這個作業我的實作方式如下:
將所有 ID 和文件內容分別讀入存取->不符合規則的詞變為空格，並將每個文件的詞整理起來->存取所有詞並計算各文件中的詞的出現頻率->建立 pthread->每一個 child thread 都計算自己和其他文件的餘弦相似係數和平均餘弦相似係數->由 main thread 找出最大相似餘弦係數

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <pthread.h>
#include <algorithm>
#include <unistd.h>
#include <sys/types.h>
#include <cmath>
#include <iomanip>
#include<ctime>
```

導入函式庫

```cpp
int list_size = 0;
double total_avg[52];
int file_size = 0, ele_size[1000] = { 0 };
string ID[52];
vector<vector<string>> str(52);
vector<string> word_list;
vector<vector<int>> frequen;
//==================Golbal variable=================//

```

定義全域變數
(後面會解釋各變數意義)

```cpp
int main()
{
    cout<<"Please enter the data file name(include filename extension .txt): ";
    string file;
    cin>>file;
    double Start = clock();
    ifstream file_in;
    file_in.open(file);
    string input, data[52];
    //=========================get data=======================//

    bool file_type = false;  //=====view data as ID or file content======//
    int i = 0, ele_s = 0;    //
    while (!file_in.eof())
    {
        getline(file_in, input);
        if (file_type == false)  //=========save the ID in golbal variable ID[]==========//
        {
            if(input[input.size()-1] == ' ')
               input.pop_back();
            ID[i] = input;
            file_type = true;
            i++;
        }
        else
        {
            data[file_size] = input;  //=========save the content in golbal variable data[]==========//
            file_type = false;
            file_size++;
        }
    }
    //==========differentiate ID and content==========//
```

讀取 input 檔案，請先輸入檔名(附檔名.txt 也要)，main function 找到檔案後才能讀取並盡興之後的動作(由於我不知道老師最後的測資檔名為何，故這樣的方式最為保險)，此橘色部分為讀入檔案的資料，並一行一行存取在不同地方，ID 存在全域變數 ID[]中，文件的內容暫存在 data[]中，之後再進行處理

輸入檔名後將檔案讀進 main function 中，此時設一個 clock start，以利之後 cpu time 的計算，，定義 input:存取每行的資料,data[]:存取文件內容用,file_type:由於 ID 和文件內容是一個 ID 一個內容，因此透過 file_type 來去做交換存取的動作,i,ele_s 都是陣列的 counter

處理讀取的檔案:while 持續進行直到檔案讀到 eof 時，getline 一行一行將資料存到 input 裡，此時 file_type 預設是 false，故執行存入 ID 的步驟:將 input 直接存進 ID[i]中，並判斷如果 ID 最後面有空格則去掉空格，此時令 file_type 變 true，告知程式下一步要存取文件資料，並讓 ID 的 counter i+1，再來為讀取資料，將資料直接存入全域變數 data[]中，並讓全域變數 file_size+1，*file_size 最後為所有文件的總數

此段為判斷是否為例外情況(一詞中含有數字忽略及文件有標點符號是為空格)

No 為傳入 judge 函式的參數，透過判斷 no 之值來決定對該文件內容作何種處裡

```
for (int i = 0; i < file_size; i++)
{
    int no = 0;
    for (long unsigned int j = 0; j < data[i].size(); j++)
    {
        no = 0;
        judge(data[i][j], no);
        if (no == 1)
        {
            data[i][j] = ' ';
        }
        else if (no == 2)
        {
            long unsigned int k = j;
            while (data[i][k] != ' ' && k != data[i].size() - 1)
            {
                data[i][k] = ' ';
                k++;
            }
            k = j;
            if (k != 0)
            {
                k = j - 1;
                while (data[i][k] != ' ' && k != 0)
                {
                    data[i][k] = ' ';
                    k--;
                }
                data[i][j] = ' ';
            }
        }
    }
}
//==============if content have punctuation or letter of the word have nuber,let it is space(filter)=============//
```

如 no=1，代表此位置視為空格

如 no=2, 代表該位置的”詞”需要被忽略，因此令兩個 while 使該詞前、後所有”字”變為空白，做到忽略之目的，方法為先往後令為空白後再往前，使整個詞為空白

見下方說明

```
13 void judge(char ele, int& no)
14 {
15     if (ele >= 65 && ele <= 90)
16         no = 0;
17     else if (ele >= 97 && ele <= 122)
18         no = 0;
19     else if (ele >= 48 && ele <= 57)
20         no = 2;
21     else
22         no = 1;
23 }
24 //=======determine whether it is a letter========//
25
```

此 function 用於判斷是否為純字母
如當前字(ele)為字母 no=0
如當前字(ele)為標點符號或空格 no=1
如當前字(ele)為數字 no=2:

```
177     for (int i = 0; i < file_size; i++)
178     {
179         for (long unsigned int j = 0; j < data[i].size(); j++)
180         {
181             string str_tmp;
182             if (data[i][j] != ' ')
183             {
184                 long unsigned int ptr = j;
185                 while (data[i][ptr] != ' ' && ptr != data[i].size() - 1)
186                 {
187                     str_tmp.push_back(data[i][ptr]);
188                     ptr++;
189                 }
190                 if (ptr == data[i].size() - 1 && data[i][ptr] != ' ')
191                     str_tmp.push_back(data[i][ptr]);
192                 str[i].push_back(str_tmp);
193                 j = ptr;
194                 ele_s++;
195             }
196         }
197         ele_size[i] = ele_s;
198         ele_s = 0;
199     }
```

此段功能為去掉空白部分，並將每一個文件中符合條件的”詞”丟到全域變數 str[][]中，並且 str[][a]的 a 指的是每一個”詞”，並非每一個”字” ex:this is a 在 str[][]中，str[][0]就是 this，str[][1]就是 is…以此類推

定義一個字串 str_tmp，如果當前字母不是空白，就代表當前為”詞”的開頭，因此由當前字母依序往後讀直到遇到空白(代表一個完整的詞)，讀取過程中依序把字 push 到 str_tmp 裡

由於上面 while 的判斷方式可能會少判斷最後的字，因此加一行這個，符合判斷就將最後一個字 push 到 str_tmp 中

當一個詞完全存進 str_tmp 中後，將整個字串 push 進 str[i][]中，也由於 str 是二維 string vector，因此直接將 str_tmp push 進 str[i][]可以使 str[i][]每一列元素都是一個”詞”而非一個”字”，做完每一個文件的所有詞後，用 ele_size[]計算每份文件的詞數，以利接下來的操作

```
203     for (int i = 0; i < file_size; i++)
204     {
205         for (long unsigned int j = 0; j < str[i].size(); j++)
206         {
207             long unsigned int repeat = 0,list_times = 0;
208             while (list_times != word_list.size() && word_list.size() != 0)
209             {
210                 if (str[i][j] == word_list[list_times])
211                 {
212                     repeat = 1;
213                     break;
214                 }
215                 list_times++;
216             }
217             if (repeat == 0)
218             {
219                 if(str[i].size()!=0)
220                   word_list.push_back(str[i][j]);
221             }
222         }
223     }
```

此段功能為建立所有詞的 list，以利後面的操作。依序讀取 str 所有詞，全域變數 word_list:所有詞的 list，repeat:查看當前詞是否存在於 word_list，list_times 為 word_list 的元素個數，它可以從頭檢閱 word_list 元素。作法為利用 list_times 持續檢閱 str 當前的詞是否和 word_list 的詞重複，如果重複 repeat=1 且跳出迴圈,如不重複則直到檢閱到 word_list 的最後一個元素(即該字詞未被收錄進 word_list)後跳出，再來判斷是否有重複，無重複則將該詞新增到 word_list 中。

```
226     for(int i=0; i<file_size; i++)
227     {
228         vector<int> f_tmp(word_list.size() , 0);
229         for(long unsigned int j=0; j<str[i].size(); j++)
230         {
231             for(long unsigned int k=0; k<word_list.size(); k++)
232             {
233                 if(str[i][j] == word_list[k])
234                     f_tmp[k]++;
235             }
236         }
237         frequen.push_back(f_tmp);
238     }
```

令一個 int 型態的 vector f_tmp，並初始化齊所有元素為 0，方便存入全域變數 frequen。此段程式碼為計算每個文件的詞在每個文件中出現的頻率(詞頻)，方式為依序讀取每個文件中的每個詞，並且讀取每個詞時要判斷它和 word_list 中的元素是否相同，相同的話則讓 f_tmp 與 word_list 同元素個數位置的值+1，如此便能算出一個文件出現的詞的頻率了。

```
241     pthread_t threads;
242     for(int i=0; i<file_size; i++)
243     {
244         int arg = i;
245         pthread_create(&threads,NULL,child,&arg);  //create//
246         usleep(1);
247         cout<<"[Main thread]: create TID:"<<tid<<",DocID:"<<ID[i]<<endl;
248         pthread_join(threads , NULL);  //end child_thread//
249     }
250     //========create the pthread========//
```

這段功能為建立子 thread，arg 為傳入 child function 的參數，指的是第幾個子 thread,由於 create pthread 後 main 和 child 是同時運行，因此為了不讓 main 在未得到 thread 時就 print TID，因此先讓 main 等 1 毫秒，此時 TID 已經得到，這時就能 print。Pthread_join 為等待子 thread 結束

```
47 void* child(void*arg)
48 {
49     double Start = clock();
50     tid = gettid();  //get tid//
51     usleep(1);
52     int n = *(int *)arg;  //current file//
53     cout<<"[TID="<<tid<<"] DocID:"<<ID[n]<<" [";
54     for(long unsigned int i=0; i<frequen[n].size()-1; i++)
55         cout<<frequen[n][i]<<",";
56     cout<<frequen[n][frequen[n].size()-1]<<"]"<<endl;
57
```

設一個 closk start 表示開始計算 child 的時間

獲取自己 thread 本身的 TID

為了避免 main 和 child 的 print 會錯亂，因此獲取 TID 後停止個 1 毫秒作為保險，並令 n 是當前為第幾個 child thread，隨後 print 題目要求之內容

```
26 struct cos
27 {
28     int value;
29     string ele1;
30     string ele2;
31 };
32 struct cos cos_list[2700];
```

再往下介紹之前我要先介紹 struct cos 和 cos_list[]，後面的介紹為計算餘弦相似係數，但每一個 thread 都要計算一次自己與其他人的係數很浪費時間，因此我利用 DP 的概念將已計算過的”值”和當前計算的 2 份文件記錄在 cos_list[]中，這樣下次有同樣的元素需要計算時便可直接從 cos_list 中讀取，可省下一半計算花費的時間

```
57   int cos_list_count = list_size , in_list = 0 ,val_count = 0;
58   double val[52] , avg = 0;
```

全域變數 list_size 為 cos_list 的大小
cos_list_count 為 cos_list[]下一個空的元素的個數，方便 push 元素進 cos_list
in_list 為查看想要計算的元素是否在 cos_list 中
val[]為記錄當前文件與其他每個文件計算後的值，
val_count 為 val[]下一個空的元素的個數，方便 push 元素進 val
avg 為所有值的平均

```
59   for(int i=0; i<file_size; i++)
60   {
61       double sum_up = 0,abs_vs = 0,abs_vx = 0 , ans = 0;
62       if(i == n)
63           continue;
64       for(int j=0; j<list_size; j++)
65       {
66           //====check if the element to be calculated is in cos_list,yes:ans = value of cos_list no:calculate
67           if((ID[n] == cos_list[j].ele1 && ID[i] == cos_list[j].ele2) || (ID[i] == cos_list[j].ele2 && ID[n] == cos_list[j].ele1))
68           {
69               ans = cos_list[j].value;
70               in_list = 1;
71           }
72       }
73       if(in_list == 0)  //not in cos_list,calculate
74       {
75           for(long unsigned int k=0; k<frequen[i].size(); k++)
76           {
77               sum_up = sum_up + frequen[n][k]*frequen[i][k];
78               abs_vs = abs_vs+pow(frequen[n][k],2);
79               abs_vx = abs_vx+pow(frequen[i][k],2);
80           }
81           ans = sum_up/(sqrt(abs_vs)*sqrt(abs_vx));
82           cos_list[cos_list_count].value = ans;
83           cos_list[cos_list_count].ele1 = ID[n];
84           cos_list[cos_list_count].ele2 = ID[i];     //record value and element of calculate in cos_list
85           list_size++;
86           cos_list_count++;
87       }
88       cout<<fixed<<setprecision(4);
89       cout<<"[TID="<<tid<<"] cosine("<<ID[n]<<","<<ID[i]<<")=";
90       cout<<ans<<endl;
91
92       val[val_count] = ans;
93       val_count++;
94   }
```

所有變數為計算所需

如 i=n 代表讀取到自己，自己不需要跟自己計算因此跳過以下的內容

查看一遍 cos_list 所有的值，如當前要計算的元素=存在 cos_list 中的元素，代表已經被計算過了，直接令 ans=存在 cos_list 的值，並令 in_list=1

如果 in_list=0 代表 cos_list 中沒有此計算的紀錄，則進行計算

$$Sim(V_s,V_x)=\cos(V_s,V_x)=\frac{V_s\cdot V_x}{|V_s|\times|V_x|}=\frac{\sum_{i=1}^{n}v_{s,i}\times v_{x,i}}{\sqrt{\sum_{i=1}^{n}v_{s,i}^2}\times\sqrt{\sum_{i=1}^{n}v_{x,i}^2}}$$

sum_up:綠色框住的部分
sbs_vs:藍色框住的部分
abs_vx:橘色框住的部分
計算後將答案和參與計算的元素存入 cos_list 中，並讓 cos_list_count+1 計算完畢後即輸出並把答案記錄在 val[]中以利之後計算平均

```
96    for(int i=0; i<val_count; i++)  //calculate the average of each value
97       avg = avg+val[i];
98    avg = avg/val_count;
99    cout<<"[TID="<<tid<<"] AVG_cosine: "<<avg<<endl;
100   total_avg[n] = avg;
101   double End = clock();
102   cout<<fixed<<setprecision(0);
103   cout<<"[TID="<<tid<<"] CPU time: "<<End-Start<<"ms"<<endl;
104   return NULL;
105 }
```

將剛剛存取的所有值全部加起來存於 avg 變數，並算出平均，隨後輸出並將平均的值加入全域變數 total_avg[]中，以利 main 找出最大的平均餘弦係數

設一個 clock End 用 End-Start 可得出 cpu time

至此 child function 結束

```
251   cout<<fixed<<setprecision(4);
252   double high = 0;
253   int local;
254   for(int i=0; i<file_size; i++)
255   {
256      if(total_avg[i]>high)
257      {
258         high = total_avg[i];
259         local = i;
260      }
261   }
262   cout<<"[Main thread] KeyDocID:"<<ID[local]<<" Highest Average Cosine: "<<high<<endl;
263   cout<<fixed<<setprecision(0);
264   double End = clock();
265   cout<<"[Main thread] CPU time: "<<End-Start<<"ms"<<endl;
266 }
```

在所有 child thread 跑完之後就由 main 計算最高平均係數並存入 high 中，做法為依序讀一遍所有平均數的 total_avg[]找出最大的，由於第 i 個 thread 的平均就是 total_avg[i]因此是第幾個文件有最大也一併找到了(local=i)

設一個 clock End 用 End-Start 可得出 main 的 cpu time 至此程式碼介紹結束

```
jacky@jacky-VirtualBox:~/Desktop$ g++ /home/jacky/Desktop/thread.cpp -o/home/jacky/Desktop/thread.out -Wall
jacky@jacky-VirtualBox:~/Desktop$ ./thread.out
Please enter the data file name(include filename extension .txt): data.txt
[Main thread]: create TID:39420,DocID:0001
[TID=39420] DocID:0001 [1,1,1,1,0,0,0]
[TID=39420] cosine(0001,0002)=0.7500
[TID=39420] cosine(0001,0003)=0.7906
[TID=39420] cosine(0001,0004)=0.5000
[TID=39420] AVG_cosine: 0.6802
[TID=39420] CPU time: 49ms
[Main thread]: create TID:39421,DocID:0002
[TID=39421] DocID:0002 [1,1,1,0,1,0,0]
[TID=39421] cosine(0002,0001)=0.7500
[TID=39421] cosine(0002,0003)=0.4743
[TID=39421] cosine(0002,0004)=0.2500
[TID=39421] AVG_cosine: 0.4914
[TID=39421] CPU time: 21ms
[Main thread]: create TID:39422,DocID:0003
[TID=39422] DocID:0003 [0,1,2,2,0,1,0]
[TID=39422] cosine(0003,0001)=0.7906
[TID=39422] cosine(0003,0002)=0.4743
[TID=39422] cosine(0003,0004)=0.6325
[TID=39422] AVG_cosine: 0.6325
[TID=39422] CPU time: 21ms
[Main thread]: create TID:39423,DocID:0004
[TID=39423] DocID:0004 [0,1,0,1,0,1,1]
[TID=39423] cosine(0004,0001)=0.5000
[TID=39423] cosine(0004,0002)=0.2500
[TID=39423] cosine(0004,0003)=0.6325
[TID=39423] AVG_cosine: 0.4608
[TID=39423] CPU time: 27ms
[Main thread] KeyDocID:0001 Highest Average Cosine: 0.6802
[Main thread] CPU time: 354ms
```

請將測資文件和.cpp 和.out 檔存在同一位置

1. 建立.out 檔方式為 :在 terminal 輸入 g++ / 存.cpp 的路徑 -o/要存.out 檔的路徑 -Wall
2. 執行方式為 :在 terminal 輸入./檔名.out
3. **請務必要輸入測資文件名稱(包含.txt)**
4. 結果