项目说明文档

数据结构课程设计

——关键字检索系统

作 者 姓 名： 刘畅

学 号： 2054164

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc91009455)

[1.1 背景分析 1](#_Toc91009456)

[1.2 功能分析 1](#_Toc91009457)

[2 设计 1](#_Toc91009458)

[2.1 类结构设计 1](#_Toc91009459)

[2.2 成员与操作设计 1](#_Toc91009460)

[2.3 系统逻辑设计 2](#_Toc91009461)

[3 实现 3](#_Toc91009462)

[3.1 生成文件的实现 3](#_Toc91009463)

[3.1.1 实现方法 3](#_Toc91009464)

[3.1.2 核心代码 3](#_Toc91009465)

[3.2 关键字检索的实现 4](#_Toc91009466)

[3.2.1 实现方法 4](#_Toc91009467)

[3.2.2 核心代码 4](#_Toc91009468)

[4 测试 5](#_Toc91009469)

[4.1 基础功能测试 5](#_Toc91009470)

[4.2 默认文件名测试 5](#_Toc91009471)

[4.2 极端情况测试 6](#_Toc91009472)

# 1 分析

## 背景分析

检索某个关键字在一段文字中出现次数，是在文字处理、文本统计等工作中经常需要使用的操作。由于其操作相对比较偏向底层，因此使用的频率相对而言较高，所以其执行的效率、复杂度等也是设计者需要考虑的一大问题。对于不同的语言，执行效率也有所不同，比如英语，其单词都以空格鲜明地分隔开来，故实现起来比较容易；但对于中文，一句话所有的字词都紧密地连接在一起，因而实现起来比较复杂。

## 1.2 功能分析

作为一个关键字检索系统，我们让用户在终端输入多段英语段落（也即多行英文），为了明确标识结束符，我们以单独一行的一个“0”作为终端输入的结束符。在程序中，我们将用户输入的英文段落读入，并将其存储于一个用户指定文件名的文件之中。这是第一个步骤。

而后，我们再通过读取这个文件，去检索关键字。这是第二个步骤。这是题目的要求，虽然看似多此一举比较繁琐，但是它实现了将输入内容存放于文件、从文件中读取内容两个操作，很好地将此题建立在了文件输入输出的基础上，并且能够保存用户输入的文字，一举两得。

# 2 设计

## 2.1 类结构设计

我们设计了两个类：一个类（文件初始化类InitFile）专门负责读取终端输入的英文段落，并将其生成文件；另一个类（关键字检索类KeywordSearch）负责读取第一个类生成的文件，然后检索关键字在其中出现的次数，最后在main函数中输出。

## 2.2 成员与操作设计

**文件初始化类（InitFile）**

public:

//构造函数：初始化文件名，并以ios::out形式创建（或打开）文件

//ios::out形式打开文件：若不存在该文件，则创建文件；若存在，则清空文件之后打开

InitFile();

//析构函数：关闭文件

~InitFile();

//获取输入的文本（以单独一行的0结尾）

void InputText();

//获取文件名

string getFilename()const;

private:

//\_name是文件名，\_input是未经处理的直接从终端输入的内容

//\_fout是文件输出流

string \_name, \_input;

ofstream \_fout;

**关键字检索类（KeywordSearch）**

public:

//构造函数传入要寻找的关键字，将要搜索的文件打开

KeywordSearch(const string& keyword, const string& filename);

//析构函数，关闭文件

~KeywordSearch();

//获取关键字出现的次数

int getCount();

private:

string \_filename,\_keyword;

ifstream \_fin;

int \_count;

## 2.3 系统逻辑设计

先生成InitFile类的对象init\_file，直接调用其InputText()函数，同时用户输出相关信息，这样就创建了一个包含了用户输入信息的文件；然后输入要检索的关键字，将其全部转化为小写；使用关键字keyword和文件名init\_file.getFilename()生成一个KeywordSearch对象kwSearch，直接调用其getCount()函数得到关键字出现的次数。

# 3 实现

## 3.1 生成文件的实现

### 3.1.1 实现方法

在InitFile构造函数中，先让用户输入文件名，如果用户输入不为空，则将文件名改为用户输入，不然则为默认文件名Default.txt；然后用该文件名以ios::out打开文件。

在InitFile::InputText()中，先让用户输入英文段落，以单独一行的0作为结束标志，使用getline读入一行，每读入一行，都全部转换为小写然后储存到打开的文件之中。

这样就实现了文件的生成。

### 3.1.2 核心代码

InitFile::InitFile()

{

\_input = "";

cout << "请输入文件名（默认情况下为Default.txt）：";

getline(cin, \_input);

\_name = (\_input == "" ? "Default.txt" : \_input);

\_fout.open(\_name, ios::out /\*| ios::app\*/);

}

void InitFile::InputText()

{

cout << "请输入英文段落（以单独一行的0作为结束标志）：" << endl;

while (getline(cin, \_input))

{

if (\_input == "0") {

break;

}

transform(\_input.begin(), \_input.end(), \_input.begin(),std::tolower);//全部转为小写

\_fout << \_input << endl;

}

}

## 3.2 关键字检索的实现

### 3.2.1 实现方法

通过关键字keyword和文件名init\_file.getFilename()（文件名要从InitFile类的对象中获取）来初始化KeywordSearch对象kwSearch，在其构造函数中将私有成员\_keyword和\_filename分别赋值，将关键字出现次数\_count初始化为-1，然后用ios::in打开\_filename，并将文件标识符设置在文件的begin位置。

而后在mian函数中调用kwSearch.getCount()，在KeywordSearch::getCount()中，先判断\_count是否为-1，若不为-1则说明已经检索过，直接返回\_count即可；不然则将其赋值为0，然后不断读取文件中的字符串（因为英文以空格为分隔符，而字符串的读取也是以空格为分界，所以能很好地模拟读取过程），逐个判断字符串是否为关键字，若是则\_count++。

这里需要注意标点符号可能和英文单词一起被读入在字符串中，因此需要判断一下字符串的末尾是否是英文，若不是，则提取字符串的纯英文子串作为新的判断对象。

### 3.2.2 核心代码

KeywordSearch::KeywordSearch(const string& keyword, const string& filename)

{

\_count = -1;

\_keyword = keyword;

\_filename = filename;

\_fin.open(\_filename, ios::in);//以ios::in形式打开文件，只能读取文件

\_fin.seekg(ios::beg);//将文件标识符设置在begin位置

}

int KeywordSearch::getCount()

{

//\_count!=-1时说明已查找过，直接返回\_count

if (\_count == -1) {

\_count = 0;

string word;

//当未读取到文件末尾时

while (!\_fin.eof()) {

//清空，防止最后一行读不到东西导致最后一个单词被两次计算

word.clear();

\_fin >> word;//从文件读取字符串，有可能连着标点符号一起读取，故需要作二次处理

int size = word.size();

if (!word.empty()) {

//如果读取字符串最后一位不是字母，则说明连着标点一起读进来了

if (word[size - 1] < 'a' || word[size - 1]>'z') {

//获取子串，重新构造字符串

word = word.substr(0, size - 1);

}

//此时字符串一定是纯英文，直接判断是否与关键字相等

if (word == \_keyword) {

\_count++;

}

}

}

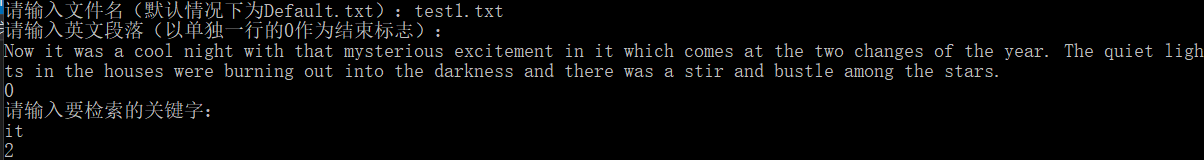
}

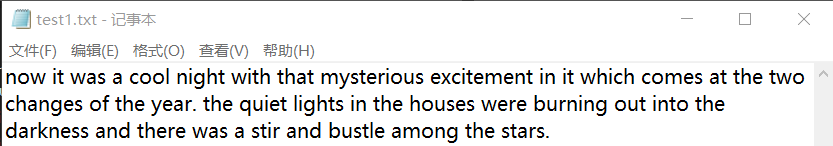
return \_count;

}

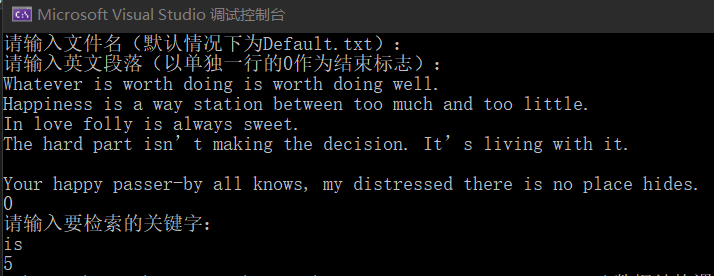
# 4 测试

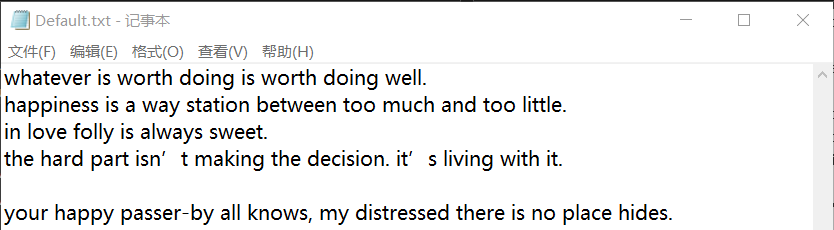
## 4.1 基础功能测试





## 4.2 默认文件名测试





## 4.2 极端情况测试

