Problem 1 -- Exploratory Questions

1A) A filesystem (volume) that does **not** use journalling (e.g. the ext2fs) is mounted (i.e. it is not the root volume) and under heavy use, with files and directories being created, renamed, and removed at a high rate. There is a sudden failure (e.g. a power failure) which halts the system. After the system reboots, the system's initialization scripts will want to mount the volume.

i) Before the volume can be mounted, what utility program will be run? What does that utility do, and what sort of issues/problems do we expect it will find and have to correct? Give at least one specific example.

ii) If the volume had previously been mounted in a non-cached manner (`-osync` option to the `mount` command), would this avoid these problems? Why / why not?

1B) A user has a substantial collection of 4K-quality videos and purchases an "8 TB" internal SATA hard disk to hold them. The disk is formatted as a single EXT4 volume. The user follows best security practices and does not download or watch the videos as the "root" user (superuser). Each video file is exactly 2^^30 (1073741824 bytes) long. Therefore, the user expected that the entire collection of nearly 8,000 videos would fit but in reality the volume ran out of disk space with substantially fewer videos. Give (at least) two reasons for this bitter disappointment.

1C) A user runs the command
`mv /A/B/F1 /A/C/F2`
where `F1` is a fairly large file. This command runs very quickly. Then the user runs
`mv /A/C/F2 /A/Z/F3`
but this takes quite a while to run. Why might that be the case?

1D) At exactly 4PM on Sep 24, 2025, (i.e. when you will have started this assignment) we create a new file with the touch command:
`touch newfile.txt`

At 4:15 PM, we run the following command:
`echo "123" >newfile.txt`

Then at 4:30 PM, we use the touch command to make it appear that we started this assignment earlier:
`touch -m -t 09182230.10 newfile.txt`

And finally at 4:55 PM, we run
`cat newfile.txt`
and turn in the assignment.

If we were to `stat` the file now, what values would we see for
`st_size`
`st_nlink`
`st_mode`
`st_atime`
`st_mtime`
`st_ctime`
Note: Assume the default values for umask and default mount options (i.e. the `noatime` or `relatime` options are

not present). You should express the timestamps as ordinary dates/times in the local timezone. There is no need to translate them into UNIX integer format.

Problem 2 -- EXT2/EXT3 block allocation

A file is newly created and filled with data as follows:

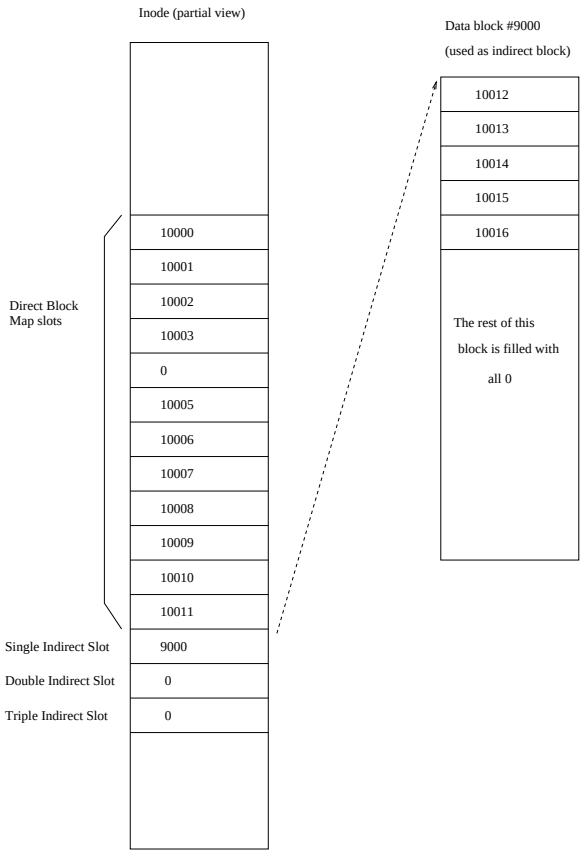
```
/* Assume all system calls are successful and that bufferA,B,C, etc. are valid*/
fd=open("thefile",O_CREAT|O_TRUNC|O_WRONLY,0666);
write(fd,bufferA,1024);
write(fd,bufferB,1024);
lseek(fd,4096,SEEK_SET);
write(fd,bufferC,0xB000);
lseek(fd,0xD000,SEEK_SET);
write(fd,bufferD,01000);
lseek(fd,0x40C123,SEEK_SET);
write(fd,"ZZZ",3);
close(fd);
```

2A) Draw the data block sections of the inode. Make sure to place a value (the block number, in decimal) in each of the 15 slots. Where indirect, double-indirect, or triple-indirect blocks are used, show these as well, and fill in the exact values within those indirect blocks where the values are non-zero. However, you don't need to write 1024 entries for each block! Just state the offset(s) of the non-zero entries and be clear if you are stating the offset in hex vs. decimal.

Below is a sample diagram but it is not at all the correct answer.

2B) What is the value of `st_size` and `st_blocks`?

Assume that:
a) This is an EXT3 filesystem with default settings. The data block size is 4K. Block numbers are 32 bits.
b) Blocks are allocated in sequential order, starting with the first free data block #42 (decimal).
c) Indirect blocks are allocated and initialized prior to allocating the direct blocks to which they point.



Problem 3 -- Simplefind

The `find` utility program on UNIX systems is a powerful tool which, at its heart, explores the filesystem tree by recursive descent. As it visits each inode, it can make decisions based on a variety of filters and then take a variety of actions when the filter(s) matches the node. For example:

```
find /a/b -type f -mtime -7 -print
```

This starts the recursive descent at `/a/b` and applies a filter to each node, matching only those which have inode type S_IFREG. Additionally, a second filter is applied matching only those nodes whose mtime is less than 7 days old. Lastly, the action taken is to print the full pathname of each matching node.

This is a very powerful command, with a syntax that is often confusing. While you are encouraged to explore it fully, for this problem we will be coding up a very simplified version of the command:

```
simplefind [-l] [-x] [-n pattern] [starting_path]
```

Start at the given `starting_path` (if not specified, default to .) and recurse. If the `-n` option is present, it is followed by a pattern which can contain shell wildcards, equivalent to the `-name` option to find. The `-x` flag is the same as the `-xdev` option to find: it does not recurse into parts of the filesystem which are on a different mounted volume from starting_path. The `-l` flag causes simplefind to output a line for each applicable node (subject to the above two filters) which is similar to the `-ls` option to find, i.e. a verbose listing. If that flag is absent, simplefind will instead do the equivalent of the `-print` option to find and just print pathnames.

Here is an example of invocation and output

```
$ ./simplefind -l testdir
  3568810      4 drwxr-xr-x   3  hak      users        4096 Sep 16 21:33 testdir
  3575071      0 brw-r--r--   1  root     root       11,   1 Sep 16 21:25 testdir/bnode
  3575066      4 drwxr-xr-t   2  hak      users        4096 Sep 16 21:22 testdir/d2
  3575067      8 -rw-r--r--   1  hak      users        8192 Sep 16 21:30 testdir/d2/f2.c
  3575068      4 -rw-r--r--   2  hak      users        2048 Sep 16 21:31 testdir/d2/.f3.c
  3575068      4 -rw-r--r--   2  hak      users        2048 Sep 16 21:31 testdir/d2/f4.c
  3575065      4 -rw-r--r--   1  hak      users           4 Sep 16 21:30 testdir/f1.c
  3575070      0 crw-r--r--   1  root     root        4,   1 Sep 16 21:24 testdir/cnode
  3575064      0 lrwxrwxrwx   1  hak      users          17 Sep 16 21:22 testdir/testlink ->
                   ../../testlinktgt
  3575072      4 -r-s---r-x   1      9999    12345        4 Sep 16 21:34 testdir/alien
```

Each line represents one inode (that has passed any applicable filters). The format (identical to the `find -ls`) is:

- The inode number
- The space consumption, actual disk blocks consumed, but in units of 1K. This information is from the st_blocks field.
- The inode mode, in human-friendly format. Read the ls documentation for full information about all the possible combinations.
- The link count
- The owner of the node. This should be represented as an actual username, if available from getpwuid, otherwise an integer.
- The group owner of the node. Likewise, a name if possible, otherwise an integer.
- The size in bytes of the node. For device inodes (BLK or CHR) this should instead be the st_rdev field represented as major,minor
- The mtime of the node, in the same format as ls. Note that timestamps are kept in UTC (GMT), but they are presented according to the local timezone.
- The full pathname of the file. If starting_path was a relative pathname, this should include starting_path but does not have to include the full absolute pathname.
- If and only if the link type is symlink, output a -> after the pathname and output the contents of the symlink (use the readlink system call)

Note that directories in UNIX are not sorted. The find command (and your simplefind) will return directory entries in whatever order the kernel returns them. Therefore the recursion will not be a strict depth-first. As you see a subdirectory, you recurse into it, making no attempt to sort the entries, nor to explore the subdirectories either first or last.

Note that the . and .. entries are not printed out. The only exception is the information about the starting_path is always printed (unless it doesn't match the filter). The example above includes a character device, a block device, a hard link, a symlink, a node with no numeric->name translation for the uid/gid, and an assortment of modes.

Here is an example using the filter:

```
$ ./simplefind -n '*.c' testdir
testdir/d2/f2.c
testdir/d2/.f3.c
testdir/d2/f4.c
testdir/f1.c
```

The -n argument here contains wildcards. This must be quoted using single or double quotes, otherwise the shell which invokes your simplefind program will match any instances of *.c in the current directory, leading to a syntax error. The quotes tell the shell to avoid wildcard expansion in that argument. The quotes are stripped off by the shell before they are received in an argument to your program. Note that in this example, testdir itself is not printed out, because it does not match the filter.

It is suggested that you use the `fnmatch` library function to do pattern matching. Use the opendir/readdir/closedir library functions for directory exploration, and do the recursion yourself. Don't try to use a library function such as ftw. Don't "shell out" to find, or other such cheat codes.

Testing

You should test your code as much as possible. Some test cases such as devices may be difficult for you to test depending on your environment. Since your program is very similar to find, you could test it as follows:

```
./simplefind -l -x -n '*.c' startdir >/tmp/f1
find startdir -xdev -name '*.c' -ls >/tmp/f2
diff /tmp/f[12]
```

If your output isn't exactly the same as find (such as column alignment / spacing) the diff will fail at every line. The case without the -l option is much easier to match up. It isn't a requirement that you perform this diff. If you can't, you should instead compare the two outputs by hand, line-for-line, and reconcile differences that are significant (not just format).

Submission

Submit (obviously) your code. Also submit demonstrations of your program being tested with various test cases. These will preferably be text captures (e.g. with I/O redirection) but if they are raster screen shots, they must be in PNG, JPG, or PDF format, and they must appear large enough to actually read the text!