**get\_next\_line**

* **Crear un liberador de memoria para que cuando se necesite hacer un free, tan solo haya que llamar a la función. En este caso podemos usar el siguiente ejemplo:**

char \*ft\_free(char \*\*x)

{

If (x && \*x)

free (\*x);

\*x = NULL;

return (NULL);

}

* **Crear una función que se encargue de leer el archivo asociado al desciptor ‘fd’ y almacenar el contenido en el buffer. También manejar la expansión del buffer según sea necesario:**
  + Hacer el malloc del BUFFER\_SIZE + 1 a un puntero tipo char (en este caso podemos llamarlo \*read).
  + Comprobar que se haya realizado correctamente el malloc.
  + Abrir un bucle while el cual se ejecutará mientras no se encuentre un carácter de salto de línea en el buffer. Habrá que realizar una función externa (la ft\_strchr de la libft) que busque la primera ocurrencia de un carácter en una cadena.
  + Utilizar la función read para leer hasta BUFFER\_SIZE bytes desde el archivo asociado al descriptor ‘fd’ y almacenarlos en el buffer ‘\*read’. La cantidad de bytes leídos se puede guardar en una variable tipo int, creada anteriormente, llamada aux, por ejemplo.
  + Tendremos que comprobar que la lectura se haya realizado correctamente, para ello podríamos implementar un ‘if (aux == -1)’. Si aux es -1, la función retornaría un free en read y un free en buffer utilizando la función creada al principio. También retornaría un NULL, indicando un error.
  + Agregamos un carácter nulo al final del buffer ‘read’ para asegurarse de que sea una cadena de caracteres válida.
  + Para comprobar si se ha leído algo o si se ha llegado al final del archivo, habría que ver si en el primer carácter de ‘read’ es un nulo. Si es el caso, la función retornaría un free en read y devolvería el buffer actual.
  + En este paso, podríamos utilizar la función ft\_strjoin de la libft para concatenar el contenido de ‘read’ al final de ‘buffer’, así acumularíamos el contenido leído en el buffer principal. En el caso de que hubiera algún error, se devolvería un NULL.
  + Si la función anterior devolviera un NULL, indicaría un error durante la concatenación. En ese caso, devolveriamos un free a ‘read’ y ‘buffer’, y un NULL.

Como resumen, estaríamos creando un while que lee el archivo y acumula el contenido el en buffer hasta que se encuentra un carácter de nueva línea ‘\n’ en el buffer o se llega al final del archivo. Cada vez que se lee, se actualiza el buffer y se verifica si ocurrieron errores durante la operación. En esta función utilizaríamos 2 funciones externas, las cuales se pueden poner en get\_next\_line\_utils.c, llamadas en este caso: ft\_strchr y ft\_strjoin.

* **Crear una función que toma el buffer actual y elimina la primera línea encontrada, devolviendo el resto del contenido. Se tiene que encargar de gestionar la memoria adecuadamente.**
  + Primero que todo comprobaríamos la existencia del buffer. En caso de que no existiera devolveríamos un NULL.
  + Se haría un bucle que avanza hasta encontrar el carácter de salto de línea ‘\n’ o el final de línea ‘\0’.
  + Cuando se encuentre un carácter de nueva línea, se incrementará la i para apuntar al siguiente carácter después del salto de línea.
  + Habrá que asignar memoria a la nueva cadena que devolveremos al final ‘result’, para ello se utilizará malloc. La cantidad de memoria asignada es calculada restando ‘i’ de la longitud total de ‘buffer’ (hasta el final de la cadena) y sumando 1 para el carácter nulo.
  + Verificar si se ha asignado la memoria. En caso contrario, devolver NULL.
  + Copiar el contenido restante de ‘buffer’ (después del carácter de nueva línea) a ‘result’. Se incrementan tanto ‘i’ (caracteres de ‘buffer’) como ‘j’ (caracteres de result).
  + Agragar el carácter nulo al final de la cadena ‘result’.
  + Liberar memoria con ‘free’ el buffer.
  + Devolver ‘result’.

En resumen, la función elimina la primera línea de la cadena de caracteres ‘buffer’ y devuelve una nueva cadena sin esa línea, liberando la memoria del buffer original.

* Crear la función principal get\_next\_line. Lee y devuelve la próxima línea del archivo asociado al descriptor ‘fd’. También gestiona la memoria y actualiza el buffer.
  + Crear una variable estatica para almacenar el contenido del archivo entre llamadas a la función. Tiene que declararse estática para que retenga su valor entre llamadas a la función.
  + Verificar si el descriptor de archivo ‘fd’ es válido y si el tamaño del búfer (‘BUFFER\_SIZE’) es mayor que cero. Si alguna de estas condiciones no se cumple, la habrá que retornar NULL indicando que ha habido un error.
  + Comprobar si el buffer es NULL o no contiene carácter de nueva línea ‘\n’, se llama una función externa (una función llamada, por ejemplo, Reading) para leer más contenido del archivo y actualizar el buffer. En caso de que la lectura no haya tenido éxito, retornar NULL.
  + Extraer de la línea el buffer, es decir, si hay un carácter de nueva línea ‘\n’ en el buffer, utilizar una función externa para extraer la línea hasta ese punto (se puede usar ft\_substr de la libft). En el caso de que no haya un carácter de nueva línea, se extrae la línea hasta el final del buffer.
  + Comprobar si la asignación de memoria para la línea tiene éxito, en caso contrario retornar NULL.
  + Llamar a la función remove\_line para eliminar la línea que se acaba de extraer del buffer.
  + Comprobar si el buffer está vacío (mirando si su primer carácter es nulo). En caso de estar vacío liberar la memoria asociada a ese buffer utilizando la función free.
  + Retornar la línea.

En resumen, esta función, como pide el enunciado, se encarga de leer líneas completas desde un archivo, actualizando un buffer estático entre llamadas, extrayendo líneas y gestionando la memoria de manera dinámica.