MicroShell

typedef struct s_cmd

```
char **args
int fds[2]
short pipe
struct s_cmd *next
```

Utils

```
int strlen(char *str)
{
    int count = -1;

    while (str[++count] ;
    return (count);
}
```

```
void _strerror(char *msg)
{
    write(2, msg, strlen(msg);
}
```

```
void fatal()
{
    _strerror("error: fatal\n");
    exit(1);
}
```

```
void clean_all(t_cmd *cmds)
{
    t_cmd *tmp;

    while (cmds)
    {
        free(cmds->args);
        tmp = cmds->next;
        free(cmds);
        cmds = tmp;
    }
}
```

Run

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
```

```
int main(int ac, char *av[], char **envs):

int i = 0;
t_cmd *cmds = NULL;
t_cmd *tmp;
```

```
while (++i < ac)
{
    if (!strcmp(av[i], ";"))
        continue ;
    if (!cmds)
    {
        cmds = parse_cmd(av, &i);
        tmp = cmds;
    }
    else
    {
        tmp->next = parse_cmd(av, &i);
        tmp = tmp->next;
    }
}
```

```
if (cmds)
{
    executor(cmds, envs);
    clean_all(cmds);
}
```

EXECUTION :

```
void executor(t_cmd *cmds, char **envs)
{
    int fdpipe[2];

    while (cmds)
    {
        if (cmds->pipe)
        {
            if (pipe(fdpipe))
                fatal();
            cmds->fds[1] = fdpipe[1];
            cmds->next->fds[0] = fdpipe[0];
        }
        execute(cmds, envs);
        cmds = cmds->next;
    }
}
```

```
void execute(t_cmd *cmd, char **envs)
{
    int pid;

    if (!strcmp(cmd->args[0], "cd"))
        return (cd(cmd->args + 1));
    if ((pid = fork()) == -1)
        fatal();
    if (!pid)
    {
        if (cmd->fds[1])
            dup2(cmd->fds[1], 1);
        if (cmd->fds[0])
            dup2(cmd->fds[0], 0);
        execve(cmd->args[0], cmd->args, envs);
        _strerror("error: cannot execute ");
        _strerror(cmd->args[0]);
        _strerror("\n");
        exit(1);
    }
    waitpid(pid, NULL, 0);
    if (cmd->fds[1])
        close(cmd->fds[1];
    if (cmd->fds[0])
        close(cmd->fds[0];
}
```

```
void cd(char **args)
{
    if (!args[0] || args[1])
        _strerror("error: cd: bad arguments\n");
    else if (chdir(args[0]))
    {
        _strerror("error: cd: cannot change
directory to ");
        _strerror(args[0]);
        _strerror("\n");
    }
}
```

PARSING :

```
t_cmd *parse_cmd(char *av[], int *i)
{
    t_cmd *_new;

    if (!(_new = malloc(sizeof(t_cmd))))
        fatal();
    _new->args = NULL;
    _new->fds[0] = 0;
    _new->fds[1] = 0;
    _new->pipe = 0;
    _new->next = NULL;
    while (av[*i] && strcmp(av[*i], ";") &&
strcmp(av[*i], "|"))
        add_arg(_new, av[(*i)++]);
    if (av[*i] && !strcmp(av[*i], "|"))
        _new->pipe++;
    return (_new);
}
```

```
void add_arg(t_cmd *cmd, char *arg)
{
    int i;
    char **safe;

    if (!cmd->args)
    {
        if (!(cmd->args = malloc(2 * sizeof(
char*))))
            fatal();
        i = 0;
    }
    else
    {
        i = -1;
        while (cmd->args[++i]) ;
        safe = cmd->args;
        if (!(cmd->args = malloc((i + 2) * sizeof(
char*))))
            fatal();
        i = -1;
        while (safe[++i])
            cmd->args[i] = safe[i];
        free(safe);
    }
    cmd->args[i++] = arg;
    cmd->args[i] = 0;
}
```