CS205 C/ C++ Programming Assignment 4

Name: 傅伟堡(Weibao Fu)

SID:11812202

Part 1-Analysis

The problem is to display on the standard output the name of the block to which most characters belong to. The first step I consider is to load the character set table into the program. The second thing is to calculate the number of the set in the file. At last, we use print() to output the result.

Part 2-Code

display.cpp

```
#include<stdio.h>
#include<iostream>
#include<fstream>
#include<string.h>
#include<sstream>
#include"utf8.h"
#define Maxn 1000
#define Max_len 1e18
using namespace std;

struct info{
    int count = 0;
    char name[Maxn];
    int start=0;
    int end=0;
};

struct info cnt[Maxn];
bool load();
void find(int target);
void print();

int main(){
    if(!load()) return 0;
    else{
```

```cpp
        unsigned char *p = new unsigned char[Maxn];
        int bytes_in_char;
        unsigned int codepoint;
        stringstream buffer;
        buffer<<cin.rdbuf();
        string contents(buffer.str());
        p = (unsigned char*)contents.c_str();
        while(*p){
            codepoint = utf8_to_codepoint(p, &bytes_in_char);
            if(codepoint){
                find(codepoint);
                _utf8_incr(p);
            }else{
                printf("%c Invalid UTF-8\n", *p);
                p++;
            }
        }
        print();
    }
    return 0;
}

bool load(){
    ifstream file("Blocks.txt");
    if(!file.is_open()){
        cout<<"Can not open the file!"<<endl;
        return false;
    }else{
        int i = 0;
        char *str1 = new char[Maxn];
        while(file.getline(str1,Maxn)){
            if(str1[0]=='#'||strlen(str1)==0) continue;
            else{
                sscanf(str1,"%x..%x; %
[^\n]",&cnt[i].start,&cnt[i].end,cnt[i].name);
                i++;
            }
        }
        cout<<"Load the file successfully\n";
        file.close();
        return true;
    }
}

void find(int target){
    for(int i=0;i<Maxn;i++){
        if(cnt[i].start<=target&&cnt[i].end>=target){
            cnt[i].count++;
            break;
        }
    }
}

void print(){
    int max_num=0;
    int max_index=0;
    for(int i=0;i<Maxn;i++){
        if(cnt[i].count>max_index){
                max_index = i;
                max_num = cnt[i].count;
            }
        }
    for(int i=0;i<Maxn;i++){
        if(cnt[i].count==0) ;
        else{
            printf("There are %d characters belong to 
%s\n",cnt[i].count,cnt[i].name);
        }
    }
    printf("Most characters belong to %s, there have %d characters of of this 
set in this file\n",cnt[max_index].name,max_num);
}
```

utf8.cpp(Given)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include "utf8.h"

extern unsigned char *codepoint_to_utf8(const unsigned int  cp,
                                        unsigned char      *val) {

   if (cp <= 0x7f) {
      sprintf((char *)val, "%c", (char)cp);
   } else {
      if ((cp >= 0x80) && (cp <= 0x07ff)) {
         // 2-char UTF8 character
         val[0] = cp / 64 + 192;
         val[1] = cp % 64 + 128;
         val[2] = '\0';
      } else if (((cp >= 0x0800) && (cp <= 0xd7ff))
               || ((cp >= 0xe000) && (cp <= 0xffff))) {
         // 1110xxxx 10xxxxxx 10xxxxxx
         val[0] = cp / 4096 + 224;
         val[1] = (cp % 4096) / 64 + 128;
         val[2] = cp % 64 + 128;
         val[3] = '\0';
      } else {
         val[0] = cp / 262144 + 240;
         val[1] = (cp % 262144) / 4096 + 128;
         val[2] = (cp % 4096) / 64 + 128;
         val[3] = cp % 64 + 128;
         val[4] = '\0';
      }
   }
   return val;
}

extern int  isutf8(const unsigned char *u) {
   // Validate utf8 character.
```

```c
   // Returns the length, 0 if invalid.
   int len = 0;

   if (u) {
      if (*u < 0xc0) {
         len = 1;
      } else {
         if ((*u & 0xe0) == 0xc0) {
            // U-00000080 - U-000007FF : 110xxxxx 10xxxxxx
            len = 2;
         } else if ((*u & 0xf0) == 0xe0) {
            // U-00000800 - U-0000FFFF : 1110xxxx 10xxxxxx 10xxxxxx
            len = 3;
         } else if ((*u & 0xf8) == 0xf0) {
            // U-00010000 - U-001FFFFF : 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
            len = 4;
         } else {
            // malformed UTF-8 character
            return 0;
         }
         // Check that the UTF-8 character is OK
         int i;
         for (i = 1; i < len; i++ ) {
            if ((u[i] & 0xC0) != 0x80) {
               return 0;
            }
         }
      }
   }
   return len;
}

extern unsigned char *decimal_to_utf8(const unsigned int  d,
                                      unsigned char      *val) {
    // Works like chr() in Oracle: convert to hex, then convert hex to
    // character. Returns NULL if invalid UTF-8 character.
    char    hex[50];
    int     len;
    int     i;

    sprintf(hex, "%X", d);
    len = strlen(hex);
    if ((len > 8) || (len % 2)){
        return NULL;  // Invalid UTF-8
    } else {
       for (i = 0; i < len/2; i++) {
          val[i] = 16 * (isdigit(hex[2*i]) ? hex[2*i] - '0' :
                                10 + hex[2*i] - 'A')
                   + (isdigit(hex[2*i+1]) ? hex[2*i+1] - '0' :
                                10 + hex[2*i+1] - 'A');
       }
       val[i] = '\0';
    }
    if (isutf8(val) == 0) {
       return NULL;
    }
    return val;
}
```

```
extern unsigned int utf8_to_codepoint(const unsigned char *u,
                                      int                 *lenptr) {
   // Returns 0 if something goes wrong
   // Passes back the length
   unsigned int cp = 0;

   *lenptr = 0;
   if (u) {
      if (*u < 0xc0) {
         cp = (unsigned int)*u;
         *lenptr = 1;
      } else {
         *lenptr = isutf8(u);
         if (*lenptr == 0) {
            return 0;
         }
         switch (*lenptr) {
            case 2:
                 cp = (u[0] - 192) * 64 + u[1] - 128;
                 break;
            case 3:
                 cp = (u[0] - 224) * 4096
                    + (u[1] - 128) * 64 + u[2] - 128;
                 break;
            default:
                 cp = (u[0] - 240) * 262144
                     + (u[1] - 128) * 4096
                     + (u[2] - 128) * 64 + u[3] - 128;
                 break;
         }
      }
   }
   return cp;
}

//
//  Returns the length of s in CHARACTERS, not bytes
//
extern int utf8_charlen(unsigned char *s) {
   int len = 0;
   unsigned char *p = s;

   while (*p != '\0') {
      len++;
      // Beware that the macro increments p
      _utf8_incr(p);
   }
   return len;
}

extern int utf8_bytes_to_charpos(unsigned char *s, int pos) {
  unsigned char *p = s;
  int charcnt = 0;

  while (p < &(s[pos])) {
     _utf8_incr(p);
     charcnt++;
  }
  return charcnt;
}

extern int utf8_charpos_to_bytes(unsigned char *s, int pos) {
  int bytecnt = 0;
  int charcnt = 0;
  int len;

  while (charcnt < pos) {
     len = isutf8((const unsigned char *)&(s[bytecnt]));
     if (len == 0) {
        return -1; // Invalid UTF-8
     }
     bytecnt += len;
     charcnt++;
  }
  return bytecnt;
}

// strchr-like function for utf8 characters. Returns NULL if
// "needle" is an invalid utf8 character.
extern unsigned char *utf8_search(const unsigned char *haystack,
                                  const unsigned char *needle) {
    unsigned char *p = (unsigned char *)NULL;

    if (haystack
        && needle
        && isutf8(needle)) {
      p = (unsigned char *)strstr((char *)haystack, (char *)needle);
    }
    return p;
}
```

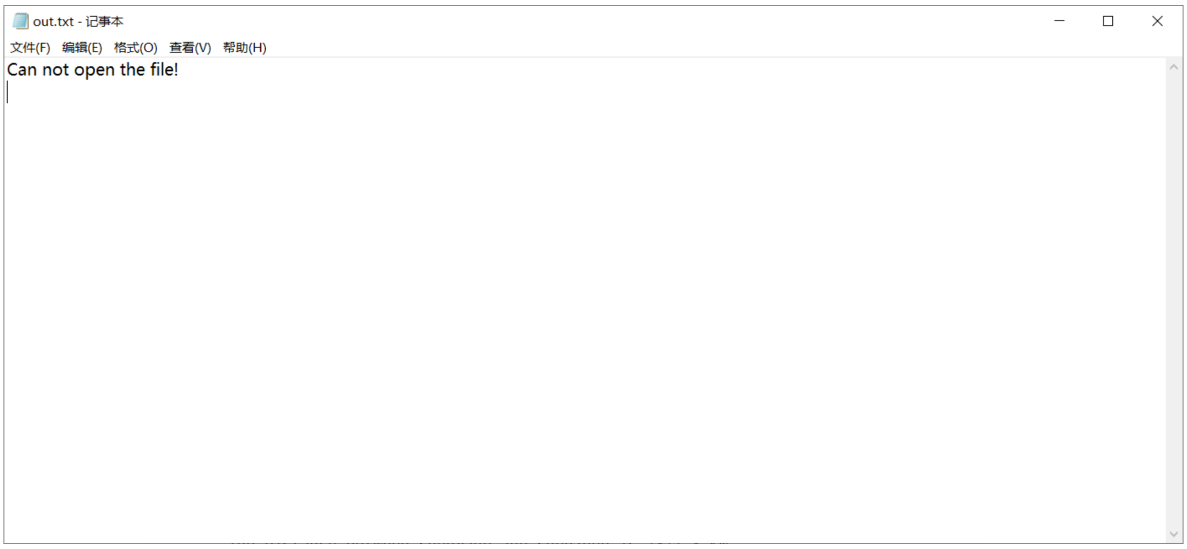
utf8.h(Given)

```
#ifndef UTF8_H
#define UTF8_H

// --- UTF-8 ---

// Macro copied from the sqlite code
// Kind of utf8-aware p++
#define _utf8_incr(zIn) {                        \
      if ((*(zIn++)) >= 0xc0) {                        \
         while ((*zIn & 0xc0) == 0x80) { zIn++; }      \
      }                                                \
   }

// Utf8-aware p--
#define _utf8_decr(p) { while ((*(--p) & 0xc0) == 0x80); }

#define _utf8_copychar(pin, pout) {       \
      if ((*pout = *pin) >= 0xc0) {                    \
         pin++;                                \
         pout++;                           \
         while ((*pin & 0xc0) == 0x80) {   \
           *pout = *pin;                   \
           pin++;                          \
           pout++;                         \
         }                                 \
      } else {                             \
         if (*pout) {                      \
           pin++;                          \
           pout++;                         \
         }                                 \
      }                                    \
   }

extern          int   utf8_charlen(unsigned char *p);
extern          int   utf8_bytes_to_charpos(unsigned char *s, int pos);
// utf8_charpos_to_bytes returns -1 if s isn't a valid UTF-8 string
extern          int   utf8_charpos_to_bytes(unsigned char *s, int pos);
// utf8_to_codepoint returns 0 if conversion fails. If it succeeds,
// the value pointed by lenptr is set to the number of bytes of the
// UTF-8 character
extern unsigned int   utf8_to_codepoint(const unsigned char *u,
                                        int *lenptr);
// Returns the length in bytes, 0 if invalid
extern          int   isutf8(const unsigned char *u);

// strchr-like function for utf8 characters. Returns NULL if
// "needle" is an invalid utf8 character.
extern unsigned char *utf8_search(const unsigned char *haystack,
                                  const unsigned char *needle);
// --------------------------------------------------------
// The second argument to these functions must be an
// array of at least 5 elements (it will store a single
// utf8 character). A pointer to this array will be
// returned if functions succeed.
// --------------------------------------------------------
extern unsigned char *decimal_to_utf8(const unsigned int d,
                                      unsigned char *utf8);
extern unsigned char *codepoint_to_utf8(const unsigned int cp,
                                        unsigned char *utf8);

#endif
```

Part 3-Result & Verification

Test case #1

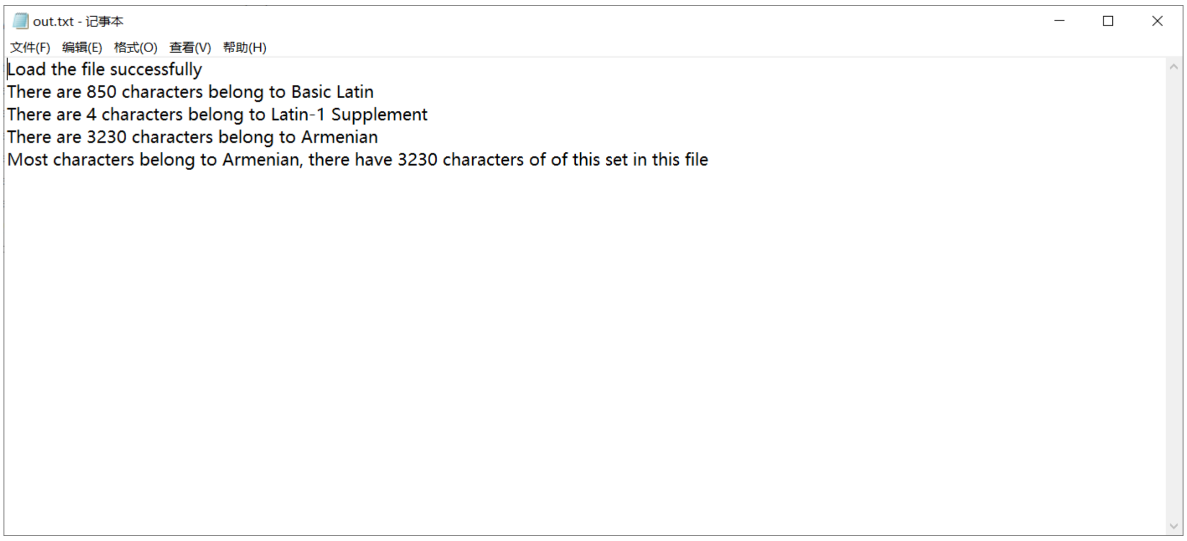
```
Set the wrong name of the information file.
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample.txt>out.txt
Output: Can not open the file!
```



```
Can not open the file!
```

Test case #2

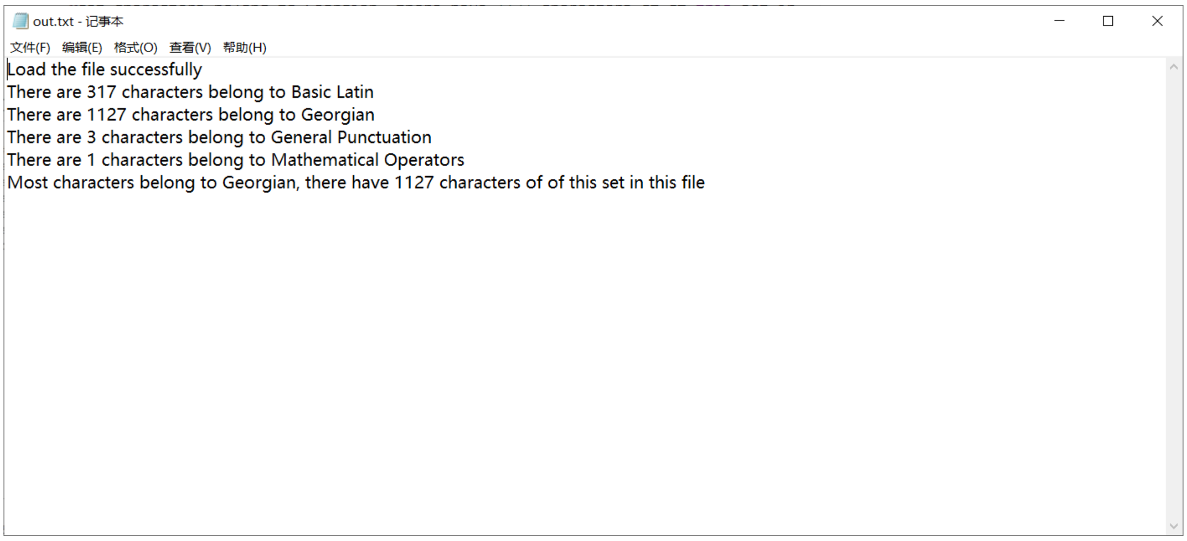
```
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample.txt>out.txt
Output: Load the file successfully
        There are 850 characters belong to Basic Latin
        There are 4 characters belong to Latin-1 Supplement
        There are 3230 characters belong to Armenian
        Most characters belong to Armenian, there have 3230 characters of of
this set in          this file
```



```
Load the file successfully
There are 850 characters belong to Basic Latin
There are 4 characters belong to Latin-1 Supplement
There are 3230 characters belong to Armenian
Most characters belong to Armenian, there have 3230 characters of of this set in this file
```

Test case #3

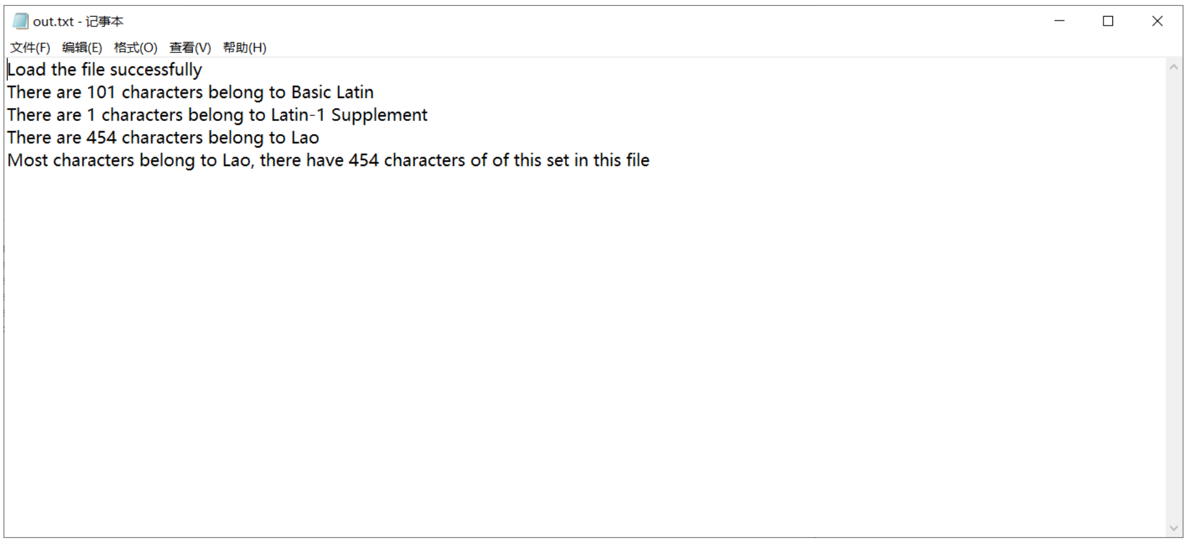
```
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample2.txt>out.txt
Output: Load the file successfully
        There are 317 characters belong to Basic Latin
        There are 1127 characters belong to Georgian
        There are 3 characters belong to General Punctuation
        There are 1 characters belong to Mathematical Operators
        Most characters belong to Georgian, there have 1127 characters of of
this set in          this file
```



```
Load the file successfully
There are 317 characters belong to Basic Latin
There are 1127 characters belong to Georgian
There are 3 characters belong to General Punctuation
There are 1 characters belong to Mathematical Operators
Most characters belong to Georgian, there have 1127 characters of of this set in this file
```

Test case #4

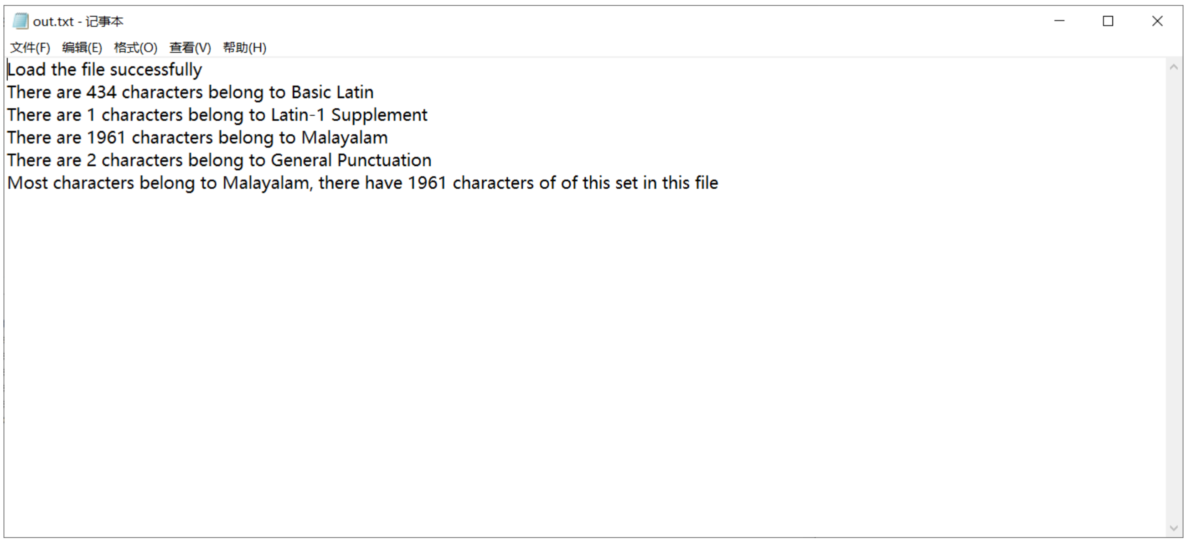
```
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample3.txt>out.txt
Output: Load the file successfully
        There are 101 characters belong to Basic Latin
        There are 1 characters belong to Latin-1 Supplement
        There are 454 characters belong to Lao
        Most characters belong to Lao, there have 454 characters of of this set
in this         file
```



```
Load the file successfully
There are 101 characters belong to Basic Latin
There are 1 characters belong to Latin-1 Supplement
There are 454 characters belong to Lao
Most characters belong to Lao, there have 454 characters of of this set in this file
```

Test case #5

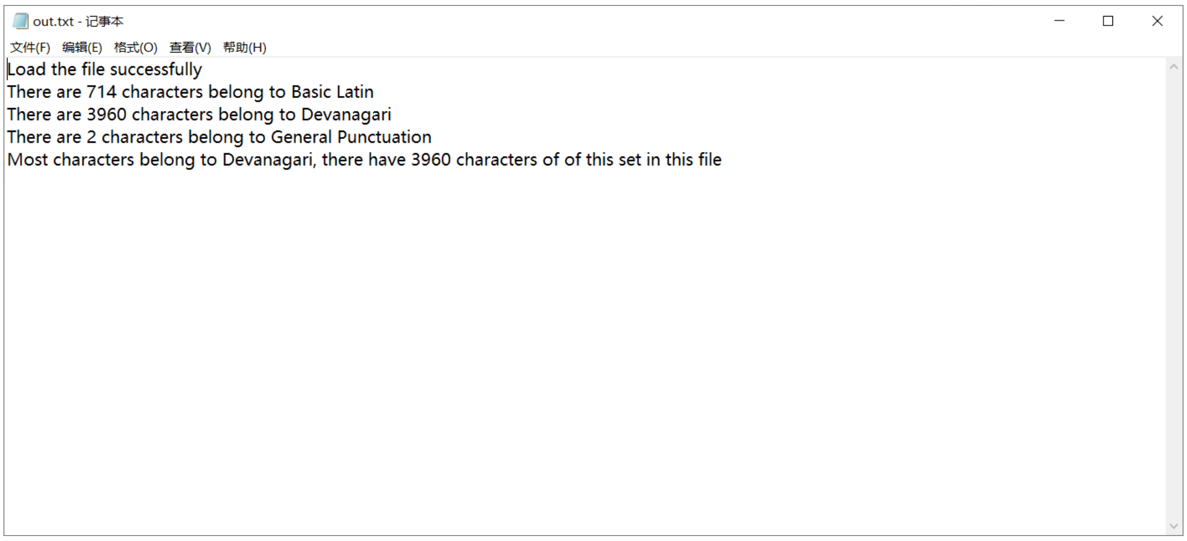
```
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample4.txt>out.txt
Output: Load the file successfully
        There are 434 characters belong to Basic Latin
        There are 1 characters belong to Latin-1 Supplement
        There are 1961 characters belong to Malayalam
        There are 2 characters belong to General Punctuation
        Most characters belong to Malayalam, there have 1961 characters of of
this set in         this file
```



```
Load the file successfully
There are 434 characters belong to Basic Latin
There are 1 characters belong to Latin-1 Supplement
There are 1961 characters belong to Malayalam
There are 2 characters belong to General Punctuation
Most characters belong to Malayalam, there have 1961 characters of of this set in this file
```

Test case #6

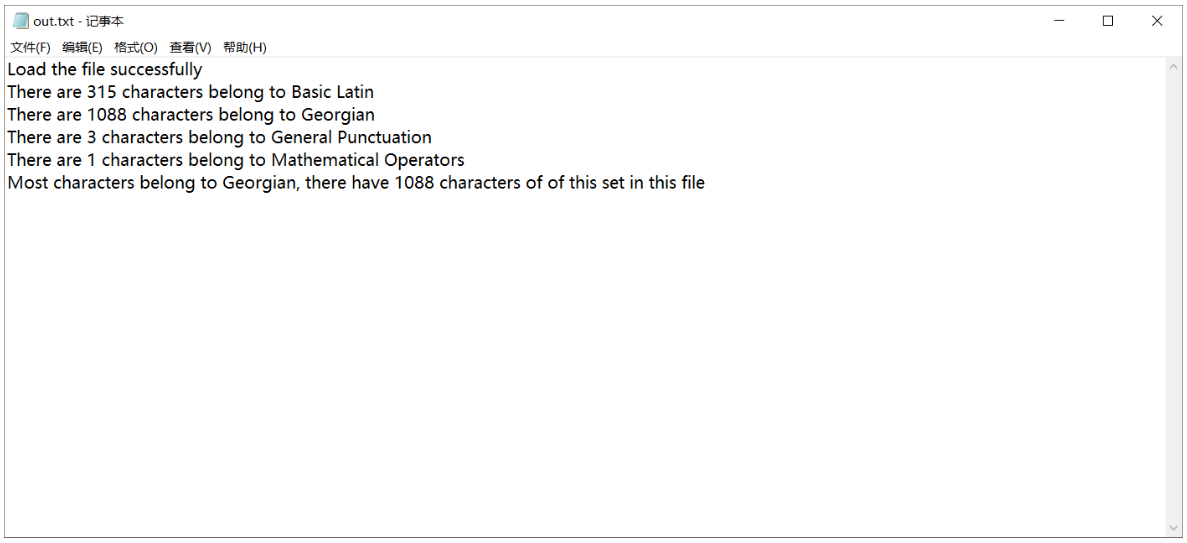
```
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample5.txt>out.txt
Output: Load the file successfully
        There are 714 characters belong to Basic Latin
        There are 3960 characters belong to Devanagari
        There are 2 characters belong to General Punctuation
        Most characters belong to Devanagari, there have 3960 characters of of
this set         in this file
```



```
Load the file successfully
There are 714 characters belong to Basic Latin
There are 3960 characters belong to Devanagari
There are 2 characters belong to General Punctuation
Most characters belong to Devanagari, there have 3960 characters of of this set in this file
```

Test case #7

```
Input: g++ -o display display.cpp utf8.cpp
       ./display <sample6.txt>out.txt
Output: Load the file successfully
        There are 315 characters belong to Basic Latin
        There are 1088 characters belong to Georgian
        There are 3 characters belong to General Punctuation
        There are 1 characters belong to Mathematical Operators
        Most characters belong to Georgian, there have 1088 characters of of
this set in         this file
```



```
Load the file successfully
There are 315 characters belong to Basic Latin
There are 1088 characters belong to Georgian
There are 3 characters belong to General Punctuation
There are 1 characters belong to Mathematical Operators
Most characters belong to Georgian, there have 1088 characters of of this set in this file
```

Part 4-Difficulties & Solutions

1.Since we need to read the information from the file, we need to use ifstream. In this case, getline() is used.
2.Since the "Block.txt" have some useless information,we need to judge whether the information is useful. If the information is useful, then we need to load it into our structure. Else we should ignore it and continue reading the file.

3.Since we need to find the relating data in the file, we need to use structure to help us link the information.

4.We can use given program to get the codepoint of the character. And then we can use find() to judge which character set it belong to.

5.When we judge which set teh character belong to, we traverse the structure, and find which range the codepoint belong to. If we find, we let the count++.

6.When we output the data, we should traverse the structure, and if the count of the set is not 0, we display the character set have how many character in this file.

7.Finally, we should output the name of the blocks which most characters belong to.