CS307 - Operating Systems — 2021-2022 Fall

Programming Assignment 5 Report

Doğukan Yıldırım — 28364

File Systems

(a) The Code of My Implementation (corrector.c)

```c
#include <stdio.h>
#include <string.h>
#include <dirent.h>

typedef struct Person {
    char name[30], surname[30];
    char gender;
} Person;

const char* SSCANF_FORMAT = "%c %s %s\n";

int modifyPerson(Person p, FILE* in) {
    char word[100];

    fscanf(in, "%s", word);
    fseek(in, -strlen(word), SEEK_CUR);

    if ((p.gender == 'f') && (strcmp(word, "Ms.") != 0)) {
        fputs("Ms.", in);
    }

    else if ((p.gender == 'm') && (strcmp(word, "Mr.") != 0)) {
        fputs("Mr.", in);
    }

    else {
        fseek(in, 3, SEEK_CUR);
    }

    fscanf(in, "%s", word);
    fscanf(in, "%s", word);
    //printf("%s => %ld\n", word, ftell(in) - strlen(word));
    fseek(in, -strlen(word), SEEK_CUR);

    if (strcmp(word, p.surname) != 0) {
        fputs(p.surname, in);
    }

    else {
        fseek(in, strlen(p.surname), SEEK_CUR);
    }

    return 0;
}

int readDatabaseAndModifyPerson(char* word, FILE* db, FILE* in) {
    char line[100];
    while (fgets(line, sizeof(line), db)) {
        Person p;
        int elements_read = sscanf(line, SSCANF_FORMAT, &p.gender, &p.name,
            &p.surname);

        if (elements_read < 3) {
            printf("Less than 3 elements in a database row. ");
            printf("Please make sure database.txt is formatted correctly.\n");
            return 1;
        }

        if (strcmp(p.name, word) == 0) {
            return modifyPerson(p, in);
        }
    }

    return 2;
}

int readAndModifySingleTextFile(char* filename, FILE* db) {
    FILE* in = fopen(filename, "r+");
    if (in == NULL) {
        printf("%s could not be opened.", filename);
        return 1;
    }

    char word[100];
    int foundPerson = 0;
    while (fscanf(in, "%s", word) != EOF) {
        //printf("%s => %ld\n", word, ftell(in) - strlen(word));

        if ((strcmp(word, "Ms.") == 0) || (strcmp(word, "Mr.") == 0)) {
            foundPerson = 1;
        }

        else if (foundPerson == 1){
            foundPerson = 0;
            fseek(in, - 4 - strlen(word), SEEK_CUR);
            readDatabaseAndModifyPerson(word, db, in);
            fseek(db, 0, SEEK_SET);
        }
    }

    fclose(in);
    return 0;
}

void readAndModifyAllTextFiles(const char* dirname, FILE* db) {
    DIR* dir = opendir(dirname);
    if (dir == NULL) {
        return;
    }

    struct dirent* direntPtr;
    direntPtr = readdir(dir);

    while (direntPtr != NULL) {
        if (strcmp(direntPtr->d_name, ".") != 0 &&
            strcmp(direntPtr->d_name, "..") != 0) {
            char path[200] = {0};
            strcat(path, dirname);
            strcat(path , "/");
            strcat(path, direntPtr->d_name);
            //printf("%s\n", path);

            if (direntPtr->d_type == DT_DIR) {
                readAndModifyAllTextFiles(path, db);
            }

            else if (direntPtr->d_type == DT_REG) {
                char* ext = strrchr(direntPtr->d_name, '.');
                if (ext != NULL) {
                    if (strcmp(ext, ".txt") == 0) {
                        if (strcmp(dirname, ".") != 0) {
                            readAndModifySingleTextFile(path, db);
                        }

                        else {
                            if (strcmp(direntPtr->d_name, "database.txt") != 0) {
                                readAndModifySingleTextFile(path, db);
                            }
                        }
                    }
                }
            }
        }

        direntPtr = readdir(dir);
    }

    closedir(dir);
}

int main(int argc, char* argv[]) {
    FILE* db = fopen("database.txt", "r");

    if (db == NULL) {
        printf("database.txt could not be opened.\n");
        return 1;
    }

    readAndModifyAllTextFiles(".", db);

    fclose(db);
    return 0;
}
```

(b) The Explanation of My Implementation

- In the **main** function, the file **database.txt** located under the root directory is opened and if it could be opened, then the function **readAndModifyAllTextFiles** is called with the parameters **"."** and **db**. **"."** is passed to the function to indicate the root directory, and the file pointer **db** is passed to the function to read from the database in further steps.

- As the name suggests, the function **readAndModifyAllTextFiles** is a function that iterates through the root directory and all of the sub-directories recursively, and corrects all the .txt files that contain conflicting titles and last names with respect to entries in the **database.txt**. It takes two parameters, path to a directory (**dirname**), and a FILE pointer to the **database.txt** (**db**).

- To explain in a few short sentences: The function **readAndModifyAllTextFiles** iterates through the all the folders, files (and etc.) in the current directory via a dirent pointer (let's call it **direntPtr**). If **direntPtr** points to a directory, the function **readAndModifyAllTextFiles** is called again with the path of the sub directory. If **direntPtr** points to a regular file, the extension of the said file is checked and if the extension is .txt, the function **readAndModifySingleTextFile** is called with the path to the .txt file and a FILE pointer to the **database.txt**. There are also the necessary if-else checks to make sure that the **database.txt** file under the root directory is not modified.

- With the given path to a .txt file and a FILE pointer **db**, the function **readAndModifySingleTextFile** opens the text file at the said path, and reads it word by word. If the word is **"Ms."** or **"Mr."**, which indicates that a person is being mentioned, than the boolean variable **foundPerson** is set to be true. By making use of the **foundPerson** variable, in the next iteration of the reading word by word loop, the program will capture name of the person, move the FILE* of the .txt file backwards (specifically to the location of the **"Mr."** or **"Ms."**) and call the **readDatabaseAndModifyPerson** with the parameters: the name of the person, a FILE pointer to the **database.txt**, a FILE pointer to the .txt file which will be corrected. After correcting the mistakes (if there is any), the pointer **db** is moved to the beginning of the **database.txt**.

- The function **readDatabaseAndModifyPerson** reads the **database.txt** line by line and puts the information in each row (line) in a Person struct (which has the member variables: gender, name and surname). If a person's name in the database is equal to the name given as a parameter to this function (i.e. there exists a person with the given name in the database), then the function **modifyPerson** is called with parameters: Person struct of the person with the given name, and a FILE pointer to the .txt file.

- With the given Person struct (**p**), and a FILE pointer to the .txt file (**in**), the function **modifyPerson** checks whether the title and the surname of the person being mentioned in the .txt file is incorrect with respect to the Person struct obtained from the database. Firstly, it gets the **"Mr."** or **"Ms."** word and moves the FILE pointer **in** backwards to the start of the title using **fseek**. If there are any mistakes with the title with respect to the gender of the person, the function corrects the title in the .txt file using **fputs**, otherwise it moves the FILE pointer **in** forwards 3 letters using **fseek**. Afterwards, the function moves two words forwards to the surname, captures the surname in the .txt file, and moves the FILE pointer **in** backwards to the start of the surname using **fseek**. If the surname is not the same with the surname mentioned in the database, the function corrects the surname in the .txt file using **fputs**, otherwise it moves the FILE pointer **in** forwards by the number of letters in the person's surname using **fseek**.

- All the functions also make sure that the files are closed when no longer needed. The **database.txt** file under the root directory is closed after every other function other than the **main** function is done executing, then the **main** function terminates.