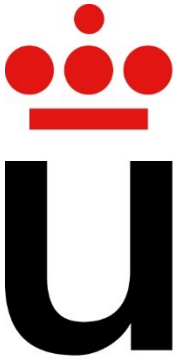

Universidad
Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA 1 – PROGRAMACIÓN C]

STEFANO TOMASINI HOEFNER

TABLA DE CONTENIDO

Descripción del código

Diseño del código

head:
Simplemente iteré sobre el string multilínea leído. Por cada carácter salto de línea encontrado, se le resta uno a N, y si N llega a cero, o no hay más caracteres a imprimir; se sale del bucle. Al final siempre se terminan imprimiendo N líneas, (o las que haya si son menos de N). Antes de finalizar, se libera el string multilínea leído ya que fue alocado en el heap.

tail:
Antes de empezar a iterar, se le suma 1 a N si el último carácter no-nulo de la cadena multilínea es un salto de línea. Después, se itera sobre cada carácter del string, solo que empezando desde el final del string y yendo hacia delante. Si el carácter actual es un salto de línea, se le resta 1 a N. Cuando N vale 0, hay que cambiar de sentido de iteración y empezar a imprimir caracteres. Antes de ejecutar este segundo for, se le suma 2 a "charI" para volver al principio de la línea que le seguía al último salto de línea encontrado. A partir de ahí, se escribe a *stdout* cada carácter entre la posición actual de "charI" y el final de la cadena. Finalmente, se libera el string multilínea leído.

longlines:
Se guarda cada línea del string multilínea introducido por el usuario dentro de un pseudo-array de punteros a char (*longestLines*, en realidad es solo un puntero a un puntero a char) cuyo tamaño asignado en el heap va creciendo dinámicamente para albergar el puntero a cada línea encontrada con *strtok*. Es decir, cada línea se vuelve un string distinto el cual se "guarda" (es apuntado a) en el "array" mencionado anteriormente. Los saltos de línea se convierten en '\0' al usar el *strtok*. Esto nos sirve para después realizar un ordenamiento de burbuja sobre este "array". En el cuerpo de los dos bucles for anidados, se comprueba si el string actual es menos largo que el string que le sigue, y si es así, se realiza un intercambio de direcciones entre las dos posiciones del "array" (la posición actual y la siguiente). Al terminar el ordenamiento de burbuja, se imprime cada línea del "array" ya ordenado de mayor a menor, y se liberan aquellos strings alocados dinámicamente que fueron utilizados.

main (test.c):
Primero se ejecuta un switch tomando como argumento (argc-1), para realizar el primer control de errores. En el caso 0, el usuario no le pasó ningún argumento al ejecutable, así que se imprime la cadena que especifica cómo se usa. En el caso 1, no se especificó #LINES, así que se toma 10 como valor predeterminado. En el caso 2, se introdujo un segundo argumento, así que se intenta parsear el string. En caso de que no se pueda parsear como int, o de que sea menor a 1, se imprime un mensaje que avisa que debe ser un int positivo el segundo parámetro. En el caso *default* (tres argumentos introducidos o más), se escribe por *stderr* que se introdujeron demasiados argumentos.

Después de pasar por el switch, hay que atravesar una cadena de *if-else*'s para comprobar si el primer argumento tiene como string contenido alguno de los parámetros asociados a una función válida. En caso de que no, se imprime un mensaje de error.

Principales funciones

	main (de test.c)	Nombre	Tipo	Descripción
Argumentos	Argumento 1	argc	Entero de 32 bits con signo	Número de argumentos pasados al ejecutable, contando el nombre del propio ejecutable.
	Argumento 2	argv	Array de punteros a const char	Es un array de punteros a const char (puntero a un string de c que es de solo lectura)
Variables locales	Variable 1	nLines	Entero de 32 bits con signo	Guarda el n.º de líneas a pasar a la función que especifique el usuario. Puede ser negativo o cero el número pasado, así que antes de llamar a la función se hace una comprobación de que sea positivo.
	Variable 2	USAGE_STRING	Puntero a const char	Almacena el string (readonly) a imprimir en caso de que se use incorrectamente la función.
Valor devuelto			Entero de 32 bits con signo	Resultado de la ejecución del programa: 0 = sin error. 1 = con error
Descripción de la función	Ejecuta la función (head, tail, o longlines) que especifique el usuario en el argumento i=1. El argumento i=2 es el número de líneas que recibiría la función específica que se llame como argumento. Si no se especifica el argumento i=2, el valor predeterminado es 10 lineas. Incluye control de errores.			

	head	Nombre	Tipo	Descripción
Argumentos	Argumento 1	N	Entero	N primeras líneas que se van a extraer de toda la entrada leída.
Variables locales	Variable 1	wholeReadString	Puntero a char	Puntero a la dirección de memoria devuelta por la llamada a *readMultipleLines()*.
	Variable 2	charI	Entero	Variable que se utiliza como “i” del bucle for
Valor devuelto			Entero	0 = sin error 1 = error
Descripción de la función	Pasa a *stdout* cada char del string multilínea introducido por el usuario, hasta llegar al salto de línea n.º N. Se imprime este último char salto de línea (que flushea el buffer) y se termina la ejecución.			

	tail	Nombre	Tipo	Descripción
Argumentos	Argumento 1	N	Entero	N últimas líneas que se van a extraer de toda la entrada leída.

Variables locales	Variable 1	wholeReadString	Puntero a char	Puntero a la dirección de memoria devuelta por la llamada a *readMultipleLines()*.
	Variable 2	charI	Entero	Variable que se utiliza como "i" de los dos bucles for.
	Constante 1	READ_STRING_LENGTH	Entero const sin signo	Longitud de wholeReadString, excluyendo el '\0'. Tuve que inicializarlo entremedio de la función porque quería hacerlo const y no se pueden declarar variables const sin inicializarlas. Se usa para no llamar strlen(wholeReadString) tres veces.
Valor devuelto			Entero	0 = sin error 1 = error
Descripción de la función	Extrae e imprime las N últimas líneas del string pasado por el usuario. Se hace con un for que recorre de atrás hacia adelante, y cuando se llega al N salto de línea, se avanza 2 (porque al salir del for se le resta 1 más a N, así que hay que sumarle 2 al "i" para volver al primer char de la primer línea a imprimir), y a partir de ahí se hace otro for, pero en sentido hacia el \0 del string, y se imprime cada char a *stdout*.			

	longlines	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	N	Entero	N líneas más largas que se van a mostrar de la ordenación resultante de todas las líneas pasadas.
Variables locales	Variable 1	wholeReadString	Puntero a char	Puntero a la dirección de memoria devuelta por la llamada a *readMultipleLines()*.
	Variable 2	temp	Puntero a char	Usado para albergar temporalmente una dirección con la que inicializar otro puntero.
	Variable 3	longestLines	Puntero a puntero a char	Puntero a una alocación de memoria en el heap que va creciendo dinámicamente para albergar cada puntero al primer char de cada línea pasada.
	Variables 4 y 5	i, j	Enteros sin signo	Ambos se usan como índices de los bucles for.
	Variable 6	linesCount	Entero sin signo	Número de líneas que contiene en total el string multilínea introducido por el usuario.
Valor devuelto			Entero	0 = sin error 1 = error
Descripción de la función	Lee todas las líneas que le pase el usuario, las ordena de mayor a menor según su longitud, y de la ordenación resultante muestra las N primeras líneas.			

Casos de prueba:

1) test -noexisto 3
Devuelto: "Passed function name not recognized. Valid functions: -head, -tail, -longlines"

2, 3, 4) test -head/-tail/-longlines 8 9
Devuelto: "Too many arguments provided. Use: -[FUNCTION] [#LINES]"

5, 6) test -head/-tail
[once o más líneas] [CTRL+D]
Devuelto: [diez primeras/últimas líneas]

7, 8, 9) test -head-/tail/-longlines 0
Devuelto: "[LINES] must be a positive integer"

10, 11) test -head/-tail 2
>lineauno [CTRL+D]
Devuelto: lineauno

12) test -head 2
>lineauno
>lineados
>lineatres
[CTRL+D]
Devuelto:
lineauno
lineados

13) test -tail 2
[misma entrada]
Devuelto:
lineados
lineatres

14) test -longlines 2
>AAAAA
>BBBBBBBBB
>CCCCCCC
[CTRL+D]
Devuelto:
BBBBBBBBB
CCCCCCC

Comentarios personales

Nota: soy consciente de que no existe “alocar” en español con el significado que tiene en inglés, pero necesito la palabra inventada para expresarme más precisamente y de forma más concisa.

Problemas encontrados

Tuve algunos problemas al usar funciones de *string.h* porque no entendía muy bien como funcionaban (de forma precisa). Lo más difícil fue el bubble sort de las líneas ya que al principio lo implementé mal y se me iba el “j+1” fuera del “array” y se producía un *segfault*. Por su parte los *malloc* y *realloc* no me parecen difíciles de usar o de comprender.

Críticas constructivas

Creo que está bien la práctica como primera práctica de C. Solo que no me gusta la restricción de que no se pueden declarar variables entremedio de funciones y no está explicado el porqué. Es tediosa la parte de escribir una memoria así de larga para unos pocos ejercicios, tal vez estaría mejor que haya menos apartados a rellenar.

Propuesta de mejoras

Menos apartados a rellenar para la memoria.

Evaluación del tiempo dedicado

Más o menos seis a ocho horas divididas entre varios días para hacer la práctica entera, pero ya no estoy seguro de cuánto tiempo fue.