SISTEMAS OPERATIVOS

Ingeniería del Software

PROGRAMACIÓN EN C

PRÁCTICA 1

Annhelen Cervera Saldaña
Sergio Blay Gonzales

Contenido

DESCRIPCIÓN DEL CÓDIGO

El problema que se nos ha planteado, consiste en desarrollar una librería con tres funcionalidades distintas: *head*, *tail* y *longlines*; recibiendo, cada una de ellas, un valor entero que será el número de líneas a leer por la entrada estándar.

1. Programa Test: *"test.c"*

Este es nuestro programa principal, el que se nos permitirá probar las distintas funcionalidades de la libraría. Este será el encargado de recibir los argumentos, así como del control de los mismos en caso erróneo, se encargará de llamar a las funciones de la librería según la invocación introducida, y además controlará los errores que se puedan producir durante la ejecución del programa.

1.1 Invocación

Para ejecutar cada una de las funciones, tendremos que hacerlo de la siguiente forma:

$ ./test *NOMBRE_FUNCION [-N] [NOMBRE_FICHERO]*

Donde:

- *NOMBRE_FUNCION.* Nombre de la función a invocar: *head*, *tail*, *longlines*.
- *-N.* Parámetro opcional para indicar el número de líneas a leer de la entrada estándar. Si no se especifica, tomará como valor por defecto 10.
- *NOMBRE_FICHERO.* A modo de funcionalidad extra, con este parámetro opcional permitimos redirigir la entrada estándar a un fichero tal y como se puede hacer en la terminal, es decir, sin utilizar el símbolo de redirección *(<)*; aunque también este puede ser utilizado. Si no se especifica, leerá las líneas introducidas por la entrada estándar por defecto.

1.2 Funcionalidad

El test se encargará de invocar el método correctamente de acuerdo a los parámetros introducidos según como se muestra el apartado anterior. El test utiliza una función auxiliar para evitar duplicar código en los distintos argumentos. Además, define el número de líneas a mostrar por defecto en caso que no se indique por argumento.

El test también es el encargado del control de errores en los argumentos, controlando tanto si no está bien el nombre del mandato, como si no encuentra el fichero.

2. Librería: *"libreria.c"*

Este archivo contiene las tres funcionalidades que serán utilizadas desde el programa principal "Test".

2.1 Función *head(int N)*

Esta función permite mostrar las N primeras líneas, por la salida estándar, recibidas por la entrada estándar.

EJECUCIÓN.

- $ ./test head
- $ ./test head -4
- $ ./test head -4 < fichero.txt
- $ ./test head -4 fichero.txt

FUNCIONAMIENTO INTERNO.

Recibe como argumento, un valor entero que será el número de líneas a mostrar (N). Utilizaremos un contador que indicará cuántas líneas se están leyendo por STDIN; mientras que dicho contador sea menor que el valor del argumento N y no se haya producido un EOF (end-of-file), la función seguirá leyendo líneas por STDIN y mostrándola por STDOUT.

2.2 Función *tail(int N)*

Permite mostrar por la salida estándar, las N últimas líneas recibidas por la entrada estándar.

EJECUCIÓN.

- $ ./test tail
- $ ./test tail -3
- $ ./test tail -3 < fichero.txt
- $ ./test tail -3 fichero.txt

FUNCIONAMIENTO INTERNO.

Recibe como argumento, un valor entero N que será el número de líneas a mostrar. Para realizar la implementación de esta función, ha sido necesario utilizar una estructura dinámica en la que almacenaremos las N últimas líneas introducidas . Para ello, primero se reserva en memoria una matriz de N elementos de tipo char * . Antes de proseguir con el algoritmo, se comprueba si el sistema ha podido reservar memoria para nuestras N últimas líneas, en caso contrario, la función devolverá un código de error (en este caso un 1) finalizando la ejecución del mismo.

Una vez reservado el espacio necesario, se irán leyendo líneas de STDIN mientras no se produzca un EOF. Al igual que en la función anterior, utilizaremos un contador que nos indique cuántas líneas se van leyendo , y si el contador es menor que el total de líneas a mostrar, se reservará tanta memoria como ocupe la nueva línea, y se copiará del buffer en esa posición que hemos reservado, hasta que el contador sea igual a N y que la matriz esté llena.

La forma en la que se reserva memoria para cada línea a leer, ha sido implementada de tal manera que hacemos al programa más eficiente, ya que reservamos exactamente el número de líneas que van siendo introducidas en lugar de reservar desde un inicio todas ellas, evitando así que, si se produce un EOF antes de completar la matriz, no desperdiciemos espacio de memoria que ha sido reservada y que no ha sido utilizada.

Al introducir una nueva línea, una vez llena la matriz, se irán desplazando las líneas almacenadas dentro de esta, para almacenar la última introducida. Para ello, lo que se hará es liberar la posición de memoria donde se encuentre la línea más antigua, y se cambiará a donde esté apuntando actualmente, a una posición más adelante; desplazando de este modo todos los punteros a la posición siguiente. Esto se realiza hasta la posición N-2, dejando la posición N-1 disponible para la nueva línea, tras lo cual se reservará memoria para guardar esta.

Una vez producido el final de línea (EOF), se muestra por la salida estándar únicamente las líneas almacenadas en la matriz dinámica. Para saber cuántas han sido, nos apoyaremos en el contador que iniciamos al empezar la ejecución de la función.

Finalmente, liberamos el espacio que reservamos previamente para la matriz y dejamos dicho puntero apuntando a NULL.

2.3 Función *longlines(int N)*

Esta función mostrará por la salida estándar, las N líneas más largas de mayor a menor introducidas por la entrada estándar.

EJECUCIÓN.

- $ ./test longlines
- $ ./test longlines -3
- $ ./test longlines -3 < fichero.txt
- $ ./test longlines -3 fichero.txt

FUNCIONAMIENTO.

Este método reservará desde un inicio tanto la matriz de punteros, como el espacio al que apuntan los mismos, reservando toda la memoria desde un inicio.

Mientras que no se produzca un EOF, se seguirán leyendo líneas por la entrada estándar y se irá comparando el tamaño de la línea con el de las que hemos almacenado, hasta encontrar una que sea menor (si no hay ninguna sencillamente se desecha sin almacenarla).

Si encontrase una línea de menor tamaño, se desplazará esa y las siguientes una posición, de modo que se deje hueco para la nueva línea desechando la más pequeña que tengamos almacenada.

Tras desplazar las líneas, se introducirá la nueva línea de manera que sigan quedando ordenadas.

Una vez se llegue a EOF, se mostrarán por la salida estándar las líneas almacenadas, tras lo cual se liberará su espacio de memoria, y el de la matriz en sí.

COMENTARIOS PERSONALES

Esta práctica nos ha ayudado a entender mejor el funcionamiento de la memoria dinámica, así como de la importancia del control de la misma, y del control de los argumentos.

El hecho de tener conocimientos previos en otros lenguajes de programación, nos ha facilitado el aprendizaje de la sintaxis de C, pero no por ello, nos ha resultado sencillo el desenvolvernos en la esencia principal que tiene este lenguaje: el manejo de punteros.

A decir verdad, este ha sido el principal "obstáculo" ya que, acostumbrados a utilizar otro tipo de lenguajes que facilitan al programador en lo que se refiere a gestionar directamente la memoria, como Javascript, Ruby, PHP o Java (que tiene el Garbage collector), nos ha dado otra visión en la programación. El hecho de realizar esta práctica utilizando memoria dinámica, nos obliga a "jugar" directamente con la memoria, lo cual esto te implica a intentar hacer un programa más eficiente, que consuma menos recursos y además muy cuidadoso en lo que a control de errores se refiere.

Otro "obstáculo" que podríamos mencionar es el desarrollo en equipo, ya que cada uno tiene su forma de programar, una visión distinta para solucionar el mismo problema. Esto también nos ha enseñado no sólo a ver que hay muchas formas de resolver un problema, sino, ha intentar consolidar una única solución óptima llegando a un acuerdo, aprendiendo a aceptar críticas constructivas. A esto, también se le suma el tener que sincronizar las distintas tareas que hemos ido desarrollando por separado pero como equipo, y para ello, nos hemos apoyado en un control de versiones (GitHub).

Repositorio: https://github.com/annyCS/libreria-basica-c