# Proyecto en LEX:
# Comparador de hardware

Julio A. Fresneda García – 49215154F – juliofresnedag@correo.ugr.es

El proyecto consiste en un programa que leerá 6 archivos de texto: 5 son bases de datos de componentes, que deben estar en el mismo directorio que el programa, y el sexto será el documento de texto a analizar. Para abrir y analizar los 6 documentos, se ha sobrecargado la función yywrap().

El programa buscará en el documento de texto el nombre de cualquier pieza de hardware, siempre que sea CPU, GPU, RAM, o disco duro . No hace falta que esté la marca, pero es indispensable que el nombre técnico esté escrito correctamente.

Una vez obtiene esos componentes, los comprueba en su base de datos, e imprime en pantalla la marca, el nombre técnico, la puntuación obtenida en benchmarkings, la frecuencia en el caso de CPUs, el precio, y en el caso de las memorias RAM, la capacidad. No imprime la capacidad de los discos duros ya que en la mayoría de ellos está impícita en el nombre técnico, por lo que es redundante.

Después de imprimir esta información, imprime en pantalla los nombres de los componentes ordenados de mayor a menor puntuación, y separados por tipo de componente.

## Reglas LEX

En este programa se usan siete reglas:

- Para el precio
- Para el precio cuando no está especificado (NA)
- Para los puntos de benchmarking
- Para el nombre técnico (si es compuesto)
- Para el nombre técnico (si es sólo una palabra)
- Para la frecuencia de reloj (CPUs)
- Para el salto de línea

El precio se caracteriza por que empieza por "$" o es "NA", si no hay precio disponible.

Los puntos se caracterizan por que es un número en la línea, sin ningún carácter, excepto la coma y el punto.

El nombre técnico compuesto se caracteriza por que hay algún espacio entre ambos. En este caso, la primera palabra se guarda como la marca del componente, y el resto, como el nombre técnico.

En el caso de que el nombre del componente sea sólo de una palabra, se almacenará tanto como marca como nombre técnico.

La frecuencia de reloj se caracteriza por que empieza por @.

## Estructura de almacenamiento

Se ha creado una clase BaseDatos, la cual guarda toda la información recogida en los cinco primeros documentos.

Esta clase tiene dentro cuatro estructuras de tipo std::unordered_map, (una para cada tipo de componente), las cuales se caracterizan por una clave (el nombre técnico del componente) y un objeto que guarda el resto de información. De esta forma, para obtener toda la información, sólo es necesario usar la clave.

También hay cuatro estructuras de tipo std::set, que guardan los objetos con toda la información. Se usa set porque así se ordenan automáticamente de mayor a menor puntuación, cosa que viene muy bien para mostrar los tops.

## Funcionamiento

El programa sigue este orden de funcionamiento:

1. Abre CPUs.txt y guarda toda la información en la base de datos
2. Abre GPUs.txt y guarda toda la información en la base de datos
3. Abre Discos.txt y guarda toda la información en la base de datos
4. Abre RAMsEsc.txt y guarda toda la información en la base de datos. Como no tenemos velocidad de lectura, la ponemos a cero.
5. Abre RAMsLec.txt y actualiza la velocidad de lectura.
6. Abre el documento a analizar. Coge cada línea del documento, y la analiza con un algoritmo.

Los cinco primeros archivos contienen el nombre completo del componente, su puntuación (velocidad de lectura o escritura en el caso de las RAMs) y su precio.

Para cambiar de un documento a otro, se ha sobrecargado yywrap(), de forma que cada vez que acaba de leer un documento, apuntamos yyin hacia el siguiente y le hacemos yyrestart() al archivo, para que vuelva a empezar a leer.

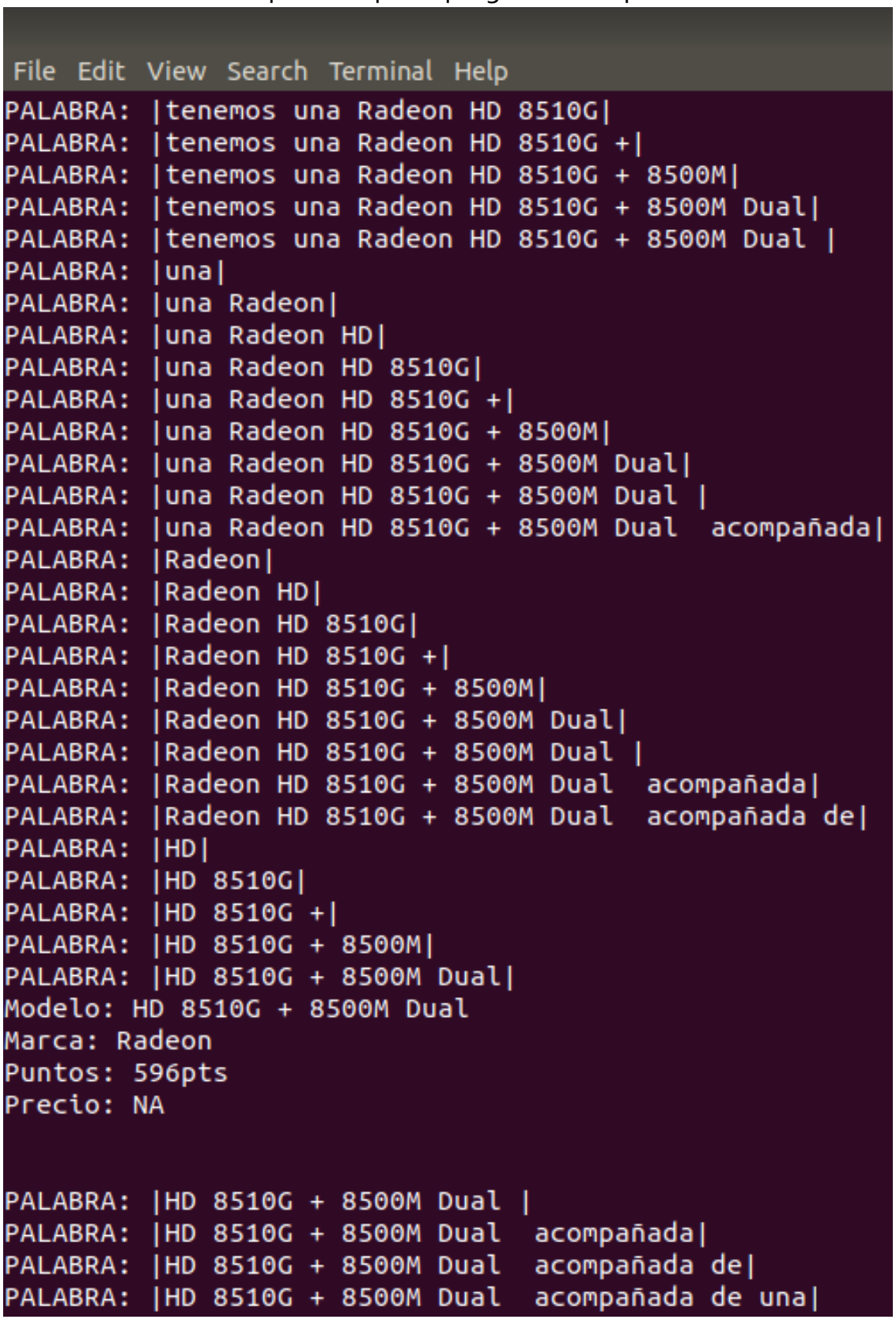
En cuanto al sexto paso, hay un inconveniente obvio para coger los componentes de una línea: No sabemos en qué palabra empieza y en qué palabra acaba el componente, ni si una palabra del lenguaje natural es parte del componente o no. Un ejemplo: A priori no sabemos si en la frase:

"Me he comprado un AMD FX-8350 nuevo" la palabra "nuevo" está dentro del nombre técnico.

Para solucionar esto se ha empleado un algoritmo, que consiste en ir palabra por palabra en cada frase, y para cada palabra comprobar si hace "match" con la base de datos esa palabra, esa palabra sumada a la palabra siguiente, las dos palabras sumadas a la siguiente, etc.

Como es difícil de entender, adjunto una foto como ejemplo, y explicaré el algoritmo sobre ella.

En esta foto vemos cada palabra que el programa comprueba si existe en su BD.



Voy a recortar la parte interesante:

```
PALABRA: |Radeon HD 8510G + 8500M Dual |
PALABRA: |Radeon HD 8510G + 8500M Dual  acompañada|
PALABRA: |Radeon HD 8510G + 8500M Dual  acompañada de|
PALABRA: |HD|
PALABRA: |HD 8510G|
PALABRA: |HD 8510G +|
PALABRA: |HD 8510G + 8500M|
PALABRA: |HD 8510G + 8500M Dual|
Modelo: HD 8510G + 8500M Dual
Marca: Radeon
Puntos: 596pts
Precio: NA


PALABRA: |HD 8510G + 8500M Dual |
PALABRA: |HD 8510G + 8500M Dual  acompañada|
PALABRA: |HD 8510G + 8500M Dual  acompañada de|
PALABRA: |HD 8510G + 8500M Dual  acompañada de una|
```

El programa comprueba si cada una de las palabras existe en la base de datos. En este fragmento, vemos que comprueba las siguientes palabras:

"Radeon HD 85… acompañada" y no existe, por lo que adhiere la siguiente palabra.

"Radeon HD 85… acompañada de" y no existe, por lo que vuelve a adherir la siguiente palabra. El límite de número de palabras que se adhieren es de 8. Una vez llegado a ese límite, pasamos a la siguiente palabra: HD

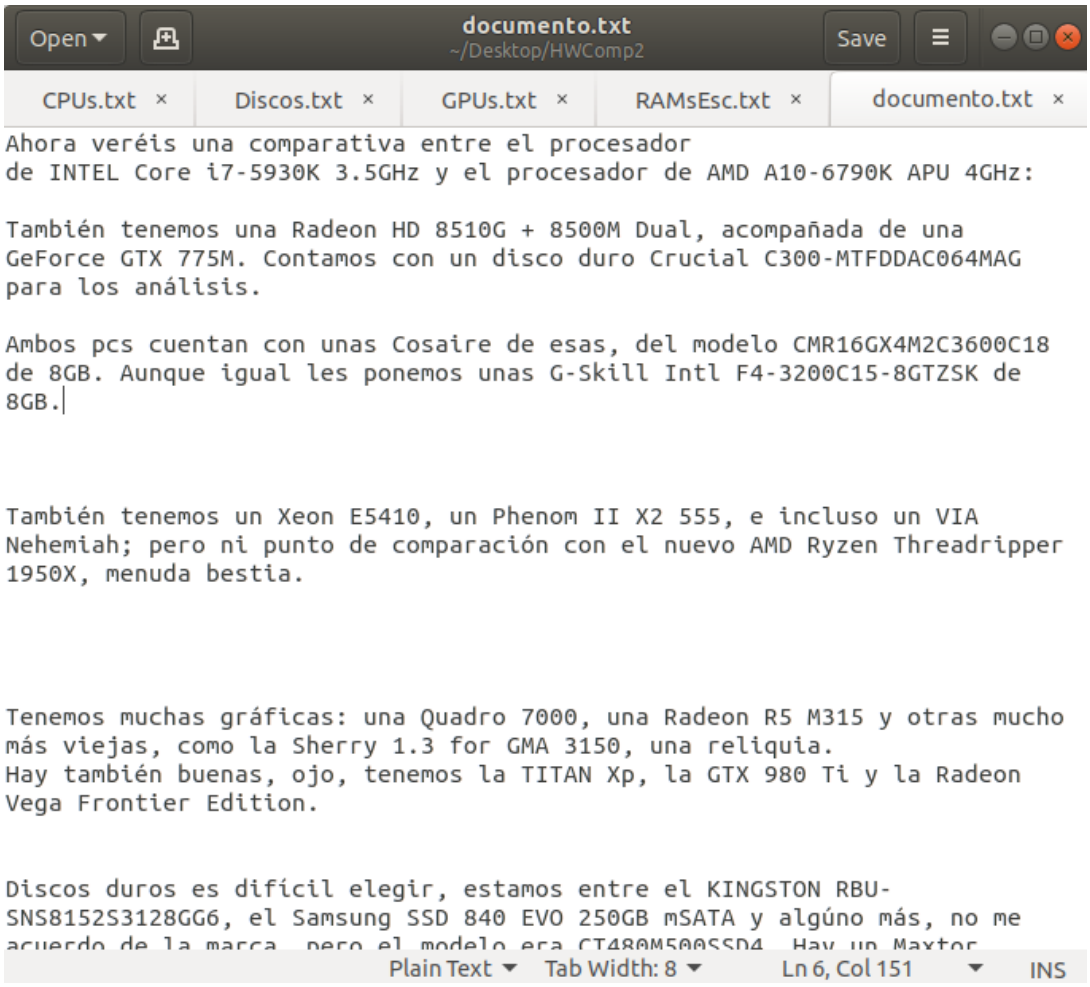
Vemos que va adhiriendo palabras hasta que tenemos la palabra "HD 8510G + 8500M Dual", la cual es el nombre técnico de la gráfica, por lo que el programa la encuentra en la base de datos, guardándolos en un set.

Cuando se acaba la línea, vuelve a empezar con la siguiente, y el proceso continúa hasta la última línea del documento.

En este ejemplo se aprecia muy bien por qué es útil usar <map> para guardar la información: A pesar de que son muchísimas comprobaciones, la buena eficiencia de los mapas hace que esto no sea un problema.

## Ejemplo de funcionamiento

Vamos a analizar un texto donde, con lenguaje natural, se mencionan muchos componentes.

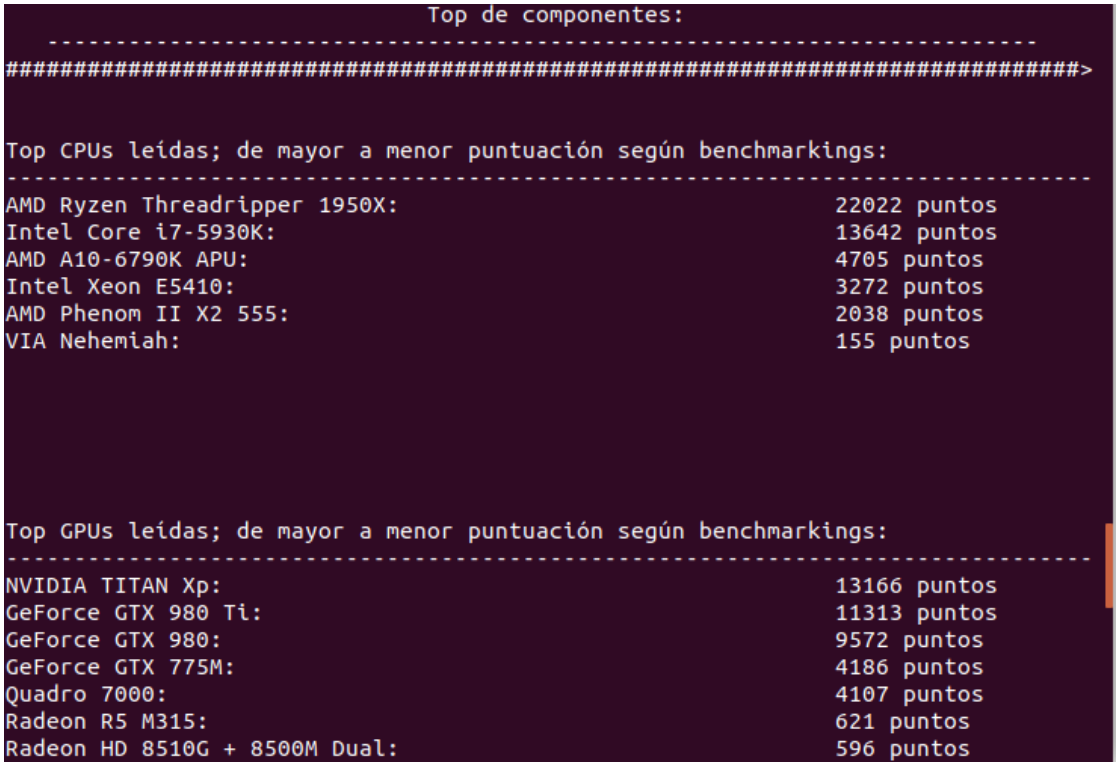



Lanzamos el programa:

```
julioxxx@julio-pc:~/Desktop/HWContrast$ lex HWContrast.l
julioxxx@julio-pc:~/Desktop/HWContrast$ g++ -o prog lex.yy.c BaseDatos.cpp  -ll
julioxxx@julio-pc:~/Desktop/HWContrast$ ./prog documento.txt
```

Primero, nos dice uno a uno las características de cada componente.


Por último, nos los ordena de mejor a peor puntuación.



Como fuente de los datos, he usado https://www.cpubenchmark.net

FIN