Islamic University of Technology (IUT)

Report on Lab 01

Submitted By

Shanta Maria (ID:200042172)

CSE 4308 Database Management Systems Lab

Submitted To

Md. Bakhtiar Hasan

Lecturer, Department of CSE

August 14, 2022

Introduction

In the lab class, we were given a problem statement with three tasks to solve. We were allowed to use any programming language to complete the tasks. I have used C programming language to solve all three of the tasks.

Problem Description

Before the advent of databases, information was recorded using the traditional file system. In this lab, we want to experience how life was back then. Consider that your database consists of two files, namely '`studentInfo.txt`' and '`grades.txt`'. Each row of the `studentInfo.txt` file contains:

- Student ID
- Name
- Age
- Blood Group
- Department

And each row of the `grades.txt` file contains:

- Student ID
- GPA
- Semester

Assume that the values of *Student ID* and *Age* fit within the range of a 32-bit integer. The values under *Name, Blood Group,* and *Department* have at most 10 characters. The *GPAs* are given as floating point values within [2.50, 4.00]. It is guaranteed that the *Student IDs* are unique for each student. The values for each *field* is separated by semicolon (;).

1 Task 1

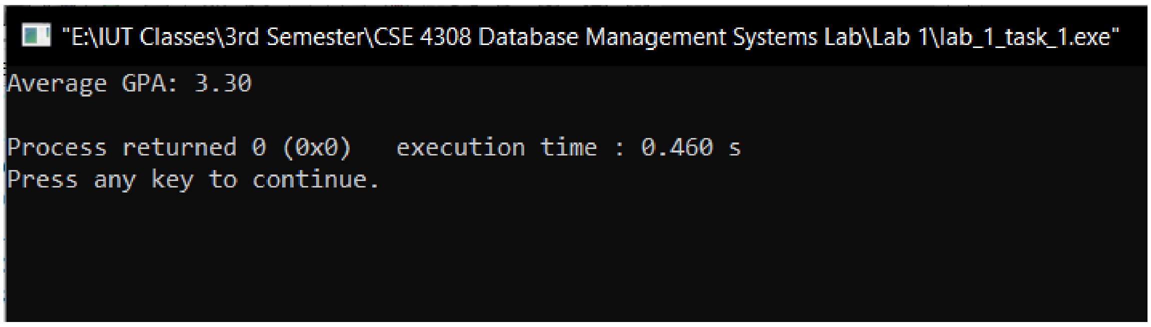
Print the average *GPA* of the students.

1.1 Solution

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

int main(void)
{
    //Opening file
    FILE *grades=fopen("grades.txt", "r");

    //Checking if file opened correctly
    if(grades==NULL)
    {
        printf("Could not open file!");
        exit(0);
    }

    //Variables
    char c;
    int column=0, count=0;
    double total=0.0, cg=0.0;

    //Scanning char one by one
    while((c=getc(grades))!=EOF)
    {
        //Starts scanning grade after first column
        if(column==1 && c!=';')
        {
            //Since after ; char, the first number of the GPA was
            //already scanned by fgetc, so we use ungetc then scan GPA
            ungetc(c, grades);
            fscanf(grades, "%lf", &cg);

            //adding to total and counting number of entries
            total+=cg;
            count++;
        }
        //Counting column using ; char occurence
        if(c==';')
        {
            column++;
        }
        //Initialing column counter after each row
        if(c=='\n')
        {
            column=0;
        }
    }

    //Printing output
    printf("Average GPA: %.2lf\n", total/count);

    return 0;
}
```

1.2 Analysis and Explanation

In the solution above, first the `grades.txt` file is opened in reading mode. Before starting the actual solution, it is checked if the file has opened correctly. The program would be terminated if the file could not be opened properly. Using a `while()` loop, a single character is scanned and stored in the variable `c` in every iteration. If `c` is ';', then a counter for the number of columns traversed is incremented. If `c` is a new line character, then the column counter variable is initiated to 0. The number of columns is counted because for the task, we require all the *GPA* of the students and no other information. The *GPA* is stored in the second column of the file for every entry. So when the column number is one that is we have already traversed through the contents of the first column and `c` is not ';', we use `ungetc()` to move back one space and then scan and store the entire *GPA* as a double variable in `cg`. The *GPA* is then added to a double variable, `total`, which stores the summation of all the *GPA*. A counter for the number of *GPA* recorded, `count`, is also incremented each time a *GPA* is added to the total. After traversing through the entire file that is when the End of File (`EOF`) character is read, the `while()` loop is exited. The average *GPA* is calculated by dividing `total` by `count`. The output is printed in the console with a precision of 2 decimal places as shown below:



```
"E:\IUT Classes\3rd Semester\CSE 4308 Database Management Systems Lab\Lab 1\lab_1_task_1.exe"
Average GPA: 3.30

Process returned 0 (0x0)   execution time : 0.460 s
Press any key to continue.
```

Figure 1: Output for Task 1

1.3 Difficulties

I faced difficulties in figuring out the following parts of my solution:

1. How to get the input from the second column only
2. Thinking of initiating the column counter variable after the end of each row

2 Task 2

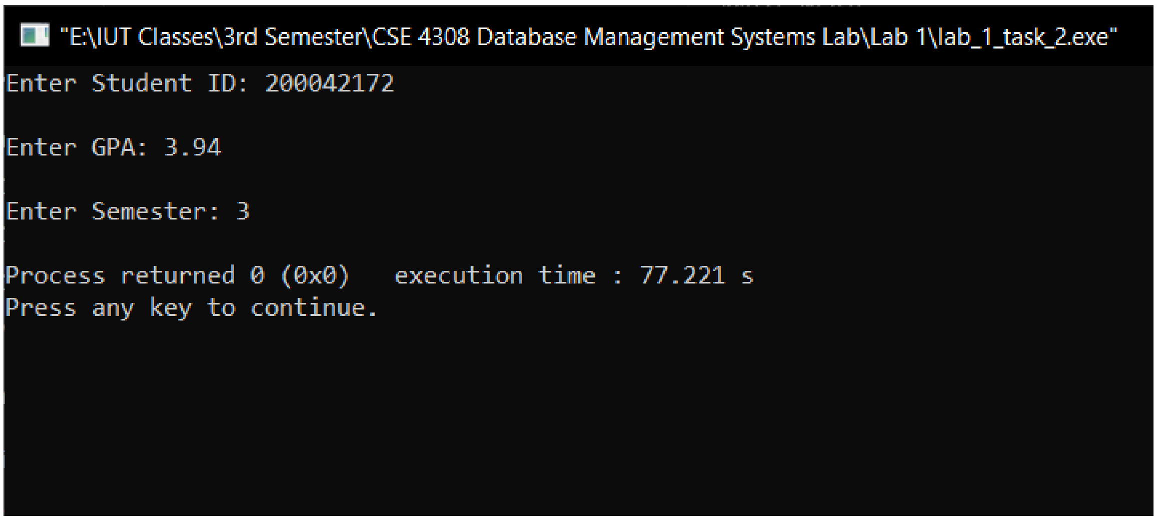
Take *Student ID, GPA,* and *Semester* as input. Then after validating the input, insert the information as a new row in the `grades.txt` file. If the information is invalid, discard the input and show an error message.

2.1 Solution

```c
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
    //Opening files
    FILE *grades=fopen("grades.txt", "a");

    //Checking if files opened correctly
    if(grades==NULL)
    {
        printf("Could not open file!");
        exit(0);
    }

    //Variables
    int id=0, semester=0;
    float gpa=0.0;

    //Taking input for new entry
    printf("Enter Student ID: ");
    scanf("%d", &id);
    printf("\nEnter GPA: ");
    scanf("%f", &gpa);
    printf("\nEnter Semester: ");
    scanf("%d", &semester);

    //Checking if inputs are valid
    if(id>0 && id<=2e9 && gpa>=2.50 && gpa<=4.00 && semester>=1 && semester<=8)
    {
        ;
    }
    else
    {
        printf("\nInvalid Input!\n");
        exit(0);
    }

    //Scanning new line char so that entry prints from next line
    fgetc(grades);

    //Printing new entry
    fprintf(grades, "\n%d;%.2f;%d", id, gpa, semester);

    return 0;
}
```

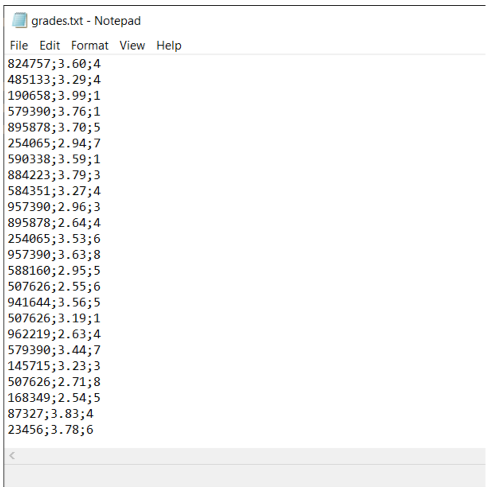
2.2 Analysis and Explanation

In the solution above, first the `grades.txt` file is opened in append mode so that information can be added at the end of the file. Before starting the actual solution, it is checked if the file has opened correctly or not. The program would be terminated if the file could not be opened properly. Inputs for the new entry which include *Student ID, GPA* and *Semester* are taken from the console. The inputs are then checked for validity using the constraints mentioned in the problem statement. If any of the inputs are not valid then an error message is shown in the console and the program is terminated. Otherwise, the new entry is printed in the `grades.txt` file using `fprintf()` with the column separators, ';'. Before printing the entry every time, a single character for new line is scanned using `fgetc()` so that each new entry is printed on a new line. The figures below show the `grades.txt` file before and after the new entry as well as the entry being inputted in the console. The error message for an invalid entry can also be seen.



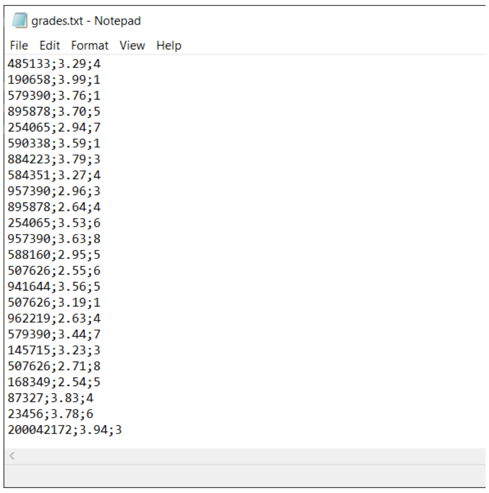
```
"E:\IUT Classes\3rd Semester\CSE 4308 Database Management Systems Lab\Lab 1\lab_1_task_2.exe"
Enter Student ID: 200042172

Enter GPA: 3.94

Enter Semester: 3

Process returned 0 (0x0)   execution time : 77.221 s
Press any key to continue.
```

Figure 2: Input for New Entry for Task 2



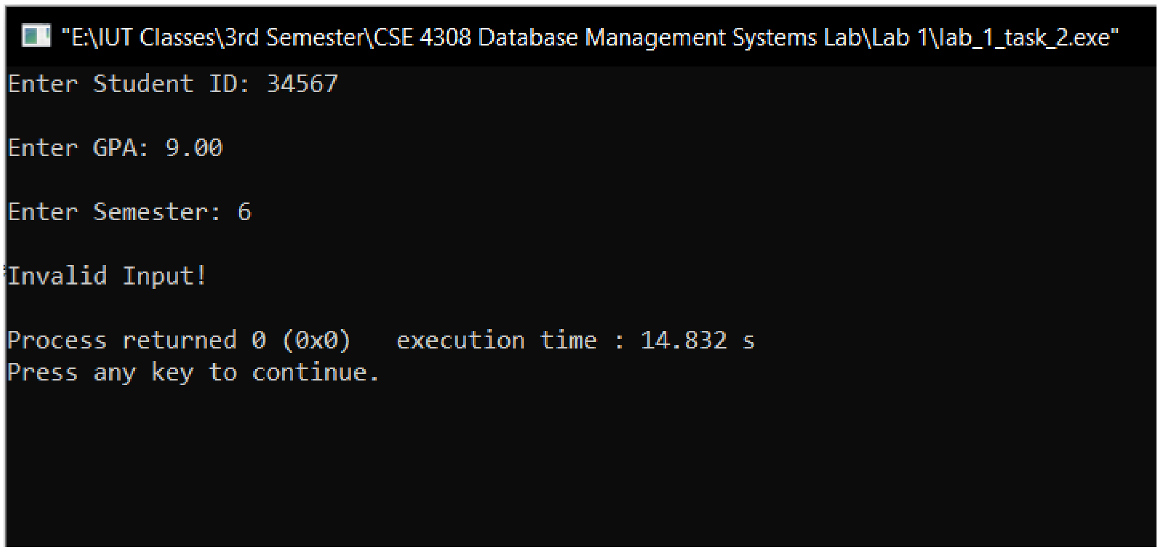
```
grades.txt - Notepad
File  Edit  Format  View  Help
824757;3.60;4
485133;3.29;4
190658;3.99;1
579390;3.76;1
895878;3.70;5
254065;2.94;7
590338;3.59;1
884223;3.79;3
584351;3.27;4
957390;2.96;3
895878;2.64;4
254065;3.53;6
957390;3.63;8
588160;2.95;5
507626;2.55;6
941644;3.56;5
507626;3.19;1
962219;2.63;4
579390;3.44;7
145715;3.23;3
507626;2.71;8
168349;2.54;5
87327;3.83;4
23456;3.78;6
```

Figure 3: File Before Adding New Entry in Task 2



```
grades.txt - Notepad
File  Edit  Format  View  Help
485133;3.29;4
190658;3.99;1
579390;3.76;1
895878;3.70;5
254065;2.94;7
590338;3.59;1
884223;3.79;3
584351;3.27;4
957390;2.96;3
895878;2.64;4
254065;3.53;6
957390;3.63;8
588160;2.95;5
507626;2.55;6
941644;3.56;5
507626;3.19;1
962219;2.63;4
579390;3.44;7
145715;3.23;3
507626;2.71;8
168349;2.54;5
87327;3.83;4
23456;3.78;6
200042172;3.94;3
```

Figure 4: File After Adding New Entry in Task 2



"E:\IUT Classes\3rd Semester\CSE 4308 Database Management Systems Lab\Lab 1\lab_1_task_2.exe"

```
Enter Student ID: 34567

Enter GPA: 9.00

Enter Semester: 6

Invalid Input!

Process returned 0 (0x0)   execution time : 14.832 s
Press any key to continue.
```

Figure 5: Error Message for Invalid Input in Task 2

2.3 Difficulties

I faced difficulties in figuring out the following parts of my solution:

1. Printing new entry to a new line each time by using `fgetc()`
2. Thinking of the constraints for *Student ID*

3 Task 3

Take *Student ID* as input and show his/her name and semester in which he/she got the lowest *GPA*. Print an error message if the *Student ID* does not exist in your database.

3.1 Solution

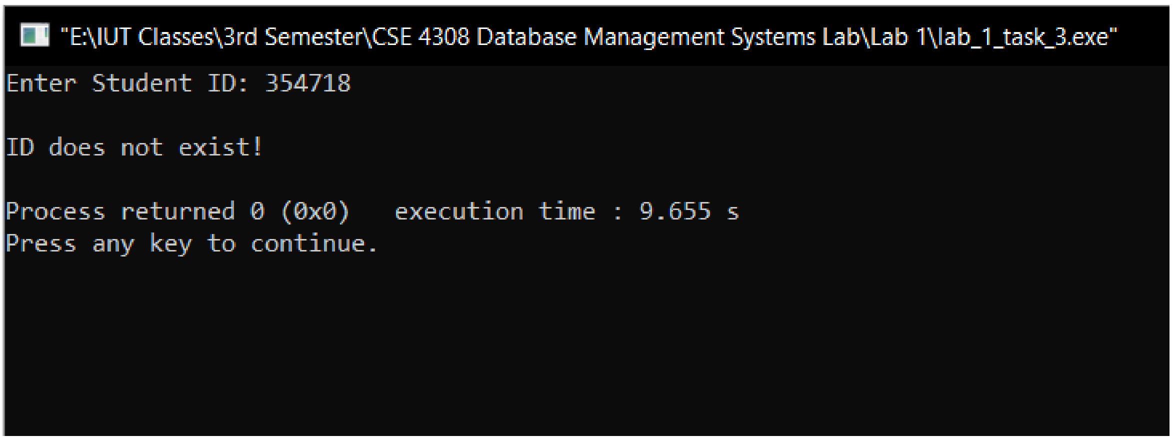
```c
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
    //Opening files
    FILE *grades=fopen("grades.txt", "r");
    FILE *info=fopen("studentInfo.txt", "r");

    //Checking if files opened correctly
    if(grades==NULL || info==NULL)
    {
        printf("Could not open files!");
        exit(0);
    }

    //Variables
    int id=0, check=0, semester=0, min_sem=0;
    char e[100]= {'\0'}, c;
    int is_present=0;
    float gpa=0.0, min=4.50;

    //Taking input
    printf("Enter Student ID: ");
    scanf("%d", &id);

    //Checking if input is valid
    if(id>0 && id<=2e9)
    {
        ;
    }
    else
    {
        printf("\nInvalid Input!\n");
        exit(0);
    }

    while(1)
    {
        //Scanning id, rest of the string and new line char separately
        fscanf(info, "%d", &check);
        //Scans ; char so that it is not scanned into e array
        fgetc(info);
        fscanf(info, "%s", &e);
        c=fgetc(info);

        //Terminates when reaches EOF
        if(c==EOF)
        {
            break;
        }

        //Checking if same ID found
        if(check==id)
        {
            is_present=1;

            //Printing name
            printf("\nName: ");

            for(int i=0; e[i]!=';'; i++)
            {
                printf("%c", e[i]);
            }

            printf("\n");

            break;
        }
    }

    while(1)
    {
        //Scanning ID, GPA and semester separately
        //fgetc scans ; char
        fscanf(grades, "%d", &check);
        fgetc(grades);
        fscanf(grades, "%f", &gpa);
        fgetc(grades);
        fscanf(grades, "%d", &semester);

        //Terminates when reaches EOF
        if(fgetc(grades)==EOF)
        {
            break;
        }

        //Checking if same ID found
        if(check==id)
        {
            is_present=1;

            //Checking and storing minimum GPA and that semester
            if(gpa<=min)
            {
                min=gpa;
                min_sem=semester;
            }
        }
    }

    //Checking if ID was found in the files or not
    if(is_present==0)
    {
        printf("\nID does not exist!\n");
        exit(0);
    }
    else
    {
        //Printing output
        printf("\nSemester with lowest GPA: %d\n", min_sem);
    }

    return 0;
}
```

3.2 Analysis and Explanation

In the solution above, first the `grades.txt` and `studentInfo.txt` files are opened in reading mode. Before starting the actual solution, it is checked if the file has opened correctly or not. The program would be terminated if the file could not be opened properly.

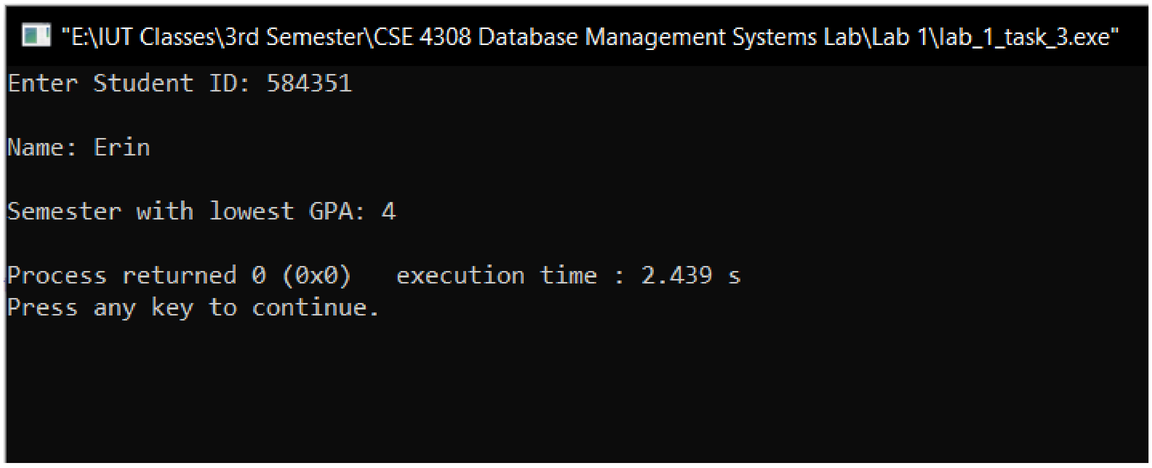
Firstly, the *Student ID* to be searched is taken as input from the console. It is then checked if the *ID* is a valid input or not. A `while()` loop is then used for traversing through the `studentInfo.txt` file. The first column contains the *ID* which is scanned and stored using `fscanf()`. Then the column separator ';' is scanned but not stored using `fgetc()` so that it is not included when rest of the entry is scanned using `fscanf()` as one single string, `e`. This is done because the second column contains the *Name* of the student which is required later when printing the output and it is also easier to scan the rest of the entry as a single string rather than the *Name* of the student only in the second column. `fgetc()` is used again to scan the character at the end of the entry which is stored in the variable `c`. This is used to check if we reached the end of the file or not. If we reach `EOF` then the `while()` loop will be terminated. The input *ID* is then checked with all the *Student ID* in the file. If a match is found, then the *Name* of the student (from the first element until ';' character is found) is printed from the string stored in the variable `e` and we break out of the `while()` loop. A flag variable named `is_present` is used to check if the *ID* was found.

A second `while()` loop is used to traverse through the `grades.txt` file. The *Student ID, GPA*



```
"E:\IUT Classes\3rd Semester\CSE 4308 Database Management Systems Lab\Lab 1\lab_1_task_3.exe"
Enter Student ID: 354718

ID does not exist!

Process returned 0 (0x0)   execution time : 9.655 s
Press any key to continue.
```

Figure 6: Invalid Input for Task 3



```
"E:\IUT Classes\3rd Semester\CSE 4308 Database Management Systems Lab\Lab 1\lab_1_task_3.exe"
Enter Student ID: 584351

Name: Erin

Semester with lowest GPA: 4

Process returned 0 (0x0)   execution time : 2.439 s
Press any key to continue.
```

Figure 7: Output for Correct Input in Task 3

and *Semester* are all scanned separately using `fscanf()`. `fgetc()` is used to scan the separator ';', new line character and the `EOF` character. If `EOF` character is found instead of new line character using the third `fgetc()` function, then we break out of the `while()` loop. We check again in this `while()` loop to see if the input *ID* is found in the file or not. If the *ID* is found, `is_present` is set to 1 and the *GPA* of that student is checked to see if it is the minimum *GPA* or not for that student. Each time it is checked for the minimum value, the *Semester* and current minimum *GPA* is stored in the variables `min` and `min_sem`.

After breaking out of both the `while()` loops, the variable `is_present` is checked to see if it is 1 (*ID* found) or 0 (*ID* not found). If it is 0, an error message is shown and the program is terminated otherwise the *Semester* stored in `min_sem` is printed as shown below:

3.3 Difficulties

I faced difficulties in figuring out the following parts of my solution:

1. Scanning the columns in both the files separately so that *ID* can be checked and the *Name* can be printed
2. Finding the breaking condition for the first `while()` loop
3. Scanning the `EOF` character for the `while()` loops without disrupting any of the other scanned variables
4. Finding the minimum *GPA* of the student
5. Printing the error message when the *ID* was not found

Conclusion

As shown in the report, I have solved and tested the solutions for all three of the tasks given in the lab. C programming language was used along with the C libraries, stdio.h and stdlib.h to solve the problems.