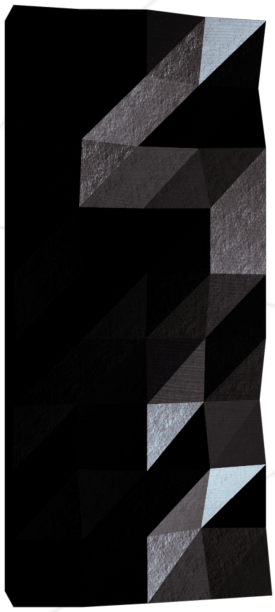

Pipex

Resumen: Este proyecto te permitirá descubrir, mediante el uso en tu programa, el funcionamiento de un mecanismo de UNIX que ya conoces.

Versión: 3.2

Índice general

Capítulo I

Introducción

Cristina: `"Ve a algún sitio a bailar salsa :)"`

Capítulo II

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) excepto en el caso de comportamientos indefinidos. Si esto sucede, tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, utilizar cc y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo III

Parte obligatoria

Nombre de programa	`pipex`
Archivos a entregar	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Argumentos	`archivo1 comando1 comando2 archivo2`
Funciones autorizadas	• `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`
Se permite usar libft	`Yes`
Descripción	`Este proyecto va sobre el manejo de pipes`

Tu programa deberá ejecutarse de la siguiente forma:

```
./pipex archivo1 comando1 comando2 archivo2
```

Deberá utilizar 4 argumentos:

- `archivo1` y `archivo2` son **nombres de arhivos**.
- `comando1` y `comando2` son **comandos de shell** con sus respectivos parámetros.

La ejecución del programa pipex deberá hacer lo mismo que el siguiente comando:

```
$> < archivo1 comando1 | comando2 > archivo2
```

III.1. Ejemplos

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

deberá hacer lo mismo que “<infile ls -l | wc -l >outfile”

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

deberá hacer lo mismo que “<infile grep a1 | wc -w >outfile”

III.2. Requisitos

Tu proyecto debe cumplir los siguientes requisitos:

- Debes entregar un `Makefile` que compile tus archivos fuente. No debe hacer relink.
- Debes gestionar los errores minuciosamente. De ninguna forma tu programa debe salir de forma inesperada (segmentation fault, bus error, double free, y similares).
- Tu programa no debee tener **fugas de memoria**.
- Si teienes alguna duda, gestiona los errores como lo hace el comando de shell: `<file1 cmd1 | cmd2 >file2`

Capítulo IV

Parte extra

- Gestionar múltiples pipes.

```
$> ./pipex archivo1 comando1 comando2 comando3 ... comandon archivo2
```

Deberá comportarse así:

```
$> < archivo1 comando1 | comando2 | comando3 ... | comandon > archivo2
```

- Aceptar << y >> cuando el primer parámetro es **“here_doc”**:

```
$> ./pipex here\_doc LIMITADOR comando comando1 archivo
```

Deberá comportarse así:

```
comando << LIMITADOR | comando1 >> archivo
```

Los bonus solo serán evaluados si tu parte obligatoria está PERFECTA. Con PERFECTA queremos naturalmente decir que debe estar completa, sin fallos incluso en el más absurdo de los casos o de mal uso, etc. Significa que si tu parte obligatoria no tiene TODOS los puntos durante la evaluación, tus bonus serán completamente IGNORADOS.

Capítulo V

Entrega y evaluación entre compañeros

Como de costumbre, entrega tu trabajo en tu repositorio `Git`. Solo el trabajo subido en tu repositorio será evaluado. Acúerdate de comprobar dos veces los nombres de tus archivos para asegurarte de que sean los correctos.

file.bfe:VACsSfsWN1cy33ROeASsmsgnY0o0sDMJev7zFHhw
QS8mvM8V5xQQpLc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZ0