```c
// Nate McCain
// CS 390
// Project 4
// 10/25/2017

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <unistd.h>
#include "ProjectFourFunctions.h"

int main(int argc, const char ** argv)
{
    DIR *dir; // Pointer to the directory.
    struct dirent *direntry; // Holds information about the current directory entry.
    struct stat statbuf; // Holds information about the current file entry.
    FILE *pFile; // Points to the output text file.
    int numberOfRegFiles = 0; // Tally for the number of regular files.
    int numberOfDirFiles = 0; // Tally for the number of directory files.
    long long totalFileSizeInBytes = 0; // Holds the byte size of everything in the
     directory.
    long long totalFileSizeInBlocks = 0; // Holds the block size of everything in
     the directory.
    int communicationPipe[2]; // The pipe used for communicating between parent and
     child.
    int childStatus; // Holds the status of the child process.
    char filepath[150]; // Used to hold string values.

    // Exit the program if it does not match a directory.
    if ((dir = opendir(argv[1])) == NULL)
    {
        printf("You entered the following: \n");
        printf("%s \n \n", argv[1]);
        printf("You did not enter a directory. Goodbye. \n \n \n");
        return 0;
    }

    // Create/Open a new report file for writing.
    // Exit the program if this operation fails.
    pFile = fopen("output.txt", "w");
    if (pFile == NULL)
    {
        printf("Error opening the file for writing results. Goodbye. \n \n \n");
        return 0;
    }

    // Spawn a pipe.
    // Exit the program if this operation fails.
    if (pipe(communicationPipe) < 0)
```

```
{
    printf("Pipe error. Goodbye. \n \n \n");
    return 0;
}

// Create a child process.
switch (fork())
{
    // Exit the program if this operation fails.
    case -1:
        printf("Error");
        return 0;

    // The child process.
    case 0:
        // Close the write-end of the child's side of the pipe.
        close(communicationPipe[1]);
        // Perform an initial read of the pipe.
        read(communicationPipe[0], &statbuf, sizeof(statbuf));

        // Keep recording statistics until an inode with
        // a value of zero is passed through the pipe.
        while (statbuf.st_ino != 0)
        {
            // Record the statistics of the current file entry.
            takeStatistics(&numberOfDirFiles, &numberOfRegFiles,
             &totalFileSizeInBytes, &totalFileSizeInBlocks, statbuf);
            // Read the next status record from the pipe.
            read(communicationPipe[0], &statbuf, sizeof(statbuf));
        }
        // Write the final statistics to the text file.
        writeStatistics(numberOfDirFiles, numberOfRegFiles,
         totalFileSizeInBytes, totalFileSizeInBlocks, pFile);
        // Close the text file.
        fclose(pFile);
        // End the child process.
        exit(0);

    // The parent process.
    default:
        // Begin reading from the directory.
        direntry = readdir(dir);

        // While not at the end of the directory.
        while (direntry != NULL)
        {
            // Put together the pathname.
            formatToPathname(filepath, argv[1], direntry->d_name);
            // Get the stats of the pathname.
            lstat(filepath, &statbuf);
            // Send the status record to the child.
            write(communicationPipe[1], &statbuf, sizeof(statbuf));
            // Read the next directory entry.
            direntry = readdir(dir);
```

```c
            }
            // Prepare to send a status record with a zero inode.
            statbuf.st_ino = 0;
            // Send the status record with a zero inode to the child.
            write(communicationPipe[1], &statbuf, sizeof(statbuf));
            // Close the directory,
            closedir(dir);
            // and wait for the child to finish.
            wait(&childStatus);
    }

    // If pFile is not null, then close it (as it was opened
    // originally for writing).
    if (pFile != NULL)
    {
        fclose(pFile);
    }
    // Open pFile for reading.
    pFile = fopen("output.txt", "r");
    // Read the statistics written in pFile.
    outputStatistics(pFile);
    // Close pFile.
    fclose(pFile);
    // End of the application.
    return 0;
}
```