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

const int ALPHABET_SIZE = 26;

// trie node

struct TrieNode
{
        struct TrieNode* children[ALPHABET_SIZE];
        // EndOfWord es true if the node represents
        // end of a word
        bool isEndOfWord;
};

// Returna un nuevo nodo (initialized to NULLs)
struct TrieNode* getNode(void)
{
        struct TrieNode* pNode = new TrieNode;
        pNode->isEndOfWord = false;
        for (int i = 0; i < ALPHABET_SIZE; i++)
                pNode->children[i] = nullptr;
        return pNode;
}

// If not present, inserts key into trie
// If the key is prefix of trie node, just
// marks leaf node

void insert(struct TrieNode* root, string key)
{
        //Agrega datos en Key[i]
        struct TrieNode* var = root;
        for (int i = 0; i < key.length(); i++)
        {
                int index = key[i] - 'a';
                if (!var->children[index]) {
                        var->children[index] = getNode();
                }
                var = var->children[index];
                cout << "\n";
                cout << "\nED Key: " << key[i] << " - a" << " -- " << "var:
" << var;
                cout << "\nED Key: " << index << " -- " << "var: " << var;
                cout << "\n----------\n\n";
```

```
                var->isEndOfWord = true;
        }

}

// Returns true if key presents in trie, else
// false

bool search(struct TrieNode* root, string key)
{
        struct TrieNode* var = root;
        for (int i = 0; i < key.length(); i++)
        {
                int index = key[i] - 'a';
                if (!var->children[index])
                        return false;
                var = var->children[index];
        }
        return (var->isEndOfWord);
}

// Driver
int main()
{
        // Input keys (use only 'a' through 'z'
        // and lower case)
        string keys[] = { "eduardo", "a", "there",
                                        "answer", "any", "by",
                                        "bye", "their" };
        int n = sizeof(keys) / sizeof(keys[0]);

        //Poner los nodos nullptr
        struct TrieNode* root = getNode();

        // Construct trie
        for (int i = 0; i < n; i++)
                insert(root, keys[i]);

        // Search for different keys
        char output[][32] = { "Not present in trie", "Present in trie" };

        // Search for different keys
        cout << "edu" << " --- " << output[search(root, "eduardo")] << endl;
        cout << "these" << " --- " << output[search(root, "these")] << endl;
        cout << "their" << " --- " << output[search(root, "their")] << endl;
        cout << "thaw" << " --- " << output[search(root, "thaw")] << endl;
```

```
        cin.get();
        cin.ignore();
        return 0;
}
```