Assignment 1: 42sh - Additional Tipps

About the shell

1. It is not necessary that your shell implements advanced features using '*', '?', or '~'.
2. If you do not know how to start, it is best to first start with simple commands, i.e., the node type `COMMAND`.

```c
if (node->type == NODE_COMMAND)
{
    char *program = node->command.program;
    char **argv = node->command.argv;
    // here comes a good combination of fork and exec
    ...
}
```

3. A shell usually supports redirections on all places of a simple command: 'ls > foo' and '>foo ls' are equivalent. The environment of your shell only supports 'ls > foo'.
4. Within a 'pipe' construction, all parts must be forked, even if they only contain built-in commands. This keeps the implementation easier.

```
exit 42 # closes the shell
exit 42 | sleep 1  # exit in sub-shell, main shell remains

cd /tmp # changes the directory
cd /tmp | sleep 1  # change directory in sub-shell, main shell does not
```

About the 'arena_*' functions

We have provided two simple extra libraries for you: `mc.c` and `arena.c`.

The first is a library that handles memory in a easier way. To start using `mc` you can call `mc_init`, with that return value you can call all `mc_*` functions. These functions enable you to register a tuple of a pointer and function that should be able to free this pointer in the `mc` struct. This way you can deallocate all memory in such a struct by calling `mc_free_all_mem`. Please see the `mc.h` file for more documentation.

The `arena` library uses this function to implement a stack of memory arena's. If we push a new arena, we have a clean space were we can allocate (or register) memory. However as this memory is all saved inside a `mc` we don't have to worry about cleaning it up, we only have to worry about popping our arena's. You might be tempted to think that you never need to pop these arena's, however please note that this is the same as never freeing your memory, which of course is bad practice. So it is good practice to keep these arena's small. Also note that unlike normal memory arena's, this library is not quick, as it uses a linked list of pointers as backing storage. So freeing them is `O(N)` (where `N` is the amount of allocations), so using arena's for small helper functions is probably a bad idea.

If you choose to use the `arena` library, please note there is a `arena_pop_all` function registered for using `atexit(3)`, this means all memory will always be cleaned at the end of your program. As described earlier it is still bad practice to not free old unused memory, however `valgrind` will not show this as an error. If you want to see these errors, to make sure your code is nice and tidy, please change the value of the `dealloc_on_pop_all` variable. If you set this variable to 0, memory will **NOT** be freed when `arena_pop_all` is called, we will simply remove the pointer to the memory creating a memory leak. A new helper function to make sure memory is freed when `arena_pop_all` is called in child processes but not in the main process is something like this:

```c
pid_t my_fork(void)
{
    pid_t p = fork();
    // In parent proc. it is 0, in child it is 1.
    dealloc_on_pop = !p;
    return p;
}
```

You are allowed to change, add and remove functions from these libraries. However you should make sure that the calls in `front.c` are still valid.

About the environment

- if `CLANG` is not available, use `make CC=gcc`
- in case of `OSX`: `make LIBS=-lreadline`
- in case of `OSX`: `valgrind --dsymutil=yes`
- your own private git repository at the UvA: https://gitlab-fnwi.uva.nl/