Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática
Paradigmas de Programación

Paradigmas de Programación

Proyecto semestral de laboratorio

Versión Preliminar - actualizada al 19/05/2021

Durante el semestre se trabajarán distintos paradigmas de programación, los cuales serán abordados mediante cuatro lenguajes de programación en cuatro entregas de laboratorio. Para el desarrollo satisfactorio del laboratorio del curso se requerirá desarrollar un sistema de redes sociales similar a Facebook, Instagram y Twitter, entre otros.

El software en su versión final (Laboratorio N°4) deberá tener una interfaz gráfica que permita una interacción similar a la presentada en una red social.

1. Descripción general

Varios aspectos del software quedarán abiertos a la creatividad de cada estudiante, sin embargo se debe cumplir con requerimientos mínimos obligatorios, los cuales se especificarán en las siguientes secciones.

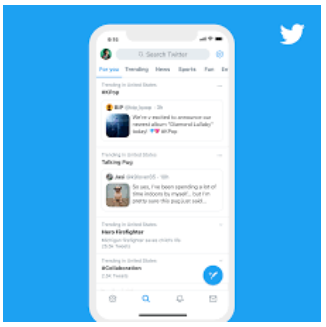
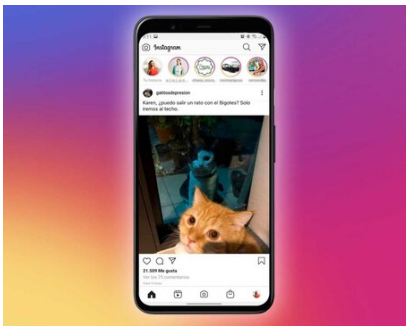
Los primeros tres laboratorios tendrán interfaces claramente definidas (esto es, formatos de lectura, cabeceras de funciones, etc). **El no respetar estas interfaces/prototipos implicarán la obtención de la nota mínima y por consecuencia la reprobación del laboratorio.**

Para este semestre se ha definido hacer una red social. Al final del semestre los estudiantes de Paradigmas de Programación habrán implementado distintas versiones de esta simulación de software en distintos lenguajes de programación, según los paradigmas que los rigen. A continuación se listan de forma general características del software que serán abordadas de forma obligatoria o complementaria durante el semestre en los distintos proyectos. Los detalles para cada laboratorio se listan posteriormente.

Las redes sociales son estructuras compuestas por un grupo de actores (personas, empresas, fundaciones, bots, etc.) interconectados a través de relaciones uni- o bi-direccionales. En la actualidad existen diversos tipos de implementaciones tecnológicas para redes sociales las que se despliegan sacando partido de Internet. Esto permite que actores distribuidos a lo largo del mundo puedan comunicarse a pesar de las barreras geográficas que los separan.

Dentro de las operaciones que típicamente se pueden realizar en estas plataformas tecnológicas de redes sociales se encuentran:

1) Crear una cuenta
2) Iniciar sesión
3) Añadir/Seguir amigos/persona/empresa/página
4) Eliminar/Dejar de seguir amigos/persona/empresa/página
5) Crear/Eliminar/Editar una publicación
6) Responder una publicación
7) Compartir una publicación
8) Enviar mensaje privado
9) Reaccionar a una publicación (e.g., like)

1.1 Cuenta: Corresponde al registro de un actor que lo habilita para crear/editar/eliminar/responder/compartir publicaciones, reaccionar a estas y administrar sus redes de contacto. Las cuentas tienen una fecha/hora de creación. La cuenta permite establecer la relación entre un actor determinado, publicaciones y otros actores con los que interactúa.

1.2 Actores/Usuarios: Se entiende por actor a cualquier usuario de la plataforma. Estos pueden ser personas naturales, celebridades, empresas, fundaciones, bots, entre otros.

1.3 Publicaciones: Se entiende por publicaciones al contenido (texto, imágenes, videos, audio, enlaces) que son creados o compartidos por los actores. Las publicaciones tienen una fecha/hora de publicación.

1.4 Reacciones: Se entiende por reacciones a las respuestas de usuarios a una publicación. Un ejemplo de una reacción es dar me gusta (like) a una publicación.

Comodines: Durante el semestre cada alumno dispondrá de **CUATRO comodines (uno para cada laboratorio)** que podrá utilizar **solo en el siguiente caso**:

- **Si su calificación** en los laboratorios 1, 2, 3 y parcialmente en el 4 (Ya sea por requerimientos funcionales, no funcionales, informe) **es mayor igual que 2.0 e inferior a 4.0**.

El comodín permitirá al alumno entregar lo necesario (requerimientos funcionales, no funcionales y/o informe) para alcanzar la nota mínima requerida (4.0) en la correspondiente entrega de laboratorio para la aprobación al final del semestre. **Es fundamental que los estudiantes siempre entreguen un avance (dentro de los plazos establecidos para cada laboratorio) que permita alcanzar al menos una calificación igual o superior a 2.0 para poder optar al uso de comodines**. **Recordar que para aprobar el laboratorio de esta asignatura, todos los laboratorios deben tener una calificación igual o mayor a 4.0.**

Para el caso del uso de comodín por versionamiento solo dispondrá de 2 oportunidades. Si el estudiante tiene una calificación menor a 4.0 debido al no uso o uso inadecuado de git (frecuencia, constancia, periodo), el estudiante podrá enmendar esta calificación haciendo uso efectivo de git en los siguientes laboratorios o en los anteriores. Para estos casos la evaluación del comodín, el estudiante debe enviar la solicitud de aplicación de comodín por versionamiento al término del semestre a través de la plataforma habilitada para tales efectos. De ser aceptado el comodín, la nota de cierre del correspondiente laboratorio será un 4.0.

Antes de proceder al uso de comodines, y si lo amerita, se recomienda a los alumnos que empleen las instancias de recorrecciones de cada laboratorio.

En caso de requerir el uso de comodín, podrá hacer entrega del correspondiente laboratorio corregido al final del semestre por el canal habilitado para estos efectos. La fecha específica y espacio de entrega estará disponible en CampusVirtual al final del semestre.

Finalmente, por temas de tiempo considerando el avance del semestre, en el laboratorio final se podrá hacer un uso limitado del comodín. Esto quedará a disposición de los profesores correctores dependiendo del nivel de los elementos que son calificados como insuficientes y la factibilidad de poder completarlos en los plazos para el cierre oficial del semestre dentro de las semanas lectivas 17.

Instrucciones generales para la entrega de TODOS los laboratorios

Versión 1.0

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Entregables: Archivo ZIP con el siguiente nombre de archivo: labN_rut_ApellidoPaterno.zip (ej: lab1_12123456_Perez.zip). **Notar que no incluye dígito verificador**. Dentro del archivo se debe incluir:

1. Informe en formato Word o PDF.
2. Carpeta con código fuente.
3. Archivo leeme.txt para cualquier instrucción especial para ejecución u otro, según aplique.
4. Autoevaluación.txt donde debe incluir una lista completa de todos los requerimientos funcionales y no funcionales señalando una evaluación para cada uno según la siguiente escala:
    a. 0: No realizado.
    b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
    c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
    d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
    e. 1: Implementación completa sin problemas (funciona 100% de las veces)
5. Archivo repositorio.txt que incluya la URL de su repositorio privado en GitHub[1] el que deberá además ser compartido con los profesores y ayudantes al momento de la evaluación a la cuenta ***paradigmasdiinf*** de GitHub (no lo comparta antes ya que las invitaciones expiran después de un tiempo). El nombre final del repositorio a compartir debe ser el mismo del entregable en Campus Virtual.

El nombre del archivo con el código fuente debe seguir el siguiente formato: NombreArchivo_rut_Apellidos.extension (ej: lab_git_12123456_PerezPeña.rkt)

Se aplicarán descuentos en caso de no incluir estos datos

Informe del proyecto: Extensión no debe superar las **10 planas** de contenido (se excluyen: portada, índice, referencias, anexos), para más detalle de requerimientos del informe ver descripción en nuestro espacio en Campus Virtual. El informe debe incluir introducción, descripción breve del problema, análisis del problema, diseño de la solución (esquematizada y explicada brevemente), consideraciones de implementación (ej: algoritmos, bibliotecas), **instrucciones con ejemplos claros de uso**, resultados obtenidos, evaluación completa y conclusiones.

---

[1] Ver tutorial de Git en Campus Virtual donde se explica como activar el Student Pack para poder crear repositorios privados de forma gratuita.

Repositorio en Git del Proyecto: El laboratorio deberá ser desarrollado incluyendo el sistema de control de versiones Git y utilizando GitHub como servidor remoto. **El repositorio debe ser privado durante la elaboración del proyecto**. Cualquier inconsistencia entre lo entregado en Campus Virtual y la versión final del repositorio será evaluado con nota mínima. **Para el proceso de revisión se considerará la versión subida a CampusVirtual**.

Integridad: Lo/as estudiantes deben desarrollar los laboratorios de forma individual. Si bien lo/as estudiantes pueden apoyarse y discutir aspectos del proyecto, el diseño e implementación del mismo se debe resolver de manera individual. No se pueden compartir diseños o implementaciones, tampoco se pueden obtener de terceros. Cómo parte de la revisión existen procedimientos de revisión de copia/plagio tanto a nivel de código como en los informes. Al detectarse plagio, el trabajo en cuestión no será revisado por no contar con evidencia concreta sobre el resultado de aprendizaje esperado para el correspondiente laboratorio. El/la estudiante tampoco podrá hacer uso de comodín en este caso ya que es el equivalente a no haber entregado dicho laboratorio. Por último se iniciará investigación sumaria a través de la Facultad de Ingeniería con el objeto de determinar las sanciones correspondientes.

Evaluación: El Informe (Inf), requerimientos funcionales (RF), requerimientos no funcionales (RNF), ejecución (Eje) se evalúan por separado. La nota final de este laboratorio (NL) se calcula de la siguiente forma considerando sólo la calificación base a partir del cumplimiento de los RF y RNF obligatorios.

$$
\begin{aligned}
&if\ (Inf\ >=\ 4.0\ \&\&\ (RF\ +\ 1.0)\ >=\ 4.0\ \&\&\ (RNF\ +\ 1.0)\ >=\ 4.0)\ then\\
&\qquad NL\ =\ 0.1\ \cdot\ Inf\ +\ 0.7\ \cdot\ (RF\ +\ 1.0)\ +\ 0.2\ \cdot\ (RNF\ +\ 3.0)\\
&else\\
&\qquad NL\ =\ min(Inf,\ RF\ +\ 1.0,\ RNF\ +\ 1.0)
\end{aligned}
$$

Laboratorio 1 (Paradigma Funcional - Lenguaje Scheme)

Versión 1.0

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Fecha de Entrega: Ver calendario clase a clase donde se señala el hito

Objetivo del laboratorio: Aplicar conceptos del paradigma de programación funcional usando el lenguaje de programación Scheme en la resolución de un problema acotado.

Resultado esperado: Programa que simula las funcionalidades y comportamientos de una red social.

Profesor responsable: Roberto González (al hacer consultas en este documento, procurar hacer la mención a **@Roberto Gonzalez Ibanez** (roberto.gonzalez.i@usach.cl) para que las notificaciones de sus consultas lleguen al profesor correspondiente)

Requerimientos No Funcionales obligatorios. Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados.

2. **(obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Scheme.

3. **(obligatorio) Versión:** Usar DrRacket versión 6.11 o superior

4. **(obligatorio) Standard:** Se deben utilizar funciones estándar de dicho lenguaje y no aquellas propias Racket. Si considera integrar su solución posteriormente con C# en el laboratorio 4 debe regirse por standard R6RS.

5. **(obligatorio) No variables:** No hacer uso de función define **en combinación** con otras como set! (o similares) para emular el trabajo con variables.

6. **(1 pts) Documentación:** Todas las funciones deben estar debidamente comentadas. Indicando descripción de la función, argumentos y retornos. En caso de que la función sea recursiva, indicar el tipo de recursión utilizada y el porqué de esta decisión.

7. **(obligatorio) Dom->Rec:** Respetar la definición de función en términos de conjunto de salida (dominio) y llegada (recorrido) sin efectos colaterales, además del nombre de las mismas (respetar mayúsculas y minúsculas).

8. **(1 pts) Organización:** Estructurar su código en archivos independientes. Un archivo para cada TDA implementado y uno para el programa principal donde solo se dispongan sólo las funciones requeridas en el apartado de requerimientos obligatorios. Debe usar la función require/provide.

9. **(2.5 pts) Historial:** Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas (no espere a terminar la materia para empezar a trabajar en el laboratorio. Puede ir haciendo pequeños incrementos conforme avance el curso)**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.375 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

10. **(obligatorio) Ejemplos:** Al final de su código incluir al menos 3 ejemplos de uso para cada una de las funciones correspondientes a requerimientos funcionales obligatorios y los extra. **Solo se revisarán aquellas funciones para las que existan los ejemplos provistos.**

11. **(obligatorio) Prerrequisitos:** Para cada función se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la función implementada. Ej: Para evaluar la función login, debe estar implementada la función create.

Requerimientos Funcionales. **Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la función no será evaluada.**

1. **(0.5 pts) TDAs**. Implementar abstracciones apropiadas para el problema. Recomendamos leer el enunciado completo a fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno. Luego, planifique bien su enfoque de solución de manera que los TDAs y representaciones escogidos sean aplicables a ambos tipos de funciones.

   Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Funciones de Pertenencia, Selectores, Modificadores y Otras Funciones. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus funciones. En el resto de las funciones se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar las funciones necesarias dentro de esta estructura**.

   Dejar claramente documentado con comentarios en el código aquello que corresponde a la estructura base del TDA. Una estructura base que deberá considerar para el resto de las funciones corresponde a “socialnetwork” que corresponde al contenedor de sesión activa, usuarios, publicaciones.

   Debe contar además con representaciones complementarias para que los usuarios puedan tener información relativa a sus credenciales de acceso y otros elementos que considere relevantes para abordar el problema.

   De igual forma, las publicaciones, además de información relativa a fecha de publicación, ID único incremental (se va generando un correlativo a partir de la última publicación registrada), debe contar con información relativa a reacciones, respuestas, contenido compartido, tipo de publicación (texto, video, url, foto, audio), y otros datos que considere relevante para abordar el problema.

Prerrequisitos para evaluación	Ninguno
Requisitos de implementación	- Usar estructuras basadas en listas - Especificar representación de manera clara para cada TDA implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, funciones de pertenencia, y según se requiera selectores, modificadores y otras funciones que pueda requerir para las otras funciones requeridas a continuación. Para el caso del TDA socialnetwork, el constructor tiene la siguiente firma: String X Date X *EncryptFunction X DecryptFunction* Donde el primer elemento (String) corresponde al nombre de la red social, Date corresponde a una fecha válida de creación de la red social y *EncryptFunction y DecryptFunction* son funciones que permiten encriptar y desencriptar respectivamente todo el contenido (texto) cargado en la red social. La firma de estas funciones para ser aceptadas por socialnetwork es: String ->String Esto es, son funciones que reciben un string y producen una versión modificada del mismo (ej: desplazar caracteres a la izquierda, derecha, sumar un 1, binarizar, etc.)
Funciones mínimas requeridas	*;Constructor socialnetwork* String X Date X *EncryptFunction X DecryptFunction* *(socialnetwork name date encryptFunction decryptFunction)* ;Integer X Integer X Integer *(date dd mm yyyy)*
Ejemplo de otras funciones asociadas al mismo TDA u otros TDAs (Solo de referencia. Cada estudiante define sus propios TDAs)	*(user ….)* *(publication …)*

2. **(0.5 pts) register:** Función que permite registrar a un nuevo usuario en la red social. Para esto se solicita la red social, nombre del usuario (identificador único, se debe verificar que no exista para su correcto registro) y contraseña. El retorno de la función es una red social con el nuevo usuario registrado.

Prerrequisitos para evaluación	Implementación de TDAs
Requisitos de implementación	- Emplear recursión natural
Dominio	socialnetwork X date X string X string
Recorrido	socialnetwork
Encabezado	*(register socialnetwork date username password)*
Ejemplos	*(register facebook (date 25 3 2020) “user” “pass”)*

3. **(0.5 pts) login:** Función que permite autenticar a un usuario registrado iniciar sesión y junto con ello permite la ejecución de comandos concretos dentro de la red social. Si la autenticación es válida (i.e., que el usuario existe). El retorno es una función currificada correspondiente a la operación (operation) parcialmente evaluada con el parámetro socialnetwork actualizado. La actualización de socialnetwork incorpora al usuario autenticado en la sesión activa (fundamental para que el comando pueda funcionar). De lo contrario retorna solo operation.

Prerrequisitos para evaluación	Implementación de TDAs, función register
Requisitos de implementación	- Debe emplear funciones de orden superior. - Retorno es una función currificada - Usar recursión natural o de cola (en alguna de las funciones debe hacer uso de alguno de los estilos)
Dominio	socialnetwork X string X string X function
Recorrido **Recorrido final**	function socialnetwork
Encabezado	*(login* socialnetwork *username password operation)*
Ejemplos	*(login* socialnetwork *“user” “pass” post)*

4. **(0.5 pts) post:** Función currificada que permite a un usuario con sesión iniciada en la plataforma realizar una nueva publicación propia o dirigida a otros usuarios. Cada publicación registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación (tipo Date), el tipo de publicación (“photo”, “video”, “url”, “text”, “audio”) y el contenido de la publicación (solo debe ser un string). El retorno final de la función es una versión actualizada de socialnetwork donde se registra la nueva publicación y se elimina la sesión activa del usuario en socialnetwork.

Prerrequisitos para evaluación	Implementación de TDAs, función login y encrypt functions
Requisitos de implementación	- Esta función no arroja ningún cambio en socialnetwork si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario. - La función debe estar currificada solo para los dos primeros parámetros. El retorno para la evaluación sobre el parámetro socialnetwork es una función que opera sobre el resto de los argumentos. Retorno es una función currificada. - El contenido creado con esta función es encriptada mediante la función encryptFunction contenida en la estructura de socialnetwork (se puede acceder a esta mediante un selector) - Solo se puede publicar mensajes en cuentas de terceros si es que está dentro de la lista de contactos del usuario
Dominio	socialnetwork
Recorrido **Recorrido final**	function: date X string (contenido) X user list socialnetwork
Encabezado	*(post socialnetwork)*
Ejemplos	*(((login facebook “user” “pass” post) (date 30 10 2020)) “mi primer post”) ;post en la cuenta del usuario* *(((login twitter “user” “pass” post) (date 30 10 2020)) “mi primer post” “user1” “user2”) ;post dirigido a los usuarios user1 y user2*
Tip	Para la definición de la función más interna que retorna post y que opera sobre los argumentos “contenido del post” y “usuarios” considere la siguiente definición: *(lambda (content . users) ……)*

5. **(0.5 pts) follow:** Función currificada que permite a un usuario con sesión iniciada en la plataforma seguir a otro usuario. El retorno final de la función es una versión actualizada de socialnetwork donde se registra el cambio y se elimina la sesión activa del usuario en socialnetwork.

Prerrequisitos para evaluación	Implementación de TDAs, función login
Requisitos de implementación	- Esta función no arroja ningún cambio en socialnetwork si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario. - La función debe estar currificada. El retorno para la evaluación sobre el parámetro socialnetwork es una función que opera sobre el resto de los argumentos. Retorno es una función currificada. - Un usuario no se puede seguir a si mismo
Dominio	socialnetwork
Recorrido **Recorrido final**	function: date X user socialnetwork
Encabezado	*(follow socialnetwork)*
Ejemplos	*(((login facebook “user” “pass” follow) (date 30 10 2020)) “user1”) ;user ahora sigue a user1, asumiendo la existencia de ambos usuarios*

6. **(0.5 pts) share:** Función currificada que permite compartir contenido de un usuario en su propio espacio o dirigido a otros usuarios. El retorno final de la función es una versión actualizada de socialnetwork donde se registra la publicación compartida y se elimina la sesión activa del usuario en socialnetwork.

Prerrequisitos para evaluación	Implementación de TDAs, función login, post
Requisitos de implementación	- Esta función no arroja ningún cambio en socialnetwork si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario. - La función debe estar currificada. El retorno para la evaluación sobre el parámetro socialnetwork es una función que opera sobre el resto de los argumentos. Retorno es una función currificada. - Procurar que el contenido compartido quede vinculado de alguna manera al usuario que creó originalmente la publicación lo cual se debe resolver internamente a partir de postID - Solo se puede compartir con usuarios que estén dentro de la lista de contactos. Si no están se omiten
Dominio	socialnetwork
Recorrido **Recorrido final**	function: date X postID X userList socialnetwork
Encabezado	*(share socialnetwork)*
Ejemplos	*(((login facebook “user” “pass” share) (date 30 10 2020)) 54) ;la publicación con ID 54 se comparte en la cuenta de user* *(((login facebook “user” “pass” share) (date 30 10 2020)) 54 “user1” “user2”) ;la publicación con ID 54 es compartida por el usuario user con sus contactos user1 y user2.*

7. **(0.5 pts) socialnetwork->string:** Función que recibe una socialnetwork y entrega una representación del mismo como un string posible de visualizar de forma comprensible al usuario (incluyendo el uso de la función para desencriptar el contenido). Debe hacer uso del char '\n' para los saltos de línea. **No utilice las funciones write y display dentro de esta función**, ya que debe retornar un string el cual pueda luego ser pasado como argumento a la función "write" o "display" para poder visualizarlo de forma comprensible al usuario.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share.
Requisitos de implementación	- El string resultante al ser impreso por la función write o display debe ilustrar con claridad los elementos de socialnetwork (i.e., usuarios y sus publicaciones, sus redes de contactos, etc.). - El resultado de aplicar socialnetwork-string directamente sobre socialnetwork sin usar función login de por medio, imprime todo lo que haya en social network (ver anexo al final de este enunciado para un ejemplo completo). - El resultado de aplicar socialnetwork->string a través de la función login entrega como resultado todos los antecedentes del usuario que está iniciando sesión (publicaciones, contactos, reacciones, publicaciones compartidas) junto a los detalles del usuario que inició sesión (nombre, fecha de creación de cuenta). - El contenido de socialnetwork se debe desencriptar mediante función decryptFunction contenida en la estructura de socialnetwork.
Dominio	socialnetwork
Recorrido	string
Encabezado	*(socialnetwork->string socialnetwork)*
Ejemplos	*(login facebook"user" "pass" socialnetwork->string)* *(socialnetwork->string twitter)*

De las siguientes funciones implementar de manera libre las que estime conveniente para alcanzar un 7.0. Considere igual los prerrequisitos señalados. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.

8. **(1 pts) comment:** Función que permite comentar publicaciones y otros comentarios existentes (propios o de terceros). Los textos de comentarios al igual que la publicación comentada quedan encriptados mediante encryptFunction registrada en socialnetwork. El retorno final de la función es una versión actualizada de socialnetwork donde se registra la publicación comentada y se elimina la sesión activa del usuario en socialnetwork.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, socialnetwork->string
Requisitos de implementación	- Esta función no arroja ningún cambio en socialnetwork si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario. - La función debe estar currificada. El retorno para la evaluación sobre el parámetro socialnetwork es una función que opera sobre el resto de los argumentos. Retorno es una función currificada. - El contenido creado con esta función es encriptada mediante la función encryptFunction contenida en la estructura de socialnetwork (se puede acceder a esta mediante un selector). Publicaciones y comentarios tienen una secuencia única de identificadores para facilitar las referencias a estos contenidos. - Solo se puede comentar publicaciones de contactos
Dominio	socialnetwork
Recorrido	function: date X postID X string (comentario) socialnetwork
Encabezado	*(comment socialnetwork)*
Ejemplos	*((((login facebook “user” “pass” comment) (date 30 10 2020)) 54) “Mi respuesta”) ;comentario sobre publicación 54* *((((login facebook “user” “pass” comment) (date 30 10 2020)) 58) “Mi respuesta”) ;comentario sobre comentario 58*

9. **(0.5 pts) like:** Función que permite reaccionar a publicaciones o comentarios existentes (propias o de terceros) a través de “me gusta”. El retorno final de la función es una versión actualizada de socialnetwork donde se registra la publicación comentada y se elimina la sesión activa del usuario en socialnetwork.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, socialnetwork->string
Requisitos de implementación	- Esta función no arroja ningún cambio en socialnetwork si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario. - La función debe estar currificada. El retorno para la evaluación sobre el parámetro socialnetwork es una función que opera sobre el resto de los argumentos. Retorno es una función currificada. - Se puede dar like a cualquier publicación
Dominio	socialnetwork
Recorrido	function: date X postID (publicación o comentario) socialnetwork
Encabezado	*(like socialnetwork)*
Ejemplos	*(((login facebook “user” “pass” like) (date 30 10 2020)) 54) ;like a la publicación 54*

10. **(1 pts) influencers:** Función que permite consultar influencers tomando como base un una función de ranking que toma un punto de corte y otros datos de libre elección como likes, followers, contactos, etc..

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, comment, like, socialnetwork->string
Requisitos de implementación	- Función de orden superior que recibe una socialnetwork y una función de ranking. La función de ranking es cualquier función que pueda operar sobre los elementos de la estructura socialnetwork para generar un indicador númerico. Luego la función influencer ordena los usuarios tomando como base los resultados de esta métrica y selecciona los k primeros usuarios de acuerdo al punto de corte señalado en topK. - Provea una función llamada score que realice un cáluclo ponderado de número de contactos y likes.
Dominio	socialnetwork X rankingFunction X Integer
Recorrido	user list
Encabezado	*(influencers socialnetwork* rankingFunction topK*)*
Ejemplos	*(influencers facebook* rankingPorLikes 5*)* *(influencers facebook* rankingPorFollowers 10*)* *(influencers facebook* rankingCompuesto 40*)* *etc.*

11. **(0,5 pts) viral:** Función que permite determinar contenido viral dentro de la red social, esto es, determinar publicaciones que hayan sido compartidas K veces.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, comment, like, socialnetwork->string
Requisitos de implementación	La función debe retornar una lista de las publicaciones junto a datos como: número de veces que se ha compartido, autor original de la publicación, tipo de publicación, fecha de publicación.
Dominio	socialnetwork X number
Recorrido	publication list
Encabezado	*(viral socialnetwork* threshold*)*
Ejemplos	*(viral facebook* 100*)* *(viral twitter* 10*)*

12. **(2 pts) search:** Función que permite buscar contenido (publicaciones) dentro de la socialnetwork a partir de una función de comparación (ejemplo: búsqueda exacta, busqueda difusa, partial match).

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, comment, like, socialnetwork->string
Requisitos de implementación	La función de orden superior La función debe retornar una lista de las publicaciones junto a datos como: contenido de la publicación, autor, tipo de publicación, fecha de publicación. Debe proporcionar junto a esta función al menos una función de comparación llamada compareFunction
Dominio	search: socialnetwork X String X compareFunction compareFunction: string X string
Recorrido	search: publication list compareFunction: number [0,N] (valores cercanos a 0 indica que son muy similares. Valores cercanos a N indica que son muy diferentes).
Encabezado	*(search socialnetwork text* compareFunction*)*
Ejemplos	*(search facebook "texto búsqueda"* partialMatch*)* *(search facebook "soda"* exactMatch*)* *(search instagram "soda"* contains*)* *(search twitter "soda"* customFunction*)*

13. **(2 pts) cluster:** Función que permite agrupar publicaciones en K grupoos a partir de la similitud entre estas determinadas por una función de comparación y un umbral.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, comment, like, socialnetwork->string
Requisitos de implementación	La función de orden superior La función debe retornar una lista de K listas de publicaciones agrupadas en base a su similitud junto a datos como: contenido de la publicación, autor, tipo de publicación, fecha de publicación. Debe proporcionar junto a esta función al menos una función de comparación llamada compareFunction
Dominio	cluster: socialnetwork X compareFunction X Integer X Number compareFunction: string X string
Recorrido	search: list of publication lists compareFunction: number [0,1] (valores cercanos a 0 indica que son muy similares. Valores cercanos a 1 indica que son muy diferentes).
Encabezado	*(cluster socialnetwork* compareFunction nClusters threshold*)*
Ejemplos	*(cluster facebook levenshtein 5 0.5)* *(cluster twitter levenshtein 10 0.4)*

14. **(1 pto) detectBots:** Función que permite detectar bots tomando como base aquellas publicaciones o comentarios que puedan resultar recurrentes dentro de la red social. En este caso se refiere,a comentarios o publicaciones que se crean o comparten múltiples veces desde un mismo origen (autor) ya sea en la cuenta misma o en la de terceros.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, comment, like, socialnetwork->string
Requisitos de implementación	La función debe retornar una lista de K cuentas de usuarios junto a datos como: contenido de la publicación que levanta la alerta, autor, tipo de publicación, fecha de publicación. Debe implementar internamente la función (bot? socialnetwork user threshold) que permite verificar si un usuario en particular es considerado un bot.
Dominio	socialnetwork X Integer
Recorrido	user lists
Encabezado	*(detectBots socialnetwork* threshold*)*
Ejemplos	*(detectBots facebook 100)* *(detectBots facebook 150)*

15. **(1 pto) botpost:** Función que permite automatizar la generación de publicaciones en la redsocial a través de un bot.

Prerrequisitos para evaluación	Implementación de TDAs, función register, encrypt function, decrypt function, post, follow, share, comment, like, socialnetwork->string
Requisitos de implementación	Función recursiva crea un usuario, añade a su lista de contactos mediante follow a K usuarios pseudo-aleatorios y comparte con estos una publicación. Si K es mayor que el número de usuarios existentes en la red social, entonces se añaden a todos los usuarios. Usar la función random provista en el curso de manera de no violar la transparencia referencial. Esta función debe ser currificada para poder iniciar parámetros iniciales desde fuera.
Dominio	socialnetwork X String X Integer X String X randomFunction
Recorrido	socialnetwork
Encabezado	*(botpost socialnetwork botName* numberOfContacts content randomFunction*)*
Ejemplos	*(botpost facebook "bot1" 100 "mensaje automático" (randFn1 100))* *(botpost twitter "bot2" 100 "mensaje automático" (randFn2 434))*

Script básico Pruebas

```
(define encryptFn (lambda (s) (list->string (reverse (string->list s)))))

;se usa la misma función de encriptación para desencriptar

;testing constructor socialnetwork

(define emptyFB (socialnetwork “fb” (date 25 10 2021) encryptFn encryptFn))

(define emptyTwitter (socialnetwork “twitter” (date 25 10 2020) encryptFn encryptFn))

(define emptyInstagram (socialnetwork “instagram” (date 25 10 2020) encryptFn encryptFn))

;puede extender esta lista de registro inicial de usuarios con mas usuarios para que pueda hacer pruebas más complejas

;testing función register

(define FB

        (register (register (register emptyFB (date 25 10 2021) “user1” “pass1”) (date 25 10 2021) “user2” “pass2”) (date 25 10 2021) "user3" "pass3"))

;otra red social de ejemplo con 3 usuarios registrados

(define Twitter

        (register (register (register emptyTwitter (date 25 10 2021) “user1” “pass1”) (date 25 10 2021) “user2” “pass2”) (date 25 10 2021) "user3" "pass3"))

;otra red social de ejemplo con 3 usuarios registrados

(define Instagram

        (register (register (register emptyInstagram (date 25 10 2021) “user1” “pass1”) (date 25 10 2021) “user2” “pass2”) (date 25 10 2021) "user3" "pass3"))


;las siguientes funciones se vinculan a nuevas definiciones de funciones constantes para mayor claridad. En la práctica se podrán expresar todas en una misma línea

;testing función login y follow

(define FB1 (((login FB “user1” “pass1” follow) (date 27 10 2021)) "user2"))

(define FB2 (((login FB1 “user1” “pass1” follow) (date 27 10 2021)) "user3"))

(define FB3 (((login FB2 “user2” “pass2” follow) (date 27 10 2021)) "user3"))

(define FB4 (((login FB3 “user2” “pass2” follow) (date 27 10 2021)) "user1"))

(define FB5 (((login FB4 “user3” “pass3” follow) (date 27 10 2021)) "user1"))

;testing función login y post

;Se asume que de aquí en adelante comienza la publicación con ID 0 y luego se incrementan los ID en 1

(define FB6 (((login FB5 “user1” “pass1” post) (date 28 10 2021)) “0st post”))
```

```
(define FB7 (((login FB6 “user1” “pass1” post) (date 28 10 2021)) “1nd post”))
(define FB8 (((login FB7 “user1” “pass1” post) (date 28 10 2021)) “2rd post”))
(define FB9 (((login FB8 “user2” “pass2” post) (date 28 10 2021)) “3th post”))
(define FB10 (((login FB9 “user2” “pass2” post) (date 28 10 2021)) “4th post”))
(define FB11 (((login FB10 “user3” “pass3” post) (date 28 10 2021)) “5th post”))
(define FB12 (((login FB11 “user3” “pass3” post) (date 28 10 2021)) “6th  post”))
(define FB13 (((login FB12 “user3” “pass3” post) (date 28 10 2021)) “7th post”))
(define FB14 (((login FB13 “user3” “pass3” post) (date 28 10 2021)) “8th post”))
(define FB15 (((login FB14 “user3” “pass3” post) (date 28 10 2021)) “9th post” “user1”))
(define FB16 (((login FB15 “user1” “pass1” post) (date 28 10 2021)) “10th post” “user2” “user3”))
;testing función share
(define FB17 (((login FB15 “user1” “pass1” share) (date 2910 2021)) 3 “user2” “user3”))
;testing función login y  socialnetwork->string
(login FB17 “user1” “pass1” socialnetwork->string)
        ;EJEMPLIFICAR EL FORMATO DEL OUTPUT (PENDIENTE)
;testing función socialnetwork->string
(socialnetwork->string FB17)
        ;EJEMPLIFICAR EL FORMATO DEL OUTPUT (PENDIENTE)
;De las siguientes funciones debe probar solo aquellas que ha implementado
;testing función comment
;tomando como base el último post realizado, el primer comentario tendría ID = 11
(define FB18 ((((login FB17 “user1” “pass1” comment) (date 2910 2021)) 3) “11th comment”))
(define FB19 ((((login FB18 “user3” “pass3” comment) (date 2910 2021)) 0) “12th comment”))
;No genera cambios pues trata de comentar publicación de “user2” al cual no sigue
(define FB20 ((((login FB19 “user3” “pass3” comment) (date 2910 2021)) 3) “13th comment”))
;Testing función like
(define FB21 (((login FB20 “user3” “pass3” like) (date 2910 2021)) 3))
(define FB22 (((login FB21 “user1” “pass1” like) (date 2910 2021)) 0))
(define FB23 (((login FB22 “user2” “pass2” like) (date 2910 2021)) 1))
(define FB24 (((login FB23 “user3” “pass3” like) (date 2910 2021)) 4))
(define FB25 (((login FB24 “user1” “pass1” like) (date 2910 2021)) 5))
;Like al comentario 12 asociado al post 0
```

```
(define FB26 (((login FB25 “user2” “pass2” like) (date 2910 2021)) 12))

;Testing función influencers (retorna los primeros 2 influencers

;Se usa la función score que usted debe proveer

(influencers FB26 scorer2)

;Testing función viral (en este caso solo debería retornar la publicación 3 que es la única compartida

(viral FB26 1)

;Testing función search

;Se usa la función compareFunction que usted debe proveer

(search FB26 “post” compareFunction)

;Testing función cluster

;Se usa la función compareFunction que usted debe proveer

(cluster FB26 compareFunction 3 0.3)

;Testing función botpost

;se usa función random provista en clases

(define FB27 (botpost FB26 “bot1” 3 “14th automatic post” (random 100)))

(define FB28 (botpost FB27 “bot2” 3 “15th automatic post” (random 100)))

;Testing functión detectBots, que debería arrojar bot1 y bot2

(detectBots FB28 1)
```

Laboratorio 2 (Paradigma lógico - Lenguaje Prolog)

Versión 1.0

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Fecha de Entrega: Ver calendario clase a clase donde se señala el hito

Objetivo del laboratorio: Aplicar conceptos del paradigma de programación lógico usando el lenguaje de programación Prolog en la resolución de un problema acotado.

Resultado esperado: Programa que simula las funcionalidades y comportamientos de una red social.

Profesor responsable: Víctor Flores (al hacer consultas en este documento, procurar hacer la mención a **@Víctor Flores Sánchez** (victor.floress@usach.cl) para que las notificaciones de sus consultas lleguen al profesor correspondiente)

Requerimientos No Funcionales obligatorios. Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales (obligatorios y extras) solicitados.

2. **(obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Prolog.

3. **(obligatorio) Versión:** Usar Swi-prolog versión 8.x.x .

4. **(1 pts) Documentación:** Todos los predicados deben estar debidamente comentados. Indicando descripción de la relación, términos de entrada y de salida.

5. **(2.5 pts) Historial:** Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas (no espere a terminar la materia para empezar a trabajar en el laboratorio. Puede ir haciendo pequeños incrementos conforme avance el curso)**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.375 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

6. **(obligatorio) Ejemplos:** Al final de su código incluir al menos 3 ejemplos de uso para cada uno de los predicados correspondientes a requerimientos funcionales obligatorios y los extra. **Solo se revisarán aquellas funcionalidades para las que existan los ejemplos provistos.**

7. **(obligatorio) Prerrequisitos:** Para cada predicado se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *login*, debe estar implementada la funcionalidad *create*.

Requerimientos Funcionales. **Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la función no será evaluada.**

1. **(0.5 pts) TDAs**. Implementar abstracciones apropiadas para el problema. Recomendamos leer el enunciado completo a fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno.

   Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Predicados de Pertenencia, Selectores, Modificadores y Otros predicados. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus predicados. En el resto de los predicados se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar los predicados necesarios dentro de esta estructura**.

   Dejar claramente documentado con comentarios en el código aquello que corresponde a la estructura base del TDA. Una estructura base que deberá considerar para el resto de los predicados corresponde a “socialnetwork” que corresponde al contenedor de sesión activa, usuarios, publicaciones, etc.

   Debe contar además con representaciones complementarias para que los usuarios puedan tener información relativa a sus credenciales de acceso y otros elementos que considere relevantes para abordar el problema.

   De igual forma, las publicaciones, además de información relativa a fecha de publicación, ID único incremental (se va generando un correlativo a partir de la última publicación registrada), debe contar con información relativa a reacciones, respuestas, contenido compartido, tipo de publicación (texto, video, url, foto, audio), y otros datos que considere relevante para abordar el problema.

Prerrequisitos para evaluación	Ninguno
Requisitos de implementación	- Usar estructuras basadas en listas - Especificar representación de manera clara para cada TDA implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, predicados de pertenencia, y según se requiera selectores, modificadores y otros predicados que pueda requerir para las otras funcionalidades requeridas a continuación.
Funcionalidades	*;Constructor socialnetwork*

mínimas requeridas	String X Date X OUT SocialNetwork *socialNetwork(Name, Date, SOut).* ;Date es una lista de: Integer X Integer X Integer *date(DD, MM, YYYY).*
Ejemplo de otros predicados asociados al mismo TDA u otros TDAs (Solo de referencia. Cada estudiante define sus propios TDAs)	*user(…. , UOut).* *publication(… , POut).*

2. **(0.5 pts) socialNetworkRegister:** Predicado que permite consultar el valor que toma un TDA *SocialNetwork* al ocurrir el registro de un nuevo usuario. Para esto se ingresa un *SocialNetwork* inicial, nombre del usuario (identificador único, se debe verificar que no exista para su correcto registro), contraseña y el *SocialNetwork* resultante luego del registro. El retorno del predicado es *true* en caso que se pudo satisfacer el registro del usuario.

Prerrequisitos para evaluación	Implementación de TDAs
Requisitos de implementación	- Debe funcionar cuando se entrega en su último argumento un TDA *SocialNetwork* como variable sin valor como también teniendo un valor. ● En el caso de ser una variable esta se unifica con el TDA *SocialNetwork* resultante luego de registrar al usuario. ● En caso de ser una variable con un valor concreto se verifica si el *SocialNetwork* cumple con ser el resultante luego de registrar al usuario. ● En caso que el usuario a registrar ya existe en el TDA “Sn1”, el predicado debe retornar *false*.
Dominio	socialNetwork X date X string X string X socialNetwork
Encabezado	*socialNetworkRegister(Sn1, Fecha, Username, Password, Sn2).*
Ejemplos	*socialNetworkRegister(S1, “2021-05-01”, “user”, “pass”, S2).*

3. **(0.5 pts) socialNetworkLogin:** Predicado que permite autenticar a un usuario registrado. Si la autenticación es válida (i.e., que el usuario existe y su contraseña es correcta). El retorno es *true*. La actualización del stack incorpora al usuario autenticado en sesión activa (fundamental para que los predicados siguientes puedan funcionar).

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkRegister*.
Requisitos de implementación	- Si “Sn2” es una variable, se debe unificar con el TDA *SocialNetwork* resultante luego de agregar una sesión activa con el usuario que realizó login. - Si el usuario y/o la contraseña son incorrectas se debe retornar *false*.
Dominio	socialNetwork X string X string X socialNetwork
Encabezado	*socialNetworkLogin(Sn1, Username, Password, Sn2).*
Ejemplos	*socialNetworkLogin(Sn1, “user”, “pass”, Sn2).*

4. **(0.5 pts) socialNetworkPost:** Predicado que permite a un usuario con sesión iniciada en la plataforma realizar una nueva publicación propia o dirigida a otros usuarios. Cada publicación registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación (tipo Date), el tipo de publicación (“photo”, “video”, “url”, “text”, “audio”) y el contenido de la publicación (solo debe ser un string). El retorno es *true* si se puede satisfacer en “Sn2” el TDA *SocialNetwork* con la nueva publicación incluida y sin la sesión activa del usuario que realizó la publicación.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin*
Requisitos de implementación	-Debe funcionar cuando se entrega en su último argumento “Sn2” un socialNetwork como variable sin valor como también teniendo un valor. ● En el caso de ser una variable esta se unifica con el TDA *SocialNetwork* resultante luego de agregar la publicación y eliminar la sesión del usuario activo.

	● En caso de tener un valor, se verifica si “Sn2 cumple con ser el resultante luego de haber realizado la publicación y sin la sesión del usuario activo. - En caso de no haber una sesión activa debe retornar *false*. - La fecha puede estar representada como lo estime conveniente, pero debe indicarlo en los ejemplos de uso. - Solo se puede publicar mensajes en cuentas de terceros si es que está dentro de la lista de contactos del usuario
Dominio	socialNetwork X date X string X string list x socialNetwork
Encabezado	*socialNetworkPost(Sn1, Fecha, Texto, ListaUsernamesDest, Sn2).*
Ejemplos	*socialNetworkPost(Sn1, “2021-04-30”, “hola a algunos”, [“u1” “u2 “u3”], Sn2).* *socialNetworkPost(Sn1, “2021-05-03”, “hola a todos”, [ ], Sn2).*

5. **(0.5 pts) socialNetworkFollow:** Predicado que permite a un usuario con sesión iniciada en la plataforma seguir a otro usuario. El retorno es *true* si se puede satisfacer en “Sn2” el TDA *SocialNetwork* con algún indicativo de que el usuario que tiene sesión iniciada en “Sn1”ahora sigue al usuario “U” y sin la sesión activa del usuario.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin*.
Requisitos de implementación	- Si no existe el “username” al cual se desea seguir, debe retornar *false*. - Si no existe sesión iniciada por algún usuario, debe retornar *false*. - Un usuario no se puede seguir a si mismo, en ese caso se retorna *false*.
Dominio	socialNetwork X string X socialNetwork
Encabezado	*socialNetworkFollow(Sn1, Username, Sn2).*

Ejemplos	*socialNetworkFollow(Sn1, “alguien”, Sn2).*

6. **(0.5 pts) socialNetworkShare:** Predicado que permite a un usuario con sesión iniciada en la plataforma compartir contenido de otro usuario en su propio espacio o dirigido a otros usuarios más. El retorno es *true* si se puede satisfacer en “Sn2” el TDA *SocialNetwork* con un registro de la publicación compartida y sin la sesión activa del usuario.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin* y socialNetworkPost.
Requisitos de implementación	- Procurar que el contenido compartido quede vinculado de alguna manera al usuario que creó originalmente la publicación lo cual se debe resolver internamente a partir de postID. - Solo se puede compartir con usuarios que estén dentro de la lista de contactos. Si no están se omiten.
Dominio	socialNetwork X date X integer X string list X socialNetwork
Encabezado	*socialNetworkShare(Sn1, Fecha, PostId, ListaUsernamesDest, Sn2).*
Ejemplos	*socialNetworkShare(Sn1, “2021-04-30”, 54, [“u1”, “u5”], Sn2). /* la publicación con ID 54 se comparte en la cuenta del usuario que tiene sesión iniciada en Sn1 hacia los usuarios “u1” y “u5”. En “Sn2” se encuentra la publicación compartida y no hay sesión iniciada. */* *socialNetworkShare(Sn1, “2021-04-30”, 54, [ ], Sn2). /*la publicación con ID 54 se comparte en la cuenta del usuario que tiene sesión iniciada en Sn1. En “Sn2” se encuentra la publicación compartida y no hay sesión iniciada. */*

7. **(0.5 pts) socialNetworkToString:** Predicado que permite obtener una representación de un TDA *socialNetwork* como un string posible de visualizar de forma comprensible al usuario. Debe hacer uso del *char* ‘\n’ para los saltos de línea. **No utilice los predicados *write* y *display* dentro de este requerimiento**, ya que debe quedar en el último argumento un string el cual pueda luego ser pasado como argumento a los predicados “write” o “display” para poder visualizarlo de forma comprensible al usuario.

Prerrequisitos para evaluación	Implementación de TDAs, predicados *socialNetworkRegister*, *socialNetworkPost*, *socialNetworkFollow* y *socialNetworkShare*.
Requisitos de implementación	- El string resultante debe quedar en el segundo argumento, al ser impreso por la función write o display debe ilustrar con claridad los elementos de *socialnetwork* (i.e., usuarios y sus publicaciones, sus redes de contactos, etc.). - El resultado de aplicar *socialnetworkToString* directamente sobre *socialnetwork* sin tener una sesión activa de por medio, imprime todo lo que haya en social network (ver anexo al final de este enunciado para un ejemplo completo). - El resultado de aplicar *socialnetworkToString* teniendo una sesión activa entrega como resultado todos los antecedentes del usuario que tiene la sesión activa (publicaciones, contactos, reacciones, publicaciones compartidas) junto a los detalles del usuario que inició sesión (nombre, fecha de creación de cuenta).
Dominio	*socialNetwork* X string
Encabezado	*socialnetworkToString(Sn1, StrOut).*
Ejemplos	*socialnetworkTostring(Sn1, Result).*

De los siguientes predicados implementar de manera libre los que estime conveniente para alcanzar un 7.0. Considere igual los prerrequisitos señalados. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.

16. **(1 pts) comment:** Predicado que permite a un usuario con sesión iniciada en la plataforma comentar publicaciones y otros comentarios existentes (propios o de terceros). El retorno es *true* si se puede satisfacer en “Sn2” el TDA *SocialNetwork* con un registro del comentario realizado y sin la sesión activa del usuario.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin* y socialNetworkPost, socialNetworkFollow, socialNetworkShare, socialNetworkToString
Requisitos de implementación	- Procurar que el comentario realizado quede vinculado de alguna manera al usuario que creó originalmente la

	publicación lo cual se debe resolver internamente a partir de postID. - Solo se puede compartir con usuarios que estén dentro de la lista de contactos. Si no están se retorna *false*. - CommentID podría ser 0, en tal caso significa que el comentario es directamente a una publicación, en cambio cuando *commentID* es distinto de 0 es porque se está comentando otro comentario (el cual igualmente es un comentario de *postID*). - Si se ingresa un *commentID* de un comentario que no fue realizado al *postID*, se debe retornar *false*.
Dominio	socialNetwork X date X integer X integer X string X socialNetwork
Encabezado	*comment (Sn1, Fecha, PostId, CommentId, TextoComentario, Sn2).*
Ejemplos	*comment(Sn1, "2021-05-04", 4, 0, "Este es un comentario sobre la publicación con id 4", Sn2).* *comment(Sn1, "2021-05-04", 4, 58, "Este es un comentario sobre la publicación con id 4, en particular sobre el comentario con id 58", Sn2).*

17. **(0.5 pts) socialNetworkLike:** Predicado que permite a un usuario con sesión iniciada en la plataforma reaccionar a publicaciones o comentarios existentes (propias o de terceros) a través de "me gusta". El retorno es *true* si se puede satisfacer en "Sn2" el TDA *SocialNetwork* con un registro del *like* realizado y sin la sesión activa del usuario.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin* y socialNetworkPost, socialNetworkFollow, socialNetworkShare, socialNetworkToString
Requisitos de implementación	- Procurar que el *like* realizado quede vinculado de alguna manera al usuario que creó originalmente la publicación lo cual se debe resolver internamente a partir de postID. - Se puede dar like a cualquier publicación, de cualquier usuario.

	- CommentID podría ser 0, en tal caso significa que el *like* es directamente a una publicación, en cambio cuando *commentID* es distinto de 0 es porque se está dando *like* a un comentario (el cual debe ser un comentario de *postID*). - Si se ingresa un *commentID* de un comentario que no fue realizado al *postID*, se debe retornar *false*.
Dominio	socialNetwork X date X integer X integer X socialNetwork
Encabezado	*socialNetworkLike (Sn1, Fecha, PostId, CommentId, Sn2).*
Ejemplos	*socialNetworkLike(Sn1, "2021-05-04", 4, 0, Sn2). ; Este es un like al post con id 4.* *socialNetworkLike(Sn1, "2021-05-04", 4, 58, Sn2). ; Este es un like al comentario con id 58 que está en el post con id 4.*

18. **(1 pts) socialNetworkViral:** Predicado que permite consultar si cierto listado de publicaciones corresponden a contenido viral o no, esto es, determinar si estas publicaciones han sido compartidas al menos K veces.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin* y socialNetworkPost, socialNetworkFollow, socialNetworkShare, socialNetworkToString, socialNetworkLike
Requisitos de implementación	- El predicado debe unificar la variable "postIds" con el listado de *ids* de publicaciones que tienen más de "*thresholdK" likes*. - Si se pasa como argumento en "*postIds*" un listado de ids, entonces se debe comprobar que todas esas publicaciones tienen más de "*thresholdK" likes*. - Basta con que un *postId* del listado no cumpla el criterio para que todo el predicado retorne *false*.
Dominio	socialNetwork X number X integer list
Encabezado	*socialNetworkViral(socialNetwork, thresholdK, postIds).*

Ejemplos	*socialNetworkViral(Sn1, 50, X). ; Entrega en la variable X un listado con los PostIds que se les ha dado like más de 50 veces.* *socialNetworkViral(Sn1, 50, [2, 4]). ; Comprueba si las publicaciones con id 2 y con id 4 se les ha dado like más de 50 veces.*

19. **(1 pts) socialNetworkSearch:** Predicado que permite consultar si cierto listado de publicaciones y cierto listado de comentarios contiene algún texto.

Prerrequisitos para evaluación	Implementación de TDAs, predicado *socialNetworkLogin* y socialNetworkPost, socialNetworkFollow, socialNetworkShare, socialNetworkToString
Requisitos de implementación	- Debe hacer uso del operador corte "!". - En PostIds y CommentIds si se pasa un listado de números, el predicado debe comprobar si todos esos *Ids* de publicaciones y todos esos ids de comentarios contienen el texto buscado. Si alguno de estos no contiene el texto entonces el predicado debe retornar *false*. - En PostIds y CommentIds si se pasa una variable entonces el predicado debe unificar en estas variables todos los *ids* de publicación e *ids* de comentario que contienen el texto buscado.
Dominio	socialNetwork X String X integer list
Encabezado	*socialNetworkSearch(socialNetwork, text, PostIds, CommentIds).*
Ejemplos	*socialNetworkSearch(Sn1, "hola", X, Z). ; Entrega en la variable X un listado con los PostIds que contienen el texto "hola" y en la variable Z el listado con los CommentIds que contienen el texto "hola".* *socialNetworkSearch(Sn1, "hola", [5, 13, 15], [2]). ; Comprueba si las publicaciones con id 5, con id 13 y con id 15 contienen el texto "hola". Y si el comentario con id 2 contiene el texto "hola".* *socialNetworkSearch(Sn1, "mundo", X, [3]). ; Un mix: Entrega en la variable X un listado con los PostIds que*

	contienen el texto “mundo” y comprueba si el comentario con id 3 contiene el texto “mundo”.

Script básico Pruebas Prolog

;testing constructor socialnetwork

; En los ejemplos asumo que tienen un constructor de “fecha” que en su último argumento entrega un TDA “Fecha”.

fecha(24, 5, 2021, F), socialNetwork(“failbok”, F, Sn1).

% Se crea una fecha y se deja en la variable F, luego se pasa esa fecha como argumento del constructor de socialNetwork, el TDA con la red social creada queda en la variable “Sn1”

;testing predicado socialNetworkRegister

fecha(24, 5, 2021, F), socialNetwork(“failbok”, F, Sn1), socialNetworkRegister(Sn1, “mmental”, “asdf”, Sn2).

% Sn2 es una variable con el TDA SocialNetwork y el usuario “mmental” creado.

%El siguiente ejemplo debe retornar false debido a que el usuario ya existe

fecha(24, 5, 2021, F), socialNetwork(“failbok”, F, Sn1), socialNetworkRegister(Sn1, “mmental”, “asdf”, Sn2), socialNetworkRegister(Sn2, “coni raf”, “vvbaa5”, Sn3), socialNetworkRegister(Sn3, “mmental”, “gggggg1234”, Sn4).

;testing predicado socialNetworkLogin y socialNetworkPost

% Este ejemplo es un post del usuario “mmental” destinado a todos

fecha(24, 5, 2021, F), socialNetwork(“failbok”, F, Sn1), socialNetworkRegister(Sn1, “mmental”, “asdf”, Sn2), socialNetworkRegister(Sn2, “coni raf”, “vvbaa5”, Sn3), socialNetworkRegister(Sn3, “jperez”, “jjjj98765”, Sn4), fecha(25, 5, 2021, F2), socialNetworkLogin(Sn4, “mmental”, “asdf”, Sn5), socialNetworkPost(Sn5, F2, “este es un mensaje destinado a todos”, [ ], Sn6).

% Este ejemplo es un post del usuario “coni raf” al usuario “mmental”

fecha(24, 5, 2021, F), socialNetwork(“failbok”, F, Sn1), socialNetworkRegister(Sn1, “mmental”, “asdf”, Sn2), socialNetworkRegister(Sn2, “coni raf”, “vvbaa5”, Sn3), socialNetworkRegister(Sn3, “jperez”, “jjjj98765”, Sn4), fecha(25, 5, 2021, F2), socialNetworkLogin(Sn4, “coni raf”, “vvbaa5”, Sn5), socialNetworkPost(Sn5, F2, “este es una publicación para mmental”, [“mmental”], Sn6).

% Este ejemplo es un login incorrecto de “coni raf”, debe retornar *false*

fecha(24, 5, 2021, F), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5", Sn3), socialNetworkLogin(Sn3, "coni raf", "1234", Sn4).

% Este es un ejemplo de login en un TDA socialNetwork con ya una sesión activa, debe retornar *false*

fecha(24, 5, 2021, F), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5", Sn3), socialNetworkLogin(Sn3, "coni raf", "vvbaa5", Sn4), socialNetworkLogin(Sn4, "mmental", "asdf", Sn5).

% Este es un ejemplo de socialNetworkPost en un TDA sin sesión activa, debe retornar *false*

fecha(24, 5, 2021, F), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5", Sn3), fecha(25, 5, 2021, F2), socialNetworkPost(Sn3, F2, "buenas vacaciones", [ ], Sn4).

;testing predicado socialNetworkLogin y socialNetworkFollow

;Se asume que de aquí en adelante comienza la publicación con ID 0 y luego se incrementan los ID en 1

% Ejemplo en que "coni raf" sigue a "jperez"

fecha(24, 5, 2021, F), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5", Sn3), socialNetworkRegister(Sn3, "jperez", "jjjj98765", Sn4), fecha(25, 5, 2021, F2), socialNetworkLogin(Sn4, "coni raf", "vvbaa5", Sn5), socialNetworkFollow(Sn5, "jperez", Sn6).

% Ejemplo en que "mmental" se intenta seguir a si mismo, debe retornar *false*

fecha(24, 5, 2021, F), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5", Sn3), socialNetworkRegister(Sn3, "jperez", "jjjj98765", Sn4), fecha(25, 5, 2021, F2), socialNetworkLogin(Sn4, "mmental", "asdf", Sn5), socialNetworkFollow(Sn5, "mmental", Sn6).

;testing predicado socialNetworkShare

% Ejemplo en que "coni raf" comparte con el usuario "jperez" la publicación con id 0 de "mmental", previamente "coni raf" debe seguir a "jperez" para poder compartirle la publicación.

fecha(24, 5, 2021, F), fecha(25, 5, 2021, F2), fecha(1, 6, 2021, F3), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5", Sn3), socialNetworkRegister(Sn3, "jperez", "jjjj98765", Sn4), socialNetworkLogin(Sn4, "mmental", "asdf", Sn5), socialNetworkPost(Sn5, F2, "este es un mensaje destinado a todos", [ ], Sn6), socialNetworkLogin(Sn6, "coni raf", "vvbaa5", Sn7), socialNetworkFollow(Sn7, "jperez", Sn8), socialNetworkLogin(Sn8, "coni raf", "vvbaa5", Sn9), socialNetworkShare(Sn9, F3, 0, ["jperez"], Sn10).

;testing socialnetworkToString, con login y sin login, en Str1 está la representación sin login y en Str2 está la representación con login, es decir debe mostrar sólo lo relacionado al usuario "coni raf".

fecha(24, 5, 2021, F), fecha(25, 5, 2021, F2), fecha(1, 6, 2021, F3), socialNetwork("failbok", F, Sn1), socialNetworkRegister(Sn1, "mmental", "asdf", Sn2), socialNetworkRegister(Sn2, "coni raf", "vvbaa5",

Sn3), socialNetworkRegister(Sn3, “jperez”, “jjjj98765”, Sn4), socialNetworkLogin(Sn4, “mmental”, “asdf”, Sn5), socialNetworkPost(Sn5, F2, “este es un mensaje destinado a todos”, [ ], Sn6), socialNetworkLogin(Sn6, “coni raf”, “vvbaa5”, Sn7), socialNetworkFollow(Sn7, “jperez”, Sn8), socialNetworkLogin(Sn8, “coni raf”, “vvbaa5”, Sn9), socialNetworkShare(Sn9, F3, 0, [“jperez”], Sn10), socialNetworkLogin(Sn10, “coni raf”, “vvbaa5”, Sn11), socialNetworkToString(Sn10, Str1), socialNetworkToString(Sn11, Str2), display(Str1), display(Str2).

% Recordar que en Str1 y Str2 hay saltos de línea (carácter ‘\n’), mientras que sólo cuando se consulte al predicado “display” se mostrará de forma legible el TDA.

% Ejemplo de lo que mostraría display de “Str1”:

####### Red social “Failbok” #######
Creada el día 24/05/2021
*** Usuarios registrados ***
mmental
	Sigue a:
coni raf
	Sigue a: jperez
jperez
	Sigue a:

*** Publicaciones ***
ID: 0
El día 25/05/2021 “mmental” publicó:
	“este es un mensaje destinado a todos”
Destinatarios: Todos
Compartido:
	El día 1/6/2021 por “Coni raf” hacia: “jperez”
*** Fin publicaciones ***

% Ejemplo de lo que mostraría display de “Str2”:

####### Red social “Failbok” #######
Creada el día 24/05/2021
*** Usuario con sesión iniciada ***
coni raf
	Sigue a: jperez

*** Publicaciones ***
Sin publicaciones
*** Fin publicaciones ***

*** Publicaciones compartidas ***
El día 1/6/2021 compartió con “jperez” la publicación con id 0 de “mmental”: “este es un mensaje destinado a todos”.
*** Fin publicaciones compartidas ***

Laboratorio 3
(Paradigma Orientado a Objetos - Lenguaje Java)

Versión 1.1 - Última Edición: 23/07/2021

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Fecha de Entrega Diurno: 12/08/2021 (12 de agosto, 2021)

Objetivo del laboratorio: Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma orientado a objetos usando el lenguaje de programación Java.

Resultado esperado: Programa que simula las funcionalidades y comportamientos de una red social.

Profesor responsable: **Daniel Gacitúa** (al hacer consultas en este documento, procurar hacer la mención a **@Daniel Gacitúa Vásquez** (daniel.gacitua@usach.cl) para que las notificaciones de sus consultas lleguen al profesor correspondiente)

Requerimientos No Funcionales obligatorios. Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados.

2. **(obligatorio) Lenguaje y herramientas de trabajo:** La implementación de este laboratorio debe ser en el lenguaje de programación Java, utilizando OpenJDK versión 11 (link) con alguno de los siguientes IDEs (a su elección):
   a. Apache NetBeans 12 (link)
   b. Eclipse IDE 2020-09 (link)
   c. IntelliJ IDEA Community 2020.3 (link)

   Para el proceso de compilación, se puede elegir entre una de las siguientes herramientas:
   1. Un *script de compilación manual* en Batch (Windows) o Bash (GNU/Linux, macOS o Windows Subsystem for Linux (WSL)) que utilice el comando `javac` para compilar sus clases. El entregable debe ser una carpeta con el código fuente y el script de compilación (no debe incluir archivos binarios ni archivos .class)
   2. Integrar a su proyecto la herramienta *Gradle* (link a guía para crear proyecto Java usando Gradle). El entregable final debe ser un proyecto creado con el IDE de su elección, integrado con Gradle (y Gradle Wrapper) e incluyendo solo código fuente y archivos de configuración (no debe incluir archivos binarios ni archivos .class)

   Independientemente de la opción escogida para compilar el proyecto, su Informe debe especificar las instrucciones de compilación, la herramienta de compilación usada (script manual o Gradle) y el ambiente de desarrollo donde se ejecutó el entregable (Sistema Operativo y versión exacta de JDK). En caso de no poder compilar y ejecutar su proyecto con las instrucciones que usted indique, el laboratorio no será revisado.

3. **(obligatorio) Interacciones con el programa:** Todas las interacciones con el programa deben ser mediante consola/terminal. Puede recurrir al uso de System.in y System.out (en Java). Para sus funcionalidades, solo se permite el uso de la biblioteca estándar de Java (sin tener dependencias externas, como las disponibles en formato JAR).

4. **(obligatorio) Uso del paradigma:** Su solución debe demostrar la aplicación del paradigma orientado a objetos. No basta con que su solución esté implementada en Java. Su diseño y correspondiente implementación debe seguir los lineamientos del paradigma Orientado a Objetos.

5. **(obligatorio) Prerrequisitos:** Para cada funcionalidad se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *ask*, debe estar implementada la funcionalidad *register/login/logout*.

6. **(1 pto) Documentación:** Se debe documentar el código indicando una breve descripción de las clases creadas, sus atributos, métodos públicos y relaciones. Procure utilizar comentarios tipo Javadoc (link) para esto.

7. **(1 pto) Organización del código:** Se debe cuidar la organización del código (orden y claridad). Procure que su diseño de clases no viole los principios de bajo acoplamiento y alta cohesión.

8. **(1.5 pto) Diagrama de análisis:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML a nivel de análisis que describa las entidades y relaciones del problema abordado. Este diagrama se debe crear antes del proceso de desarrollo.

9. **(1.5 pto) Diagrama de diseño:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML tras la implementación de la solución, este diagrama debe ser coherente con la implementación en código de su solución incluyendo todas las clases de su código. Este diagrama se debe crear después del desarrollo de la solución.

10. **(1 pto) Uso de git:** Historial de trabajo en GitHub tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.25 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

Requerimientos Funcionales. **Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la funcionalidad no será evaluada.**

1. **(1 pto) Clases e interfaces que forman el programa:** Como parte del diseño orientado a objetos de su solución, considere como mínimo modelar las siguientes entidades (y sus respectivas relaciones) dentro de su programa:
    a. Usuario: Clase que representa al usuario en la red social. Algunas características a considerar en su implementación son: ID (único y autoincremental), nombre de usuario (único), contraseña y lista de publicaciones.
    b. Publicación: Clase que representa las publicaciones dentro de la aplicación. Algunas características a considerar son:
        i. ID (único y autoincremental)
        ii. Contenido
        iii. Tipo de publicación
        iv. Fecha de publicación
        v. Autor (usuario)
    c. Reacción: Clase que representa las respuestas a publicaciones dentro de la aplicación. Considere incluir un ID (único y autoincremental), Autor, Fecha de publicación, Contenido y Tipo (Like, Dislike, Respuesta, etc.).
    d. Red Social: Interfaz que especifica todos los comportamientos de una red social la cual es implementada a través de una clase. Esta clase en sus atributos agrupa instancias de las clases anteriores a través de estructuras de datos adecuadas (ej: listas, vectores, arreglos, etc.)

Prerrequisitos para evaluación	Ninguno
Requisitos de implementación	- Usar estructuras basadas en clases - Especificar representación de manera clara para cada clase implementada (en el informe y en el código a través de comentarios). Luego implementar constructores, getters, setters y otros métodos según lo requerido en los requisitos a continuación.

2. **(1 pto) Menú interactivo por terminal:** Su programa debe incluir un menú por terminal/consola que permita la interacción del usuario con la solución, implementando las entidades y funcionalidades que permitan utilizar su simulación de una red social. El sistema debe estar poblado con algunos elementos al iniciar y además debe proveer de retroalimentación al usuario de las acciones realizadas por él (si la acción pudo concretarse con éxito o no). Las instrucciones de funcionamiento de su solución deben estar documentadas en su Informe de Laboratorio.

   Un ejemplo del menú (con un usuario registrado) se representa a continuación:

```
### RED SOCIAL: FailBook ###
## Registrado como: User123 ##
Escoja su opción:
1. Realizar publicación
2. Seguir a un usuario
3. Compartir publicación
4. Visualizar red social
5. Cerrar sesión
6. Salir del programa
INTRODUZCA SU OPCIÓN: _
```

Prerrequisitos para evaluación	- Clases y estructuras que forman el programa
Requisitos de implementación	- Implementar sistema de menú basado en terminal/consola - El menú puede variar según la cantidad de requisitos funcionales implementados durante la realización del laboratorio. - La aplicación debe permitir cargar de forma inicial al menos 1 Red Social, 5 usuarios registrados, 10 publicaciones, 3 follows y 1 follow mutuo.

Las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser ejecutadas mediante uno o más métodos dentro de su programa.

3. **(0.5 pto) authentication:** Esta funcionalidad debe permitir el registro de nuevos usuarios en la plataforma (*Register*), además de su ingreso (*Login*) y su salida (*Logout*) de la plataforma con sus credenciales.

Prerrequisitos para evaluación	- Clases y estructuras que forman el programa - Menú interactivo por terminal
Requisitos de implementación	- Register: Se debe almacenar el nuevo usuario en la red social, almacenando su fecha de registro y un identificador único - Login: El programa debe validar las credenciales del usuario en la red social antes de permitir el ingreso. - Logout: Debe existir una sesión iniciada previamente en la red social para salir.
Parámetros de entrada	Nombre de usuario y contraseña (*register* y *login*)

4. **(0.5 pto) post:** Funcionalidad que permite a un usuario con sesión iniciada en la red social realizar una nueva publicación propia o dirigida a otros usuarios. Cada publicación debe almacenar el ID (único y autoincremental), el autor de la misma (obtenido desde la sesión iniciada con *authentication*), fecha de publicación (tipo Date), el tipo de publicación (“photo”, “video”, “url”, “text”, “audio”) y el contenido de la publicación (solo debe ser un string).

Prerrequisitos para evaluación	- authentication
Requisitos de implementación	- Se debe almacenar la nueva publicación en la red social, rellenando los atributos asociados a esta al momento de crear la pregunta (cómo el ID, Fecha de publicación, Autor, Tipo y Contenido). - Solo los usuarios con sesión iniciada pueden crear publicaciones en la red social.
Parámetros de entrada	Tipo de publicación, Contenido y Lista de usuarios al cual va dirigida la publicación (solo si es necesario)

5. **(0.5 pto) follow:** Funcionalidad que permite a un usuario (con sesión iniciada) poder seguir a otro usuario.

Prerrequisitos para evaluación	- authentication
Requisitos de implementación	- Esta funcionalidad debe permitir seguir a otro usuario por su nombre de usuario. - Los *follow* son unidireccionales, pero dos usuarios se pueden seguir mutuamente. - Solo los usuarios con sesión iniciada pueden seguir a otros usuarios. Un usuario no se puede seguir a si mismo.
Parámetros de entrada	Usuario a seguir

6. **(0.5 pto) share:** Funcionalidad que permite a un usuario (con sesión iniciada) compartir contenido de un usuario en su propio espacio o dirigido a otros usuarios.

Prerrequisitos para evaluación	- authentication - post - follow
Requisitos de implementación	- Se puede compartir publicaciones en la lista de publicaciones propia del usuario, o en la lista de publicaciones de usuarios que sigue (puede ser uno o varios). - Procurar que el contenido compartido quede vinculado de alguna manera al usuario que creó originalmente la publicación. - Debe quedar registrada la fecha de la compartición y el ID del post.
Parámetros de entrada	ID del post a compartir, Lista de nombres de usuario con quienes compartir

7. **(0.5 pto) visualize:** Funcionalidad que permite a obtener una representación visual y comprensible por el usuario de la red social. Esta funcionalidad está dividida en dos partes: Transformar la red social a un string (*SocialNetworkToString*) e imprimir el string recién generado (*PrintSocialNetwork*).

Prerrequisitos para evaluación	- authentication - post - follow - share
Requisitos de implementación	- SocialNetworkToString: Esta funcionalidad puede ser ejecutada con sesión iniciada o sin ella. En caso de ejecutarla con sesión iniciada debe entregar todos los elementos del usuario con sesión activa (publicaciones, lista de follows, reacciones, publicaciones compartidas, etc.) junto con los detalles del usuario (nombre de usuario y fecha de creación de cuenta. En caso de ejecutarla sin sesión iniciada, debe entregar todo lo que haya en la red social (usuarios registrados, listas de follows, publicaciones, etc.). - PrintSocialNetwork: Debe imprimir en pantalla el string obtenido de SocialNetworkToString en un formato comprensible por el usuario. Evite imprimir datos sensibles (como contraseñas).
Parámetros de entrada	<ninguno>

De los siguientes predicados implementar de manera libre los que estime conveniente para alcanzar un 7.0. Considere igual los prerrequisitos señalados. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.

1. **(1 pto) comment:** Funcionalidad que permite a un usuario con sesión iniciada en la plataforma comentar publicaciones y otros comentarios (propios o de terceros).

Prerrequisitos para evaluación	- authentication - post
Requisitos de implementación	- Se puede comentar tanto publicaciones como otros comentarios. Los comentarios son siempre texto. - Se debe registrar la fecha en que se realizó el comentario. Los comentarios deben poseer un ID (único y autoincremental). - Procurar que el comentario realizado quede vinculado de alguna manera al usuario que creó originalmente la publicación.
Parámetros de entrada	Publicación o comentario seleccionado, Texto comentario

2. **(0.5 pto) like:** Funcionalidad que permite a un usuario con sesión iniciada reaccionar a publicaciones o comentarios (propios o de terceros) a través de un “me gusta”.

Prerrequisitos para evaluación	- authentication - post
Requisitos de implementación	- Se puede dar like a cualquier publicación (o comentario), de cualquier usuario. - Se debe registrar la fecha en que se realizó el *like*. Los *likes* deben poseer un ID (único y autoincremental). - Procurar que el *like* realizado quede vinculado de alguna manera al usuario que creó originalmente la publicación.
Parámetros de entrada	Publicación o comentario seleccionado.

3. **(1 pto) isViral:** Funcionalidad que permite consultar si cierto listado de publicaciones corresponden a contenido viral o no, esto es, determinar si estas publicaciones han sido compartidas al menos K veces.

Prerrequisitos para evaluación	- post - share - visualize
Requisitos de implementación	- La funcionalidad debe recibir una lista de publicaciones (mediante su ID) y debe retornar cuales de esas publicaciones han sido compartidas al menos K veces. - No es necesario tener sesión iniciada para usar la funcionalidad isViral.
Parámetros de entrada	Red social seleccionada, Lista de publicaciones, K
Parámetros de salida	Lista de publicaciones que complan con el criterio K

4. **(2 pto) search:** Funcionalidad que permite buscar publicaciones en base a una coincidencia parcial de texto. Se debe identificar el tipo de búsqueda a realizar (si se hará por cualquier texto dentro de las publicaciones o comentarios). La función se puede usar directamente sin estar registrado.

Prerrequisitos para evaluación	- post - visualize
Requisitos de implementación	- La funcionalidad debe permitir buscar por publicaciones (o comentarios) dentro de la red social. - La funcionalidad debe permitir coincidencias parciales de texto (substring) y debe ser insensible a las mayúsculas. - La funcionalidad debe retornar la publicación completa que contiene la coincidencia. - No es necesario tener sesión iniciada para usar la funcionalidad search.
Parámetros de entrada	Red social seleccionada, criterio de búsqueda (publicaciones o comentarios), string a buscar.

Parámetros de salida	Publicaciones que coincidan con los criterios de búsqueda.

5. **(1 pto) botPost:** Funcionalidad que permite automatizar la generación de publicaciones en la red social a través de un bot.

Prerrequisitos para evaluación	- post - share - follow
Requisitos de implementación	- La funcionalidad crea un nuevo usuario, añade a su lista de follow a K usuarios aleatorios (por ID) y comparte con estos una publicación. Si K es mayor que el número de usuarios existentes en la red social, entonces se añaden a todos los usuarios. - Puede utilizar el método nextInt de la biblioteca estándar de Java (java.util.Random) para generar números aleatorios. - No es necesario tener sesión iniciada para usar la funcionalidad botPost.
Parámetros de entrada	Nombre del bot, Número de seguidos (K), Contenido de la publicación y Tipo de la publicación

Laboratorio 4 (Multiparadigma - Lenguaje Java o C#)

Versión 1.0 - Última Edición: 22/07/2021

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Fecha de Entrega Vespertino: 9/08/2021 (9 de agosto del 2021)

Fecha de Entrega Diurno: 23/08/2021 (23 de agosto del 2021)

Notas:

1. **La revisión se realizará de forma presencial (remotamente) durante la clase posterior al día de la entrega. Usted deberá mostrar desde su computador la ejecución de su entrega.**
    a. **El día de la presentación usted debe presentar al profesor:**
        i. **Su entrega funcionando**
        ii. **Casos de uso de la entrega. Es decir, evidenciar el funcionamiento de los RF implementados.**
        iii. **Diagramas de componente y clases.**
2. **Si desea utilizar C#, este solo puede ser ejecutado en máquinas con el sistema operativo Windows. Si desea utilizar otro sistema operativo, debe realizar su entrega en Java.**

Objetivo del presente laboratorio: Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma Orientado a Objetos (OOP) y dirigido por eventos (Event-Driven Programming) usando el lenguaje de programación C# o Java.

Resultado esperado: Programa con interfaz gráfica (GUI) implementado en C# o Java que simula las funcionalidades y comportamientos de un sistema de red social como Facebook.

Profesor responsable: **Gonzalo Martinez** (al hacer consultas en este documento, procurar hacer la mención a **@Gonzalo Martinez** (gonzalo.martinez@usach.cl) para que las notificaciones de sus consultas lleguen al profesor correspondiente)

Requerimientos No Funcionales obligatorios. Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados.

2. **(obligatorio) Lenguaje y herramientas de trabajo:** En este laboratorio, el lenguaje de programación a ocupar queda a elección del estudiante. No obstante, los lenguajes a seleccionar pueden ser **C#** o **Java**. Si implementa su solución en otro lenguaje, será calificado con la nota mínima.

   a. En el caso de utilizar C#:
   - La implementación de este laboratorio debe ser en el lenguaje de programación **C# en su versión 8.0 o superior** (link) con alguno de los siguientes IDEs (a su elección):

   a. Visual Studio Community (gratuito) (link)
   b. Visual Studio Ultimate (en caso de disponer de esta) (link)

   - La implementación debe utilizar **WPF** o **Windows Form** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de C#, por lo que no se debe agregar dependencias externas al proyecto.
   - WPF o Windows Form **solo puede ser implementado y ejecutado en sistemas operativos Windows. No es compatible con Linux ni con MacOS. En caso de querer utilizar estas plataformas, optar por una implementación basada en Java**.
   - Su entregable final debe ser el proyecto generado por el IDE, sin incluir archivos binarios.

   b. En el caso de utilizar Java:
   - La implementación de este laboratorio debe ser en el lenguaje de programación Java, utilizando **OpenJDK versión 11** (link) con alguno de los siguientes IDEs (a su elección):

   a. Apache NetBeans 12 (link)
   b. Eclipse IDE 2020-09 (link)
   c. IntelliJ IDEA Community 2020.3 (link)

   - La implementación debe utilizar **AWT** y/o **Swing** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de Java, por lo que no se debe agregar dependencias externas al proyecto.

Cabe destacar que la evaluación será presencial, por lo que el aspecto del cómo compilar y ejecutar su proyecto queda a disposición del estudiante. Si el estudiante decide ejecutar todo por su IDE es válido, no habrá penalización en este punto. Lo importante es que al momento de la evaluación presencial el estudiante ejecute su entrega para que el profesor pueda evaluar en tiempo de ejecución.

3. **(obligatorio) Interacciones con el programa:** Todas las interacciones con el programa deben ser mediante el **uso de la interfaz gráfica**. Dependiendo de su lenguaje escogido, su implementación debe utilizar las librerías gráficas previamente nombradas. En Java, para sus funcionalidades, solo se permite el uso de la biblioteca estándar de Java (sin tener dependencias externas, como las disponibles en formato JAR). De igual forma, en C# debe utilizar las funciones y librerías nativas de dicho lenguaje.

4. **(obligatorio) Uso del paradigma:** Su solución debe demostrar la aplicación del **Paradigma Orientado a Objetos y Dirigido por Eventos**.
    a. No basta con que su solución esté implementada en Java o en C#. Su diseño y correspondiente implementación debe seguir los lineamientos de dichos paradigmas.

5. **(obligatorio) Separación responsabilidad Modelo-Vista**: Es obligatorio separar las capas de modelo y vista. **Los objetos del modelo no deben tener ningún grado de acoplamiento con respecto a los objetos de la vista**. Solo se permitirá que la vista tenga dependencia con objetos del modelo.
    - Los objetos del modelo no pueden realizar modificaciones en la vista o realizar llamados a métodos de las mismas. Por otro lado, los objetos de la vista, si pueden operar con objetos del modelo. Finalmente, los objetos que sean implementados con respecto a la interfaz gráfica de usuario (GUI) solo deben manejar las interacciones del usuario.

6. **(obligatorio) Prerrequisitos:** Para cada funcionalidad se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada.

7. **(1 pto) Documentación:** Se debe documentar el código indicando una breve descripción de las clases creadas, sus atributos, métodos públicos y relaciones. Procure utilizar comentarios tipo Javadoc (link) para esto. Si utiliza C# debe utilizar los comentarios relativos a esta tecnología.

8. **(1 pto) Organización del código:** Se debe cuidar la organización del código (orden y claridad). Procure que su diseño de clases no viole los principios de bajo acoplamiento y alta cohesión.

9. **(1.5 pto) Diagrama de análisis:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML a nivel de análisis que describa las entidades y relaciones del problema abordado. Este diagrama se debe crear antes del proceso de desarrollo.

10. **(1.5 pto) Diagrama de diseño:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML tras la implementación de la solución, este diagrama debe ser coherente con la implementación en código de su solución

incluyendo todas las clases de su código. Este diagrama se debe crear después del desarrollo de la solución.

11. **(1 pto) Uso de git:** Historial de trabajo en GitHub tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas.

    - Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 1 semana**.
    - Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios.
    - A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.25 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

Requerimientos Funcionales. **Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la funcionalidad no será evaluada.**

1. **(1 pto) Clases y estructuras que forman el programa:** Como parte del diseño orientado a objetos de su solución, considere como mínimo modelar las siguientes entidades (y sus respectivas relaciones) dentro de su programa:

   Puede tomar como referencia su implementación del laboratorio 3.

   a. Usuario: Representa al usuario en la red social. Algunas características a considerar en su implementación son: ID (único y autoincremental), nombre de usuario (único), contraseña y lista de publicaciones.
   b. Publicación: Representa las publicaciones dentro de la aplicación. Algunas características a considerar son:
      i. ID (único y autoincremental)
      ii. Contenido
      iii. Tipo de publicación
      iv. Fecha de publicación
      v. Autor (usuario)
   c. Reacción: Representa las respuestas a publicaciones dentro de la aplicación. Considere incluir un ID (único y autoincremental), Autor, Fecha de publicación, Contenido y Tipo (Like, Dislike, Respuesta, etc.).
   d. Red Social: Agrupa las estructuras de datos necesarias para una red social.

Prerrequisitos para evaluación	Ninguno
Requisitos de implementación	- Usar estructuras basadas en clases - Especificar representación de manera clara para cada clase implementada (en el informe y en el código a través de comentarios). Luego implementar constructores, getters, setters y otros métodos según lo requerido en los requisitos a continuación.

Las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser ejecutadas mediante uno o más métodos dentro de su programa.

Al igual que el laboratorio 3, las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser ejecutadas mediante uno o más métodos dentro de su programa. Puede utilizar como referencia su implementación del laboratorio pasado.

El funcionamiento de estos RF es similar al laboratorio 3. No obstante, acá usted debe implementar una interfaz gráfica que permita la interacción con el usuario.

2. **(0.7 pto) Authentication:** Esta funcionalidad debe permitir el registro de nuevos usuarios en la plataforma (*Register*), además de su ingreso (*Login*) y su salida (*Logout*) de la plataforma con sus credenciales.

Prerrequisitos para evaluación	- Clases y estructuras que forman el programa - Menú interactivo por terminal
Requisitos de implementación	- Register: Se debe almacenar el nuevo usuario en la red social, almacenando su fecha de registro y un identificador único - Login: El programa debe validar las credenciales del usuario en la red social antes de permitir el ingreso. - Logout: Debe existir una sesión iniciada previamente en la red social para salir.
Parámetros de entrada	Nombre de usuario y contraseña (*register* y *login*)

3. **(0.7 pto) post:** Funcionalidad que permite a un usuario con sesión iniciada en la red social realizar una nueva publicación propia o dirigida a otros usuarios. Cada publicación debe almacenar el ID (único y autoincremental), el autor de la misma (obtenido desde la sesión iniciada con *authentication*), fecha de publicación (tipo Date), el tipo de publicación (“photo”, “video”, “url”, “text”, “audio”) y el contenido de la publicación (solo debe ser un string).

Prerrequisitos para evaluación	- authentication
Requisitos de implementación	- Se debe almacenar la nueva publicación en la red social, rellenando los atributos asociados a esta al momento de

	crear la pregunta (cómo el ID, Fecha de publicación, Autor, Tipo y Contenido). - Solo los usuarios con sesión iniciada pueden crear publicaciones en la red social.
Parámetros de entrada	Tipo de publicación y Contenido

4. **(0.7 pto) follow:** Funcionalidad que permite a un usuario (con sesión iniciada) poder seguir a otro usuario.

Prerrequisitos para evaluación	- authentication
Requisitos de implementación	- Esta funcionalidad debe permitir seguir a otro usuario por su nombre de usuario. - Los *follow* son unidireccionales, pero dos usuarios se pueden seguir mutuamente. - Solo los usuarios con sesión iniciada pueden seguir a otros usuarios. Un usuario no se puede seguir a si mismo.
Parámetros de entrada	Usuario a seguir

5. **(0.7 pto) share:** Funcionalidad que permite a un usuario (con sesión iniciada) compartir contenido de un usuario en su propio espacio o dirigido a otros usuarios.

Prerrequisitos para evaluación	- authentication - post - follow
Requisitos de implementación	- Se puede compartir publicaciones en la lista de publicaciones propia del usuario, o en la lista de publicaciones de usuarios que sigue (puede ser uno o varios). - Procurar que el contenido compartido quede vinculado de alguna manera al usuario que creó originalmente la publicación.

	- Debe quedar registrada la fecha de la compartición y el ID del post.
Parámetros de entrada	ID del post a compartir, Lista de nombres de usuario con quienes compartir

De los siguientes predicados implementar de manera libre los que estime conveniente para alcanzar un 7.0. Considere igual los prerrequisitos señalados. La nota máxima en total será un 7.0 aun cuando se desborde el puntaje.

Recuerde centrarse en la implementación de la interfaz gráfica

1. **(1 pto) comment:** Funcionalidad que permite a un usuario con sesión iniciada en la plataforma comentar publicaciones y otros comentarios (propios o de terceros).

Prerrequisitos para evaluación	- authentication - post
Requisitos de implementación	- Se puede comentar tanto publicaciones como otros comentarios. Los comentarios son siempre texto. - Se debe registrar la fecha en que se realizó el comentario. Los comentarios deben poseer un ID (único y autoincremental). - Procurar que el comentario realizado quede vinculado de alguna manera al usuario que creó originalmente la publicación.
Parámetros de entrada	Publicación o comentario seleccionado, Texto comentario

2. **(0.5 pto) like:** Funcionalidad que permite a un usuario con sesión iniciada reaccionar a publicaciones o comentarios (propios o de terceros) a través de un “me gusta”.

Prerrequisitos para evaluación	- authentication - post
Requisitos de implementación	- Se puede dar like a cualquier publicación (o comentario), de cualquier usuario.

	- Se debe registrar la fecha en que se realizó el *like*. Los *likes* deben poseer un ID (único y autoincremental). - Procurar que el *like* realizado quede vinculado de alguna manera al usuario que creó originalmente la publicación.
Parámetros de entrada	Publicación o comentario seleccionado.

3. **(1 pto) isViral:** Funcionalidad que permite consultar si cierto listado de publicaciones corresponden a contenido viral o no, esto es, determinar si estas publicaciones han sido compartidas al menos K veces.

Prerrequisitos para evaluación	- post - share - visualize
Requisitos de implementación	- La funcionalidad debe recibir una lista de publicaciones (mediante su ID) y debe retornar cuales de esas publicaciones han sido compartidas al menos K veces. - No es necesario tener sesión iniciada para usar la funcionalidad isViral.
Parámetros de entrada	Red social seleccionada, Lista de publicaciones, K
Parámetros de salida	Lista de publicaciones que complan con el criterio K

4. **(2 pto) search:** Funcionalidad que permite buscar publicaciones en base a una coincidencia parcial de texto. Se debe identificar el tipo de búsqueda a realizar (si se hará por cualquier texto dentro de las publicaciones o comentarios). La función se puede usar directamente sin estar registrado.

Prerrequisitos para evaluación	- post - visualize
Requisitos de implementación	- La funcionalidad debe permitir ordenar buscar por publicaciones (o comentarios) dentro de la red social.

	- La funcionalidad debe permitir coincidencias parciales de texto (substring) y debe ser insensible a las mayúsculas. - La funcionalidad debe retornar la publicación completa que contiene la coincidencia. - No es necesario tener sesión iniciada para usar la funcionalidad search.
Parámetros de entrada	Red social seleccionada, criterio de búsqueda (publicaciones o comentarios), string a buscar.
Parámetros de salida	Publicaciones que coincidan con los criterios de búsqueda.

5. **(1 pto) botPost:** Funcionalidad que permite automatizar la generación de publicaciones en la red social a través de un bot.

Prerrequisitos para evaluación	- post - share - follow
Requisitos de implementación	- La funcionalidad crea un nuevo usuario, añade a su lista de follow a K usuarios aleatorios (por ID) y comparte con estos una publicación. Si K es mayor que el número de usuarios existentes en la red social, entonces se añaden a todos los usuarios. - Puede utilizar el método nextInt de la biblioteca estándar de Java (java.util.Random) para generar números aleatorios. - No es necesario tener sesión iniciada para usar la funcionalidad botPost.
Parámetros de entrada	Nombre del bot, Número de seguidos (K), Contenido de la publicación y Tipo de la publicación