◻ Exercise 5.7

Run Program 5.3 for different directories. Compare the output with that from running the `ls` shell command for the same directories. Why are they different?

Answer:

The `ls` command sorts filenames in alphabetical order. The `readdir` function displays filenames in the order in which they occur in the directory file.

Program 5.3 does not allocate a `struct dirent` variable to hold the directory information. Rather, `readdir` returns a pointer to a static `struct dirent` structure. This return structure implies that `readdir` is not thread-safe. POSIX includes `readdir_r` as part of the POSIX:TSF Extension, to provide a thread-safe alternative.

POSIX only requires that the `struct dirent` structure have a `d_name` member, representing a string that is no longer than `NAME_MAX`. POSIX does not specify where additional information about the file should be stored. Traditionally, UNIX directory entries contain only filenames and inode numbers. The inode number is an index into a table containing the other information about a file. Inodes are discussed in Section 5.3.

5.2.1 Accessing file status information

This section describes three functions for retrieving file status information. The `fstat` function accesses a file with an open file descriptor. The `stat` and `lstat` functions access a file by name.

The `lstat` and `stat` functions each take two parameters. The `path` parameter specifies the name of a file or symbolic link whose status is to be returned. If `path` does not correspond to a symbolic link, both functions return the same results. When `path` is a symbolic link, the `lstat` function returns information about the link whereas the `stat` function returns information about the file referred to by the link. Section 5.4 explains symbolic links. The `buf` parameter points to a user-supplied buffer into which these functions store the information.

SYNOPSIS

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

  int lstat(const char *restrict path, struct stat *restrict buf);
  int stat(const char *restrict path, struct stat *restrict buf);
```

POSIX

If successful, these functions return 0. If unsuccessful, they return –1 and set `errno`. *The* `restrict` *modifier on the arguments specifies that* `path` *and* `buf` *are not allowed to overlap.* The following table lists the mandatory errors for these functions.

`errno`	cause
`EACCES`	search permission on a `path` component denied
`EIO`	an error occurred while reading from the file system
`ELOOP`	a loop exists in resolution of `path`
`ENAMETOOLONG`	the length of the pathname exceeds `PATH_MAX` (`lstat`), the length of `path` exceeds `PATH_MAX` (`stat`), or a pathname component is longer than `NAME_MAX`
`ENOENT`	a component of `path` does not name an existing file
`ENOTDIR`	a component of the path prefix is not a directory
`EOVERFLOW`	the file size in bytes, the number of blocks allocated to file or the file serial number cannot be represented in the structure pointed to by `buf`

The `struct stat` structure, which is defined in `sys/stat.h`, contains at least the following members.

```
dev_t    st_dev;      /* device ID of device containing file */
ino_t    st_ino;      /* file serial number */
mode_t   st_mode;     /* file mode */
nlink_t  st_nlink;    /* number of hard links */
uid_t    st_uid;      /* user ID of file */
gid_t    st_gid;      /* group ID of file */
off_t    st_size;     /* file size in bytes (regular files) */
                      /* path size (symbolic links) */
time_t   st_atime;    /* time of last access */
time_t   st_mtime;    /* time of last data modification */
time_t   st_ctime;    /* time of last file status change */
```

■ Example 5.8

The following function displays the time that the file `path` was last accessed.

```
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>

void printaccess(char *path) {
   struct stat statbuf;

   if (stat(path, &statbuf) == -1)
      perror("Failed to get file status");
   else
      printf("%s last accessed at %s", path, ctime(&statbuf.st_atime));
}
```

`printaccess.c`