A Contexto

CUACKER es nuestro nuevo sistema de nano-blogging. Igual que TWITTER, permitirá mandar mensajes cortos de texto a través del móvil. Pero, además, también se podrán crear mensajes mediante llamadas perdidas a un número de teléfono; según el número de toques, se interpretará como un mensaje predefinido. De estar forma el coste para el usuario es cero.

Queremos que CUACKER se convierta en el nuevo referente de la web a nivel mundial. Por ello, estamos decididos a que sea capaz de manejar ingentes volúmenes de mensajes, de usuarios y de consultas. Ahora mismo, por ejemplo, se estima que TWITTER es capaz de manejar 300 millones de usuarios activos, con 500 millones de mensajes y más de 2.000 millones de consultas al día. ¡Nuestro sistema no será menos!

B El Problema

Analizar, diseñar e implementar el motor de un sistema de nano-blogging, que permitirá mandar mensajes (los llamados *cuac*), consultarlos y buscarlos por distintos criterios. El programa admitirá una serie de operaciones, que se leerán siempre desde la entrada estándar, produciendo el resultado en la salida estándar. Las operaciones admisibles serán las siguientes:

- **Insertar MCUAC**: inserta un nuevo mensaje de texto corto (denominado *mcuac*) de un usuario, con una fecha y hora dadas.
- **Insertar PCUAC**: inserta un nuevo mensaje del tipo "llamada perdida" (denominado *pcuac*) de un usuario, con una fecha y hora dadas.
- **Buscar últimos**: busca los últimos mensajes insertados en el sistema.
- **Buscar por fechas**: busca los mensajes enviados entre dos fechas dadas.
- **Buscar usuario**: busca todos los mensajes de un usuario.
- **Buscar temas**: busca todos los mensajes que contengan cierto tema.
- **Búsqueda combinada**: búsqueda combinada por varios criterios.

Una vez introducidos los *cuac* en el sistema, es indiferente –de cara a las operaciones de búsqueda– que sean de tipo *mcuac* o *pcuac*.

El programa diseñado deberá cumplir los siguientes requisitos:

- Se requiere que no se pierda ni un solo mensaje en las búsquedas.
- El programa tiene que estar bien diseñado para conseguir la máxima eficiencia de tiempo y de memoria (con especial hincapié en lo primero).

C Formato de entrada

La entrada está compuesta de una secuencia de operaciones. Las operaciones pueden ocupar una o varias líneas. Los comandos admisibles son: **mcuac** y **pcuac** (para insertar un nuevo *cuac*), **last** (buscar los últimos *cuac*), **follow** (buscar los *cuac* de un usuario), **date** (buscar por fechas), **tag** (buscar un tema), **search** – **endsearch** (búsqueda combinada) y **exit** (salir). La entrada acabará siempre con una operación **exit**.

El formato de las operaciones es el siguiente:

- **mcuac**: después de la palabra clave **mcuac** aparecerá el mensaje. En primer lugar el nombre de usuario, en la siguiente línea la fecha/hora y en la siguiente línea el texto, con un máximo de 140 caracteres. Ejemplo:

  ```
  mcuac RafaelNaval
  25/10/2018 13:45:11
  ¡Feliz Navidad #amigosdenaval y próspero año nuevo!
  ```

  En nuestro sistema no es necesario que los usuarios se registren. Cuando un usuario manda su primer mensaje, ya queda registrado de forma automática.

- **pcuac**: el formato es muy similar al del comando **mcuac**, con la única diferencia de que en lugar del mensaje aparecerá un número entre 1 y 30. Ejemplo:

  ```
  pcuac RafaelNaval
  28/11/2018 11:27:08
  5
  ```

  Existe una tabla de traducción entre el número y el contenido del mensaje. Por ejemplo, el valor 5 significa: “Enhorabuena, campeones!”.

- **last**: lista los últimos **n** *cuac* (los más recientes) y los muestra por pantalla. A todos los efectos, los mensajes de tipo **mcuac** y **pcuac** son tratados de la misma manera, con la única diferencia de que los *pcuac* se sustituyen por el texto prefijado correspondiente. Por ejemplo, podemos tener un comando:

  ```
  last 5
  ```

- **follow**: lista todos los *cuac* de un usuario dado. El nombre de usuario nunca tiene espacios en blanco. Por ejemplo:

  ```
  follow RafaelNaval
  ```

- **date**: dadas dos fechas, lista los *cuac* que fueron enviados entre esas fechas, ambas inclusive. Por ejemplo:

  ```
  date 28/10/2018 11:30:00 29/11/2018 18:00:55
  ```

- **tag (operación opcional)**: los *cuac* pueden tener temas (o etiquetas) que son palabras que empiezan por el símbolo “#”. Por ejemplo, #vivacuacker, #baloncesto, #curso_2018_2019, etc. Esta operación sirve para listar todos los *cuac* que contienen la etiqueta indicada. Por ejemplo:

  ```
  tag #amigosdenaval
  ```

- **search** – **endsearch (operación opcional)**: sirve para realizar búsquedas combinadas, utilizando alguno de los comandos de búsqueda anteriores. En primer lugar, aparecerá una línea que solo contiene la palabra **search**; después vendrá una o varias operaciones de búsqueda; y finalmente otra línea con

endsearch. El resultado listará los *cuac* que cumplen todos los criterios de búsqueda. Por ejemplo, una búsqueda combinada podría ser:

```
search
follow RafaelNaval
tag #navidad
last 5
endsearch
```

D Formato de salida

Después de cada operación, se mostrará por pantalla información sobre su resultado. La salida tendrá el siguiente formato:

- **Operaciones mcuac y pcuac**: la salida será una línea que indica el número de *cuac* insertados hasta este momento. Por ejemplo:

  ```
  462 cuac
  ```

- **Operaciones de búsqueda (last, follow, date, tag, search)**: los resultados de las búsquedas irán numerados de forma consecutiva. Para cada uno se indicará en la primera línea el nombre de usuario y la fecha de envío; en la siguiente línea vendrá el texto del *cuac* (si es un *pcuac*, el texto equivalente). La lista estará ordenada desde el *cuac* más reciente hasta el más antiguo. Al final aparecerá otra línea indicando el número total de *cuac* en esta búsqueda. Por ejemplo, el resultado de una búsqueda podría ser:

  ```
  1. RafaelNaval 25/11/2011 13:45:11
     ¡Feliz Navidad #amigosdenaval y próspero año nuevo!
  2. RafaelNaval 28/10/2011 11:27:08
     Enhorabuena, campeones!
  Total: 2 cuac
  ```

E Fases de desarrollo

Para una correcta resolución de esta práctica, se proponen las siguientes fases de desarrollo, que marcan de manera aproximada el ritmo que se seguirá en las clases de prácticas. Se usará el juez on-line de la asignatura, que contendrá los ejercicios correspondientes a cada fase. Las fases de desarrollo son:

E1. Semanas 1, 2 y 3 (hasta el 1 de noviembre): resolver los problemas básicos referentes a conversión de los *pcuac* (001, 002), la clase fecha (003), la clase *cuac* (004), el intérprete de comandos (005), implementación sencilla del conjunto de *cuac* con listas, de las operaciones last, follow y date (006).

E2. Semanas 4 y 5 (hasta el 19 de noviembre): implementar un tipo **Hash** de tablas de dispersión para almacenar los *cuac* indexados por usuario, implementando la operación follow (201).

E3. Semanas 6 y 7 (hasta el 1 de diciembre): añadir una estructura de árboles (ya sea trie, AVL o B) para almacenar los *cuac* por orden de fecha, implementando las operaciones last y follow (301).

E4. Semana 8 (hasta el 7 de diciembre): completar y documentar la práctica. Ese mismo día será la entrega final de la memoria de la práctica.

F Documentación

Al terminar la práctica se entregará la documentación final. Se debe **documentar el resultado final de la práctica, no los programas intermedios**. La memoria se entregará en formato PDF en el Aula Virtual y contendrá obligatoriamente los siguientes apartados:

1. **Portada**. Nombre de los alumnos y e-mail de cada uno.
2. **Análisis y diseño del problema**. Se debe responder a las siguientes preguntas, explicando y justificando todas las decisiones de diseño:
   - ¿Qué clases se han definido y qué relación existe entre ellas? Incluir una representación gráfica de las clases.
   - ¿Qué módulos existen, qué contienen y cuál es la relación de uso entre ellos? Poner una representación gráfica de los ficheros y sus dependencias (includes).
   - ¿Contiene el makefile todas las dependencias existentes?
   - ¿Qué tipo de tablas de dispersión se ha usado y por qué? Justificar la decisión.
   - ¿Qué función de dispersión se ha usado? ¿Se han probado varias? Explicar y comparar los resultados de las distintas funciones probadas.
   - Si se hace reestructuración de las tablas, explicar cómo y justificar la decisión.
   - ¿Cómo se libera la tabla de dispersión?
   - ¿Qué tipo de árboles se han implementado y por qué?
   - ¿Cómo es la definición del tipo árbol y del tipo nodo?
   - Si se han implementado árboles AVL, ¿cómo se hace el balanceo?
   - ¿Cómo se liberan los árboles?
   - ¿Se usan variables globales en el programa final?
3. **Listado del código**. Incluyendo el fichero `makefile` necesario para compilar. Listar solo el programa final (el que contiene todos los comandos).
4. **Informe de desarrollo**. Describir cómo ha sido la coordinación y el reparto del trabajo entre los miembros del grupo. Completar la tabla 1.6 de la página 27 del texto guía, que debe rellenarse mientras se hace la práctica (ver fragmentos del texto guía).
5. **Conclusiones y valoraciones personales**.

G Evaluación de la práctica

G.1 Obligatorio

Para aprobar la práctica se requiere que:

- El programa se pueda ***compilar sin errores*** y, para todos los ejercicios, el código debe haber sido aceptado (resultado "*Accepted*") por el juez on-line de la asignatura (Mooshak). En particular, el programa estará escrito en C++, y el código se deberá compilar en Linux.
- El programa debe ***funcionar correctamente***, sin colgarse y produciendo ***resultados correctos*** para el conjunto de pruebas que se determinen. Para ello, el profesor puede usar (pero no está limitado a) los casos de prueba incluidos en el juez on-line de la asignatura.

- La ***memoria de la práctica*** debe contener todos los puntos indicados en el apartado F, y debe ser entregada en el plazo que se establezca. ¡La documentación entregada no debe contener *faltas de ortografía* (incluida la omisión de tildes)!

G.2 Criterios de valoración

La práctica se puntuará de acuerdo con los siguientes criterios de calidad del software:

- **Análisis y diseño**. Se valorará la calidad y adecuación del diseño y análisis realizados, y la elección de las estructuras de datos más adecuadas para cada necesidad.
- **Modularidad.** La funcionalidad debe estar bien repartida entre los módulos. Se debe respetar la ocultación de la implementación, usar *makefile* y compilación separada.
- **Uso del lenguaje**. El código deber ser claro, legible, robusto y eficiente. No crear procedimientos muy largos y complejos. Usar correctamente las características de C++.
- **Eficiencia**. Un aspecto fundamental en la valoración de la práctica será la eficiencia conseguida por el programa, tanto de tiempo como de uso de memoria.

G.3 Otras cuestiones

La práctica se deberá realizar preferiblemente en **grupos de dos alumnos**. De forma extraordinaria se permiten **grupos de 1 alumno**, pero no se prevé ninguna reducción del trabajo para los mismos.

Para realizar pruebas y para la verificación de las fases de desarrollo, los profesores dejarán en la página web del juez on-line (http://dis.um.es/~mooshak/), dentro del concurso “AED1, 11/12. Práctica”, los problemas mencionados en el apartado E. Cada alumno dispondrá de un usuario y clave para acceder a este sistema; el grupo deberá elegir y utilizar una de las cuentas para hacer los envíos al juez.

La fecha de entrega definitiva de esta práctica es el **7 de diciembre de 2018**. El concurso del juez se cerrará ese día a las 10:00, siendo la entrega de la documentación en PDF a lo largo del mismo día, a través de la Tarea correspondiente del Aula Virtual.

AVISO IMPORTANTE

Existe un mecanismo de comprobación automática de todos los envíos, de todos los grupos, de todas las titulaciones. La copia de esta práctica (fuera de la coincidencia casual) supondrá el suspenso fulminante de toda la asignatura en la convocatoria que corresponda, para los grupos implicados en la copia, con la consiguiente anulación de las actividades de evaluación continua.