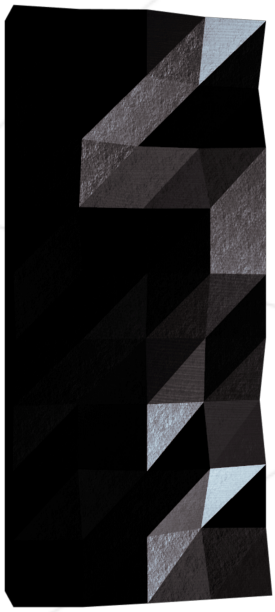

Pipex

Summary:
This project will let you discover in detail a UNIX mechanism that you already know by using it in your program.

Version: 2

Contents

Chapter I

Foreword

Cristina: `"Go dance salsa somewhere :)"`

Chapter II

Common Instructions

- Your project must be written in C.
- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a `0` if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, use cc, and your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, librairies or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}` if the subject does not specify anything else. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated Makefile. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done

after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Mandatory part

Program name	`pipex`
Turn in files	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`file1 cmd1 cmd2 file2`
External functs.	• `open, close, read, write, malloc, free, perror, strerror, access, dup, dup2, execve, exit, fork, pipe, unlink, wait, waitpid` • `ft_printf and any equivalent YOU coded`
Libft authorized	`Yes`
Description	`This project is about handling pipes.`

Your program will be executed as follows:

```
./pipex file1 cmd1 cmd2 file2
```

It must take 4 arguments:

- `file1` and `file2` are **file names**.
- `cmd1` and `cmd2` are **shell commands** with their parameters.

It must behave exactly the same as the shell command below:

```
$> < file1 cmd1 | cmd2 > file2
```

III.1 Examples

```
$> ./pipex infile "ls -l" "wc -l" outfile
```

Should behave like: `< infile ls -l | wc -l > outfile`

```
$> ./pipex infile "grep a1" "wc -w" outfile
```

Should behave like: `< infile grep a1 | wc -w > outfile`

III.2 Requirements

Your project must comply with the following rules:

- You have to turn in a `Makefile` which will compile your source files. It must not relink.
- You have to handle errors thoroughly. In no way your program should quit unexpectedly (segmentation fault, bus error, double free, and so forth).
- Your program mustn't have **memory leaks**.
- If you have any doubt, handle the errors like the shell command:
  `< file1 cmd1 | cmd2 > file2`

Chapter IV

Bonus part

You will get extra points if you:

- Handle multiple pipes.

  This:

  ```
  $> ./pipex file1 cmd1 cmd2 cmd3 ... cmdn file2
  ```

  Should behave like:

  ```
  < file1 cmd1 | cmd2 | cmd3 ... | cmdn > file2
  ```

- Support « and » when the first parameter is "here_doc".

  This:

  ```
  $> ./pipex here_doc LIMITER cmd cmd1 file
  ```

  Should behave like:

  ```
  cmd << LIMITER | cmd1 >> file
  ```

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter V

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.