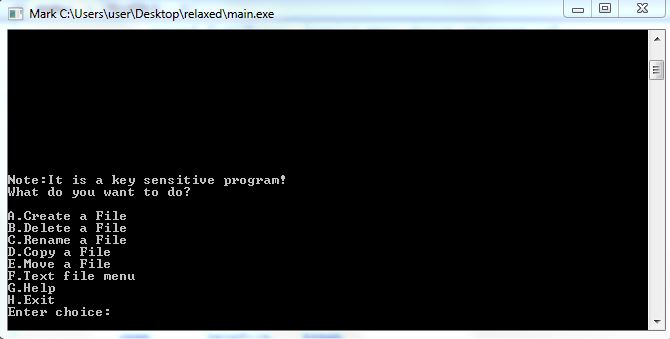
**Introduction:**

In this report, we provide with a full illustration of the project and how it was approached to the final workable program. Portable File Manipulator, PFM is a program which does multiple tasks when a user runs it. It provides different file organizational functionalities such as creating, deleting, or renaming different kinds of files. The organization of information may involve access, updates and movement of information between devices.

In addition to the organizational functionalities, the program provides the accessibility and flexibility to edit text files. Not only that, an organized interactive instruction screen was made to make it easier for the user to utilize. In fact, the program is a key sensitive. To illustrate, when the user runs it, he has to choose between different tasks by entering only a letter as it shows below:



So the user has to enter A, B, C, D, E, F, G or H. If there is an input mistake, the program returns nothing and returns to the main menu as above. The program clears the screen, of course without using a system call, once the user chooses one of the options and start working in the functions as they will be all described later.

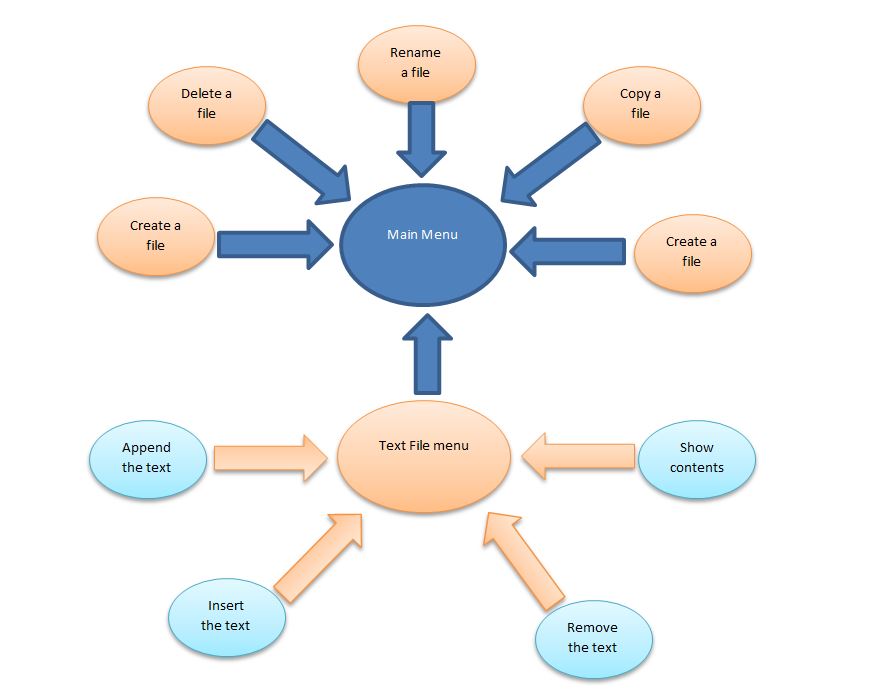
PFM works in any operating system either in Windows and/or iOS. It was intended to avoid any system calls which might arise any errors in our program. Therefore, the following report illustrates all of the design and implementation. In addition; it shows the user decision making through the process of running the program.

**Contents Page:**

1. The Main menu.
2. Create a file.
3. Delete a file.
4. Rename a file.
5. Copy a file.
6. Move a file.
7. Editing text files:
8. Append text to the file.
9. Insert text in a specific position.
10. Remove all text in a file.
11. Show the content of the text file.
12. **The main menu:**

The main functionality provides a flexible interactive phase between the functions’ headers and the user input. All of the functions shall return to the main menu in both cases of conducting the function or to the user’s input failure. In either cases, the program makes sure that memory management goes flawlessly. We used all the memory location options in order to avoid any external segmentation in each function. We used for example char \*file\_name = malloc(sizeof(\*file\_name)); for the dynamical memory allocation in order to avoid too much allocation by using char file\_name[50]; . Plus, we had to free the memory once we don’t use that allocation anymore by using free(file\_name); . All the functions provide the accessibility to return to the main menu in case any unexpected failures occur by entering “R” which means returning to the main menu. In addition, the program covers any input mistakes as it goes in an infinite loop until the user enters the letter “R”.

The following chart shows how each of the functions’ headers are connected to the main function:

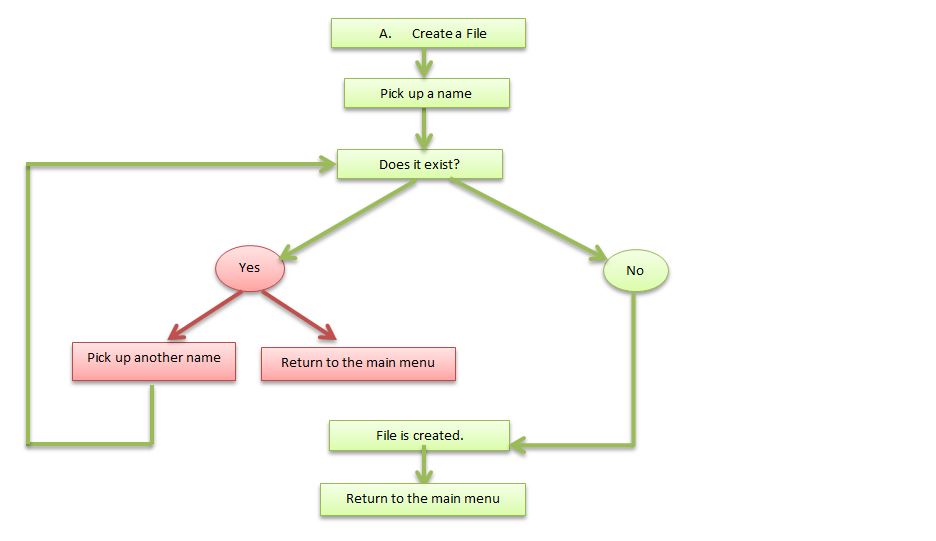


1. **Create a file:**

This function allows the user to create new file. The type the file is flexible to the user as there are various kinds of file types such as pdf, mp3, txt etc. But the function doesn’t create a directory unfortunately as we were considering it not a file type. However, before creating the file, it checks whether that file exists in the directory or not using

(access(buffer1, 0) == -1);

Then the program creates the file in the current working directory where the program is running. The current working directory will be print to the user screen in order to provide him/her the full information where his file will be created. There are different constraints the program should avoid which are the following:

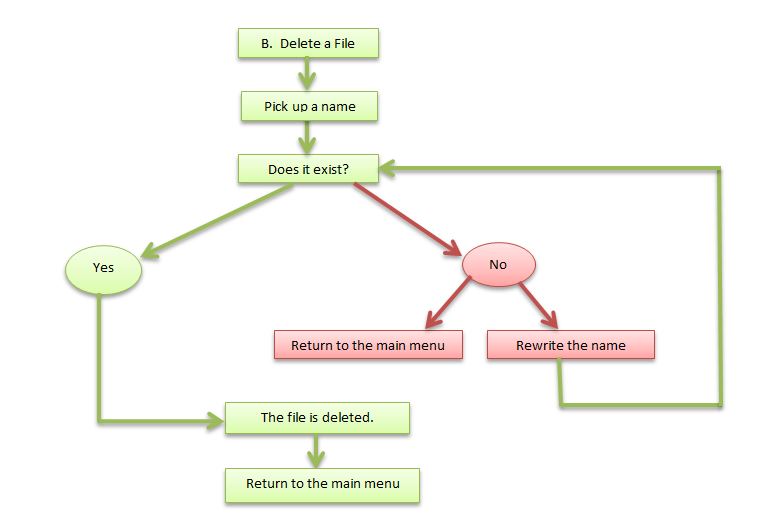


1. **Delete a file:**

This function allows the user to delete the file. The type the file is flexible as well it can be mp3, pdf, txt etc. The function checks the existence of the file using as well (access(buffer1, 0) == -1); , we used 0 instead of F\_OK as F\_OK returns with an error in iOS system.

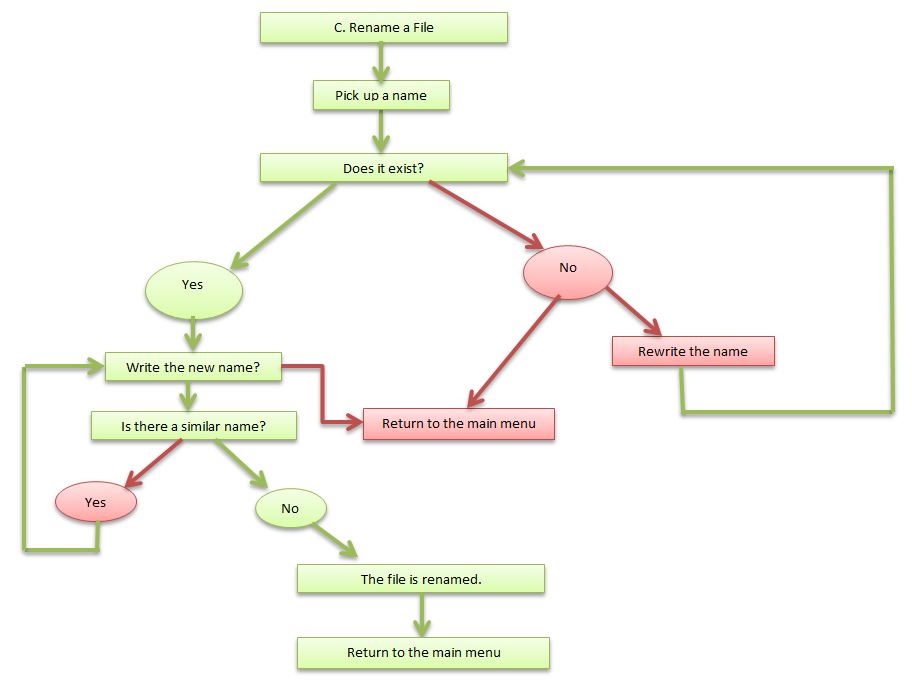
|  |  |
| --- | --- |
| F\_OK or 0 | Check for existence |

Then the program deletes the file in the current working directory where the program is running. The current working directory will be print to the user screen along with the files inside the directory in order to provide him/her the full information which file he/she desires to delete. There are different constraints the program should avoid which are the following:



1. **Rename a file:**

This function allows the user to rename the file using rename(old\_name,new\_name) function call. The type the file is flexible as well it can be mp3, pdf, txt etc. The function is able to rename a directory as well. The program renames the file in the current working directory which is shown by the program when it is running. The files inside the directory are shown as well in order to provide him/her the full information which file he/she desires to rename. There are different constraints the program should avoid which are the following:



1. **Copy a file:**

This function allows the user to copy the file by using the following function calls:

file1 = fopen(buffer1,"rb");

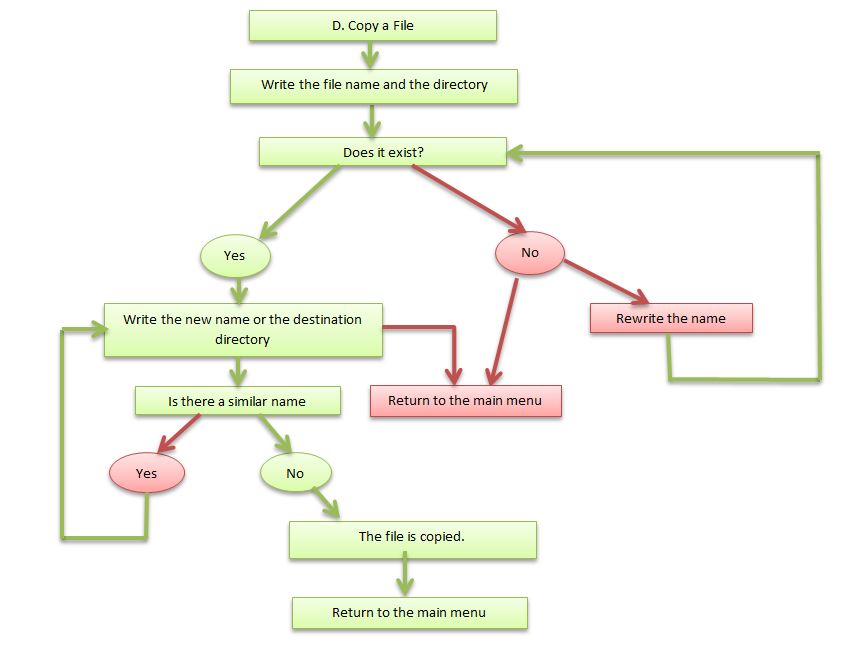
file2 = fopen(buffer3,"wb");

The type the file is flexible as well. The function is able to copy the file from directory to another, or in the same directory. From the code above, it is obvious that it reads the binary file, and the other one writes the binary file. So, we used while loop although we know it is somehow expensive and slow, but it does the job:

while((r=fgetc(file1))!=EOF) { fputc(r,file2); }

As known, the current working directory is shown by the program when it is running. The user has to provide only the name of the file first, and then the program prompts for the location of the file where it is originally located. After that it will prompts for the location where the user wants to copy it to.

There are different constraints the program should avoid which are the following:



1. **Move a file:**

This function allows the user to move the file by using the following function calls as well:

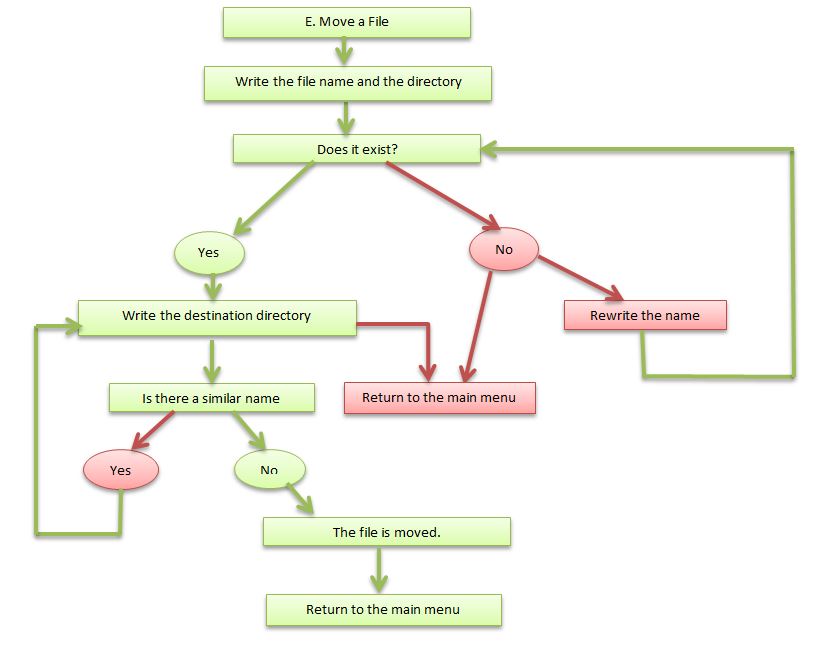
file1 = fopen(buffer1,"rb");

file2 = fopen(buffer3,"wb");

Although we could have used rename(old\_destination , new\_destination) which is faster, we used the same strategy of copying the file, but this time after copying, we delete the file from the root. The type of the file is flexible and the function is able to move the file from one directory to another. From the code above, it is obvious that it reads the binary file as well, and the other one writes the binary file. So, we used while loop as the copy file although we know it is somehow expensive and slow, but it does the job. Of course we could have manipulated another way with rename(old\_destination , new\_destination); .

As known, the current working directory is shown by the program when it is running along with the files in that directory. The user has to provide only the name of the file first, and then the program prompts for the location of the file where it is originally located. After that it will prompts for the location where the user wants to move it to. The program doesn’t provide moving directories though.

There are different constraints the program should avoid which are the following:



1. **Editing text files:**

As it is shown before in the main menu function graph, this function is considered a main menu for other sub-functions. It includes all the options of editing a text file, and it also provides the help utility which will be described later on here.



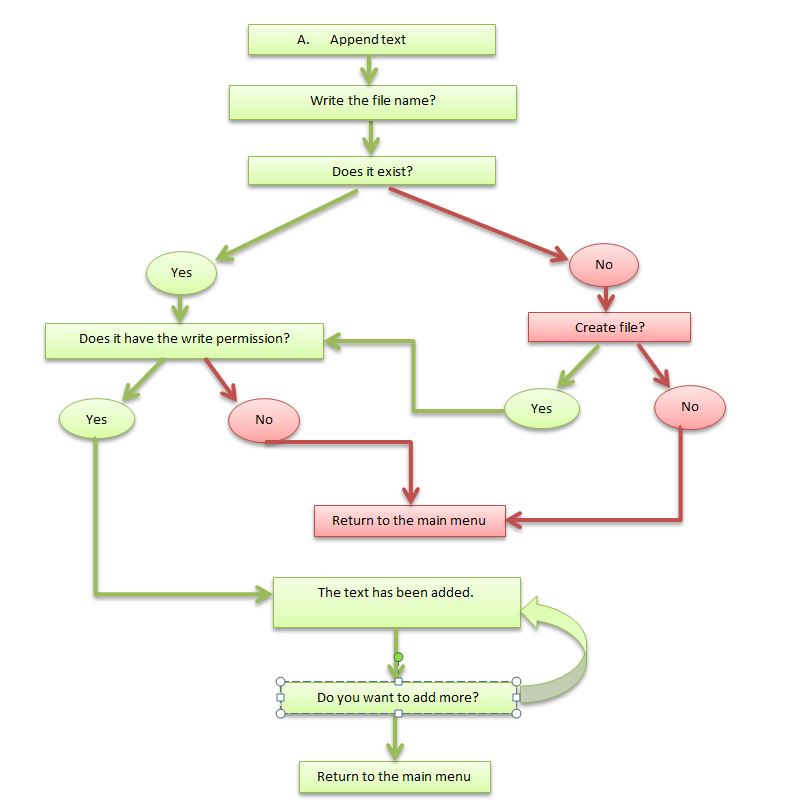
The function provides the flexibility to return to the main menu and vice versa. The same interactive screen was used here as the main menu so it allows the user to access all functions easily. The functions in this menu as follows:

1. **Appending a text:**

This function provides the accessibility and flexibility to append user’s text input to an existing file in the current working directory. It shows also the current working directory where the file should exist in; thus, the user shall know the file which he shall append text to. We used the permission accesses in (access(buffer1, 0) == -1);. The following table will illustrates it.

|  |  |
| --- | --- |
| F\_OK or 0 | Check for existence |
| W\_OK or 2 | **Check for write permission** |
| R\_OK or 4 | **Check for read permission** |

There are different constraints the program should avoid while running. The constraints are as follows:



1. **Inserting a text:**

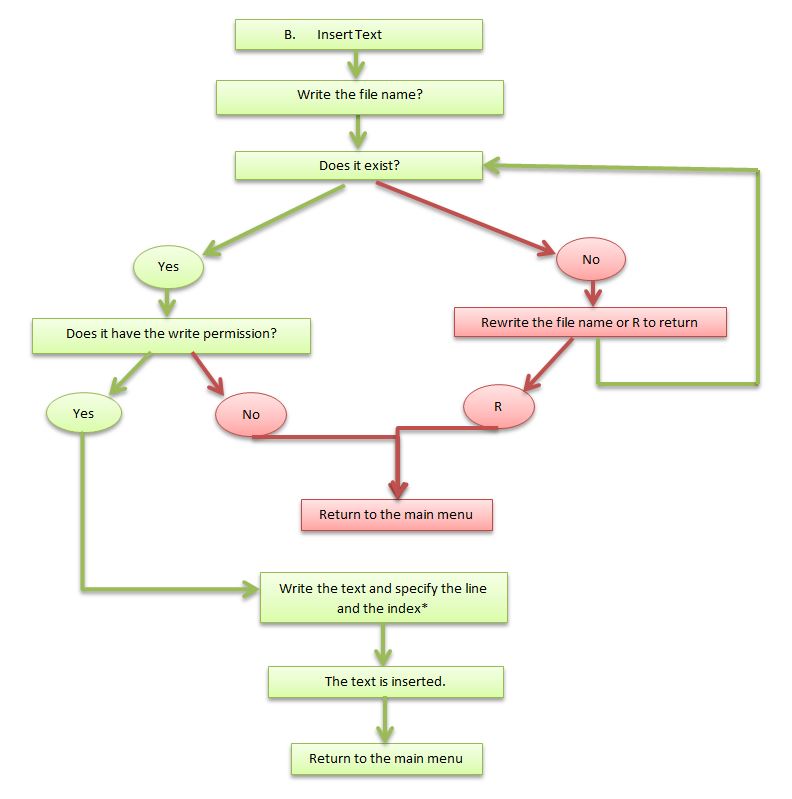
This function provides the accessibility and flexibility to insert the user’s text to an existing file in the current working directory. It shows also the current working directory where the file should exist in; thus, the user shall know the file which he shall insert the text to. We used the permission accesses also in (access(buffer1, 2) == -1);.

|  |  |
| --- | --- |
| F\_OK or 0 | Check for existence |
| W\_OK or 2 | **Check for write permission** |
| R\_OK or 4 | **Check for read permission** |

This function, after checking the existence of the file, it shows the content of the file first on the screen for the user to know where he wants to insert his text. Not only that, it prompts him in which line he wants to insert his line, then in which index. For example, let’s assume that this text is in a specific line in the text file *“Professor Shervin is a professional professor”*, and then I want to insert *“such”* between *is* and *a professional.* So, the program counts the index in this line and prints it in the screen. The user shall in this position insert his desired index. In our case for example, the index is 21. So, the program will prints the changes after editing automatically on the screen. It shall print in our example *“Professor Shervin is such a professional professor”.*

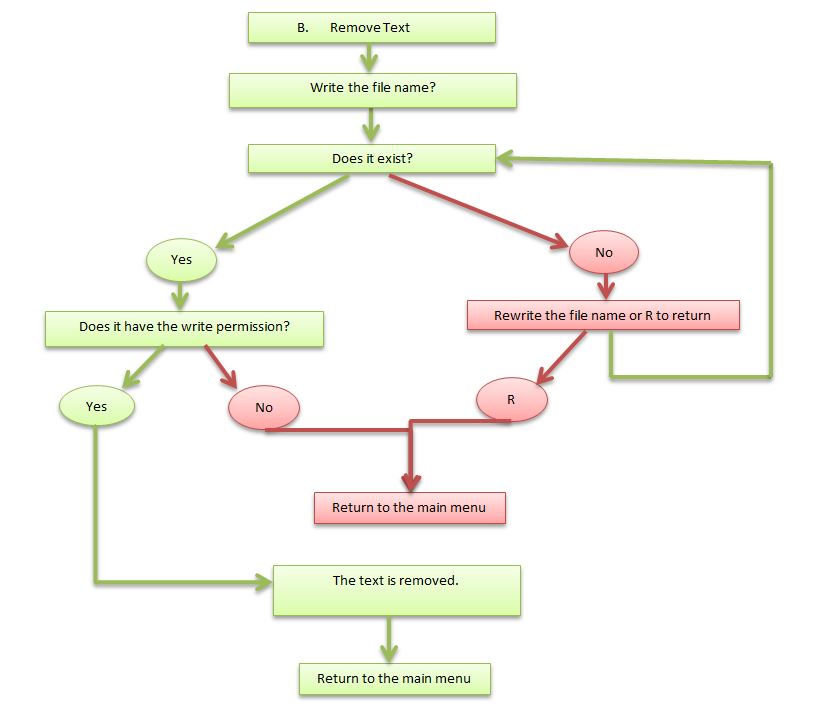
Note: the program will not insert the text once the user input is wrong although it prints that the text has been inserted successfully.

There are different constraints the program should avoid while running. The constraints are as follows:



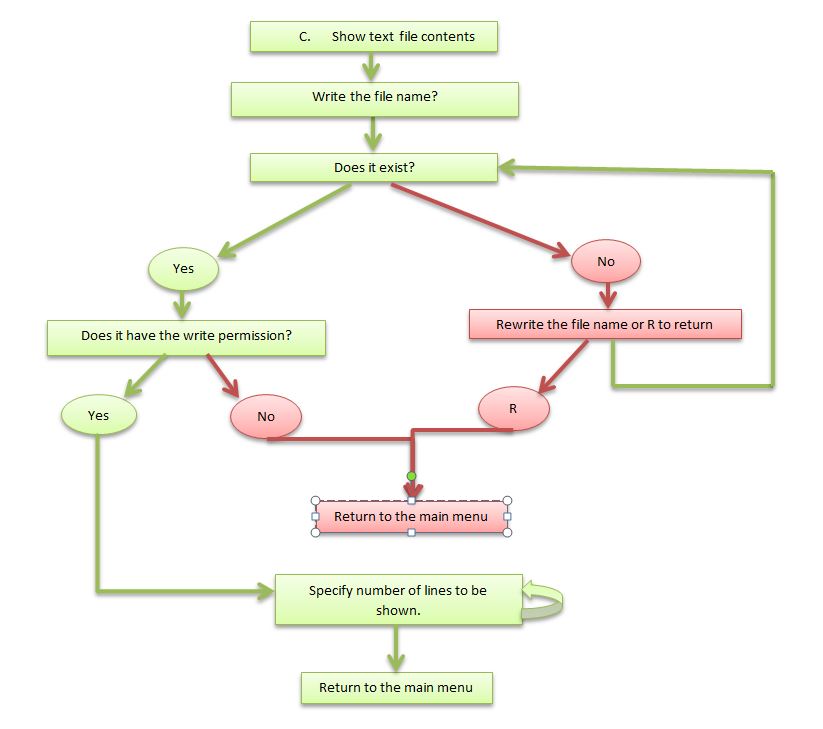
1. **Removing the text:**

This function provides the easiness to remove the file’s whole text. The file shall exist and it used the same strategy to check the existence of the file and such as Inserting and Appending functions. Mostly, it has the same features as Appending and inserting functions.



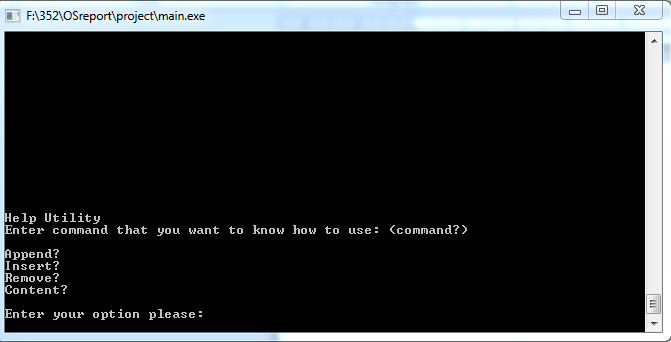
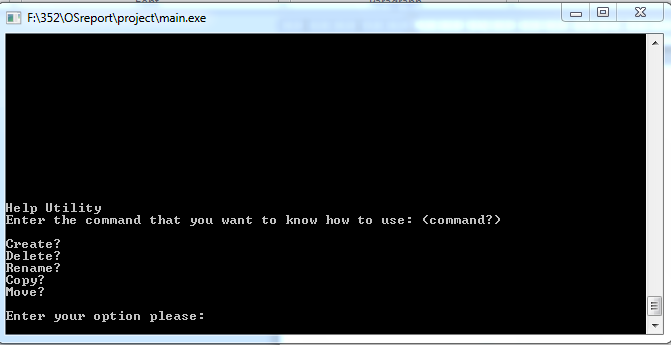
1. **Show the content of a text file:**

This function provides the user to ability to show the file’s whole text. The file shall exist and it used the same strategy to check the existence of the file and such as Inserting and Appending functions. Mostly, it has the same features as Appending and inserting functions; however, in case the file is readable, it prompts the user for how many lines he desires to see. Then within an infinite loop, it shows the contents. The user has to put a random input in order to show the other lines; otherwise, it will not show. In case the user is done reading and there are no more lines to show, the user shall input *R* to return to the main menu. Here are the constraints and user’s decision making options as follows.



1. Help Utilities:

Just like any other function above, we have approached it through the main menu options. It shall print the description of that specific function although we believe there was no need for making it since we made the user manual part. However, it shall print the same things as the user manual.



The user shall add ? to the end of his desired help function in order to have more details on that specific function.