GTU Department of Computer Engineering
CSE 344 – Spring 2024
Homework 1 Report

Barış Batuhan Bolat
210104004029

Algorithm and Code

1. Main

The shell repeatedly reads user input and executes commands.

a) **Loop**

- An infinite loop (**while(1)**) keeps the shell running until the user exits.
- The prompt (**$**) is written to standard output (**STDOUT_FILENO**) using **write** system call.
- The **read** system call reads user input from standard input (**STDIN_FILENO**) and stores it in the command array.
- Error handling is done using **perror** if read fails.
- A newline character (**\n**) at the end of the input is replaced with a null terminator (**\0**) using strcspn.

b) **Commands**

- The program checks if the user entered "**exit**". If yes, the loop exits, memory allocated for arguments is freed.
- If the user enters "**gtuStudentGrades**", it prints available commands.
- If the user enters a command other than "**exit**" or "**gtuStudentGrades**", the **split_command** function is called to split the command and arguments.
- Based on the number of arguments and the content of the first argument, different functionalities are executed:

  2 arguments:

  - If the command is "**gtuStudentGrades**" followed by a filename, the code checks if the file exists. If not, it creates an empty file.
  - If the command is "**sortAll**" or "**showAll**" or "**listGrades**" followed by a filename, the corresponding functions are called to sort or display all(**showAll)** or first few entries(**listGrades)** of the file. Sorting is done alphabetically.

  3 arguments:

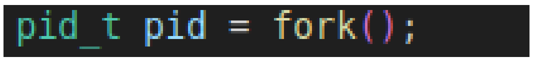
  - If the command is "**searchStudent**" followed by a name and a filename, the **searchStudent** function is called to find the student in the file.

  4 arguments:

  - If the command is "**addStudentGrade**" followed by a name, grade and filename, the addStudentGrade function is called to add the student and grade to the file.
  - If the command is "**listSome**" followed by number of entries, page number and filename, the listSome function is called to display entries on a specific page.
- After all operations, the log file is updated according to the result of the operation.

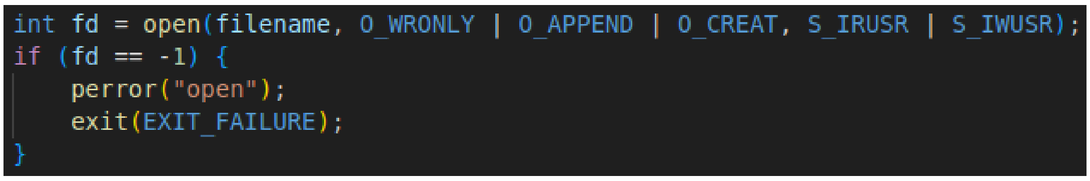
All following functions:

- All file descriptors closed at the end of function.
- The functions begins by creating a child process using fork.
- If fork fails, an error message is written using perror and the process exits.



```
pid_t pid = fork();
```

- The functions opens the file specified by filename in append mode (**O_APPEND**) , read only mode (**O_RDONLY**) and write only mode (**O_WRONLY**) for the owners (**S_IRUSR | S_IWUSR**).

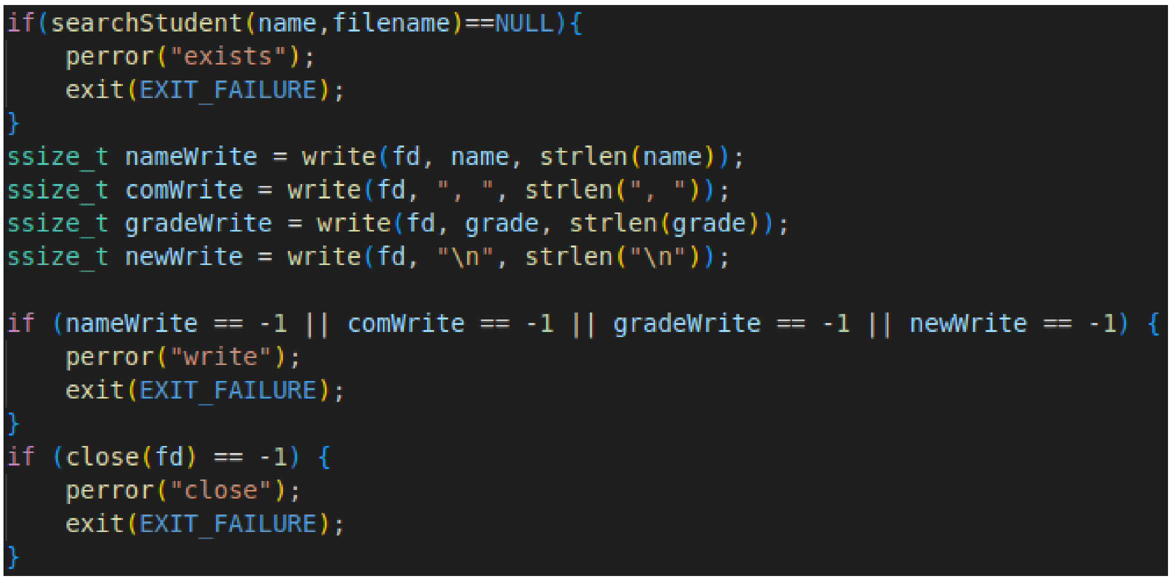


```
int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, S_IRUSR | S_IWUSR);
if (fd == -1) {
    perror("open");
    exit(EXIT_FAILURE);
}
```

2. addStudentGrade

a) Child Process (pid == 0)

- The function checks if the given student is exists in file. If exists an error message is written and the process exits.
- The function writes the student's name and grade at the end of file using write system call.
- Error handling is done using **perror** if any write operation fails. The child process exits if there is a write error.
- The file is closed using close. If closing fails, an error message is written and the process exits.
- The child process exits successfully (**EXIT_SUCCESS**).

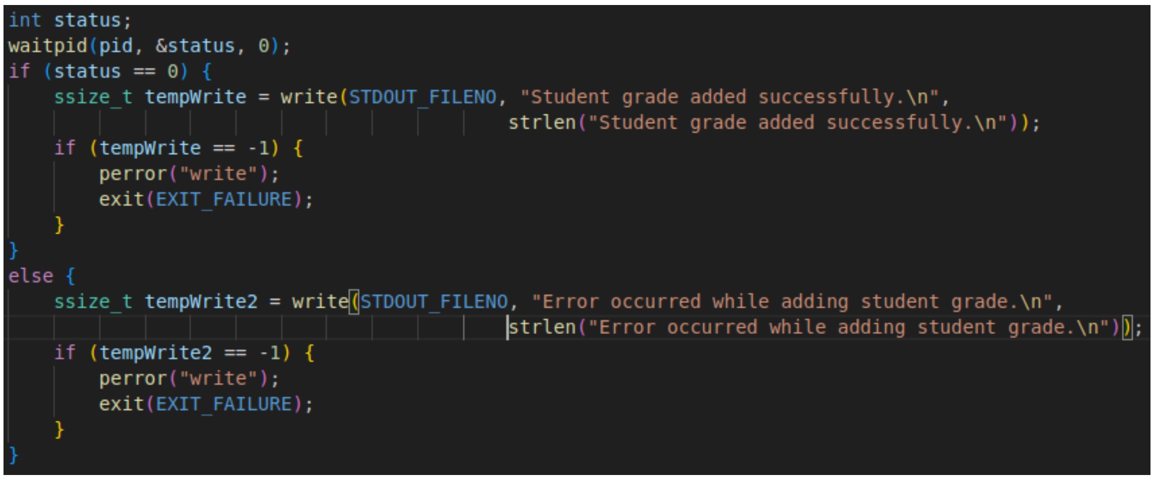


```
if(searchStudent(name,filename)==NULL){
    perror("exists");
    exit(EXIT_FAILURE);
}
ssize_t nameWrite = write(fd, name, strlen(name));
ssize_t comWrite = write(fd, ", ", strlen(", "));
ssize_t gradeWrite = write(fd, grade, strlen(grade));
ssize_t newWrite = write(fd, "\n", strlen("\n"));

if (nameWrite == -1 || comWrite == -1 || gradeWrite == -1 || newWrite == -1) {
    perror("write");
    exit(EXIT_FAILURE);
}
if (close(fd) == -1) {
    perror("close");
    exit(EXIT_FAILURE);
}
```

b) Parent Process (pid > 0)

- The parent process waits for the child process to finish using **waitpid**.
- The return status of the child process is stored in status.
- If the child process exited successfully (**status == 0**), a message indicating successful addition is written to standard output (**STDOUT_FILENO**).
- If the child process failed (**status != 0**), an error message indicating the failure is written to standard output.
- Error handling is done using **perror** if writing to standard output fails.

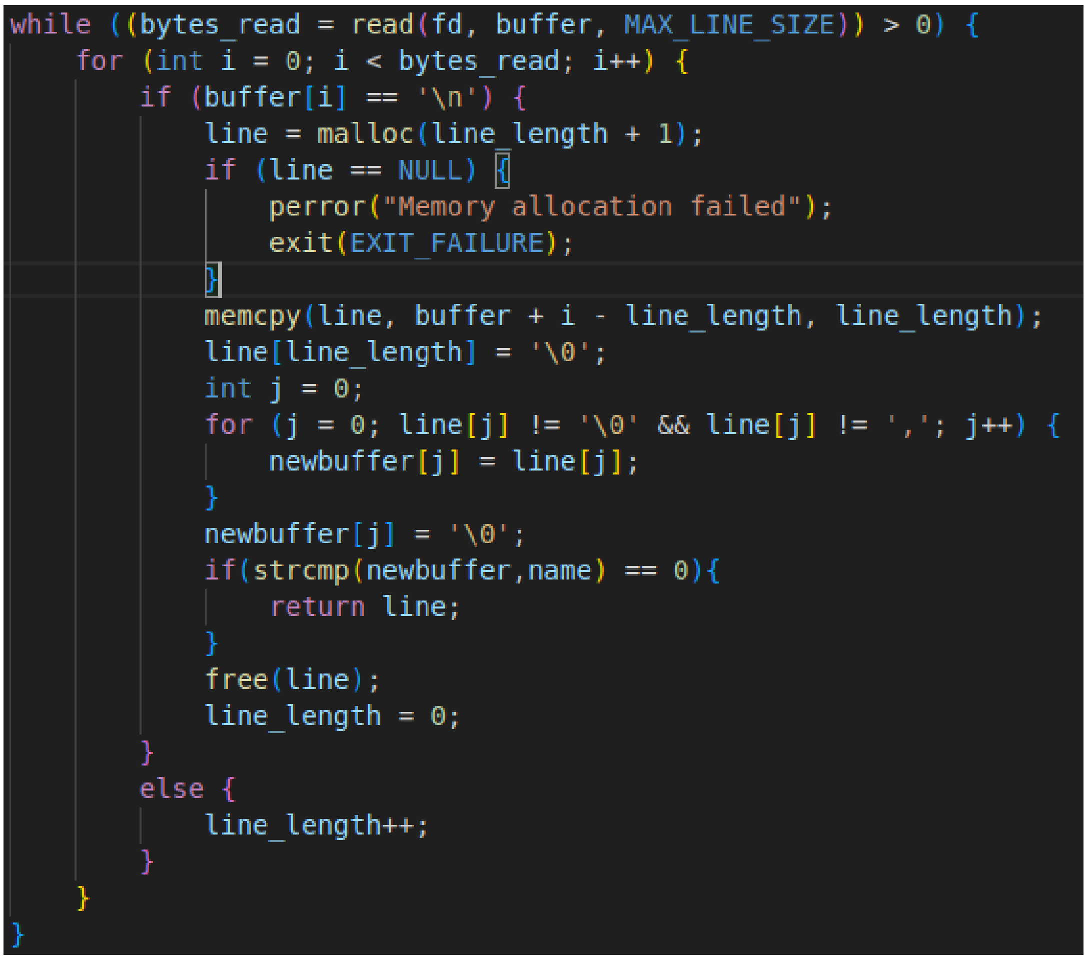


```c
int status;
waitpid(pid, &status, 0);
if (status == 0) {
    ssize_t tempWrite = write(STDOUT_FILENO, "Student grade added successfully.\n",
                                            strlen("Student grade added successfully.\n"));
    if (tempWrite == -1) {
        perror("write");
        exit(EXIT_FAILURE);
    }
}
else {
    ssize_t tempWrite2 = write(STDOUT_FILENO, "Error occurred while adding student grade.\n",
                                            strlen("Error occurred while adding student grade.\n"));
    if (tempWrite2 == -1) {
        perror("write");
        exit(EXIT_FAILURE);
    }
}
```

3. searchStudent

a) Child Process(pid == 0)

- A while loop continues as long as bytes are successfully read from the file using **read**.
- Inside the loop, each byte in the buffer is iterated through.
- If a newline character (**\n**) is encountered, it means a full line has been read.
- Memory is allocated using malloc for a new line to store the complete line from the buffer.
- Error handling is done using perror if memory allocation fails. The child process exits.
- The data in the buffer corresponding to the line is copied to the allocated memory using memcpy.
- A null terminator (**\0**) is added to the end of the copied line.
- The name stored in name is compared with the contents of newbuffer using **strcmp**.
- If a match is found, the entire line stored in allocated memory (containing name and grade) is returned by the function.
- If no match is found, the allocated memory for the line is freed using free and line_length is reset for the next line.
- After the loop exits, the file is closed using close.


```c
while ((bytes_read = read(fd, buffer, MAX_LINE_SIZE)) > 0) {
    for (int i = 0; i < bytes_read; i++) {
        if (buffer[i] == '\n') {
            line = malloc(line_length + 1);
            if (line == NULL) {
                perror("Memory allocation failed");
                exit(EXIT_FAILURE);
            }
            memcpy(line, buffer + i - line_length, line_length);
            line[line_length] = '\0';
            int j = 0;
            for (j = 0; line[j] != '\0' && line[j] != ','; j++) {
                newbuffer[j] = line[j];
            }
            newbuffer[j] = '\0';
            if(strcmp(newbuffer,name) == 0){
                return line;
            }
            free(line);
            line_length = 0;
        }
        else {
            line_length++;
        }
    }
}
```

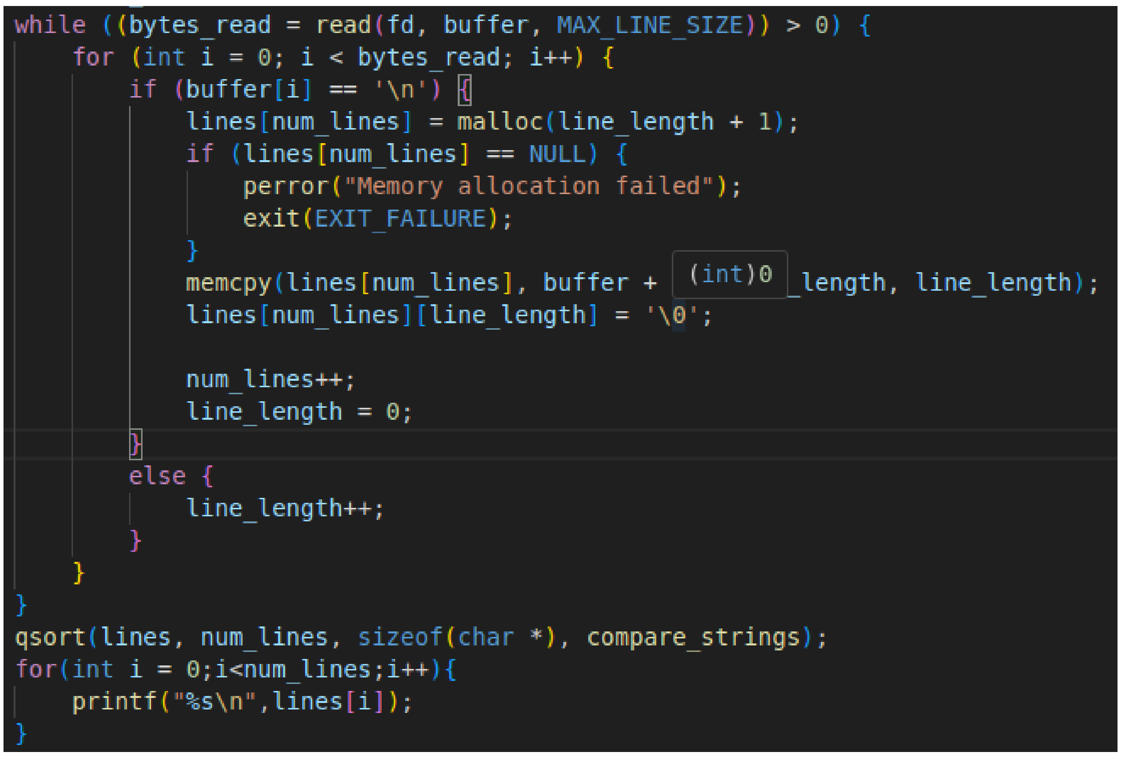
b) Parent Process(pid > 0)

- The parent process waits for the child process to finish using **waitpid**.
- The return status of the child process is stored in status.
- If the child process exited successfully (**status == 0**), a message indicating successful search is written to standard output (**STDOUT_FILENO**).
- If the child process failed (**status != 0**), an error message indicating the failure is written to standard output.
- Error handling is done using **perror** if writing to standard output fails.
- Same as previous functions parent code block.

4. sortAll

a) Child Process(pid == 0)

- A while loop continues as long as bytes are successfully read from the file using read.
- Inside the loop, each byte in the buffer is iterated through.
- If a newline character (**\n**) is encountered, it means a full line has been read.
- Memory is allocated using **malloc** for a new line to store the complete line from the buffer.
- The data in the buffer corresponding to the line is copied to the allocated memory using memcpy.
- After processing the line, line_length is reset to zero for the next line.
- Once the loop exits, the **qsort** function from the C library is used to sort the lines array.
- After sorting, the sorted lines are printed to standard output (stdout) using printf.



```c
while ((bytes_read = read(fd, buffer, MAX_LINE_SIZE)) > 0) {
    for (int i = 0; i < bytes_read; i++) {
        if (buffer[i] == '\n') {
            lines[num_lines] = malloc(line_length + 1);
            if (lines[num_lines] == NULL) {
                perror("Memory allocation failed");
                exit(EXIT_FAILURE);
            }
            memcpy(lines[num_lines], buffer + (int)0 _length, line_length);
            lines[num_lines][line_length] = '\0';

            num_lines++;
            line_length = 0;
        }
        else {
            line_length++;
        }
    }
}
qsort(lines, num_lines, sizeof(char *), compare_strings);
for(int i = 0;i<num_lines;i++){
    printf("%s\n",lines[i]);
}
```

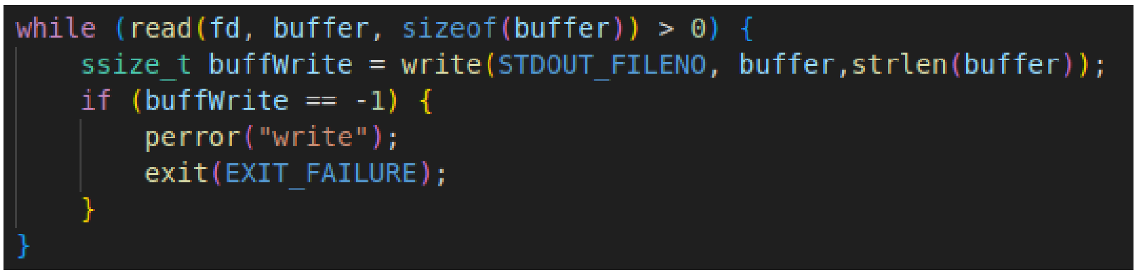
b) Parent Process(pid > 0)

- The parent process waits for the child process to finish using **waitpid**.
- The return status of the child process is stored in status.
- If the child process exited successfully (status == 0), a message indicating successful sort is written to standard output (STDOUT_FILENO).
- If the child process failed (status != 0), an error message indicating the failure is written to standard output.
- Error handling is done using perror if writing to standard output fails.
- Same as previous functions parent code block.

5. showAll

a) Child Process(pid == 0)

- The function opens the file specified by filename in read-only mode (**O_RDONLY**).
- If the file cannot be opened, an error message is written using **perror** and the child process exits.
- A while loop continues as long as bytes are successfully read from the file using **read**.
- Inside the loop, the entire contents of the buffer are written to standard output (STDOUT_FILENO) using **write**.
- Error handling is done using **perror** if writing to standard output fails. The child process exits.



```c
while (read(fd, buffer, sizeof(buffer)) > 0) {
    ssize_t buffWrite = write(STDOUT_FILENO, buffer,strlen(buffer));
    if (buffWrite == -1) {
        perror("write");
        exit(EXIT_FAILURE);
    }
}
```

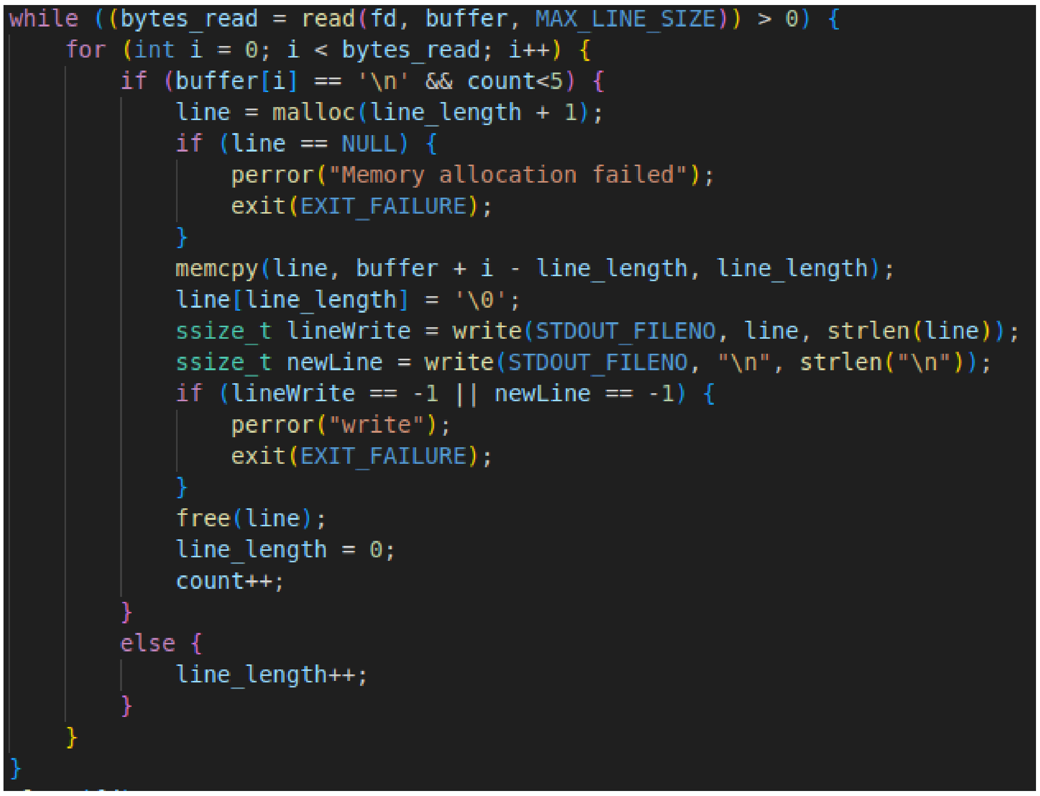
b) Parent Process(pid > 0)

- The parent process waits for the child process to finish using **waitpid**.
- The return status of the child process is stored in status.
- If the child process exited successfully (**status == 0**), a message indicating successful display is written to standard output.
- If the child process failed (**status != 0**), an error message indicating the failure is written to standard output.
- Error handling is done using **perror** if writing to standard output fails.
- Same as previous functions parent code block.

6. listGrades

a) Child Process(pid == 0)

- Initializes a counter count to keep track of the lines printed.
- A while loop continues as long as bytes are successfully read from the file using **read**.
- Inside the loop, each byte in the buffer is iterated through.
- If a newline character (**\n**) is encountered and count is less than 5:
    - Allocates memory for the line using **malloc**.
    - Copies the line from the buffer to the allocated memory using **memcpy**.
    - Writes the line to standard output (**STDOUT_FILENO**) using **write**.
    - Frees the allocated memory for the line.
    - Resets **line_length** for the next line.
    - Increments count.



```c
while ((bytes_read = read(fd, buffer, MAX_LINE_SIZE)) > 0) {
    for (int i = 0; i < bytes_read; i++) {
        if (buffer[i] == '\n' && count<5) {
            line = malloc(line_length + 1);
            if (line == NULL) {
                perror("Memory allocation failed");
                exit(EXIT_FAILURE);
            }
            memcpy(line, buffer + i - line_length, line_length);
            line[line_length] = '\0';
            ssize_t lineWrite = write(STDOUT_FILENO, line, strlen(line));
            ssize_t newLine = write(STDOUT_FILENO, "\n", strlen("\n"));
            if (lineWrite == -1 || newLine == -1) {
                perror("write");
                exit(EXIT_FAILURE);
            }
            free(line);
            line_length = 0;
            count++;
        }
        else {
            line_length++;
        }
    }
}
```

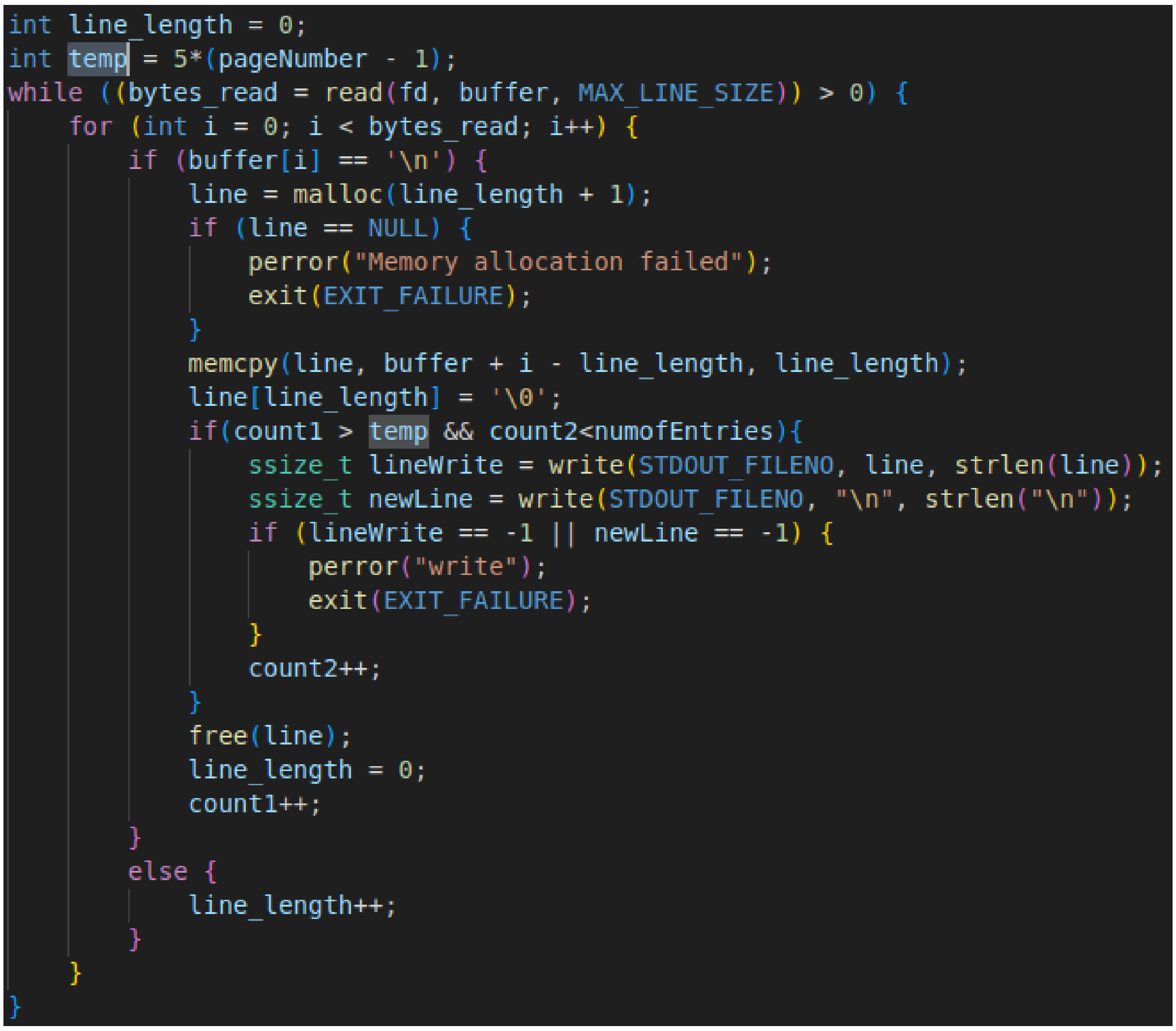
b) Parent Process(pid > 0)

- Waits for the child process to finish using **waitpid**.
- If the child process exited successfully (**status == 0**):
- Writes a success message to standard output, indicating that the first 5 students were listed successfully.
- If the child process failed (**status != 0**), writes an error message to standard output.
- If any write operations fail, exits with **EXIT_FAILURE**.
- Same as previous functions parent code block.

7. listSome

a) Child Process(pid == 0)

- Each page has 5 entries.
- A while loop continues as long as bytes are successfully read from the file using read.
- Inside the loop, each byte in the buffer is iterated through.
- If a newline character **(\n)** is encountered, a full line has been read.
- Allocates memory for the line using **malloc**.
- Copies the line from the buffer to the allocated memory using **memcpy**.
- Checks if the current line is within the specified page and if fewer than **numofEntries** entries have been displayed for that page.
- If so, writes the line to standard output (**STDOUT_FILENO**) using **write** and increments count2.
- Frees the allocated memory for the line.
- Resets **line_length** for the next line.
- Increments count1 for the total line count.



```c
int line_length = 0;
int temp = 5*(pageNumber - 1);
while ((bytes_read = read(fd, buffer, MAX_LINE_SIZE)) > 0) {
    for (int i = 0; i < bytes_read; i++) {
        if (buffer[i] == '\n') {
            line = malloc(line_length + 1);
            if (line == NULL) {
                perror("Memory allocation failed");
                exit(EXIT_FAILURE);
            }
            memcpy(line, buffer + i - line_length, line_length);
            line[line_length] = '\0';
            if(count1 > temp && count2<numofEntries){
                ssize_t lineWrite = write(STDOUT_FILENO, line, strlen(line));
                ssize_t newLine = write(STDOUT_FILENO, "\n", strlen("\n"));
                if (lineWrite == -1 || newLine == -1) {
                    perror("write");
                    exit(EXIT_FAILURE);
                }
                count2++;
            }
            free(line);
            line_length = 0;
            count1++;
        }
        else {
            line_length++;
        }
    }
}
```

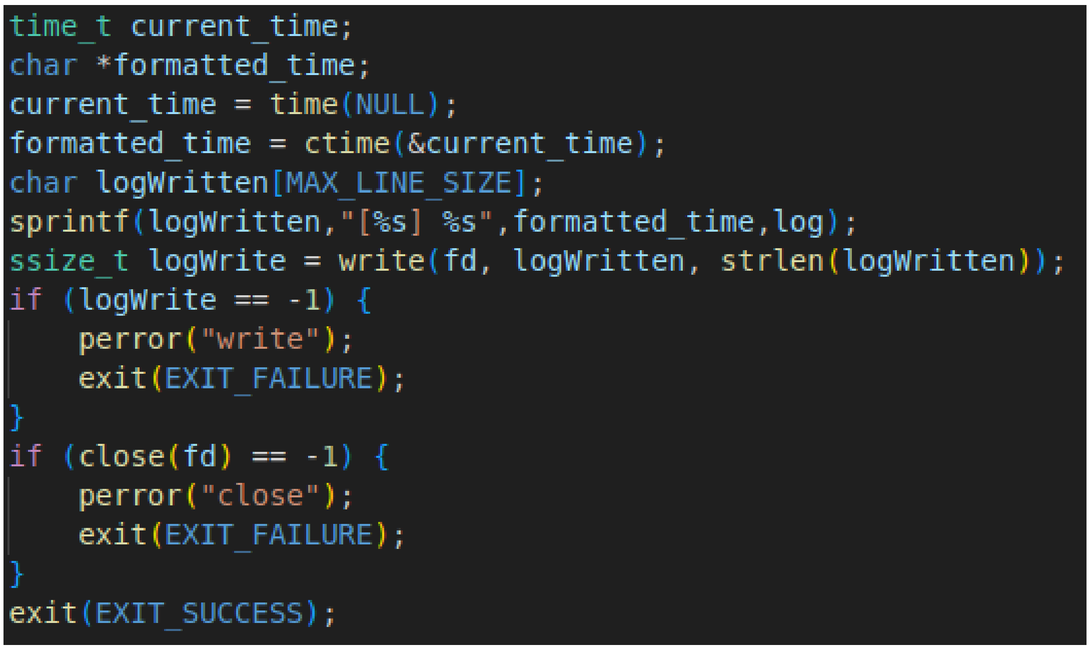
b) **Parent Process(pid > 0)**

- Waits for the child process to finish using **waitpid**.
- If the child process exited successfully (**status == 0**):
- Writes a success message to standard output, indicating the page number and number of entries displayed.
- If the child process failed (**status != 0**), writes an error message to standard output.
- If any write operations fail, exits with **EXIT_FAILURE**.
- Same as previous functions parent code block.

8. Logging

a) **Child Process (pid == 0):**

- Gets the current time using time(NULL).
- Converts the time to a human-readable format using ctime(&current_time).
- Creates a new string logWritten by prepending the formatted timestamp to the input message using sprintf.
- Writes the logWritten string to the log file using write(fd, logWritten, strlen(logWritten)). Exits on failure with error message.

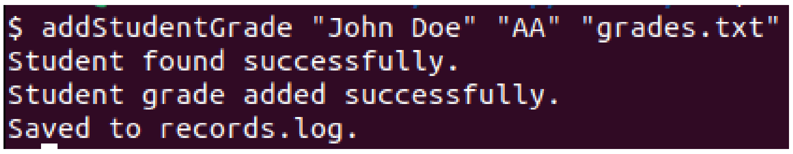


```c
time_t current_time;
char *formatted_time;
current_time = time(NULL);
formatted_time = ctime(&current_time);
char logWritten[MAX_LINE_SIZE];
sprintf(logWritten,"[%s] %s",formatted_time,log);
ssize_t logWrite = write(fd, logWritten, strlen(logWritten));
if (logWrite == -1) {
    perror("write");
    exit(EXIT_FAILURE);
}
if (close(fd) == -1) {
    perror("close");
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
```

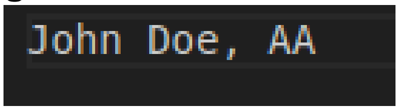
b) **Parent Process (if pid > 0):**

- Waits for the child process to finish using **waitpid(pid, &status, 0).**
- Checks the child process exit status (**status**).
- If **status == 0**, the child process exited successfully. Prints a confirmation message to standard output using write.
- If **status** is not 0, an error occurred in the child process. Prints an error message to standard output using write.
- Same as previous functions parent code block.

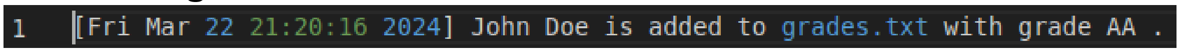
Test Cases

1. **Test Case :** Add Student Grade
   Input : addStudentGrade "John Doe" "AA" "grades.txt"
   Results :
   Terminal
   

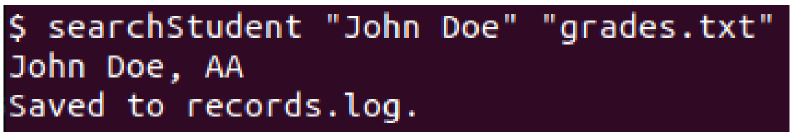
```
$ addStudentGrade "John Doe" "AA" "grades.txt"
Student found successfully.
Student grade added successfully.
Saved to records.log.
```

   grades.txt
   

```
John Doe, AA
```

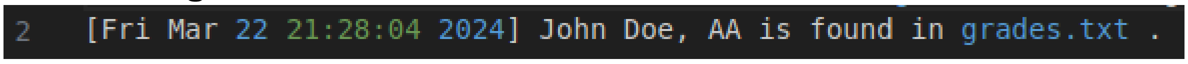
   records.log
   

```
[Fri Mar 22 21:20:16 2024] John Doe is added to grades.txt with grade AA .
```

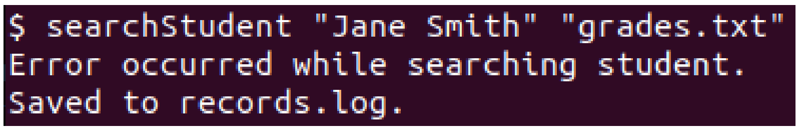
2. **Test Case :** Search for Student Grade that exists
   Input : searchStudent "John Doe" "grades.txt"
   Results :
   Terminal
   

```
$ searchStudent "John Doe" "grades.txt"
John Doe, AA
Saved to records.log.
```

   grades.txt
   - No change

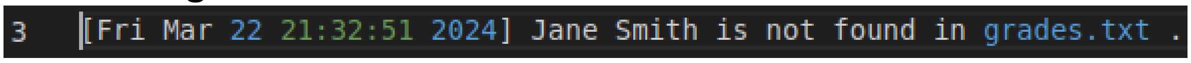
   records.log
   

```
[Fri Mar 22 21:28:04 2024] John Doe, AA is found in grades.txt .
```

3. **Test Case :** Search for Student Grade that doesn't exists
   Input : searchStudent "Jane Smith" "grades.txt"
   Results :
   Terminal
   

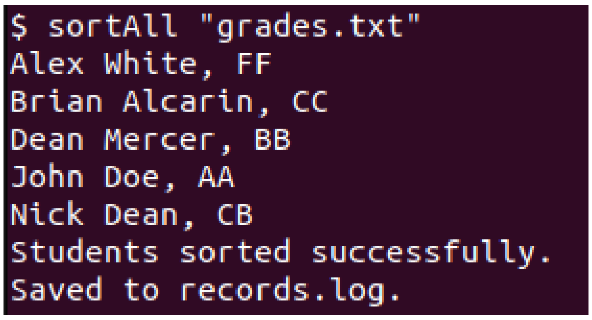
```
$ searchStudent "Jane Smith" "grades.txt"
Error occurred while searching student.
Saved to records.log.
```

   grades.txt
   - No change

   records.log
   

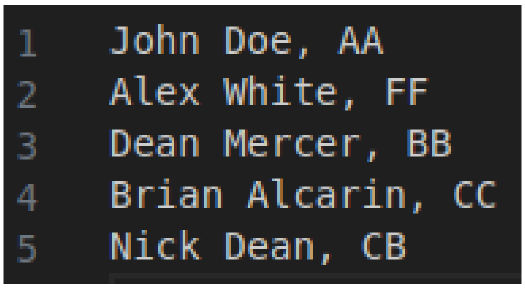
```
[Fri Mar 22 21:32:51 2024] Jane Smith is not found in grades.txt .
```

4. **Test Case :** Sort Student Grades by Name
   Input : sortAll "grades.txt"
   Results :
   Terminal



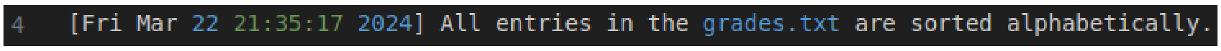
```
$ sortAll "grades.txt"
Alex White, FF
Brian Alcarin, CC
Dean Mercer, BB
John Doe, AA
Nick Dean, CB
Students sorted successfully.
Saved to records.log.
```

   grades.txt
   - Different **grades.txt** is used for testing. Unsorted list:



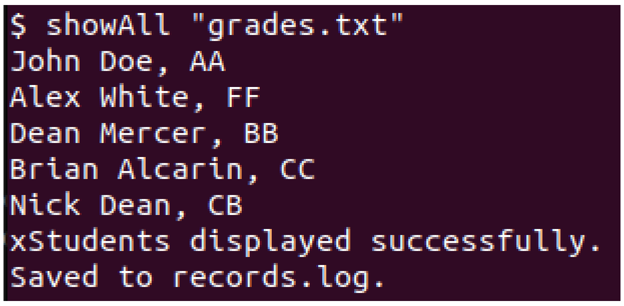
```
John Doe, AA
Alex White, FF
Dean Mercer, BB
Brian Alcarin, CC
Nick Dean, CB
```

   records.log



```
[Fri Mar 22 21:35:17 2024] All entries in the grades.txt are sorted alphabetically.
```

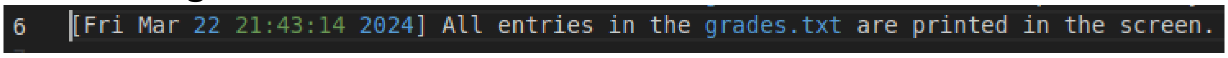
5. **Test Case :** Display All Student Grades
   Input : showAll "grades.txt"
   Results :
   Terminal



```
$ showAll "grades.txt"
John Doe, AA
Alex White, FF
Dean Mercer, BB
Brian Alcarin, CC
Nick Dean, CB
xStudents displayed successfully.
Saved to records.log.
```

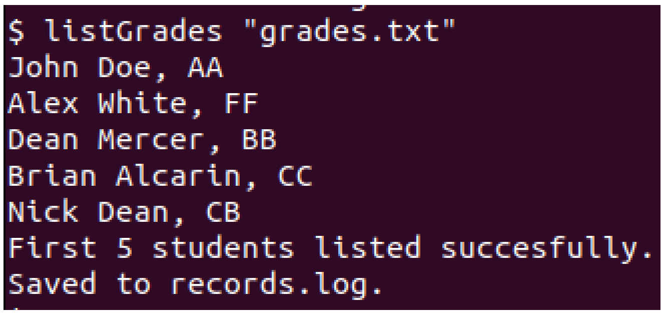
   grades.txt
   - No change

   records.log



```
[Fri Mar 22 21:43:14 2024] All entries in the grades.txt are printed in the screen.
```

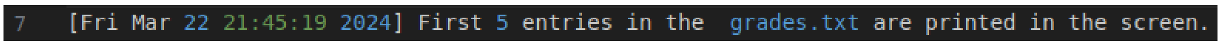
6. **Test Case :** Display First 5 Student Grades
   Input : listGrades "grades.txt"
   Results :
   Terminal



```
$ listGrades "grades.txt"
John Doe, AA
Alex White, FF
Dean Mercer, BB
Brian Alcarin, CC
Nick Dean, CB
First 5 students listed succesfully.
Saved to records.log.
```

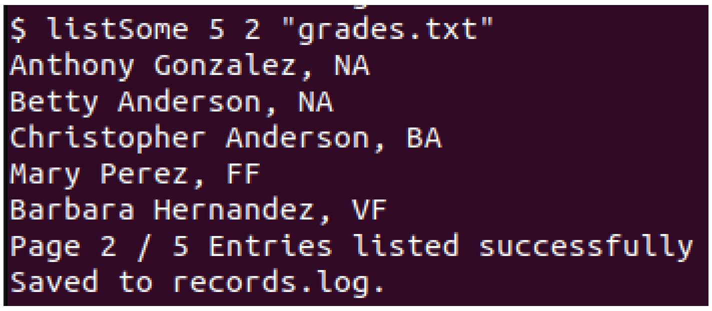
   grades.txt
   - No change

   records.log



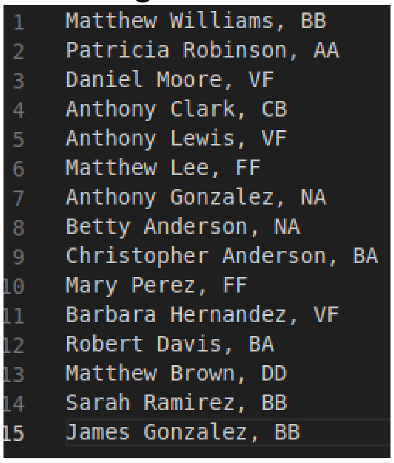
```
[Fri Mar 22 21:45:19 2024] First 5 entries in the  grades.txt are printed in the screen.
```

7. **Test Case :** Display a Range of Student Grades
   Input : listSome 5 2 "grades.txt"
   Results :
   Terminal



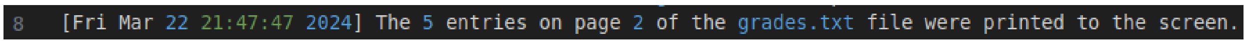
```
$ listSome 5 2 "grades.txt"
Anthony Gonzalez, NA
Betty Anderson, NA
Christopher Anderson, BA
Mary Perez, FF
Barbara Hernandez, VF
Page 2 / 5 Entries listed successfully
Saved to records.log.
```

   grades.txt
   - Different **grades.txt** is used.



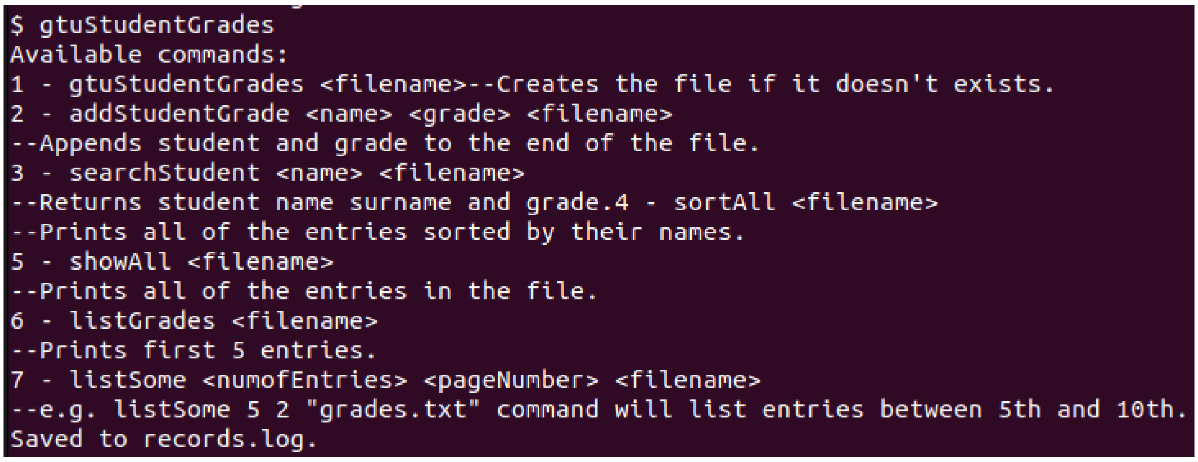
```
Matthew Williams, BB
Patricia Robinson, AA
Daniel Moore, VF
Anthony Clark, CB
Anthony Lewis, VF
Matthew Lee, FF
Anthony Gonzalez, NA
Betty Anderson, NA
Christopher Anderson, BA
Mary Perez, FF
Barbara Hernandez, VF
Robert Davis, BA
Matthew Brown, DD
Sarah Ramirez, BB
James Gonzalez, BB
```

   records.log



```
[Fri Mar 22 21:47:47 2024] The 5 entries on page 2 of the grades.txt file were printed to the screen.
```

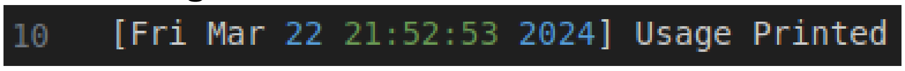
8. **Test Case :** Display Usage Instructions
   Input : gtuStudentGrades
   Results :
   Terminal



```
$ gtuStudentGrades
Available commands:
1 - gtuStudentGrades <filename>--Creates the file if it doesn't exists.
2 - addStudentGrade <name> <grade> <filename>
--Appends student and grade to the end of the file.
3 - searchStudent <name> <filename>
--Returns student name surname and grade.4 - sortAll <filename>
--Prints all of the entries sorted by their names.
5 - showAll <filename>
--Prints all of the entries in the file.
6 - listGrades <filename>
--Prints first 5 entries.
7 - listSome <numofEntries> <pageNumber> <filename>
--e.g. listSome 5 2 "grades.txt" command will list entries between 5th and 10th.
Saved to records.log.
```

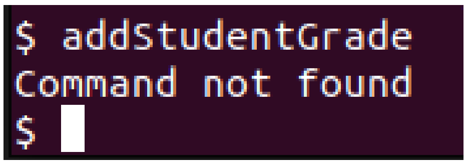
   grades.txt
   - No change

   records.log



```
[Fri Mar 22 21:52:53 2024] Usage Printed
```

9. **Test Case :** Error Handling
   Input : addStudentGrade without parameters
   Results :
   Terminal

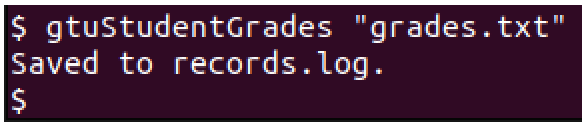


```
$ addStudentGrade
Command not found
$
```

   grades.txt
   - No change

   records.log
   - No change

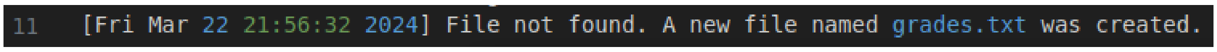
10. **Test Case :** Error Handling
    Input : addStudentGrade without parameters
    Results :
    Terminal



```
$ gtuStudentGrades "grades.txt"
Saved to records.log.
$
```

   grades.txt
   - No change

   records.log



```
[Fri Mar 22 21:56:32 2024] File not found. A new file named grades.txt was created.
```