate(1, 3, fmap, "") to start generating the permutations.

Step-by-Step Execution:

1. **First Call: generate(1, 3, {'a': 1, 'b': 1, 'c': 1}, "")**
    - **cs = 1, ts = 3** (we want a total of 3 characters in the permutation).
    - We iterate over the characters in fmap. Starting with 'a':
        - **Character 'a'**:
            - Count > 0: Use 'a', decrease count in fmap to {'a': 0, 'b': 1, 'c': 1}.
            - Recursively call generate(2, 3, {'a': 0, 'b': 1, 'c': 1}, "a").
2. **Second Call: generate(2, 3, {'a': 0, 'b': 1, 'c': 1}, "a")**
    - **cs = 2, ts = 3**.
    - Iterate again, start with 'a' but it's count 0, so skip it.
    - Move to **character 'b'**:
        - **Character 'b'**:
            - Count > 0: Use 'b', decrease count in fmap to {'a': 0, 'b': 0, 'c': 1}.
            - Recursively call generate(3, 3, {'a': 0, 'b': 0, 'c': 1}, "ab").
3. **Third Call: generate(3, 3, {'a': 0, 'b': 0, 'c': 1}, "ab")**
    - **cs = 3, ts = 3**.
    - Iterate again, starting with 'a' and 'b', both of which have counts 0, so skip them.
    - Move to **character 'c'**:
        - **Character 'c'**:
            - Count > 0: Use 'c', decrease count in fmap to {'a': 0, 'b': 0, 'c': 0}.
            - Recursively call generate(4, 3, {'a': 0, 'b': 0, 'c': 0}, "abc").
4. **Base Case: generate(4, 3, {'a': 0, 'b': 0, 'c': 0}, "abc")**
    - **cs = 4, ts = 3**: We've reached the required length of 3 characters.

	○ **Output the permutation**: "abc". 5. **Backtrack**: ○ Return to the previous state where fmap = {'a': 0, 'b': 1, 'c': 1} and asf = "a". ○ Restore the count of 'c' and continue the loop. **Second Iteration of First Call:** 1. **Second Character 'b' in First Call:** ○ **Character 'b'**: ▪ Count > 0: Use 'b', decrease count in fmap to {'a': 1, 'b': 0, 'c': 1}. ▪ Recursively call generate(2, 3, {'a': 1, 'b': 0, 'c': 1}, "b"). 2. **Second Call: generate(2, 3, {'a': 1, 'b': 0, 'c': 1}, "b")** ○ **cs = 2, ts = 3**. ○ Skip 'b' (count 0) and move to 'a': ▪ **Character 'a'**: ▪ Count > 0: Use 'a', decrease count in fmap to {'a': 0, 'b': 0, 'c': 1}. ▪ Recursively call generate(3, 3, {'a': 0, 'b': 0, 'c': 1}, "ba"). 3. **Third Call: generate(3, 3, {'a': 0, 'b': 0, 'c': 1}, "ba")** ○ **cs = 3, ts = 3**. ○ Skip 'a' and 'b', move to 'c': ▪ **Character 'c'**: ▪ Count > 0: Use 'c', decrease count in fmap to {'a': 0, 'b': 0, 'c': 0}. ▪ Recursively call generate(4, 3, {'a': 0, 'b': 0, 'c': 0}, "bac"). 4. **Base Case: generate(4, 3, {'a': 0, 'b': 0, 'c': 0}, "bac")** ○ **Output the permutation**: "bac".
Output:- cba cab bca bac acb abc	