// AddAndSearch.cpp : 定义控制台应用程序的入口点。

//

/\* 第二周 第四题

4) 设计一个支持以下两种操作的数据结构：

void addWord(word)

bool search(word)

search(word) 可以搜索文字或正则表达式字符串，字符串只包含字母 . 或 a-z 。 . 可以表示任何一个字母。

示例:

addWord("bad")

addWord("dad")

addWord("mad")

search("pad") -> false

search("bad") -> true

search(".ad") -> true

search("b..") -> true

编译环境：Visual Studio 2015

作者：杨政权

\*/

//实验代码

#include "stdafx.h"

#include <iostream>

using namespace std;

const int MAX\_CHILD = 26; //26个英文字母

//定义一个字典树类

class TrieNode {

public:

int count;

TrieNode\* child[MAX\_CHILD];

TrieNode() {

for (int i = 0; i < 26; i++)

child[i] = NULL;

count = 0;

}

};

//具体实现

class WordDictionary {

public:

WordDictionary() {

root = new TrieNode();

}

//添加单词

void addWord(string word) {

//为空直接返回

if (root == NULL || word.size() == 0) {

return;

}

int len = word.size();

TrieNode\* t = root;

int i = 0;

while (i<len) {

if (t->child[word[i] - 'a'] == NULL) {

TrieNode\* temp = new TrieNode();

t->child[word[i] - 'a'] = temp;

t = t->child[word[i] - 'a'];

}

else {

t = t->child[word[i] - 'a'];

}

i++;

}

t->count = 1;

}

//查询函数

bool search(string word) {

bool ret=search(word, root, 0);

return ret; //返回查询结果

}

bool search(string &word, TrieNode \*p, int i) {

if (i == word.size()) {

return p->count;

}

//当遇到字符为'.'的时候，一个 . 可以代表一个任何的字母

//这里用递归的方式判断输入字符串是否存在

if (word[i] == '.') {

for (auto a : p->child) {

if (a && search(word, a, i + 1)) {

return true;

}

}

return false;

}

else {

return p->child[word[i] - 'a'] && search(word, p->child[word[i] - 'a'], i + 1);

}

}

private:

TrieNode \*root;

};

int main()

{

WordDictionary WordDict;

bool search1, search2, search3,search4;

//添加数据

WordDict.addWord("bad");

WordDict.addWord("dad");

WordDict.addWord("mad");

//查询

search1= WordDict.search("pad");

search2= WordDict.search("bad");

search3 = WordDict.search(".ad");

search4 = WordDict.search("b..");

//打印查询结果

//用boolalpha 否则默认是将true=1 false=0 打印

cout << boolalpha << search1 << endl;

cout << boolalpha << search2 << endl;

cout << boolalpha << search3 << endl;

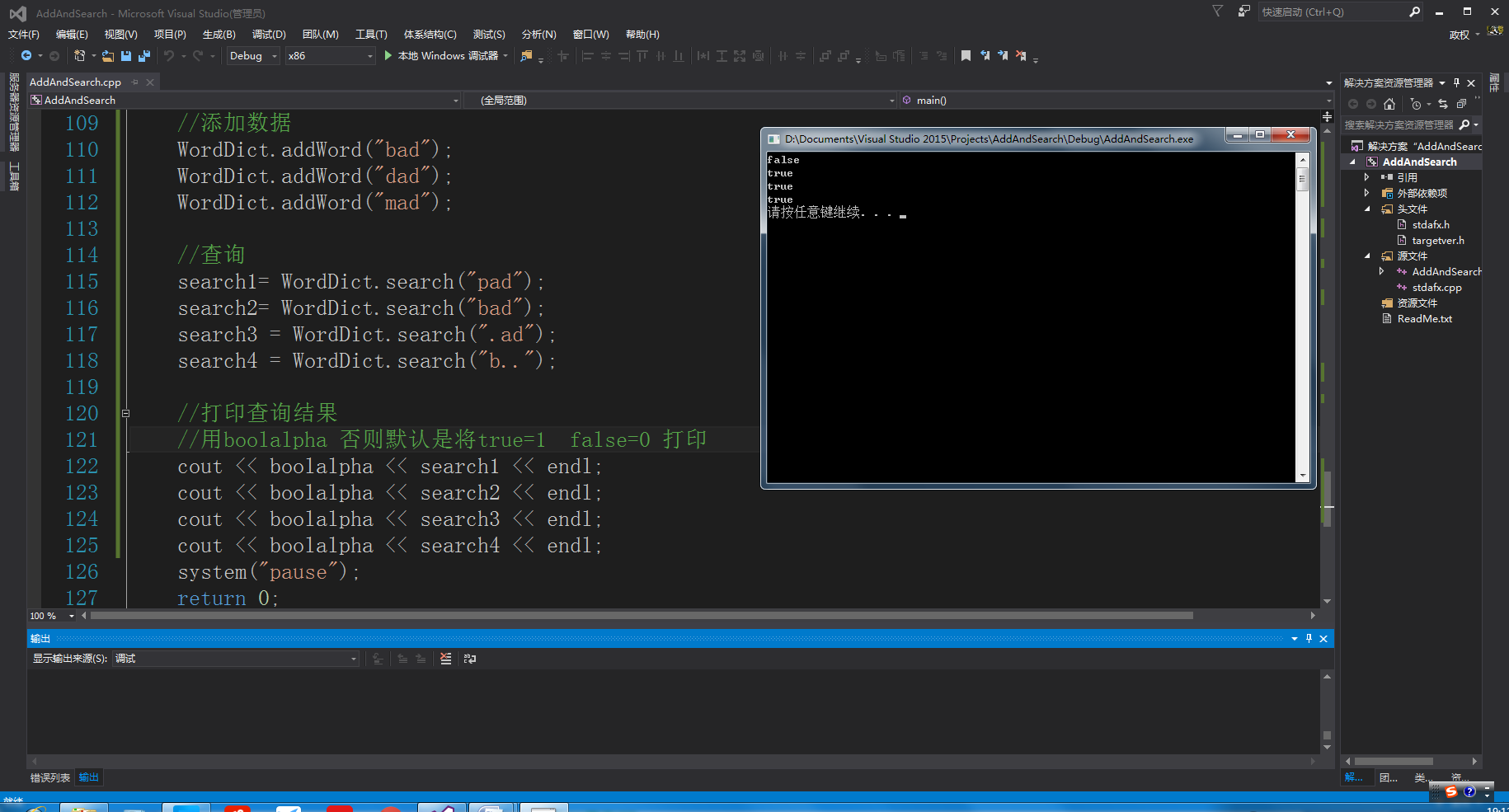
cout << boolalpha << search4 << endl;

system("pause");

return 0;

}

//程序运行截图



//实验总结：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* 这个题目主要用了字典树的知识，

\*\*\* 之前没怎么用过这个，

\*\*\* 好在网上例子很多。

\*\*\* 看了一些资料，

\*\*\* 很快就能上手

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/