CSE344 Systems Programming

Homework 1 Report

Ömer Faruk Çolakel

200104004043

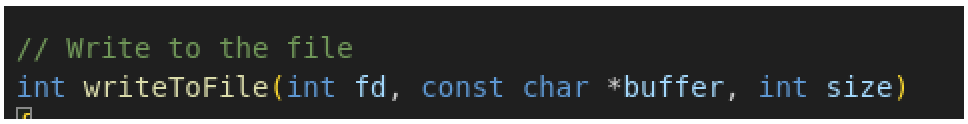
How to Compile and Run

In the root directory, you can run “make” in the terminal to compile the program. “make run” runs the compiled program. “make clean” deletes the executable but doesn’t do anything to the logger or the “grades.txt”.

Logger and “Grades.txt”

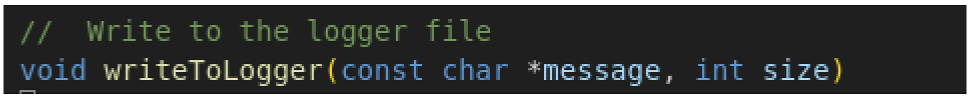
You can find the logger and grades files in the zip that I have submitted.

Function Purposes and I/O’s



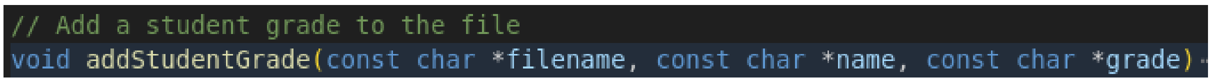
```
// Write to the file
int writeToFile(int fd, const char *buffer, int size)
```

1) This function handles general writing to file operations. It requires a file descriptor to write to, a buffer to get what will be written and the size of the string that will be written. It doesn’t open a file and only writes to it. Returns the number of bytes written to file and -1 if there was an error. Ignores EINTR while writing and waits until the error is handled. I generally used it to write to the terminal with stdout and stderr.



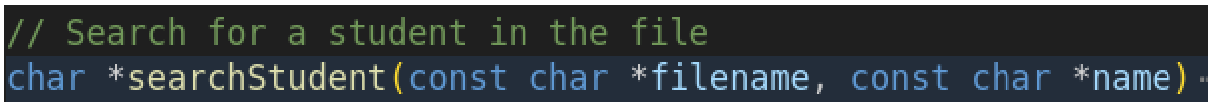
```
//  Write to the logger file
void writeToLogger(const char *message, int size)
```

2) Very close to writeToFile. It writes LOGGER_NAME. LOGGER_NAME is a predefined keyword and holds the name of the logger file. It opens the logger, gets the current date and time and writes date, time and the message with the length of size. It checks the errors also. Creates a logger if it’s absent.

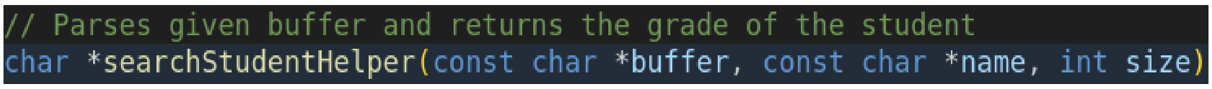


```
// Add a student grade to the file
void addStudentGrade(const char *filename, const char *name, const char *grade)
```

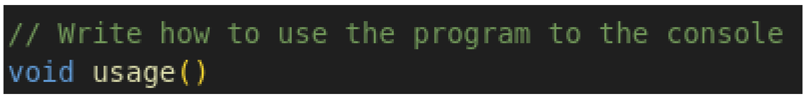
3) Adds a student with his/her grade to the file with the given name. Requires a file name, student name and grade that all are a char array. Opens the file on write only and append mode. It gives an error if there was a problem while opening (there are no files with a given name for example). Creates a char array with the size of name + grade + 2. Writes the name and grade to it and then uses writeToFile to write to the given file. Writes error if the returned value from writeToFile is not equal to bytes to be written. Closes files on both success and failure. Prints the result to both terminal and logger.


```
// Search for a student in the file
char *searchStudent(const char *filename, const char *name)
```

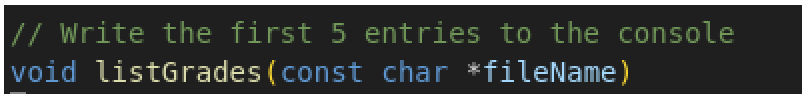
4) Searches for the student with the given name in the given file. Opens the file and checks if it's successful. If it fails, print the error to terminal and logger and returns NULL. Uses buffering and reads the file by 1KB at each iteration. The benefit of this is that the program doesn't have to switch from user to kernel space constantly. The downside is that sometimes a line is cut from the middle. Since I used the newline character to parse the lines, this causes us to lose some lines. To avoid that, the program checks the last newline and cuts the part after it. Then sends the remaining part to searchStudentHelper (which I will explain later). After that function executes, writes the saved part to the start of the new buffer and fills the rest of it by reading the file again. This works until it finds the student or reaches to the end without finding it.


```
// Parses given buffer and returns the grade of the student
char *searchStudentHelper(const char *buffer, const char *name, int size)
```

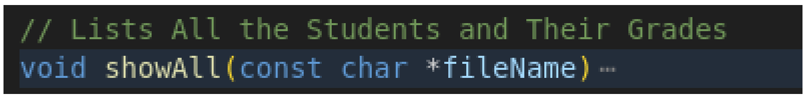
5) This function searches for the given name in the buffer in the given range (size). Parses the buffer using newline characters. A line contains a grade and a name. I assumed that a grade will always be a string without a space while a name can have space(s). To parse a line, it starts from the end of the line and searches for a space. The part after the space is grade and everything before that is the name. Then it compares the found name with the given name and if they are the same, returns the grade. If there are no lines left, it returns NULL since the name is not in the buffer.


```
// Write how to use the program to the console
void usage()
```

6) Prints how to use the program to the terminal. Utilizes the writeToFile to do it. Checks if there was an error while printing. Writes "Usage printed" to the logger.

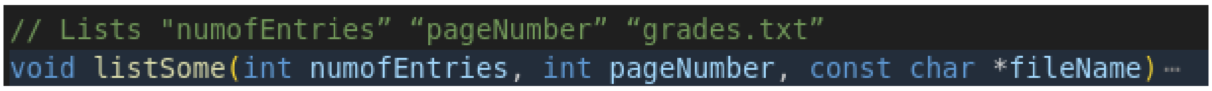

```
// Write the first 5 entries to the console
void listGrades(const char *fileName)
```

7) Write the first 5 entries to the terminal. Opens the file in read only mode and checks if it was successful. Reads the buffer until either there was an error, at the end of the file or successfully writing all 5 entries. On error, writes the error to both terminal and logger. On the other two scenarios, writes the success message to both terminal and logger. The 5 entries are printed only to the terminal. If there are less than 5 entries, it prints all the entries. In most situations it's not necessary to carry an incomplete line at the end of the buffer to the start of the buffer in the next iteration but for safety reasons I used it anyway.



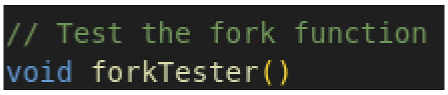
```c
// Lists All the Students and Their Grades
void showAll(const char *fileName)⋯
```

8) showAll prints all students and their grades to the terminal. It opens the file with the given name or returns if there was an error while opening. It reads the file by 1KB at max at each iteration and writes it.



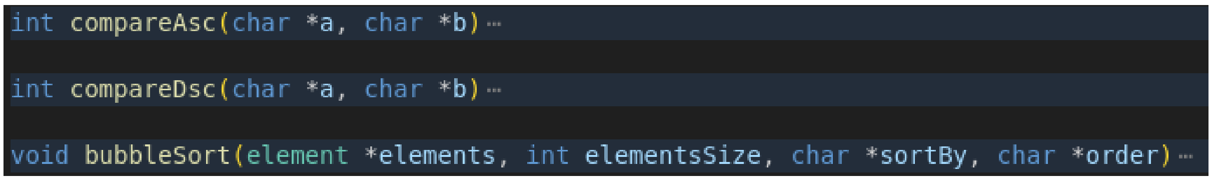
```c
// Lists "numofEntries" “pageNumber” “grades.txt”
void listSome(int numofEntries, int pageNumber, const char *fileName)⋯
```

9) Separates the students in the file called filename into groups of “numofEntries” and prints the “pageNumber”th group. For example, if the page number is 1 and the number of entries is 10, the program prints the first 10 students. If the page number is 2, then it prints the next 10. The function opens the file with the given name and reads it into the buffer by 1KB. Then parses the buffer with the newline character and counts the lines. The carry logic is working here as well.



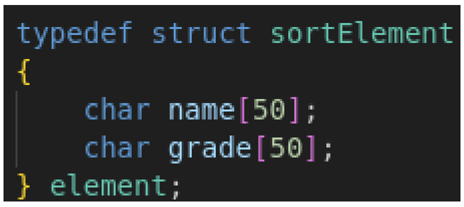
```c
// Test the fork function
void forkTester()
```

10) This function has a while loop that runs infinitely. I used a fork to run this function. It proves that the fork is utilized as intended since the user can run the program while this function runs in a child process.



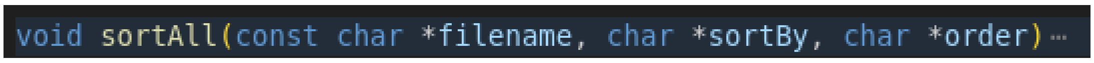
```c
int compareAsc(char *a, char *b)⋯

int compareDsc(char *a, char *b)⋯

void bubbleSort(element *elements, int elementsSize, char *sortBy, char *order)⋯
```

11) compareAsc is used to compare two strings in ascending order. So, when the string “A” and “B” is given as input it returns “-1”. compareDsc does the opposite. bubbleSort sorts the elements list with the given size according to the given parameters. These parameters are sortBy and order. sortBy can be either “name” or “grade” and order can be either “asc” or “dsc”. Anything other than these makes the function return and print errors.



```c
typedef struct sortElement
{
    char name[50];
    char grade[50];
} element;
```

12) This isn’t a function but since I have mentioned I want to give some information about the “element”. Element is just a bias for sortElement which holds a name and grade with fixed sizes 50. When the user enters “sortAll” and gives a valid file name, the file is read in the main function and every line is parsed into an element. These elements are in an array which is pointed by a pointer. To sort this array, I used the bubbleSort function.



```
void sortAll(const char *filename, char *sortBy, char *order)…
```

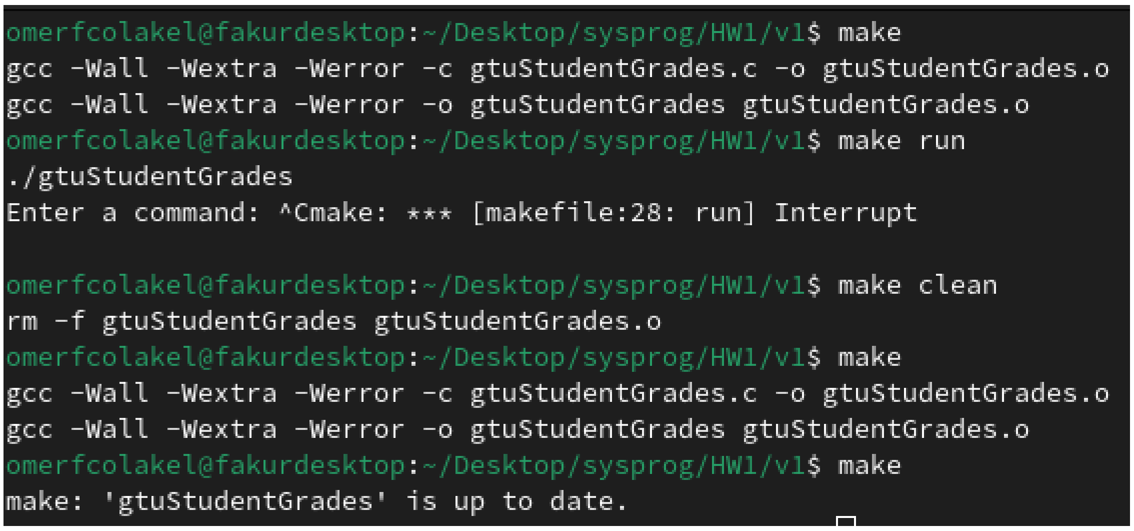
13) sortAll sorts all the students in the file with given name by either name or grade in ascending or descending order. I already explained what it is doing in the twelfth entry.

14) Finally, the main function reads the inputs from the user and parses each of them to command and args. “Command” determines which function(s) to run and args are parameters to run those functions. If the given command is wrong or the args are not enough, the main function calls the usage function to remind the user how the input is expected to be. It also uses fork(), so the called function works in the background and main still expects input from the user.

Example Inputs and Corresponding Outputs

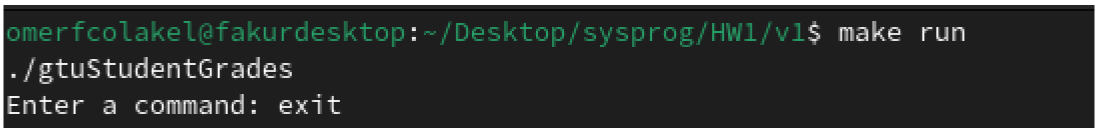
Note: All arguments should be between “ “ characters. Command shouldn’t be in between them though.

makefile:



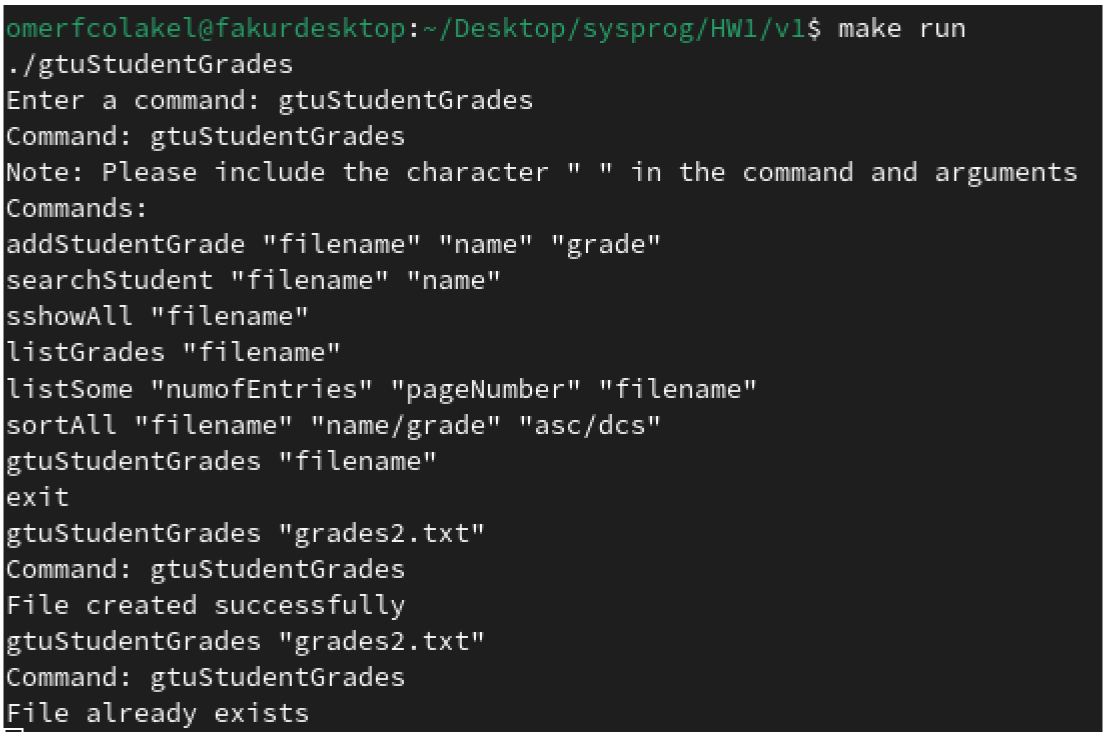
```
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make
gcc -Wall -Wextra -Werror -c gtuStudentGrades.c -o gtuStudentGrades.o
gcc -Wall -Wextra -Werror -o gtuStudentGrades gtuStudentGrades.o
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make run
./gtuStudentGrades
Enter a command: ^Cmake: *** [makefile:28: run] Interrupt

omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make clean
rm -f gtuStudentGrades gtuStudentGrades.o
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make
gcc -Wall -Wextra -Werror -c gtuStudentGrades.c -o gtuStudentGrades.o
gcc -Wall -Wextra -Werror -o gtuStudentGrades gtuStudentGrades.o
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make
make: 'gtuStudentGrades' is up to date.
```

exit:



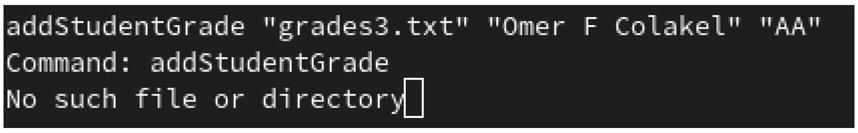
```
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make run
./gtuStudentGrades
Enter a command: exit
```

gtuStudentGrades: This has 2 functionalities. One is that, it calls usage() and the other is creating a file if it doesn't already exist.

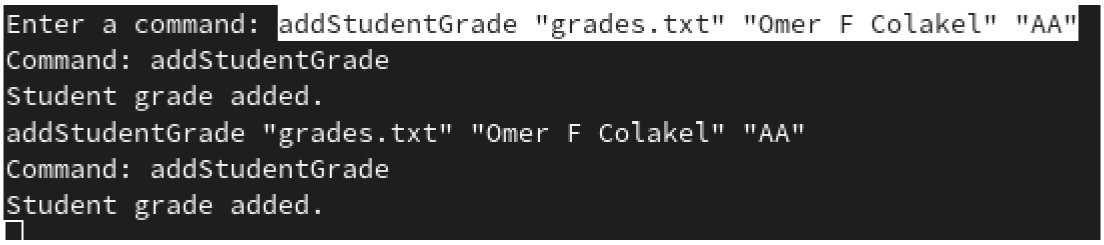


```
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make run
./gtuStudentGrades
Enter a command: gtuStudentGrades
Command: gtuStudentGrades
Note: Please include the character " " in the command and arguments
Commands:
addStudentGrade "filename" "name" "grade"
searchStudent "filename" "name"
sshowAll "filename"
listGrades "filename"
listSome "numofEntries" "pageNumber" "filename"
sortAll "filename" "name/grade" "asc/dcs"
gtuStudentGrades "filename"
exit
gtuStudentGrades "grades2.txt"
Command: gtuStudentGrades
File created successfully
gtuStudentGrades "grades2.txt"
Command: gtuStudentGrades
File already exists
```

addStudentGrade:



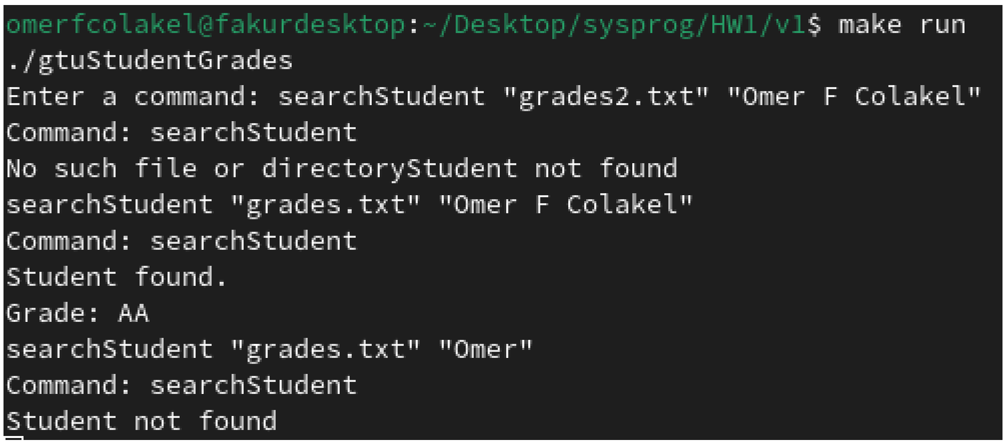
```
addStudentGrade "grades3.txt" "Omer F Colakel" "AA"
Command: addStudentGrade
No such file or directory
```



```
Enter a command: addStudentGrade "grades.txt" "Omer F Colakel" "AA"
Command: addStudentGrade
Student grade added.
addStudentGrade "grades.txt" "Omer F Colakel" "AA"
Command: addStudentGrade
Student grade added.
```

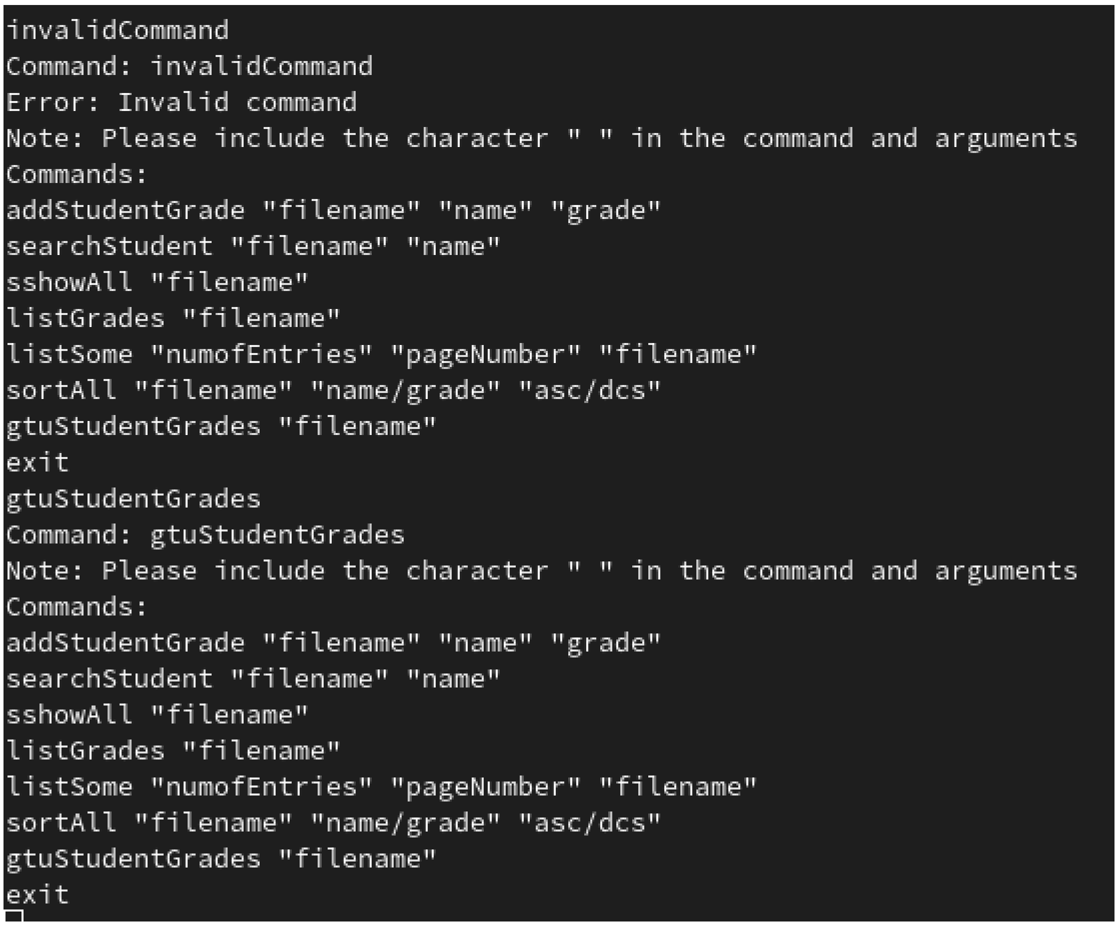
Please note that this function doesn't overwrite existing students with the same name.

searchStudent:



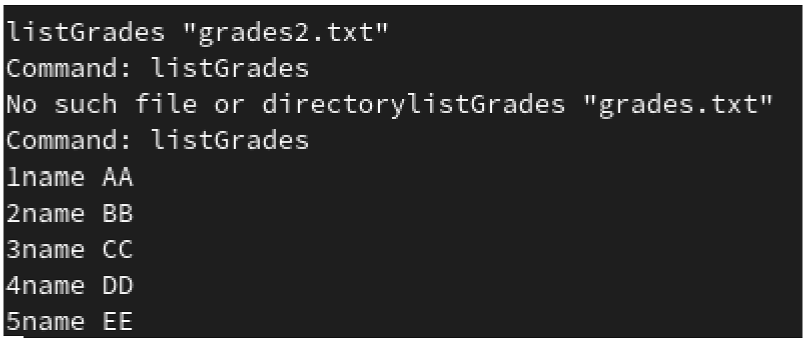
```
omerfcolakel@fakurdesktop:~/Desktop/sysprog/HW1/v1$ make run
./gtuStudentGrades
Enter a command: searchStudent "grades2.txt" "Omer F Colakel"
Command: searchStudent
No such file or directoryStudent not found
searchStudent "grades.txt" "Omer F Colakel"
Command: searchStudent
Student found.
Grade: AA
searchStudent "grades.txt" "Omer"
Command: searchStudent
Student not found
```

usage:



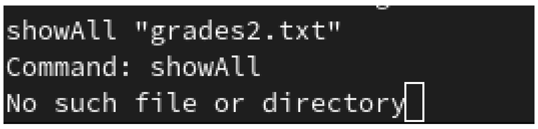
```
invalidCommand
Command: invalidCommand
Error: Invalid command
Note: Please include the character " " in the command and arguments
Commands:
addStudentGrade "filename" "name" "grade"
searchStudent "filename" "name"
sshowAll "filename"
listGrades "filename"
listSome "numofEntries" "pageNumber" "filename"
sortAll "filename" "name/grade" "asc/dcs"
gtuStudentGrades "filename"
exit
gtuStudentGrades
Command: gtuStudentGrades
Note: Please include the character " " in the command and arguments
Commands:
addStudentGrade "filename" "name" "grade"
searchStudent "filename" "name"
sshowAll "filename"
listGrades "filename"
listSome "numofEntries" "pageNumber" "filename"
sortAll "filename" "name/grade" "asc/dcs"
gtuStudentGrades "filename"
exit
```

listGrades:

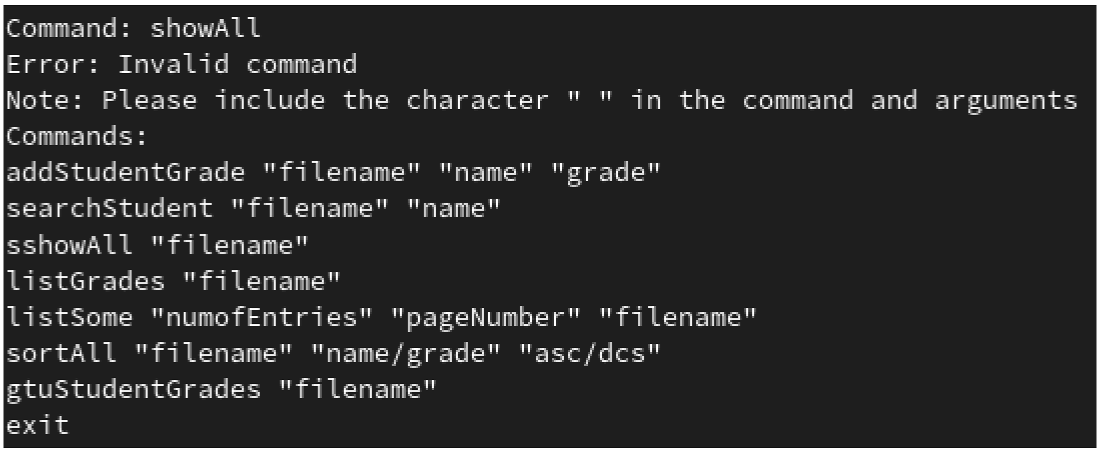


```
listGrades "grades2.txt"
Command: listGrades
No such file or directorylistGrades "grades.txt"
Command: listGrades
1name AA
2name BB
3name CC
4name DD
5name EE
```

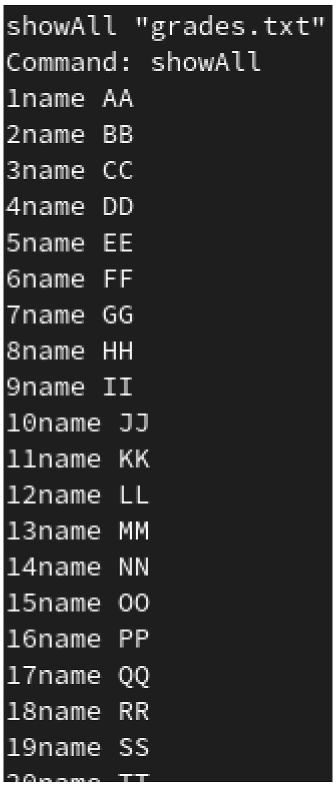
showAll:



```
showAll "grades2.txt"
Command: showAll
No such file or directory
```

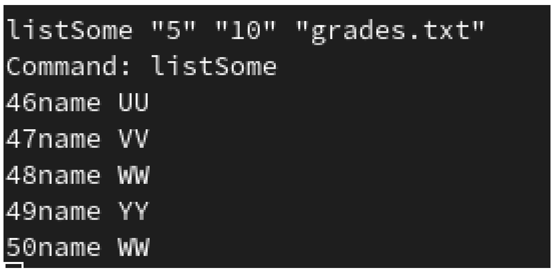


```
Command: showAll
Error: Invalid command
Note: Please include the character " " in the command and arguments
Commands:
addStudentGrade "filename" "name" "grade"
searchStudent "filename" "name"
sshowAll "filename"
listGrades "filename"
listSome "numofEntries" "pageNumber" "filename"
sortAll "filename" "name/grade" "asc/dcs"
gtuStudentGrades "filename"
exit
```



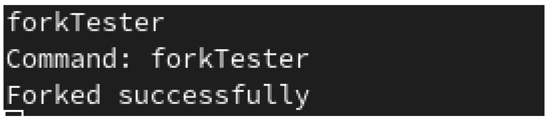
```
showAll "grades.txt"
Command: showAll
1name AA
2name BB
3name CC
4name DD
5name EE
6name FF
7name GG
8name HH
9name II
10name JJ
11name KK
12name LL
13name MM
14name NN
15name OO
16name PP
17name QQ
18name RR
19name SS
```

listSome:



```
listSome "5" "10" "grades.txt"
Command: listSome
46name UU
47name VV
48name WW
49name YY
50name WW
```

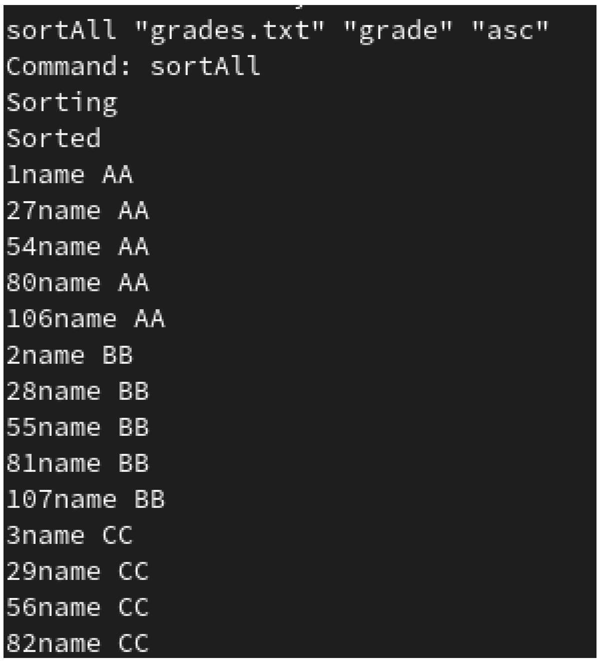
forkTester:



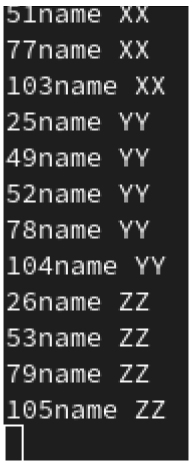
```
forkTester
Command: forkTester
Forked successfully
```

Note that we can still use the program.

sortAll:



```
sortAll "grades.txt" "grade" "asc"
Command: sortAll
Sorting
Sorted
1name AA
27name AA
54name AA
80name AA
106name AA
2name BB
28name BB
55name BB
81name BB
107name BB
3name CC
29name CC
56name CC
82name CC
```



```
51name XX
77name XX
103name XX
25name YY
49name YY
52name YY
78name YY
104name YY
26name ZZ
53name ZZ
79name ZZ
105name ZZ
```