Trabajo Práctico 2

El trabajo práctico número 2 tiene fecha de entrega para el día **20/11**, y está divido en tres partes:

Contenido

Introducción

Se quiere implementar una nueva red social llamada AlgoGram. De momento, debemos realizar la implementación del sistema de posteos y que esto le llegue a los demás usuarios, y que cada usuario pueda tener un *feed* que priorice la relación con otros usuarios, poder *likear* un *post*, entre otras opciones.

Funcionamiento de la red social

Para simplificar esta etapa, la red social deberá funcionar como una *aplicación de consola*: al iniciar deberá leer un archivo donde se encontrarán los usuarios registrados (con un formato definido a continuación). Luego, la aplicación se quedará pidiendo *órdenes* (o *comandos*) por entrada estándar. Algunas de estas órdenes requieren que haya un usuario *loggeado*.

El programa recibirá la ruta del archivo de usarios como argumento del mismo:

```
./tp2 usuarios.txt
```

Dicho archivo contiene un nombre de usuario por línea, y nada más. Cada línea corresponde a un usuario diferente. Se les garantiza que no habrán nombres duplicados dentro del sistema.

Datos disponibles

Se deja en el sitio de descargas un archivo con casos de prueba que pueden utilizar tanto de archivos de entrada, como de comandos a ser ejecutados y pruebas automatizadas, además de los restantes archivos que se mencionan de aquí en adelante.

Para ejecutar las pruebas correr:

```
$ ./pruebas.sh PATH-A-EJECUTABLE-TP2
```

Interacción

Como se indica antes, una vez que el programa carga los usuarios, debe quedarse esperando por comandos a ejecutar. La mayoría de los comandos requerirán de parámetros (que pueden ser más de uno). Por simplificación, cada parámetro vendrá en líneas separadas. Algunos comandos pueden tener casos de error, que serán detallados en cada caso. Todas las salidas (correctas o de error) deben hacerse a salida estándar.

El programa terminará cuando termine la entrada estándar.

Comandos a implementar

1. Login

Se recibe por parámetro nombre de usuario. Por ejemplo:

```
login
chicho1994
```

Si no había ya un usuario que se haya loggeado, el programa debe escribir por salida estándar `Hola <USUARIO>`. En nuestro ejemplo:

```
Hola chicho1994
```

En caso que ya hubiera un usuario loggeado, debe escribir: `Error: Ya habia un usuario loggeado`. En caso que el usuario indicado no exista, debe escribir: `Error: usuario no existente`. Para nuestro ejemplo, `chicho1994` existe.

2. Logout

No se reciben parámetros. Ejemplo:

```
logout
```

Si había un usuario loggeado (i.e. se había utilizado el comando anterior), este se debe "desloggear", e imprimir `Adios`. Cualquiera de los comandos que requieran un usuario loggeado (incluyendo este) no deben considerar más al usuario anteriormente loggeado.

En caso que no haya habido un usuario loggeado, debe imprimir `Error: no habia usuario loggeado`.

Por ejemplo:

```
login
chicho1994
logout
logout
```

Debe tener la siguiente salida:

```
Hola chicho1994
Adios
Error: no habia usuario loggeado
```

3. Publicar un post

Se crea un post con un texto determinado, cuyo publicador es el usuario loggeado actualmente. Dicho post se le publicará al resto de los usuarios (que podrán verlo cuando se loggeen, según se indicará en el siguiente comando).

El texto a publicar es el único comando del programa. Por ejemplo:

```
publicar
Tiene todo el dinero del mundo, pero hay algo que no puede comprar... un dinosaurio
```

Si hay un usuario loggeado, se debe crear dicho post como es indicado antes, y todos los usuarios (salvo el loggeado actualmente) deben ahora tenerlo para ver en sus respectivos *feeds*. El post debe generarse con un `id`, comenzando en 0 y aumentando en 1 por cada post publicado (para simplificar las pruebas de este trabajo). Se debe imprimir `Post publicado`.

En caso que no hubiera un usuario loggeado, se debe imprimir `Error: no habia usuario loggeado`.

4. Ver proximo post en el feed

Este comando permite visualizar la información del siguiente post del feed. ¿Cuál es el siguiente post? de todos los posts que queden para ver, será el del usuario con *más afinidad* (dentro de los publicadores de esos posts) al usuario loggeado.

¿Cómo se define la afinidad? Por simplicidad en esta etapa, será simplemente quienes se encuentren *más cerca* en el archivo de usuarios (más adelante en la materia veremos otras formas de hacer esto). Por ejemplo:

```
chorch
cacatua2030
mondi
chicho1994
eldiego
```

Según la lógica planteada, `chicho1994` tiene más afinidad con `mondi` y `eldiego` que con `cacatua2030`, pero tiene más afinidad con este que con `chorch`. A su vez, tiene la misma afinidad con `mondi` que con `eldiego`.

En caso que dos posts tengan sendos usuarios con misma afinidad al loggeado (sea como el caso entre `mondi` y `eldiego`, o porque se trata del mismo usuario), se debe visualizar el post que primero se haya creado.

Al determinarse cuál es el siguiente post a revisar, se debe imprimir su información: su ID, su publicador, qué se dijo, y cuántos likes tiene. El formato exacto se muestra en el ejemplo a continuación:

```
login
chicho1994
publicar
Tiene todo el dinero del mundo, pero hay algo que no puede comprar... un dinosaurio
logout
login
chorch
publicar
te corto internet
publicar
es por el teorema de chuck norris
logout
login
cacatua2030
ver_siguiente_feed
ver_siguiente_feed
ver_siguiente_feed
logout
login
mondi
ver_siguiente_feed
ver_siguiente_feed
logout
login
chicho1994
ver_siguiente_feed
```

El resultado esperado en este caso será:

```
Hola chicho1994
Post publicado
Adios
Hola chorch
Post publicado
Post publicado
Adios
Hola cacatua2030
Post ID 1
chorch dijo: te corto internet
Likes: 0
Post ID 2
chorch dijo: es por el teorema de chuck norris
Likes: 0
Post ID 0
chicho1994 dijo: Tiene todo el dinero del mundo, pero hay algo que no puede comprar... un dinosaurio
Likes: 0
Adios
Hola mondi
Post ID 0
chicho1994 dijo: Tiene todo el dinero del mundo, pero hay algo que no puede comprar... un dinosaurio
Likes: 0
Post ID 1
chorch dijo: te corto internet
Likes: 0
Adios
Hola chicho 1994
Post ID 1
chorch dijo: te corto internet
Likes: 0
```

En caso que un usuario no tenga más posts para ver, o bien que no haya usuario loggeado, imprimir `Usuario no loggeado o no hay mas posts para ver`.

5. Likear un post

Se recibe el id del post a likear, y se agrega al usuario loggeado entre los que likean a dicho post. Ejemplo:

```
login
chicho1994
publicar
a la grande le puse cuca
likear_post
0
```

Suponiendo que el post de `chicho1994` es el primer post (y se auto-likeó), los demás usuarios al ver el post en el feed verán que tiene 1 like (a diferencia del ejemplo anterior, donde todos los posts tenían 0).

En caso que haya un usuario loggeado y el post en cuestión exista, se debe imprimir `Post likeado`. En caso contrario: `Error: Usuario no loggeado o Post inexistente`. En caso que un usuario likee dos veces el mismo post, se debe simplemente imprimir `Post likeado`, pero no contará como otro like más.

6. Mostrar likes

Se recibe el id de un post, y se muestran los nombres de todos los usuarios que *le dieron like*, en orden alfabético, uno por línea. No se requiere que haya un usuario loggeado para aplicar esta operación.

Ejemplo:

```
mostrar_likes
0
```

Una salida podría ser:

```
El post tiene 4 likes:
	chorch
	chicho1994
	eldiego
	mondi
```

Suponiendo que, por supuesto, dichos 4 usuarios le dieron like a esa publicación, y son los únicos que así lo hicieron. Notar que la separación es con un caracter de tabulación. En caso que un post no tenga likes, o bien no exista el id, imprimir `Error: Post inexistente o sin likes`.

Criterios de aprobación

Los siguientes aspectos son condición necesaria para la aprobación del trabajo práctico.

Estructuras de datos

Es necesario emplear la estructura de datos más apropiada para cada función del programa, en particular **teniendo en cuenta la complejidad temporal**. También es necesario utilizar dicha estructura de la forma más apropiada posible para optimizar de manera correcta su funcionamiento.

Se puede (y alentamos fuertemente) implementar otros TDAs o estructuras que faciliten o mejoren la implementación de este Trabajo Práctico. Les recordamos que un TDA no es un simple `struct` donde se guarda información, sino que debe tener comportamiento. Esto será un punto muy importante en la evaluación del Trabajo Práctico.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas. No deben modificar los TDAs ya implementados hasta ahora.

Complejidad de los comandos

Estudios de mercado han determinado que es muy probable que hayan muchos usuarios utilizando la red social social en muy poco tiempo (fuente del estudio). Por esto, es necesario que el programa sea muy eficiente. Se impusieron las siguientes restricciones para los comandos:

- Login: debe funcionar en $\mathcal{O}(1)$.

- Logout: debe funcionar en $\mathcal{O}(1)$.
- Publicar Post: debe funcionar en $\mathcal{O}(u \log(p))$, siendo u la cantidad de usuarios y p la cantidad de posts que se hayan creado hasta ese momento.
- Ver próximo post en el feed: debe funcionar en $\mathcal{O}(\log(p))$.
- Likear un post: debe funcionar en $\mathcal{O}(\log u)$.
- Mostrar likes: debe funcionar en $\mathcal{O}(u)$.

Código del programa

El código entregado debe:

- ser claro y legible, y estar estructurado en funciones lo más genéricas posible, adecuadamente documentadas.
- compilar sin advertencias y correr sin errores de memoria.
- ajustarse a la especificación de la consigna y pasar todas las pruebas automáticas.

Informe

Este trabajo práctico tiene como principal foco el diseño del mismo (modularización, creación de nuevas estructuras y TDAs específicos del TP) así como el uso apropiado de las estructuras de datos vistas en clase. Por lo tanto, se les pide que escriban un informe de dicho diseño, puesto que además de ser capaces de escribir código correcto, también es necesario que aprendan a poder *transmitir* el porqué de sus decisiones, utilizando el lenguage técnico y justificaciones correspondientes.

El informe deberá consistir de las siguientes partes:

- carátula con los datos personales del grupo, y ayudante asignado.
- análisis y diseño de la solución, en particular: algoritmos y estructuras de datos utilizados, justificando conforme a los requisitos pedidos en este TP.

Entrega

La entrega incluye, obligatoriamente, los siguientes archivos:

- El código de los TDAs utilizados.
- El código del TP.
- El informe (en formato `.pdf`).
- Un archivo *deps.mk* que exprese las dependencias del proyecto en formato makefile. Este archivo deberá contener solamente dos líneas que indiquen, para cada programa, de qué *objetos* depende su ejecutable; por ejemplo:

```
# Ejemplo de archivo deps.mk para el TP2
algogram: tp2.o algogram.o hash.o
```

La entrega se realiza exclusivamente en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.