

CSE344

Homework #1

Muhammet Çağrı Yılmaz

1901042694

22/03/2024



1-) Introduction

You are tasked with creating a student grade management system in C programming language. The system should allow the user to manage student grades stored in a file. The user should be able to perform the following operations: gtuStudentGrades “grades.txt” should create a file. Use fork() system call for process creation.

1.1-) Methods

1. Add Student Grade: The user should be able to add a new student's grade to the system. The program should prompt the user to enter the student's name and grade, and then add this information to the file using a separate process for file manipulation. addStudentGrade “Name Surname” “AA” command should append student and grade to the end of the file.
2. Search for Student Grade: The user should be able to search for a student's grade by entering the student's name. The program should then display the student's name and grade if it exists in the file. searchStudent “Name Surname” command should return student name surname and grade.
3. Sort Student Grades: The user should be able to sort the student grades in the file. The program should provide options to sort by student name or grade, in ascending or descending order. sortAll “grades.txt” command should print all of the entries sorted by their names.
4. Display Student Grades: The user should be able to display all student grades stored in the file. The program should display the student name and grade for each student. Also display content page by page and first 5 entries. showAll “grades.txt” command should print all of the entries in the file. listGrades “grades.txt” command should print first 5 entries. listSome “numofEntries” “pageNumber” “grades.txt” e.g. listSome 5 2 grades.txt command will list entries between 5th and 10th.
5. Logging: Create a log file that records the completion of each task as desired.
6. Usage: The user should be able to display all of the available commands by calling gtuStudentGrades without an argument.

2-) How to compile

We have a makefile so if you just run make comment in the terminal, the program will be compiled. If you want to delete compiled files(I also added test files) you need to run 'make clean' comment.

```
cagriyilmaz04@DESKTOP-J7JJ4NB:/mnt/c/Users/yilma/Desktop/Cagri$ make
gcc -Wall -pedantic-errors -o hw1 gtuStudentGrades.c
cagriyilmaz04@DESKTOP-J7JJ4NB:/mnt/c/Users/yilma/Desktop/Cagri$ ./gtuStudentGrade
GTU Student Grade Management System
> gtuStudentGrades grade.txt
grade.txt is ready for use.
> addStudentGrade "Cagri Yilmaz" "AA"
Grade added for Cagri Yilmaz
> ^C
cagriyilmaz04@DESKTOP-J7JJ4NB:/mnt/c/Users/yilma/Desktop/Cagri$ make clean
rm -f action.log hw1
```

3-) Implementantation

Adding Student

```
void addStudentGrade(const char *filename, const char *name, const char *grade) {
    pid_t pid = fork();

    if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        int fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
        if (fd == -1) {
            perror("Error opening/creating the file");
            exit(EXIT_FAILURE);
        }

        if (dprintf(fd, "%s, %s\n", name, grade) < 0) {
            perror("Error writing to the file");
            close(fd);
            exit(EXIT_FAILURE);
        } else {
            printf("Grade added for %s\n", name);
            close(fd);
            exit(EXIT_SUCCESS);
        }
    } else {
        int status;
        waitpid(pid, &status, 0);
        if (WIFEXITED(status) && WEXITSTATUS(status) == EXIT_SUCCESS) {
            writeLog("Grade added successfully");
        } else {
            fprintf(stderr, "Failed to add grade\n");
        }
    }
}
```

Search Student

```

void searchStudent(const char *filename, const char *searchName) {
    pid_t pid = fork();

    if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        int fd = open(filename, O_RDONLY);
        if (fd == -1) {
            perror("Error opening the file");
            exit(EXIT_FAILURE);
        }
        char line[256];
        int flag = 0;
        ssize_t nread;
        while ((nread = read(fd, line, sizeof(line) - 1)) > 0) {
            line[nread] = '\0';
            char *start = line;
            char *end;
            while ((end = strchr(start, '\n')) != NULL) {
                *end = '\0';
                if (strstr(start, searchName) != NULL) {
                    printf("%s\n", start);
                    flag = 1;
                    break;
                }
                start = end + 1;
            }
            if (flag) {
                break;
            }
        }
        close(fd);
        char logMessage[512];
        if (flag) {
            snprintf(logMessage, sizeof(logMessage), "Search completed. Student flag: %s", searchName);
        } else {
            snprintf(logMessage, sizeof(logMessage), "Search completed. Student not flag: %s", searchName);
        }
        writeLog(logMessage);

        exit(flag ? EXIT_SUCCESS : EXIT_FAILURE);
    } else {
        int status;
        waitpid(pid, &status, 0);
    }
}

```

Show All

```
void showAll(const char *filename) {
    pid_t pid = fork();
    if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        int fd = open(filename, O_RDONLY);
        if (fd == -1) {
            perror("Error opening file");
            exit(EXIT_FAILURE);
        }
        char buffer[256];
        ssize_t bytes_read;
        while ((bytes_read = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytes_read] = '\0';
            printf("%s", buffer);
        }
        close(fd);
        exit(EXIT_SUCCESS);
    } else {
        int status;
        waitpid(pid, &status, 0);
        if (WIFEXITED(status) && WEXITSTATUS(status) == EXIT_SUCCESS) {
            writeLog("Displayed all student grades successfully.");
        } else {
            writeLog("Failed to display all student grades.");
        }
    }
}
```

Sorted

```
void sortStudentGrades(const char *filename) {
    pid_t pid = fork();
    if (pid == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid > 0) {
        int status;
        waitpid(pid, &status, 0);
        if (WIFEXITED(status)) {
            printf("Sorting completed.\n");
        }
    } else {
        int fd = open(filename, O_RDONLY);
        if (fd == -1) {
            perror("Error opening file for reading");
            exit(EXIT_FAILURE);
        }
        const int maxLines = 100;
        const int maxLineLength = 260;
        char buffer[maxLineLength * maxLines];
        Student students[maxLines];
        ssize_t bytesRead = read(fd, buffer, sizeof(buffer) - 1);
        if (bytesRead == -1) {
            perror("Error reading file");
            close(fd);
            exit(EXIT_FAILURE);
        }
        buffer[bytesRead] = '\0';
        close(fd);
        char *line = strtok(buffer, "\n");
        int lineCount = 0;
        while (line != NULL && lineCount < maxLines) {
            char *comma = strchr(line, ',');
            if (comma) {
                *comma = '\0'; // Split the string at the comma
                students[lineCount].name = strdup(line);
                students[lineCount].grade = strdup(comma + 2); // Skip comma and space
            } else { // In case there's a line without a comma
                students[lineCount].name = strdup(line);
                students[lineCount].grade = strdup("");
            }
            lineCount++;
            line = strtok(NULL, "\n");
        }
    }
}
```

```
    fd = open(filename, O_WRONLY | O_TRUNC);
    if (fd == -1) {
        perror("Error opening file for writing");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < lineCount; ++i) {
        dprintf(fd, "%s, %s\n", students[i].name, students[i].grade);
        free(students[i].name);
        free(students[i].grade);
    }
    close(fd);

    exit(EXIT_SUCCESS);
}
```

4-) Outputs/Tests


```
cagri.yilmaz@DESKTOP-97994NB: ~/mte/c/OSCI-3/yilmaz/Desktop/ea
GTU Student Grade Management System
> gtuStudentGrades grades.txt
grades.txt is ready for use.
>
```

The initialization of the file is performed first. If the file already exists, new entries are appended to it; if not, a new file is created.

```
> addStudentGrade "Cagri Yilmaz" "BB"
Grade added for Cagri Yilmaz
> addStudentGrade "Emir Yilmaz" "AA"
Grade added for Emir Yilmaz
> addStudentGrade "Beyzanur Polat" "FF"
Unknown command or file not initialized.
> addStudentGrade "Beyzanur Polat" "FF"
Grade added for Beyzanur Polat
> addStudentGrade "Onur Gumuser" "CC"
Grade added for Onur Gumuser
> addStudentGrade "Abdullah istemihan" "BA"
Grade added for Abdullah istemihan
> addStudentGrade "Yusuf Ucar" "DD"
Grade added for Yusuf Ucar
>
```

Adding a student and their letter grade is performed in this manner, providing output based on whether the operation is successful or not.

Here is the grades.txt:



```
grades - Not Defteri
Dosya  Düzen  Biçim  Görünüm  Yardım
Cagri Yilmaz, BB
Emir Yilmaz, AA
Beyzanur Polat, FF
Onur Gumuser, CC
Abdullah istemihan, BA
Yusuf Ucar, DD
|
```


In the search section, there are two types of warning situations. The first is that if you don't write in the required format, you receive an error, and if the student you are looking for does not exist, you receive an error. Otherwise, if the student you are searching for exists, you can see their output along with the letter grade.

```
> searchStudent Yusuf Ucar
Invalid command format. Correct format: searchStudent "Name Surname"
> searchStudent "Yusuf Ucar"
Yusuf Ucar, DD
```


```
> searchStudent "Osman Ucar"
Search completed. Student not flag: Osman Ucar
>
```

In the showAll section, we see all students and their letter grades.

```
> showAll
Cagri Yilmaz, BB
Emir Yilmaz, AA
Beyzanur Polat, FF
Onur Gumuser, CC
Abdullah istemihan, BA
Yusuf Ucar, DD
>
```

In the Sorting section, there are two different ways to sort: If you write sortAll, it sorts by name, whereas if you write sortAllByGrades, it sorts according to letter grades.


```
> sortAll
Sorting completed.
```

 grades - Not Defteri

Dosya	Düzen	Biçim	Görünüm
-------	-------	-------	---------

Abdullah istemihan, BA
Beyzanur Polat, FF
Cagri Yilmaz, BB
Emir Yilmaz, AA
Onur Gumuser, CC
Yusuf Ucar, DD

```
> addStudentGrade "Cagri Yilmaz" "BA"
Grade added for Cagri Yilmaz
> addStudentGrade "Emir Yilmaz" "AA"
Grade added for Emir Yilmaz
> sortAllByGrades
Sorting completed.
> sortAll
Sorting completed.
> addStudentGrade "Beyzanur Polat" "BB"
Grade added for Beyzanur Polat
> addStudentGrade "Huseyin Omer Guray" "DD"
Grade added for Huseyin Omer Guray
> sortAllByGrades
Sorting completed.
```

 yeni - Not Defteri

Dosya	Düzen	Biçim	Görünüm	Yardım
-------	-------	-------	---------	--------

Emir Yilmaz, AA
Cagri Yilmaz, BA
Beyzanur Polat, BB
Huseyin Omer Guray, DD

In the Logging section, I write everything that is required into the action file as follows.

```
[2024-03-22 17:02:10] Grade added successfully
[2024-03-22 17:02:30] Grade added successfully
[2024-03-22 17:02:54] Grade added successfully
[2024-03-22 17:03:13] Grade added successfully
[2024-03-22 17:03:49] Grade added successfully
[2024-03-22 17:05:14] Grade added successfully
[2024-03-22 17:13:58] Search completed. Student flag: Yusuf Ucar
[2024-03-22 17:14:40] Search completed. Student not flag: YTL OSMAN
[2024-03-22 17:16:56] Search completed. Student not flag: YTL OSMAN
[2024-03-22 17:18:04] Search completed. Student not flag: Osman Ucar
[2024-03-22 17:19:28] Displayed all student grades successfully.
[2024-03-22 17:31:34] Grade added successfully
[2024-03-22 17:32:24] Grade added successfully
[2024-03-22 17:38:39] Grade added successfully
[2024-03-22 17:38:48] Grade added successfully
[2024-03-22 17:47:39] Grade added successfully
[2024-03-22 17:47:47] Grade added successfully
[2024-03-22 17:51:32] Grade added successfully
[2024-03-22 17:51:46] Grade added successfully
```