

Sistemas Operativos

[Práctica 1 – PROGRAMACIÓN C]

LUCÍA DOMÍNGUEZ RODRIGO, MARCOS JIMÉNEZ PULIDO

Tabla de contenido

[Autores 2](#_Toc115812632)

[Descripción del Código 3](#_Toc115812633)

[Diseño del Código 3](#_Toc115812634)

[Principales Funciones 3](#_Toc115812635)

[Casos de Prueba 3](#_Toc115812636)

[Comentarios Personales 4](#_Toc115812637)

# Autores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Apellidos | Grado | Correo Institucional | Usuario Github |
| Lucía | Domínguez Rodrigo | GII + GIS | l.dominguez.2021@alumnos.urjc.es | [LuciaDominguezRodrigo](https://github.com/LuciaDominguezRodrigo) |
| Marcos | Jiménez Pulido | GII + GIS | m.jimenezp.2021@alumnos.urjc.es | [MarJ03](https://github.com/MarJ03) |

# Descripción del Código

## Diseño del Código

De manera general, se ha implementado el código con las funciones predeterminadas de C explicadas en clase. En nuestro caso, contamos con las tres funciones principales explicadas en la práctica, y con dos funciones auxiliares para poder representar de una manera más modular y limpia el código de la función ***longlines***. Se ha definido una constante que indica el tamaño total que puede tener una línea, ya que era utilizado en muchas partes del código (referenciado en el mismo como MAX\_LINE\_LENGTH).

***Función Head***

Para esta función se ha desarrollado en código la siguiente idea: al tener que imprimir por salida estándar las primeras N líneas, se inicializa un contador a 0, y mientras ese contador no supere N, se va llenando y sobrescribiendo un buffer (cuyo tamaño máximo es el indicado en clase, 1024 bytes), el cual se va leyendo y mostrando en la salida estándar, incrementando el contador en cada iteración.

La función devolverá un valor numérico 0 si la ejecución ha sido correcta, y un valor numérico 1 si ha habido algún error (como por ejemplo si ha habido algún error de lectura).

***Función Tail***

Para esta función se ha desarrollado en código la siguiente idea: primeramente, se inicializa un buffer dinámico que contiene punteros a *char* (que apuntarán a su vez a las frases elegidas para imprimir por salida estándar), devolviendo un error numérico con valor 1 en caso de que la asignación de memoria sea errónea. Tras esto, se reserva memoria para cada frase apuntada por cada posición del buffer, es decir, las cadenas de caracteres donde irán alojadas las frases elegidas, liberando estas posiciones de memoria e imprimiendo un error numérico con valor 2 si alguna de las asignaciones no es correcta.

A continuación, se procede a la lectura de las líneas, inicializando un contador a 0 y aumentando su valor en 1 en cada línea correctamente leída. Cada una de las líneas leídas se almacenará en una posición del buffer, dependiendo del orden de llegada (por ejemplo: si se quiere mostrar las 5 últimas líneas, y cuenta con 8 líneas de entrada, la sexta línea se almacenará en la posición del buffer número 0 (entendiendo que el buffer va del 0 al N-1, en este caso, 7)). De esta forma, sólo se usa el tamaño de memoria requerido para mostrar las N líneas necesarias, independientemente del tamaño de la entrada.

Como aspecto crucial de esta función, destacamos que, a la hora de imprimir las líneas por salida estándar, se comienza a imprimir por la última casilla del buffer sobrescrita incrementada una unidad (ya que esa fue la primera frase de las N en introducirse en el buffer). Cabe destacar que, tal y como se ve en el código, se utiliza en varias partes ***“%N”*** para no acceder a espacios de memoria no permitidos.

Por último, se libera primeramente la memoria reservada para almacenar las frases anteriores, y posteriormente se elimina el buffer que contiene los punteros a las líneas.

***Función Longlines***

Para esta función se ha desarrollado en código la siguiente idea: primeramente, se reserva memoria para almacenar la lista de N líneas (cabe destacar que dicha “lista” simulará un TAD lista ordenada de N posiciones), devolviendo un valor de error numérico 1 en caso de que la asignación de memoria resulte fallida. A continuación, se inicializa un buffer para almacenar cada línea leída y se leen por entrada estándar cada una de las líneas. A la vez, se inicializa el tamaño de la lista, el cual va creciendo según se añadan elementos a dicha lista. Tras esto, se leen cada una de las líneas de la entrada estándar, se crea una instancia del tipo *Line* con su contenido, y se insertan de forma ordenada en la lista de líneas declaradas.

Cuando no quedan más líneas por leer, imprime la lista, que contendrá las N líneas ordenadas requeridas en stdout. Cuando imprime cada línea, libera el espacio utilizad por la misma, y al acabar de imprimir todas las líneas, libera el espacio ocupado por la lista.

***Estructura del programa Test***

En la ejecución de la función main() del programa *Test*, primero se realiza un control del número de argumentos introducidos al ejecutar alguna de las funciones. En el caso de haber más de 3 argumentos, el programa lanza un error numérico de valor 1, indicando al usuario que el número de argumentos no es correcto. En el caso de haber 3 argumentos, se toma como tercer argumento el número de líneas (siendo el primero la dirección del ejecutable, y el segundo la extensión de la función a ejecutar), que se convierte a un número entero y se comprueba su formato, devolviendo un valor de error numérico 2 en el caso de que el formato no sea el correcto. En caso de haber solamente 2 argumentos, se asumirá como tercer argumento el valor numérico 10.

Acto seguido, se comprueba que las extensiones sean las correctas. En caso de serlo, se procederá a la ejecución de estas. En caso contrario, devuelve un error de tipo numérico con valor 3, indicando al usuario que la extensión introducida no es correcta.

Si la ejecución de *Test* ha ido bien, se devolverá como resultado la salida de la función correspondiente a la extensión, con el número de líneas indicado.

## Principales Funciones

**<< DESCRIPCIÓN DE LAS PRINCIPALES FUNCIONES IMPLEMENTADAS INCLUYENDO LA FUNCIÓN PRINCIPAL, MAIN>>**

**<< UTILIZAR EL FORMATO DE LA SIGUIENTE TABLA PARA CADA FUNCIÓN >>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Head** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 | **N** | **int** | **Número de líneas a imprimir** |
| **Variables Locales** | Variable 1 | **count** | **int** | **Contador de línea impresas** |
|  | Variable 2 | **buffer** | **Array de tipo char** | **Almacenamiento de las líneas de caracteres** |
| **Valor Devuelto** |  |  | **int** | **0 en caso de ejecutarse sin ningún error. 1 en caso de ocurrir un error de lectura.** |
| **Descripción de la Función** |  |  |  | **Imprime en la salida estándar las N primeras líneas procedentes de la entrada estándar.** |

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Tail** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 | **N** | **int** | **Número de líneas a imprimir** |
| **Variables Locales** | Variable 1 | **lines\_buffer** | **Puntero a punteros de tipo char** | **Almacenamiento de los punteros a las líneas seleccionadas.** |
|  | Variable 2 | **lines\_count** | **int** | **Contador de líneas leídas por entrada estándar** |
|  | Variable 3 | **start** | **int** | **Posición del buffer en la que se encuentra la primera línea a imprimir de las N líneas requeridas** |
|  | Variable 4 | **total\_lines** | **int** | **Índice tope de las líneas a leer** |
| **Valor Devuelto** |  |  | **int** | **0 en caso de ejecutarse sin ningún error. 1 en caso de ocurrir un error de asignación de memoria en el buffer de punteros a línea. 2 en caso de ocurrir un error de asignación de memoria en una de las líneas** |
| **Descripción de la Función** |  |  |  | **Imprime en la salida estándar las N últimas líneas procedentes de la entrada estándar.** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Longlines** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 | **N** | **int** | **Número de líneas a imprimir** |
| **Variables Locales** | Variable 1 | **list** | **Puntero a Line** | **Almacenamiento de las líneas leídas (Mediante las funciones complementarias se comporta como un TAD lista ordenada).** |
|  | Variable 2 | **lsize** | **int** | **Tamaño de la lista ordenada** |
|  | Variable 3 | **linebuffer** | **Array de tipo char** | **Almacenamiento temporal de las líneas de caracteres** |
|  | Variable 4 | **line** | **Line** | **Tipo Line devuelto por la función createLine** |
| **Valor Devuelto** |  |  | **int** | **0 en caso de ejecutarse sin ningún error. 1 en caso de ocurrir un error de asignación de memoria en el buffer de líneas** |
| **Descripción de la Función** |  |  |  | **Encuentra las N líneas más largas de la entrada estándar, y las imprime en salida estándar en orden descencente** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Main** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 | **argc** | **int** | **Contador de argumentos del programa** |
|  | Argumento 2 | **argv** | **Puntero a array de punteros de tipo char** | **Argumentos del programa** |
| **Variables Locales** | Variable 1 | **i** | **int** | **Asignador de número de líneas** |
| **Valor Devuelto** |  |  | **int** | **1 en el caso de haber más de tres argumentos. 2 en el caso de que el formato del segundo argumento no sea correcto. 3 en el caso de que la extensión introducida no sea correcta. Otros casos: se devuelve el valor correspondiente a la función ejecutada** |
| **Descripción de la Función** |  |  |  | **Función principal del programa de pruebas de la librería** |

## Casos de Prueba

**<< LISTA Y BREVE EXPLICACIÓN DE CASOS DE PRUEBA UTILIZADOS PARA VALIDAR LA PRÁCTICA >>**

# Comentarios Personales

## PROBLEMAS ENCONTRADOS

El problema principal fue simular el funcionamiento de una lista ordenada en un lenguaje de programación que no cuenta con orientación a objetos. Prácticamente cada vez que se hacía uso de memoria, daba algún tipo de error, como el indicado arriba referente a acceso a posiciones de memoria no permitidas.

Además, cuando se estaba empezando a realizar esa simulación del TAD, la función no funcionaba, pero tampoco lanzaba un error, lo que hizo que fuese bastante costoso encontrar el error (estaba en una asignación de una variable en una de las funciones secundarias).

## CRÍTICAS CONSTRUCTIVAS

No tenemos ninguna crítica. Se nos ha dado bastante tiempo, y el temario necesario para poder hacer la mayoría de la práctica sin problemas. La única función un poco más complicada era la última, pero era de una dificultad asumible respecto a la asignatura.

## PROPUESTA DE MEJORAS

Nada.

## EVALUACIÓN DEL TIEMPO DEDICADO

En total, tras haber hablado los dos de esta ponderación del tiempo medio, hemos llegado a la conclusión de que, por cada uno de nosotros, se ha requerido un tiempo medio de unas 50 horas. En estas horas se incluye mayoritariamente el tiempo de estudio y repaso requerido para entender el temario, ya que la práctica en sí no ha llevado tanto tiempo.