Testing

Based on the four basic functions of mycat, I run the whole-system test. Here below is my test script:

1. For `mycat` w/o any other arguments

   `> ./mycat`

   Successfully prints out what I typed in by keyboard.

   ```
   asdf
   asdf
   ```

2. For `mycat` w/ `exist_file_name` which is smaller than 64 KiB

   `> ./mycat file1`

   Successfully prints out the ENTIRE file content

3. For `mycat` w/ `exist_file_name` which is larger than 64KiB

   `> ./mycat file2 file3 file 4`

   Successfully prints out ALL the file content

4. For `mycat` w/ a `directory_path`

   `> ./mycat directory_path`

   Successfully prints out ERROR message

5. For `mycat` w/ a `not_exist_file_name`

   `> ./mycat not_exist_file_name`

   Successfully prints out ERROR message

6. For `mycat` w/ all possible cases

   `> ./mycat file1 directory_path not_exist_file_name`

   Successfully implemented

7. For mycat w/ *.file_type

   > ./mycat *.txt

   Successfully print out all files with .txt

8. For `mycat` w/ unix command and exist_file_name

   `> ./mycat < file1`

   Successfully prints out content of file1

   `> ./mycat > file1`

   Successfully saved standard input in file1

   `> ./mycat file1 > file2`

   Successfully copied file1 to file2

   `> ./mycat file1 < file2`

   Successfully print out the content of file2

> `./mycat >> file1`

Successfully appended standard input after content of file1

> `./mycat file1 >> file2`

Successfully append file1 to file2

> `./mycat << file1`

Bash Warning but successfully print out the standard input

> `./mycat file1 << file2`

Bash Warning but successfully print out the content of file1

> `./mycat < directory_path`

Successfully prints out error message

> `./mycat < not_exist_file_name`

Successfully prints our error message

Bash_permission_denied:

Files that have no read or write permission to read.

Bash will give error messages directory.

QUESTION: How does the code for handling a file differ from that for handling standard input? What concept is this an example of?

To handle file, I did `open(2)`, `read(2)`, and `write(2)` and to handle standard input, only `read(2)` and `write(2)` are required. Furthermore, the arguments we pass to the functions are different. For standard input, 0 is the first argument indicating that this is a standard input, and 1 is indicating that this is a file read.

The relevant concept might be "*Avoid excessive generality*". Though these two methods have different goals, they use the same functions, and re-using the same much more basic functions like `read(2)` and `write(2)` will reduce complexity of the system.