# 1 项目简介

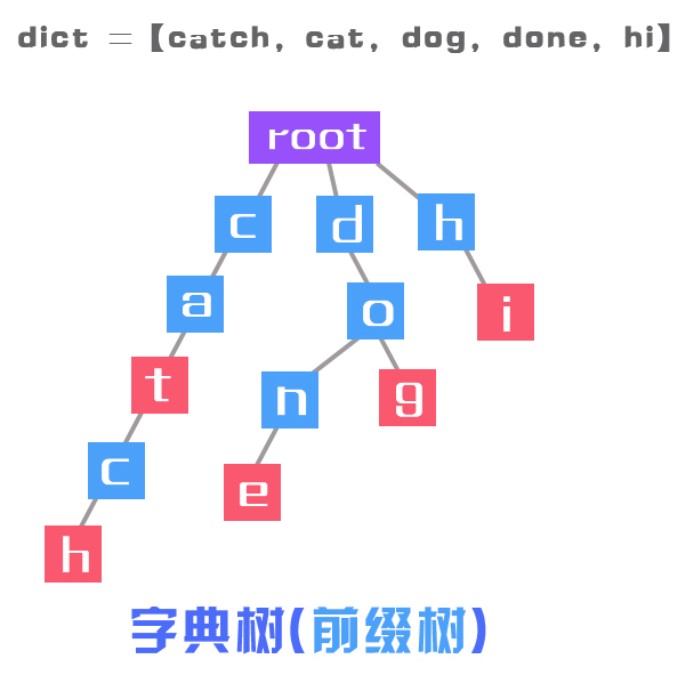
建立一个文本文件，文件名由用户用键盘输入，输入一个不含空格的关键字，统计输出关键字在文本中的出现次数。

# 2 程序说明

## 2.1 数据结构设计

解决这一问题最直接粗暴的思路莫过于直接将控制台输入的字符存储在字符串中，之后遍历字符串计数关键词数量。可是，这样一来，一方面在处理大量文本时程序的空间开销会非常大，而且搜索的时间复杂度也会极高。为了避免上述问题，本程序采用

**“字典树”**作为存储结构。下面先首先简要介绍本程序所用的字典树的概念：

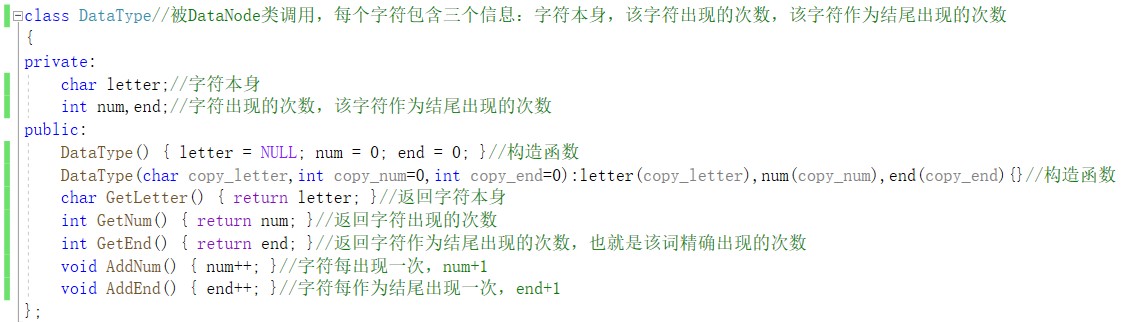


字典树（Trie Tree）是一种树形结构，是一种哈希树的变种。典型应用于但不局限于统计、排序和保存大量字符串。其优点是：通过公用字符串公共的、重叠前缀部分来减少无实际效益的重复存储与重复检索。相比于直接用字符串数组来解决问题的方案，字典树既能够极大限度地减少存储空间浪费，又能够极大地提高检索速度。

例如：若用户输入catch, cat, dog, done, hi 这5个单词，如左侧的字典树所示，单词中“ca” 、“do”等重复的前缀部分能够公用同一个存储空间，而且，当输入关键词进行比较适时，数组的时间复杂度是和输入的文本字符串的长度成正比的，而字典树的搜索的时间复杂度是和关键词的单词长度，或者说，树的深度成正比的，这就体现了字典树的巨大优势。例如：在1000个字符的文本中寻找其中包含某5个字符的关键词，直接遍历数组需要1000多次，而字典树只需要搜索5次，这是巨大的时间差。而且，英文单词的长度有长度上限，日常常见的英文单词比较长的也只是10多个字母，这样一来，利用字典树搜索关键词的时间复杂度几乎是一个常数级别的算法，具有显著优势。

## 2.2 类结构设计

①DataType类：



DataType类作为一种自定义数据类型，包含字符本身的种类，字符出现的数量、字符作为结尾出现的次数，即以该字符为结尾的单词出现的数量这三方面的学习。可以说，DataType类是专门为字典树节点的数据定义的类型，为字典树节点功能的实现提供最基础的接口。

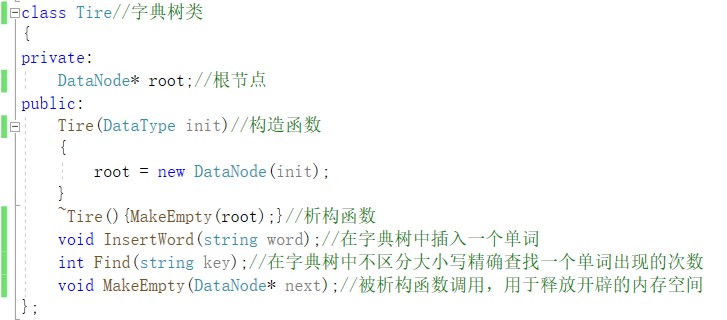
## ②字典树节点DataNode类：

## 屏幕截图 2021-12-18 221056

## 一方面，字典树节点DataNode类需要包含的信息是节点本身字符的性质，这一点由先前定义的DataType类满足，另一方面，字典树节点需要提供当前字符与后续字符的连接关系，或者通俗来讲，出现的单词中，前一个字母后面出现的先一个字母是什么。而本程序设定的是不区分大小写的精确检索，而小写英文字母数量是固定的26个，因此，使用大小为26的指针数组来表述单词中各字母间的前后关系。采用指针数组的另一个好处是：利用26个字母的ASCII码可以直接通过输入的字母定位到后面该由哪个指针指向下一个字母，这样一来直接通过赋值操作就可以实现字典树节点的节点插入。

③Trie字典树类：

带根节点的字典树类基于上面两个类直接完成插入单词、检索单词的功能，这也是本程序的核心功能。下文将对类函数的**核心代码**进行详细说明：



1. InsertWord函数：

功能：在字典树中插入一个单词。具体来看：从字典树的根节点开始，遍历单词中的每一个字母，若字典树中各层字母已存在，则num++即可，否则新建节点。若该字符作为结尾出现，则end++，这也是该词出现的次数。

1. Find函数：

功能：在字典树中不区分大小写精确查找一个单词出现的次数。具体来看：从字典树的根节点开始，通过循环结构遍历单词中的每一个字母。若该单词的两个字母间的联系在字典树中不存在，则该单词在字典树中不存在，返回0即可。若当前字母与后一个字母的关联存在，则移动到下一个字母节点继续此判断。如果到了单词词尾，则返回end值，即为该单词精确出现的次数，此时循环结束。

1. 析构函数：利用递归思路，依照字典树各层递归检索，在所有后续字母都删除完成后释放该字母本身的内存空间，即可最终删除字典树全部节点。

# 3. 程序实现流程

本程序只是模拟一个简单的关键词检索过程，其流程是直线型的顺序结构，因此，这里提供简要流程的文字说明：

首先实例化字典树类，输入并判断用户想要建立的文件名，若文件成功建立则接着由用户输入一段英文文本，将这段文本写入先前建立的文件。此后由用户输入符合规范的检索关键词并暂存。之后执行读文件操作，即在遍历文件的过程中建立字典树。接着在字典树中匹配用户输入的检索关键词，返回最后一个字母的end值即为该单词在不区分大小写的情况下，在文件中出现的次数。最终输出关键词输出次数，程序结束。

在这个流程中，需要特别注意的几点如下：

1. 对用户输入的文件名的规范性判断：

首先，程序的运行平台主要是windows系统，而windows系统文件命名的基本规范是：不能在文件名中出现 \ / : \* ? \ " < > | 以及空格这些字符，如果不加判断与处理，一旦用户输入的文件名出现了上述字符，会影响程序的正常运行。因此，通过设定stopwords，即上述字符的集合，一旦在用户输入的文件名中出现了stopwords字符，即提示用户重新输入。

此功能通过while循环结构实现：



此外，该循环中还考察了用户输入的文件名的后缀是否是符合题意的 .txt ，这一点可以利用string自带的字符串截取字符 .之后的内容，判断是否是txt，这样一来，不仅考差了文件后缀是否是txt，同时也可以判断用户输入的文件名中是否包含了多个字符 点，造成文件后缀错误。

2.对用户输入的英文进行规范化清洗之后再建立字典树：

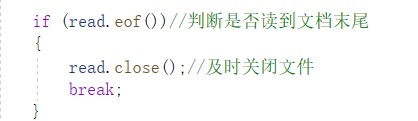
一般的英文文本包含了大量的标点符号，以及字母的大小写，有时甚至在文本中存在着空行 、中文、数字等其他非英文字符。而本程序在将用户在控制台中输入的文本写入文件时，还是最大限度地保留了文本原样。但是，在建立字典树时，根据前文的思路，字典树中仅仅应该存在英文字母以及英文字母间的前后位置关系，因此，在读取文件建立字典树的时候，每读一个单词，必须对该单词进行规范化清洗。将大写字母统一转成小写形式，并忽略空行以及其他非英文字母，从而保证建立字典树时不会出错。当然，对用户输入的目标检索关键词尤其需要这种规范化清洗。

此功能通过函数Wash来实现：

# 屏幕截图 2021-12-18 231006

额外需要说明的几点是：

1.在操作文件时，要特别注意文件是否成功建立、是否成功打开、是否读取到文件结尾，是否及时关闭了文件。这些操作对文件的稳定具有重要意义。

本程序通过调用头文件<fstream>中的ifstream以及ofstream来实现对文件的C++方式读写。当执行建立或打开文件的操作后，首先通过判断file.fail()函数是否被触发来判断文件建立或打开操作是否成功。若触发，则输出操作失败的提示语，并通过exit（0）结束程序。是否读取到文件末尾，则通过下面的程序段来执行。

1. 用户给出的英文字段有两种方式：一种是直接在控制台输入一段英文，然后给出关键词进行检索匹配，但这样一般只能适用于小段的文字。所以，为了更面向实际，本程序是支持通过直接复制wps的word文件中的长篇幅英文文本到控制台中，再给出需要检索匹配的关键词进行匹配的。所以，为了支持这一功能，考略到wps的word的文档中的分段时，如果直接复制到控制台里，那么后一段的第一个字母会直接紧挨着前一段的最后一个标点符号，这一方面违背了日常英语的写作规范，还会造成程序的误判，导致从文件中读入单词时受到影响，直接把标点符号分隔的两个词识别为一个词，引发BUG。因此，程序为了规避这一问题，在控制台中每读入一行字符串后额外在txt文档中写入一个空格用来分隔两个不同段落。

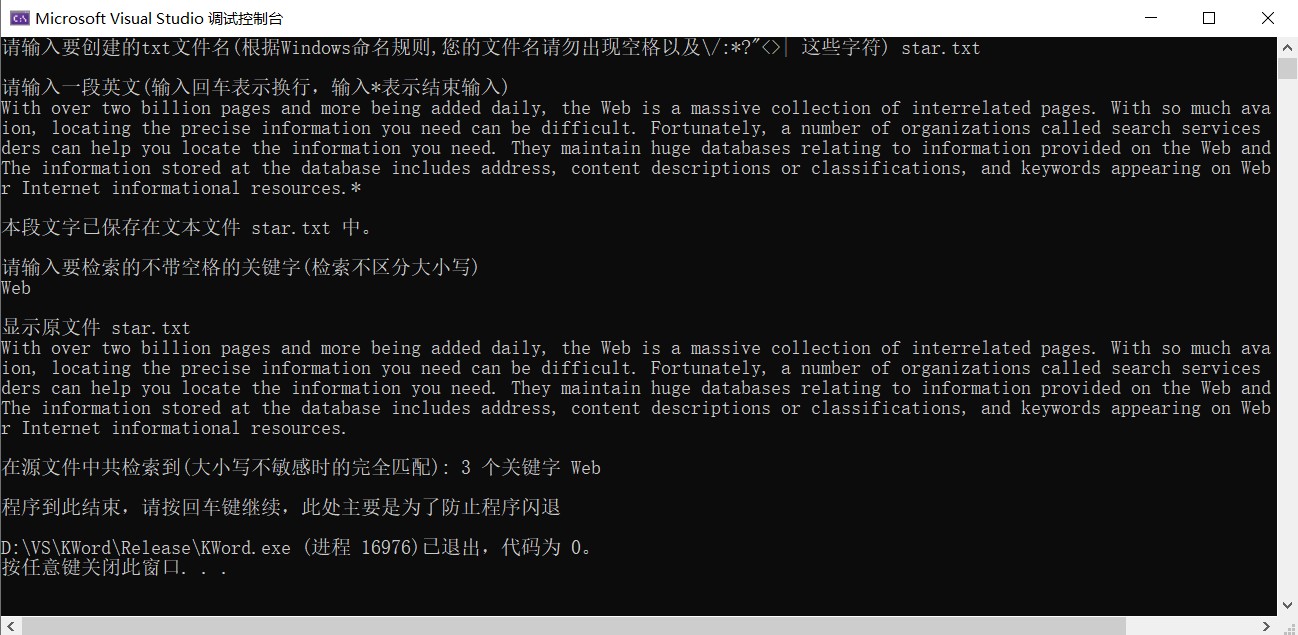
# 4.测试部分

特别注意：

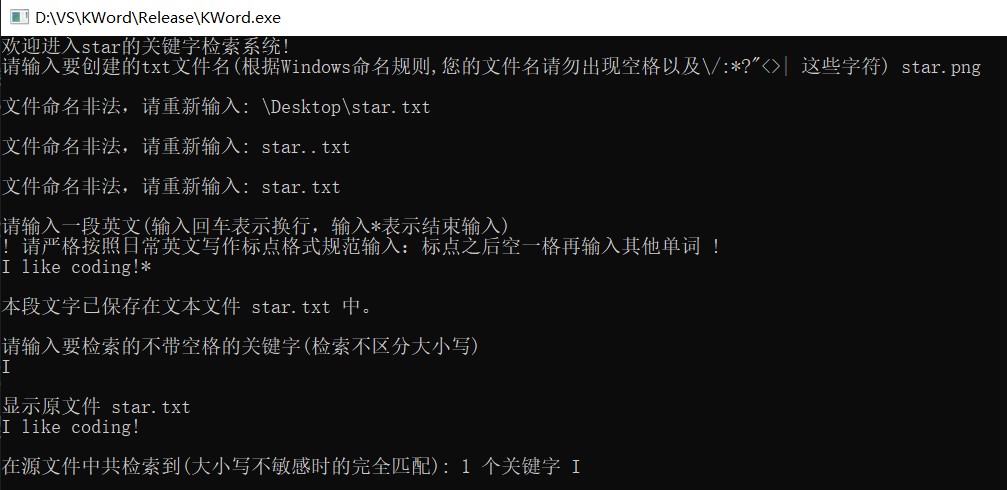
在输入完一段文本后记得输入一个字符 \* 表示结束输入 ! ! !

不要误以为程序卡住 ! ! !

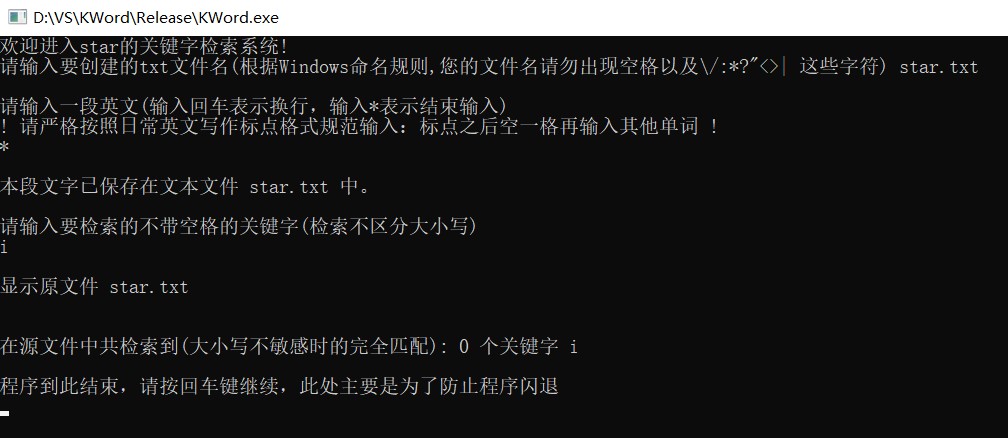
## 正常功能测试

①输入和题目文档中一模一样的一段英文文字，得到相同的结果：3个Web关键词

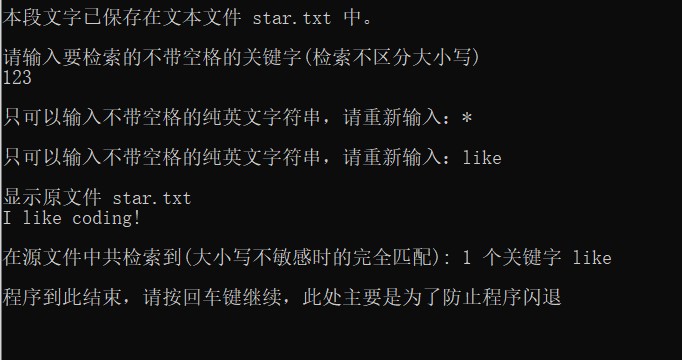
②不规范的文件名处理测试：



③边界测试：当不输入任何英文字母，直接输入字符 \* 表示结束输入，则重新依旧正常运行。



④不合规的关键词输入处理：



⑤面向wps的word文档直接复制到控制台进行测试：

说明：该文档是在英语课里随机选的一片文章，共2503个单词，文章中夹杂着中文、标点字符等其他非英文字符，已检验过，其中有单独的27个is单词，如下是直接将该文章从wps的word文档里复制到程序控制台中，检索匹配关键词is的结果：

（备注：该测试用例已随文档提交至服务器）

