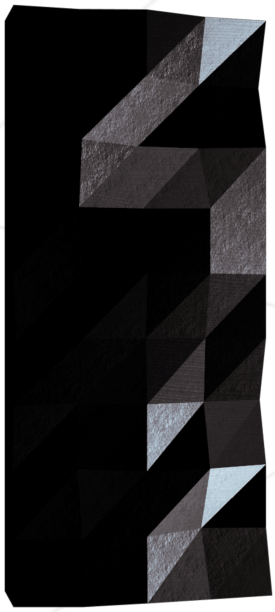

Pipex

Resumen: Este proyecto es útil para descubrir el funcionamiento de un mecanismo de UNIX que ya conoces.

Índice general

Capítulo I

Avance

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

Capítulo II

Instrucciones generales

- 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 dentro.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) ni teener comportamientos indefinidos. Si esto pasa tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria alocada en heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el subject lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, 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 a tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus compañeros. Si se encuentra un error durante la evaluación de Deepthought, la evalaución terminará.

- El nombre del ejecutable debe ser `pipex`.
- Debes gestionar los errores cuidadosamente. Bajo ningún concepto tu programa puede terminar inesperadamente (segfault, bus error, double free, etc). Si no lo tienes claro, gestiona los errores como los comandos de shell `<archivo1 comando1 | comando2 >archivo2`.
- Tu programa no puede tener leaks de memoria.
- Tienes permitido utilizar las siguientes funciones:
  - `access`
  - `open`
  - `unlink`
  - `close`
  - `read`
  - `write`
  - `malloc`
  - `waitpid`
  - `wait`
  - `free`
  - `pipe`
  - `dup`
  - `dup2`
  - `execve`
  - `fork`
  - `perror`
  - `strerror`
  - `exit`

Capítulo III

Objetivos

Tu objetivo es programar Pipex.
Deberá ejecutarse de la siguiente forma:

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

Por si acaso: archivo1 y archivo2 son nombres de archivos, comando1 y comando2 son comandos de shell con sus respectivos parámetros.

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

```
$> < 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”

Capítulo IV

Parte extra

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.

- Gestionar múltiples pipes:

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

Será equivalente a:

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

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

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

Será el equivalente a:

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

Capítulo V

Entrega y evaluación de compañeros

Como de costumbre, entrega tu trabajo en tu repositorio `Git`. Solo el trabajo subido en tu repositorio será evaluado.