Arguments

Before walking through directories, the optional arguments are parsed converted into a bit-mask: the `arg_mask`. Every argument maps to a unique number from 0 to 13; if the *i*-argument is passed, the *i*-bit of a`arg_mask` is 1.

In this way it is easier to pass argument to other functions and less memory is required.

Arguments can be given in three ways:

- all in one (e.g. `-adf`);
- one by one (e.g. `-a -d -f`);
- separated by other arguments (e.g. `-a --help -d -- "path" -f`).

If a directory has the same name of an argument, the path has to be passed with `--` before.

The Printing-Process

The program does not print one string all in once, it instead print several pieces one by one, in the following order.

1. The first thing to be printed are the whitespaces:
   - the ‘ ’ character and three whitespaces for every level above this file;
   - *or*, if this is the last file of the super-level, the ‘ ’ character is replaced with another whitespace.

   To determine which super-levels are came to their last file, a `level_mask` is used. It is a binary number: if the *i*-level is terminated, the i-bit is set to 1.
2. If at least one argument that requires brakets to be printed has been give, the opening square braket ‘[’ is printed.
3. Optional arguments are printed.
4. The closing braket ‘]’ is printed.

This choice requires not to create a lot of strings as they will be printed right on-the-go. I could have used an array of pointers to functions to avoid repeating checks, but, as I don’t see that being doing much around the web, I prefered to maintain the old-fashioned if-else solution.

LS_COLORS

I read in the documentation that `tree` colors files using the `LS_COLORS` environment variable. I tried to understand how it worked and I understood the following: *you can write that varible in the format* *`file_type=terminal_color_format: ...others`*. So I parsed that variable with this convinction. I have to implement a dictionary, but I had no time so I execute the parsing every time.

I didn't implement all file types yet.

The Sorting-Process

At first, the function walks through a directory and makes a double-linked-list out of it. A node of the list is a custom-struct called `file_node` and it represents a file. Every node is inserted at the bottom and is raised up until its position respect all sorting parameters (`-r`, `-t`, `--dirsfirst` and default).

I've notice that `tree` functions sorts alphabetically by default and it is case *in*-sensitive. Moreover, some non-letterals characters, when they compare at the beginning of the string, are ignored while others are not. I tried to understand which comparing method the tree function uses and I made my researches but I found nothing, thus I used the standard comparison method. I used the `strcasecmp` instead of `strcmp` to keep the case insensitive at least.