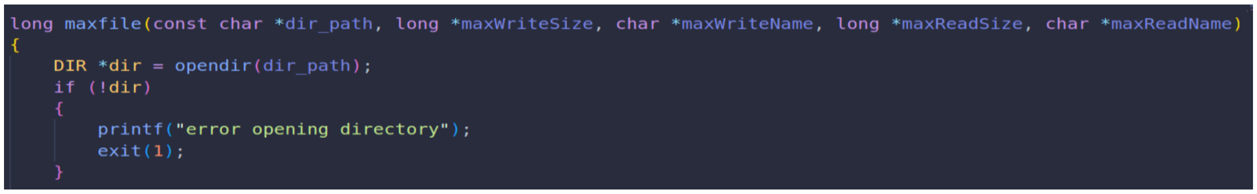
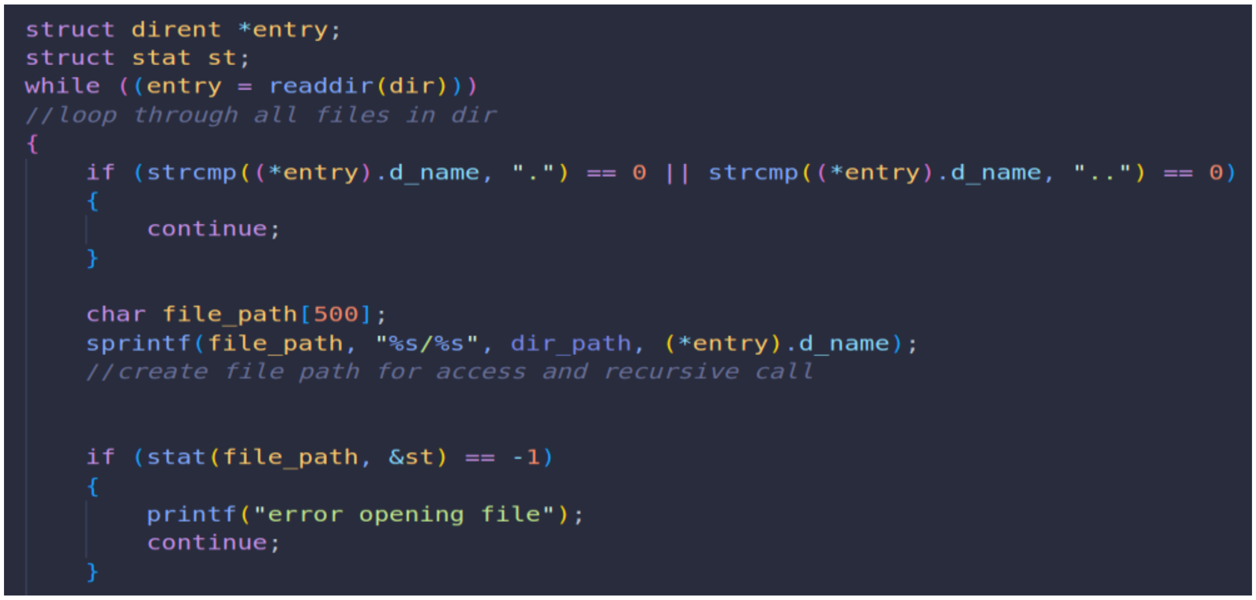
Benjamin Griepp
392 HW 4

Program works recursively.



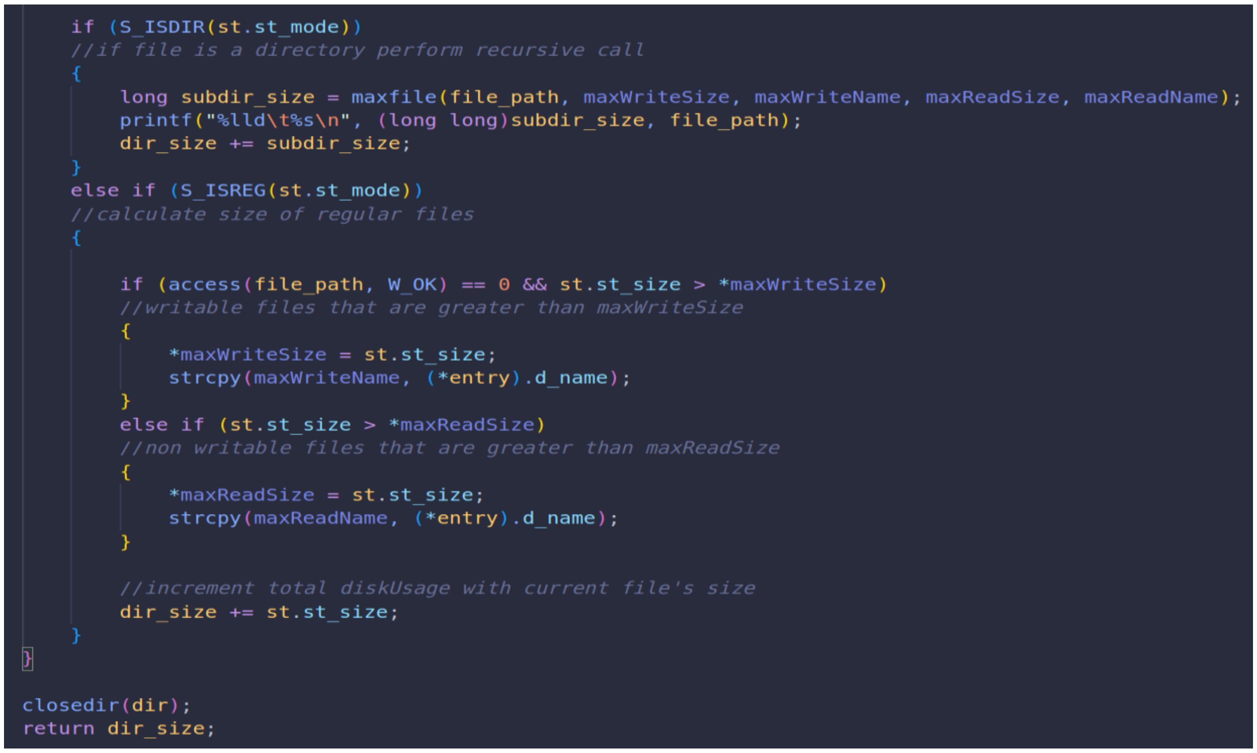
```
long maxfile(const char *dir_path, long *maxWriteSize, char *maxWriteName, long *maxReadSize, char *maxReadName)
{
    DIR *dir = opendir(dir_path);
    if (!dir)
    {
        printf("error opening directory");
        exit(1);
    }
```

pass parameters as pointers so that recursive calls can view and set the same variables. open directory and exit with error code if directory not opened successfully.



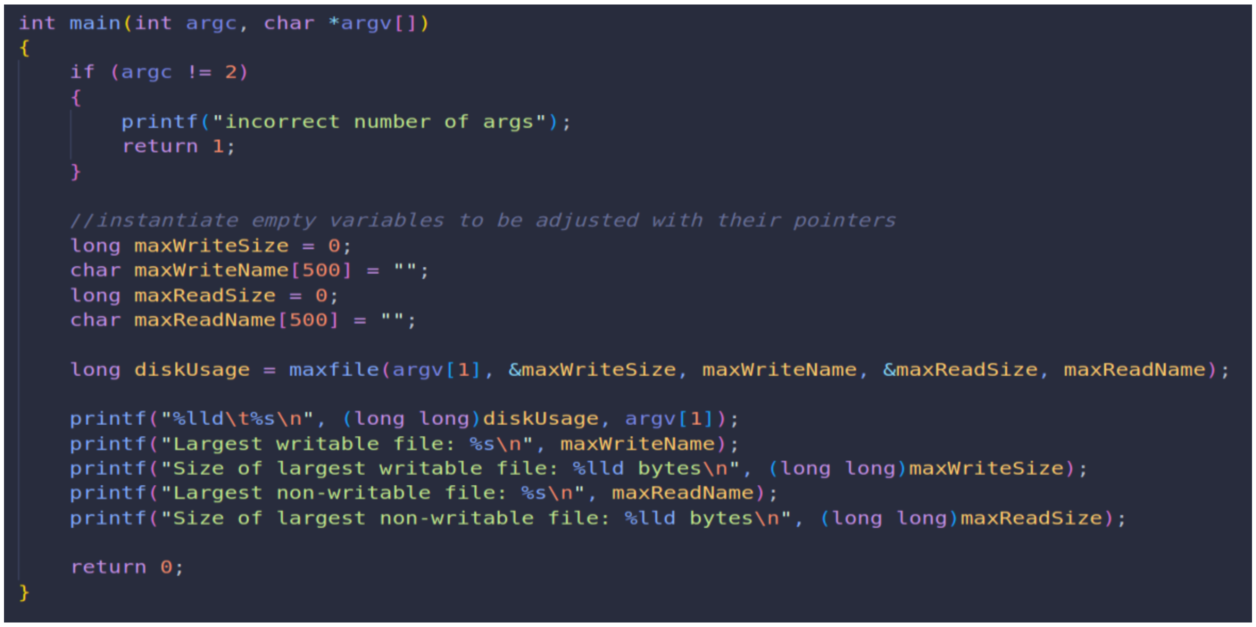
```
struct dirent *entry;
struct stat st;
while ((entry = readdir(dir)))
//loop through all files in dir
{
    if (strcmp((*entry).d_name, ".") == 0 || strcmp((*entry).d_name, "..") == 0)
    {
        continue;
    }

    char file_path[500];
    sprintf(file_path, "%s/%s", dir_path, (*entry).d_name);
    //create file path for access and recursive call


    if (stat(file_path, &st) == -1)
    {
        printf("error opening file");
        continue;
    }
```

declare entry struct for iterating through directory and stat struct for determining file types. iterate through dir using readdir() and storing in entry. check if entry is “.” or “..” and if it is, skip over that entry using continue. declare file_path char array, concatenate the directories path and the name of the entry into file_path. store stat of file_path into st and if unsuccessful print an error and skip over file.

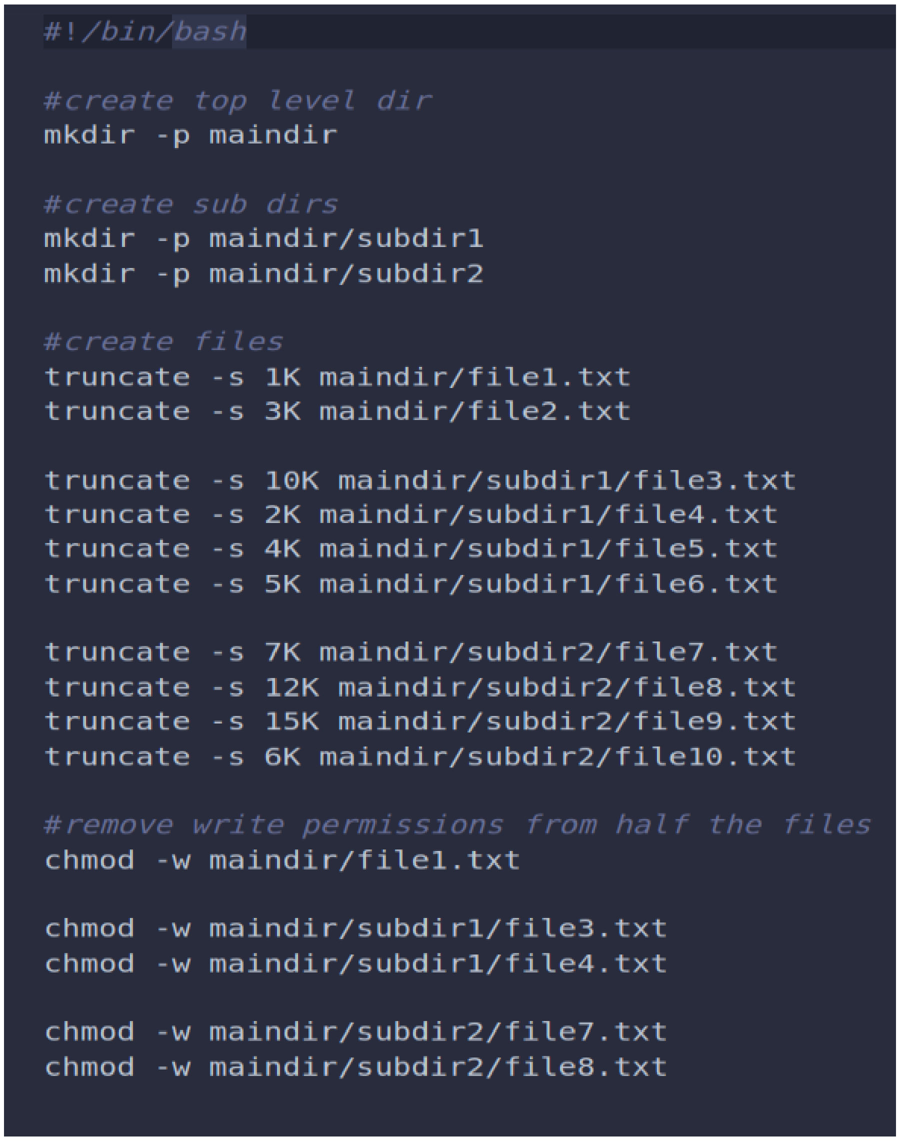


```
    if (S_ISDIR(st.st_mode))
    //if file is a directory perform recursive call
    {
        long subdir_size = maxfile(file_path, maxWriteSize, maxWriteName, maxReadSize, maxReadName);
        printf("%lld\t%s\n", (long long)subdir_size, file_path);
        dir_size += subdir_size;
    }
    else if (S_ISREG(st.st_mode))
    //calculate size of regular files
    {

        if (access(file_path, W_OK) == 0 && st.st_size > *maxWriteSize)
        //writable files that are greater than maxWriteSize
        {
            *maxWriteSize = st.st_size;
            strcpy(maxWriteName, (*entry).d_name);
        }
        else if (st.st_size > *maxReadSize)
        //non writable files that are greater than maxReadSize
        {
            *maxReadSize = st.st_size;
            strcpy(maxReadName, (*entry).d_name);
        }

        //increment total diskUsage with current file's size
        dir_size += st.st_size;
    }
}

closedir(dir);
return dir_size;
```

check if file is a directory and if it is, perform a recursive call on it, store and print directories size and add it to the total size. if a regular file, check if file is writable and has size greater than maxWriteSize and if so store the name and size of the file as new max, if not writable, check if size greater than maxReadSize and if so store its name and size as new max. add the size of the file to the total diskUsage and finally after all files are iterated through, close the directory and return the total size.


```c
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("incorrect number of args");
        return 1;
    }

    //instantiate empty variables to be adjusted with their pointers
    long maxWriteSize = 0;
    char maxWriteName[500] = "";
    long maxReadSize = 0;
    char maxReadName[500] = "";

    long diskUsage = maxfile(argv[1], &maxWriteSize, maxWriteName, &maxReadSize, maxReadName);

    printf("%lld\t%s\n", (long long)diskUsage, argv[1]);
    printf("Largest writable file: %s\n", maxWriteName);
    printf("Size of largest writable file: %lld bytes\n", (long long)maxWriteSize);
    printf("Largest non-writable file: %s\n", maxReadName);
    printf("Size of largest non-writable file: %lld bytes\n", (long long)maxReadSize);

    return 0;
}
```

in main, first check if incorrect number of args. instantiate empty variables so that their pointers can be passed into function. perform function call with arrays as they will pass pointers to the their first elements as well as pointers to the long values and store the value for total size. print total size and all variables. finally return 0 to indicate the program ran successfully.



```bash
#!/bin/bash

#create top level dir
mkdir -p maindir

#create sub dirs
mkdir -p maindir/subdir1
mkdir -p maindir/subdir2

#create files
truncate -s 1K maindir/file1.txt
truncate -s 3K maindir/file2.txt

truncate -s 10K maindir/subdir1/file3.txt
truncate -s 2K maindir/subdir1/file4.txt
truncate -s 4K maindir/subdir1/file5.txt
truncate -s 5K maindir/subdir1/file6.txt

truncate -s 7K maindir/subdir2/file7.txt
truncate -s 12K maindir/subdir2/file8.txt
truncate -s 15K maindir/subdir2/file9.txt
truncate -s 6K maindir/subdir2/file10.txt

#remove write permissions from half the files
chmod -w maindir/file1.txt

chmod -w maindir/subdir1/file3.txt
chmod -w maindir/subdir1/file4.txt

chmod -w maindir/subdir2/file7.txt
chmod -w maindir/subdir2/file8.txt
```

bash file makes maindir and then two subdirectories inside of it. create two files in main dir and then four files in each subdirectory all of various sizes. finally remove write permissions from half the files.