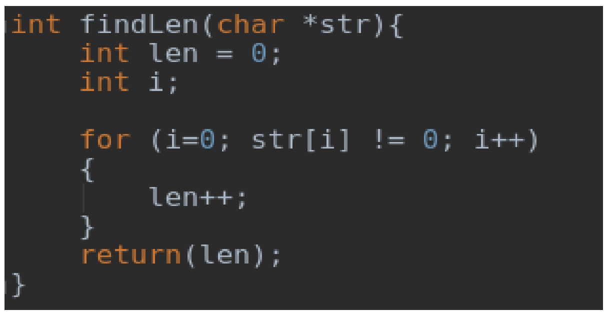
Introduction:

In this practical, We had to implement the command line tool “ls -n”. We were required to use the C programming language and Linux system calls without any standard or external libraries to develop this system utility. Briefly, the program would take a file/ directory name or path and should display the details of said file/directory as “ls -n” would.

Design and Implementation:

Before explaining the design and implementation of this program, We must first explain the prequel functions that we are going to use as tools to make our coding easier and cleaner. We know that we are required to do some string manipulation in the main algorithm, for this reason, we implemented some <string.h> functions and proposed a modified implementation of itoa().

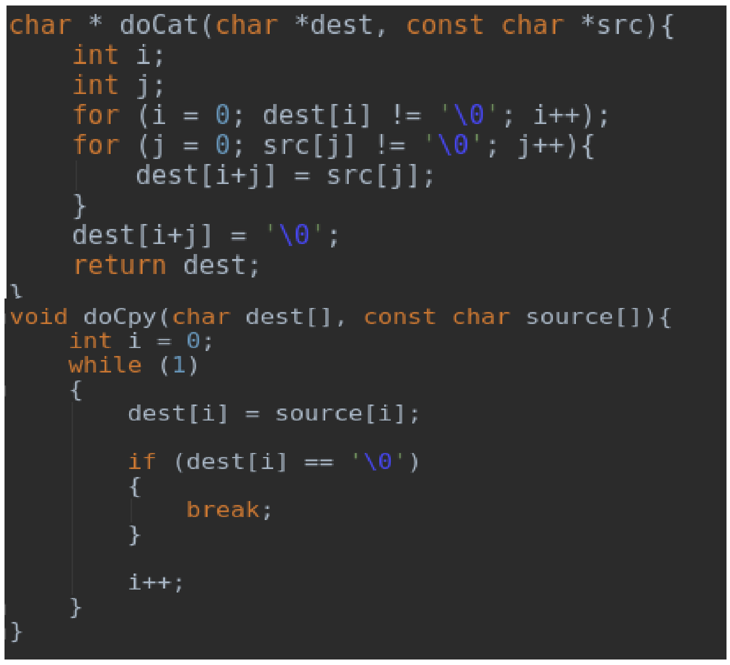
First, We implemented a function that would find the number of characters in a string given as a parameter.



```
int findLen(char *str){
    int len = 0;
    int i;

    for (i=0; str[i] != 0; i++)
    {
        len++;
    }
    return(len);
}
```

This method is pretty straight-forward. It iterates through the given string and returns it’s length.

Then, We had to also implement two string manipulation functions that we'll have to use in our main method, strcat() and strcpy().

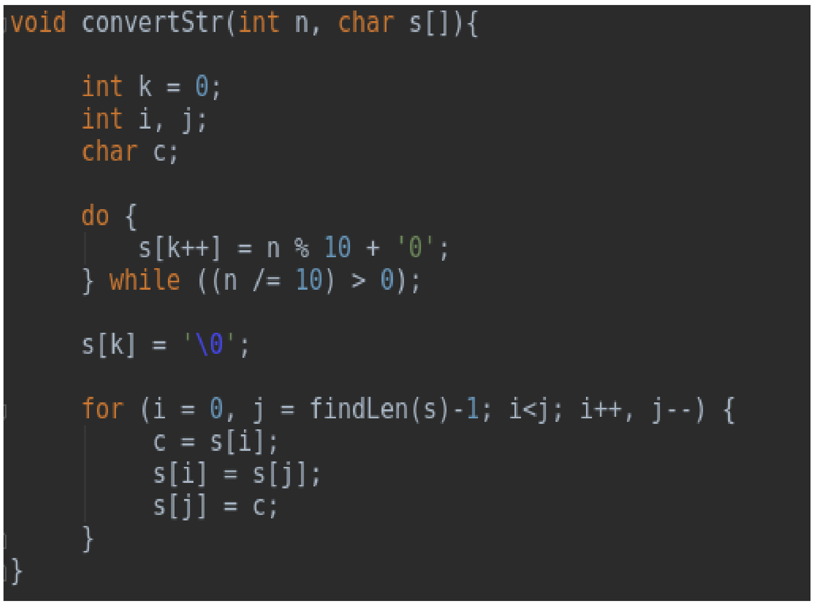


```
char * doCat(char *dest, const char *src){
    int i;
    int j;
    for (i = 0; dest[i] != '\0'; i++);
    for (j = 0; src[j] != '\0'; j++){
        dest[i+j] = src[j];
    }
    dest[i+j] = '\0';
    return dest;
}
void doCpy(char dest[], const char source[]){
    int i = 0;
    while (1)
    {
        dest[i] = source[i];

        if (dest[i] == '\0')
        {
            break;
        }

        i++;
    }
}
```

The concatenate implementation takes a src string and adds it at the end of the dest string. This is done by using iteration.

The copy implementation copies each characters at all indexes of the source string to the dest string. This is also done by using iteration but we can note that here we use a while loop and we break out of it when we encounter the string ending character “\0”.

Finally, We had to create a special itoa() implementation for printing purposes since the write() system call only takes a string and not numeric types. This is inspired by the itoa() implementation in “The C programming language” by Brian Kernighan and Dennis Ritchie although it is heavily modified and personalized.



```
void convertStr(int n, char s[]){

    int k = 0;
    int i, j;
    char c;

    do {
        s[k++] = n % 10 + '0';
    } while ((n /= 10) > 0);

    s[k] = '\0';

    for (i = 0, j = findLen(s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

This part puts the digits in reverse order inside our string. It gets the next digit and deletes it.

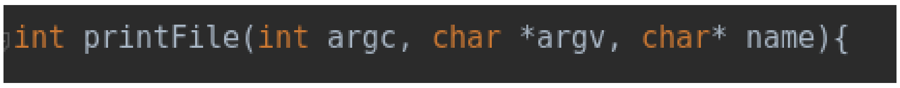
Puts the string terminator character at the end of the string to actually make it a valid string.

This part reverses the digits in the string in place to result in the desired converted string.

All these functions are going to be used in our main algorithm and explaining them before-hand will now allow us to mention them without giving extra details.

Our main algorithm is divided into 2 parts, The printFile() function and the main method that contains the program’s loop where printFile() is called.

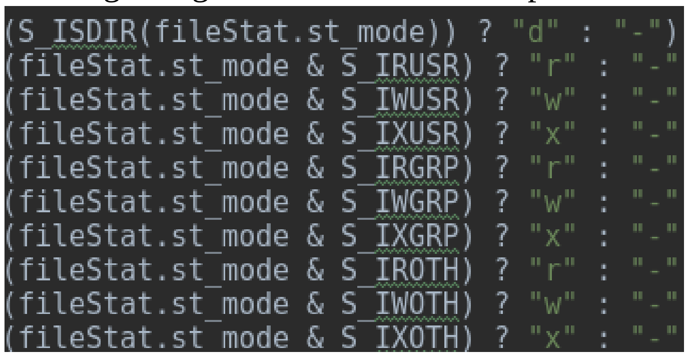
We are going to explain how printFile() operates before generalizing to the main loop:



```
int printFile(int argc, char *argv, char* name){
```

This method prints all the information wanted for a single file given in the parameters. Here, argv is the full file name (including it’s relative path to the current directory) and name is the file name used for printing purposes.

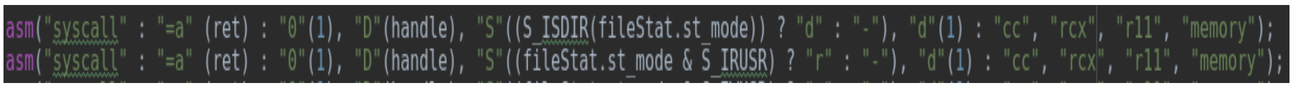
The first thing we do is some standard error checking to make sure the file exists and is processed correctly by the function. Then we get into the printing.
The beginning of the line is the file’s permissions properties.



```
(S_ISDIR(fileStat.st_mode)) ? "d" : "-")
(fileStat.st_mode & S_IRUSR) ? "r" : "-"
(fileStat.st_mode & S_IWUSR) ? "w" : "-"
(fileStat.st_mode & S_IXUSR) ? "x" : "-"
(fileStat.st_mode & S_IRGRP) ? "r" : "-"
(fileStat.st_mode & S_IWGRP) ? "w" : "-"
(fileStat.st_mode & S_IXGRP) ? "x" : "-"
(fileStat.st_mode & S_IROTH) ? "r" : "-"
(fileStat.st_mode & S_IWOTH) ? "w" : "-"
(fileStat.st_mode & S_IXOTH) ? "x" : "-"
```

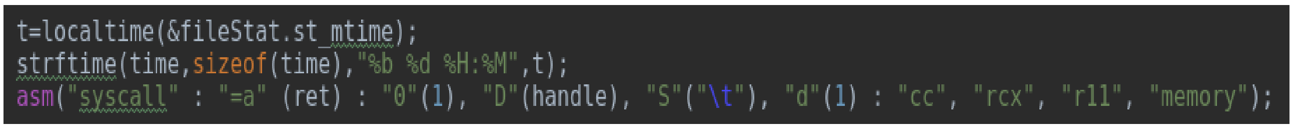
Here, we check if it is a directory. If it is we print “d”, else we print “-”.
We use the name conditional logic to print the properties for user, group and other.

The printing is done using asm calls like so:



```
asm("syscall" : "=a" (ret) : "0"(1), "D"(handle), "S"((S_ISDIR(fileStat.st_mode)) ? "d" : "-"), "d"(1) : "cc", "rcx", "r11", "memory");
asm("syscall" : "=a" (ret) : "0"(1), "D"(handle), "S"((fileStat.st_mode & S_IRUSR) ? "r" : "-"), "d"(1) : "cc", "rcx", "r11", "memory");
```

The following information we want to display are the number of links, the user id, the group id and the file size. This is done simply by calling the respective stat commands. However, most of these values are returned as unsigned longs and are required to be converted to strings. This is done by using the functions above.
Afterward, we want to display the file's time of creation or last modification.

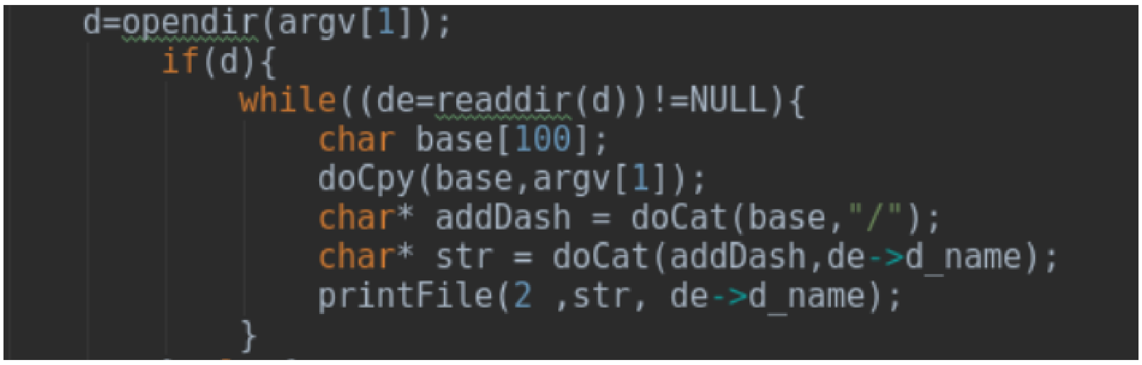


```
t=localtime(&fileStat.st_mtime);
strftime(time,sizeof(time),"%b %d %H:%M",t);
asm("syscall" : "=a" (ret) : "0"(1), "D"(handle), "S"("\t"), "d"(1) : "cc", "rcx", "r11", "memory");
```

Here, we use the strftime() method which comes with the few libraries we were allowed to use. It allows us to convert the local time we get from the fileStat to the same format given by "ls -n".
Then we just display it using asm. We can note here that all the elements in the file info line are separated by "\t". That makes the display easier to read and debug later on.

Finally, We print the file's name that is given in the function's parameter.

Now moving to the main method, before any kind of processing, we need to perform some error checking.
The main algorithm loop is the following :



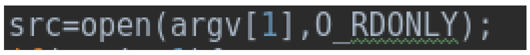
```
    d=opendir(argv[1]);
        if(d){
            while((de=readdir(d))!=NULL){
                char base[100];
                doCpy(base,argv[1]);
                char* addDash = doCat(base,"/");
                char* str = doCat(addDash,de->d_name);
                printFile(2 ,str, de->d_name);
            }
```

Simply put, This is the core of our "ls -n" implementation. Let's explain what is going on here.
The first thing we do is open our file/ directory that is given as an input when calling the program.
We call readdir() on "d" which is a call to getdents() and returns a pointer to a dirent structure. That allows us to obtain the files' names in our directory in a way where we can call stat on these files.

Here, we use a while loop to process all the files in our directory. If we call the program on a singular file, We will only go through the loop once and process this file.
In the loop, We use the concatenate function to result in a usable pathname for each file so that we can call the printFile function of it.
We can note here that we add a dash at the end of the file's name, this is to prevent an error from occurring.
Finally, we call printFile() on the file to print a line of information similar to what we would get if we called "ls -n" on this file.
The focus here was on implementing a really simple solution where we would split the logics to make is as understandable as possible.

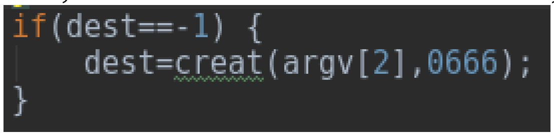
Extensions:

I attempted two extensions, one normal and one difficult as specified in the practical specifications.
First, I proposed a "mv" command implementation. This program copies a file or directory to a new location and deletes the source.
This is done by opening the source file using a read-only flag.



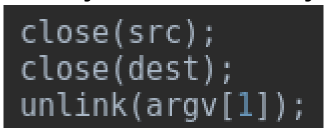
```
src=open(argv[1],O_RDONLY);
```

Then, if the destination file doesn't exist, we create one.



```
if(dest==-1) {
    dest=creat(argv[2],0666);
}
```

Afterwards, we write for 1 file to the other using a buffer and a while loop.
Finally, we close everything and delete the source file.



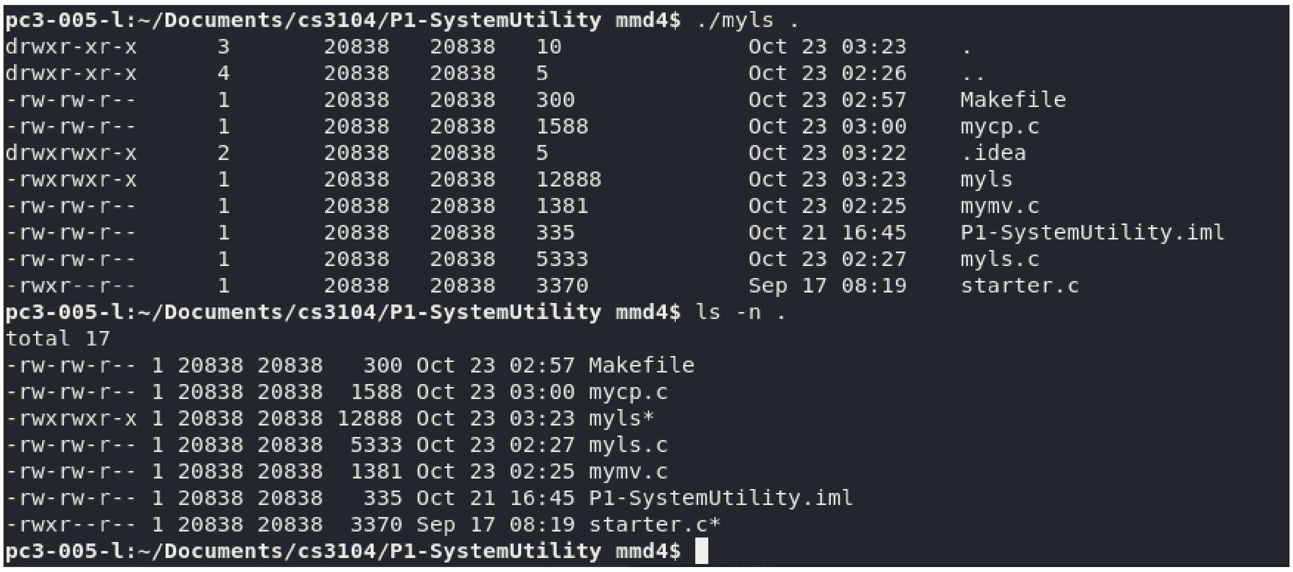
```
close(src);
close(dest);
unlink(argv[1]);
```

Once again, The error checking is done and on point.

The second extension is the "cp" command implementation which is very similar to the "mv" one but doesn't delete the source file.
It also checks if the destination file exists since it needs to be valid because cp doesn't create new files.

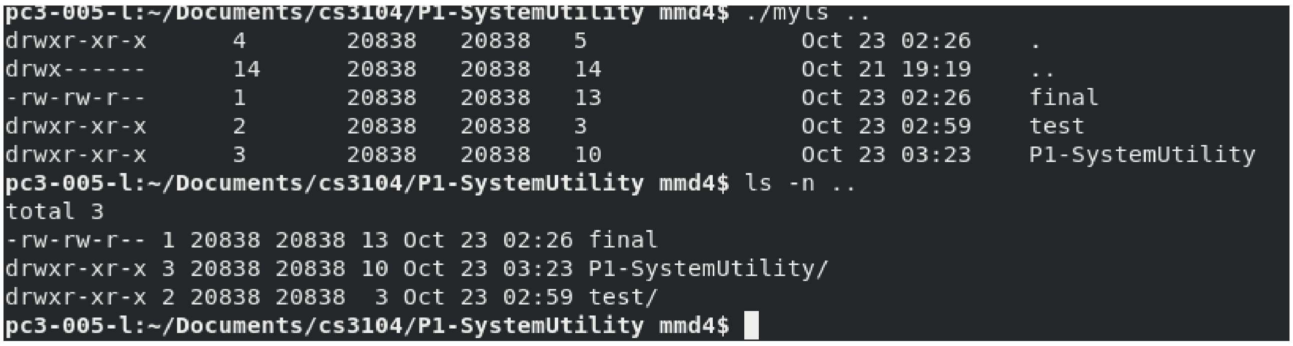
Testing and Debugging:

I tested my program against various inputs and compared the outputs with "ls -n" calls.
I also tested the error checking. Here are the results.



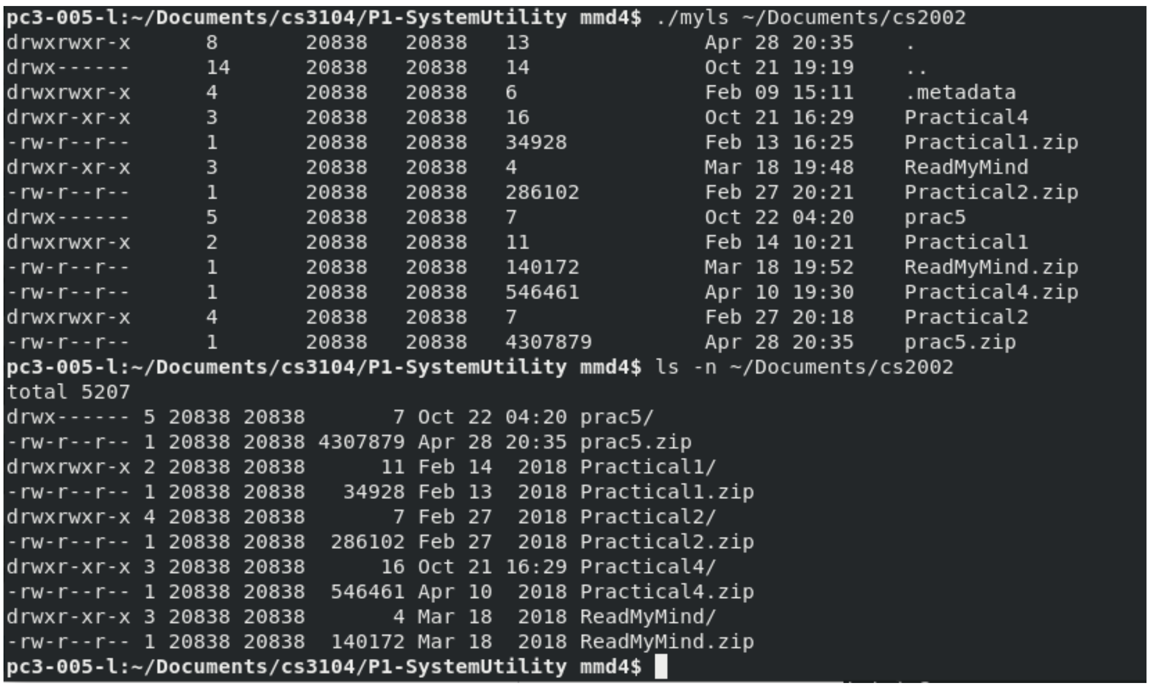
```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./myls .
drwxr-xr-x      3       20838   20838   10              Oct 23 03:23    .
drwxr-xr-x      4       20838   20838   5               Oct 23 02:26    ..
-rw-rw-r--      1       20838   20838   300             Oct 23 02:57    Makefile
-rw-rw-r--      1       20838   20838   1588            Oct 23 03:00    mycp.c
drwxrwxr-x      2       20838   20838   5               Oct 23 03:22    .idea
-rwxrwxr-x      1       20838   20838   12888           Oct 23 03:23    myls
-rw-rw-r--      1       20838   20838   1381            Oct 23 02:25    mymv.c
-rw-rw-r--      1       20838   20838   335             Oct 21 16:45    P1-SystemUtility.iml
-rw-rw-r--      1       20838   20838   5333            Oct 23 02:27    myls.c
-rwxr--r--      1       20838   20838   3370            Sep 17 08:19    starter.c
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ls -n .
total 17
-rw-rw-r-- 1 20838 20838   300 Oct 23 02:57 Makefile
-rw-rw-r-- 1 20838 20838  1588 Oct 23 03:00 mycp.c
-rwxrwxr-x 1 20838 20838 12888 Oct 23 03:23 myls*
-rw-rw-r-- 1 20838 20838  5333 Oct 23 02:27 myls.c
-rw-rw-r-- 1 20838 20838  1381 Oct 23 02:25 mymv.c
-rw-rw-r-- 1 20838 20838   335 Oct 21 16:45 P1-SystemUtility.iml
-rwxr--r-- 1 20838 20838  3370 Sep 17 08:19 starter.c*
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ 
```

We notice that for the current directory, the output is the same. Now let’s test if it the pathing has no issues. We first check the directory right above the current:



```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./myls ..
drwxr-xr-x      4       20838   20838   5               Oct 23 02:26    .
drwx------      14      20838   20838   14              Oct 21 19:19    ..
-rw-rw-r--      1       20838   20838   13              Oct 23 02:26    final
drwxr-xr-x      2       20838   20838   3               Oct 23 02:59    test
drwxr-xr-x      3       20838   20838   10              Oct 23 03:23    P1-SystemUtility
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ls -n ..
total 3
-rw-rw-r-- 1 20838 20838 13 Oct 23 02:26 final
drwxr-xr-x 3 20838 20838 10 Oct 23 03:23 P1-SystemUtility/
drwxr-xr-x 2 20838 20838  3 Oct 23 02:59 test/
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ 
```

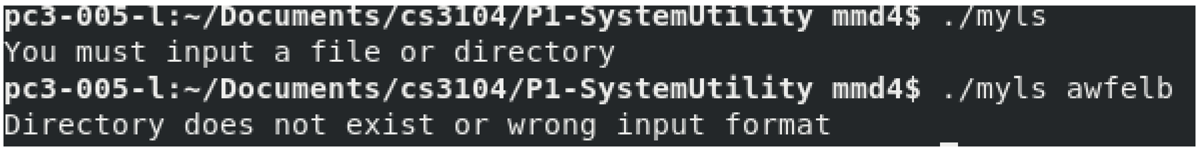
Here again, the output is the same.
Finally, we want to check pathing again but with a file that is “far away”.
We are going to travel to the cs2002 directory.



```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./myls ~/Documents/cs2002
drwxrwxr-x      8       20838   20838   13              Apr 28 20:35    .
drwx------      14      20838   20838   14              Oct 21 19:19    ..
drwxrwxr-x      4       20838   20838   6               Feb 09 15:11    .metadata
drwxr-xr-x      3       20838   20838   16              Oct 21 16:29    Practical4
-rw-r--r--      1       20838   20838   34928           Feb 13 16:25    Practical1.zip
drwxr-xr-x      3       20838   20838   4               Mar 18 19:48    ReadMyMind
-rw-r--r--      1       20838   20838   286102          Feb 27 20:21    Practical2.zip
drwx------      5       20838   20838   7               Oct 22 04:20    prac5
drwxrwxr-x      2       20838   20838   11              Feb 14 10:21    Practical1
-rw-r--r--      1       20838   20838   140172          Mar 18 19:52    ReadMyMind.zip
-rw-r--r--      1       20838   20838   546461          Apr 10 19:30    Practical4.zip
drwxrwxr-x      4       20838   20838   7               Feb 27 20:18    Practical2
-rw-r--r--      1       20838   20838   4307879         Apr 28 20:35    prac5.zip
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ls -n ~/Documents/cs2002
total 5207
drwx------ 5 20838 20838       7 Oct 22 04:20 prac5/
-rw-r--r-- 1 20838 20838 4307879 Apr 28 20:35 prac5.zip
drwxrwxr-x 2 20838 20838      11 Feb 14  2018 Practical1/
-rw-r--r-- 1 20838 20838   34928 Feb 13  2018 Practical1.zip
drwxrwxr-x 4 20838 20838       7 Feb 27  2018 Practical2/
-rw-r--r-- 1 20838 20838  286102 Feb 27  2018 Practical2.zip
drwxr-xr-x 3 20838 20838      16 Oct 21 16:29 Practical4/
-rw-r--r-- 1 20838 20838  546461 Apr 10  2018 Practical4.zip
drwxr-xr-x 3 20838 20838       4 Mar 18  2018 ReadMyMind/
-rw-r--r-- 1 20838 20838  140172 Mar 18  2018 ReadMyMind.zip
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ 
```

P1-SystemUtility 160020220

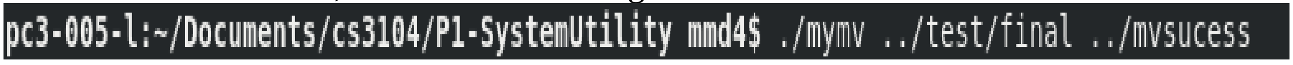
Everything is correct! The program runs perfectly when given the right inputs. Now let's check if it is solid.



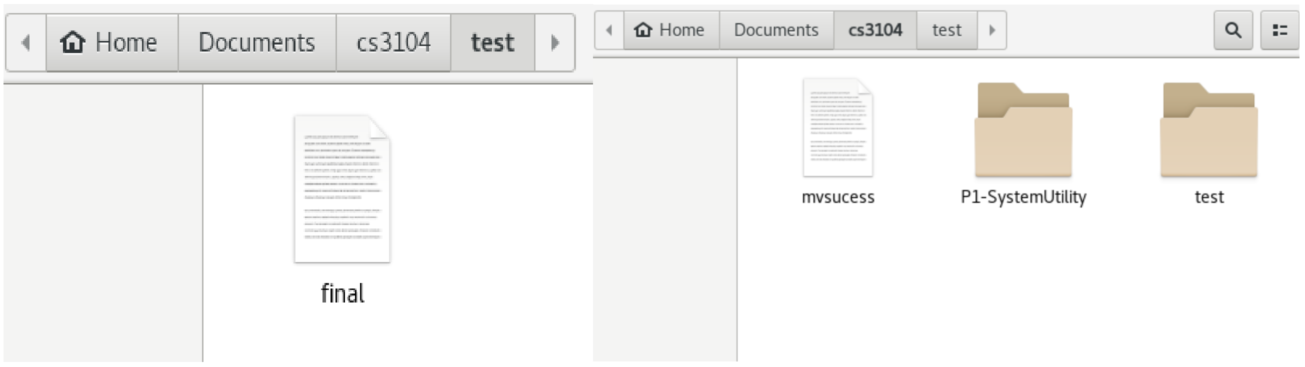
```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./myls
You must input a file or directory
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./myls awfelb
Directory does not exist or wrong input format
```

The error checking seems to work fine. The basic implementation is running perfectly!

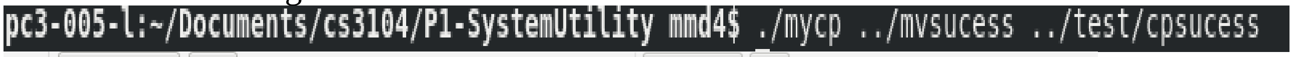
Now for the extensions, The tests are pretty straight forward.
We have a file "final.txt" in our test directory. We want to move it to the directory above and name it mvsucess. Thus, we call the following command line:



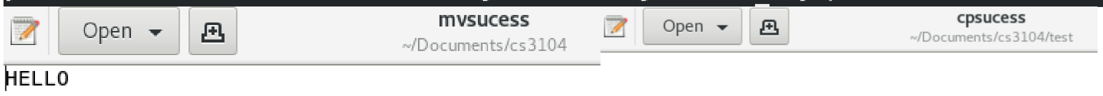
```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./mymv ../test/final ../mvsucess
```



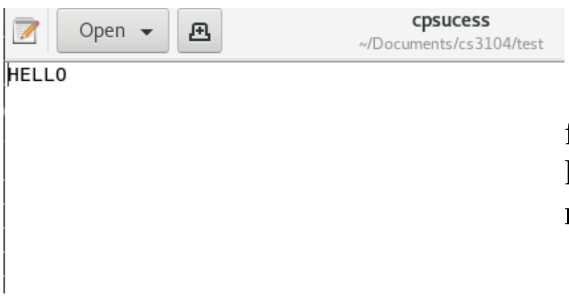
The "mv" implementation is a success. The final file is deleted from test and moved to the directory above.
Now testing the "cp" command, We will also test the writing logic here for both programs. (I have tested it for the 2 implementations and it works fine but it's not relevant to put twice the screen-shots in the report for the same writing logic)
We call the following line:



```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./mycp ../mvsucess ../test/cpsucess
```



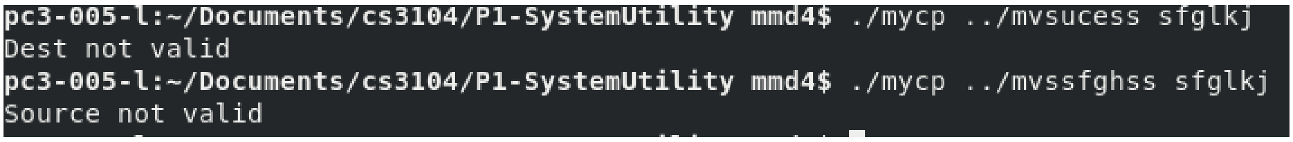
```
HELLO
```



```
HELLO
```

the copy is a success since hello is written from mvsucess to cpsucess ! The writing logic thus works fine and the file manipulation look on point.

Now we should also test the error checking on both extensions. We note that this test is common to both programs since they have the same error checking.



```
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./mycp ../mvsucess sfglkj
Dest not valid
pc3-005-l:~/Documents/cs3104/P1-SystemUtility mmd4$ ./mycp ../mvssfghss sfglkj
Source not valid
```

We see that the error checking works fine.

Note on the testing :

The testing of my programs was pretty straight-forward since I did not encounter any bugs after implementing the logics
The reason for this is that I worked a lot on logic theory to make sure that the program would be simple to debug before hard-coding the program.

Conclusion:

This practical was really interesting and pushed me to do and show some independent reading and research as I used some material from "The C programming language" by Brian Kernighan and Dennis Ritchie. Furthermore, I find the report to be insightful and give a respectable amount of detail. The program ran perfectly, so did the extensions and the error checking was good.
Finally, I found that working on the logic in this practical was really important and it worked as I feel like it made everything simple and understandable.