CS 332/532 Systems Programming

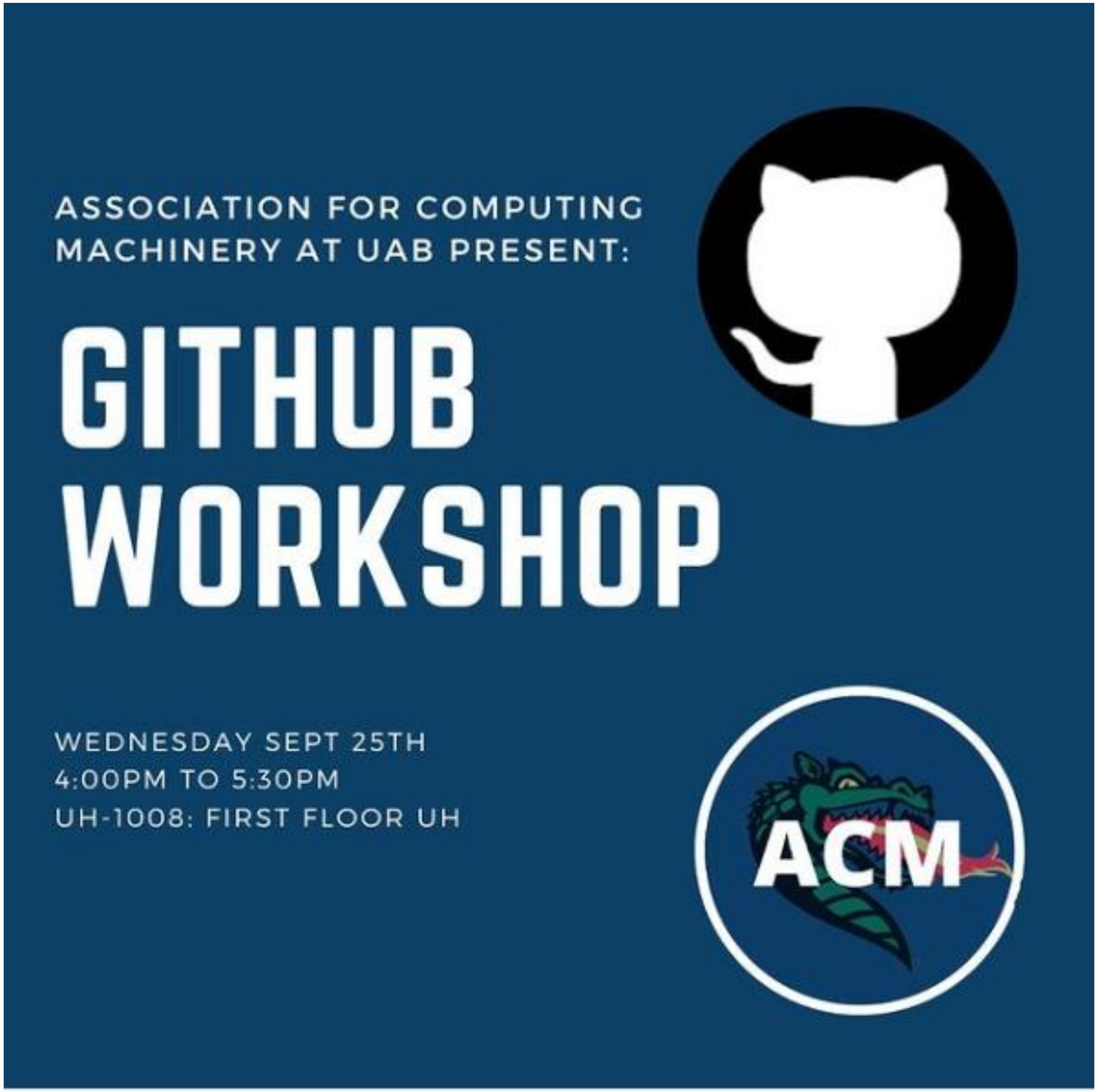
Lecture 13

-Unix Files & Directories-

Professor : Mahmut Unan – UAB CS

Agenda

Unix Files & Directories
stat lstat functions
File Types
Open & Read Directories



ASSOCIATION FOR COMPUTING MACHINERY AT UAB PRESENT:

GITHUB WORKSHOP

WEDNESDAY SEPT 25TH
4:00PM TO 5:30PM
UH-1008: FIRST FLOOR UH

ACM

stat lstat

- We’ll start with the `stat` and `lstat` functions.
- Both function return a structure called *stat*, and members of stat structure provide information about the file or directory which was provided as the argument to these functions.

stat lstat

- stat, fstat, lstat, fstatat - get file status

```
int stat(const char *pathname, struct stat
*statbuf);

int fstat(int fd, struct stat *statbuf);

int lstat(const char *pathname, struct stat
*statbuf);
```

https://man7.org/linux/man-pages/man2/stat.2.html

- `stat()` and `fstatat()` retrieve information about the file pointed to by *pathname*

- `lstat()` is identical to `stat()`, except that if *pathname* is a symbolic link, then it returns information about the link itself, not the file that the link refers to.

- `fstat()` is identical to `stat()`, except that the file about which information is to be retrieved is specified by the file descriptor *fd*.

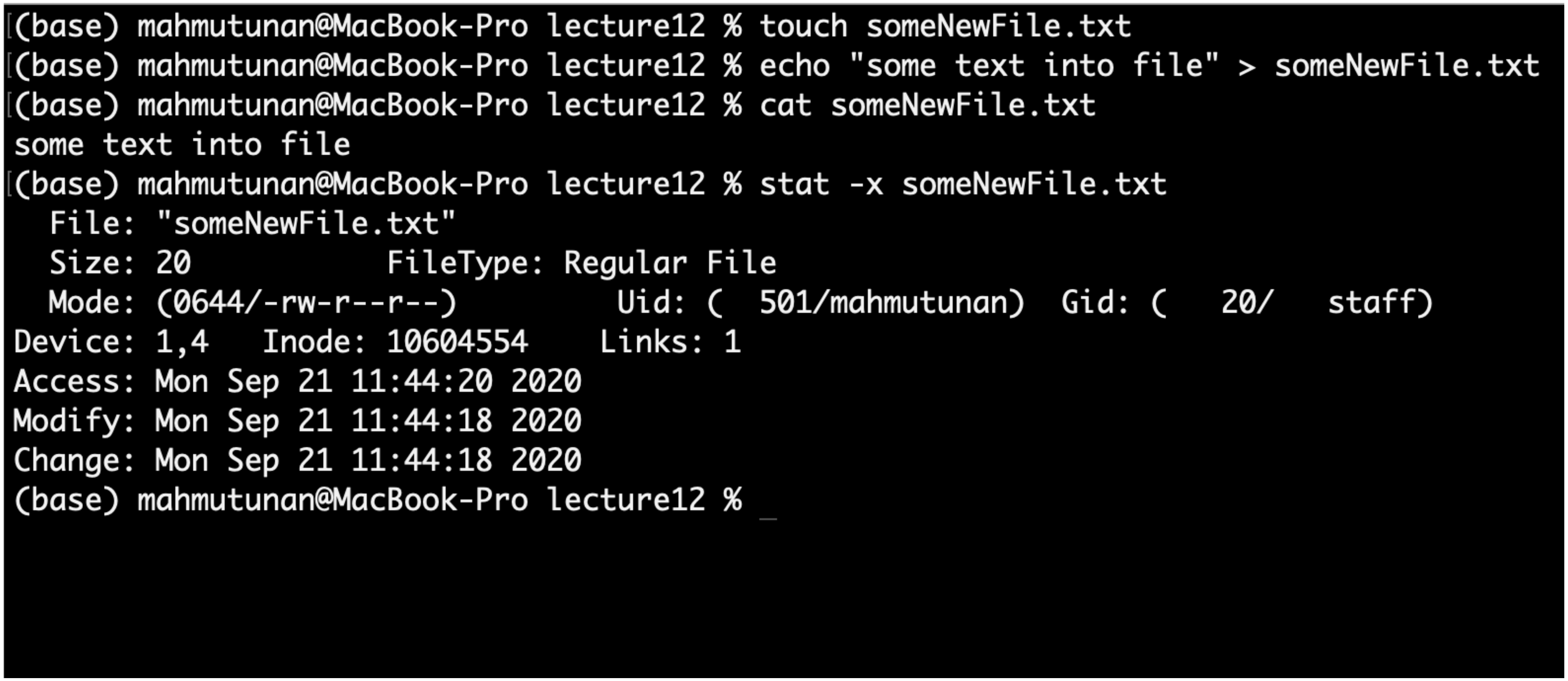
https://man7.org/linux/man-pages/man2/stat.2.html

The `stat` structure

- All of these system calls return a *stat structure*, which contains the following fields:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    nlink_t   st_nlink;       /* Number of hard links */
    uid_t     st_uid;         /* User ID of owner */
    gid_t     st_gid;         /* Group ID of owner */
    dev_t     st_rdev;        /* Device ID (if special file) */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

    /* Since Linux 2.6, the kernel supports nanosecond
       precision for the following timestamp fields.
       For the details before Linux 2.6, see NOTES. */

    struct timespec st_atim;  /* Time of last access */
    struct timespec st_mtim;  /* Time of last modification */
    struct timespec st_ctim;  /* Time of last status change */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

https://man7.org/linux/man-pages/man2/stat.2.html

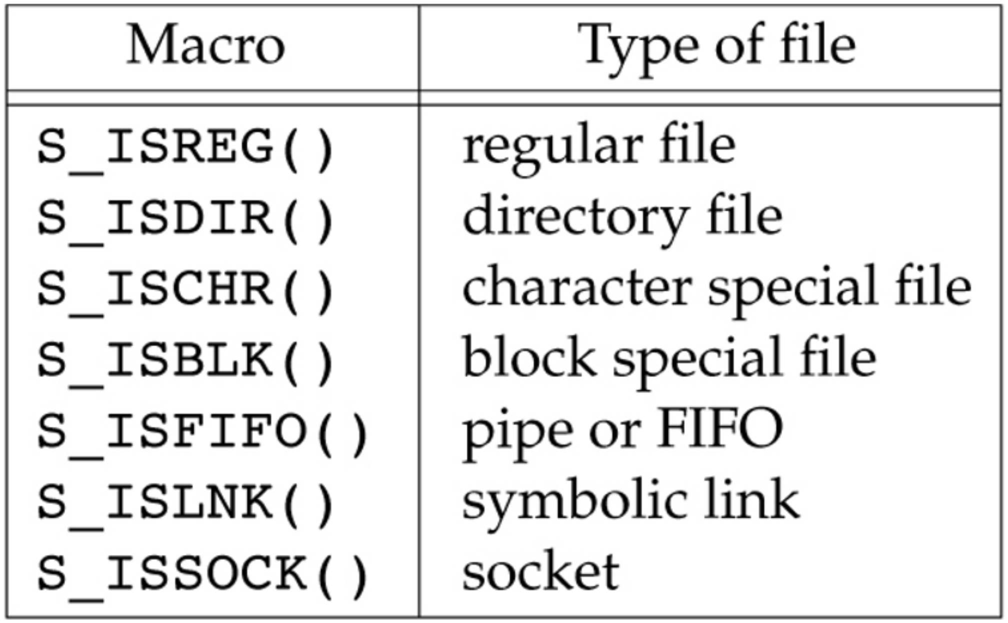


```
(base) mahmutunan@MacBook-Pro lecture12 % touch someNewFile.txt
(base) mahmutunan@MacBook-Pro lecture12 % echo "some text into file" > someNewFile.txt
(base) mahmutunan@MacBook-Pro lecture12 % cat someNewFile.txt
some text into file
(base) mahmutunan@MacBook-Pro lecture12 % stat -x someNewFile.txt
  File: "someNewFile.txt"
  Size: 20           FileType: Regular File
  Mode: (0644/-rw-r--r--)         Uid: (  501/mahmutunan)  Gid: (   20/   staff)
Device: 1,4   Inode: 10604554    Links: 1
Access: Mon Sep 21 11:44:20 2020
Modify: Mon Sep 21 11:44:18 2020
Change: Mon Sep 21 11:44:18 2020
(base) mahmutunan@MacBook-Pro lecture12 % 
```

File Types

- Regular File
- Directory File
- Block Special File
- Character Special File
- FIFO
- Socket
- Symbolic links

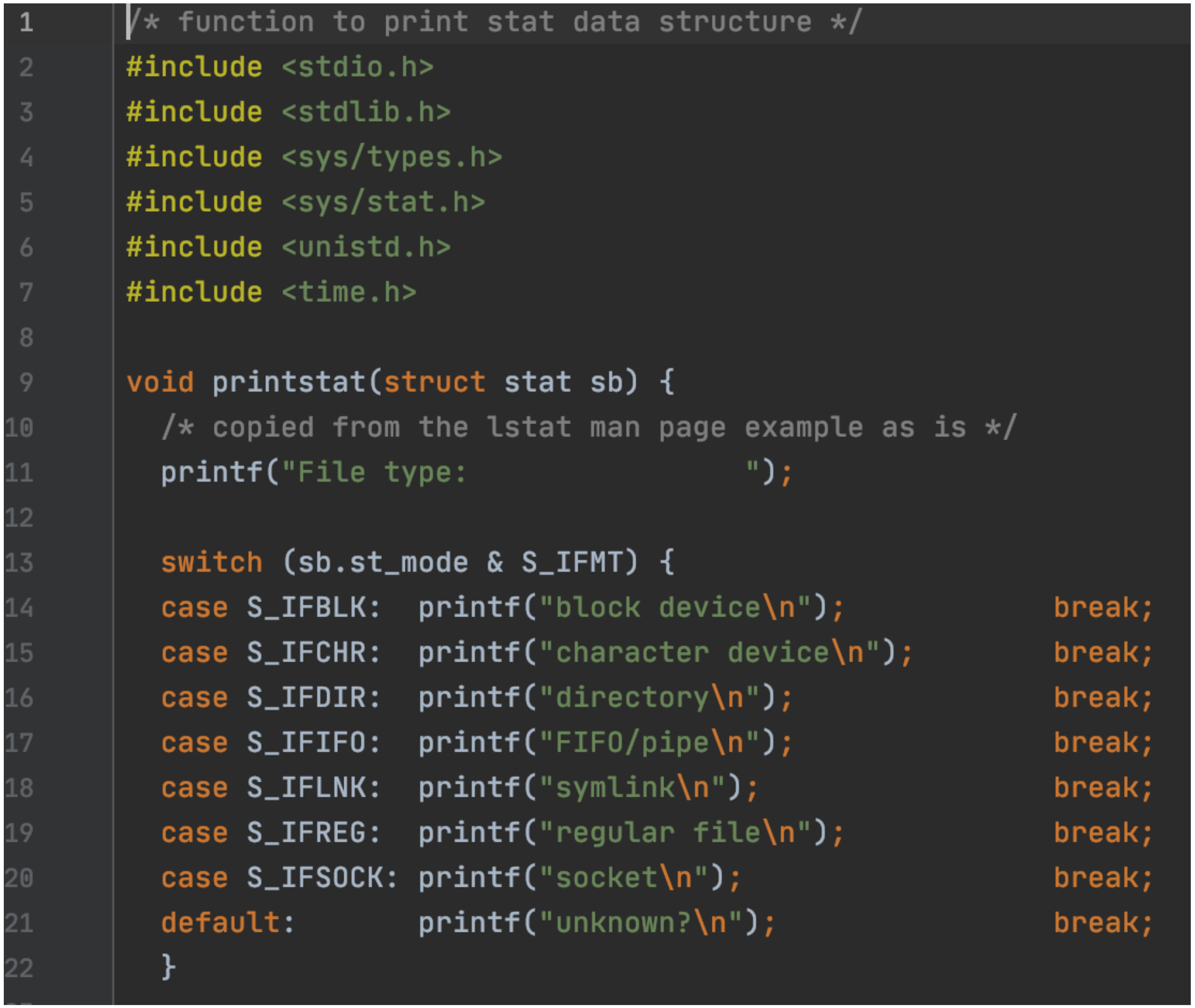
The type of the file



Macro	Type of file
`S_ISREG()`	regular file
`S_ISDIR()`	directory file
`S_ISCHR()`	character special file
`S_ISBLK()`	block special file
`S_ISFIFO()`	pipe or FIFO
`S_ISLNK()`	symbolic link
`S_ISSOCK()`	socket

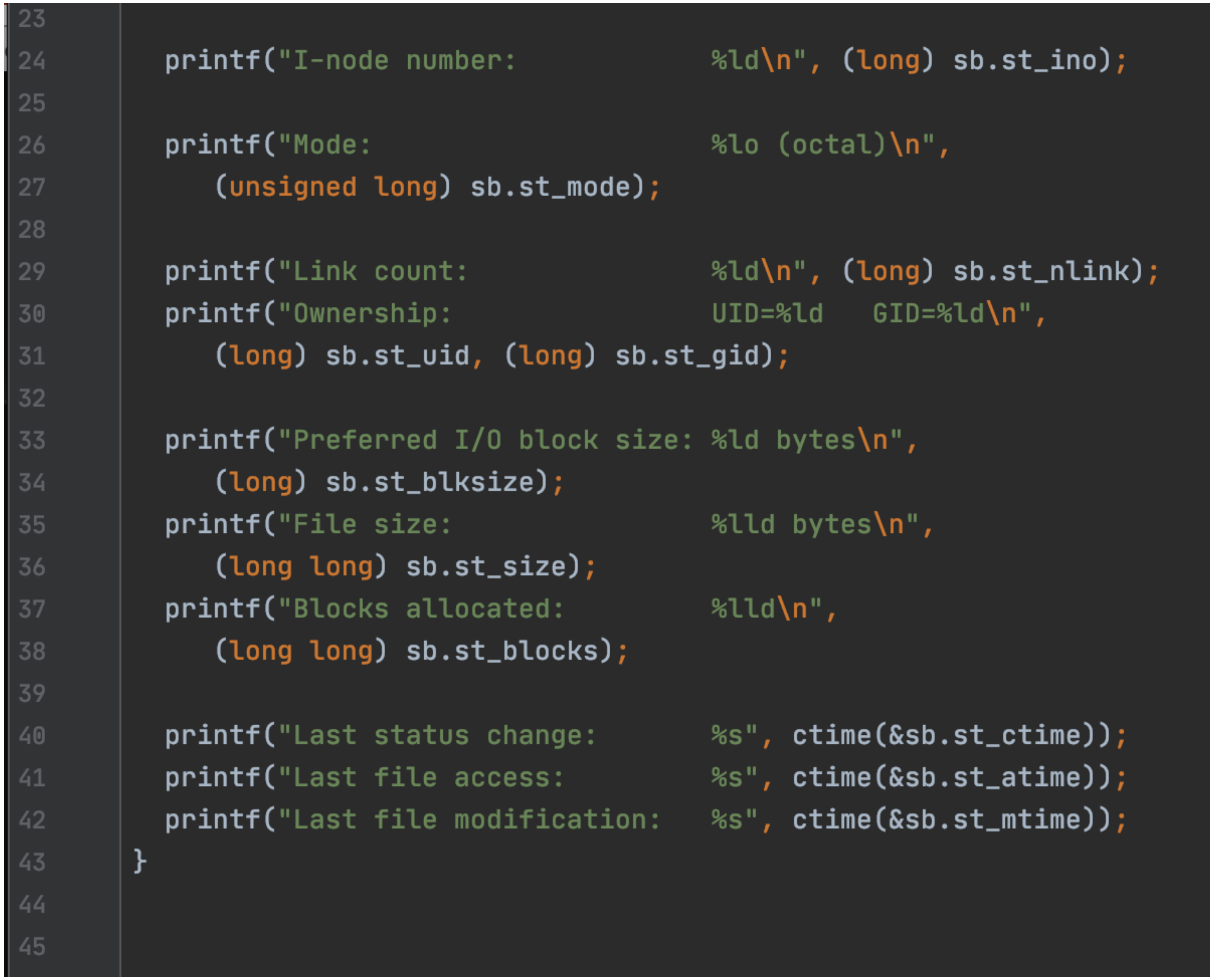
Figure 4.1 File type macros in `<sys/stat.h>`

Exercise 1 - printstat.c



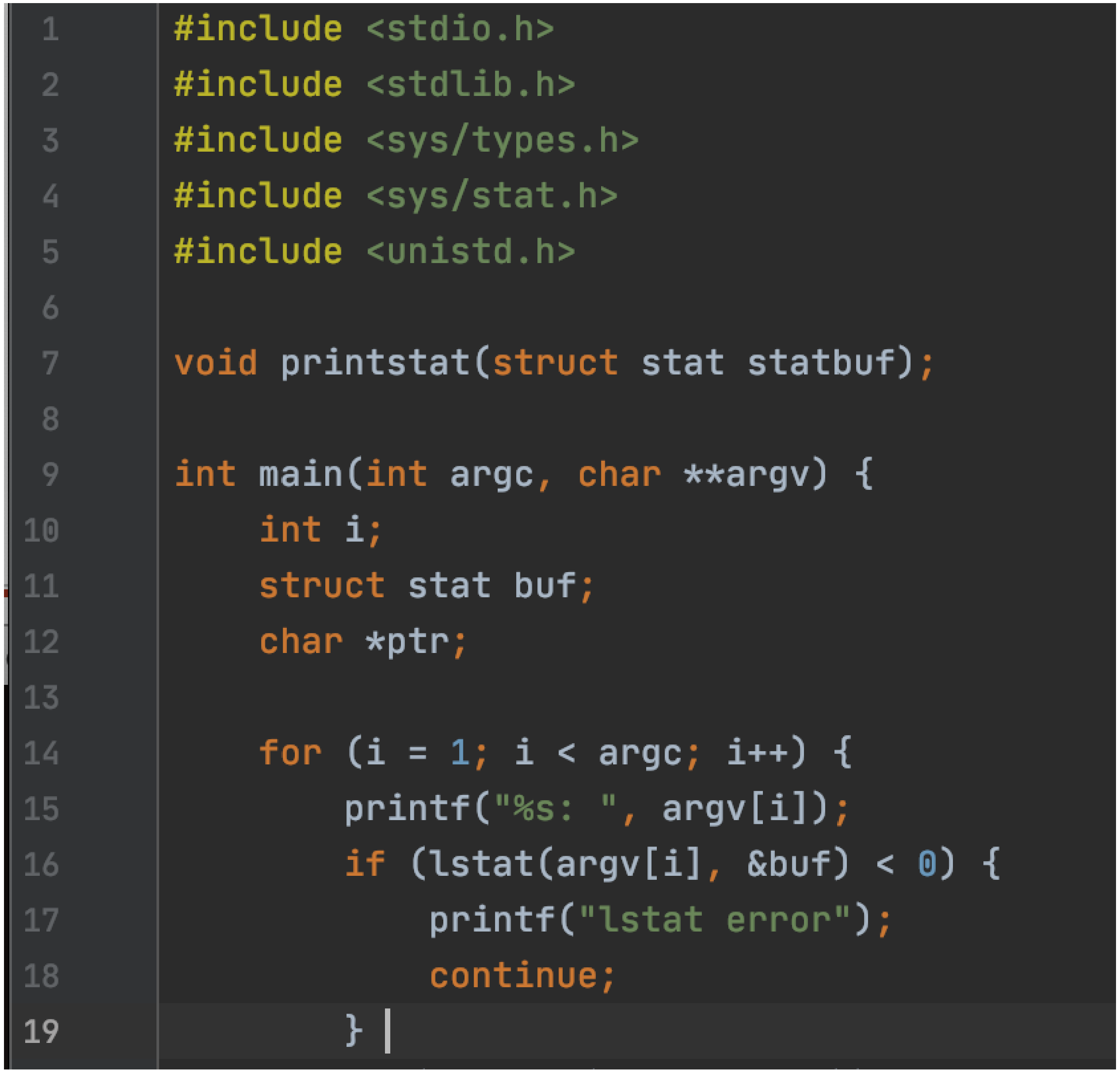
```c
/* function to print stat data structure */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>

void printstat(struct stat sb) {
  /* copied from the lstat man page example as is */
  printf("File type:                ");

  switch (sb.st_mode & S_IFMT) {
  case S_IFBLK:  printf("block device\n");            break;
  case S_IFCHR:  printf("character device\n");        break;
  case S_IFDIR:  printf("directory\n");               break;
  case S_IFIFO:  printf("FIFO/pipe\n");               break;
  case S_IFLNK:  printf("symlink\n");                 break;
  case S_IFREG:  printf("regular file\n");            break;
  case S_IFSOCK: printf("socket\n");                  break;
  default:       printf("unknown?\n");                break;
  }
```

Exercise 1 - printstat.c /2



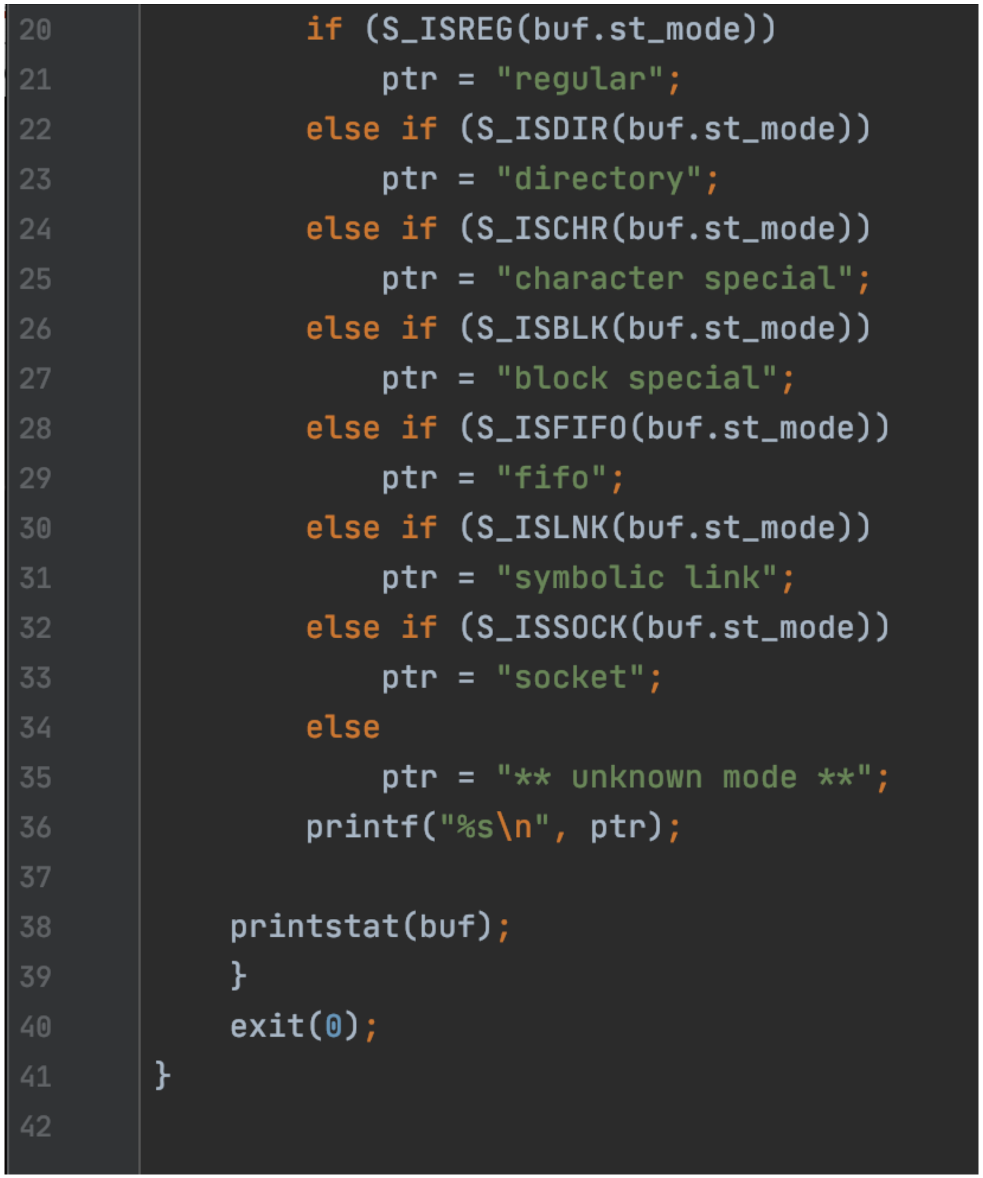
```

    printf("I-node number:              %ld\n", (long) sb.st_ino);

    printf("Mode:                       %lo (octal)\n",
        (unsigned long) sb.st_mode);

    printf("Link count:                 %ld\n", (long) sb.st_nlink);
    printf("Ownership:                  UID=%ld   GID=%ld\n",
        (long) sb.st_uid, (long) sb.st_gid);

    printf("Preferred I/O block size: %ld bytes\n",
        (long) sb.st_blksize);
    printf("File size:                  %lld bytes\n",
        (long long) sb.st_size);
    printf("Blocks allocated:           %lld\n",
        (long long) sb.st_blocks);

    printf("Last status change:         %s", ctime(&sb.st_ctime));
    printf("Last file access:           %s", ctime(&sb.st_atime));
    printf("Last file modification:     %s", ctime(&sb.st_mtime));
}


```

Exercise 1 - lstat.c



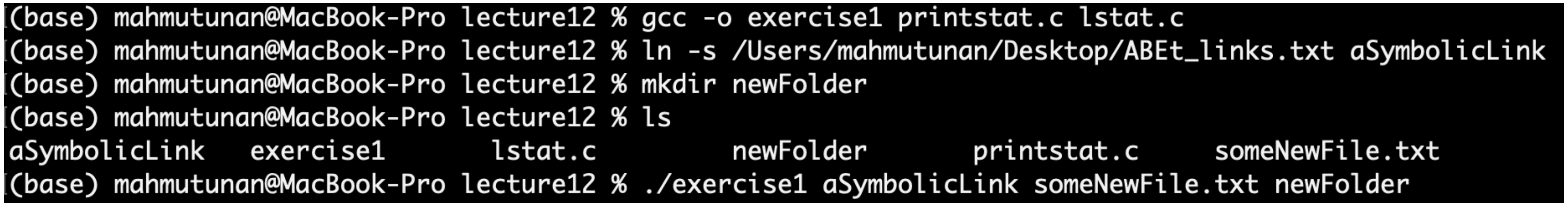
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

void printstat(struct stat statbuf);

int main(int argc, char **argv) {
    int i;
    struct stat buf;
    char *ptr;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            printf("lstat error");
            continue;
        }
```

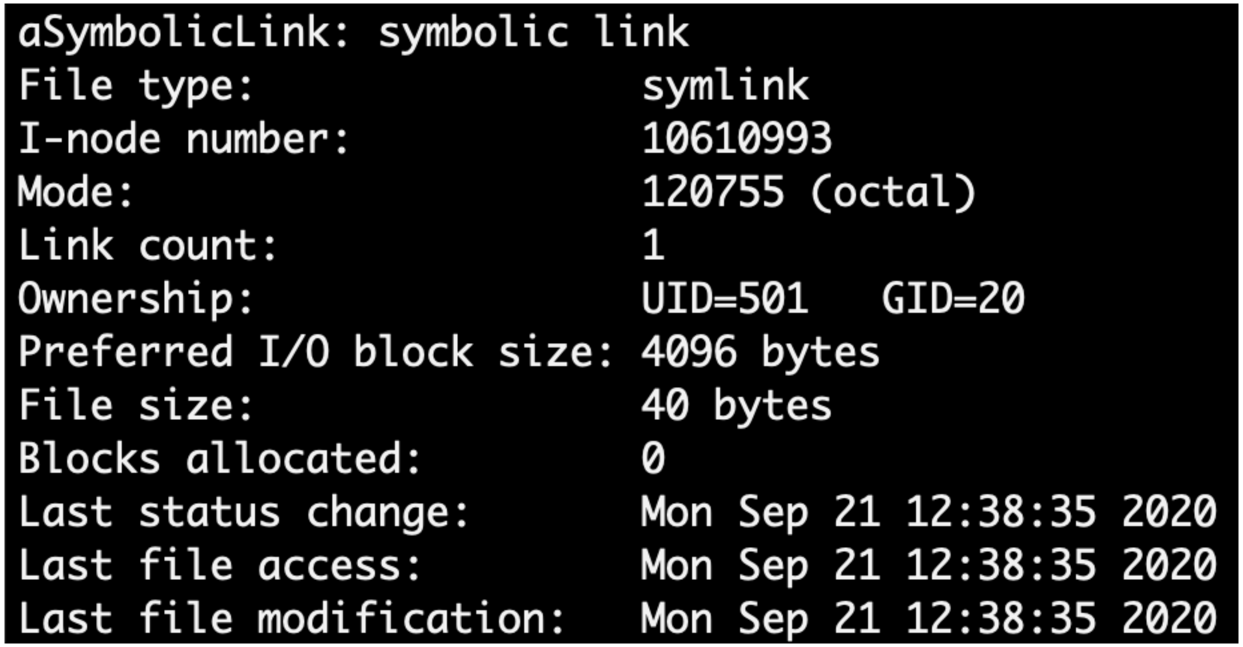
Exercise 1 - lstat.c /2



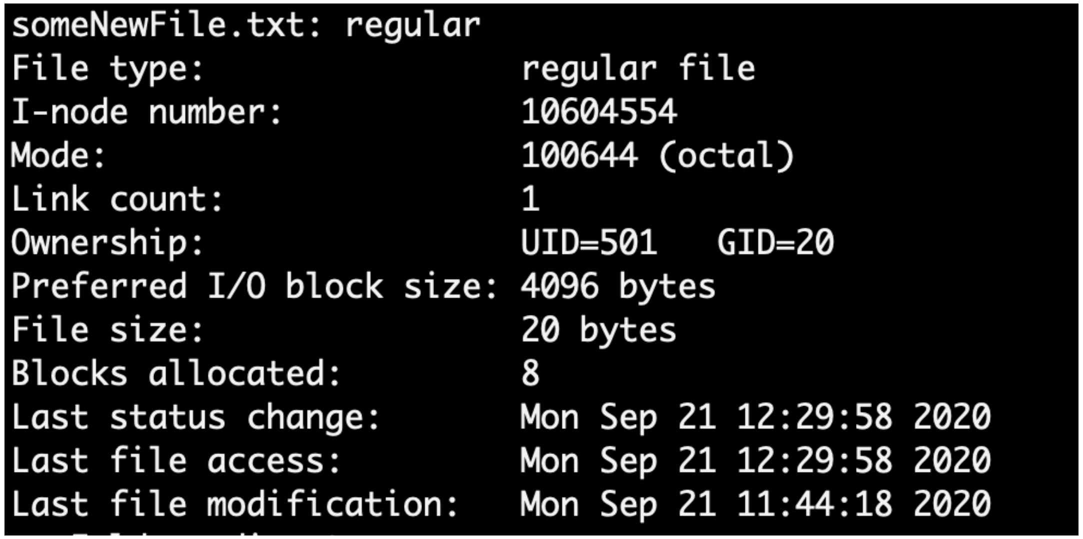
```
        if (S_ISREG(buf.st_mode))
            ptr = "regular";
        else if (S_ISDIR(buf.st_mode))
            ptr = "directory";
        else if (S_ISCHR(buf.st_mode))
            ptr = "character special";
        else if (S_ISBLK(buf.st_mode))
            ptr = "block special";
        else if (S_ISFIFO(buf.st_mode))
            ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))
            ptr = "symbolic link";
        else if (S_ISSOCK(buf.st_mode))
            ptr = "socket";
        else
            ptr = "** unknown mode **";
        printf("%s\n", ptr);

    printstat(buf);
    }
    exit(0);
}

```

Example 1 -compile&run

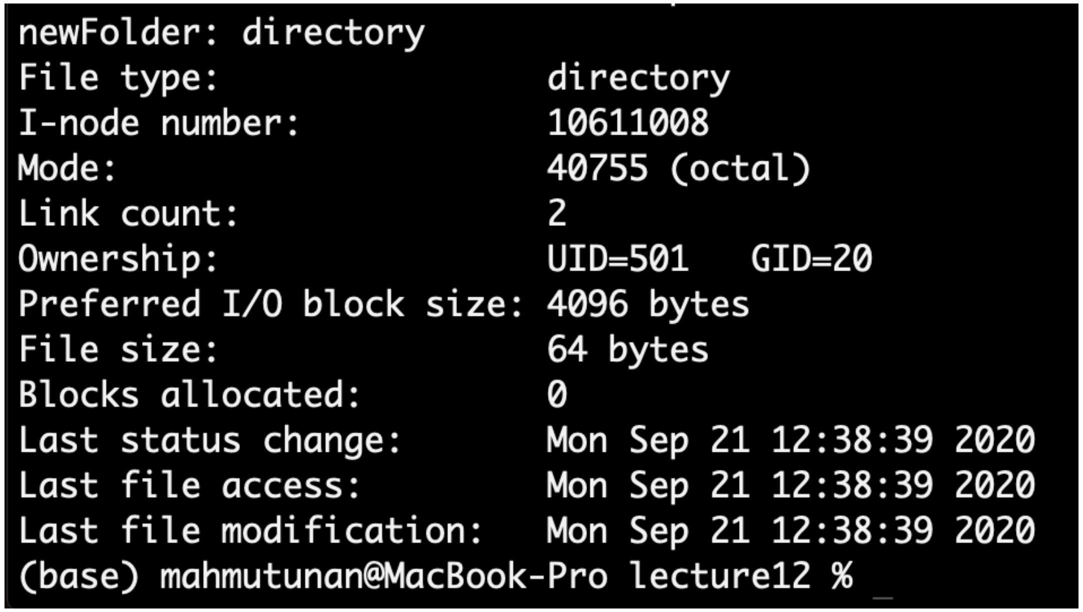

```
(base) mahmutunan@MacBook-Pro lecture12 % gcc -o exercise1 printstat.c lstat.c
(base) mahmutunan@MacBook-Pro lecture12 % ln -s /Users/mahmutunan/Desktop/ABEt_links.txt aSymbolicLink
(base) mahmutunan@MacBook-Pro lecture12 % mkdir newFolder
(base) mahmutunan@MacBook-Pro lecture12 % ls
aSymbolicLink   exercise1       lstat.c         newFolder       printstat.c     someNewFile.txt
(base) mahmutunan@MacBook-Pro lecture12 % ./exercise1 aSymbolicLink someNewFile.txt newFolder
```


```
aSymbolicLink: symbolic link
File type:                symlink
I-node number:            10610993
Mode:                     120755 (octal)
Link count:               1
Ownership:                UID=501   GID=20
Preferred I/O block size: 4096 bytes
File size:                40 bytes
Blocks allocated:         0
Last status change:       Mon Sep 21 12:38:35 2020
Last file access:         Mon Sep 21 12:38:35 2020
Last file modification:   Mon Sep 21 12:38:35 2020
```

Example 1 -compile&run /2



```
someNewFile.txt: regular
File type:                regular file
I-node number:            10604554
Mode:                     100644 (octal)
Link count:               1
Ownership:                UID=501   GID=20
Preferred I/O block size: 4096 bytes
File size:                20 bytes
Blocks allocated:         8
Last status change:       Mon Sep 21 12:29:58 2020
Last file access:         Mon Sep 21 12:29:58 2020
Last file modification:   Mon Sep 21 11:44:18 2020
```



```
newFolder: directory
File type:                directory
I-node number:            10611008
Mode:                     40755 (octal)
Link count:               2
Ownership:                UID=501   GID=20
Preferred I/O block size: 4096 bytes
File size:                64 bytes
Blocks allocated:         0
Last status change:       Mon Sep 21 12:38:39 2020
Last file access:         Mon Sep 21 12:38:39 2020
Last file modification:   Mon Sep 21 12:38:39 2020
(base) mahmutunan@MacBook-Pro lecture12 % _
```

Open & Read the directories

- Till now we talked about how filesystem store files and directories information and access details. Now it’s time to learn how to open and read the directories and traverse the file system. To achieve this task, you need to learn about three functions:
- *opendir* - this function will allow us to open a directory with the given path
- *readdir* - this function will read what’s inside the directory
- *closedir* - this will close the open directory

opendir

- opendir, fdopendir - open a directory

```
DIR *opendir(const char *name);
DIR *fdopendir(int fd);
```

The `opendir()` function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

The `fdopendir()` function is like opendir(), but returns a directory stream for the directory referred to by the open file descriptor *fd*. After a successful call to fdopendir(), *fd* is used internally by the implementation, and should not otherwise be used by the application.

https://man7.org/linux/man-pages/man3/opendir.3.html

readdir

- readdir - read a directory

```
struct dirent *readdir(DIR *dirp);
```

The readdir() function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by dirp.

On success, readdir() returns a pointer to a dirent structure. (This structure may be statically allocated; do not attempt to free(3) it.)

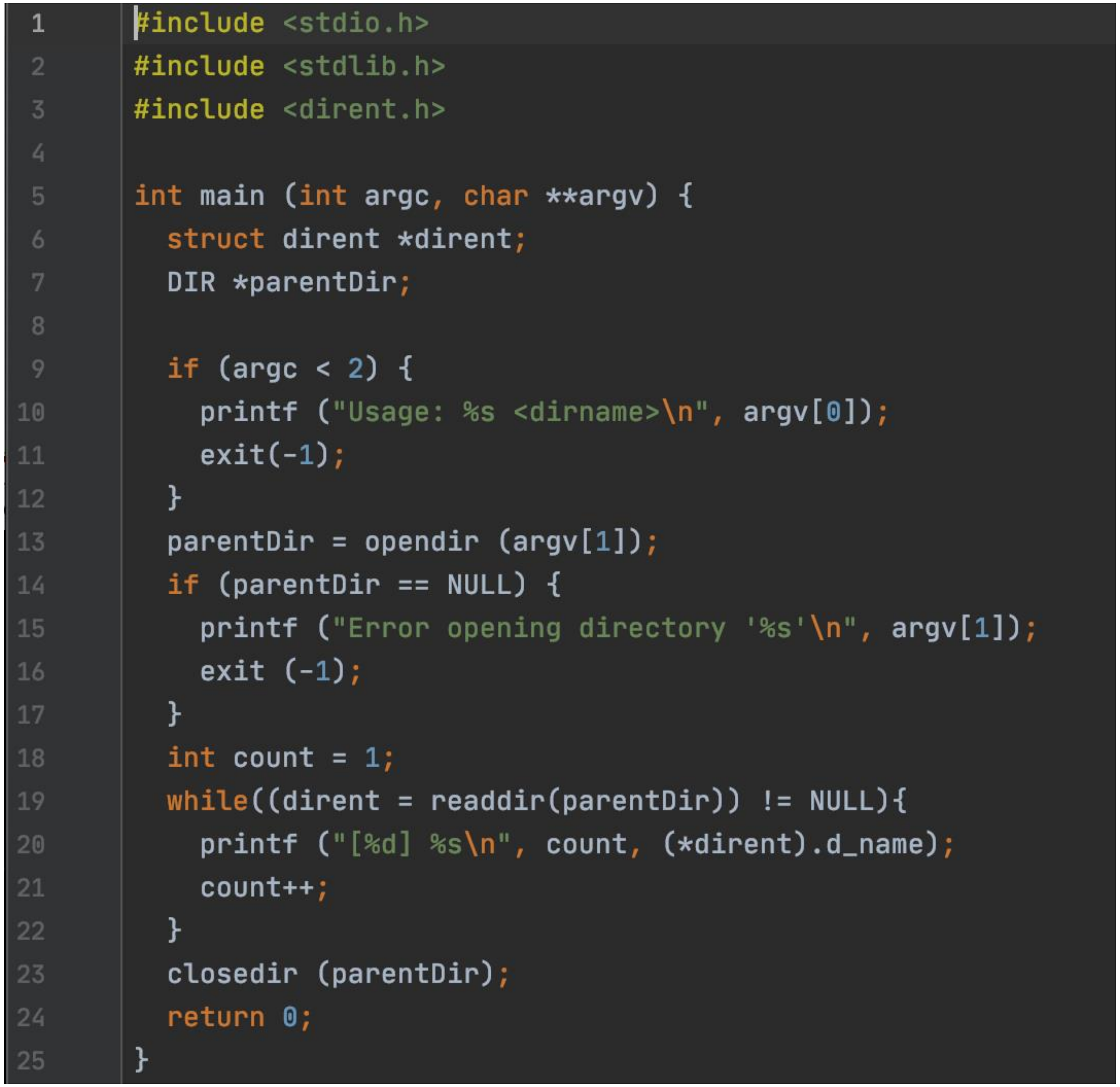
closedir

- closedir — close a directory stream

```
int closedir(DIR *dirp);
```

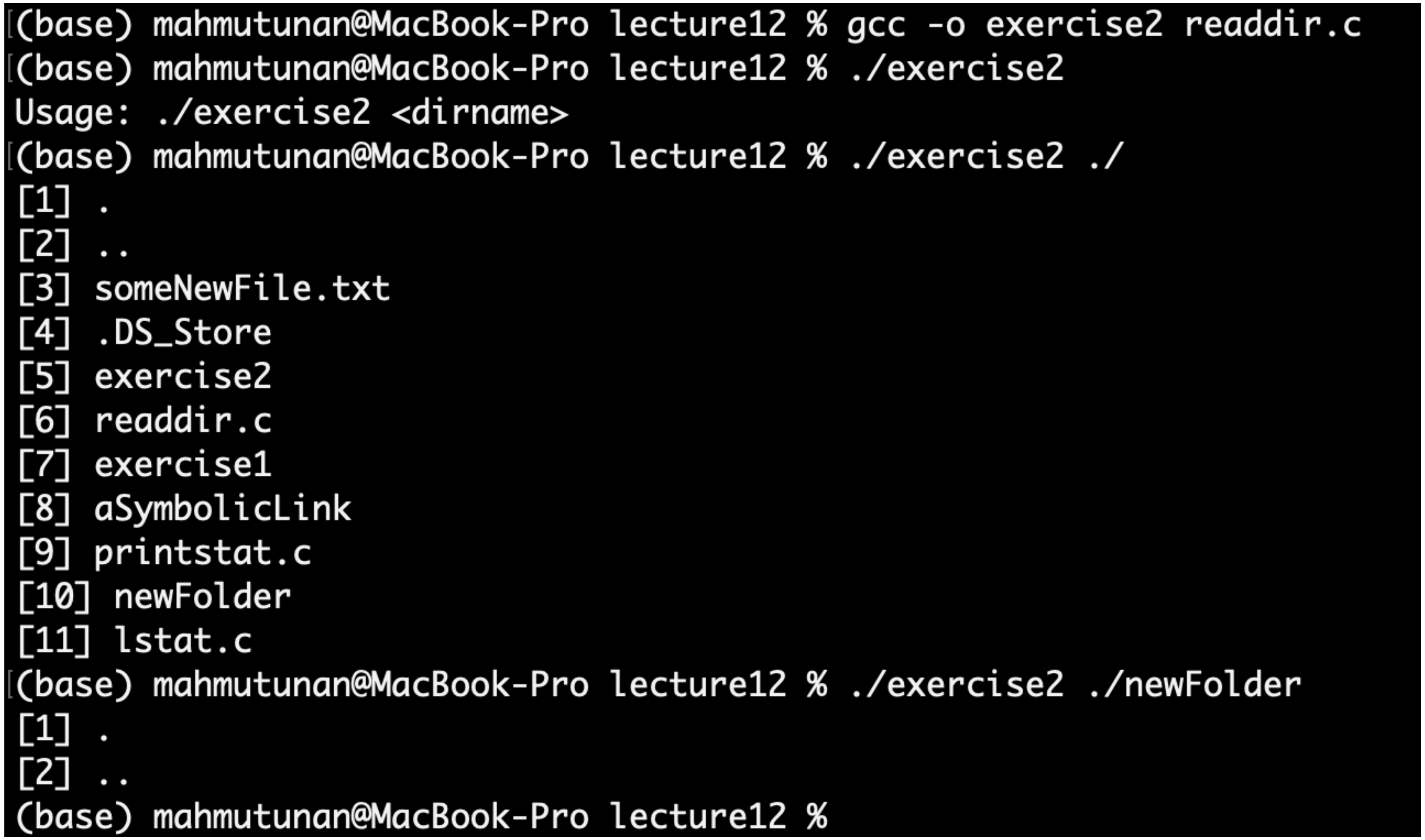
- The closedir() function shall close the directory stream referred to by the argument dirp. Upon return, the value of dirp may no longer point to an accessible object of the type DIR. If a file descriptor is used to implement type DIR, that file descriptor shall be closed.

Exercise 2 - readdir.c



```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main (int argc, char **argv) {
  struct dirent *dirent;
  DIR *parentDir;

  if (argc < 2) {
    printf ("Usage: %s <dirname>\n", argv[0]);
    exit(-1);
  }
  parentDir = opendir (argv[1]);
  if (parentDir == NULL) {
    printf ("Error opening directory '%s'\n", argv[1]);
    exit (-1);
  }
  int count = 1;
  while((dirent = readdir(parentDir)) != NULL){
    printf ("[%d] %s\n", count, (*dirent).d_name);
    count++;
  }
  closedir (parentDir);
  return 0;
}
```

Exercise 2 - compile & run



```
[(base) mahmutunan@MacBook-Pro lecture12 % gcc -o exercise2 readdir.c
[(base) mahmutunan@MacBook-Pro lecture12 % ./exercise2
Usage: ./exercise2 <dirname>
[(base) mahmutunan@MacBook-Pro lecture12 % ./exercise2 ./
[1] .
[2] ..
[3] someNewFile.txt
[4] .DS_Store
[5] exercise2
[6] readdir.c
[7] exercise1
[8] aSymbolicLink
[9] printstat.c
[10] newFolder
[11] lstat.c
[(base) mahmutunan@MacBook-Pro lecture12 % ./exercise2 ./newFolder
[1] .
[2] ..
(base) mahmutunan@MacBook-Pro lecture12 %
```