

Test assignment for candidates for the position of C developer

Name: Talha Mansoor Qureshi

Task #1

Write a bash script that finds all files containing C programming language texts (the names of such files are determined by the *.c mask) in the working directory and its subdirectories and makes a copy of each file, adding the .orig extension to the file name.

Example: let the working directory be /home/some_project. The structure of the directory and its subdirectories is as follows

/home/some_project

Makefile

Include

Common.h

util.h

src

Main.c

util.c

After the script runs, the directory structure will change to:

/home/some_project

Makefile

Include

common.h

util.h

src

main.c

main.c.orig

util.c

Util.c.orig

Solution:

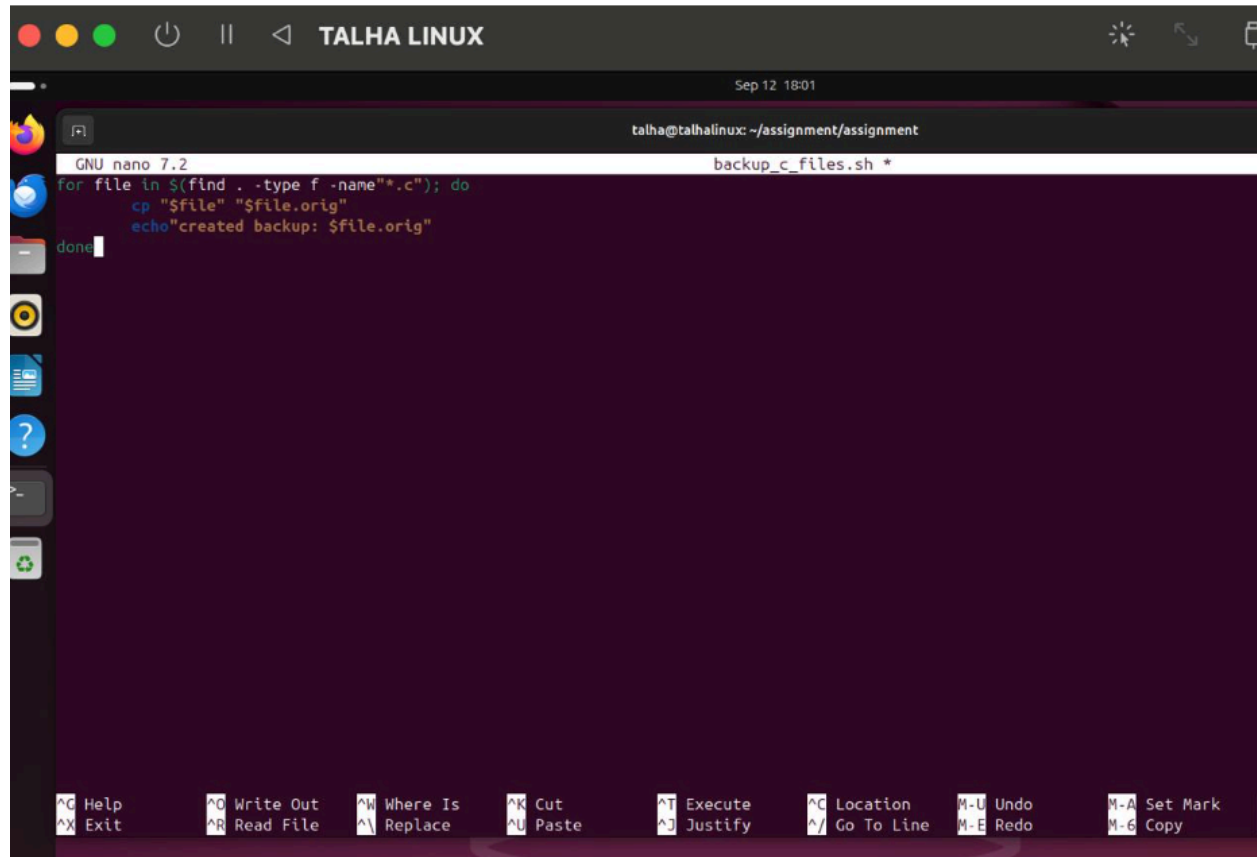
As step 1, I created a script named as backup_c_files.sh in my subdirectory, with the following code added in it in order to achieve the final goal.

File Code:

None

```
for file in $(find . -type f -name "*.c"); do
    cp "$file" "$file.orig"
    echo "Created backup: $file.orig"
done
```





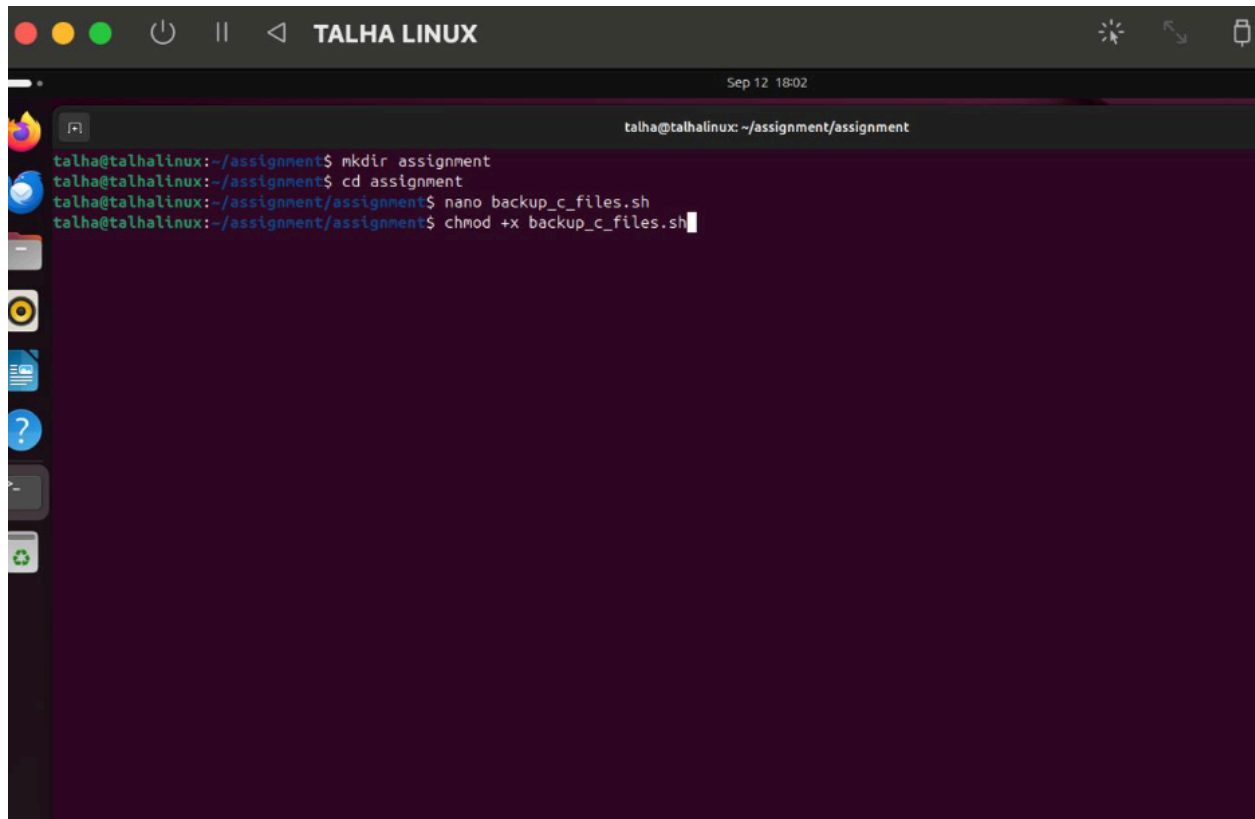
The screenshot shows a Linux desktop environment with a terminal window titled "TALHA LINUX". The terminal displays the GNU nano 7.2 editor editing a file named "backup_c_files.sh". The script content is as follows:

```
for file in $(find . -type f -name "*.c"); do
  cp "$file" "$file.orig"
  echo "created backup: $file.orig"
done
```

The terminal window includes a top status bar with the date and time "Sep 12 18:01" and a bottom status bar with various keyboard shortcuts for nano editor operations.

Shortcut	Command
<code>^G</code>	Help
<code>^O</code>	Write Out
<code>^W</code>	Where Is
<code>^K</code>	Cut
<code>^T</code>	Execute
<code>^C</code>	Location
<code>^U</code>	Undo
<code>^M</code>	Set Mark
<code>^X</code>	Exit
<code>^R</code>	Read File
<code>^I</code>	Replace
<code>^P</code>	Paste
<code>^J</code>	Justify
<code>^_</code>	Go To Line
<code>^E</code>	Redo
<code>^C</code>	Copy

Then I make it executable with `chmod` and execute the file to test the working.



```
talha@talhalinux:~/assignment$ mkdir assignment
talha@talhalinux:~/assignment$ cd assignment
talha@talhalinux:~/assignment/assignment$ nano backup_c_files.sh
talha@talhalinux:~/assignment/assignment$ chmod +x backup_c_files.sh
```

Here's my final output for task 1 as it was successful.

```
talha@talhalinux:~/assignment$ ./backup_c_files.sh
Created backup: ./pipeline.c.orig
Created backup: ./mini_ls.c.orig
Created backup: ./assignment/pipeline.c.orig
Created backup: ./assignment/mini_ls.c.orig
```

Task #2

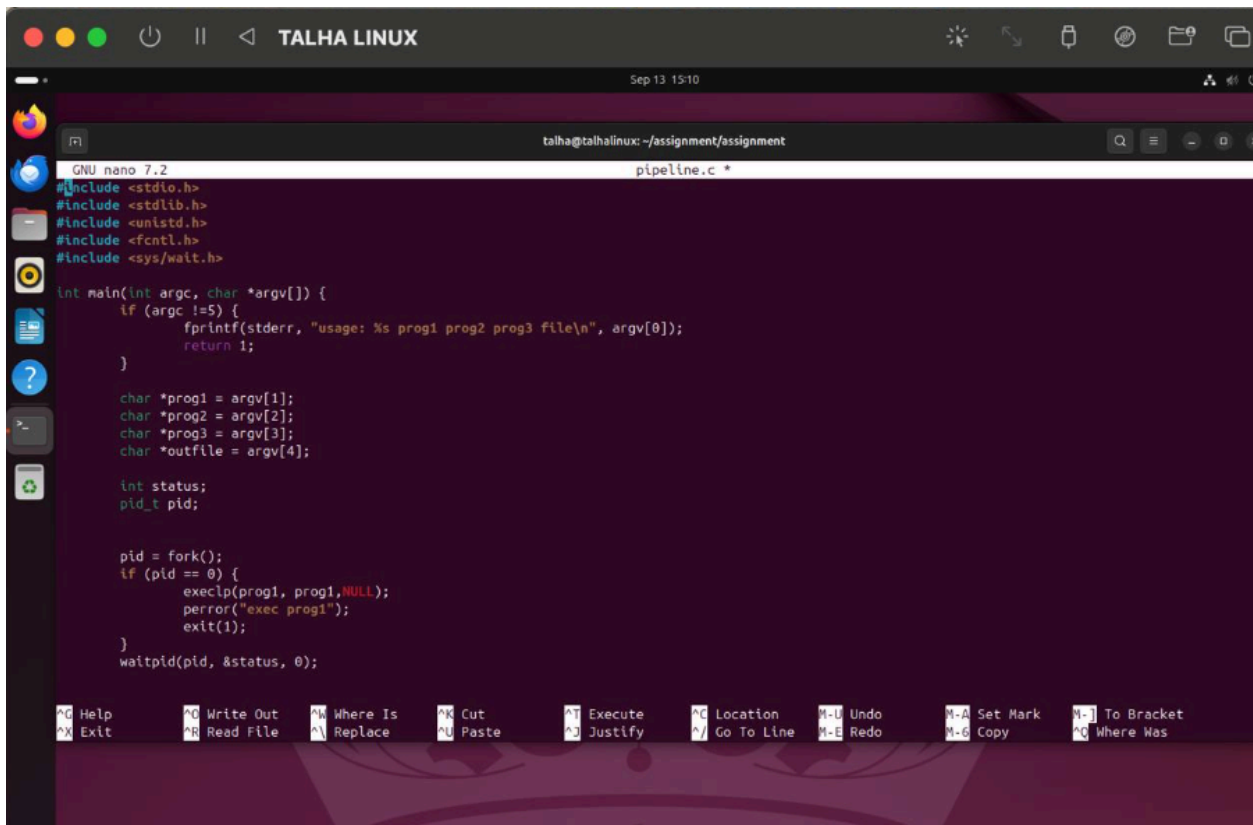
Write in C the simplest analog of the ls utility that prints the contents of the working directory and all its subdirectories. The program's output should be listings similar to those presented in the first task. To accomplish this, use the following library functions: opendir(), readdir(), closedir().

Solution:

To solve this problem, I created a source file named as mini_ls.c, and added the following C code using opendir(), readdir() and closedir().

```
talha@talhalinux:~/assignment/assignment$ nano mini_ls.c
talha@talhalinux:~/assignment/assignment$
```

Here's the code:



```
GNU nano 7.2 pipeline.c *
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "usage: %s prog1 prog2 prog3 file\n", argv[0]);
        return 1;
    }

    char *prog1 = argv[1];
    char *prog2 = argv[2];
    char *prog3 = argv[3];
    char *outfile = argv[4];

    int status;
    pid_t pid;

    pid = fork();
    if (pid == 0) {
        execlp(prog1, prog1, NULL);
        perror("exec prog1");
        exit(1);
    }
    waitpid(pid, &status, 0);
}
```

None

```
#include <stdio.h>
```

```

#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <stdlib.h>

void walkFolders(const char *dirPath, int indent) {
    DIR *d;
    struct dirent *thing;

    d = opendir(dirPath);
    if (d == NULL) {
        // if the folder does not exist
        perror("opendir failed");
        return;
    }

    while ((thing = readdir(d)) != NULL) {

        if (strcmp(thing->d_name, ".") == 0 || strcmp(thing->d_name, "..") ==
0) {
            continue;
        }

        // giving spaces
        for (int k = 0; k < indent; k++) {
            printf(" ");    // two spaces each level
        }

        printf("%s\n", thing->d_name);

        // Building a path for this
        char full[1024];
        snprintf(full, sizeof(full), "%s/%s", dirPath, thing->d_name);

        struct stat info;
        int stat_ok = stat(full, &info);
        if (stat_ok == 0) {
            if (S_ISDIR(info.st_mode)) {

                walkFolders(full, indent + 1);
            }
        } else {
        }
    }
}

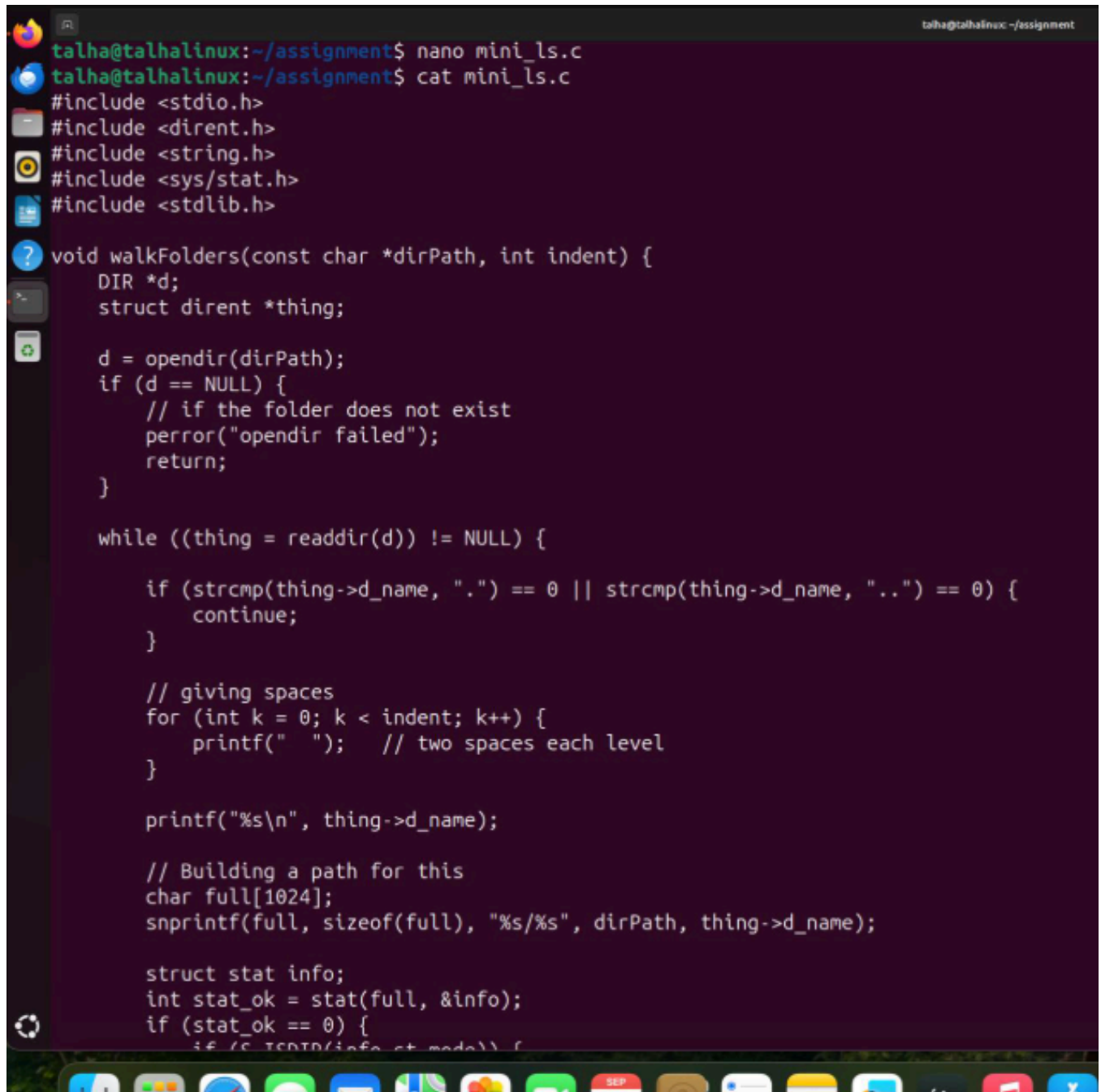
```

```
        closedir(d);
    }

    int main(int argc, char *argv[]) {
        const char *rootDir = ".";    // for current directory
        if (argc > 1) {
            rootDir = argv[1];
        }

        walkFolders(rootDir, 0);

        return 0;
    }
```

A terminal window with a dark purple background. The top bar shows the user 'talha' at 'talhalinux' in the directory '~/assignment'. The user has executed 'nano mini_ls.c' and 'cat mini_ls.c'. The code for 'mini_ls.c' is displayed, featuring standard C headers, a recursive 'walkFolders' function that uses 'opendir', 'readdir', and 'stat' to traverse a directory tree and print its structure with indentation. The code is partially visible, ending with a line that is cut off.

```
talha@talhalinux:~/assignment$ nano mini_ls.c
talha@talhalinux:~/assignment$ cat mini_ls.c
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <stdlib.h>

void walkFolders(const char *dirPath, int indent) {
    DIR *d;
    struct dirent *thing;

    d = opendir(dirPath);
    if (d == NULL) {
        // if the folder does not exist
        perror("opendir failed");
        return;
    }

    while ((thing = readdir(d)) != NULL) {

        if (strcmp(thing->d_name, ".") == 0 || strcmp(thing->d_name, "..") == 0) {
            continue;
        }

        // giving spaces
        for (int k = 0; k < indent; k++) {
            printf("  "); // two spaces each level
        }

        printf("%s\n", thing->d_name);

        // Building a path for this
        char full[1024];
        snprintf(full, sizeof(full), "%s/%s", dirPath, thing->d_name);

        struct stat info;
        int stat_ok = stat(full, &info);
        if (stat_ok == 0) {
            if (S_ISDIR(info.st_mode)) {
```

Next I compiled the code and execute it to test the output. Following screenshot reflects that it was successful as we can see the comparison of the output with ls - R to confirm the directory structure.


```
talha@talhalinux: ~/assignment
talha@talhalinux:~/assignment$ nano pipeline.c
talha@talhalinux:~/assignment$ gcc mini_ls.c -o mini_ls
talha@talhalinux:~/assignment$ ./mini_ls
.backup.swp
backup_c_files.sh
pipeline.c
mini_ls
mini_ls.c
assignment
  pipeline
  backup_c_files.sh
  pipeline.c
  out.txt
  mini_ls
  mini_ls.c
  output.txt
talha@talhalinux:~/assignment$ ls -R
.:
assignment backup_c_files.sh mini_ls mini_ls.c pipeline.c

./assignment:
backup_c_files.sh mini_ls.c out.txt pipeline.c
mini_ls          output.txt pipeline
talha@talhalinux:~/assignment$
```

Task #3

Write a program that takes 4 arguments: prog1, prog2, prog3, and file, and implements with the standard library and system calls the shell command: prog1 && prog2 | prog3 > file.

Solution:

So in this task I have to implement shell command structure prog1 && prog2 | prog3 > File using system calls and standard library functions. At first I created a source file with the name pipeline.c. I wrote it with fork(), pipe(), dup2(), and execvp().

```
talha@talhalinux:~/assignment/assignment$ nano pipeline.c
talha@talhalinux:~/assignment/assignment$
```

Here's the code:

```
talha@talhalinux: ~/assignment
GNU nano 7.2 pipeline.c
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <stdlib.h>

// My attempt at a recursive directory printer (kind of like `tree` but simpler)
void showDir(const char *folder, int level) {
    DIR *d;
    struct dirent *item;

    d = opendir(folder);
    if (d == NULL) {
        // NOTE: sometimes permissions block us here
        perror("Couldn't open directory");
        return;
    }

    // Loop through everything we find in the folder
    while ((item = readdir(d)) != NULL) {
        [ Read 60 lines ]
    }
}

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^Y Exit      ^P Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

None

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s prog1 prog2 prog3 file\n", argv[0]);
        exit(1);
    }

    char *prog1 = argv[1];
    char *prog2 = argv[2];
    char *prog3 = argv[3];
    char *file = argv[4];

    int pipefd[2];
```

```

pid_t p1, p2, p3;

// run prog1
if ((p1 = fork()) == 0) {
    execlp(prog1, prog1, NULL);
    perror("exec prog1");
    exit(1);
}
waitpid(p1, NULL, 0); // wait for prog1

// pipe between prog2 and prog3
pipe(pipefd);

if ((p2 = fork()) == 0) {
    dup2(pipefd[1], STDOUT_FILENO);
    close(pipefd[0]);
    close(pipefd[1]);
    execlp(prog2, prog2, NULL);
    perror("exec prog2");
    exit(1);
}

if ((p3 = fork()) == 0) {
    int fd = open(file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    dup2(pipefd[0], STDIN_FILENO);
    dup2(fd, STDOUT_FILENO);
    close(fd);
    close(pipefd[0]);
    close(pipefd[1]);
    execlp(prog3, prog3, NULL);
    perror("exec prog3");
    exit(1);
}

close(pipefd[0]);
close(pipefd[1]);

waitpid(p2, NULL, 0);
waitpid(p3, NULL, 0);

return 0;
}

```

```
talha@talhalinux: ~/assignment
talha@talhalinux:~/assignment$ nano pipeline.c
talha@talhalinux:~/assignment$ cat pipeline.c
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <stdlib.h>

// My attempt at a recursive directory printer (kind of like `tree` but simpler)
void showDir(const char *folder, int level) {
    DIR *d;
    struct dirent *item;

    d = opendir(folder);
    if (d == NULL) {
        // NOTE: sometimes permissions block us here
        perror("Couldn't open directory");
        return;
    }

    // Loop through everything we find in the folder
    while ((item = readdir(d)) != NULL) {
```

Now I compiled the code and executed it as shown below in the screenshot. It contains the sorted list of the files in the current directory and it was successful.

```
talha@talhalinux:~/assignment$ gcc pipeline.c -o pipeline
talha@talhalinux:~/assignment$ ./pipeline echo ls sort output.txt

talha@talhalinux:~/assignment$ cat output.txt
assignment
backup_c_files.sh
mini_ls
mini_ls.c
output.txt
pipeline
pipeline.c
talha@talhalinux:~/assignment$
```