Assignment #: Lab-01A

Mirza Mohammad Azwad

ID: 200042121

Date of Submission: 09/08/202

Write a report for the following task. Click Here

Contents 1

Programming Language Used For Solution: **C**

<u>Introduction</u>

The introduction of this lab report is a brief analysis of the problem stated as per the lab task. The lab task asks us to manage data contained in files with the ‘;’ used as a delimiter and each information contained in each line of a file with a newline at the end of each line. Although there was no language constraint for this task, as per my ease of understanding I chose C as the language to implement the solution. The solution deals with the retrieval of the information into more tangible data structures to manipulate the information and carry out read and write operations to and from files. I used an array of struct as my chosen data structure to carry out this task. Further details about how I used them is mentioned later in the **Detailed Lab Report**. I followed a modular approach to solve this task and divided the problem into functions for better understanding of the code and to maintain a degree of clean coding within my ability. This further helped me to understand the requirements of this problem and implement it as properly as possible.

Detailed Lab Report

The detailed code for the three tasks is given here.

The library files used

The library files used throughout the code are given below:

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

The first task asks us to print the student's name with the highest GPA. This task has three main issues, firstly how do you deal with multiple students who have the highest GPA. In this regard, my solution shows the last student with a tie of the highest GPA if it is present. [1]

I used two array of structs to store the information of the different files. The first array stores the information of the grades file.

The first struct array

```
typedef struct information1{
    char name[NAME_LEN];
    long studentID;
    long age;
    char department[DEPT_LEN];
    char bloodGroup[BG_LEN];
}Info1;
Info1 nameList1[NUMBEROFRECORDSMAX+1];
```

The second struct array

```
typedef struct information2{
    long studentID;
    float GPA;
    int semester;
}Info2;
Info2 nameList2[NUMBEROFRECORDSMAX+1];
```

The first struct array is used especially in task 1 whereas the second struct array is used in tasks 2 and 3.

Macro Definitions

For these struct arrays I utilized certain macro definitions, namely:

```
#define NAME_LEN 10
#define BG_LEN 10
#define DEPT_LEN 10
#define NUMBEROFRECORDSMAX 1000005
```

My aim for the tasks was to follow a modular approach and divide the task into functions that can be reused for the upcoming tasks to deal with the time constraint of completing the task in the lab.

Record Index and Length Determination

I used two global pointers to indicate the number of records passed into the struct array:

```
int g_numberOfRecords=0;
int g_numberOfRecords2=0;
```

For the first task

We open the file with "r" as the mode string to indicate that the file is being opened for reading along with the absolute address using the fopen function which is passed to the **FILE** pointer fp.

```
void firstFileRead(){
    FILE* fp;
    fp=fopen("D:\\Lab-01\\grades.txt","r");
    if(fp==NULL){
        printf("Error in opening input file");
        return;
    }
    else{
```

```
char a;
char temp[15+5+10];
while(fscanf(fp,"%s",temp)!=EOF){
 char studentID[15];
 char GPA[5];
 char semester[10];
 long StudentID;
 int Semester;
 float gpa;
 memset(studentID,'\0',sizeof(studentID));
 memset(GPA,'\0',sizeof(GPA));
 memset(semester,'\0',sizeof(semester));
 int j=0;
 int i=0;
 while(temp[j]!=';'){
     studentID[i]=temp[j];
     i++;
     j++;
 }
 i=0;
 j++;
 while(temp[j]!=';'){
     GPA[i]=temp[j];
     i++;
     j++;
 }
 i=0;
 j++;
 while(temp[j]!='\0'){
     semester[i]=temp[j];
     i++;
     j++;
 }
 sscanf(studentID,"%ld",&StudentID);
 sscanf(GPA,"%f",&gpa);
 sscanf(semester,"%d",&Semester);
```

```
        nameList2[g_numberOfRecords2].studentID=StudentID;
        nameList2[g_numberOfRecords2].GPA=gpa;
        nameList2[g_numberOfRecords2++].semester=Semester;
      }
    }
    fclose(fp);
}
```

In the first task, I initially checked whether the file can be opened if the file pointer **fp** is **NULL** it indicates that the file cannot be opened so it shows an error message and then returns from the function. After which I used **fscanf** to read each line and then store the values for each line in a temp array. Then I filtered the temp array using multiple while loops and temporarily stored the results in character arrays which I memset with '\0' or null character to prevent erroneous character array(string) storage. Initially, the filtering was done by searching for the ';' delimiter but later for the final column, I searched for the null value and hence determined the results for all the columns. Following this, I used sscanf to store the results of the string to the required data types for instance long for studentID, int for semester, and float for GPA. Then I passed the results to the struct array as per the global pointer g_numberOfRecords2 while incrementing it at the end of each line to allow the next record to be stored.

The first task asked us to find the maximum GPA of the students, as per the aforementioned idea in [1]. I implemented the function given below:

```
int FindMax(){
    float maxGPA=-1.00f;
    int maxIx=0;
    for(int i=0;i<g_numberOfRecords2;i++){
        if(nameList2[i].GPA>maxGPA){
            maxGPA=nameList2[i].GPA;
            maxIx=i;
        }
        //printf("%ld %f\n",nameList2[i].studentID,nameList2[i].GPA);
    }
    //printf("%f\n",maxGPA);
    return maxIx;
}
```

This function gives us the index for the student with the maximum GPA or the highest GPA and this index can later be used to retrieve the student ID from the struct array nameList2. Then we can just print the studentID at the maximum index position of nameList2 to obtain the result. And then we close the file pointer **fp**.

```
int main(void){
    firstFileRead();
    int maxIx=FindMax();
    printf(" Student ID Max GPA: %ld\n",nameList2[maxIx].studentID);

    return 0;
}
```

<u>For the second task</u>

We implement another function that essentially reads and stores the information in another struct array in a similar process used in firstFileRead function. The implementation is given below:

```
void secondFileRead(){
    FILE* fp;
    fp=fopen("D:\\Lab-01\\studentinfo.txt","r");
    if(fp==NULL){
        printf("Error in opening input file");
        readError=true;
        return;
    }
    else{
       char a;
       char temp[15+5+10];
       while(fscanf(fp,"%s",temp)!=EOF){
        char studentID[15];
        char name[10];
        char age[15];
        char bloodGroup[10];
        char department[10];
        long StudentID;
        long Age;

        memset(studentID,'\0',sizeof(studentID));
        memset(name,'\0',sizeof(name));
        memset(bloodGroup,'\0',sizeof(bloodGroup));
        memset(age,'\0',sizeof(age));
        memset(department,'\0',sizeof(department));
        int j=0;
        int i=0;
        while(temp[j]!=';'){
            studentID[i]=temp[j];
            i++;
            j++;
        }
        i=0;
        j++;
        while(temp[j]!=';'){
            name[i]=temp[j];
            i++;
            j++;
        }
        i=0;
        j++;
        while(temp[j]!=';'){
            age[i]=temp[j];
            i++;
            j++;
        }
        i=0;
        j++;
        while(temp[j]!=';'){
            bloodGroup[i]=temp[j];
            i++;
            j++;
        }
        i=0;
        j++;
        while(temp[j]!='\0'){
            department[i]=temp[j];
            i++;
            j++;
        }
        sscanf(studentID,"%ld",&nameList1[g_numberOfRecords].studentID);
        sscanf(name,"%s",nameList1[g_numberOfRecords].name);
        sscanf(age,"%ld",&nameList1[g_numberOfRecords].age);
        sscanf(bloodGroup,"%s",nameList1[g_numberOfRecords].bloodGroup);
        sscanf(department,"%s",nameList1[g_numberOfRecords++].department);
       }
    }
    fclose(fp);
}
```

The key difference here is that I used a direct method of just entering it directly into the struct array using **sscanf** instead of separately initializing it as I had done earlier. The other alternative would require us to use memcpy but it would cause us to use more redundant lines of code that could have been avoided. In the second task, I also used firstFileRead() function to initialize the struct array for the grades. txt.

The second task also requires us to write input of student ID, GPA, and semester into the grades.txt text file upon validating it. For that I used the function given below:

```
void writeValidatedGrade(){
    FILE* fp=fopen("D:\\Lab-01\\grades.txt", "a");
    if(fp==NULL){
        printf("Error in opening input file");
        writeError=true;
        return;
    }
    else{
        long studentID;
        float gpa;
        int semester;
        scanf("%ld %f %d",&studentID,&gpa,&semester);
        bool found=false;
        for(int i=0;i<g_numberOfRecords;i++){
            if(studentID==nameList1[i].studentID){
                found=true;
            }
        }
        if(!found){
            printf("Student ID not found");
            return;
        }
        if(gpa<2.50f || gpa>4.00f){
            printf("GPA is out of range");
            return;
        }
        if(semester<1 || semester>8){
            printf("semester out of range");
            return;
        }
        for(int i=0;i<g_numberOfRecords2;i++){
            if(studentID==nameList2[i].studentID && nameList2[i].semester==semester){
                puts("GPA for semester already exists");
                return;
            }
        }
        fprintf(fp,"%ld %f %d\n",studentID,gpa,semester);

    }
    fclose(fp);
}
```

We open the **FILE** pointer with “a” as the mode string to indicate that the file is being opened for appending information.

First I checked whether the student ID exists in the nameList1 struct array using a for loop. If it does then it's a valid student ID. Next, I checked whether the GPA is between 2.50 and 4.00 and whether the semester is between 1 and 8. Otherwise, it's invalid and the function is returned with an appropriate error message before it. Next, I checked whether there was an earlier record of that particular entered student ID using a for loop and a complex condition in the if statement checking if there is a record with the entered student ID and semester. If there is such a record then we output an error message stating that “GPA for a semester already exists” and we return from the function. If it passes all these validation checks then we write the information to the file using fprintf. Then we close the file pointer. Finally my main function for the second task:

```
int main(void){
    firstFileRead();
    secondFileRead();
    writeValidatedGrade();


    return 0;
}
```

For the third task

We had to find the Cumulative GPA for a particular student ID for that my implementation is given below.

```
void calculateCGPA(){
        long studentID;
        scanf("%ld",&studentID);
        bool found=false;
        int ix=0;
        for(int i=0;i<g_numberOfRecords;i++){
            if(nameList1[i].studentID==studentID){
                found=true;
                ix=i;
            }
        }
        if(!found){
            puts("Student ID not in file");
            return;
        }
        printf("%s",nameList1[ix].name);

        float total=0;
        int count=0;
        for(int i=0;i<g_numberOfRecords2;i++){
            if(studentID==nameList2[i].studentID){
                total+=nameList2[i].GPA;
                count++;
            }
        }
        float cgpa=total/(float)count;
        printf(" %.2f",cgpa);
}
```

For the third task, I also used firstFileRead() and secondFileRead() functions to initialize the struct arrays. Following which, I searched for the student ID whose CGPA I want to calculate in the nameList1 struct array which holds information from the studentinfo.txt file using a for loop. If the student ID is present in the struct array then its a valid ID, otherwise I gave an error message and returned the function. Taking the student ID as valid, I would loop through the nameList2 struct and I would add up the GPA for all the semesters by totalling and storing the value in float variable named total. I would increment a count in the loop to find the number of semesters spent by the student. Finally to calculate the CGPA I would divide the total by a typecasted count and obtain the cgpa stored in the float variable cgpa and then I would print the cgpa with a %f format specifier with a .2 precision.

The main function for the third task is given below:

```c
int main(void){
    firstFileRead();
    secondFileRead();
    calculateCGPA();
    return 0;
```

Conclusion

This lab report clearly shows how file management system isn't as efficient or feasible as database management system. This task relied on a great degree of hard coding to carry out simple filtering of data as well as carrying out various data validation tasks. The code written itself has many fundamental flaws which if it were to be addressed would increase the complexity of the code and decrease the optimality. Keeping all of this in mind, this task shows the pain the earlier programmers had to endure before the advent of database management systems.

Some of the flaws that the code I implemented contains is the not displaying for a large number of student IDs containing the same GPA. Another flaw would be the use of sscanf, it worked because the sample text did not contain full names of students with spaces in between but if it did, then sscanf would give erroneous readings. An alternative could be to use memcpy instead to pass the name strings and address strings to the struct arrays but it wasn't necessary here as per the given data. I also used fsanf to read each line which would only be possible if the name does not contain any spaces as per the data provided. Additional problems include the limitation I set on the data upto which the struct can store. Storing further data would require us to allocate memory using malloc or utilize a linked list which would further complicate the task at hand. It would also end up decreasing the optimality for larger data.

The code implemented here was also very big and time consuming to write as opposed to defining a database using DDL or modifying it or performing queries on it using DML. Essentially, the abstraction achieved by the invention of databases helped ease the task greatly for programmers and also saved time and optimised the programs to a great extent.