Part III Project description
Copy or move a file or a group of files to a destination specified by a path.

Write a C program (with modular programming) that provides the functionalities of copying and moving a file or a group of files to a destination.

- The program copies files when used as follows:
  copy source1 [source2 ...] destination

- The program moves files when used as follows:
  move source1 [source2 ...] destination

The `sources` and the `destination` are absolute path or relative path to a file/directory. The program must be able to determine the file/directory name when needed[1]. Assume that there are no spaces in a file /directory name. When the program performs moving (not copying), it copies a file to the new location then deletes the file (unlinks the old path). Coping all bytes takes time; the more efficient way would be to create a new link (the new path linked to the old bytes on a disk). I suggest you try this way first if allowed by the OS.

For example,

```
gcc prog.c -o copy; ln copy move
```
[2]

After the above command, the file with the same executable code can be run under the two different names: `copy` or `move`. The program(`main`) determines on coping or moving based on the user-entered choice. For example, if the following command is entered at the command line, `copy` is requested.

```
./copy MyFile.c NextFile.c ../backups/
```

The program should read the command line arguments[3] and determine the user-entered request.

There are two types of file links: hard link and soft (symbolic) link

- **Hard links** (each file must have at least one):
  In addition to its file name, each file in a file system has an identification number, called an inode number, that is unique in its file system. The inode number refers to the physical file, the data stored in a particular location. A file also has a device number, and the combination of its inode number and device number is unique throughout all the file systems in the hierarchical file system. There may be multiple files names that link to the same inode number(to the physical file).

- **Soft links** (not a must-have link):
  It is a link to a file whose purpose is to point to a file or directory. For example, an alias/a Window shortcut is a soft link.

Once the program has determined the request (copying or moving), it must check the destination:

- only a directory or device can be the destination for copying more than one file and
- only a directory can be the destination for moving more than one file.

An invalid destination must result in the error message and program termination. For this project, you must understand and use the system call stat()[4]. The structure `stat` has the field `stat.st_mode` that contains

the `file type` and `mode`. POSIX refers to the `stat.st_mode` bits corresponding to the mask S_IFMT (see below) as the file type, the 12 bits corresponding to the mask 07777 as the file mode bits, and the least significant 9 bits (the mask 0777) as the file permission bits. The following mask values are defined for the file type:

```
S_IFMT     0170000   bit mask for the file type bit field
S_IFSOCK   0140000   socket
S_IFLNK    0120000   symbolic link
S_IFREG    0100000   regular file
S_IFBLK    0060000   block device
S_IFDIR    0040000   directory
S_IFCHR    0020000   character device
S_IFIFO    0010000   FIFO
```

Thus, to test for a regular file (for example), one could write:

```
stat(pathname, &sb);
if((sb.st_mode & S_IFMT) == S_IFREG) {
      /* Handle regular file */
}
```

And then, the program must process each file to copy or move. If a source file does not exist, an error message must be generated.

To copy a file, you must use system calls that read and write big blocks (use BUFSIZ[5] macro).

- A file should not be copied to itself.
- If a file already exists in the destination folder, permission for overwriting should be asked.

To move a file, first link it to the new path. If it does not work (because the OS may block it or for another reason), you must copy this file and then delete the source.

- You can add this code for link error checking:

```
if( link( src, dst ) < 0 ){
    printf( "Can't link to directory %s\n", dst );
    perror( "link" );
}
```

- Use unlink for deleting a file.

Here are the sample program executions (# is the prompt):

Sample program execution 1:

```
# ./move
# Usage: move source1 [source2 ...] destination
```

Sample program execution 2:

```
# ./copy MyFile.c NextFile.c ../backups/
# MyFile.c NextFile.c succesefuly copied to ../backups
```

Some ideas on functions/others:

- A struct containing file type, device id and inode number of a file.
- A function that returns the file type when a file name is given.
- A function that returns the file/directory name when its path is given.
- A file copying function and possible helper functions.
- A file moving function and possible helper functions.
- main(int argc, char* argv)
- …

Notes:

1. Write a function that returns the file/directory name when the path is given. This C function may be useful.
   a. char *strrchr(const char *s, int c);
      The `strrchr()` function returns a pointer to the last occurrence of the character c in the string s.

2. Linux Semicolon (;)
   Two or more commands can be on the same line separated by the semicolon. All the arguments before (;) will be treated as a separate command from all the arguments after the (;). All the commands will be executed sequentially.

   ```
   Syntax: command1; command2
   ```

3. Command-line arguments
   The command line arguments are handled using main() function arguments where `argc` refers to the number (`n`) of arguments passed, and `argv[]` is a pointer array which points to each argument passed to the program. `argv[0]` holds the name of the program itself and `argv[1]` is a pointer to the first command line argument supplied, and `argv[n]` is the last argument.

4. Check `stat` system call on `man 2` page or https://man7.org/linux/man-pages/man7/inode.7.html for helpful information on how to detect the file type.

   Linux Programmer's Manual (man 2 stat)

   ```
   #include <sys/types.h>
   #include <sys/stat.h>
   #include <unistd.h>
   int stat(const char *pathname, struct stat* statbuf);
   ```

   - Write information about a file in the buffer pointed to by `statbuf`
   - `statbuf` must be allocated
   - Return zero on success or -1 on error

5. https://c-for-dummies.com/blog/?p=4711 ;
   https://www.gnu.org/software/libc/manual/html_node/Controlling-Buffering.html