Stage	1	2	3	4	Sum
Points	4	4	4	4	16
Result					

L1: Caecilius

Lucius Caecilius Iucundus is a Roman banker living in Pompei in the 1st century CE. As his banking business expands, he finds his current record keeping books insufficient. He bought a used computer from a Greek merchant. Unfortunately, the computer lacks any file management capabilities. Caecilius is unable to obtain any such program, as there is no internet connection and software vendors in Pompei. Without it, he cannot use the computer for any useful task. Write him a file management application according to his requirements.

A file with a template for the task is provided. Place the code for each stage in the corresponding function (you can create helper function and call them from `*_stageX`, and you can call `*stageX` from `*stageY`).

You should free all unnecessary resources in each stage, e.g. you should close files which are no longer necessary. If a given OS function can return an error, you check if the function executed successfully (if not the program can display an error message and be aborted).

Console line interface

The application should display the following available commands:

- `A. write`
- `B. show`
- `C. walk`
- `D. exit`

and then wait for user input. The user should be able to input the letter of one of these commands (lower- and upper-case letters should be allowed). If the user types something else, display an error about an invalid command and return 1 from `interface_stage1`.

If the command is correct, load a string with a path from `stdin` and check whether it points to an existing file (use the `stat` function). If not, print an error and return 1 from `interface_stage1`. If the command is correct, call its related function (`*_stageX`) with appropriate arguments, and return 1 from `interface_stage1`. The command `exit` is an exception in that it should cause `interface_stage1` to return 0 immediately.

File overwriting

After entering `write_stage2`, check whether the given file is a regular file. If not, print an error and return. If it is a regular file, remove it. Then create a new file with the same path. Wait for user input on `stdin`, replace all lower-case letters with upper case and write each line into the newly created file. If the line read is empty, close the file and return from the function.

File and directory reading

The function `show_stage3` should work for both directories and regular files. If arguments point to a directory, print names of all the files in this directory. If it points to a regular file:

1. load its contents and print it onto `stdout`,
2. print the size of the file in bytes

3. user ID of the owner
4. group ID of the owner

Use only **low level** IO API for file reading in this function, i.e. `open`, `close` and `read`. You might find `bulk_read` useful. If the function arguments point to a file which is neither a directory nor a regular file, print a message that the type of the file is unknown and return.

Directory walk

`walk_stage4` function displays directory contents recursively. For each file display

- if the file is a regular file, ‘N + 1‘ spaces, where ‘N‘ is the recursion depth
- if the file is a directory, ‘N‘ spaces and ‘+‘,
- name of the file (not its path!).

An example output should look like this:

```
+domus
  liber.pdf
 +tabulae
 +picturae
   feles.png
   canis.jpg
```

Use `ntfw`, do not implement the recurrence manually. Etapy:

1. [4 p.] `interface_stage1` function
2. [4 p.] `write_stage2` function
3. [4 p.] `show_stage3` function
4. [4 p.] `walk_stage4` function