CS205 C/ C++ Programming - Lab Assignment 3

Name: 郑瑜果(Zheng Yuguo)

SID: 11712622

Problem 1

Part 1 Analysis

Each line of the file consists of three parts: the starting value, the ending value and the name of the block of these characters. Therefore we need a structure containing three variables to load such data.
When we are reading the file, pay attention that we should ignore those lines that start with '#', which are comments.

Part 2 Code & Result

```c
#define MAX_ARR_SIZE 300
#define MAX_LINE_LEN 100

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

typedef struct {
    int first_code;
    int last_code;
    char type[MAX_LINE_LEN];
} Unicode;

// return the number of successfully read items
int read_file(char *file_name, Unicode unicodes[]);

void search_block(int value, char *block, Unicode *unicodes, int size);

int main(int argc, char const *argv[]) {
    Unicode unicodes[MAX_ARR_SIZE];
    int num = read_file("Blocks.txt", unicodes);

    Unicode* p_unicode = unicodes;
    char block[MAX_LINE_LEN];
    // an example: which block does 47897 belong to?
    search_block(47897, block, p_unicode, num);
    printf("%s\n", block);
    return 0;
}

int read_file(char *filename, Unicode unicodes[]) {
    FILE *fp;
    fp = fopen(filename, "r");

    if (fp == NULL) {
        printf("Open file error!");
        exit(1);
    }

    char buffer[MAX_LINE_LEN];
    int counter = 0;

    while (fgets(buffer, MAX_LINE_LEN, fp) != NULL) {
        if (buffer[0] != '#' && buffer[0] != '\n' && buffer[0] != '\r') {
            char *range;   // the first part of a line
            char *content; // the second part of a line

            range = strtok(buffer, ";");
            content = strtok(NULL, ";");
            char first[10];
            char second[10];
            // split range
            int r_counter = 0;
            while (1) {
                if (range[r_counter] == '.') {
```

```
                if (range[r_counter] == '.') {
                    strncpy(first, range, r_counter);
                    range += r_counter + 2;
                    r_counter = 0;
                } else if (range[r_counter] == '\0') {
                    strncpy(second, range, r_counter);
                    break;
                } else {
                    r_counter++;
                }
            }
            unicodes[counter].first_code = (int)strtol(first, NULL, 16);
            unicodes[counter].last_code = (int)strtol(second, NULL, 16);
            strncpy(unicodes[counter].type, content + 1, MAX_LINE_LEN);
            counter++;
        }
    }
    return counter;
}

void search_block(int value, char *block, Unicode *unicode, int size) {
    for (size_t i = 0; i < size; i++) {
        if (unicode->first_code <= value && value <= unicode->last_code) {
            strncpy(block, unicode->type, sizeof(unicode->type));
        }else{
            unicode++;
        }
    }
}
```

Output:

```
Hangul Syllables
```

Problem 2

Part 1 Analysis

Read the file from the standard input character by character. Use the program in the last problem to search each character belonging to which block. Define a counter to record how many characters are there in each block. Then output the block with the most characters in it.

Part 2 Code

```c
#define MAX_ARR_SIZE 300
#define MAX_LINE_LEN 100

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

typedef struct {
    int first_code;
    int last_code;
    char type[MAX_LINE_LEN];
} Unicode;

// return the number of successfully read items
int read_file(char *file_name, Unicode unicodes[]);

// return block number
int search_block(int value, Unicode *unicodes, int size);

int main(int argc, char const *argv[]) {

    Unicode unicodes[MAX_ARR_SIZE];
    int num = read_file("Blocks.txt", unicodes);
    Unicode *p_unicode = unicodes;
    char block[MAX_LINE_LEN];
    int counters[num];
    memset(counters, 0, sizeof(counters));

    char file_name[MAX_LINE_LEN];
    int value;
    while ((value = fgetc(stdin)) != EOF) {
        int block = search_block(value, p_unicode, num);
        counters[block]++;
    }
    int max = 0;
    int max_idx = 0;
    for (size_t i = 0; i < num; i++) {
        if (counters[i] > max) {
            max = counters[i];
            max_idx = i;
        }
    }
    //printf("%d\n", max);
    printf("%s\n", unicodes[max_idx].type);

    return 0;
}

int read_file(char *filename, Unicode unicodes[]) {
    FILE *fp;
    fp = fopen(filename, "r");

    if (fp == NULL) {
        printf("Open file error!");
        exit(1);
```

```
        exit(1);
    }

    char buffer[MAX_LINE_LEN];
    int counter = 0;

    while (fgets(buffer, MAX_LINE_LEN, fp) != NULL) {
        if (buffer[0] != '#' && buffer[0] != '\n' && buffer[0] != '\r') {
            char *range;   // the first part of a line
            char *content; // the second part of a line

            range = strtok(buffer, ";");
            content = strtok(NULL, ";");
            char first[10];
            char second[10];
            // split range
            int r_counter = 0;
            while (1) {
                if (range[r_counter] == '.') {
                    strncpy(first, range, r_counter);
                    range += r_counter + 2;
                    r_counter = 0;
                } else if (range[r_counter] == '\0') {
                    strncpy(second, range, r_counter);
                    break;
                } else {
                    r_counter++;
                }
            }
            unicodes[counter].first_code = (int)strtol(first, NULL, 16);
            unicodes[counter].last_code = (int)strtol(second, NULL, 16);
            strncpy(unicodes[counter].type, content + 1, MAX_LINE_LEN);
            counter++;
        }
    }
    return counter;
}

int search_block(int value, Unicode *unicode, int size) {
    for (size_t i = 0; i < size; i++) {
        if (unicode->first_code <= value && value <= unicode->last_code) {
            return i;
        } else {
            unicode++;
        }
    }
}
```

Part 3 Result & Verification

Running Commmand:

```
$ ./lab3_2 < Blocks.txt
```

where `lab3_2` is the name of the program and `Blocks.txt` is the file to be analyzed.

Output:

```
Basic Latin
```

Part 4 Difficulties & Solutions

The reading program strictly relies on the format of the file "Blocks.txt". The reading process may fail if the format of "Blocks.txt" is changed even a little bit.