

## 1 Overview

### 1.1 user interface program

For this project you will write a text-based user interface to the document manager system you implemented in project #2. In addition, you will add some extra functionality to your system. There are two deadlines associated with the project. Those deadlines are:

- Mon Mar 12, 8:00PM - Your code must pass the first two public tests (public01, public02). That is the only requirement for this deadline. We will not grade the code for style. This first part is worth .5% of your course grade (NOT .5% of this project grade). Notice you can still submit late for this part.
- Thu Mar 15, 8:00PM - Final deadline for the project. Notice you can still submit late (as usual).

Remember that you need to satisfy the good faith attempt for every project in order to pass the class (see syllabus). The good faith attempt information for this project (e.g., requirements and deadline) will be posted on the class web page later on.

## 2 Objectives

To practice text parsing and file I/O.

### 2.1 Obtain the project files

To obtain the project files copy the folder project3 available in the 216 public directory to your 216 directory. Keep in mind that the Makefile and document.h files for this project are different from the ones used in project2.

### 2.2 Fixing problems with your project #2 code

After the late deadline for project2 you will be able to see results for release/secret tests in the submit server. A TA during office hours (and only during office hours) will be able to show you any test and why the test failed (if that is the case). You are responsible for fixing your code before submitting this project. Keep in mind that if you passed all the project2 tests that does not mean you don't have bugs. In this project we will be testing your document functions again so it is in your best interest to test your code thoroughly.

## 3 Specification

### 3.1 Document manager update

You need to add two functions to your document manager system. Remember to use the provided document.h file (not the one from project #2).

1. `int load_file(Document *doc, const char *filename)` - This function is similar to `load_document`, except data will be loaded from a file instead of using an array. By default a paragraph will be added and any blank lines (line with only spaces as defined by `isspace()`) will mark the beginning of a new paragraph. The function will fail and return `FAILURE` if `doc` is `NULL`, `filename` is `NULL`, or if opening the file failed; otherwise the function will return `SUCCESS`. Notice no error message will be generated if the file cannot be opened.
2. `int save_document(Document *doc, const char *filename)` - This function will print the paragraphs associated with a document to the specified file (overwriting the file). Each paragraph will be separated

by a newline. The function will fail and return FAILURE if doc is NULL, filename is NULL, or the file cannot be opened; otherwise the function will return SUCCESS. Notice no error message will be generated if the file cannot be opened.

## 3.2 Method of operation

Your program will be in a file named `user_interface.c`. A user calls your program in one of two ways (assuming the executable is named `user_interface`):

```
user_interface
user_interface filename
```

The program should have zero or one arguments (in addition to the executable name) on the command line; if there are more the program prints the following usage message to standard error, and exits with exit code `EX_USAGE`<sup>1</sup>.

```
Usage: user_interface
Usage: user_interface <filename>
```

If there is no file specified when the program is started, the program should read its data from standard input. The program will display a prompt (represented by `>`) after which commands will be entered. If a file is named, however, the program reads its data from that file; in this case no prompt will be used.

In case of an error opening the file your program should print (to standard error) the message "FILENAME cannot be opened." where FILENAME represents the file name. The program will then exit with the exit code `EX_OSERR`.

Upon starting execution your program should initialize a single document with the name "main.document", and perform operations on that document as instructed by the commands the program reads.

Make sure you name the file with your program `user_interface.c`. This program will include `document.h` (the version provided for this project and not the one from project #2).

## 3.3 File format

### 3.3.1 Valid Lines

An input file (or input coming from standard input) contains multiple lines with commands, and the commands are executed in the order they are encountered. No valid line can be more than 1024 characters (including the newline character). A valid line takes one of three forms:

1. a comment, where the first non-whitespace character is a hash symbol (`#`)
2. a command, where the line is composed of one or more strings of non-whitespace characters
3. a blank line, where the line contains 1 or more spaces (as defined by the `isspace()` function in `ctype.h`)

For example, the following file contains valid lines:

```
# creating a paragraph and inserting some lines
add_paragraph_after 0
add_line_after 1 0 *first line of the document

add_line_after 1 1 *second line of the document
  # let's print it
print_document
quit
```

---

<sup>1</sup>This and the other exit codes beginning with `EX_` mentioned here are all obtained by including `<sys/exit.h>` in your C program file.

Valid commands must follow one of the formats specified in Section 3.4 below.

### 3.3.2 Invalid Lines/Commands

If your program encounters an invalid line it should print the message "Invalid Command". An invalid line includes not only an invalid command, but a command without the expected values. For example, the `add_paragraph_after` command requires an integer. If the value provided is not an integer the command will be considered invalid and the message "Invalid Command" will be generated. Notice the program will not end when an invalid command is provided. Make sure you print to the standard output and not to the standard error.

## 3.4 Commands

Unless output is associated with a command the successful execution of a command will not generate any confirmation message (similar to successful execution of commands in Unix). If a command cannot be successfully executed the message "COMMAND\_NAME failed", where COMMAND\_NAME represents the command, should be printed. Make sure you print to the standard output and not to the standard error.

Any number of spaces can appear between the different elements of a command, and before and after a command. A blank line (as defined above) and a comment will be ignored (no processing). When a comment or blank line is provided, and standard input is being used, a new prompt will be generated.

Although the quit and exit commands will end/terminate the command processor, the command processor should terminate when end of file is seen. A set of commands in a file do not need quit nor exit.

#### 1. `add_paragraph_after` PARAGRAPH\_NUMBER

This command will add a paragraph to the document. The "Invalid Command" message will be generated when:

- a. PARAGRAPH\_NUMBER does not represent a number
- b. PARAGRAPH\_NUMBER is a negative value
- c. PARAGRAPH\_NUMBER is missing
- d. Additional information is provided after the PARAGRAPH\_NUMBER

If the command cannot be successfully executed the message "add\_paragraph\_after failed" will be generated.

#### 2. `add_line_after` PARAGRAPH\_NUMBER LINE\_NUMBER \* LINE

This command will add a line after the line with the specified line number. The line to add will appear after the \* character. The "Invalid Command" message will be generated when:

- a. PARAGRAPH\_NUMBER does not represent a number
- b. PARAGRAPH\_NUMBER is a negative value or 0
- c. PARAGRAPH\_NUMBER is missing
- d. LINE\_NUMBER does not represent a number
- e. LINE\_NUMBER is a negative value
- f. LINE\_NUMBER is missing
- g. \* is missing

If the command cannot be successfully executed the message "add\_line\_after failed" will be generated.

#### 3. `print_document`

This command will print the document information (`print_document` function output). The "Invalid Command" message will be generated if any data appears after `print_document`.

4. `quit`

This command will exit the user interface. The "Invalid Command" message will be generated when any data appears after `quit`.

5. `exit`

This command will exit the user interface. The "Invalid Command" message will be generated when any data appears after `exit`.

6. `append_line PARAGRAPH_NUMBER * LINE`

This command will append a line to the specified paragraph. The line to add will appear after the `*` character. The "Invalid Command" message will be generated when:

- a. `PARAGRAPH_NUMBER` does not represent a number
- b. `PARAGRAPH_NUMBER` is a negative value or 0
- c. `PARAGRAPH_NUMBER` is missing
- d. `*` is missing

If the command cannot be successfully executed the message "append\_line failed" will be generated.

7. `remove_line PARAGRAPH_NUMBER LINE_NUMBER`

This command will remove the specified line from the paragraph. The "Invalid Command" message will be generated when:

- a. `PARAGRAPH_NUMBER` does not represent a number
- b. `PARAGRAPH_NUMBER` is a negative value or 0
- c. `PARAGRAPH_NUMBER` is missing
- d. `LINE_NUMBER` does not represent a number
- e. `LINE_NUMBER` is a negative value or 0
- f. `LINE_NUMBER` is missing
- g. Any data appears after the line number

If the command cannot be successfully executed the message "remove\_line failed" will be generated.

8. `load_file FILENAME`

This command will load the specified file into the current document. The "Invalid Command" message will be generated when:

- a. `FILENAME` is missing
- b. Any data appears after `FILENAME`

If the command cannot be successfully executed the message "load\_file failed" will be generated.

9. `replace_text "TARGET" "REPLACEMENT"`

This command will replace the string "TARGET" with "REPLACEMENT". The "Invalid Command" message will be generated when:

- a. Both "TARGET" and "REPLACEMENT" are missing
- b. Only "TARGET" is provided

For this command you can assume that if "TARGET" and "REPLACEMENT" are present there is no additional data after "REPLACEMENT".

If the command cannot be successfully executed the message "replace.text failed" will be generated.

10. `highlight_text "TARGET"`

This command will highlight the string "TARGET". The "Invalid Command" message will be generated when "TARGET" is missing.

For this command you can assume that if "TARGET" is present there is no additional data after it. Notice no fail message is associated with this command; either the text was highlighted or not.

11. `remove_text "TARGET"`

This command will remove the string "TARGET". The "Invalid Command" message will be generated when "TARGET" is missing.

For this command you can assume that if "TARGET" is present there is no additional data after it. Notice no fail message is associated with this command; either a deletion took place or not.

12. `save_document FILENAME`

This command will save the current document to the specified file. The "Invalid Command" message will be generated when:

- a. FILENAME is missing.
- b. Any data appears after the filename.

If the command cannot be successfully executed the message "save\_document failed" will be generated.

13. `reset_document`

This command will reset the current document. The "Invalid Command" message will be generated when any data appears after `reset_document`. Notice no fail message will be associated with `reset_document`.

### 3.5 Important Points and Hints

1. Data should only be allocated statically. You may not use `malloc()` etc.
2. Do not use `perror` to generate error messages; use `fprintf` and `stderr` instead.
3. IMPORTANT: You may not use `memset`, `strtok`, `strtok_r`, `memcpy` in this project.
4. IMPORTANT. You may not use regular expressions for this project; if you do you will lose at least 30 points. Regular expressions can be used in the format string of a `scanf` statement in order to recognize string patterns. The following are characters typically associated with regular expressions:

[,],\*,^,-,\$,?

5. Do not include `.c` files using `#include`.
6. You can assume a filename will not exceed 80 characters.
7. If you remove a line that line should not be printed (no blank line for it).
8. If you remove all the lines from a paragraph, the paragraph will not be removed (paragraph count will stay the same).

## 4 Grading Criteria

Your project grade will be determined with the following weights:

Results of public tests	20%
Results of release tests	45%
Results of secret tests	25%
Code style grading	10%

## 4.1 Style grading

For this project, your code is expected to conform to the following style guidelines:

- Your code must have a comment at the beginning with your name, university ID number, and UMD Directory ID (i.e., your username on Grace).
- Do not use global variables.
- Feel free to use helper functions for this project; just make sure to define them as static.
- Follow the C style guidelines available at:

<http://www.cs.umd.edu/~nelson/classes/resources/cstyleguide/>

## 5 Submission

You can submit your project by executing in your project directory the **submit** command.

## 6 Academic Integrity

Please see the syllabus for project rules and academic integrity information. All programming assignments in this course are to be written individually (unless explicitly indicated otherwise in a written project handout). Cooperation between students is a violation of the Code of Academic Integrity.