SLP-assignment #7: printdir

(12 points, in groups of two)

Develop a program `printdir` that - similar to the UNIX command `ls(1)` - lists the contents of different directories.

It is recommended to approach the problem step-by-step:

(a) First, write a program that prints all entries of the current directory (`'.'`) line by line. Entries that have a name starting with `'.'` (i.e., hidden files) should be ignored. (`opendir(3)`, `readdir(3)`, `closedir(3)`)

(b) Extend the program to also print out the file size right in front of the name. Size and name should be divided by a tabulator (`'\t'`). At the end of the output, the total number of files and their combined size should be printed. For these two values you can ignore all non-regular files. (`lstat(2)`)

(c) Now evaluate the parameters `argv`. All passed parameters have to interpreted as paths to directories and be handled like in (a) and (b). The output of a directory should start with `'<Directory_Name>:\n'`. If no parameter is given, the current directory should be listed.

Hints:

- Your program only needs to handle file names and paths up to a length of 1024 characters[1]. Make sure to provide a suitable error message when this limit is exceeded.
- The functions for dealing with strings from `string.h` are of great use for this assignment.
- Always give a reason why you use the `volatile` keyword. If the same reasoning holds for multiple variables, you can justify them together.
- In the directory `/proj/i4spic/<login>/pub/aufgabe7/` you will find the file `printdir` that contains a reference implementation.
- Always make sure to give out meaningful error messages on the standard error stream. (`fprintf(stderr,...)(3)` / `perror(3)`)
- You can test your program with `valgrind`. This may help when searching for errors. *Suppressed errors* can be ignored. More error messages can be obtained by using `valgrind` with the flags `--leak-check=full --show-reachable=yes` and building the binary program with debug symbols.
- Your program has to compile with the following flags:
  `gcc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -o printdir printdir.c`
  This configuration is also used for grading.
- Functions of the `libc` that do not require error handling in SLP can be seen online in the Linux `libc`-Doku.
- You are free to write a `makefile` that includes instructions on how to build the program with the tool `make`. To do this, you can create a file called `Makefile` inside the submission directory (`aufgabe7/`). In the first line write
  `CFLAGS = -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3`
  Then you can build the file from the terminal by calling `make printdir` or with the `make` button inside the SPiC-IDE.

[1] Alternatively `PATH_MAX` from `limits.h` can be used.

Example Output

```
$ cd /proj/i4spic/<login>/pub/aufgabe7
$ ./printdir test/first_path test/second_path
test/first_path:
157     file2.txt
127     file1.txt
4096    test_dir
2 Files; 284 Bytes
test/second_path:
115     fileB.txt
4096    dir2
116     fileA.txt
4096    dir1
2 Files; 231 Bytes
```

You can test your implementation for the same directory and should get the same output:

```
$ cd /proj/i4spic/<login>/aufgabe7/
$ ./printdir /proj/i4spic/<login>/pub/aufgabe7/test/first_path \
             /proj/i4spic/<login>/pub/aufgabe7/test/second_path
[...]
```

This, however, does not replace extensive testing with other directories.

Deadline

T01	29.06.2026	18:00:00
SLP	01.07.2026	18:00:00

Autonomous Solving

All tasks must be completed independently. The generative use of AI tools of any kind is prohibited.