Exercises with `readdir, opendir, closedir`

In preparation for these exercises, read through `man 3 readdir`, `man 3 opendir`, and `man 3 closedir`. Additionally, read through `man 3 fstatat` as it has a better description of some of the functionality provided by `stat` than is present in `man 3 stat`.

Take note especially of a suite of functions packed in with `<sys/stat.h>`:

given some `struct stat sb;` that we then fill using `stat`...

```
S_ISBLK(sb.st_mode);  /* is block special? */
S_ISCHR(sb.st_mode);  /* is character special? */
S_ISDIR(sb.st_mode);  /* is directory? */
S_ISFIFO(sb.st_mode); /* is fifo (first-in first-out)? */
S_ISLNK(sb.st_mode);  /* is symlink? */
S_ISREG(sb.st_mode);  /* is regular file? */
S_ISSOCK(sb.st_mode); /* is socket? */
```

You can use these functions to to tell if a file is of a certain type.

Directory reading and traversal is controlled by two things: A directory pointer (`DIR *`), and a `struct dirent` as follows (and as defined in `man 3 readdir`):

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; this can be ignored */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   * by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};
```

The `DIR *` maintains the currently open directory, and is handed to us by a call to `opendir(const char *name)` (where `name` is a character array holding a string representing the directory). We can perform two operations on a `Dir *`: `readdir(DIR *dirp)` and `closedir(DIR *dirp)`. `readdir` allows us to systematically read through each entry in a directory. It reads in no particular order, and will read in hidden files (those starting with a `.`), as well as current directory `.` and parent directory `..`. Ultimately a directory is just a structured file that a filesystem can read to figure out where the children of a directory are, and doesn't innately have any sense of order. `ls` appearing ordered alphabetically is a feature of its implementation, not of the directory itself.

Each call to `readdir` will give us a pointer to a `struct dirent`, or to NULL if we have exhausted the directory. The `struct dirent` can tell us several things about the file to which it refers. For example, we can tell the file's name. Keep in mind, this is just the name of the file, not its entire path.

So, let us simply read through a directory, and each time we reach a file, detect if it is another directory or not.

```
#include <sys/stat.h>
#include <sys/types.h>

#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main() {
```

```c
    DIR *dp;
    struct dirent *dent;
    struct stat sb;

    /* open directory so we can traverse it. */
    /* NOTE: opendir returns NULL if there was some error, and sets `errno` appropriately */
    if ((dp = opendir(".")) == NULL) {
        fprintf(stderr, "Error: Cannot open directory. %s.\n", strerror(errno));
        return EXIT_FAILURE;
    }

    /* We want to iterate until readdir returns NULL. This can mean one of two things:
     *   1. we have reached the end of the directory.
     *   2. there was an error.
     * As such, to distinguish between an error and the end of the directory, we set
     * errno to 0 before we readdir, then again at the end of every loop. Then
     * when we exit, we check if errno is 0. If it is not zero, then an error occured
     * in readdir. */
    errno = 0;
    while ((dent = readdir(dp)) != NULL) {
        /* get information on file so we can tell if it is a directory or a non-directory */
        if (stat(dent->d_name, &sb) < 0) {
            fprintf(stderr, "Error: Cannot stat file '%s'. %s.\n", dent->d_name, strerror(errno));
            continue;
        }

        /* check if the directory entry is a directory itself */
        if (S_ISDIR(sb.st_mode)) {
            printf("Directory: '%s'\n", dent->d_name);
        } else {
            printf("Non-directory: '%s'\n", dent->d_name);
        }
        errno = 0;
    }

    closedir(dp);

    if (errno != 0) {
        fprintf(stderr, "Error: An error occured while reading the directory. %s.\n", strerror(errno));
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Try playing around with this code. What happens if you try to read a file that you don't have permissions on? What if, instead of calling `opendir` on `"."`, we call it on some other directory? What if we don't have read or execute permissions on that directory?

How can we recursively call this for all subdirectories? Since we know how to check the type of a file, how will this information help us recurse?