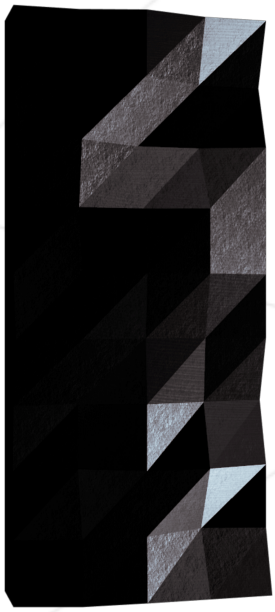

Get Next Line

Leer una línea de un descriptor de archivo es demasiado tedioso.

Resumen: El objetivo de este proyecto es simple: programar una función que devuelva una línea desde un descriptor de archivo (file descriptor).

Versión: 14.0

Índice general

Capítulo I

Objetivos

Este proyecto no solo te permitirá añadir una función bastante práctica a tu colección, también te hará aprender el increíble concepto de las variables estáticas en `C`.

Capítulo II

Instrucciones generales

- El proyecto deberá estar escrito en C.
- El proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas deberán estar incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Las funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc), excepto en el caso de comportamientos indefinidos. Si esto sucede, el proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en la pila (heap) deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, se deberá entregar un `Makefile` que compilará tus archivos fuente a la salida requerida con las flags `-Wall`, `-Werror` y `-Wextra`. También se deberá utilizar cc y, por supuesto, el `Makefile` no debe hacer relink.
- El `Makefile` entregado debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus del proyecto se deberá incluir una regla `bonus` en el `Makefile`, en la que se añadirán 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 el proyecto permite el uso de la `libft`, se deberá copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` del proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Es recomendable crear programas de prueba para el proyecto, aunque este trabajo **no será entregado ni evaluado**. Esto ofrece la oportunidad de verificar que el programa funciona correctamente durante las evaluaciones. Y sí, está permitido utilizar estas pruebas durante cualquier evaluación.
- Entrega el trabajo en el repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si el proyecto tiene que ser evaluado por Deepthought, la evaluación se realizará después de las evaluaciones personales. Si, durante la evaluación de Deepthought, se encuentra un error se, se interrumpirá su evaluación.

Capítulo III

Instrucciones sobre la IA

Contexto

Este proyecto está diseñado para ayudarte a descubrir los fundamentos que construirán tu formación en TIC (lo que conocemos como Tecnologías de la Información y la Comunicación).

Para afianzar los conocimientos y habilidades clave, es esencial adoptar un enfoque reflexivo sobre el uso de herramientas de IA.

El auténtico aprendizaje de los fundamentos requiere un esfuerzo intelectual real, a través de desafíos, repetición y el intercambio de conocimiento que procede del aprendizaje entre pares.

Para una visión más completa de nuestra postura sobre la IA (ya sea como herramienta de aprendizaje, como parte del plan de estudios de TIC o como una expectativa en el mercado laboral) puedes consultar las preguntas frecuentes dedicadas en la intranet.

Mensaje principal:

- Construir fundamentos sólidos sin atajos.
- Desarollar de forma real habilidades técnicas y transversales.
- Experimentar el aprendizaje entre pares de forma inmersiva, empezando por aprender a aprender y por resolver nuevos problemas.
- El proceso de aprendizaje es más importante que el resultado.
- Aprender sobre los riesgos asociados a la IA y desarrollar prácticas de control efectivas y medidas que neutralicen los errores comunes.

Reglas para estudiantes:

- Aplica la lógica y el razonamiento a las tareas asignadas, especialmente antes de recurrir a la IA.
- No deberías pedir respuestas directas a la IA.
- Infórmate sobre el enfoque global de 42 respecto la IA.

Resultados de esta etapa:

Durante esta fase de construcción de los fundamentos, conseguirás:

- Obtener una base adecuada en tecnología y en programación.
- Comprender por qué y cómo la IA puede ser peligrosa durante esta fase.

Comentarios y ejemplos:

- Sí, sabemos que la IA existe.Y si, también sabemos que puede resolver tus proyectos. Pero estás aquí para aprender, no para demostrar que la IA ha aprendido. No pierdas tiempo solo para demostrar que la IA puede resolver un problema determinado.
- Aprender en 42 no tiene nada que ver con saber una respuesta, sino con desarrollar las habilidades para encontrarla. La IA te dará la respuesta directa, lo que impide que desarrolles tu propio razonamiento. Razonar requiere tiempo, esfuerzo y cometer errores. Nadie dijo que el proceso iba a sér fácil.
- Ten en cuenta que, durante los exámenes, no tendrás acceso a la IA (no tenemos internet ni dispositivos inteligentes). Te vas a dar cuenta rápidamente si has confiado demasiado en la IA durante tu proceso de aprendizaje de la forma más directa: frente a una hoja en blanco donde vas a tener que escribir tu propio código.
- El aprendizaje entre pares te expone a diferentes ideas y enfoques, mejorando tus habilidades transversales y tu capacidad de pensar de forma diferente. Eso es mucho más valioso que sentarte a chatear con un bot. Así que, ¡sin miedo! Habla, haz preguntas y aprende con el resto de estudiantes.
- Sí, la IA formará parte del plan de estudios, tanto como herramienta de aprendizaje como tema en sí mismo. Incluso tendrás la oportunidad de crear tu propio software de IA. Para aprender más sobre nuestro enfoque progresivo, puedes consultar la documentación disponible en la intranet.

✓ Buenas prácticas:

Me atasco en un nuevo concepto. Le pregunto a alguien cercano cómo lo ha abordado. Hablamos durante 10 minutos y, de repente, todo encaja. Lo entiendo. No entiendo algo concreto del proyecto y no sé cómo continuar. Le pregunto a otra persona cómo lo ha abordado, hablamos sobre el tema y, si es necesario, incluso utilizamos otros métodos (papel y boli, dibujos, metáforas, etc.) hasta conseguir entenderlo.

✗ Mala práctica:

Utilizo la IA en secreto, copio un código que parece correcto. Durante la evaluación entre pares, no puedo explicar nada. Suspendo. Durante el examen, sin IA, me vuelvo a atascar. Suspendo.

Capítulo IV

Parte obligatoria

Nombre de función	`get_next_line`
Prototipo	`char *get_next_line(int fd);`
Archivos a entregar	`get_next_line.c, get_next_line_utils.c, get_next_line.h`
Parámetros	`fd: El descriptor de archivo que leer`
Valor devuelto	`Si todo va bien: la línea leída` `En caso de fallo o si la lectura termina: NULL.`
Funciones autorizadas	`read, malloc, free`
Descripción	`Escribe una función que devuelva la línea desde un descriptor de archivo`

- Llamar a la función `get_next_line` de manera repetida (por ejemplo, usando un bucle) permitirá leer el contenido del archivo hacia el que apunta el descriptor de archivo, **línea a línea**, hasta el final.
- La función deberá devolver la línea que se acaba de leer.
  Si no hay nada más que leer o si ha ocurrido un error, deberá devolver `NULL`.
- Hay que asegurarse de que la función se comporta adecuadamente cuando lea de un archivo y cuando lea de 'stdin'.
- Importante: la línea devuelta debe terminar con el `\n`, excepto si se ha llegado al final del archivo y éste no termina con un `\n`.
- En el archivo de cabecera `get_next_line.h` se deberá incluir tener como mínimo el prototipo de la función `get_next_line`.
- Se pueden añadir todas las funciones de ayuda que se necesiten en el archivo `get_next_line_utils.c`

Un buen comienzo sería saber qué es una variable estática.

- El programa debe compilar con el indicador (*flag*) `-D BUFFER_SIZE=n`. Este indicador se utilizará para determinar el tamaño del *buffer* de las lecturas de la función `read()` en el `get_next_line()`. Este parámetro será modificado por las personas que hagan la evaluación y por Moulinette para probar tu programa.

Se debería poder compilar este proyecto con y sin el indicador -D BUFFER_SIZE, junto a los indicadores habituales. Se puede elegir el valor por defecto que se prefiera.

- El programa se compilará de la siguiente forma (se utiliza como ejemplo un tamaño de *buffer* de 42):
`cc -Wall -Werror -Wextra -D BUFFER_SIZE=42 <archivos>.c`.
- Se considera que `get_next_line()` tiene un comportamiento indeterminado si el archivo al que apunta el descriptor de archivo ha cambiado desde la última vez que se llamó, siempre que `read()` no haya llegado al final del archivo.
- Se considera que `get_next_line()` tiene un comportamiento indeterminado cuando lo que se lee es un archivo binario. Sin embargo, es posible implementar alguna manera lógica de sortear este problema, si se desea.

¿Funciona correctamente get_next_line si el BUFFER_SIZE es 9999? ¿Y si es 1? ¿Qué tal con 10000000? ¿Sabes por qué?

Se debería leer lo menos posible cada vez que se llame a get_next_line(). Si hay un salto de línea, se debería devolver la línea actual. No hay que leer el archivo entero y luego procesar cada línea.

`Prohibido`

- No se permite el uso de `libft` en este proyecto.
- Se prohíbe el uso de `lseek`
- Se prohíbe el uso de variables globales.

Capítulo V

Requisitos del Readme

Debe incluirse un archivo `README.md` en la raíz del repositorio Git. Su propósito es permitir que cualquier persona que no esté familiarizada con el proyecto (pares, personal, responsables de selección, etc.) pueda entender rápidamente de qué trata el proyecto, cómo ejecutarlo y dónde encontrar más información sobre el tema.
El `README.md` debe incluir, como mínimo:

- La primera línea debe estar en cursiva y decir: *Este proyecto ha sido creado como parte del currículo de 42 por <login1>[, <login2>[, <login3>[...]]].*
- Una sección de "**Descripción**"que presente claramente el proyecto, incluyendo su objetivo y una breve visión general.
- Una sección de "**Instrucciones**"que contenga cualquier información relevante sobre compilación, instalación y/o ejecución.
- Una sección de "**Recursos**"que enumere referencias clásicas relacionadas con el tema (documentación, artículos, tutoriales, etc.), así como una descripción del uso de IA, especificando para qué tareas y en qué partes del proyecto se ha utilizado.

➠ **Podrían requerirse secciones adicionales dependiendo del proyecto** (por ejemplo, ejemplos de uso, lista de características, decisiones técnicas, etc.).

Cualquier contenido extra requerida se listará explícitamente a continuación.

- También debe incluirse una explicación detallada y la justificación del algoritmo seleccionado para este proyecto.

`La elección del idioma queda a tu criterio. Se recomienda escribir en inglés, pero no es obligatorio.`

Capítulo VI

Parte bonus

Este proyecto es bastante directo y no deja mucho margen a los bonus. Sin embargo, confiamos en la creatividad de cada estudiante. Si se ha completado la parte obligatoria, es hora de probar con estos bonus.

Aquí están los requisitos de la parte bonus:

- Desarrollo de `get_next_line()` con una sola variable estática.
- Capacidad de `get_next_line` para gestionar múltiples descriptores de archivos a la vez.
  Es decir, si hay tres descriptores de archivo disponibles para lectura (por ejemplo: 3, 4 y 5), se debería poder leer desde un descriptor distinto en cada llamada, sin perder el seguimiento del estado de lectura de cada descriptor ni devolver una línea perteneciente a otro descriptor.
  Esto significa que se debería poder utilizar `get_next_line` una vez sobre el fd 3, otra vez sobre el fd 4, y otra vez sobre el fd 5 de forma alterna, y así sucesivamente, sin perder el seguimiento del estado de lectura de cada descriptor de archivo.

Se debe añadir el sufijo `_bonus.[c\h]` a los archivos de la parte bonus.
Esto quiere decir que, además de los archivos de la parte obligatoria, hay que entregar los tres archivos siguientes:

- `get_next_line_bonus.c`
- `get_next_line_bonus.h`
- `get_next_line_utils_bonus.c`

La parte bonus solo será evaluada si la parte obligatoria está perfecta. "Perfecta"significa que se ha completado en su totalidad y funciona perfectamente sin ningún fallo. Si no se han completado TODOS los requisitos de la parte obligatoria, la parte bonus no será evaluada de ninguna manera.

Capítulo VII

Entrega y evaluación

Entrega tus ejercicios el directorio `Git`, como se hace habitualmente. Solo se evaluará el trabajo que haya dentro del repositorio. Se debe comprobar varias veces que el nombre de los archivos es el correcto.

```
Al hacer el proyecto, es importante:
1) Tanto el tamaño del buffer como el de la línea pueden tener
valores muy diferentes.
2) Un descriptor de archivo no solo apunta a archivos normales.
Alguien inteligente compraría y verificaría los ejercicios con otras
personas. Es más: prepararía una lotería de pruebas de cara a la
evaluación.
```

Una vez superado, no dudes en añadir `get_next_line()` a `libft`.

Durante la evaluación, es posible que se solicite una ligera **modificación del proyecto**. Esto puede consistir en ajustar ligeramente el comportamiento, modificar unas cuantas líneas de código o incorporar una característica fácil de implementar.

Puede que este paso **no sea necesario en todos los proyectos**, pero hay que tenerlo en cuenta si así se especifica en la hoja de evaluación.

Este paso sirve para verificar la comprensión real de una parte específica del proyecto. La modificación se puede realizar en cualquier entorno de desarrollo que se elija (por ejemplo, la configuración habitual), y debería ser factible en unos pocos minutos, a menos que se defina un plazo específico como parte de la evaluación.
Por ejemplo, se puede pedir hacer una pequeña actualización en una función o *script*, modificar lo que se vería en pantalla o ajustar una estructura de datos para almacenar nueva información, etc.

Los detalles (alcance, objetivo, etc.) se especificarán cada **hoja de evaluación** y pueden

variar de una evaluación a otra para el mismo proyecto.

/=∂/\/\[](_)§ /\/\@|V †|-|@¯|¯ /-/!570®1<|-\£1_`/ ¢@/\/\ε vv!7}{ ???