**Project: Portable File Manipulator (pofm)**

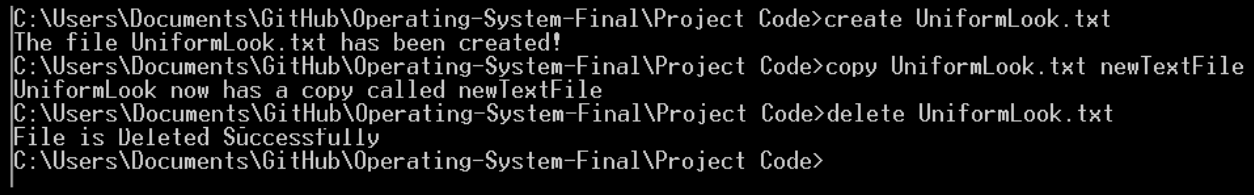
**Design Goals**

* Uniform look and functionality.
* Modular design.
* Collaborative environment for the team to work in.
* Do all the questions asked of us.

Uniformity

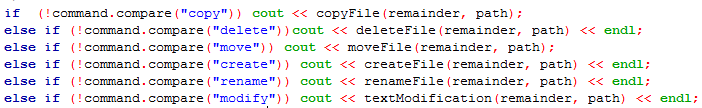
**Look**

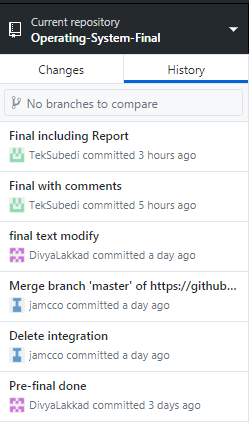
For having a uniform look, we have all decided how our POFM will accept user inputs. This in turn would make sure all the commands look the same when entering them in. The only acceptation to this was the text modification. The commands would have been too long and cumbersome to the user. Because of this we had a second prompt that shows up when the person selected the text modification command.



**Functionality**

To have similar functionality we set out right at the start how the main file would interact with each header file. We had to agree that we would accept two strings. First path, when the user is going through the file system, they able to change directories. This path that the user is viewing was kept and passed into the header files. Secondly the command it self is passed into the header files. Since all our header files had the same two inputs, and require a string to be returned, we in turn had uniform functionality across the whole project and could use each others code to help us with each others’ parts. In the below example “Remainder” is the rest of the command.

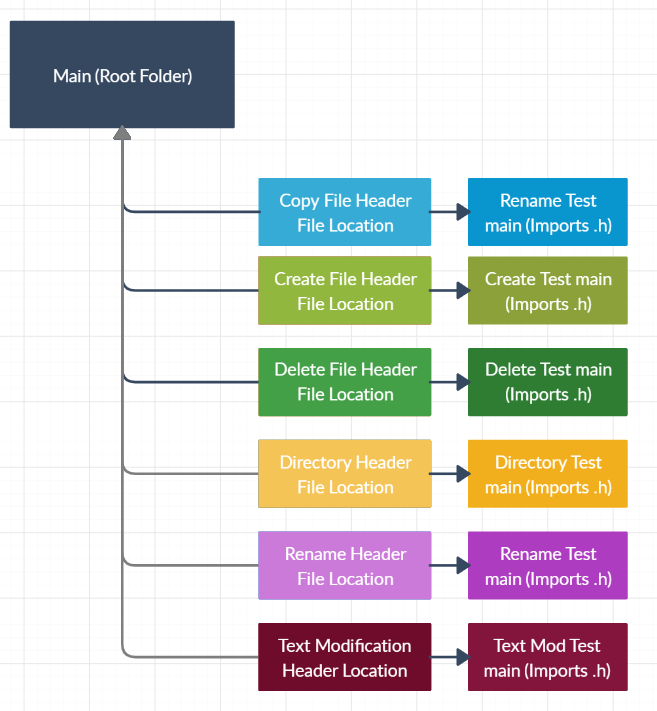




Collaborative Environment

Some of the project members agreed to use GIT in order to share our code to each other. This was a choice because we had so many members in our group and not having some way to share code it would get unmanageable. Another reason to use something like GIT is we could have documentation and saved states we could revert too. This was one of the first things we set up together.

Modular Design



Before the first line was codded and our parts assigned to each other, we first though about how we wanted to structure our project and how to work on our own parts of code. This is the file space structure, we had to think about how all our pieces would connect. To accomplish this, every task was split up into a header file. These header files also had a main file in the folder, this was for testing and developing your assigned header file. We had a main.cpp that is the “root” of the project. This was delegated to one person to manage and control. This root file would be the main thing the user interfaces with until they decide what command they want to use.

Another reason to pick the modular design is because we were using GIT. If we all tried to code in the same place, in the same folder, it would have been a nightmare. We knew this was going to be the case, and for the people who have never used GIT, we wanted them to have a non merge conflict experience.

Once we completed our parts, we were able to put all our commands into the main.cpp (root folder). Because of our choice of modular design this was one of the easiest tasks of the whole project.

Do all the questions asked of us.

This was a big part of our project, we had to do the functionality requested by the teacher. Without this we would get a bad mark, so we really tried to accomplish this.

Now we will have each person explain their section and their design choices.

**Copy Files (James McCormack)**:

For the copy file command, it was quite simple. In the directory you would be able to copy a file in the same place with a new name. I did not allow for identical names as I know would in fact overwrite the original file. While it would overwrite it with the same file, it would technically not be a new copy.

**CMD functionality (James McCormack):**

The idea of being able to do things like change directory was a big priority for us. Otherwise this program is not that useful since it would only work in one directory at a time. We did not use any system commands here, but instead just mimicked it’s functionality with our own code. Because of this the user can go in and out of folders and then type their command they wish to use. All of this happens while the user is seeing what directory they are in and can list all the files and folders.

**Create File / Delete File (Prabjeet):**

When you run the code, it will take you to the main directory file from which you can open the file you want from the main directory. If searchable file doesn't exist, it will show that Unable to find the file. Once it will find the file you are looking for then you are good to go to delete it. Once you delete it will give you the output that File has been deleted. Thank you

**Rename A File (Tek):**

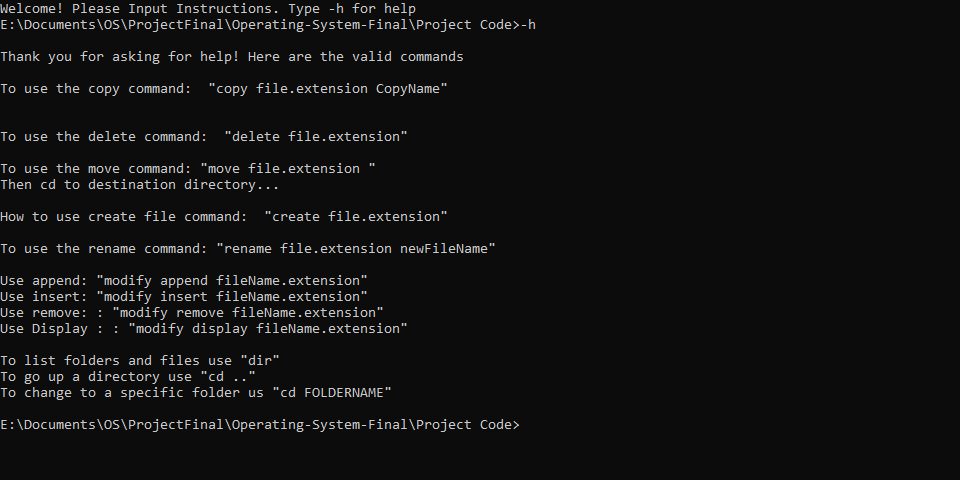
**Move a File (Nick):**

There where a couple implementation options when designing this function. I decided to do this function in two parts: pass in the file then navigate to the destination directory. I chose to do it in this fashion to take advantage of the CMD functionality and simplicity. Instead of the user having to copy or input the destination file path, they can instead navigate to the destination increasing flexibility and decrease typo errors. To physically move the file, my function creates a new file with the path of the current directory the user navigated to, copies the content of the original file to the new one, then deletes the file from the source destination. Once user completes this operation, the path returns to the original where the file resided.

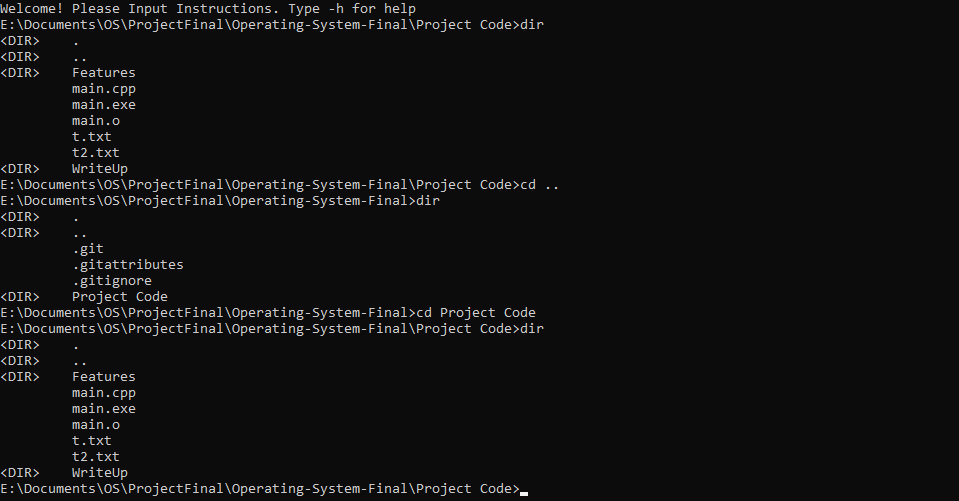
**Rename A File (Divya):**

User Manual

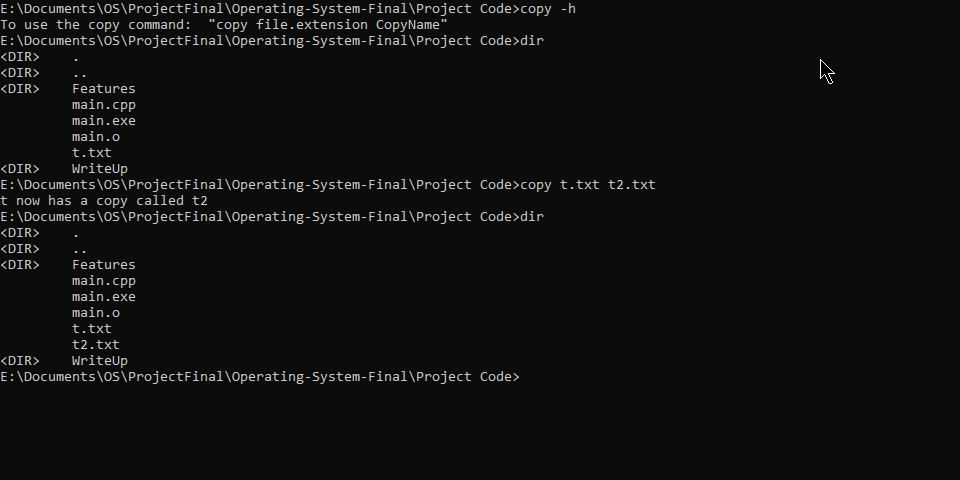
**Generalities**

* “-h” – Displays the help text from each function in this project.
* Each command is executed by first typing the name of the function then passing parameters associated with that command (possible parameters can be viewed by invoking the help of that command).

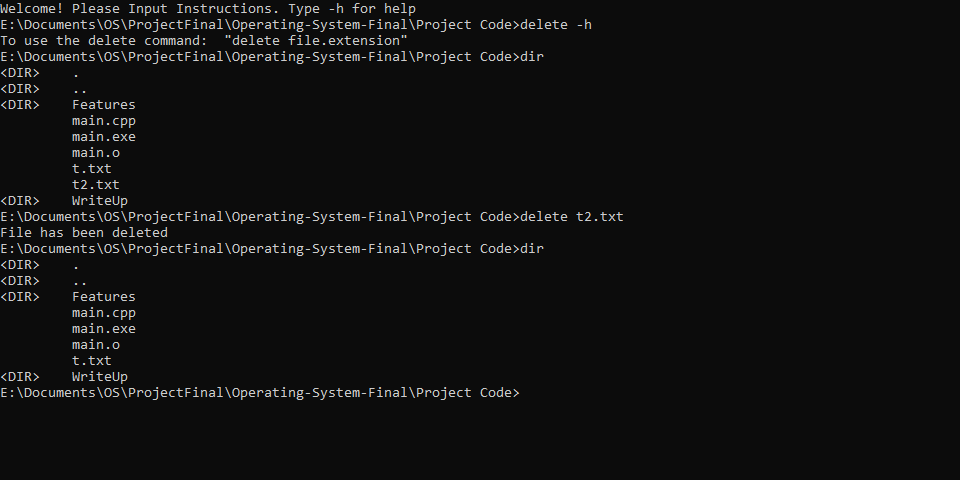
**CMD Functionalities**

* “dir” – View contents of the current directory
* To change the working directory: “cd extension” where extension can be
  + “..” – move up to parent directory.
  + ****“<directoryName>” – move into the directory specified by directoryName.

**Copy Command**

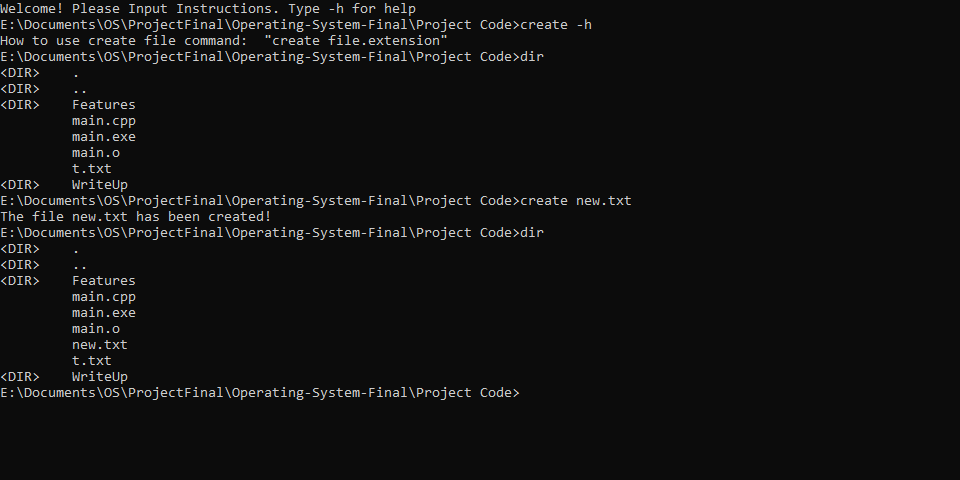
* “copy -h” – Display help
* “copy file.extension copyname” - Copy file specified by file.extension to a new file named copyname (copyname must be different from the name of the file being copied).

**Delete Command**

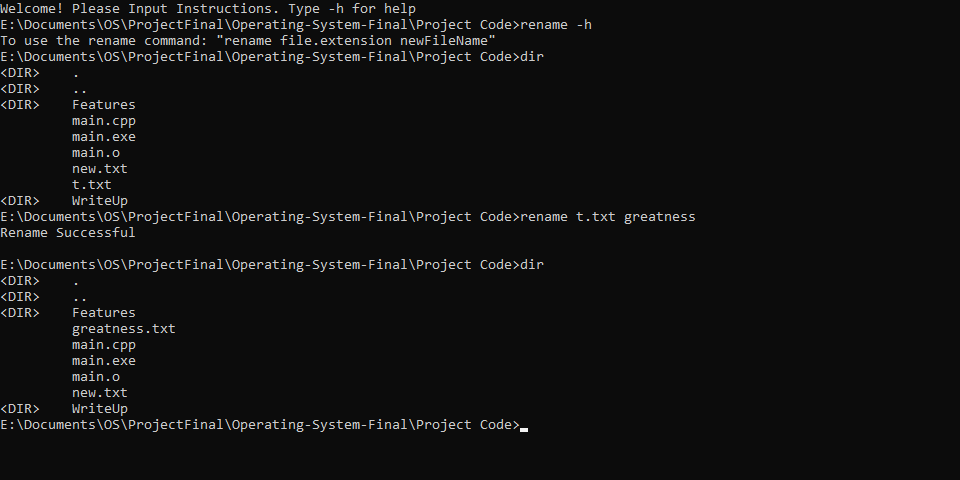
* “delete -h” – Display help
* “delete file.extension” – Deletes a file specified by file.extension

**Create Command**

* “create -h” – Display help
* “create file.extension” – Creates a new file named file.extension



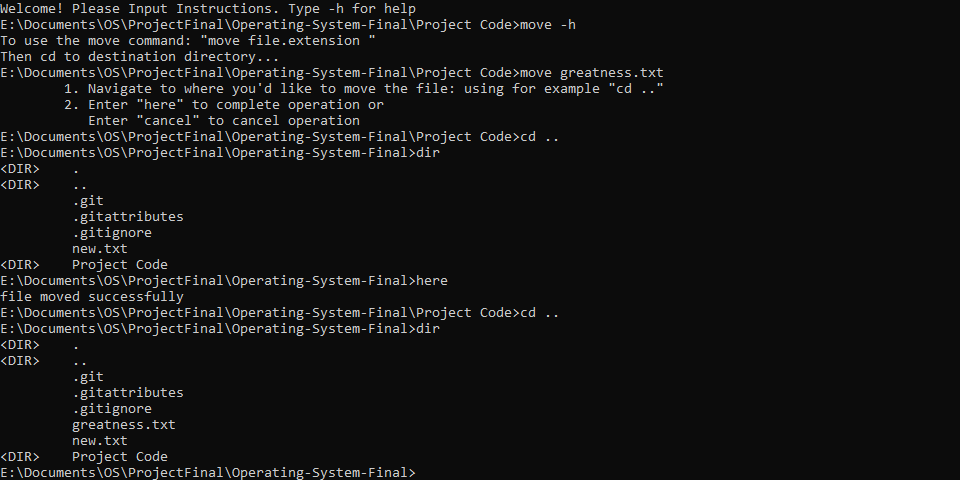
**Rename Command**

* “rename -h” – Display help
* “rename file.extension newFileName” – renames the file specified by file.extension to newFileName. Extension is carried over.

**Move Command**

* “move -h” – Display help
* 3 steps to move a file:
  1. “move file.extension” – file.extension specifies the file to be moved
  2. Using CMD functionalities, move to desired destination location
  3. “here” – To complete the operation or

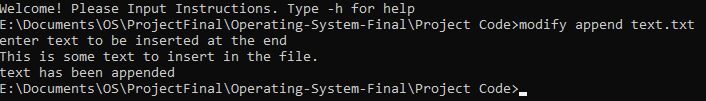
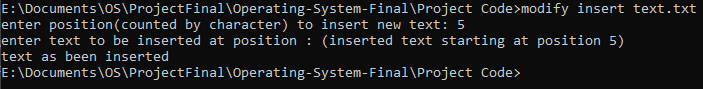
“cancel” – To cancel the operation

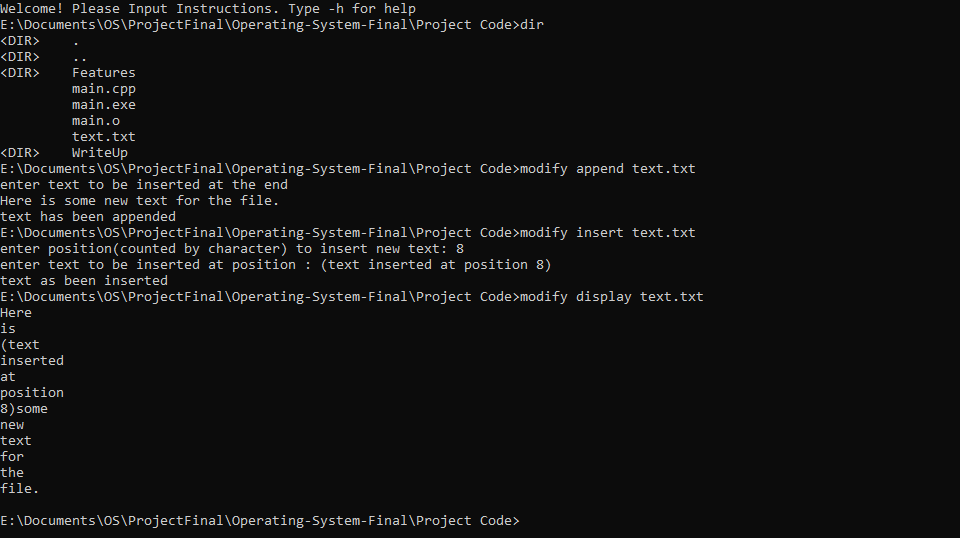


**Modify Command**

* “modify -h” – Display help

4 sub commands:

* “modify **append** file.extension” – Append text to the end of the file specified by file.extension
* “modify **insert** file.extension” – Append text to the file specified by file.extension starting from custom position
* E:\Documents\ShareX\Screenshots\2019-11\cb_console_runner_mmxqtUGknt.png“modify **remove** file.extension” – Removes all text from the file specified by file.extension
* “modify **display** file.extension” – Displays the content of the file specified by file.extension



**Cat Command**

* “cat” – Friendly cat

Rename a file

1. **Final report (60%)**: no later than December 2. This report consists of 3 components:
2. A design guide (30%): start by a short introduction, followed by explanation of your design and implementation. Give reasons for your design and implementation choices. There should be enough details in your report to explain everything. I do not set number of pages: it’s up to you and how many pages you need to explain things.
3. A user manual including examples and screenshots (20%).
4. All your code, with in-line comments (10%).