

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 =
"ilikepeppereatingmangoinepcoding";

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

    return 0;
}
```

Iterative Tabular Dry Run for Word Break Problem

We will dry-run the **fixed DP approach** using the sentence: **"ilikepeppereatingmangoinepcoding"**

Dictionary:

```
{"i", "like", "pep", "coding", "pepper",
"eating", "mango", "man", "go", "in",
"pepcoding"}
```

Step 1: Define DP Table

- Let $dp[i]$ represent whether the substring $sentence[0 \dots i-1]$ can be segmented.
- We initialize $dp[0] = \text{true}$ (empty string is always valid).
- We will iterate over all positions i and check all possible substrings $sentence[j \dots i-1]$ to see if they exist in the dictionary and if $dp[j]$ is true.

Step 2: Iterative Dry Run in Tabular Form

i	Substring (sentence[0...i-1])	Valid Segment Found?	dp[i] Value
0	""	Base case	true
1	"i"	✓ ("i" in dict)	true
2	"il"	✗	false
3	"ili"	✗	false
4	"ilik"	✗	false
5	"ilike"	✓ ("like" in dict, dp[1] is true)	true
6	"ilikep"	✗	false
7	"ilikepe"	✗	false
8	"ilikepep"	✓ ("pep" in dict, dp[5] is true)	true

		in dict, dp[5] is true)	
9	"ilikepepp"	✗	false
10	"ilikepeppe"	✗	false
11	"ilikepepper"	✓ ("pepper" in dict, dp[5] is true)	true
12	"ilikepeppere"	✗	false
13	"ilikepepperea"	✗	false
14	"ilikepeppereat"	✗	false
15	"ilikepeppereati"	✗	false
16	"ilikepeppereatin"	✗	false
17	"ilikepeppereating"	✓ ("eating" in dict, dp[11] is true)	true
18	"ilikepeppereatingm"	✗	false
19	"ilikepeppereatingma "	✗	false
20	"ilikepeppereatingma n"	✓ ("man" in dict, dp[17] is true)	true
21	"ilikepeppereatingma ng"	✗	false
22	"ilikepeppereatingma ngo"	✓ ("mango" in dict, dp[17] is true)	true
23	"ilikepeppereatingma ngoi"	✗	false
24	"ilikepeppereatingma ngoin"	✓ ("in" in dict, dp[22] is true)	true

	25	"ilikepeppereatingma ngoinp"	✗	false
	26	"ilikepeppereatingma ngoinpe"	✗	false
	27	"ilikepeppereatingma ngoinpep"	✓ ("pep" in dict, dp[24] is true)	true
	28	"ilikepeppereatingma ngoinpepc"	✗	false
	29	"ilikepeppereatingma ngoinpepc"	✗	false
	30	"ilikepeppereatingma ngoinpepcod"	✗	false
	31	"ilikepeppereatingma ngoinpepcodi"	✗	false
	32	"ilikepeppereatingma ngoinpepcodin"	✗	false
	33	"ilikepeppereatingma ngoinpepcoding"	✓ ("pepcodi ng" in dict, dp[24] is true)	true
	<p>Step 3: Final dp Array</p> <p>[T T F F F T F F T F F T F F F F F T F F T F T F T F F T F F F F F T]</p> <p>Since $dp[n] = dp[33] = \text{true}$, we conclude that the sentence can be segmented into words from the dictionary.</p>			
	<p>Output:-</p> <p>4</p> <p>true</p>			