**Introduction:**

In this project we have to Design a portable file manipulator that operates on files called pofm. In which it includes below characteristics: Since it must be portable, it must NOT use any OS commands which means that *pofm* can be easily ported to any operating system and platform. The exceptions and issues must be resolved by all commands; e.g., file does not exist, file is read-only, file name already exists, invalid command use, etc.

In this report, we give a full outline of the task and how it was drawn nearer to the last serviceable program. Convenient File Manipulator, PFM is a program which manages numerous responsibilities when a client runs it. It gives diverse document hierarchical functionalities, for example, making, erasing, or renaming various types of records. The association of data may include access, updates and development of data between gadgets. Notwithstanding the hierarchical functionalities, the program gives the openness and adaptability to alter text documents. Not just that, a coordinated intelligent guidance screen was made to make it simpler for the client to use. Truth be told, the program is a key touchy.

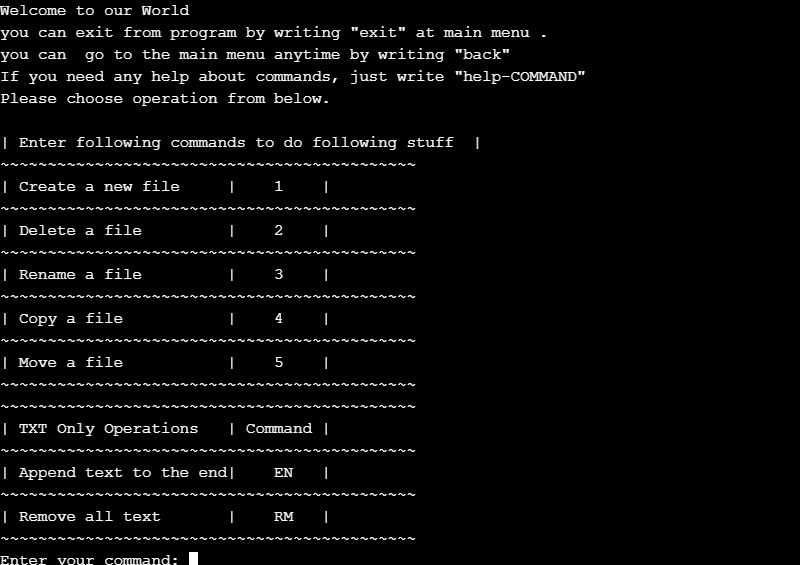
PFM works in any working framework either in Windows as well as iOS. It was expected to dodge any framework calls which may emerge any blunders in our program. Subsequently, the accompanying report delineates the entirety of the plan and usage. What's more; it shows the client dynamic through the way toward 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.

**The main menu:**

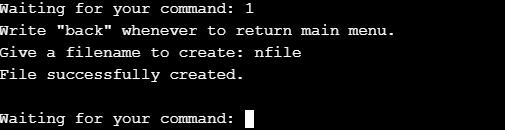
The primary usefulness gives an adaptable intelligent stage between the capacities' headers and the client input. In one or the other case, the program ensures that memory the board goes perfectly. We utilized all the memory area alternatives to maintain a strategic distance from any outside division in each capacity. We utilized for instance roast \*file\_name = malloc(sizeof(\*file\_name)); for the dynamical memory portion to stay away from an excess of allotment by utilizing singe file\_name[50]; . Furthermore, we needed to free the memory once we don't utilize that portion any longer by utilizing free (file\_name); All the capacities give the openness to re-visitation of the primary menu on the off chance that any startling disappointments happen by entering "back" which means getting back to the fundamental menu. Moreover, the program covers any information botches as it goes in a boundless circle until the client enters the letter "back".

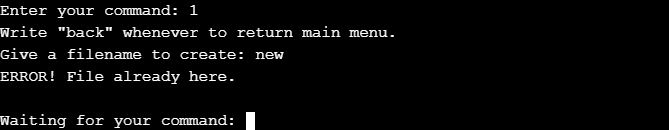


**Create a file:**

This capacity permits the client to make new record. The sort the document is adaptable to the client as there are different sorts of record types, for example, pdf, mp3, txt and so on in any case, the capacity doesn't make an index sadly as we were thinking about it not a record type. In any case, prior to making the document, it checks if that record exists in the catalog utilizing at that point the program makes the record in the current working catalog where the

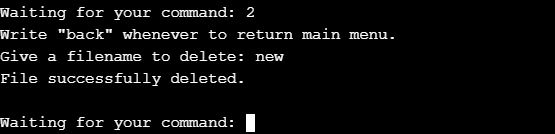
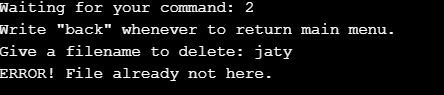
program is running. The current working registry will be print to the client screen to give the full data where his record will be made. For example, if give command 1 then it ask us for a filename. After when we enter filename, a message will be prompted “file successfully created” and it will create a file of your given name. And if the given file name already exists, then it will prompt a message “error- file already exist”.



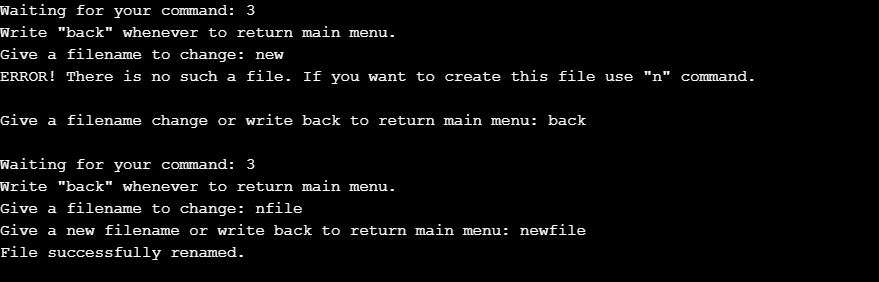


**Delete a file:**

This capacity permits the client to erase the document. The sort the record is adaptable also it tends to be pdf, txt and so forth the capacity checks the presence of the document utilizing too, we utilized 0 rather than F\_OK as F\_OK gets back with a blunder in iOS framework. At that point the program erases the record in the current working index where the program is running. The current working catalog will be print to the client screen alongside the documents inside the index to give him/her the full data which record we wants to erase. For example, if you give command 2 then it ask us for a filename. After when we enter filename, a message will be prompted “file deleted successfully” and it will delete a file of your given name. And if the given file name not exist, then it will prompt a message “error- file not exist”.

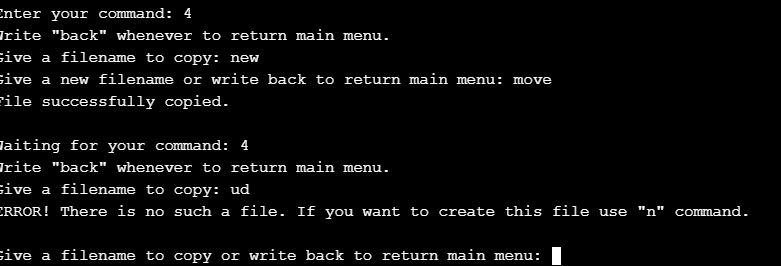


**Rename a file:** This capacity permits the client to rename the document utilizing rename(old\_name,new\_name) work call. The sort the document is adaptable too it very well may be mp3, pdf, txt and so forth the capacity can rename a registry too. The program renames the document in the current working catalog which is appeared by the program when it is running. The documents inside the index are appeared also to give the full data which record we wants to rename. For example, if you give command 3 then it ask us for a filename that you want to change. After when we enter filename, then it will ask us for a new filename that you want. After a message will be prompted “file successfully renamed”. And if the given file name not exist, then it will prompt a message “error- there is no such file”.



**Copy a file:**

The sort the document is adaptable also. The capacity can duplicate the document from index to another, or in a similar registry. From the code above, clearly it peruses the parallel record, and the other one composes the paired document. As known, the current working index is appeared by the program when it is running. The client needs to give just the name of the record first, and afterward the program prompts for the area of the document where it is initially found. After that it will prompts for the area where the client needs to duplicate it to. For example, if you give command 4 then it ask us for a filename that you want to change. After when we enter filename, then it will ask us for a existing filename that you want to copy. Then it will ask for different filename and in that it will create a new file and copy the content of existing file into it.



**Move a file:**

Despite the fact that we might have utilized rename (old\_destination , new\_destination) which is quicker, we utilized a similar methodology of replicating the document, however this time subsequent to duplicating, we erase the record from the root. The sort of the record is adaptable and the capacity can move the document starting with one registry then onto the next. From the code above, clearly it peruses the parallel document too, and the other one composes the twofold record. In this way, we utilized while circle as the duplicate record despite the fact that we realize it is by one way or another costly and moderate, yet it takes care of the work. As known, the current working registry is appeared by the program when it is running alongside the records in that catalog. The client needs to give just the name of the document first, and afterward the program prompts for the area of the record where it is initially found. After that it will prompts for the area where the client needs to move it to. For example, if you give command 5 then it ask us for a filename that you want to move. After when we enter filename, then it will ask for a pathname where you want to move your existing file that you want to copy. Then a message will be prompted “file has been moved”.

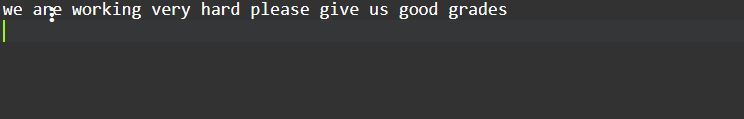
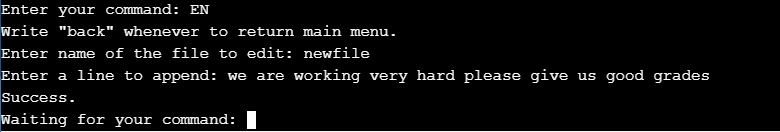
**Editing text files:**

As it is appeared before in the fundamental menu work chart, this capacity is viewed as a principle menu for other sub-capacities. It incorporates all the alternatives of altering a book document, and it likewise gives the assistance utility which will be depicted later on here. The capacity gives the adaptability to re-visitation of the fundamental menu and the other way around. A similar intuitive screen was utilized here as the primary menu so it permits the client to get to all capacities without any problem. The capacities in this menu as follows:

**Appending and inserting a text:**

This capacity gives the openness and adaptability to add client's content contribution to a current document in the current working registry. It shows additionally the current working catalog where the document should exist in; hence, the client will know the record which he

will add text. “Inserting” capacity gives the availability and adaptability to embed the client's content to a current document in the current working registry. It shows additionally the current working catalog where the document should exist in; in this manner, the client will know the record which he will embed the content to. This capacity, subsequent to checking the presence of the record, it shows the substance of the document first on the screen for the client to know where he needs to embed his content. Not just that, it prompts him in which line he needs to embed his line, at that point wherein list. After that it will prompts for the area where the client needs to move it to. For example, if you give command EN then it asks us for a filename that you want to edit. After when we enter filename, then it will ask for a line to append. Then you can type anything on that file.



**Removing the text:**

This capacity gives the effortlessness to eliminate the record's entire content. The document will exist and it utilized a similar procedure to check the presence of the record and, for example, Inserting and Appending capacities. Generally, it has similar highlights as Appending and embeddings capacities. For example, if you give command RM then it asks us for a filename that you want to empty. After when we enter filename, then it will empty the file.

