CS205 C/ C++ Programming - Lab Assignment 3

Name: Jiachen Zhang

SID: 11713020

Part 1 - Analysis

> This Lab Assignment focuses on the storing as well as the searching of the struct array.
> We need to process storing, searching and reading a file from the standard input.

> I finish this Lab Assignment with the following small steps.
>
> - First, we should load the `Blocks.txt` file into the array, whose element type is a struct, which contains the `start code`, the `end code` and the `block name` of each block.
>   For this step, we should skip blank lines and comments (, which begin with `#`).
> - Second, we need to write a function to search this array when provided with a Unicode value, and a small test program.
> - Third,we need to read a file from the standard input.
> - Finally, we need to display on the standard output the name of the block to which most characters belong (there may be characters from different blocks).

Part 2 - Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "utf8.c"

#define F_PATH "Blocks.txt"
#define MAX_ARRAY_SIZE 300
#define MAX_LINE_LENGTH 1000
#define MAX_BLOCK_NAME_SIZE 100

typedef struct blocks
{
    int Start_Code;
    int End_Code;
    char Block_Name[MAX_BLOCK_NAME_SIZE + 1];
} Blocks, *pBlocks;
int test(void);
void data_Initial(void);
int readBlockFile(void);
int read_state;
void printBlock(pBlocks);
pBlocks data[MAX_ARRAY_SIZE];
int search_Block(int);
int process(void);

int main(int argc, char const *argv[])
{
    /* This function must be processed as it's used to load the data of blocks */
    data_Initial();
    /* It will be better if you only run one of the next two functions at one time */
    // test(); /* This function is used to test finding block name with a unicode */
    process(); /* This function is used to display on the standard output */
```

```
                /* the name of the block to which most characters belong. */
    return 0;
}

int process()
{
    printf("------------------------------------------------------------\n");
    printf("Process a file from the standard input:\n");
    int counters[read_state];
    for (int i = 0; i < read_state; i++) /*initialize the counters*/
        counters[i] = 0;
    unsigned char *pt;
    int lenptr = 0;
    int codept = 0;
    char line[MAX_LINE_LENGTH];
    while (fgets(line, MAX_LINE_LENGTH, stdin))
    { /* Read the file from the standard input line by line. */
        pt = (unsigned char *)line;
        while (*pt != '\0' && *pt != '\n' && *pt != '\r')
        { /* Read the file from the standard input character by character. */
            /* Break when the pointer got to the end of the line */
            codept = utf8_to_codepoint(pt, &lenptr);
            /* Move the pointer depending on the lenptr */
            pt += lenptr;
            /* Record the times of blocks appeared which the charaters belongs to */
            counters[search_Block(codept)]++;
        }
    }
    /* Read the file from the standard input character by character. */
    int max = 0;
    for (size_t i = 0; i < read_state; i++)
        /* Then find the index of the block with the most characters in. */
        if (counters[i] > max)
            /* Record the times of each block appearing */
            max = counters[i];

    /* Print the block which the most characters appear in */
    printf("> The block which most of the characters appear in:\n");
    for (size_t i = 0; i < read_state; i++)
        if (counters[i] == max)
            printf("> -> %s\n", data[i]->Block_Name);
    printf("------------------------------------------------------------\n");
}

/* return i(>= 0): get the index of which block this value belongs to
 * return -1 : not found
 */
int search_Block(int value)
{
    for (size_t i = 0; i < read_state; i++)
      if (data[i]->Start_Code <= value && value <= data[i]->End_Code)
            return i;
    return -1;
}

/* return -1 : file is opened but no accessable data.
 * return 0 : file is not opened succeedly.
 * return i (> 0): file is opened and the number of accessable data is i.
 */
int readBlockFile(void)
{
    FILE *fp = NULL;
    char line[1000];
    /* Load the Blocks.txt and program will exit with announcement when fail in loading. */
    int i = 0;
    if ((fp = fopen(F_PATH, "r")) != NULL)
    {
        printf("------------------------------------------------------------\n");
        printf("Initializing...\n");
        printf("> File is opened\n");
        /* Read and store the date in the file line by line */
        while (fgets(line, MAX_LINE_LENGTH, fp) && i < MAX_ARRAY_SIZE)
        {
            // skip all the blank lines and comments
```

```
            if (line[0] == '\n' || line[0] == '\r' || line[0] == '#')
                continue;
            pBlocks pBlock = (pBlocks)malloc(sizeof(Blocks));
            /* convert the 16-base number to 10-base number */
            pBlock->Start_Code = strtol(strtok(line, "."), NULL, 16);
            pBlock->End_Code = strtol(strtok(NULL, ";") + 1, NULL, 16);
            strncpy(pBlock->Block_Name, strtok(NULL, "\n") + 1, MAX_BLOCK_NAME_SIZE);
            data[i] = pBlock;
            i++;
        }
        fclose(fp);
        fp = NULL;
        return i;
    }
    else
    {
        return 0;
    }
}

void data_Initial(void)
{
    // store the number of the data we readFile may less than what we want
    read_state = readBlockFile();
    switch (read_state)
    {
    case -1:
        printf("> File is opened but there is no accessable data.\n");
        system("pause");
        exit(0);
    case 0:
        printf("> File is not found!\n");
        system("pause");
        exit(0);
    default:
        printf("> Finished reading the file\n");
    }
    printf("> We get %d data\n", read_state);
    printf("------------------------------------------------------------\n\n");
    return;
}

void printBlock(pBlocks b)
{
    printf("[");
    printf(" %d,", b->Start_Code);
    printf(" %d,", b->End_Code);
    printf(" %s", b->Block_Name);
    printf(" ]\n");
    return;
}

int test()
{
    int index = 0;
    int UnicodeValue = 0;
    printf("------------------------------------------------------------\n");
    printf("Small test function:\n");
    printf("> Print the unicode value:\n> ");
    fflush(stdin);
    scanf("%d", &UnicodeValue);
    index = search_Block(UnicodeValue);
    if (index < 0)
        printf("invalid\n");
    else
    {
        printf("> It belongs to the block named:\n");
        printf("> > %s\n", data[index]->Block_Name);
    }

    fflush(stdin);
    printf("------------------------------------------------------------\n\n");
    return 0;
}
```

Part 3 - Result & Verification

Test case #1:

```
If the file name is modified or file is disappeared.
Output:
File is not found!
```

Test case #2:

```
If we comment out the process function before and use test function.
Input:
New
Output:
------------------------------------------------------------
Initializing...
> File is opened
> Finished reading the file
> We get 262 data
------------------------------------------------------------

------------------------------------------------------------
Small test function:
> Print the unicode value:
> new
> It belongs to the block named:
> > Basic Latin
------------------------------------------------------------
```

Test case #3:

```
If we comment out the test function before and use process function.
The content of the document named test.txt is [感谢龚玥学姐]
Input:
./Lab3<test.txt
Output:
------------------------------------------------------------
Initializing...
> File is opened
> Finished reading the file
> We get 262 data
------------------------------------------------------------

------------------------------------------------------------
Process a file from the standard input:
> The block which most of the characters appear in:
> -> CJK Unified Ideographs
------------------------------------------------------------
```

Test case #4:

```
If we comment out the test function before and use process function.
The content of the document named test.txt is [嘤嘤嘤555]
Input:
./Lab3<test.txt
Output:
------------------------------------------------------------
Initializing...
> File is opened
```

```
> Finished reading the file
> We get 262 data
------------------------------------------------------------

------------------------------------------------------------
Process a file from the standard input:
> The block which most of the characters appear in:
> -> Basic Latin
> -> CJK Unified Ideographs
------------------------------------------------------------
```

Part 4 - Difficulties & Solutions

> There is a blank space betw een the seperator ";" and the block name.
> So I use *strncpy(pBlock->Block_Name, strtok(NULL, "\n")* ***+ 1****, MAX_BLOCK_NAME_SIZE)* to skip the blank space.

> There may be characters from different blocks w ith same frequency.
> So after store the frequency, I find the max frequency at first by loop and display the block name w hich has the max frequency.

> This code need to use utf8.c.
> So I w rite a makefile document for easy-compiling.

> I also make some progress to make the program more beautiful.
> Welcome to use it.