摘 要

###### 程序设计课程个人大作业的作业要求是锻炼学生熟练编程语言的使用。

###### 实验设计主要遵循题目要求、模块化程序设计的思想

###### 实验内容主要包括：用C++语言编写的文本处理统计程序，可以统计文本中单词出现次数位置等。

###### 作业结论为完成预期作业目标。

**关键词：**高级语言程序设计、C++、词频统计、单词索引

**目 录**

## 1 实验目的和意义

#### 实验目的

本实验的实验目的是通过编码和测试指定问题，综合运用所学高级语言程序设计课程中的知识，提高熟练使用高级语言编写代码和解决问题的能力，锻炼高级语言程序设计的思维以及编程能力。

#### 实验意义

上过 C++课程之后，只能完成一些简单的编程练习，但对于如何开发有实际用途的程序往往感到束手无策。该实验由个人独立设计，独立完成编码、测试以及实验报告工作、从而熟悉高级语言程序设计方法，提高高级语言编程和解决问题的能力。

## 实验设计

#### 问题描述

整理一批文本文件作为输入数据，实现以下处理功能：

* + 1. 词频统计：统计输入文件中的每个单词在所有文件中出现的总次数，保存统计结果。
    2. 建立单词索引：对所有单词记录每次出现时的信息，包括文件名、行号、所在的语句 内容等。
    3. 查询词频：查询显示文件从出现频度最高的前 10 个单词；查询显示最热门的 10 个检索单词；输入一个单词，从统计结果中查询出该单词的在文件中出现总频次，查询出该单词被查询过的总次数。
    4. 关键词检索：输入一个单词，查询出所有该单词出现过的文件名、行号和所在语句内容。

【基本要求】

以上述功能为基础，设计所需的数据结构、程序功能模块等，可以根据需要扩展所需 功能。

程序运行过程中，有方便合理的用户操作界面，能直观展示输入数据、处理过程及输 出结果。

词频统计结果和索引数据，可以存储在数据文件或数据库中，数据结构、具体数据项 自定。

输入文件内容为英文，不统计英文中出现的虚词，包括冠词、介词、连词和感叹词等

输入的文件允许单个输入，也允许批量文件输入。

#### 实验原理

统计文本的过程主要为

读取文件，按行读取字符串，读取单词，统计单词词频，保存数据

这五个步骤。批量文本的处理也即对这五个步骤的重复。

用fstream流的方式读取文件，

while循环结构用getline函数读取行字符串，

用for循环遍历字符串提取出单个单词，通过字符的字母判断，将读取的字母拼接成单词。

统计词频时先遍历储存单词的数组判断是否以存在该单词，再进行相关储存。

保存数据采用vector数组的嵌套式结构，每个单词的相关信息为数组的单个元素，而单词的各个位置储存在另一个数组中。

批量文件的处理，通过用户输入路径，采用vector数组储存，再对数组中的路径进行文件读取处理。

#### 详细设计

2.3.1数据结构：

使用结构体将单个单词的单词名，出现的频率，被查询次数和每次出现的位置进行储存。

将单个结构体储存在Vector动态数组中实现多个单词的储存。

struct WordFrequency {

string Wordname;//单词名

int frequency = 0;//词频

int required = 0;//被查询次数

vector<Place>place;//储存位置的动态数组

};

vector<WordFrequency>WF(sum);

对单词每次出现的位置进行了单独的统计，同样采用结构体的方式储存单词出现时所在的文件名，出现的行数和其所处的语句数。这也是后续单词查询时所处的整段语句的输出依据

struct Place {

string FileName = "None";//文件名

int linenum = 0;//行数

int SentenceNum = 0;//所在语句标号

};

vector<string>FN；

2.3.2 词频统计:

1：建立fstream只读对象，关联到待处理的文本。

ifstream fin; //创建只可读对象

fin.open(filename);//关联文件

//判断文件是否成功打开

if (!fin.is\_open()) {

system("cls");

cout << name << "没有成功打开\n" << endl;

}

else

cout << "成功打开了" << name << "文件" << endl;

2：使用while循环逐行读取文本中的字符，然后调用预处理函数

//按行分析操作

string LineText; //储存每一行字符串

int linenum = 0; //储存行数

string word; //储存单词

regex reg("[(--);.?！]");//匹配句号、问号、感叹号

int totallinenum = totlalline(filename, name);

//按行读取

while (getline(fin, LineText)) {

//统一为小写

LineText = WordTransition(LineText);

3：对一整行字符串先进行预处理，统一为小写英文字母；

string WordTransition(string word) {

for (int i = 0; i < int(word.size()); i++) { //获取字符串长度，使用length()也可以

if (word[i] >= 'A' && word[i] <= 'Z') //判断字符是否是大写字母

word[i] = word[i] + 32; //ASCII码表中大小写字符相差32

} //for

return word; //返回小写单词

}；

4：然后采用循环读取单个字符的方式，将符合条件的字符，放入统一字符串中，拼接为一个单词。

for (char & ch : LineText) {//读取单词

//读取一个字符

if (ch >= 'a' && ch <= 'z') {

if (word == "\0") //word为空，放入第一个字母

word = ch;

else

word += ch; //word不为空，拼接字母组成单词

} //if

//一个单词读取完毕

5：当读取不为小写字母的字符将作为单词读取结束的标准，进入else分支，

首先将统计的单词总数加一，并进行统计总数的检测。使单词统计的单词数据不要太多

else {

//判断之前的word里面是否有单词

if (word != "\0") {//有单词，总数+1

sum++;

//限制单个文件读取单词数量不超过MaxSize

if (sum >= MaxSize) {

cout << "文章单词超出统计上限，系统已退出" << endl;

fin.close(); //关闭文件

exit(0); //退出程序

system("pause"); //暂停

}

6：对读取的单词进行虚词的过滤，这里采用遍历在文件头部的虚词库进行匹配来判断

vector<string>FunctionWord = {

"a","an","the",

"in","on","from","above","behind",

"at","by","for","from","in","of","on","to","with",

"onto","into","within",

"and","both","as","not","but","yet","while",

"or","either","so","that","whether","if","when","while"

"before","after","once","untill","till","because","althought",

"thought","even","whatever","however","unless","than","wherever",

"oh","hi","hello","am","is","are","were","was",

"i","you","me","he","him","she","her","it","they","them","them","his","your","their","hers","theirs","my"

};

//排掉虚词,保存单词信息

//迭代器访问vector数组

bool functionwordtest = true;//默认通过

vector<string>::const\_iterator iter;

iter = FunctionWord.begin();

for (iter; iter != FunctionWord.end(); iter++)

if (word == \*iter)

functionwordtest = false;

7：进行重复性检测，重复的单词其词频增加，没重复的单词则加入到WF动态数组中

//对单词进行重复性检测

bool repeat = false;//默认不重复

vector<WordFrequency>::iterator iter\_0=WF.end();

for (vector<WordFrequency>::iterator iter = WF.begin(); iter != WF.end(); iter++) {

if (iter->Wordname == word) {

repeat = true;

iter\_0 = iter;

}

8：存入单词信息或频次加一

//通过虚词检测则准备存入WF数组

if (functionwordtest) {

Place pl;

pl.FileName = name;

pl.linenum = linenum;

pl.SentenceNum = sentencenum;

WordFrequency wf;//当前单词信息

wf.Wordname = word;

wf.required = 0;

wf.frequency = 1;

wf.place.push\_back(pl);

if(empty){

WF.push\_back(wf);

}

else {

if (repeat) {

//单词重复

iter\_0->frequency++; //词频+1

//位置储存

iter\_0->place.push\_back(pl);

} //if

else {

//单词不重复

WF.push\_back(wf);

}

}

9：重置word和循环退出语句

word = "\0"; //重置word，用于存放新单词

linenum = linenum + 1;//每轮循环行数加1

if (linenum == totallinenum+1)

break;

}//while

fin.close();//关闭文件

2.3.3数据保存

数据的保存经由元素排序和保存两个步骤

元素排序采用冒泡排序，通过遍历数组，比对频次实现元素的排序。具体方式为使用vector数组自带的迭代器，用两个for循环遍历进行冒泡对比

void frequencysort() {

//反向迭代器

vector<WordFrequency>::reverse\_iterator iter\_0 = WF.rbegin();

//用0号迭代器遍历WF数组

for (iter\_0; iter\_0 < WF.rend(); iter\_0++) {

vector<WordFrequency>::reverse\_iterator iter\_1 = WF.rbegin();

//用1号迭代器遍历WF数组

for (iter\_1; iter\_1 < WF.rend(); iter\_1++) {

if (iter\_0->frequency > iter\_1->frequency)

//交换两个迭代器所指元素

iter\_swap(iter\_0, iter\_1);

}

}

数据的保存通过建立ofstream对象，以<<的方式将WF数组的输入到ofstream对象

ofstream fout;//创建只写对象fout

fout.open("output.txt", ios::trunc);//准备写入统计数据,并重置文本

for (int i = 0; i < WF.size(); i++) { //写入WF结构体到文件中

string part1 = WF[i].Wordname;

string part2 = "{" + to\_string(WF[i].frequency) + "," + to\_string(WF[i].required) + "}";

string part3 ="{";

for (int n=0; n < WF[i].place.size(); n++) {

if (regex\_match(part3, regex(WF[i].place[n].FileName)))

part3 += " "+WF[i].place[n].FileName;

part3 += "(" + to\_string(WF[i].place[n].linenum) + "," + to\_string(WF[i].place[n].SentenceNum) + ")";

}

fout <<setw(20)<<part1<< setw(10)<<part2<<setw(50)<<part3+"}" << endl;

2.3.4显示最高词频:

经过排序的数组，最高频次的在数组某位，直接将WF数组的后十位输出来即可

for (iter; iter <= (WF.rbegin() + 9); iter++)

2.3.5查询单词:

读取用户输入的字符，在WF数组遍历寻找，找到后输出除Place数组的其他信息

string WORD;

cout << "请输入你要查找的单词" << endl<<"输入3返回上一级\n";

cin >> WORD;

if (WORD == "3") {

cout << "正在返回上一级";

SearchMenu();

}

//mark用于判断输入的词是否在词库中

bool mark = true;

for (int i = 0; i < WF.size(); i++) {

if (WORD == WF[i].Wordname) {

mark = false;//mark标记着存在此单词

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*英文单词的词频统计和检索系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl;

cout << "单词名:" << WF[i].Wordname << endl

<< "出现频次：" << WF[i].frequency << " "

<< "被查询次数:" << WF[i].required << endl;

WF[i].required++;//被查询数加一

调用其Place数组，根据其中的file元素对比储存文件路径FN数组，再次根据地址打开文件进行读取，

string address;//储存此次打开的文件地址

string str = WF[i].place[n].FileName;

//find函数如果找到了则返回指针

for (int u = 0; u < FN.size(); u++) {

if (FN[u].find(str) != string::npos) {

address = FN[u];

}

else;读取到特定行数，对此行字符串进行与语句识别处理。用正则表达式匹配句末标点。当sentencenum变量匹配时输出当前读取的字符，从而将 一整个语句输出

if (linenum == WF[i].place[n].linenum) {

//LineText = WordTransition(LineText);//统一为小写

int sentencenum = 0;//标记所在语句号用于后期查询

for (char& ch : LineText) {//读取单词

//用正则表达式匹配句子结束符号

const char\* CH = &ch;

string str(CH);

//检测是否为句子末尾

bool ret = regex\_match(str, reg);

if (ret)

sentencenum++;

//匹配对应语句

if (sentencenum == WF[i].place[n].SentenceNum) {

cout << ch;

mark = false;

}

}//for

}

2.3.6 批量导入函数

FN动态数组储存路径，然后用循环语句进行分别处理

void BulkImport() {

cout << "正在导入文本" << endl;

int TxtNum = FN.size();//文本数量

for (int i = 0; i < TxtNum; i++) {

string filename = FN[i];//filename表示当前处理的文本路径

StatisticsData(filename);

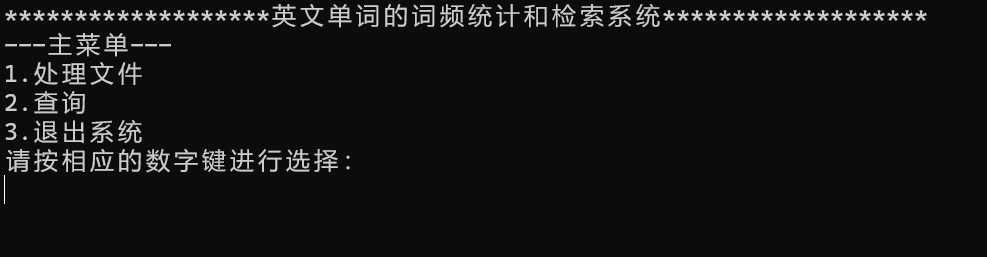
};

};

## 实验结果

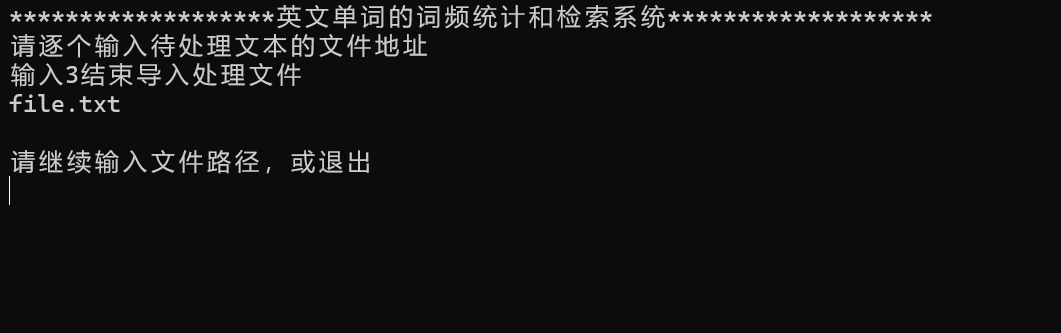
**主菜单界面**

用户交互的第一个界面，可以是通过数字按键的输入选择处理和查询两个功能

****

**文件处理界面**

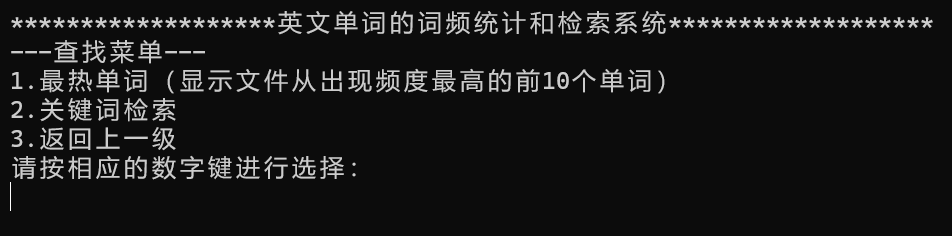
依次输入文件的路径，程序会在输入完成后自动处理文件

****

**搜索界面**

输入1实现词频最高的前10个单词的输出

输入2实现对特定单词的查找

****

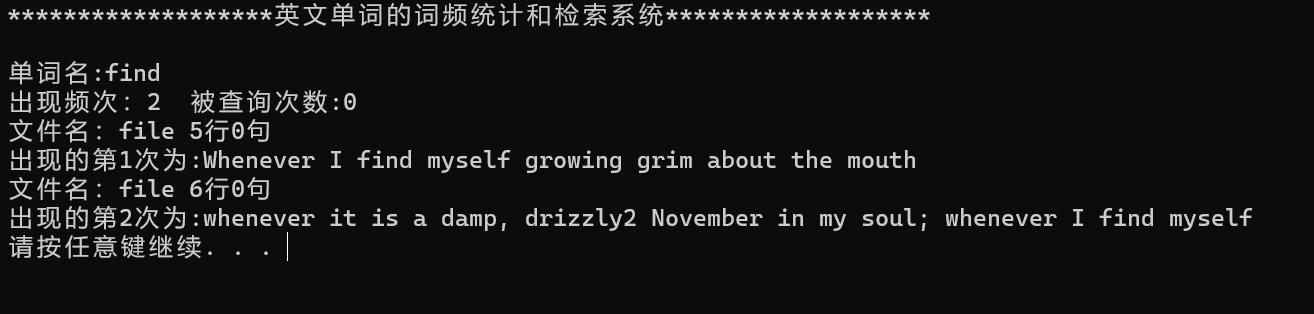
**最热门的10个单词**

在数据储存时，已经根据词频做好了排序，将次数最高的前十单词输出

****

**单词查询界面**

输入单词名，程序将遍历储存单词信息的数组，匹配单词名，读取该单词储存的位置信息，然后再将储存路径的数组中匹配到单词储存的路径，打开文件，匹配特定行数和特定句数，将该剧单词输出，实现单词的查询和所在语句的输出

****

**测试文本**

