Gebze Technical University

Department of Computer Engineering

CSE344 System Programming

Homework 1 Report

Burak Yıldırım
1901042609

Contents

1 Introduction

1.1 Project Description

The project focuses on creating a student grade management system in C, highlighting process creation with the **fork()** system call. This system is tasked with managing a file that stores student grades, enabling functionalities to add, search, sort, and display grades. Users interact with the system through commands that allow for the manipulation and retrieval of grade information. Additionally, the system keeps a log file that captures all commands input by the user as well as changes made to grades. To guarantee reliable operation, the system is built to handle a variety of signals that cause a process to terminate.

1.2 Compilation

```
CC = gcc
CFLAGS = -w
DFLAGS = -g
INCDIR = include
SRCDIR = src
TARGET = main
LOG_EXT = log
SOURCES = $(wildcard $(SRCDIR)/*.c) $(wildcard $(SRCDIR)/**/*.c)
HEADERS = $(wildcard $(INCDIR)/*.h) $(wildcard $(INCDIR)/**/*.h)
VALGRIND_OPTIONS = --leak-check=full --show-leak-kinds=all
                   --track-origins=yes --track-fds=yes

all: $(TARGET)

$(TARGET): $(SOURCES) $(HEADERS)
        $(CC) $(CFLAGS) -o $@ $(SOURCES) -I$(INCDIR)

debug: CFLAGS += $(DFLAGS)
debug: $(TARGET)

clean:
        rm -f $(TARGET) *.$(LOG_EXT)

valgrind: debug
        valgrind $(VALGRIND_OPTIONS) ./$(TARGET)
```

The provided Makefile automates the build process: executing **make** compiles the project, **make clean** removes executables and logs, and **make valgrind** launches the program in Valgrind for memory leak and file descriptor checks, streamlining development and debugging.

2 Implementation

2.1 Utility

2.1.1 I/O

2.1.1.1 Macros

- `FILE_PERMS`: Sets the default file permissions to 0666 when creating a new file, allowing read and write operations for the user, group, and others.
- `MAX_INPUT`: Sets the maximum input size to 128 characters, limiting the input buffer in I/O operations.
- `MAX_OUTPUT`: Sets the maximum output size to 256 characters, establishing the buffer size for output operations.
- `OVERWRITE`: Sets the file opening mode to overwrite, using flags `O_WRONLY`, `O_CREAT`, and `O_TRUNC`.
- `READ_ONLY`: Sets the file opening mode to read-only, appending to the file if it exists, using flags `O_RDONLY`, `O_CREAT`, and `O_APPEND`.
- `READ_WRITE`: Sets the file opening mode to both read and write, appending to the file, using flags `O_RDWR`, `O_CREAT`, and `O_APPEND`.
- `WRITE_ONLY`: Sets the file opening mode to write-only, appending to the file, using flags `O_WRONLY`, `O_CREAT`, and `O_APPEND`.

2.1.1.2 Functions

my_fgets

```c
/**
 * Reads a line from a file descriptor into a buffer, handling files and stdin.
 * Logs an error and exits on error.
 *
 * @param buffer The buffer to store the read data.
 * @param size The buffer size including null terminator.
 * @param fd File descriptor for reading.
 * @param filename Name of the file for logging, NULL for stdin.
 * @return Pointer to the buffer with the read line, or NULL on error or EOF.
 */
char *my_fgets(char *buffer, size_t size, int fd, char *filename);
```

file_exists

```c
/**
 * Checks if a file exists.
 *
 * @param filename The name of the file to check.
 * @return 1 if the file exists, 0 otherwise.
 */
int file_exists(const char *filename);
```

my_open

```c
/**
 * Opens a file with specified flags and mode. Logs an error and exits on error.
 *
 * @param filename The name of the file to open.
 * @param flags Flags to control how the file is opened.
 * @param mode Permissions to set if a new file is created.
 * @return The file descriptor for the opened file.
 */
int my_open(const char *filename, int flags, mode_t mode);
```

my_fprintf

```c
/**
 * Formats a string and writes it to the specified file descriptor.
 *
 * @param fd The file descriptor to write the formatted string to.
 * @param format The format string.
 * @param ... Additional arguments for the format string.
 * @return The number of bytes written, or -1 on error.
 */
ssize_t my_fprintf(int fd, const char *format, ...);
```

my_printf

```c
/**
 * Formats a string and writes it to the stdout. Logs an error and exits on error.
 *
 * @param format The format string.
 * @param ... Additional arguments for the format string.
 * @return The number of bytes written.
 */
ssize_t my_printf(const char *format, ...);
```

my_vprintf

```c
/**
 * Formats a string using a va_list and writes it to the stdout. Logs an error
 * and exits on error.
 *
 * @param format The format string.
 * @param args A va_list of arguments for the format string.
 * @return The number of bytes written.
 */
ssize_t my_vprintf(const char *format, va_list args);
```

my_close

```c
/**
 * Closes a file descriptor. Logs an error and exits on error.
 *
 * @param fd The file descriptor to close.
 * @param filename The name of the file associated with the file descriptor for
 *                 logging, or NULL if not applicable.
 */
void my_close(int fd, const char *filename);
```

2.1.2 Log

2.1.2.1 Macros

- `LOG_FNAME_FORMAT`: Specifies the format for log file names using strftime-compatible placeholders, resulting in names like ”2024-03-20T15:30:00.log”.
- `LOG_FNAME_LEN`: Sets the maximum length for log file names to 24 characters, accounting for the format specified.
- `MAX_LOG`: Defines the maximum size of a log message, set to 256 characters.
- `TIMESTAMP_LEN`: Specifies the length of the timestamp string to 64 characters, ensuring it fits the predefined format.

2.1.2.2 Functions

get_timestamp

```
/**
 * Generates a timestamp string based on the given format.
 *
 * @param format The format string for strftime.
 * @return A pointer to the formatted timestamp string.
 */
char *get_timestamp(const char *format);
```

init_log

```
/**
 * Initializes a log file with a timestamped filename, opening and closing
 * it to ensure it's created. Displays error and exits on error.
 */
void init_log();
```

log_message

```
/**
 * Logs a formatted message to a predefined log file. Displays error and exits
 * on error.
 *
 * @param format The format string for the message to log.
 * @param ... Additional arguments for the format string.
 */
void log_message(const char *format, ...);
```

2.1.3 String

2.1.3.1 Functions

my_strsignal

```
/**
 * Converts a signal number to its corresponding signal name string.
 *
 * @param signo The signal number.
 * @return The name of the signal or "Unknown signal" if the signal number is not
 * recognized.
 */
const char *my_strsignal(int signo);
```

my_strdup

```c
/**
 * Duplicates a string, allocating memory for the new string. Logs an error and
 * exits on error.
 *
 * @param str The string to duplicate.
 * @return A pointer to the duplicated string.
 */
char *my_strdup(const char *str);
```

my_strtok

```c
/**
 * Tokenizes a string based on a specified delimiter character.
 *
 * @param str The string to tokenize. Pass NULL to get the next token.
 * @param delim The character delimiter to tokenize the string.
 * @return The next token or NULL if there are no more tokens.
 */
char *my_strtok(char *str, const char delim);
```

ordinal_suffix

```c
/**
 * Determines the ordinal suffix for a given integer.
 *
 * @param num The integer to determine the ordinal suffix for.
 * @return The ordinal suffix as a string ("st", "nd", "rd", "th").
 */
char *ordinal_suffix(int num);
```

trim

```c
/**
 * Trims leading and trailing spaces from a string.
 *
 * @param str The string to be trimmed. If NULL, returns NULL.
 * @return The trimmed string.
 */
char *trim(char *str);
```

is_integer

```
/**
 * Checks if a string represents a valid integer.
 *
 * @param str The string to check.
 * @return 1 if the string is an integer, 0 otherwise.
 */
int is_integer(const char *str);
```

my_strcmp

```
/**
 * Compares two strings lexicographically.
 *
 * @param str1 The first string to compare.
 * @param str2 The second string to compare.
 * @return An integer less than, equal to, or greater than zero if str1 is found,
 *         respectively, to be less than, to match, or be greater than str2.
 */
int my_strcmp(const char *str1, const char *str2);
```

parse_int

```
/**
 * Parses a string and converts it to an integer.
 *
 * @param str The string to convert to an integer.
 * @return The converted integer or INT_MIN if the conversion is not possible.
 */
int parse_int(const char *str);
```

my_sprintf

```
/**
 * Formats a string and stores it into a buffer.
 *
 * @param buffer The buffer to print into.
 * @param size The maximum number of bytes to write, including the null terminator.
 * @param format The format string.
 * @param ... Additional arguments for the format string.
 * @return The number of bytes written.
 */
size_t my_sprintf(char *buffer, size_t size, char *format, ...);
```

my_strlen

```
/**
 * Calculates the length of a string.
 *
 * @param str The string to calculate the length of.
 * @return The length of the string.
 */
size_t my_strlen(const char *str);
```

my_strlen_utf8

```
/**
 * Calculates the length of a UTF-8 encoded string, counting each multi-byte
 * character as a single character.
 *
 * @param str The UTF-8 encoded string to calculate the length of.
 * @return The length of the string in terms of UTF-8 characters.
 */
size_t my_strlen_utf8(const char *str);
```

my_vsprintf

```
/**
 * Formats a string and stores it in a buffer using a va_list.
 *
 * @param buffer The buffer to store the formatted string.
 * @param size The size of the buffer.
 * @param format The format string.
 * @param args A va_list of arguments to format.
 * @return The number of bytes written to the buffer.
 */
size_t my_vsprintf(char *buffer, size_t size, char *format, va_list args);
```

2.1.4 Array

2.1.4.1 Functions

indexof_int

```
/**
 * Searches for an integer value in an array and returns its index.
 *
 * @param arr The array to search.
 * @param size The size of the array.
 * @param value The integer value to search for.
 * @return The index of the first occurrence of the value in the array, or -1 if
 *         not found.
 */
int indexof_int(int *arr, size_t size, int value);
```

indexof_char

```
/**
 * Searches for a character value in an array and returns its index.
 *
 * @param arr The array to search.
 * @param size The size of the array.
 * @param value The character value to search for.
 * @return The index of the first occurrence of the value in the array, or -1 if
 *         not found.
 */
int indexof_char(char *arr, size_t size, char value);
```

indexof_str

```
/**
 * Searches for a string value in an array of strings and returns its index.
 *
 * @param arr The array of strings to search.
 * @param size The size of the array.
 * @param value The string value to search for.
 * @return The index of the first occurrence of the value in the array, or -1 if
 *         not found.
 */
int indexof_str(char **arr, size_t size, const char *value);
```

indexof_struct

```
/**
 * Searches for a value in an array of elements of any type and returns its index.
 *
 * @param arr The array to search.
 * @param size The number of elements in the array.
 * @param elem_size The size of each element in the array.
 * @param value The value to search for.
 * @param cmp A comparison function that takes two void pointers, compares the
 *            pointed-to values, and returns 0 if they are equal.
 * @return The index of the first occurrence of the value in the array, or -1 if
 *         not found.
 */
int indexof_struct(const void *arr, size_t size, size_t elem_size, const void *value,
                   int (*cmp)(const void *, const void *));
```

rm_dups_int

```
/**
 * Removes duplicate integers from an array, maintaining the order of unique
 * elements.
 *
 * @param arr The array of integers to remove duplicates from.
 * @param size The size of the array.
 * @return The new size of the array after removing duplicates.
 */
int rm_dups_int(int *arr, size_t size);
```

rm_dups_char

```
/**
 * Removes duplicate characters from an array, maintaining the order of unique
 * elements.
 *
 * @param arr The array of characters to remove duplicates from.
 * @param size The size of the array.
 * @return The new size of the array after removing duplicates.
 */
int rm_dups_char(char *arr, size_t size);
```

rm_dups_str

```
/**
 * Removes duplicate strings from an array of strings, maintaining the order of
 * unique elements.
 *
 * @param arr The array of string pointers to remove duplicates from.
 * @param size The size of the array.
 * @return The new size of the array after removing duplicates.
 */
int rm_dups_str(char **arr, size_t size);
```

2.1.5 Memory

2.1.5.1 Functions

my_malloc

```
/**
 * Allocates memory of a specified size. Logs error and exits on error.
 *
 * @param size The amount of memory to allocate in bytes.
 * @return A pointer to the allocated memory.
 */
void *my_malloc(size_t size);
```

my_realloc

```
/**
 * Reallocates memory block to a new size. Logs an error and exits on error.
 *
 * @param ptr Pointer to the memory previously allocated with malloc, calloc,
 *            or realloc.
 * @param size New size for the memory block in bytes.
 * @return A pointer to the newly allocated memory.
 */
void *my_realloc(void *ptr, size_t size);
```

2.2 Main

2.2.1 Macros

- `ENTRY_PER_PAGE`: Specifies the number of student entries per page, set to 5.
- `MAX_ARGS`: Defines the maximum number of arguments allowed, set to 3.
- `CMDS_NUM`: Indicates the number of commands available, set to 8.
- `TERM_SIGS_NUM`: Represents the number of termination signals, set to 6.
- `CORNER_LEFT_DOWN`: Unicode character for the bottom-left corner box drawing.
- `CORNER_LEFT_UP`: Unicode character for the top-left corner box drawing.
- `CORNER_RIGHT_DOWN`: Unicode character for the bottom-right corner box drawing.
- `CORNER_RIGHT_UP`: Unicode character for the top-right corner box drawing.
- `CROSS`: Unicode character for the cross box drawing.
- `HORIZONTAL_LINE`: Unicode character for the horizontal line box drawing.
- `T_DOWN`: Unicode character for the T-down box drawing.
- `T_LEFT`: Unicode character for the T-left box drawing.
- `T_RIGHT`: Unicode character for the T-right box drawing.
- `T_UP`: Unicode character for the T-up box drawing.
- `VERTICAL_LINE`: Unicode character for the vertical line box drawing.
- `MY_CLOSE_1` and `MY_CLOSE_2`: Macros for handling file closure with optional filename for logging.
- `my_close`: Macro that selects between `MY_CLOSE_1` and `MY_CLOSE_2` based on the number of arguments.
- `MY_FGETS_3` and `MY_FGETS_4`: Macros for handling file reading with an optional filename for logging.
- `my_fgets`: Macro that chooses between `MY_FGETS_3` and `MY_FGETS_4` based on argument count.

2.2.2 Structs

- `Command`: A structure representing a command, which includes:
  - `const char *name`: A string representing the name of the command.
  - `int args_num`: An integer representing the number of arguments the command accepts.
- `Student`: A structure representing a student, which includes:
  - `const char *fullname`: A string representing the full name of the student.
  - `const char *grade`: A string representing the grade of the student.

2.2.3 Functions

cmp_student_by_grade_asc

```
/**
 * Compares two Student objects by their grades in ascending order.
 *
 * @param student1 Pointer to the first Student object.
 * @param student2 Pointer to the second Student object.
 * @return An integer less than, equal to, or greater than zero if the grade of the
 *         first student is found, respectively, to be less than, to match, or be
 *         greater than the grade of the second student.
 */
int cmp_student_by_grade_asc(const void *student1, const void *student2);
```

cmp_student_by_grade_desc

```
/**
 * Compares two Student objects by their grades in descending order.
 *
 * @param student1 Pointer to the first Student object.
 * @param student2 Pointer to the second Student object.
 * @return An integer less than, equal to, or greater than zero if the grade of the
 *         second student is found, respectively, to be less than, to match, or be
 *         greater than the grade of the first student.
 */
int cmp_student_by_grade_desc(const void *student1, const void *student2);
```

cmp_student_by_name_asc

```
/**
 * Compares two Student objects by their full names in ascending order.
 *
 * @param student1 Pointer to the first Student object.
 * @param student2 Pointer to the second Student object.
 * @return An integer less than, equal to, or greater than zero if the name of the
 *         first student is found, respectively, to be less than, to match, or be
 *         greater than the name of the second student.
 */
int cmp_student_by_name_asc(const void *student1, const void *student2);
```

cmp_student_by_name_desc

```
/**
 * Compares two Student objects by their full names in descending order.
 *
 * @param student1 Pointer to the first Student object.
 * @param student2 Pointer to the second Student object.
 * @return An integer less than, equal to, or greater than zero if the name of the
 *         second student is found, respectively, to be less than, to match, or be
 *         greater than the name of the first student.
 */
int cmp_student_by_name_desc(const void *student1, const void *student2);
```

cmp_student_to_name

```
/**
 * Compares a Student object's full name with a given name string.
 *
 * @param student Pointer to the Student object.
 * @param name Pointer to the name string to compare with the Student's full name.
 * @return An integer less than, equal to, or greater than zero if the Student's
 *         full name is found, respectively, to be less than, to match, or be
 *         greater than the given name string.
 */
int cmp_student_to_name(const void *student, const void *name);
```

get_cmd_args_num

```
/**
 * Retrieves the number of arguments for command instances matching a given
 * command name.
 *
 * @param command The command name to search for.
 * @param commands The array of Command structures to search within.
 * @param size The size of the commands array.
 * @return An array of integers containing the number of arguments for each
 *         matching command, or {-1, -1} if no matches are found. The array
 *         can contain up to two matches.
 */
int *get_cmd_args_num(const char *command, const Command *commands, size_t size);
```

is_valid_cmd

```
/**
 * Checks if a given command name is valid within a set of predefined commands.
 *
 * @param command The command name to check for validity.
 * @param commands The array of Command structures to validate against.
 * @param size The size of the commands array.
 * @return 1 if the command is valid, 0 otherwise.
 */
int is_valid_cmd(const char *command, const Command *commands, size_t size);
```

read_file

```
/**
 * Reads student data from a file and populates a buffer with Student structures,
 * resizing the buffer as needed.
 *
 * @param fd The file descriptor of the file to read from.
 * @param filename The name of the file for logging.
 * @param buffer A pointer to an array of Student pointers to store the data.
 * @param init_size The initial size of the buffer.
 * @param longest_name A pointer to an integer to store the length of the longest
 *                     name encountered.
 * @param start_line The line number to start reading from (inclusive).
 * @param end_line The line number to end reading at (exclusive); use -1 for no limit.
 * @return The number of students read into the buffer.
 */
int read_file(int fd, const char *filename, Student **buffer, int init_size,
              int *longest_name, int start_line, int end_line);
```

add_student

```
/**
 * Adds or updates a student in a file. If the student already exists, their grade
 * is updated; otherwise, the student is added to the file. Logs an error and exits
 * on error.
 *
 * @param fullname The full name of the student.
 * @param grade The grade of the student.
 * @param filename The name of the file where the student data is stored.
 */
void add_student(const char *fullname, const char *grade, const char *filename);
```

cleanup

```
/**
 * Cleans up resources before program termination. If called in the parent process,
 * logs termination. In the child process, it closes file descriptors and frees
 * allocated memory for student data.
 */
void cleanup();
```

create_file

```
/**
 * Creates a new file if it does not already exist.
 *
 * @param filename The name of the file to be created.
 */
void create_file(const char *filename);
```

dispatch_cmd

```
/**
 * Dispatches the command to the corresponding function based on the command name
 * and its arguments.
 *
 * @param command The command to execute.
 * @param args_num The number of arguments passed to the command.
 * @param args The arguments passed to the command.
 */
void dispatch_cmd(const char *command, int args_num, char **args);
```

list_grades

```
/**
 * Lists the grades of students from a file, displaying the first five entries.
 *
 * @param filename The name of the file containing student grades to list.
 */
void list_grades(const char *filename);
```

list_some

```
/**
 * Lists a specific range of student entries from a file, based on the specified
 * number of entries and page number.
 *
 * @param entry_num The number of entries to display.
 * @param page_num The page number to display the entries from.
 * @param filename The name of the file containing student grades.
 */
void list_some(int entry_num, int page_num, const char *filename);
```

print_all_cmds

```
/**
 * Prints all unique commands available in the system.
 *
 * @param commands An array of Command structures containing the command names
 *                 and other details.
 * @param size The size of the commands array.
 */
void print_all_cmds(const Command *commands, size_t size);
```

print_cmd

```
/**
 * Prints the description for a specific command based on the command name.
 *
 * @param command The name of the command to print the description for.
 */
void print_cmd(const char *command);
```

print_students

```
/**
 * Prints a table of students' full names and grades.
 *
 * @param students An array of Student structures to print.
 * @param size The number of students in the array.
 * @param longest_name The length of the longest full name in the array, used for
 *                     formatting.
 */
void print_students(const Student *students, int size, int longest_name);
```

reset_globals

```
/**
 * Resets global variables by freeing allocated memory for student buffer and
 * resetting related globals.
 */
void reset_globals();
```

search_student

```
/**
 * Searches for a student's grade in a file by their full name.
 *
 * @param fullname The full name of the student to search for.
 * @param filename The name of the file to search in.
 */
void search_student(const char *fullname, const char *filename);
```

show_all

```
/**
 * Displays all student entries from a specified file.
 *
 * @param filename The name of the file containing student entries to display.
 */
void show_all(const char *filename);
```

sort_all

```
/**
 * Sorts all student entries from a specified file based on a given field and order,
 * then displays them.
 *
 * @param field The field to sort by ("name" or "grade").
 * @param order The order to sort in (0 for ascending, 1 for descending).
 * @param filename The name of the file containing student entries to sort.
 */
void sort_all(const char *field, int order, const char *filename);
```

term_sigs_handler

```
/**
 * Handles termination signals, setting a flag and exiting the program successfully.
 *
 * @param signo The signal number received.
 */
void term_sigs_handler(int signo);
```

main

```
/**
 * The main function initializes signal handling, processes user commands, and
 * manages child processes. It sets up a command-line interface that allows users to
 * input commands, validates these commands, and dispatches them to the appropriate
 * handlers. It also ensures proper cleanup and logging.
 *
 * - Signal Handling Setup: Registers handlers for termination signals.
 * - Command Processing Loop:
 *   - Prompts the user for input. Arguments must be enclosed in double quotes.
 *   - Tokenizes the input to separate the command from its arguments.
 *   - Validates the command and the number of arguments.
 *   - Creates a child process using fork to handle the command execution.
 *   - Waits for the child process to complete using waitpid.
 * - Command Execution: Calls dispatch_cmd in the child process to execute the
 *                      corresponding function.
 * - Cleanup: Registers a cleanup function with atexit for resource deallocation and
 *            logging upon program termination.
 * - Logging: Logs significant events like program start, signal reception, and
 *            process termination.
 */
 int main();
```

3 Testing

3.1 Printing All the Commands

```
burakyildirim@ubuntu:~/Desktop/Hw1$ make
gcc -w -o main src/main.c src/utils/io.c src/utils/log.c src/utils/memory.c src/utils/string.c src/utils/array.c -Iinclude
burakyildirim@ubuntu:~/Desktop/Hw1$ ./main
Enter (q) to quit.
> Enter a command: gtuStudentGrades

Note: Please ensure all arguments are enclosed in double quotes ("").

Here are the possible commands:
  gtuStudentGrades: Displays a list of all available commands and their descriptions.
  gtuStudentGrades <filename>: Creates a new file with the specified filename for storing student grades.
  addStudentGrade <fullname> <grade> <filename>: Appends the specified student's full name and grade to the designated file.
  searchStudent <fullname> <filename>: Searches for and displays the grade of a student with the given full name in the specified file.
  sortAll <field> <order> <filename>: Sorts and displays all student entries by the specified field in the specified order from the specified file.
  showAll <filename>: Displays all student entries stored in the specified file.
  listGrades <filename>: Displays the first five student entries from the specified file.
  listSome <numOfEntries> <pageNumber> <filename>: Lists a specific range of student entries based on the provided number of entries and page number from the specified file.
```

3.2 Creating a File

```
> Enter a command: gtuStudentGrades "test.txt"
File created successfully.
```

3.3 Creating a File with An Extra Argument

```
> Enter a command: gtuStudentGrades "file.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  gtuStudentGrades: Displays a list of all available commands and their descriptions.
  gtuStudentGrades <filename>: Creates a new file with the specified filename for storing student grades.
```

3.4 Adding a Student

```
> Enter a command: addStudentGrade "Burak Yıldırım" "AA" "test.txt"
Student added successfully.
```

3.5 Updating a Student

```
> Enter a command: addStudentGrade "Burak Yıldırım" "BB" "test.txt"
Student updated successfully.
```

```
> Enter a command: showAll "test.txt"

| Full Name | Grade |
|---|---|
| Burak Yıldırım | BB |
```

3.6 Adding a Student with a Missing Argument

```
> Enter a command: addStudentGrade "Burak Yıldırım" "AA"
Invalid number of arguments. Here are the possible options for the command:
  addStudentGrade <fullname> <grade> <filename>: Appends the specified student's full name and grade to the designated file.
```

3.7 Adding a Student with an Extra Argument

```
> Enter a command: addStudentGrade "Burak Yıldırım" "AA" "test.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  addStudentGrade <fullname> <grade> <filename>: Appends the specified student's full name and grade to the designated file.
```

3.8 Searching for a Student

```
> Enter a command: searchStudent "Burak Yıldırım" "test.txt"
Grade of Burak Yıldırım is BB.
```

3.9 Searching for a Student in a Non-Existing File

```
> Enter a command: searchStudent "Burak Yıldırım" "nofile.txt"
File not found.
```

3.10 Searching for a Non-Existing Student

```
> Enter a command: searchStudent "No Student" "test.txt"
Student not found.
```

3.11 Searching for a Student with a Missing Argument

```
> Enter a command: searchStudent "Burak Yıldırım"
Invalid number of arguments. Here are the possible options for the command:
  searchStudent <fullname> <filename>: Searches for and displays the grade of a student with the given full name in the specified file.
```

3.12 Searching for a Student with an Extra Argument

```
> Enter a command: searchStudent "Burak Yıldırım" "test.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  searchStudent <fullname> <filename>: Searches for and displays the grade of a student with the given full name in the specified file.
```

3.13 Sorting Students by Name

Ascending

```
> Enter a command: sortAll "name" "asc" "grades_short.txt"
```

Full Name	Grade
Anthony Clark	CB
Anthony Gonzalez	NA
Anthony Lewis	VF
Barbara Garcia	BA
Barbara Hernandez	VF
Barbara Martin	VF
Betty Anderson	NA
Charles Ramirez	CC
Christopher Anderson	BA
Christopher Robinson	NA
Daniel Moore	VF
James Gonzalez	BB
James Sanchez	DD
Jessica Brown	VF
Jessica Martin	CC
Jessica Williams	AA
John Thomas	FF
Joseph Jones	BA
Joseph Rodriguez	DC
Karen Taylor	AA
Lisa Harris	BB
Margaret Anderson	AA
Mary Martinez	DD
Mary Perez	FF
Matthew Brown	DD
Matthew Lee	FF
Matthew Moore	CB
Matthew Williams	CC
Michael Jackson	BB
Michael Perez	CC
Nancy Harris	VF
Patricia Robinson	AA
Richard Thompson	AA
Robert Davis	BA
Sandra Perez	BB
Sarah Ramirez	BB
Sarah Taylor	DC
Sarah Thompson	AA
William Jones	BA
William Wilson	DD

Descending

```
> Enter a command: sortAll "name" "desc" "grades_short.txt"
```

Full Name	Grade
William Wilson	DD
William Jones	BA
Sarah Thompson	AA
Sarah Taylor	DC
Sarah Ramirez	BB
Sandra Perez	BB
Robert Davis	BA
Richard Thompson	AA
Patricia Robinson	AA
Nancy Harris	VF
Michael Perez	CC
Michael Jackson	BB
Matthew Williams	CC
Matthew Moore	CB
Matthew Lee	FF
Matthew Brown	DD
Mary Perez	FF
Mary Martinez	DD
Margaret Anderson	AA
Lisa Harris	BB
Karen Taylor	AA
Joseph Rodriguez	DC
Joseph Jones	BA
John Thomas	FF
Jessica Williams	AA
Jessica Martin	CC
Jessica Brown	VF
James Sanchez	DD
James Gonzalez	BB
Daniel Moore	VF
Christopher Robinson	NA
Christopher Anderson	BA
Charles Ramirez	CC
Betty Anderson	NA
Barbara Martin	VF
Barbara Hernandez	VF
Barbara Garcia	BA
Anthony Lewis	VF
Anthony Gonzalez	NA
Anthony Clark	CB

3.14 Sorting Students by Grade

Ascending

```
> Enter a command: sortAll "grade" "asc" "grades_short.txt"
```

Full Name	Grade
Patricia Robinson	AA
Karen Taylor	AA
Sarah Thompson	AA
Jessica Williams	AA
Margaret Anderson	AA
Richard Thompson	AA
Christopher Anderson	BA
Robert Davis	BA
Barbara Garcia	BA
William Jones	BA
Joseph Jones	BA
Sarah Ramirez	BB
James Gonzalez	BB
Sandra Perez	BB
Michael Jackson	BB
Lisa Harris	BB
Anthony Clark	CB
Matthew Moore	CB
Matthew Williams	CC
Charles Ramirez	CC
Jessica Martin	CC
Michael Perez	CC
Joseph Rodriguez	DC
Sarah Taylor	DC
Matthew Brown	DD
James Sanchez	DD
Mary Martinez	DD
William Wilson	DD
Matthew Lee	FF
Mary Perez	FF
John Thomas	FF
Anthony Gonzalez	NA
Betty Anderson	NA
Christopher Robinson	NA
Daniel Moore	VF
Anthony Lewis	VF
Barbara Hernandez	VF
Jessica Brown	VF
Barbara Martin	VF
Nancy Harris	VF

Descending

```
> Enter a command: sortAll "grade" "desc" "grades_short.txt"
```

Full Name	Grade
Daniel Moore	VF
Anthony Lewis	VF
Barbara Hernandez	VF
Jessica Brown	VF
Barbara Martin	VF
Nancy Harris	VF
Anthony Gonzalez	NA
Betty Anderson	NA
Christopher Robinson	NA
Matthew Lee	FF
Mary Perez	FF
John Thomas	FF
Matthew Brown	DD
James Sanchez	DD
Mary Martinez	DD
William Wilson	DD
Joseph Rodriguez	DC
Sarah Taylor	DC
Matthew Williams	CC
Charles Ramirez	CC
Jessica Martin	CC
Michael Perez	CC
Anthony Clark	CB
Matthew Moore	CB
Sarah Ramirez	BB
James Gonzalez	BB
Sandra Perez	BB
Michael Jackson	BB
Lisa Harris	BB
Christopher Anderson	BA
Robert Davis	BA
Barbara Garcia	BA
William Jones	BA
Joseph Jones	BA
Patricia Robinson	AA
Karen Taylor	AA
Sarah Thompson	AA
Jessica Williams	AA
Margaret Anderson	AA
Richard Thompson	AA

3.15 Sorting Students from a Non-Existing File

```
> Enter a command: sortAll "name" "asc" "nofile.txt"
File not found.
```

3.16 Sorting Students with an Invalid Field Argument

```
> Enter a command: sortAll "something" "asc" "grades_short.txt"
Invalid argument. Please enter either "name" or "grade" as the first argument.
```

3.17 Sorting Students with an Invalid Order Argument

```
> Enter a command: sortAll "name" "something" "grades_short.txt"
Invalid argument. Please enter either "asc" or "desc" as the second argument.
```

3.18 Sorting Students with a Missing Argument

```
> Enter a command: sortAll "name" "asc"
Invalid number of arguments. Here are the possible options for the command:
  sortAll <field> <order> <filename>: Sorts and displays all student entries by the specified field in the specified order from the specified file.
```

3.19 Sorting Students with an Extra Argument

```
> Enter a command: sortAll "name" "asc" "grades_short.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  sortAll <field> <order> <filename>: Sorts and displays all student entries by the specified field in the specified order from the specified file.
```

3.20 Displaying All Students

```
> Enter a command: showAll "grades_short.txt"

| Full Name           | Grade |
|---------------------|-------|
| Matthew Williams    | CC    |
| 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    |
| Matthew Moore       | CB    |
| Barbara Garcia      | BA    |
| Joseph Rodriguez    | DC    |
| James Sanchez       | DD    |
| Karen Taylor        | AA    |
| Charles Ramirez     | CC    |
| Sarah Thompson      | AA    |
| Jessica Martin      | CC    |
| William Jones       | BA    |
| Mary Martinez       | DD    |
| Jessica Brown       | VF    |
| Barbara Martin      | VF    |
| Nancy Harris        | VF    |
| Michael Perez       | CC    |
| Joseph Jones        | BA    |
| Sarah Taylor        | DC    |
| Christopher Robinson| NA    |
| Jessica Williams    | AA    |
| Sandra Perez        | BB    |
| Michael Jackson     | BB    |
| William Wilson      | DD    |
| Margaret Anderson   | AA    |
| Richard Thompson    | AA    |
| John Thomas         | FF    |
| Lisa Harris         | BB    |
```

3.21 Displaying All Students with a Missing Argument

```
> Enter a command: showAll
Invalid number of arguments. Here are the possible options for the command:
  showAll <filename>: Displays all student entries stored in the specified file.
```

3.22 Displaying All Students with an Extra Argument

```
> Enter a command: showAll "grades_short.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  showAll <filename>: Displays all student entries stored in the specified file.
```

3.23 Displaying the First Five Students

```
> Enter a command: listGrades "grades_short.txt"

| Full Name          | Grade |
|--------------------|-------|
| Matthew Williams   | CC    |
| Patricia Robinson  | AA    |
| Daniel Moore       | VF    |
| Anthony Clark      | CB    |
| Anthony Lewis      | VF    |
```

3.24 Displaying the First Five Students from a Non-Existing File

```
> Enter a command: listGrades "nofile.txt"
File not found.
```

3.25 Displaying the First Five Students with a Missing Argument

```
> Enter a command: listGrades
Invalid number of arguments. Here are the possible options for the command:
  listGrades <filename>: Displays the first five student entries from the specified file.
```

3.26 Displaying the First Five Students with an Extra Argument

```
> Enter a command: listGrades "grades_short.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  listGrades <filename>: Displays the first five student entries from the specified file.
```

3.27 Displaying Some Students Starting from Some Page

```
> Enter a command: listSome "5" "2" "grades_short.txt"

| Full Name           | Grade |
|---------------------|-------|
| Matthew Lee         | FF    |
| Anthony Gonzalez    | NA    |
| Betty Anderson      | NA    |
| Christopher Anderson| BA    |
| Mary Perez          | FF    |
```

3.28 Displaying Some Students Starting from a Non-Integer Page

```
> Enter a command: listSome "no-int" "no-int" "test.txt"
Invalid arguments. Please enter valid numbers.
```

3.29 Displaying Some Students Starting from a Negative Page

```
> Enter a command: listSome "-1" "-1" "test.txt"
Invalid arguments. Please enter positive numbers.
```

3.30 Displaying Some Students with a Missing Argument

```
> Enter a command: listSome "5" "2"
Invalid number of arguments. Here are the possible options for the command:
  listSome <numOfEntries> <pageNumber> <filename>: Lists a specific range of student entries based on the provided number of entries and page number from the specified file.
```

3.31 Displaying Some Students with an Extra Argument

```
> Enter a command: listSome "5" "2" "grades_short.txt" "extra-arg"
Invalid number of arguments. Here are the possible options for the command:
  listSome <numOfEntries> <pageNumber> <filename>: Lists a specific range of student entries based on the provided number of entries and page number from the specified file.
```

3.32 Displaying an Empty File

```
> Enter a command: gtuStudentGrades "test2.txt"
File created successfully.

> Enter a command: showAll "test2.txt"
No students found.

> Enter a command: listGrades "test2.txt"
No students found.

> Enter a command: sortAll "name" "asc" "test2.txt"
No students found.
```

3.33 Signal Handling

```
> Enter a command: ^C
Program terminated.
```

3.34 User Quitting

```
Enter (q) to quit.
> Enter a command: q

Program terminated.
```

4 Logs

```
[21:54:20] Program started with PID 7742.
[21:54:43] Created a child process with PID 7746 for the command "gtuStudentGrades".
[21:54:43] Displayed all available commands in the child process with PID 7746.
[21:54:43] Child process with PID 7746 terminated with status 0.
[21:55:22] Created a child process with PID 7749 for the command "gtuStudentGrades 'test.txt'".
[21:55:22] Created the file "test.txt" in the child process with PID 7749.
[21:55:22] Child process with PID 7749 terminated with status 0.
[21:55:57] Created a child process with PID 7752 for the command "addStudentGrade 'Burak Yıldırım' 'AA' 'test.txt'".
[21:55:57] Added the student "Burak Yıldırım" with grade "AA" to the file "test.txt" in the child process with PID 7752.
[21:55:57] Child process with PID 7752 terminated with status 0.
[21:56:19] Created a child process with PID 7754 for the command "addStudentGrade 'Burak Yıldırım' 'BB' 'test.txt'".
[21:56:19] Updated the grade of the student "Burak Yıldırım" to "BB" in the file "test.txt" in the child process with PID 7754.
[21:56:19] Child process with PID 7754 terminated with status 0.
[21:56:44] Created a child process with PID 7756 for the command "searchStudent 'Burak Yıldırım' 'test.txt'".
[21:56:44] Searched for the student "Burak Yıldırım" in the file "test.txt" in the child process with PID 7756.
[21:56:44] Child process with PID 7756 terminated with status 0.
[21:58:17] Created a child process with PID 7780 for the command "sortAll 'name' 'asc' 'grades.txt'".
[21:58:17] Sorted all student entries from the file "grades.txt" in the child process with PID 7780.
[21:58:17] Child process with PID 7780 terminated with status 0.
[22:01:56] Created a child process with PID 7795 for the command "sortAll 'name' 'desc' 'grades.txt'".
[22:01:56] Sorted all student entries from the file "grades.txt" in the child process with PID 7795.
[22:01:56] Child process with PID 7795 terminated with status 0.
[22:02:10] Created a child process with PID 7797 for the command "sortAll 'grade' 'asc' 'grades.txt'".
[22:02:10] Sorted all student entries from the file "grades.txt" in the child process with PID 7797.
[22:02:10] Child process with PID 7797 terminated with status 0.
[22:02:29] Created a child process with PID 7798 for the command "sortAll 'grade' 'desc' 'grades.txt'".
[22:02:29] Sorted all student entries from the file "grades.txt" in the child process with PID 7798.
[22:02:29] Child process with PID 7798 terminated with status 0.
[22:02:49] Created a child process with PID 7800 for the command "showAll 'grades.txt'".
[22:02:49] Displayed all student entries from the file "grades.txt" in the child process with PID 7800.
[22:02:49] Child process with PID 7800 terminated with status 0.
[22:03:10] Created a child process with PID 7803 for the command "listGrades 'grades.txt'".
[22:03:10] Displayed the first 5 student entries from the file "grades.txt" in the child process with PID 7803.
[22:03:10] Child process with PID 7803 terminated with status 0.
[22:03:46] Created a child process with PID 7806 for the command "listSome '5' '2' 'grades.txt'".
[22:03:46] Displayed the student entries from 6th to 10th from the file "grades.txt" in the child process with PID 7806.
[22:03:46] Child process with PID 7806 terminated with status 0.
[22:04:23] Encountered SIGINT. Initiating shutdown.
[22:04:23] Program terminated.
```