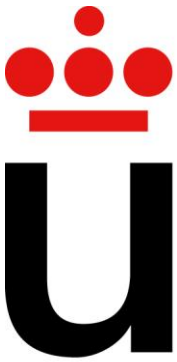

Universidad
Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA 1 – PROGRAMACIÓN C]

JON MAZCUÑÁN HERNÁNDEZ / ALEJANDRO RICO GONZÁLEZ

TABLA DE CONTENIDO

Autores

Jon Mazcuñán Hernández

- DNI:
- GitHub: jonmazh
- Encargado de la implementación las funciones “Head” y “Tail”.

Alejandro Rico González

- DNI: 70067434G
- GitHub: ALEJANDR0-RIC0
- Encargado de la implementación de la función “Longlines”.

Si se quiere ver todo lo respectivo al proyecto, como funciona y como se ha ido actualizando, se deja el link al GitHub del proyecto:

https://github.com/jonmazh/Practica1_LibreriaC

Descripción del Código

En esta práctica, el objetivo es desarrollar una biblioteca en C que facilite la gestión de líneas de texto desde la entrada estándar, proporcionando tres funciones distintas para mostrar cierto número de líneas bajo diferentes condiciones. Las funciones a implementar son:

1. **head(int N)**: Muestra las primeras N líneas recibidas.
2. **tail(int N)**: Muestra las últimas N líneas recibidas.
3. **longlines(int N)**: Muestra las N líneas más largas en orden descendente de longitud.

Además de la biblioteca, se desarrollará un programa de prueba, test.c, para verificar que cada función funcione como se espera. Este programa recibirá argumentos que le permitirán elegir la función y la cantidad de líneas a procesar, con un valor predeterminado de 10 líneas si no se especifica el número.

Por último, se creará un script de compilación, compila.sh, que automatizará la creación de la biblioteca y la vinculación del programa de prueba en una biblioteca estática llamada libreria.a.

Diseño del Código

<< EXPLICACIÓN DEL DISEÑO DEL CÓDIGO:

- **ALGORITMOS UTILIZADOS PARA CADA FUNCIONALIDAD PEDIDA**
- **ESTRUCTURA DEL PROGRAMA TEST**

>>

<< NO INCLUIR CÓDIGO FUENTE >>

<< SE PUEDE UTILIZAR PSEUDOCÓDIGO O DIAGRAMAS DE APOYO A LA EXPLICACIÓN >>

Principales Funciones

	MAIN	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**argc**	**int**	**Cuenta el número de argumentos de línea de comandos.**
	Argumento 2	**argv**	**char *[]**	**Array de cadenas que contiene los argumentos de línea de comandos.**
Variables Locales	Variable 1	**MSGERRM**	**char ***	**Mensaje de error que se muestra cuando la cantidad de argumentos no es la esperada o no son válidos. Informa al usuario sobre el uso correcto del programa.**

	Variable 2	**funct**	**char ***	**Apunta al primer argumento que representa el nombre de la función a ejecutar. Se asigna el valor de argv[1]**
	Variable 3	**nLinesM**	**int**	**Representa el número de líneas a procesar. Se asigna desde el segundo argumento si existe, de lo contrario, se le da un valor predeterminado de 10.**
Valor Devuelto			**int**	**Devuelve el resultado de la función callFunct. Si la cantidad de argumentos no es válida, devuelve -1.**
Descripción de la Función	**La función principal main analiza los argumentos de línea de comandos para determinar una función (funct) a ejecutar y un número (nLinesM) de líneas a procesar. Si los argumentos no son válidos, imprime un mensaje de error. Si recibe dos argumentos (nombre de la función y número de líneas), llama a callFunct con estos valores. Si recibe solo un argumento (nombre de la función), asigna un valor predeterminado de 10 a nLinesM y llama a callFunct.**			

	CALLFUNCT	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**funct**	**Char***	**Es un puntero a una cadena de caracteres que representa el comando introducido por el usuario. Se utiliza para determinar cuál función se debe llamar. Los valores esperados son "-head", "-tail" y "-longlines".**
	Argumento 2	**nLines**	**int**	**Especifica el número de líneas que se utilizarán como argumento al llamar a cualquiera de las funciones (head, tail, o longlines). Este valor se pasa directamente a la función seleccionada.**
	Argumento 3	**MSGERR**	**Const char***	**Es un puntero a una cadena de caracteres constante que contiene un mensaje de error. Se muestra cuando el comando funct no coincide con ninguno de los valores esperados, indicando un uso incorrecto de la función.**
Valor Devuelto			**int**	**La función retorna 0 si el comando (funct) coincide con una función y esta se ejecuta correctamente. Retorna -1 si el**

				comando no coincide con ninguna función conocida, indicando un error.
Descripción de la Función	**La función callFunct selecciona y llama a una función específica (head, tail o longlines) en función de un comando de texto (funct) introducido por el usuario. Esta función permite gestionar la ejecución de otras funciones según el comando recibido. Si el comando no coincide con ninguna de las funciones reconocidas, se muestra un mensaje de error y se retorna -1.**			

	HEAD	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**nLines**	**int**	**Indica el número máximo de líneas que se desea leer y mostrar desde la entrada estándar. Este valor controla la cantidad de iteraciones en los bucles que acumulan y procesan las líneas leídas.**
Variables Locales	Variable 1	**MSGERR**	**char***	**Contiene un mensaje de error que se muestra si ocurre un fallo durante la asignación de memoria. En este caso, el mensaje sugiere el uso correcto del programa.**
	Variable 2	**size**	**int**	**Almacena un valor que determina la cantidad inicial de líneas que se intentará leer y almacenar en memoria.**
	Variable 3	**counter**	**int**	**Lleva un conteo del número de líneas que se han leído de stdin. Se usa para limitar la lectura de líneas de acuerdo con el valor de nLines.**
	Variable 4	**i**	**int**	**Variable de control para iterar y asignar memoria en cada línea.**
	Variable 5	**j**	**Int**	**Variable de control de bucles, se usa en un bucle anidado para liberar memoria en caso de error.**
	Variable 6	**lines**	**Char****	**Apunta a un array de punteros que almacenan las líneas leídas de stdin. Cada posición de lines es un puntero a una cadena de caracteres (char *) que almacena una línea completa. Se utiliza memoria dinámica para manejar este array de cadenas.**

Valor Devuelto			**int**	**Retorna 0 si la función se ejecuta correctamente y ha leído el número de líneas especificado. En caso de error (como un fallo en la asignación de memoria o problemas en la lectura de la entrada estándar), retorna -1.**
Descripción de la Función	**Asigna memoria dinámica para el almacenamiento hasta N líneas de la entrada estándar. Emplea fgets para almacenar líneas y mostrarlas. Controla los errores de asignación de memoria, para de esta manera poder liberar la memoria en caso de error. Devuelve 0 o -1 en caso de error.**			

	TAIL	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**nLines**	**int**	**Especifica la cantidad de últimas líneas que se desea leer de la entrada estándar. Este valor determina el tamaño del array circular que se usa para almacenar las líneas.**
Variables Locales	Variable 1	**lines**	**Char****	**Es un puntero a un array de punteros, donde cada puntero apunta a una cadena de caracteres (char*) que representa una línea leída de stdin. Almacena las últimas nLines líneas.**
	Variable 2	**i**	**int**	**Variable de control de bucles. Se usa en el bucle principal para asignar memoria.**
	Variable 3	**j**	**int**	**Variable de control de bucles. Se usa en el bucle principal para liberar memoria en caso de error durante la asignación de líneas.**
	Variable 4	**counter**	**int**	**Lleva un conteo del número de líneas leídas hasta el momento y se utiliza para gestionar el índice de almacenamiento en el array lines. Usa aritmética modular (%) para funcionar de forma circular.**
	Variable 5	**Stdin_size**	**int**	**Lleva la cuenta del total de líneas leídas de stdin. Esto permite comprobar si el número de líneas leídas es menor que nLines, en cuyo caso se devuelve un error.**

	Variable 6	**auxCounter**	**int**	**Se usa para controlar la posición desde la cual se imprimen las líneas en el bucle final, ajustando el índice de inicio para mantener el orden correcto de las últimas nLines leídas.**
	Variable 7	**MSGERR**	**char***	**Contiene un mensaje de error que se muestra si ocurre un fallo durante la asignación de memoria o si nLines es menor que 1.**
Valor Devuelto			**int**	**Retorna 0 si la función ejecuta correctamente y lee las últimas nLines líneas desde stdin. Retorna -1 si hay un error en la asignación de memoria, si el número de líneas leídas es menor que nLines o si nLines es menor que 1.**
Descripción de la Función	**La función tail lee las últimas nLines líneas de la entrada estándar (stdin). Similar al comando tail en Unix, esta función usa un método de "ventana circular" para almacenar solo las últimas líneas leídas, evitando así el uso de grandes cantidades de memoria para entradas largas. La función maneja errores en la asignación de memoria y la lectura de la entrada, devolviendo -1 si ocurre algún fallo.**			

	ORDER_LINES	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**lines**	**char****	**Es un puntero a un array de cadenas de caracteres (puntero doble). Cada elemento de este array representa una línea de texto que se va a ordenar.**
	Argumento 2	**lenghts**	**int***	**Es un puntero a un array de enteros. Cada posición en este array corresponde a la longitud de la cadena en la posición equivalente del array "lines".**
	Argumento 3	**count**	**int**	**Entero que indica la cantidad de elementos en los arrays "lines" y "lengths". Controla el número de iteraciones en el bucle de ordenación.**
Variables Locales	Variable 1	**i**	**int**	**Variable de control para el bucle externo del algoritmo de ordenación bubble sort, que recorre cada elemento del array.**

	Variable 2	**j**	**int**	**Variable de control para el bucle interno del bubble sort, que se utiliza para comparar y ordenar los elementos en el array.**
	Variable 3	**aux**	**int**	**Variable temporal utilizada para intercambiar valores entre elementos del array lengths durante la ordenación.**
	Variable 4	**aux2**	**char[1024]**	**Array de caracteres temporal que almacena una línea de texto de "lines" durante el intercambio de posiciones. Tiene un tamaño fijo de 1024 caracteres.**
Valor Devuelto			**void**	**La función no devuelve ningún valor. Realiza la ordenación en el mismo array lines que se pasa por referencia, modificando los elementos del array directamente.**
Descripción de la Función	**La función organiza un array de cadenas (lines) en orden descendente de longitud, utilizando el algoritmo de ordenación bubble sort. Para ello, compara los elementos del array lengths, que contiene la longitud de cada línea, y realiza intercambios en ambos arrays (lines y lengths) cuando detecta un valor mayor en el array lengths. Este proceso permite que las cadenas de mayor longitud queden en las primeras posiciones del array lines.**			

	LONGLINES	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	**nLines**	**int**	**Indica la cantidad de líneas que se deben imprimir después de ordenar.**
Variables Locales	Variable 1	**capacity**	**int**	**Tamaño inicial de los arrays "lines" y "lengths", definido en 100. Posteriormente se usa para ampliar la memoria.**
	Variable 2	**lines**	**char****	**Array de cadenas que almacena las líneas de texto leídas. Cada posición contiene una línea de la entrada estándar.**

	Variable 3	**lenghts**	**int***	**Array de enteros en el que cada elemento guarda la longitud de la línea correspondiente en lines.**
	Variable 4	**count**	**int**	**Contador que lleva la cuenta de la cantidad de líneas leídas. Aumenta cada vez que se almacena una nueva línea en lines.**
	Variable 5	**max_lenght**	**int**	**Tamaño máximo permitido para cada línea de texto, definido en 1024 caracteres.**
	Variable 6	**i**	**int**	**Variable de control para bucles, usada para gestionar la asignación y liberación de memoria, y también para la impresión de las líneas ordenadas.**
	Variable 7	**j**	**int**	**Variable de control auxiliar utilizada en el bucle interno durante la liberación de memoria en caso de error.**
	Variable 8	**buffer**	**char[max_lenght]**	**Array temporal que almacena cada línea de texto leída de la entrada estándar antes de copiarla a "lines".**
Valor Devuelto			**int**	**La función devuelve 0 si se ejecuta correctamente sino devuelve -1.**
Descripción de la Función	**La función lee líneas de texto desde la entrada estándar, las almacena en el array "lines", y sus longitudes en el array "lengths". Si la capacidad actual de los arrays es insuficiente, la función redimensiona la memoria de ambos para manejar más líneas. Luego, ordena las líneas en orden descendente por longitud, utilizando la función order_lines. Finalmente, imprime un número específico de líneas (definido por el argumento nLines) en orden y libera la memoria utilizada.**			

Casos de Prueba

Para validar la correcta implementación de las funciones tail, head y longlines, se han diseñado una serie de casos de prueba que permiten comprobar su comportamiento bajo distintas condiciones. A continuación, se describe brevemente cada uno de los casos de prueba utilizados:

1. **Prueba llamando a la función sin el guión delante del nombre:**

   Se ejecuta el programa utilizando ./test nombre_de_funcion (sin el prefijo - en el nombre de la función, como head en lugar de -head). Esta prueba verifica si el programa identifica y gestiona adecuadamente un comando de función incorrecto.

2. **Prueba sin especificar un número de líneas:**

   La función se llama correctamente (-head, -tail o -longlines), pero sin indicar la cantidad de líneas a mostrar. Esta prueba permite comprobar si el programa detecta la falta de un parámetro obligatorio y responde con un mensaje de error o comportamiento adecuado.

3. **Prueba con valores no válidos para el número de líneas (0 y negativos):**

   Se ejecuta cada función con un número de líneas igual a 0 o un valor negativo. Este caso permite validar si el programa maneja correctamente valores de entrada que no tienen sentido en el contexto, retornando un error y mostrando un mensaje explicativo cuando se reciben valores no válidos.

4. **Prueba con un número positivo válido:**

   Cada función es ejecutada con un número positivo, indicando la cantidad de líneas a procesar. Este caso asegura que el programa ejecuta la función de forma correcta en situaciones normales, mostrando el número de líneas indicado según el comportamiento esperado de head, tail o longlines.

5. **Prueba de superación de espacio en longlines:**

   Para la función longlines, se realiza una prueba que fuerza al programa a superar el espacio asignado inicialmente. Este caso permite verificar si longlines es capaz de gestionar dinámicamente el crecimiento de la memoria cuando la cantidad de líneas excede el espacio inicialmente reservado, evitando errores de segmentación o pérdida de datos.

Comentarios Personales

PROBLEMAS ENCONTRADOS

Hemos encontrado problemas durante todo el desarrollo de la práctica. Al principio de la práctica, encontramos más bien problemas conceptuales, como por ejemplo errores por no liberar una estructura de datos. Durante el desarrollo de la práctica encontramos errores lógicos como por ejemplo que el funcionamiento de head no era el esperado si lo basábamos en el funcionamiento del head de terminal, ya que head cuando lee la entrada estándar muestra cada línea a la vez que se escribe. Esto lo solucionamos introduciendo la escritura de datos en el bucle de lectura. En cuanto a tail, también tenía errores lógicos, como que mostraba las líneas desordenadas. Esto lo solucionamos empleando un nuevo índice cíclico, que nos permitía saber en que punto había acabado el último contador y así poder imprimir las últimas líneas en orden.

Fallos en la función head

Durante las pruebas realizadas, detectamos que la función que habíamos desarrollado no replicaba correctamente el comportamiento del comando head de Ubuntu. En este comando, las palabras o frases introducidas por la entrada estándar se imprimen inmediatamente a medida que se reciben. Sin embargo, nuestra implementación almacenaba todas las entradas y las imprimía de forma conjunta al final.

Este fallo se corrigió de manera sencilla. Para lograrlo, colocamos la función put dentro del primer condicional del bucle while, asegurándonos de que se ejecutara antes de avanzar en el contador. Este ajuste permitió que las entradas se imprimieran de manera inmediata, logrando así un funcionamiento consistente con el comando original.

Fallos en la función tail

En esta función, identificamos dos principales problemas que afectaban su funcionamiento:

1. **Desorden en la impresión de las líneas**:

   La función debía imprimir las últimas “N” líneas introducidas, pero el programa las mostraba de manera desordenada. Esto ocurría porque las líneas se almacenaban en un array de “N” posiciones y, al llegar al final, las primeras posiciones eran sobrescritas, generando un desorden en el resultado.
   Para resolver este inconveniente, implementamos una variable auxiliar denominada auxCounter, que permite controlar la posición exacta donde se quedó el array. De este modo, se garantiza que las líneas se impriman en el orden correcto.

2. **Comportamiento inesperado con entradas no válidas**:

   Durante las pruebas, observamos fallos al introducir valores de entrada como 0 o números negativos. En estos casos, la función generaba resultados impredecibles, mostrando líneas aleatorias. Tras analizar el comportamiento de la función tail de Ubuntu, notamos que, para una entrada 0, no se muestra ninguna línea, y que no se contemplan entradas negativas.
   Por ello, añadimos una validación al inicio de la función que verifica si el valor de entrada es menor que 1. En tales casos, la función simplemente no imprime nada, alineándose con el comportamiento esperado.

Fallos en la función longlines

1. **Problemas con el uso de memoria estática**:
   La función se diseñó inicialmente utilizando memoria estática, lo que generó problemas al procesar entradas cuyo tamaño excedía la capacidad de memoria predefinida. Para solucionar este inconveniente, se optó por implementar memoria dinámica, lo que permitió manejar entradas de tamaño variable y mejorar la escalabilidad de la función.

2. **Errores en la comprobación de longitudes de las líneas**:
   Aunque el algoritmo de ordenación implementado, "bubble sort", funcionaba correctamente, la función solo verificaba las primeras "N" líneas de la entrada estándar. Esto provocaba que, si una línea más larga aparecía después de las primeras "N" posiciones, no se tuviera en cuenta.
   Para corregir este error, primero se ordenaron todas las líneas almacenadas en el array "lines". Posteriormente, se imprimieron únicamente las "N" primeras líneas del array, garantizando que se seleccionaran correctamente las más largas independientemente de su posición en la entrada.

CRÍTICAS CONSTRUCTIVAS

La práctica nos ha parecido muy bien estructurada para afianzar los conceptos básicos del lenguaje de programación C. Nos ha resultado especialmente interesante tener que decidir entre el uso de memoria estática o dinámica, ya que esto nos permitió comprender de manera práctica las ventajas, desventajas y aplicaciones de cada enfoque.

En cuanto a críticas, no encontramos aspectos negativos destacables. La práctica estaba bien diseñada y era adecuada para el nivel de conocimientos adquirido, ya que todos los contenidos necesarios habían sido previamente explicados.

PROPUESTA DE MEJORAS

Aunque no tenemos quejas al respecto, consideramos que las funciones head y tail presentan una estructura y funcionalidad muy similares, con ciertas diferencias en su implementación. Para futuros años en los que se realice esta misma práctica, podría ser interesante sustituir alguna de estas funciones por una alternativa diferente, que aporte mayor variedad en los desafíos planteados.

EVALUACIÓN DEL TIEMPO DEDICADO

En total, hemos dedicado entre 6 y 8 horas cada uno para completar esta práctica. A continuación, desglosamos el tiempo invertido por cada etapa:

- **Programación:** Ha sido la fase más extensa, con una dedicación aproximada de 4 horas por persona.
- **Pruebas y corrección de errores:** Esta etapa nos llevó entre 1 y 2 horas por persona.
- **Documentación:** Estimamos un tiempo de entre 1 y 2 horas por persona para esta parte.

Además, si consideramos la preparación previa, como ejercicios, ejemplos y otros materiales, podríamos añadir aproximadamente 4 horas adicionales por persona. Sin embargo, no hemos contabilizado este tiempo en la estimación principal, ya que no está directamente relacionado con la práctica en sí.