

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) |
| Repositorio utilizado para la práctica | | | [CLibrary](https://github.com/LuciaDominguezRodrigo/CLibrary) | |

# Descripción del Código

## Diseño del Código

De manera general, se ha implementado el código utilizando si es posible las funciones y librerías predeterminadas 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 global 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, que en este caso se corresponderá con 1024 bytes).

***Función Head***

Para imprimir por salida estándar las primeras N líneas, primeramente se inicializa un contador a 0, y mientras ese contador no supere N, se va llenando y sobrescribiendo un buffer con el tamaño que indique MAX\_LINE\_LENGTH, el cual se va leyendo y mostrando en la salida estándar, incrementando el contador y sobrescribiendo su contenido 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).

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media

***Función Tail***

Para la implementación de esta función, primeramente se inicializa un buffer dinámico que contiene datos de tipo char\* (que apuntarán 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 asigna memoria dinámica para cada línea apuntada por cada posición del buffer, es decir, las cadenas de caracteres donde irán alojadas las frases elegidas, 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, empezando a sobrescribir las primeras líneas almacenadas en el buffer de forma ascendente cada vez que este se encuentra lleno. De esta forma, sólo se usa el tamaño de memoria requerido para mostrar las N líneas necesarias, independientemente del tamaño de la entrada.

**Diagrama

Descripción generada automáticamente**

current\_line apuntará en cada lectura a la siguiente posición de lines\_buffer, comenzando por la 0 y escribiendo en la posición line\_count mod N, siendo line\_count el número de la línea leída por la entrada estándar en esa iteración (por ejemplo, si N = 7 y se lee la 9ª línea de la entrada estándar, entonces line\_count valdrá 9 y se sobrescribirá en la cadena de caracteres apuntada por el puntero situado en la posición 9 mod 7 = 2 de lines\_buffer).

Tabla

Descripción generada automáticamente

Siguiendo con esta idea, podría darse que la última línea leída por la entrada estándar no esté apuntada por el puntero en la última posición de lines\_buffer. Para localizar la primera de las últimas N líneas se usa la variable start, calculada como line\_count mod N en el caso de que la entrada supere las N líneas y 0 en caso contrario, y teniendo line\_count recogido el número total de líneas leídas por la entrada estándar (en el caso anterior, la primera línea a imprimir en la salida estándar sería la apuntada por el puntero situado en la posición 9 mod 7 = 2 en el caso de que line\_count fuera igual a 9, y es el valor que tendría recogido la variable entera start).

A la hora de imprimir las líneas por salida estándar, se imprimen por salida N líneas empezando por la cadena de caracteres apuntada por el puntero situado en la posición (start + i) mod N, para no acceder a espacios de memoria no permitidos.

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

***Función Longlines***

En primer lugar, 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 de mayor a menor longitud.

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 utilizado 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 ruta relativa 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 valr numérico 10.

Acto seguido, se comprueba que las extensiones sean las correctas. En caso de serlo, se procederá a la ejecución de la función correspondiente. 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 del programa *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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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** | **Número total de 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 para almacenar 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** | **Dato de 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** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **createLine** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 | **lineContent** | **Puntero constante a char** | **Contenido de la línea a crear** |
| **Variables Locales** | Variable 1 | **line** | **Line** | **Instancia de la nueva línea** |
| **Valor Devuelto** |  |  | **line** | **Devuelve un tipo de dato line completo, completándolo con el contenido que se le pasa como argumento a la función.** |
| **Descripción de la Función** |  |  |  | **Función para crear los tipos de datos líneas necesarios para la función longlines** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **insertLineList** | **Nombre** | **Tipo** | **Descripción** |
| **Argumentos** | Argumento 1 | **newLine** | **Line** | **Línea que se tiene que insertar** |
|  | Argumento 2 | **list** | **Puntero a Line** | **Lista en la que se tiene que insertar la línea** |
|  | Argumento 3 | **size** | **Puntero a int** | **Tamaño actual de la lista con la que se está trabajando** |
|  | Argumento 4 | **N** | **int** | **Tamaño máximo de la lista** |
| **Variables Locales** | Variable 1 | **pos** | **Puntero a int** | **Valor al que apunta size** |
| **Valor Devuelto** |  |  | **-** | **No devuelve valor** |
| **Descripción de la Función** |  |  |  | **Función que inserta las líneas en la lista ordenada** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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 que hemos encontrado ha sido 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. Finalmente, se ha podido implementar una lista ordenada especializada en cadenas de caracteres que inserta (si es posible) la cadena en la primera posición disponible teniendo en cuenta la longitud de las cadenas de caracteres ya presentes en la misma.

Otro problema relacionado con el proceso de elaboración del TAD ha sido la falta retroalimentación de errores de memoria por parte del compilador, haciendo que a veces las funciones no funcionaran correctamente, pero tampoco lanzasen ningún error. Esto hizo bastante costoso el proceso de refinamiento y comprobación de las funciones.

## 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

Por nuestra parte, no hemos encontrado ningún aspecto de la práctica que pueda o deba ser mejorado.

## EVALUACIÓN DEL TIEMPO DEDICADO

En total, teniendo en cuenta el tiempo empleado en la lectura y comprensión del tema y en la realización de los ejercicios correspondientes al tema 3 y las funciones pedidas en esta práctica llegamos a la conclusión de que, por cada uno de nosotros, se ha requerido un tiempo medio de unas 30 horas. Indudablemente, la función de la librería pedida en la que hemos empleado más tiempo ha sido la función longlines, por el hecho de tener que implementar una estructura de datos auxiliar no disponible en C para el proceso de ordenación de cada una de las líneas introducidas.