```
#include "apue.h"
#include <dirent.h>
#include <limits.h>

/* function type that is called for each filename */
typedef int Myfunc(const char *, const struct stat *, int);

static Myfunc   myfunc;
static int      myftw(char *, Myfunc *);
static int      dopath(Myfunc *);

static long nreg, ndir, nblk, nchr, nfifo, nslink, nsock, ntot;

int
main(int argc, char *argv[])
{
    int     ret;

    if (argc != 2)
        err_quit("usage:  ftw  <starting-pathname>");

    ret = myftw(argv[1], myfunc);       /* does it all */

    ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
    if (ntot == 0)
        ntot = 1;       /* avoid divide by 0; print 0 for all counts */
    printf("regular files  = %7ld, %5.2f %%\n", nreg,
      nreg*100.0/ntot);
    printf("directories    = %7ld, %5.2f %%\n", ndir,
      ndir*100.0/ntot);
    printf("block special  = %7ld, %5.2f %%\n", nblk,
      nblk*100.0/ntot);
    printf("char special   = %7ld, %5.2f %%\n", nchr,
      nchr*100.0/ntot);
    printf("FIFOs          = %7ld, %5.2f %%\n", nfifo,
      nfifo*100.0/ntot);
    printf("symbolic links = %7ld, %5.2f %%\n", nslink,
      nslink*100.0/ntot);
    printf("sockets        = %7ld, %5.2f %%\n", nsock,
      nsock*100.0/ntot);
    exit(ret);
}

/*
 * Descend through the hierarchy, starting at "pathname".
 * The caller's func() is called for every file.
 */
#define FTW_F   1       /* file other than directory */
#define FTW_D   2       /* directory */
```

Figure 4.22 *(continues)*

```
#define FTW_DNR 3       /* directory that can't be read */
#define FTW_NS  4       /* file that we can't stat */

static char *fullpath;      /* contains full pathname for every file */
static size_t pathlen;

static int                  /* we return whatever func() returns */
myftw(char *pathname, Myfunc *func)
{
    fullpath = path_alloc(&pathlen);    /* malloc PATH_MAX+1 bytes */
                                        /* (Figure 2.16) */
    if (pathlen <= strlen(pathname)) {
        pathlen = strlen(pathname) * 2;
        if ((fullpath = realloc(fullpath, pathlen)) == NULL)
            err_sys("realloc failed");
    }
    strcpy(fullpath, pathname);
    return(dopath(func));
}

/*
 * Descend through the hierarchy, starting at "fullpath".
 * If "fullpath" is anything other than a directory, we lstat() it,
 * call func(), and return.  For a directory, we call ourself
 * recursively for each name in the directory.
 */
static int                  /* we return whatever func() returns */
dopath(Myfunc* func)
{
    struct stat     statbuf;
    struct dirent   *dirp;
    DIR             *dp;
    int             ret, n;

    if (lstat(fullpath, &statbuf) < 0)  /* stat error */
        return(func(fullpath, &statbuf, FTW_NS));
    if (S_ISDIR(statbuf.st_mode) == 0)  /* not a directory */
        return(func(fullpath, &statbuf, FTW_F));

    /*
     * It's a directory.  First call func() for the directory,
     * then process each filename in the directory.
     */
    if ((ret = func(fullpath, &statbuf, FTW_D)) != 0)
        return(ret);

    n = strlen(fullpath);
    if (n + NAME_MAX + 2 > pathlen) {   /* expand path buffer */
        pathlen *= 2;
        if ((fullpath = realloc(fullpath, pathlen)) == NULL)
            err_sys("realloc failed");
    }
    fullpath[n++] = '/';
```

Figure 4.22 *(continues)*

```
    fullpath[n] = 0;

    if ((dp = opendir(fullpath)) == NULL)   /* can't read directory */
        return(func(fullpath, &statbuf, FTW_DNR));

    while ((dirp = readdir(dp)) != NULL) {
        if (strcmp(dirp->d_name, ".") == 0  ||
            strcmp(dirp->d_name, "..") == 0)
                continue;       /* ignore dot and dot-dot */
        strcpy(&fullpath[n], dirp->d_name); /* append name after "/" */
        if ((ret = dopath(func)) != 0)      /* recursive */
            break;  /* time to leave */
    }
    fullpath[n-1] = 0;  /* erase everything from slash onward */

    if (closedir(dp) < 0)
        err_ret("can't close directory %s", fullpath);
    return(ret);
}

static int
myfunc(const char *pathname, const struct stat *statptr, int type)
{
    switch (type) {
    case FTW_F:
        switch (statptr->st_mode & S_IFMT) {
        case S_IFREG:   nreg++;     break;
        case S_IFBLK:   nblk++;     break;
        case S_IFCHR:   nchr++;     break;
        case S_IFIFO:   nfifo++;    break;
        case S_IFLNK:   nslink++;   break;
        case S_IFSOCK:  nsock++;    break;
        case S_IFDIR:   /* directories should have type = FTW_D */
            err_dump("for S_IFDIR for %s", pathname);
        }
        break;
    case FTW_D:
        ndir++;
        break;
    case FTW_DNR:
        err_ret("can't read directory %s", pathname);
        break;
    case FTW_NS:
        err_ret("stat error for %s", pathname);
        break;
    default:
        err_dump("unknown type %d for pathname %s", type, pathname);
    }
    return(0);
}
```

Figure 4.22 Recursively descend a directory hierarchy, counting file types