**题目：程序源代码的相似性**

**1、问题描述**

对于两个C++语言的源程序代码，用哈希表的方法分别统计两个程序中使用C++语言关键字的情况，并最终按定量的计算结果，得出两个程序的相似性。  
  基本要求：建立C++语言关键字的哈希表，统计在每个源程序中C++关键字出现的频度, 得到两个向量X1和X2，通过计算向量X1和X2的相对距离来判断两个源程序的相似性。

设s是向量X1和X2的相对距离，s=sqrt( ∑(xi1-xi2) 2 )，当X1=X2时，s=0, 反映出可能是同一个程序；s值越大，则两个程序的差别可能也越大。试分析计算结果，得出两个程序的相似性。

提高要求：建立源代码用户标识符表，比较两个源代码用户标识符出现的频度，综合关键字频度和用户标识符频度判断两个程序的相似性。

**2、需求分析**

软件的基本功能：通过读取存取在文件中的C++代码，程序可以统计该代码中存在的用户标识符及关键字的个数，并通过分别计算两个代码基于用户标识符的相对距离及基于关键字的相对距离，判断两个代码的相似性。

输入/输出形式：用户可以通过控制台，根据提示输入。

输入形式：

①任意单个字符（字符“E”为结束程序）；

②文件1的名称；

③文件2的名称。

输出形式：

①对于正确的输入，可以分别输出两个代码的用户标识符统计，两个代码的用户标识符统计的整合，两个代码基于用户标识符的相对距离，两个代码的关键字的统计，两个代码基于关键字的相对距离，并根据两个代码的两个相对距离给出两个代码抄袭可能性，及下一步输入的提示。

②当第一个输入的为多个字符时，程序结束；

③当输入的文件名与存于文件夹中的文件名不同时，即找不到所输入的

文件时，程序结束。

测试数据要求: 选择若干组编译和运行都无误的C++程序，程序之间有相近的和差别大的，用上述方法求s, 对比两个程序的相似性。

**3、概要设计**

**（1）抽象数据类型**：

根据题目要求，考虑用哈希表比较合适，我对本题的实现中没有单独定义数据类型，而是直接定义函数，并在主函数中调用。因此，没有抽象数据类型。

**（2）主程序流程：**

开始

输入字符

**Y**

是否为E

**N**

**N**

输入文件1的名字

**Y**

输入文件2的名字

**N**

**Y**

输出文件1,2的用户标识符统计与相对距离

结束

**（3）模块调用关系：**

本程序中函数包括main函数，SetHash函数，InputKT函数，FilterFile函数，InputArray函数，TransformCode函数，Statistics函数，UserStatisticMixed函数，similarity函数，其中SetHash函数的主要功能是读取文件（存放所有关键字的文件）并将文件中的每一个单词存储于一个string类型数组，其ASCII码之和存储于一个int型数组，并一一对应；InputKT函数的主要功能是根据ASCII码将单词及其ASCII码之和存储于之前定义的结构体数组中；FilterFile函数的主要功能是读取存储代码的文件，并将文件中的标点符号及注释过滤；InputArray函数的主要功能是分词；TransformCode函数的主要功能是将InputArray函数中的单词注意转换为ASCII码之和；Statistics函数的主要功能是统计代码中的关键字；similarity函数的主要功能是计算两个代码的相对距离；UserStatisticsMixed函数的主要功能是针对用户标识符的一系列操作（类似于以上所述的函数功能的总和）。

其函数调用关系如下：

main

Similarity

InputKT

SetHash

UserStatisticMixed

Statistics

TransformCode

InputArray

FilterFile

**4、详细设计**

**主程序以及其他模块的算法描述：**

**（1）主函数具体代码：**

int main() {

SetHash();

InputKT();

char ch;

cout << "按任意键开始比较，按‘E’键结束程序" << endl;

while (cin >> ch && ch != 'E') {

string word1[M], word2[M];

string file1, file2;

cout << "请输入第一个要比较的文件的文件名" << endl;

cin >> file1;

FilterFile(file1, file1);

cout << "请输入第二个要比较的文件的文件名" << endl;

cin >> file2;

FilterFile(file2, file2);

int len1 = InputArray(file1, word1);

int len2 = InputArray(file2, word2);

int w1[M], w2[M];

TransformCode(word1, len1, w1);//将word字符串数组中的字符，转化成ASCII码之和，存入w的整型数组中

TransformCode(word2, len2, w2);

int x1[73], x2[73];

Statistics(x1, w1, len1);

Statistics(x2, w2, len2);

string u1[50], u2[50];

int v1[50], v2[50];

int w3[50], w4[50];

cout << "文件1的用户标识符统计如下：" << endl;

int p = UserStatisticMixed(word1, len1, u1, v1, w3);

cout << endl;

cout << "文件2的用户标识符统计如下：" << endl;

int q = UserStatisticMixed(word2, len2, u2, v2, w4);

cout << endl;

int z = p + q;

int \*r = new int[z];

int \*t = new int[z];

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

r[i] = 0;

t[i] = 0;

}

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

for (int j = 0; j < q; j++) {

if (u1[i] == u2[j]) {

t[i] = w4[j];

z--;

for (int k = j; k < q; k++) {

u2[k] = u2[k + 1];

v2[k] = v2[k + 1];

w4[k] = w4[k + 1];

}

q--;

break;

}

}

}

string \*r2 = new string[z];

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

r2[i] = u1[i];

int j1 = 0;

for (int i = p; i < z; i++) {

r2[i] = u2[j1];

j1++;

}

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

r[i] = w3[i];

int j2 = 0;

for (int i = p; i < z; i++) {

t[i] = w4[j2];

j2++;

}

int \*r3 = new int[z];

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

r3[i] = v1[i];

int j3 = 0;

for (int i = p; i < z; i++) {

r3[i] = v2[j3];

j3++;

}

cout << "文件1与文件2的用户标识符整合：" << endl;

cout << "\t" << "序号" << "\t\t" << "关键字" << "\t\t\t\t" << "ASCII值" << "\t\t" << "频度1" << "\t\t" << "频度2" << endl;

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

cout << "\t" << i << "\t\t";

cout << setiosflags(ios::left)/\*左对齐方式输出\*/ << setw(20)/\*设域宽为20个字符\*/ << r2[i];

cout << "\t\t" << r3[i] << "\t\t" << r[i] << "\t\t" << t[i] << endl;

}

int sum1 = 0;

double sim1 = 0;

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

sum1 = sum1 + (r[i] - t[i])\*(r[i] - t[i]);

sim1 = sqrt(sum1);

}

cout << "以用户标识符为基准的相对距离为：" << sim1 << endl;

cout << endl;

cout << "文件1与文件2的关键字统计如下：" << endl;

double s = similarity(x1, x2);

cout << "\t" << "序号" << "\t\t" << "关键字" << "\t\t\t\t" << "ASCII值" << "\t\t" << "频度1" << "\t\t" << "频度2" << endl;

for (int i = 0; i < 73; i++) {

cout << "\t" << i << "\t\t";

cout << setiosflags(ios::left)/\*左对齐方式输出\*/ << setw(20)/\*设域宽为20个字符\*/ << key\_table[i].data;

cout << "\t\t" << key\_table[i].code << "\t\t" << x1[i] << "\t\t" << x2[i] << endl;

}

cout << endl;

cout << "这两个源代码的相对距离为：" << s << endl;

if (s > 20)

cout << "这两个代码基本不可能抄袭" << endl;

else if (sim1 >= 10 && (s >= 10 && s <= 20))

cout << "这两个代码可能存在小部分抄袭" << endl;

else if ((s >= 10 && s <= 20 && sim1 < 10) || (s < 10 && sim1 >= 10))

cout << "这两个代码可能存在大部分抄袭" << endl;

else

cout << "这两个代码可能为全部抄袭" << endl;

cout << "按任意键进行下一轮对比，按‘E’键结束程序" << endl;

delete r, t, r2, r3;

}

return 0;

}

这个函数主要调用了实现功能的各个函数。其步骤为：先通过读取文件，将文件中存储的关键字存入字符串型数组，并计算出每个关键字的各字母的ASCII码之和，存入整形数组，形成与字符串数组一一对应的关系。之后，将关键字与其ASCII码存入结构体类型的哈希表中。再通过按键，启动比较两个代码的进程。先输入第一个文件的名称，读取文件并进行文件过滤，然后输入第二个文件的名称，读取文件并进行文件过滤。然后对文件中的代码分别进行分词，并计算出它们的ASCII码和进行统计。然后对用户标识符进行相似的操作，并将两个代码的用户标识符统计进行整合。最后进行相对距离的计算以及相似性的定性判断。

**（2）其它模块的算法描述**

读取文件并建立存有关键字与其对应的ASCII码的数组

功能：读取文件并将文件中的关键字存入string型数组，通过size()函数计算出每个元素的具体长度，并利用for循环计算出每个元素的ASCII码之和，最后将ASCII码存于整型数组，并形成一一对应的关系。

void SetHash() {//通过读取文件，建立数组

ifstream ifile("keywords.txt", ios::in);

if (!ifile)

cerr << "fail to open" << endl;

string key;

int i = 0;

while (getline(ifile, key)) {//逐行读入

keyword\_table[i] = key;

ASCII\_table[i] = 0;

int j = key.size();

for (int k = 0; k < j; k++)

ASCII\_table[i] = ASCII\_table[i] + int(key[k]);//将每个关键字的ASCII码总和暂存于ASCII\_table中

i++;

}

ifile.close();

}

将关键字及其ASCII码总和存于建立的哈希表的函数

功能：通过分别计算由上一个函数得出的整型数组除以哈希表长（73）的余数，确定每一个关键字在哈希表的位置，并将关键字及其ASCII码一并存入结构体数组中。

void InputKT() {//将关键字及其ASCII码总和存于key\_table中

for (int i = 0; i < 73; i++)

key\_table[i].code = -1;

for (int i = 0; i < 73; i++) {

int j = ASCII\_table[i] % 73;

if (key\_table[j].code == -1) {

key\_table[j].code = ASCII\_table[i];

key\_table[j].data = keyword\_table[i];

}

else {

while (key\_table[j].code != -1) {

j = (j + 1) % 73;

if (key\_table[j].code == -1) {

key\_table[j].code = ASCII\_table[i];

key\_table[j].data = keyword\_table[i];

break;

}

}

}

}

}

文件过滤的函数

功能：通过扫描读取的文件，利用while循环将注释语句（即//后的内容与/\*\*/内的内容）及大部分标点符号过滤。

int FilterFile(string name1, string name2) {//读取要比较的文件，并将其中的注释部分忽略，然后输入进另一文件

ifstream ifile(name1, ios::in);

while (!ifile) {

cerr << "fail to open" << endl;

exit(0);

}

char ch;

string word = "";

while (ifile.get(ch)) {

if (ch == '/') {

ifile.get(ch);

if (ch == '/') {

ifile.get(ch);

while (ch != '\n')

ifile.get(ch);

word = word + ' ';

}

else if (ch == '\*') {

for (;;) {

ifile.get(ch);

if (ch == '\*') {

ifile.get(ch);

if (ch == '/') {

word = word + ' ';

break;

}

}

}

}

}

else if ((ch >= 'A'&&ch <= 'Z') || (ch >= 'a'&&ch <= 'z') || ch == '\_' || (ch >= '0'&&ch <= '9'))

word = word + ch;

else

word = word + ' ';

}

ifile.close();

ofstream ofile(name2, ios::out);

ofile << word;

ofile.close();

}

分词函数

功能：利用上一函数得出的字符串，以空格为界，利用for循环进行对原代码的分词。

int InputArray(string name, string word[M]) {

ifstream ifile(name, ios::in);

char ch;

int j = 0, k = 0;

string input;

string str = "";

while (ifile.get(ch)) {//将上一函数中文件中的内容全部读入input字符串中

input = input + ch;

k++;

}

ifile.close();

for (int i = 0; i < k; i++) {//对input字符串进行分词，并录入word字符串数组中

if (input[i] != ' ')

str = str + input[i];

else if (input[i] == ' '&&input[i + 1] != ' '&&i != 0) {

word[j] = str;

j++;

str = "";

}

}

ofstream ofile(name, ios::out);//将字符串数组录入文件

for (int k = 0; k < j; k++)

ofile << word[k] << " ";

ofile.close();

return j;//返回的是word中的单词数量

}

ASCII码转化函数：

功能：通过对string头文件中的size()函数以及for循环的运用，计算出代码分词后每个单词的ASCII码总和。

void TransformCode(string word[], int j, int code[]) {

string str = "";

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

str = word[i];

int k = str.size();

code[i] = 0;

for (int j = 0; j < k; j++)

code[i] = code[i] + int(str[j]);

str = "";

}

}

统计函数

功能：利用上一个函数计算出的每一个分词ASCII码，将其除以哈希表的长度（73），并比较该ASCII码与相应位置的ASCII码，若相同则计数，否则向后推移比较，最后统计出每个代码各个关键字的频度。

void Statistics(int x[], int word[], int len) {

for (int i = 0; i < 73; i++)

key\_table[i].count = 0;

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

int m = word[i] % 73;

if (key\_table[m].code == word[i])

key\_table[m].count++;

else {

int u = m;

m = m + 1;

while (m % 73 != u) {

int n = m % 73;

if (key\_table[n].code != -1 && key\_table[n].code == word[i]) {

key\_table[n].count++;

break;

}

else m++;

}

}

}

for (int i = 0; i < 73; i++)

x[i] = key\_table[i].count;

}

相似性函数（计算相对距离）

功能：通过利用for循环，计算出两个代码关于关键字的相对距离，并利用math.h头文件内的sqrt函数进行开方，得出两个代码基于关键字的相对距离。

double similarity(int a[], int b[]) {

double sum = 0;

for (int i = 0; i < 73; i++)

sum = sum + (a[i] - b[i])\*(a[i] - b[i]);

double sim = sqrt(sum);

return sim;

}

关于用户标识符的函数

该函数基本操作为关于关键字的一系列操作的综合，在此不再赘述。

**5、编码与调试分析**

**1.编码与调试过程中遇到的问题及解决方法：**

**【**问题一】统计用户标识符的时候，字符“i”总是出现两次，并且总是第一次是代码中出现的i的次数，第二次为0。

解决办法：根据题设可知，当分别统计两个代码的用户标识符时，因为是在存储代码的文件中查找用户标识符，所以用户标识符的数量不可能为0。因此使用一个循环，将数量为0的用户标识符删除。

解决此问题的核心代码：

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

if (user\_table[i].count == 0) {

for (int k = i; k < j - 1; k++)

user\_table[k] = user\_table[k + 1];

j--;

}

}

【问题二】当整合两个代码的用户标识符时，无法将两个代码中相同的用户标识符整合到一起，即无法同时在一个统计表中整合两个代码的用户标识符。

解决方法：根据题意可知，整合两个代码的用户标识符时，若某个用户标识符只在一个代码中出现，那么另一个代码中该用户标识符的个数即为零，不妨重新设两个数组并初始化为零，统计时若两个代码的该用户标识符频度均非零，就进行覆盖，并将用户标识符的总个数减一。

解决此问题的核心代码：

int z = p + q;

int \*r = new int[z];

int \*t = new int[z];

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

r[i] = 0;

t[i] = 0;

}

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

for (int j = 0; j < q; j++) {

if (u1[i] == u2[j]) {

t[i] = w4[j];

z--;

for (int k = j; k < q; k++) {

u2[k] = u2[k + 1];

v2[k] = v2[k + 1];

w4[k] = w4[k + 1];

}

q--;

break;

}

}

}

string \*r2 = new string[z];

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

r2[i] = u1[i];

int j1 = 0;

for (int i = p; i < z; i++) {

r2[i] = u2[j1];

j1++;

}

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

r[i] = w3[i];

int j2 = 0;

for (int i = p; i < z; i++) {

t[i] = w4[j2];

j2++;

}

int \*r3 = new int[z];

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

r3[i] = v1[i];

int j3 = 0;

for (int i = p; i < z; i++) {

r3[i] = v2[j3];

j3++;

}

【问题三】当输出用户标识符统计表即关键字统计表时，由于输出时将关键字、ASCII码及频度分别一起输出的，因此无法达到一一对应的效果，查看困难，同时测试代码时也不便于判别代码运行是否正确。

解决办法：加入头文件“iomanip”，并运用该头文件中的“setiosflags”函数进行左对齐，运用“setw”函数使每个关键字均享用20个字符的长度。

解决此问题的核心代码：

#include<iomanip>

for (int i = 0; i < 73; i++) {

cout << "\t" << i << "\t\t";

cout << setiosflags(ios::left) << setw(20) << key\_table[i].data;

cout << "\t\t" << key\_table[i].code << "\t\t" << x1[i] << "\t\t" << x2[i] << endl;

}

cout << endl;

**2.待解决的问题：**

①统计用户标识符时，只能统计关键字之后第一个用户标识符，例如，统计“int a，b；”时，只能统计a，却统计不了b；

②当自定义一个结构体时，以该结构体名定义变量时，无法统计该变量名；

③用户标识符的定量统计对两个代码相似性的定性判断作用不是特别大，因为当一个代码改变了另一个代码的用户标识符的名字，甚至只是交换原代码的两个自定义变量的名字，这两个代码基于用户标识符的相对距离就会改变，反而会对定性判断产生不必要的影响；除此之外，若比较两个代码的用户标识符数量，只有当两个用户标识符数量相等且已通过分析两个代码基于关键字的相对距离大致判断出两个代码有很大可能是抄袭的时，才能佐证判断。

**6、使用说明**

①按任意键进行比较，“E”键退出程序；

②输入要比较的第一个代码的文件名；

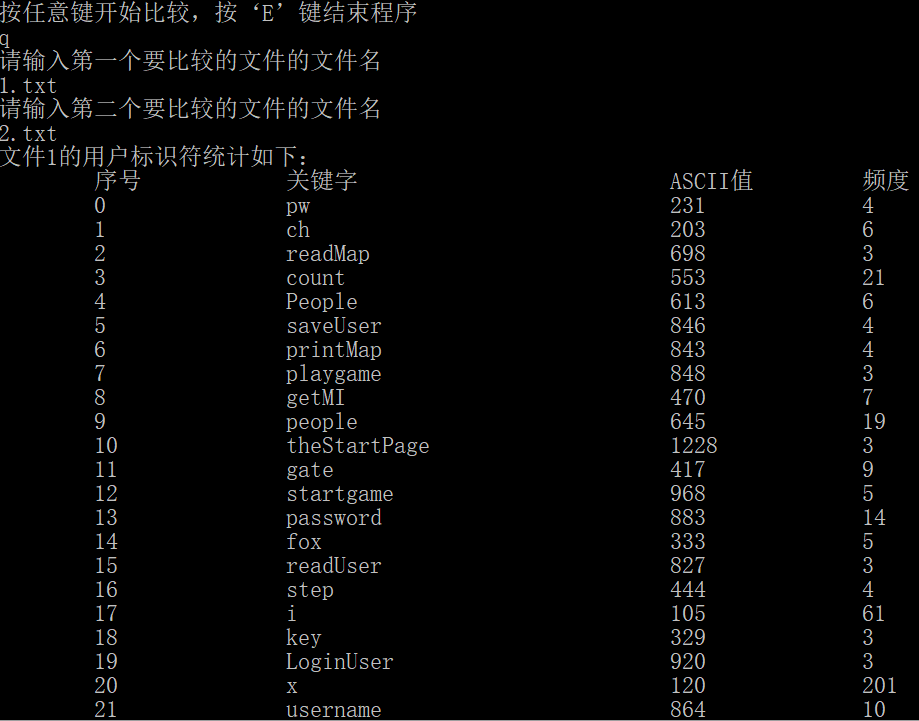
③输入要比较的第二个代码的文件名；

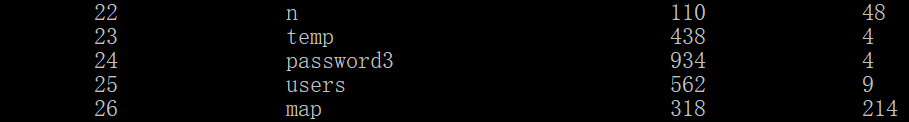
④若要继续进行比较，则按任意键，否则按“E”键退出程序。

**7、测试结果**

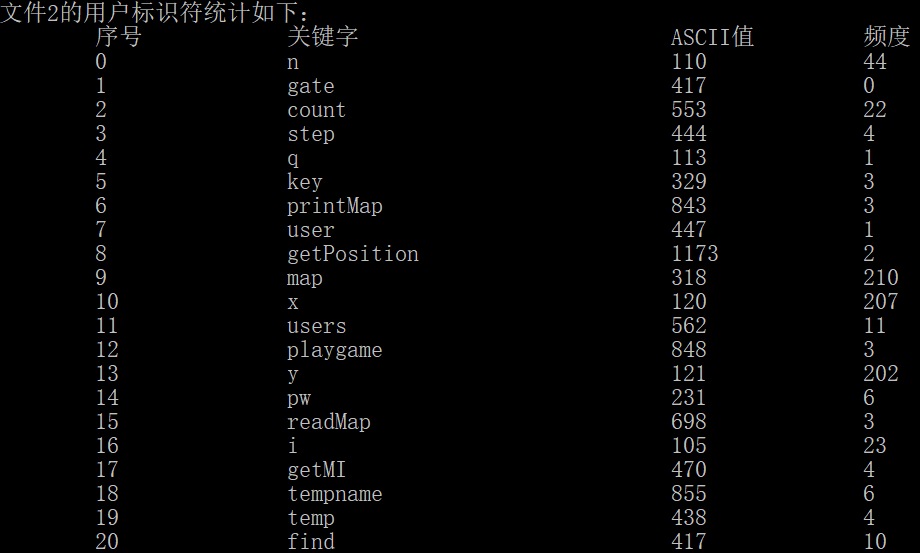
**两个代码基本无抄袭可能：**

文件1的用户标识符统计：

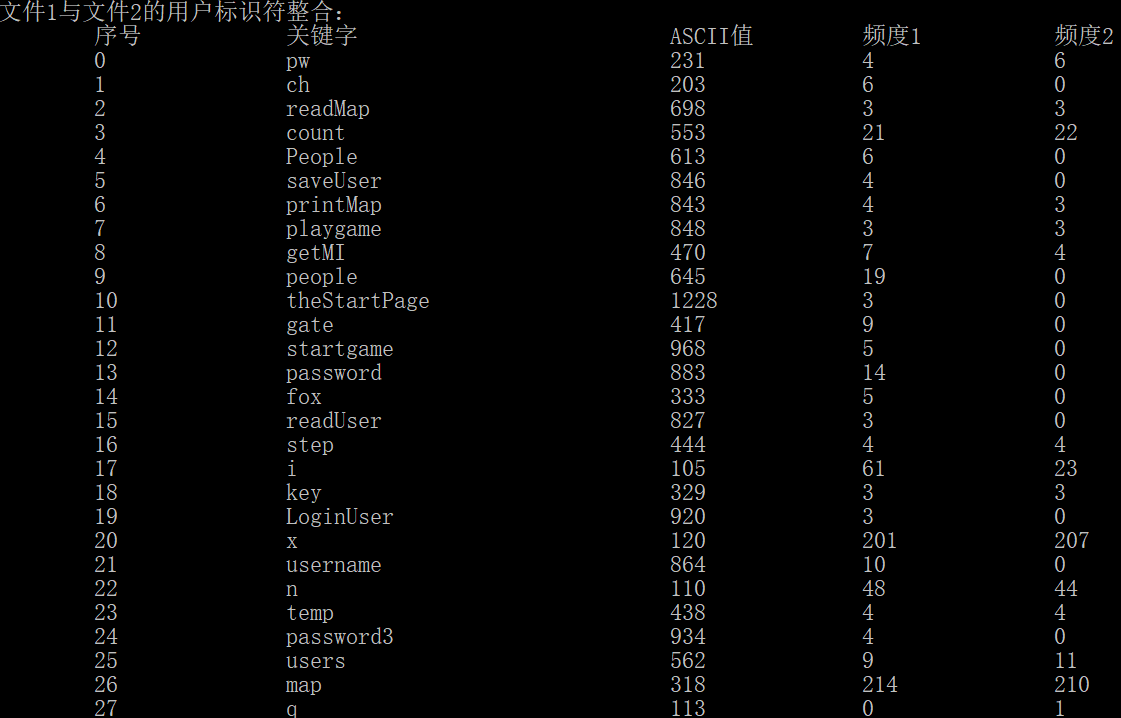
****

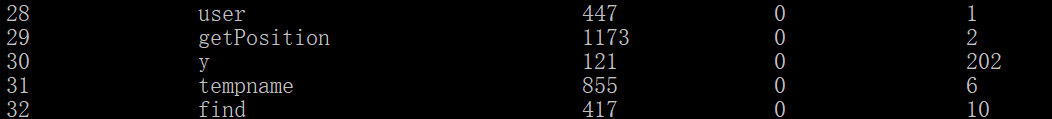
****

文件2的用户标识符统计：



文件1与文件2的用户标识符整合：

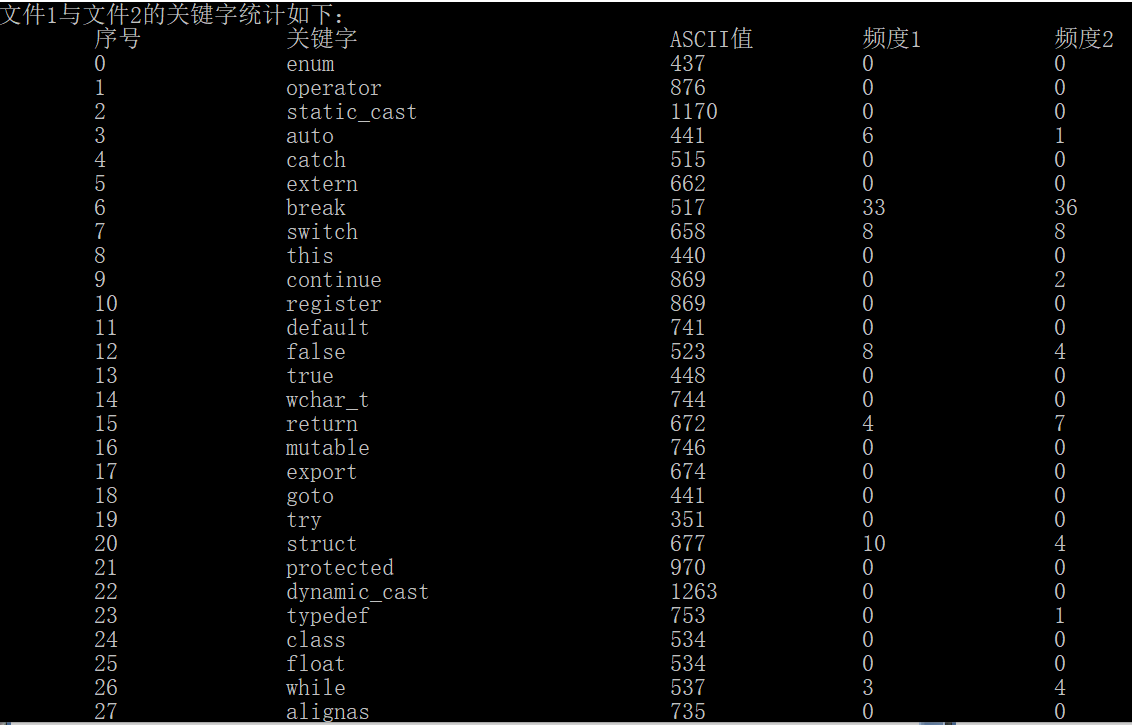


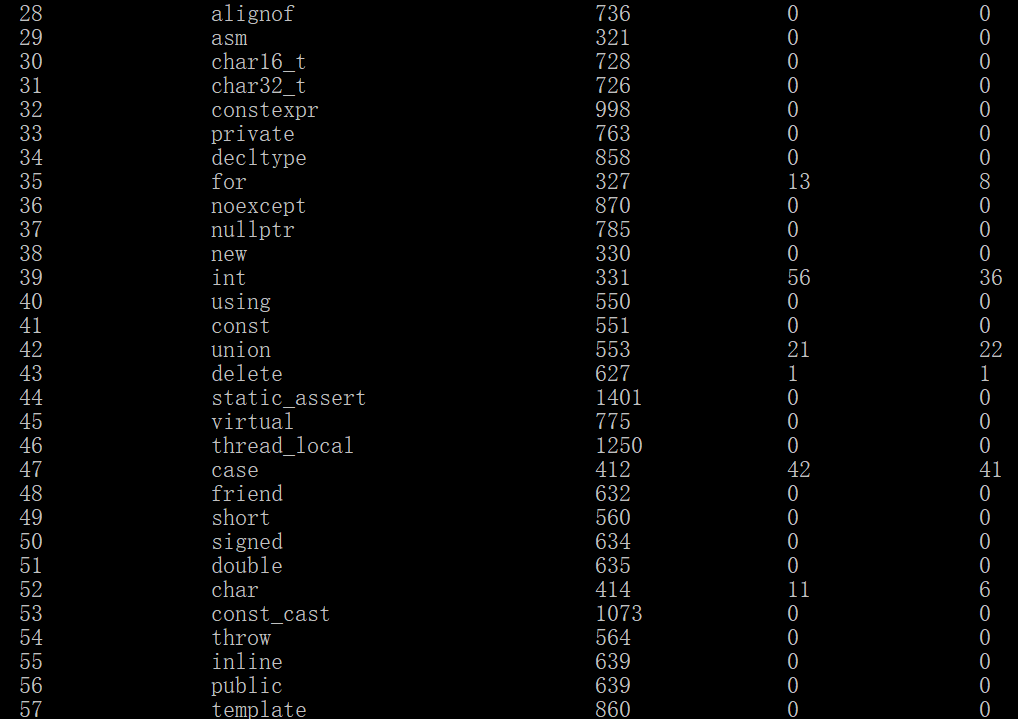


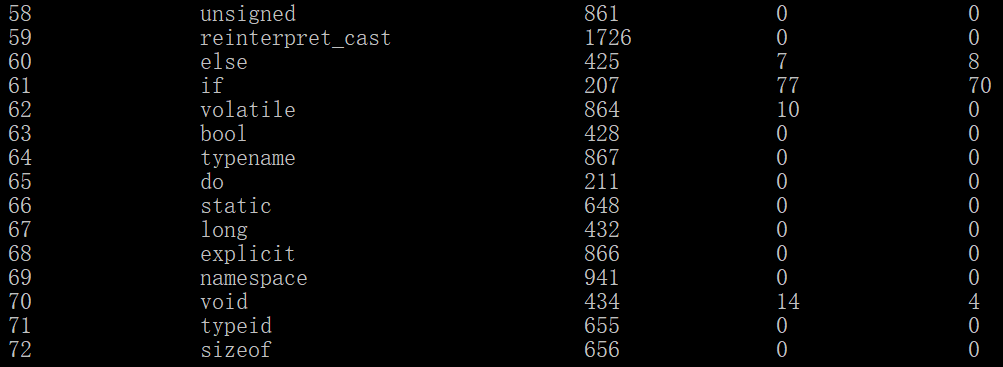
文件1与文件2以用户标识符为基准的相对距离：

QQ图片20190109090437.png

文件1与文件2的关键字统计：







文件1与文件2以关键字为基准的相对距离：

QQ图片20190109091133.png

综合两个相对距离，得出关于两个代码是否抄袭的结论：

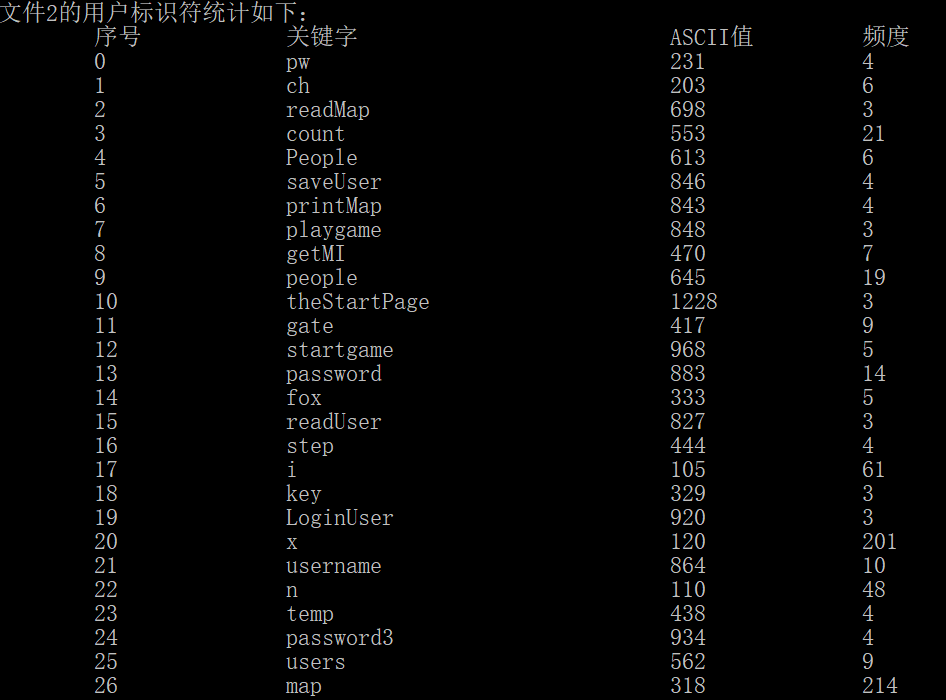
QQ图片20190109091356.png

**两个代码可能为全部抄袭：**

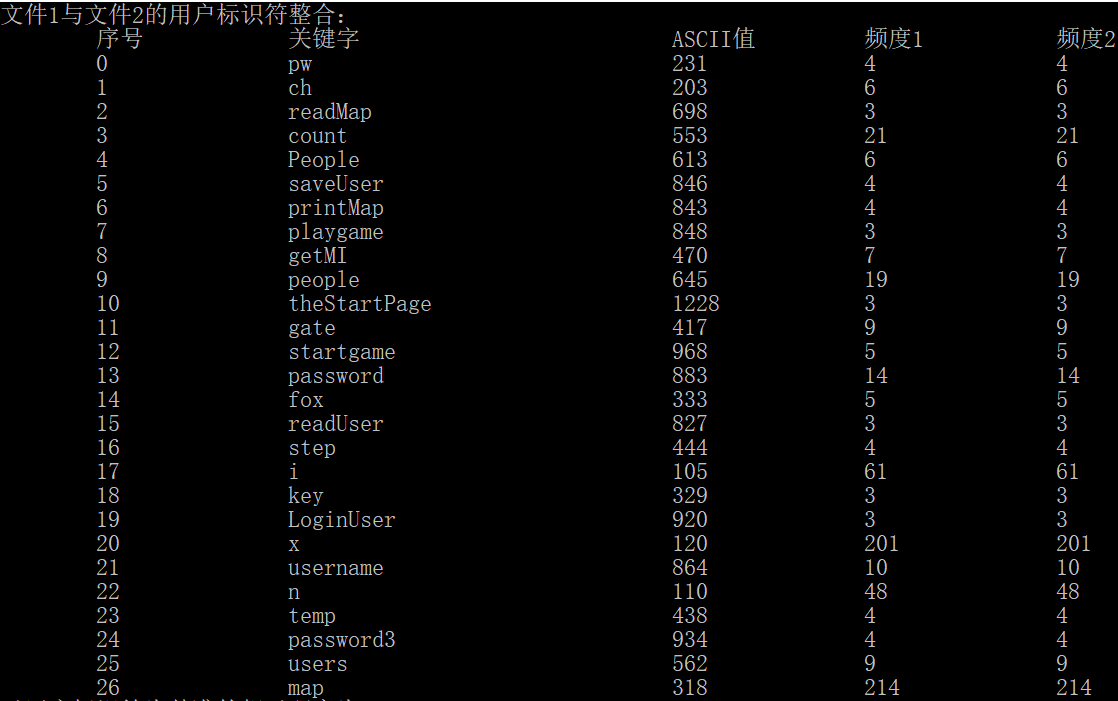
文件1的用户标识符统计：



文件2的用户标识符统计：



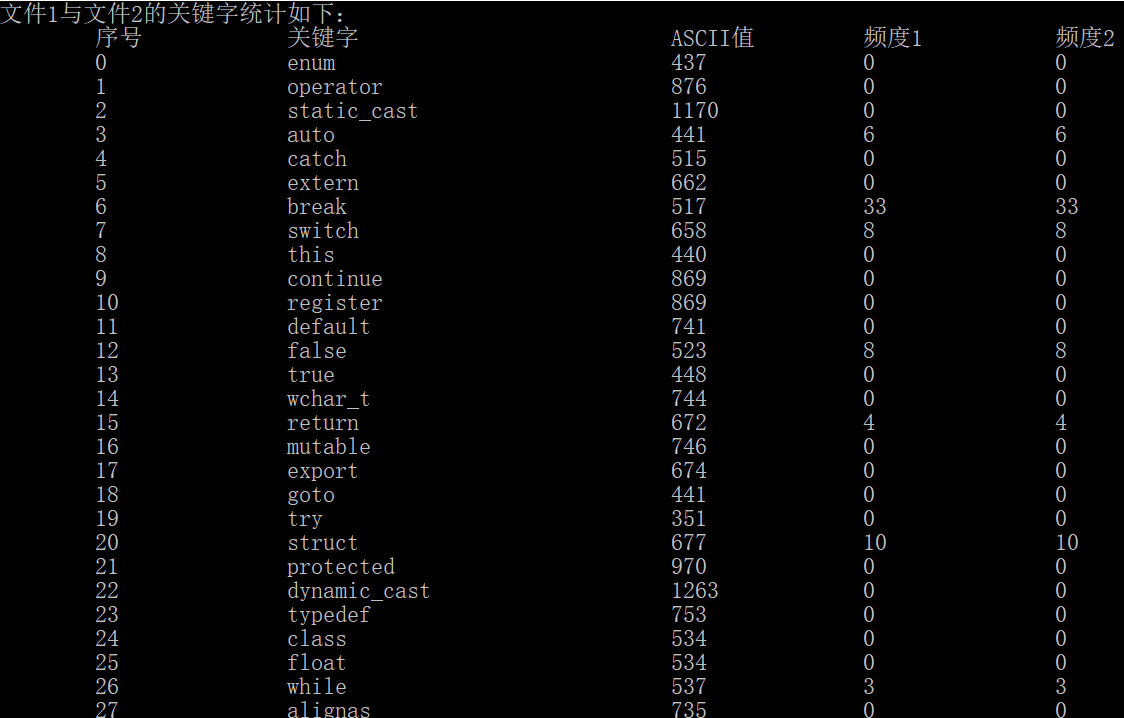
文件1与文件2的用户标识符整合：

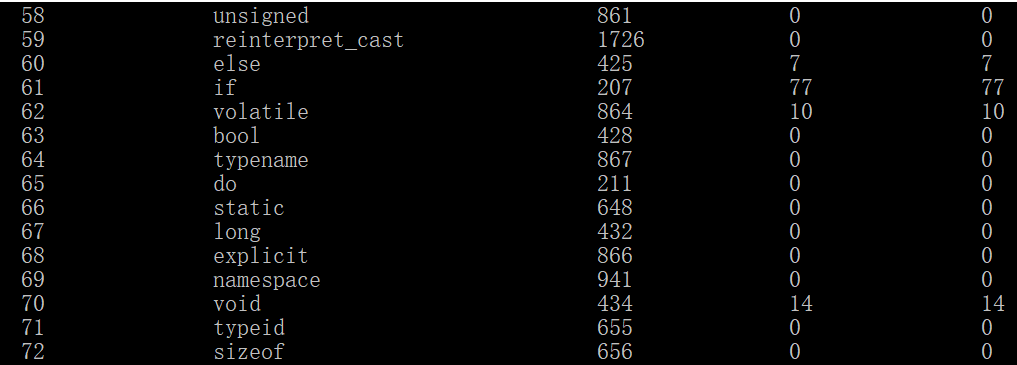


文件1与文件2以用户标识符为基准的相对距离：

QQ图片20190109092346.png

文件1与文件2的关键字统计：





文件1与文件2以关键字为基准的相对距离：

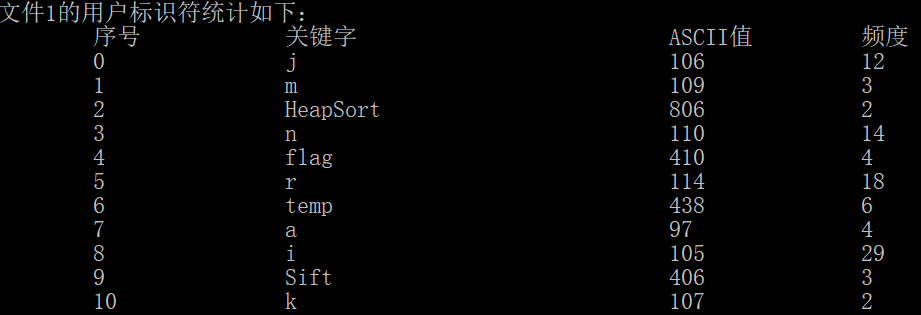
QQ图片20190109092818.png

综合两个相对距离，得出关于两个代码是否抄袭的结论：

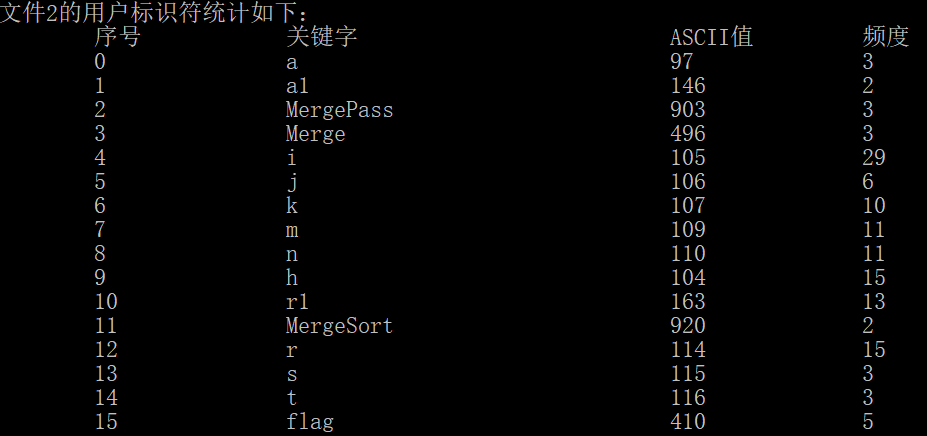
QQ图片20190109092822.png

**两个代码可能为部分抄袭：**

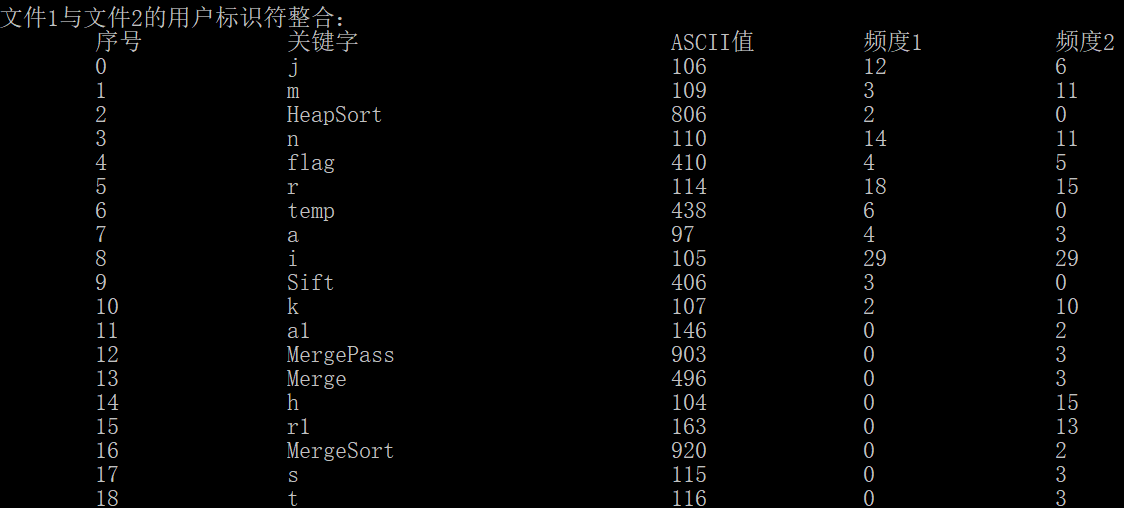
文件1的用户标识符统计：



文件2的用户标识符统计：



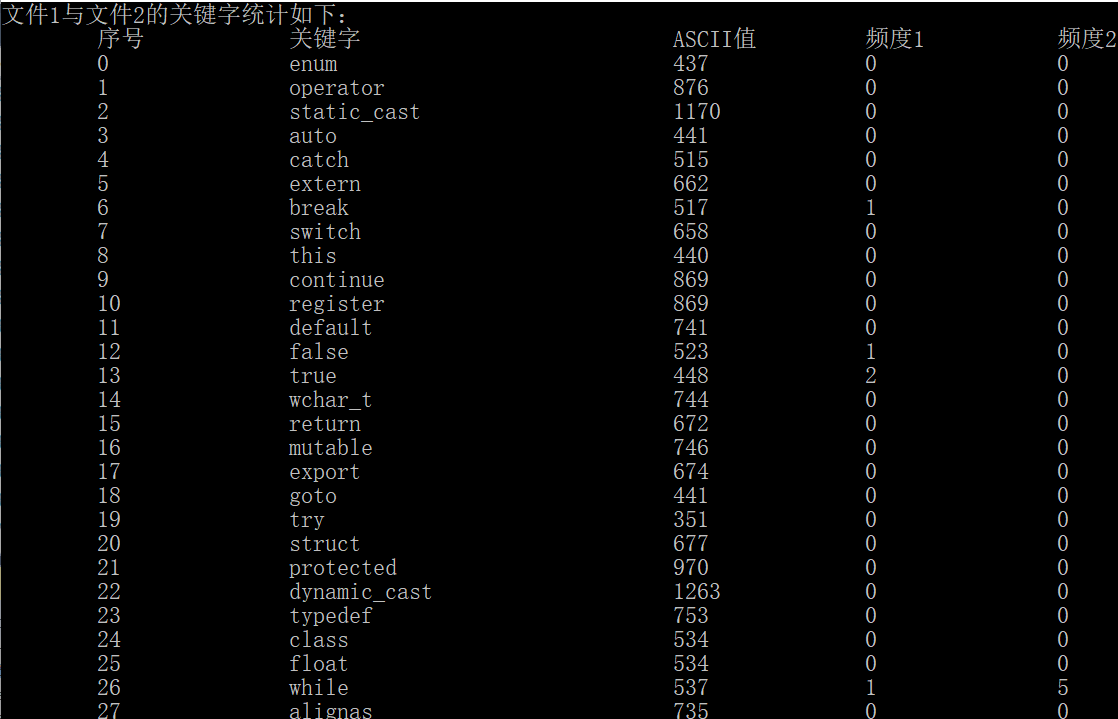
文件1与文件2的用户标识符整合：

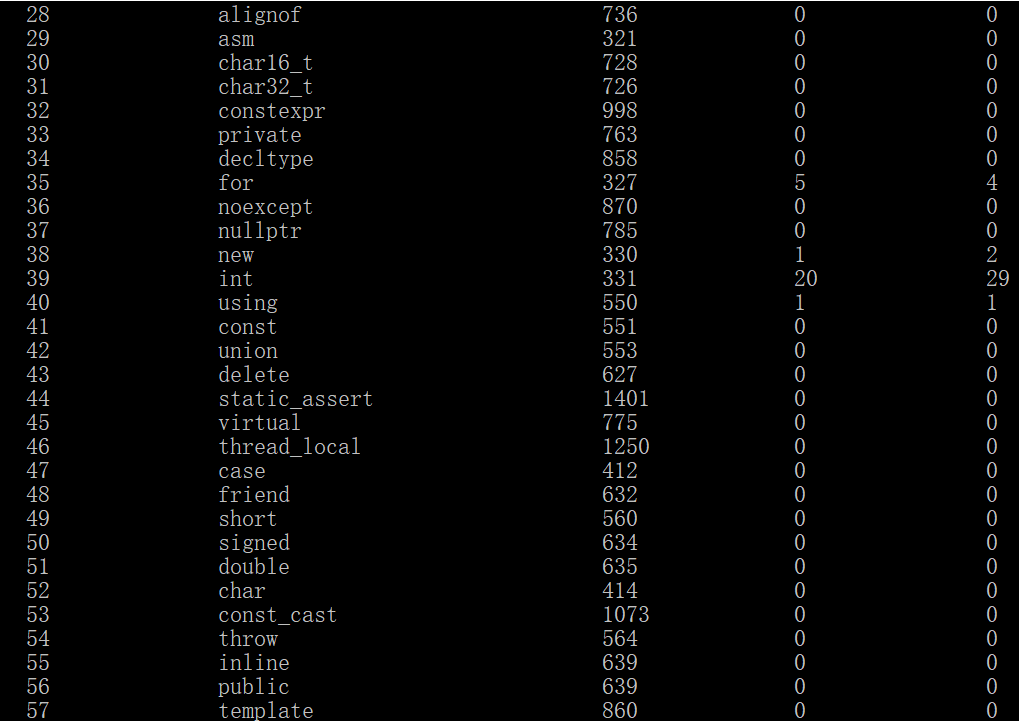


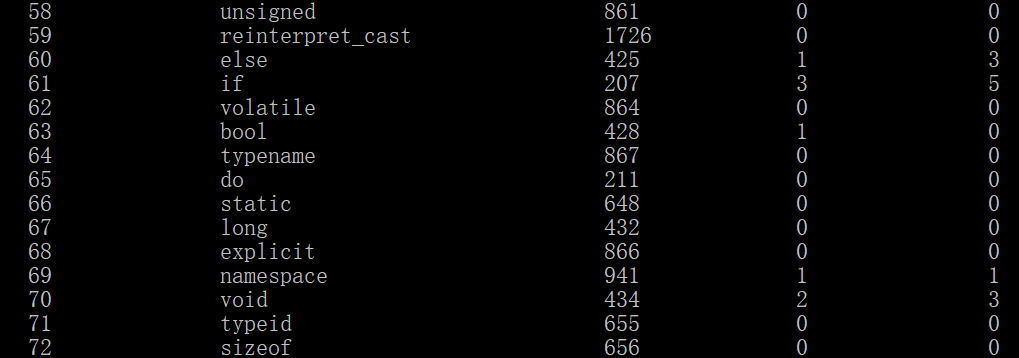
文件1与文件2以用户标识符为基准的相对距离：

QQ图片20190109103412.png

文件1与文件2的关键字统计：







文件1与文件2以关键字为基准的相对距离：

QQ图片20190109103425.png

综合两个相对距离，得出关于两个代码是否抄袭的结论：

**QQ图片20190109103436.png**

**8、自学知识**

在课程设计过程中，特别是在代码编写和调试过程中，自学了一些新的知识。例如ifstream类与ofstream类，包含于表头文件 #include <fstream> 。 其功能是:读取文件内容与将数据输入到文件。具体的函数说明是：通过定义ifstream类的对象，从硬盘中读取文件到内存，并通过ios::in方式，使文件以输入方式打开，即文件数据输入到内存；通过定义ofstream类的对象，从内存中读取文件到硬盘，并通过ios::out方式，使文件以输出方式打开，即内存数据输出到文件。除此之外，setiosflags()函数与setw()，包含于表头文件 #include<iomanip> 。 其功能是:使每一行的数据对齐。具体的函数说明是：通过设定setiosflags()函数的参数（如ios::left或ios：：right），令输出的数据每一行进行对齐（如左对齐或右对齐）；通过设定setw()函数的参数，给予要输出的数据一定的长度，这样可以保证下一个要输出的数据在纵向也是对齐的。

**9、课程设计心得体会**

这次课程设计，再一次夯实了我C++的基础，锻炼了我的思维能力与独立性。在做课程设计的过程中，我遇到了很多问题，比如题目要求用哈希表，但我并不知道该怎么使用，以及读入及输出文件的操作，但最终都圆满解决了。在这其中，我不仅在网上查找了相应的资料，同时也请教了同学，并从中学到了很多。此外，在这次课程设计中，我也有一些地方做得不够完善，比如用户标识符统计不够完善等，我还会继续努力，尽力将程序做到完美。

在编写程序的过程中，得到了部分同学的帮助，如：哈希表的使用，我从顾佳辉同学那里知道可以利用字符串的ASCII码创建哈希表；数据对齐则是我从王佳琦同学那里得到的知识。还有关于文件的操作，则是我从网上搜寻的资料而获得的解决方法。

这次课程设计让我明白，一个项目的成功，即需要自身过硬的能力，也需要团队合作的精神。在今后的学习生活中，我不仅要提高自己的专业水平，也要提高自己的团队协作能力，在团队中更好地发展。

参考书：

[1]《c++面向对象程序设计》 清华大学出版社 谭浩强著

[2]《数据结构（C++版）》清华大学出版社 王红梅、胡明、王涛著

[3] http://www.360doc.com/content/11/0718/14/54470\_134277081.shtml