Word Break In C++

```
#include <iostream>
#include <unordered_set>
#include <vector>
using namespace std;

bool solution(string sentence,
unordered_set<string>& dict) {
    int n = sentence.length();
    vector<int> dp(n, 0);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            string word = sentence.substr(j, i - j
+ 1);
            if (dict.find(word) != dict.end()) {
                if (j > 0) {
                    dp[i] += dp[j - 1];
                } else {
                    dp[i] += 1;
                }
            }
        }
    }

    cout << dp[n - 1] << endl;
    return dp[n - 1] > 0;
}

int main() {
    unordered_set<string> dict = {"i", "like",
"pep", "coding", "pepper", "eating",
"mango", "man", "go", "in", "pepcoding"};
    string sentence =
"ilikepeppereatingmangoinpepcoding";

    cout << boolalpha << solution(sentence,
dict) << endl;

    return 0;
}
```

Step-by-Step Dry Run

Input:

- sentence = "ilikepeppereatingmangoinpepcoding"
- dict = {"i", "like", "pep", "coding", "pepper", "eating", "mango", "man", "go", "in", "pepcoding"}

Initialization:

- dp = [0, 0, 0, ..., 0] (length of 39 for the sentence "ilikepeppereatingmangoinpepcoding").
- We iterate over each character of the sentence.

Loop Execution:

- **At i = 0** (i.e., the first character 'i'):
  - Substring: "i" (valid word found in the dictionary), so dp[0] = 1.
- **At i = 1** (i.e., the second character 'l'):
  - Substrings: "il" (not in dictionary), "l" (not in dictionary).
- **At i = 2** (i.e., the third character 'i'):
  - Substrings: "ili" (not in dictionary), "li" (not in dictionary), "i" (valid word, so dp[2] += dp[0] = 1).
- **At i = 3** (i.e., the fourth character 'k'):
  - Substrings: "like" (valid word found in dictionary), so dp[3] = dp[0] + 1 = 1.
- **At i = 4** (i.e., the fifth character 'e'):
  - Substrings: "like" (valid word found in dictionary), "i" (valid word found in dictionary), so dp[4] = 1 (from "like") and dp[4] = dp[3] + 1 = 2 (from "i").
- **And so on...** The algorithm continues checking for substrings, updating the dp array for each valid word found.

Output:

- After filling the dp array, dp[38] contains the number of ways to segment the entire sentence. It turns out that there are 4 ways to split the sentence into valid words:
  - "i like pepper eating mango in pep coding"
  - "i like pepper eating mango in pepcoding"

	○ "i like pep per eating mango in pep coding" ○ "i like pep per eating mango in pepcoding" So, the function will print 4 and return true because the sentence can indeed be segmented.
Output:- 4 true	