Entrega 1:

Análisis léxico y sintáctico

PROCESADORES DE LENGuAJEs

Carlos Ruiz Ballesteros

Héctor ruiz-poveda coca

GIS-GII

Índice

[AUTORES 2](#_Toc446068892)

[DESCRIPCIÓN DEL CÓDIGO 2](#_Toc446068893)

[Analizador léxico 2](#_Toc446068894)

[Analizador sintáctico 2](#_Toc446068895)

[TextAreaOutputStream.java 2](#_Toc446068896)

[Función tail 3](#_Toc446068897)

[Función longlines 3](#_Toc446068898)

[Interfaz gráfica y su uso 3](#_Toc446068899)

[EJEMPLOS DE EJECUCIÓN 5](#_Toc446068900)

[Control de errores en la entrada de argumentos 5](#_Toc446068901)

[Ejecución de la función *head* 5](#_Toc446068902)

[Ejecución de la función *tail* 6](#_Toc446068903)

[Ejecución de la función *longlines* 7](#_Toc446068904)

[Ejecución de la función *longlines* con un fichero como *stdin* 8](#_Toc446068905)

[Comentarios personales 9](#_Toc446068906)

[Problemas encontrados y decisiones tomadas 9](#_Toc446068907)

# AUTORES

Carlos Ruiz Ballesteros.

Héctor Ruiz-Poveda Coca.

Ambos autores somos del Doble Grado en Ingeniería del Software + Ingeniería Informática.

# DESCRIPCIÓN DEL CÓDIGO

El código se divide en tres partes:

## Analizador léxico

Por medio de la herramienta *“Flex”* se ha desarrollado un analizador léxico que permite identificar los distintos elementos: valores numéricos, identificadores, constantes literales, comentarios, retornos de carro…

El analizador léxico transforma estos elementos, así como palabras reservadas, en tokens y símbolos que utilizará el analizador sintáctico.

## Analizador sintáctico

La gramática que se encarga de comprobar si el código es sintácticamente correcto ha sido generada por *“CUP”*.

Este analizador se encarga de comprobar que la estructura y el orden del programa son correctas; es decir, que no existan elementos donde no deban estar. Corrobora que la zona de declaración de variables sea correcta y esté situada al inicio del programa, seguida de un bloque *“BEGIN … END.”*. Asímismo se asegura de que dentro de este bloque haya una o más sentencias correctamente formadas.

Cabe destacar que se han añadido las modificaciones pertinentes para añadir la gramática que permite el reconocimiento de matrices y registros, así como sentencias*”IF”*, *“WHILE”* y *“FOR”*.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### TextAreaOutputStream.java

Comienza con una comprobación: Si el número de líneas a mostrar no es válido (mayor que cero), no realiza la función.

Tras inicializar las variables necesarias, lee líneas de *stdin* y las muestra por pantalla hasta que se alcance el número (N) de líneas pedido.

### Función tail

Al igual que head, comprueba que el número de líneas a mostrar es válido.

Tras la inicialización pertinente, se reserva un espacio de memoria correspondiente al número de líneas a mostrar. Cada línea admite un máximo de 512 caracteres (definido en MAX.)

Se llena la memoria con las N primeras líneas, y a partir de ahí se sustituye la primera línea que entró por la nueva leída, quedando así siempre las N últimas líneas en memoria.

A la hora de mostrar por pantalla, se muestra desde la primera línea introducida en la memoria hasta el final de esta, y acto seguido desde el inicio de la memoria hasta justo la anterior a la primera introducida. De esta manera se imprimen en *stdout* en el mismo orden en el que se introdujeron.

Finalmente se libera toda la memoria reservada.

### Función longlines

Realiza la misma comprobación inicial que las anteriores funciones.

Se inicializan las variables y se reserva un espacio al igual que en *tail*. Se llena la memoria con las N primeras líneas, y para cada nueva línea leida:

-Se calcula cual es la línea de menor longitud almacenada, utilizando para ello la función auxiliar *menor*.

-Si la nueva línea tiene mayor longitud, se guarda en lugar de la actual.

Una vez terminada la entrada, se ordena la memoria mediante el algoritmo de la burbuja, para situar en primer lugar las líneas de mayor longitud, y se muestran por pantalla.

Finalmente, al igual que en *tail*, se libera la memoria utilizada.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Interfaz gráfica y su uso

Contiene el método *main* del programa, que hace uso de la librería construida, utilizando una función u otra dependiendo de los argumentos recibidos.

Comienza con la inclusión de las librerías necesarias, tras lo cual se comprueban los argumentos recibidos:

-Si no se reciben al menos dos argumentos (la ruta del ejecutable y el nombre de una función), no se realiza ninguna función, se avisa y finaliza la ejecución del programa.

-El segundo argumento recibido se corresponde con el nombre de la función a ejecutar, con lo que en el posterior *switch* se decidirá qué función realizar.

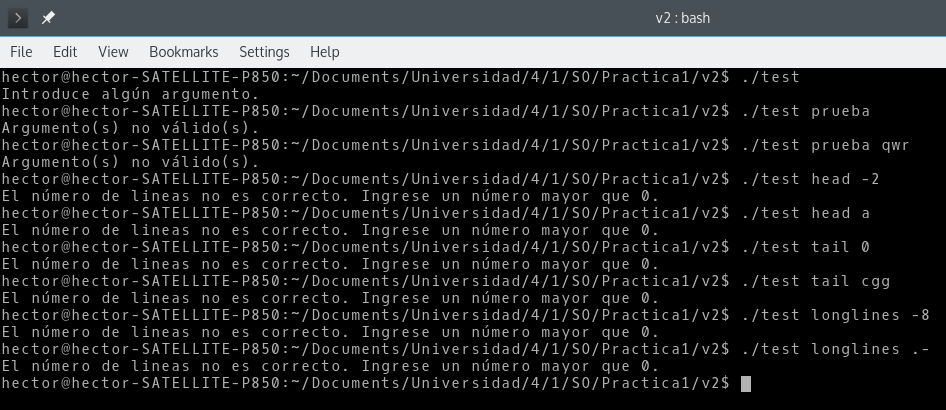
-En caso de recibir un tercer argumento, éste indicará el número de líneas que se aplica a la función. Se pasará a la función como argumento N.

-Si no se recibe el tercer argumento, se realiza la función con un argumento fijo DEF, cuyo valor es 10 (al igual que las funciones que se pueden llamar desde el terminal del sistema operativo).

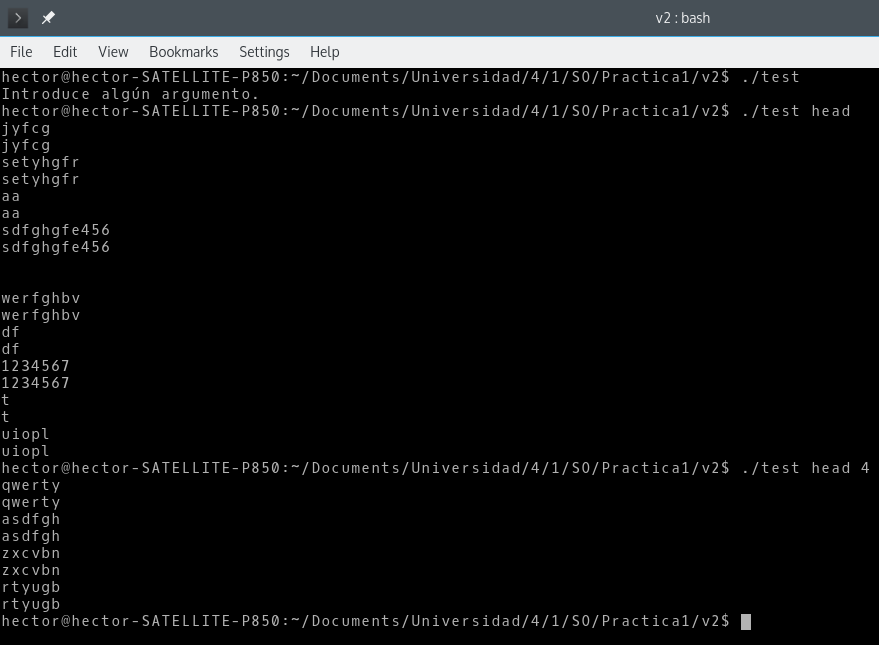
# EJEMPLOS DE EJECUCIÓN

A continuación, se muestran las capturas de pantallas correspondientes a la ejecución del código:

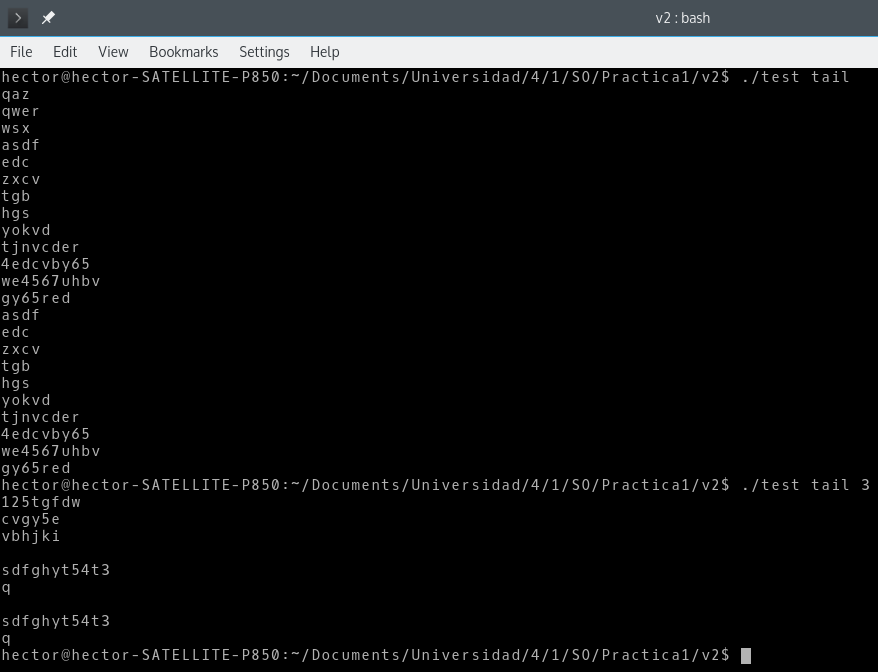
## Control de errores en la entrada de argumentos



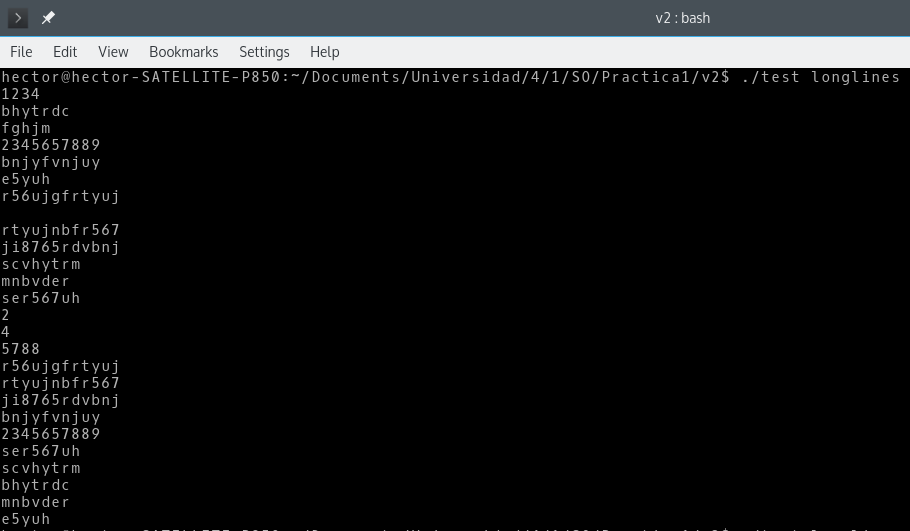
## Ejecución de la función *head*



## Ejecución de la función *tail*

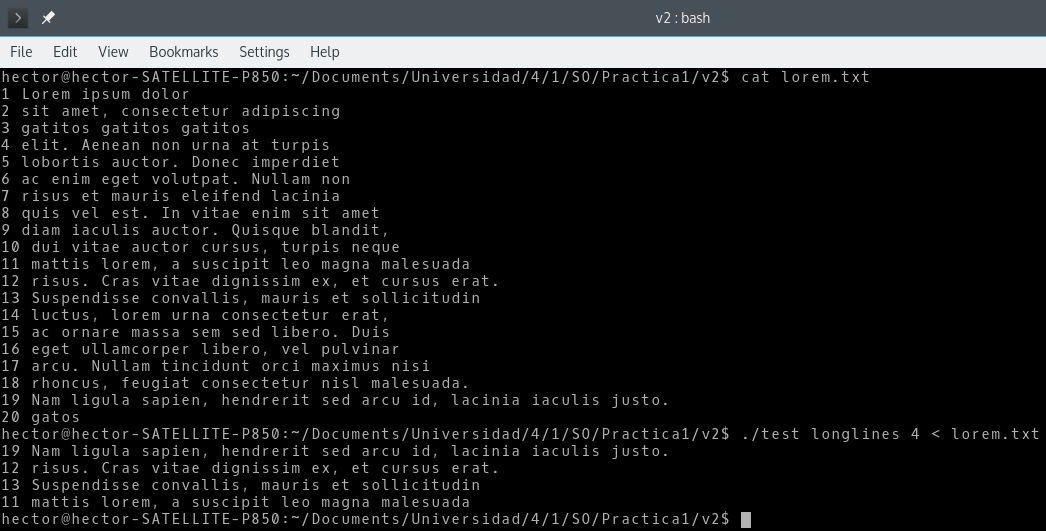


## Ejecución de la función *longlines*





## Ejecución de la función *longlines* con un fichero como *stdin*



# Comentarios personales

## Problemas encontrados y decisiones tomadas

He encontrado grandes dificultades al encontrarme ante un lenguaje que no he utilizado nunca, así como la dificultad añadida de no contar con un compañero en una práctica orientada a su resolución en grupos de dos y, además, escasez de tiempo debido a compaginar los estudios con una beca de colaboración concedida por la universidad.

Sin embargo, he sido capaz de solventar estas dificultades gracias a los apuntes de la asignatura, un esfuerzo extra y conocimientos previos de otros lenguajes como Java.

Durante la realización de la práctica me he encontrado con varios aspectos de la misma sobre los que he tenido que tomar decisiones.

Por ejemplo, la elección de la implementación de las funciones, generalmente buscando la mejor optimización y eficacia del código.

-En la función *tail* se ha elegido esta implementación porque resulta la más eficaz, pues se reserva el menor espacio de memoria posible y en cada iteración la complejidad es O(1).

-En la función *longlines*, por el contrario, la complejidad se eleva a O(n), pues por cada línea recibida tiene que volver a comprobar cuál es la de menor longitud.

Otro ejemplo es la decisión de limitar el tamaño máximo de las líneas de texto, buscando robustecer el código.

El tiempo dedicado a esta práctica me ha servido para asentar las bases de otro lenguaje de programación más, permitiendome tener un punto de partida para seguir evolucionando en el aprendizaje del lenguaje.