OPERATING SYSTEMS

Assignment 4

Aristarchos Kaloutsas
Pavlos Dais

Contents

1 General Information

The assignment has been implemented in C. The functionalities described in the assignment, have been implemented. Though, we implemented additional features such as handling of symbolic links outside the directory hierarchy, as we'll explore later on.

2 Directory & File Explanation

We have chosen the following file organization:
src/: Source files for the program.
  src/compare_files.c: Functions to compare two directories.
  src/compare_utils.c: Utilities to compare files.
  src/copy_file.c: Copy utilities for creating the output directory
  src/main.c: Main driver for the program.
  src/merge_files.c: Merging files in a directory.
  src/parsing.c: Parsing command line argument functions.
  src/utilities.c: General utility functions needed throughout the program.
  src/modules/: Implementation of data structs used.
  src/modules/hash_table.c: Implementation of linear hashing hashtable.
  src/modules/vector.c: Implementation of vector.
include/: Header files for the program. Same file names as **src**.
Makefile: Makefile that allows the compilation of the program.
A: Test directory A (first directory).
B: Test directory B (second directory).

3 Compare files

To compare the two directories, the following steps are followed:

1. Using scandir we get each name of the file contained in each directory, in a sorted array. Since the file names are sorted, we keep an index for each array to current file name.

2. While the are still files in **both** arrays we compare the file names pointed by the index.

3. Since the names are unique and they are sorted if the files are not equal, we store the differences of the alphanumerically smaller file name and advance its index.

4. If the file names are equal then we compare the files. We have the following cases:

   - If they are of different type, we store both of the differences.
   - If they are both directories they are equal because they have the same name, so we just call this function recursively for those directories to find their differences.

   We advance both indexes after the comparison.

5. If there are remaining entries in the array for one directory they are all differences since they do not exist in the other directory, so we store them.

4 Merge files

To merge the two directories, the following steps are followed:

1. Using scandir we get each name of the file contained in each directory, in a sorted array. Since the file names are sorted, we keep an index for each array to current file name.
2. While the are still files in **both** arrays we compare the file names pointed by the index.
3. Since the names are unique and they are sorted if the files are not equal, we copy the contents of the alphanumerically smaller file name and advance its index.
4. If the file names are equal then we compare the files. We have the following cases:
   - If they are of different type, we copy the most recently modified file.
   - If they are both directories they are equal because they have the same name, so we just call this function recursively for those directories to copy their files.

   We advance both indexes after the comparison.
5. If there are remaining entries in the array for one directory they are all differences since they do not exist in the other directory, so we copy them.

5 Technical Details

5.1 File comparison

The file comparison function are only called on files with the same name and relative path. Therefore, it is the callers responsibility to make sure these conditions are met.

If they are of the same type:

1. **Regular file**: Open the files and compare its contents with memcmp.
2. **Symbolic link**:
   - We get the path of the file the link points to. Then recursively call the compare files function.
   - We handle links outside the directory hierarchy by removing the full path of each directory. If that removal fails that means we have a link outside the hierarchy. If both of them point outside the hierarchy to the same fullpath they are the equal, otherwise different.
   - We check for dangling links by looking at the error code of lstat on the links target. If they point to the same full path and they are both dangling they are considered equal, otherwise different.
3. **Directory**: They are always equal because they have the same name.
4. **Hard link**: We use the the previous cases to compare them.

If they are not of the same type, they are different.

We save the differences in a vector data structure in order to later on print them in a readable way.

5.2 Merge

- The same logic as file comparison has been applied in order to find the file differences (as we can also see by the steps followed). The only difference is that instead of storing the difference we create and copy the file in the new directory.
- In order to not copy hard links more than once, we store the inode of its copied file in a (linear) hash table. The hash table used is the same as assignment 1. If the inode already exists we simply link the file instead of copying it. Otherwise, we insert it in the hash table.

6 Test example

We also provide a relatively comprehensive test example covering a variety of scenarios and edge cases.

6.1 Test cases

We test the following cases (paths starting with / are relative to the starting directories):

1. **Regular file**:
   - Files with same relative path and different content: **/dir2/a**
   - Files with same relative path and same content: **/dir2/a**
2. **Symbolic link**:
   - link to directory: **/dir1/dir2/link**
   - link to regular file (outside the hierarchy): **/linkm**
   - link to link: **/dir1/tlink**
   - danglink link: **/linkdang**
3. **Directory**:
   - **/dir1**
   - **/dir2**

   ...
4. **Hard link**:
   - hard link to regular file: **dir2/alink** (points to /dir2/a)
   - hard link to symbolic link: **./A/dir2/hardlink** (points to ./A/dir1/dir2/link)
5. **Same name different type**:
   - **/dir5**

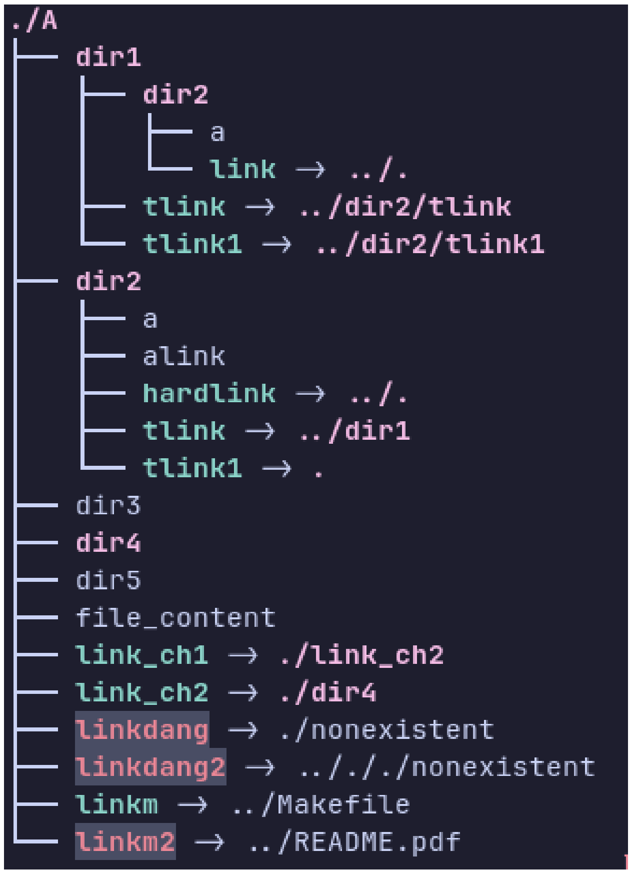
6.2 Input

Now let's see the actual example as described in section **6.1**.

- At subsections **Simple** we show the directory in its simple form.
- At subsections **Detailed** we show the directory in a more detailed form (we show the creation date as well the inode number).

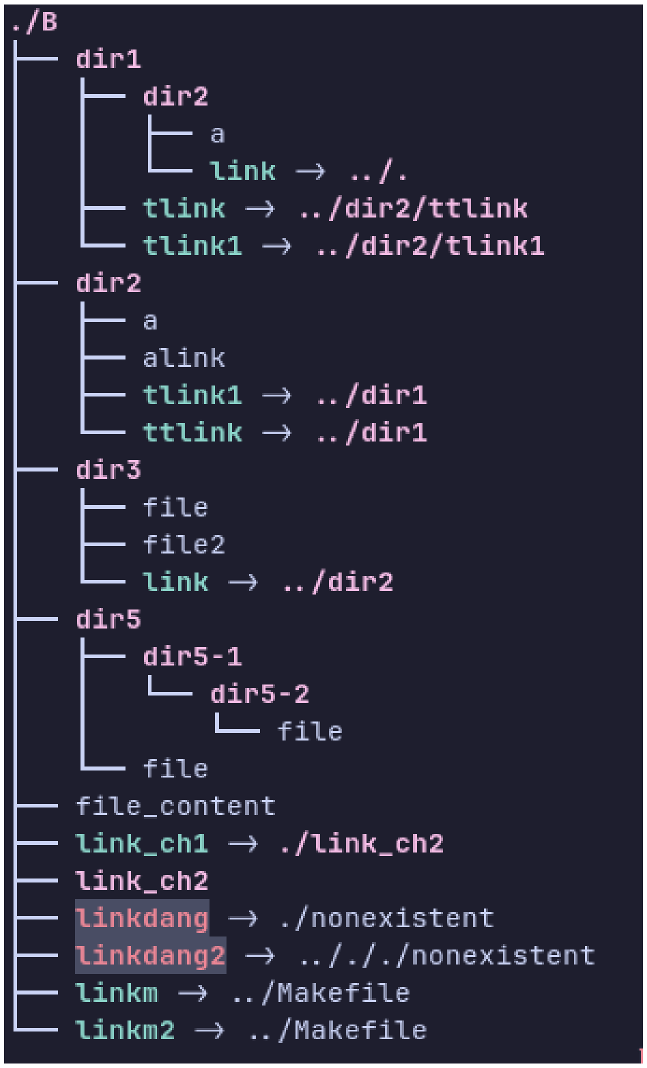
6.2.1 Simple

Let's see directories A & B [Simple].



```
./A
├── dir1
│   ├── dir2
│   │   ├── a
│   │   └── link → ../.
│   ├── tlink → ../dir2/tlink
│   └── tlink1 → ../dir2/tlink1
├── dir2
│   ├── a
│   ├── alink
│   ├── hardlink → ../.
│   ├── tlink → ../dir1
│   └── tlink1 → .
├── dir3
├── dir4
├── dir5
├── file_content
├── link_ch1 → ./link_ch2
├── link_ch2 → ./dir4
├── linkdang → ./nonexistent
├── linkdang2 → .././../nonexistent
├── linkm → ../Makefile
└── linkm2 → ../README.pdf
```

Figure 1: Folder A [Simple]

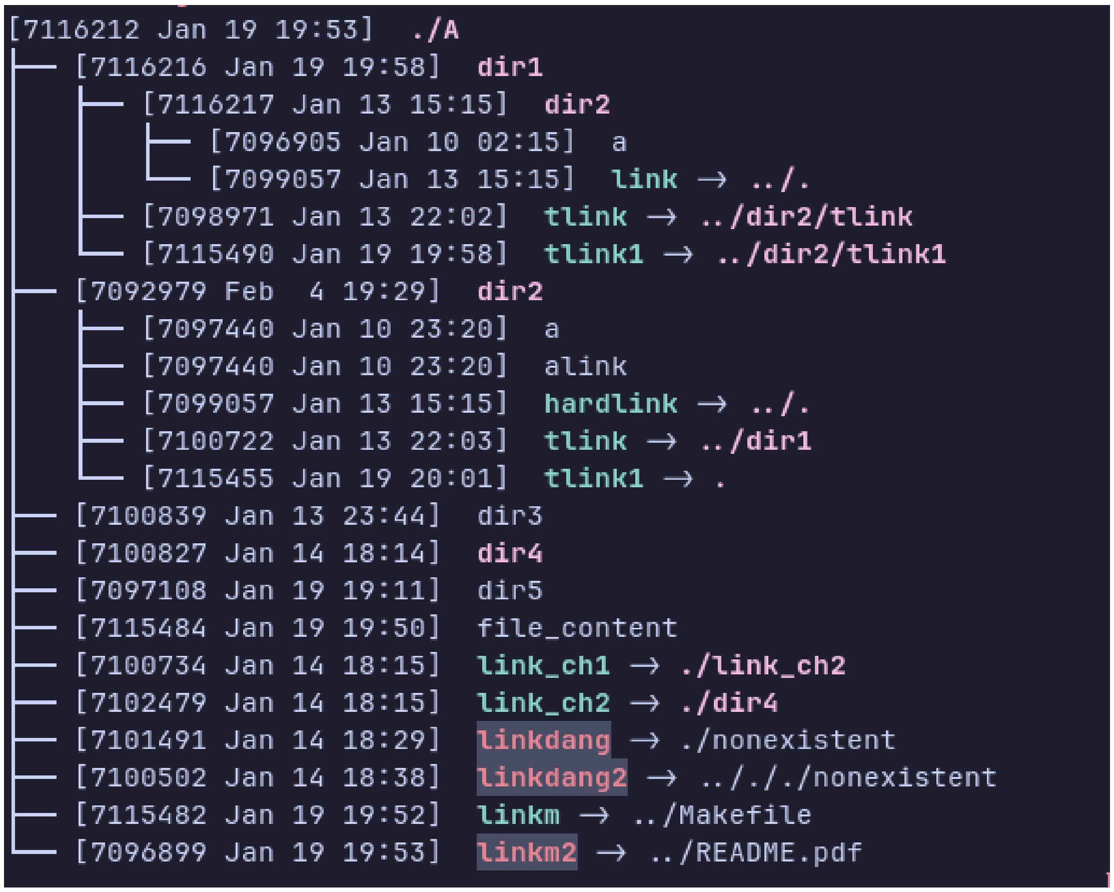


```
./B
├── dir1
│   ├── dir2
│   │   ├── a
│   │   └── link -> ../.
│   ├── tlink -> ../dir2/ttlink
│   └── tlink1 -> ../dir2/tlink1
├── dir2
│   ├── a
│   ├── alink
│   ├── tlink1 -> ../dir1
│   └── ttlink -> ../dir1
├── dir3
│   ├── file
│   ├── file2
│   └── link -> ../dir2
├── dir5
│   ├── dir5-1
│   │   └── dir5-2
│   │       └── file
│   └── file
├── file_content
├── link_ch1 -> ./link_ch2
├── link_ch2
├── linkdang -> ./nonexistent
├── linkdang2 -> ../././nonexistent
├── linkm -> ../Makefile
└── linkm2 -> ../Makefile
```

Figure 2: Folder B [Simple]

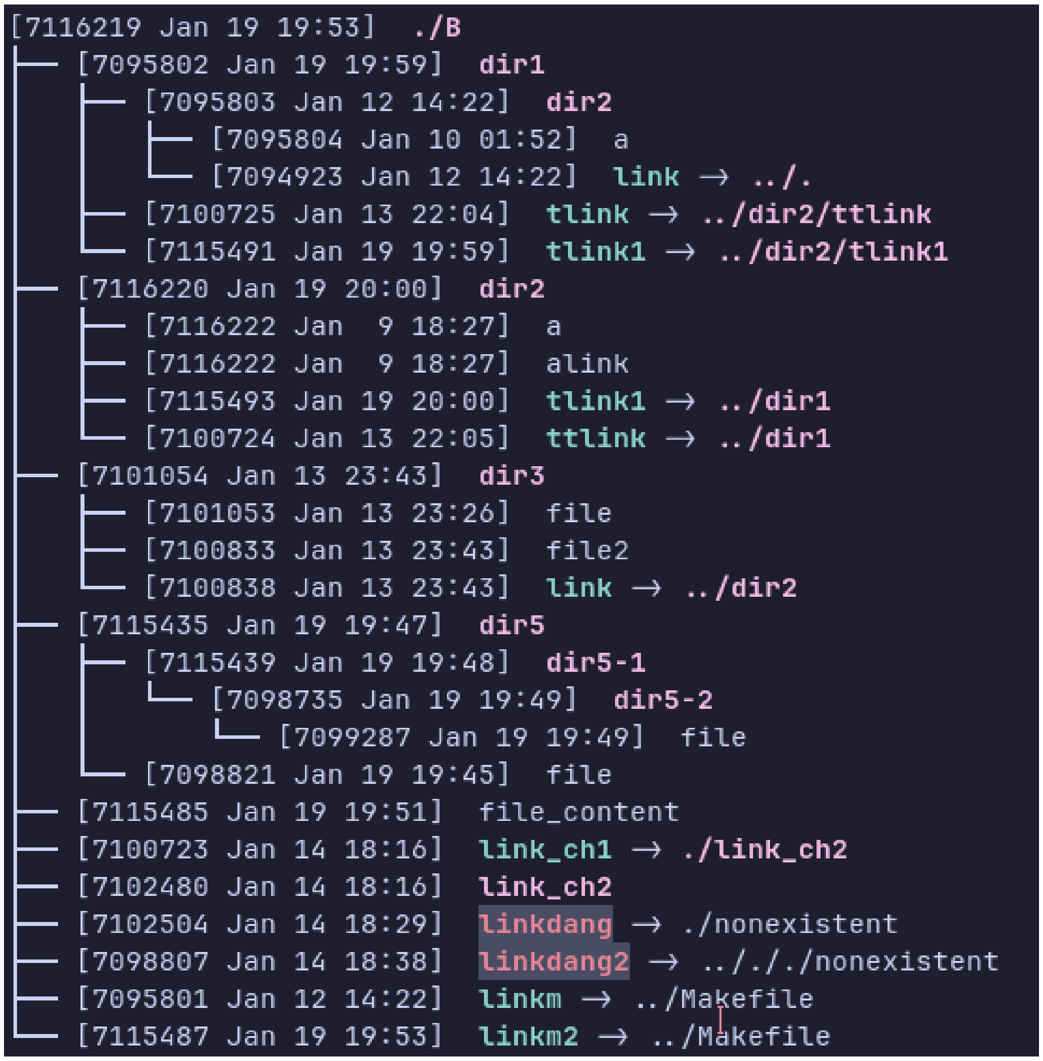
6.2.2 Detailed

Let's see directories A & B [Detailed].



```
[7116212 Jan 19 19:53]  ./A
├── [7116216 Jan 19 19:58]  dir1
│   ├── [7116217 Jan 13 15:15]  dir2
│   │   ├── [7096905 Jan 10 02:15]  a
│   │   └── [7099057 Jan 13 15:15]  link → ../.
│   ├── [7098971 Jan 13 22:02]  tlink → ../dir2/tlink
│   └── [7115490 Jan 19 19:58]  tlink1 → ../dir2/tlink1
├── [7092979 Feb  4 19:29]  dir2
│   ├── [7097440 Jan 10 23:20]  a
│   ├── [7097440 Jan 10 23:20]  alink
│   ├── [7099057 Jan 13 15:15]  hardlink → ../.
│   ├── [7100722 Jan 13 22:03]  tlink → ../dir1
│   └── [7115455 Jan 19 20:01]  tlink1 → .
├── [7100839 Jan 13 23:44]  dir3
├── [7100827 Jan 14 18:14]  dir4
├── [7097108 Jan 19 19:11]  dir5
├── [7115484 Jan 19 19:50]  file_content
├── [7100734 Jan 14 18:15]  link_ch1 → ./link_ch2
├── [7102479 Jan 14 18:15]  link_ch2 → ./dir4
├── [7101491 Jan 14 18:29]  linkdang → ./nonexistent
├── [7100502 Jan 14 18:38]  linkdang2 → .././././nonexistent
├── [7115482 Jan 19 19:52]  linkm → ../Makefile
└── [7096899 Jan 19 19:53]  linkm2 → ../README.pdf
```

Figure 3: Folder A [Detailed]

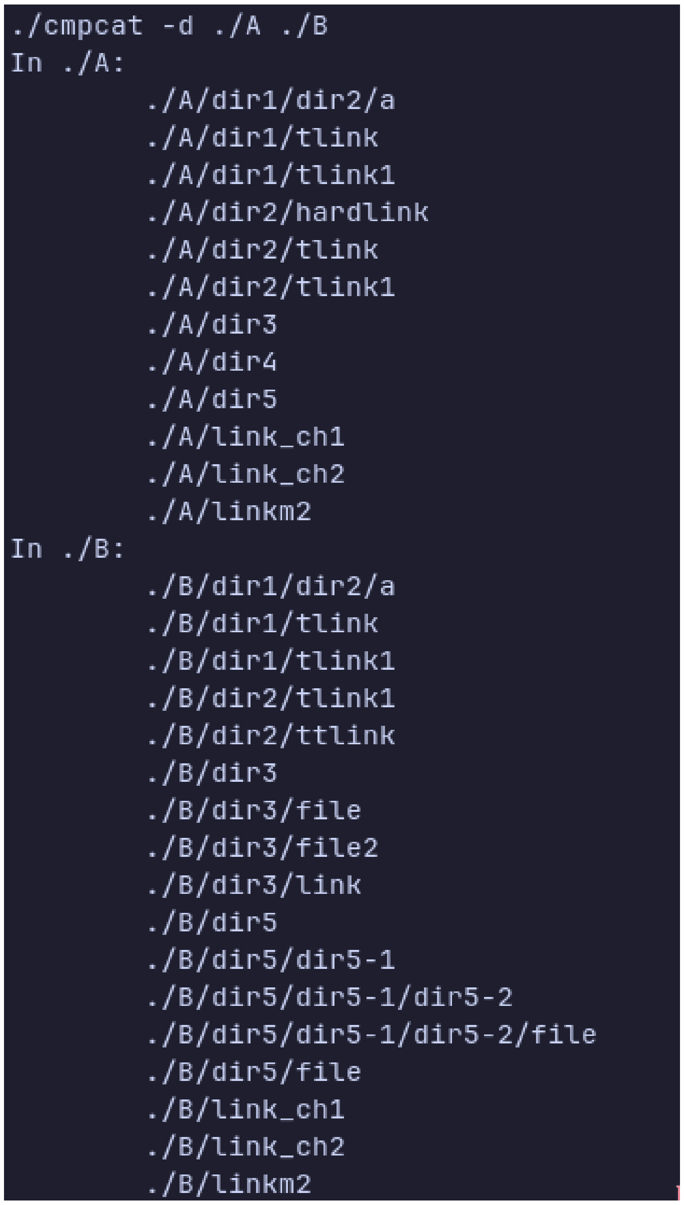


```
[7116219 Jan 19 19:53]  ./B
├── [7095802 Jan 19 19:59]  dir1
│   ├── [7095803 Jan 12 14:22]  dir2
│   │   ├── [7095804 Jan 10 01:52]  a
│   │   └── [7094923 Jan 12 14:22]  link -> ../.
│   ├── [7100725 Jan 13 22:04]  tlink -> ../dir2/ttlink
│   └── [7115491 Jan 19 19:59]  tlink1 -> ../dir2/tlink1
├── [7116220 Jan 19 20:00]  dir2
│   ├── [7116222 Jan  9 18:27]  a
│   ├── [7116222 Jan  9 18:27]  alink
│   ├── [7115493 Jan 19 20:00]  tlink1 -> ../dir1
│   └── [7100724 Jan 13 22:05]  ttlink -> ../dir1
├── [7101054 Jan 13 23:43]  dir3
│   ├── [7101053 Jan 13 23:26]  file
│   ├── [7100833 Jan 13 23:43]  file2
│   └── [7100838 Jan 13 23:43]  link -> ../dir2
├── [7115435 Jan 19 19:47]  dir5
│   ├── [7115439 Jan 19 19:48]  dir5-1
│   │   └── [7098735 Jan 19 19:49]  dir5-2
│   │       └── [7099287 Jan 19 19:49]  file
│   └── [7098821 Jan 19 19:45]  file
├── [7115485 Jan 19 19:51]  file_content
├── [7100723 Jan 14 18:16]  link_ch1 -> ./link_ch2
├── [7102480 Jan 14 18:16]  link_ch2
├── [7102504 Jan 14 18:29]  linkdang -> ./nonexistent
├── [7098807 Jan 14 18:38]  linkdang2 -> .././/nonexistent
├── [7095801 Jan 12 14:22]  linkm -> ../Makefile
└── [7115487 Jan 19 19:53]  linkm2 -> ../Makefile
```

Figure 4: Folder B [Detailed]

6.3 Compare output

Here we can see the differences between directory A and directory B.



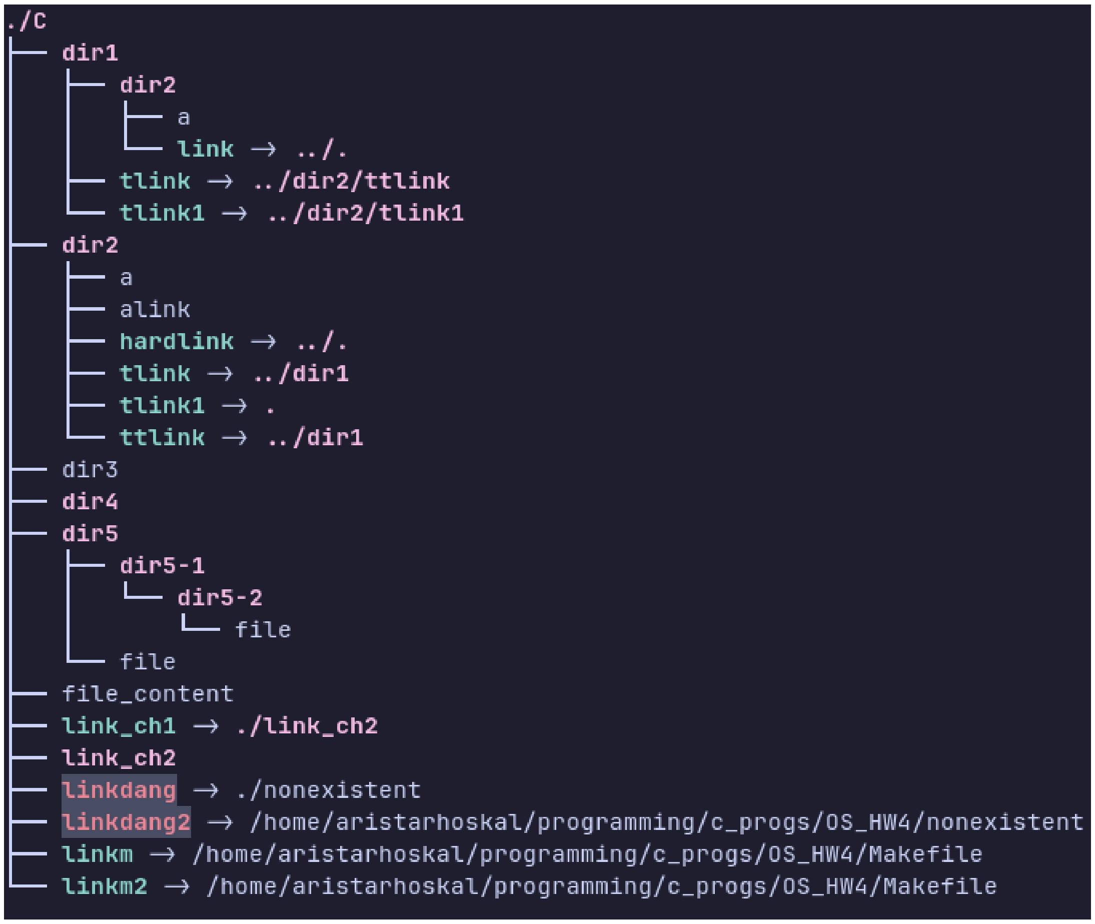
```
./cmpcat -d ./A ./B
In ./A:
        ./A/dir1/dir2/a
        ./A/dir1/tlink
        ./A/dir1/tlink1
        ./A/dir2/hardlink
        ./A/dir2/tlink
        ./A/dir2/tlink1
        ./A/dir3
        ./A/dir4
        ./A/dir5
        ./A/link_ch1
        ./A/link_ch2
        ./A/linkm2
In ./B:
        ./B/dir1/dir2/a
        ./B/dir1/tlink
        ./B/dir1/tlink1
        ./B/dir2/tlink1
        ./B/dir2/ttlink
        ./B/dir3
        ./B/dir3/file
        ./B/dir3/file2
        ./B/dir3/link
        ./B/dir5
        ./B/dir5/dir5-1
        ./B/dir5/dir5-1/dir5-2
        ./B/dir5/dir5-1/dir5-2/file
        ./B/dir5/file
        ./B/link_ch1
        ./B/link_ch2
        ./B/linkm2
```

Figure 5: Compare output

6.4 Merge output

Here we cam see the merged directory.

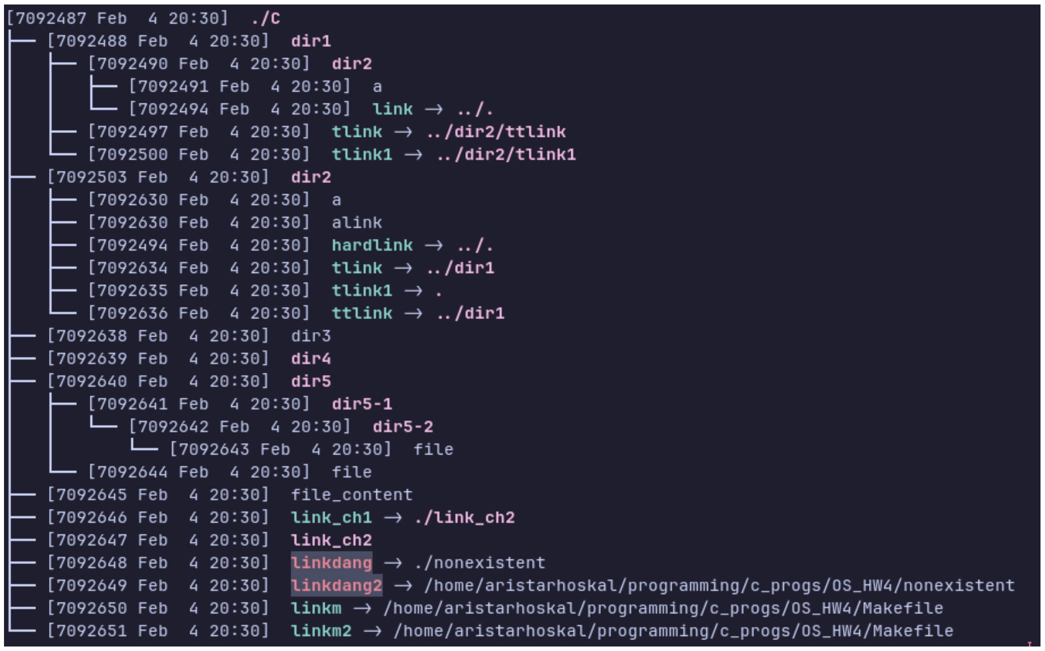
6.4.1 Simple



```
./C
├── dir1
│   ├── dir2
│   │   ├── a
│   │   └── link → ../.
│   ├── tlink → ../dir2/ttlink
│   └── tlink1 → ../dir2/tlink1
├── dir2
│   ├── a
│   ├── alink
│   ├── hardlink → ../.
│   ├── tlink → ../dir1
│   ├── tlink1 → .
│   └── ttlink → ../dir1
├── dir3
├── dir4
├── dir5
│   ├── dir5-1
│   │   └── dir5-2
│   │       └── file
│   └── file
├── file_content
├── link_ch1 → ./link_ch2
├── link_ch2
├── linkdang → ./nonexistent
├── linkdang2 → /home/aristarhoskal/programming/c_progs/OS_HW4/nonexistent
├── linkm → /home/aristarhoskal/programming/c_progs/OS_HW4/Makefile
└── linkm2 → /home/aristarhoskal/programming/c_progs/OS_HW4/Makefile
```

Figure 6: Merged Directory C [Simple]

6.4.2 Detailed



```
[7092487 Feb  4 20:30]  ./C
├── [7092488 Feb  4 20:30]  dir1
│   ├── [7092490 Feb  4 20:30]  dir2
│   │   ├── [7092491 Feb  4 20:30]  a
│   │   └── [7092494 Feb  4 20:30]  link -> ../.
│   ├── [7092497 Feb  4 20:30]  tlink -> ../dir2/ttlink
│   └── [7092500 Feb  4 20:30]  tlink1 -> ../dir2/tlink1
├── [7092503 Feb  4 20:30]  dir2
│   ├── [7092630 Feb  4 20:30]  a
│   ├── [7092630 Feb  4 20:30]  alink
│   ├── [7092494 Feb  4 20:30]  hardlink -> ../.
│   ├── [7092634 Feb  4 20:30]  tlink -> ../dir1
│   ├── [7092635 Feb  4 20:30]  tlink1 -> .
│   └── [7092636 Feb  4 20:30]  ttlink -> ../dir1
├── [7092638 Feb  4 20:30]  dir3
├── [7092639 Feb  4 20:30]  dir4
├── [7092640 Feb  4 20:30]  dir5
│   ├── [7092641 Feb  4 20:30]  dir5-1
│   │   └── [7092642 Feb  4 20:30]  dir5-2
│   │       └── [7092643 Feb  4 20:30]  file
│   └── [7092644 Feb  4 20:30]  file
├── [7092645 Feb  4 20:30]  file_content
├── [7092646 Feb  4 20:30]  link_ch1 -> ./link_ch2
├── [7092647 Feb  4 20:30]  link_ch2
├── [7092648 Feb  4 20:30]  linkdang -> ./nonexistent
├── [7092649 Feb  4 20:30]  linkdang2 -> /home/aristarhoskal/programming/c_progs/OS_HW4/nonexistent
├── [7092650 Feb  4 20:30]  linkm -> /home/aristarhoskal/programming/c_progs/OS_HW4/Makefile
└── [7092651 Feb  4 20:30]  linkm2 -> /home/aristarhoskal/programming/c_progs/OS_HW4/Makefile
```

Figure 7: Merged Directory C [Detailed]

7 Usage

To compare the folders execute:

```
$ ./cmpcat -d <folderA> <folderB>
```

or to merge the folders execute:

```
$ ./cmpcat -d <folderA> <folderB> -s <output_folder> [-f]
```

If the output directory already exists, with the -f flag we delete it. Otherwise an error message is shown.

Though, a makefile is provided with the following parameters:
make - Compilation and linking of the program.
make run - Execution of the program (with the default command line arguments).
make leaks - Execution of the program using valgrind.
make clean - Deletion of object files and the executable created during compilation.