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 3/10/2020

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 foros similar a Stack Overflow.

El software en su versión final (Laboratorio N°4) deberá tener una interfaz gráfica que permita una interacción similar a la presentada en el sistema de preguntas y respuestas stackoverflow.

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.

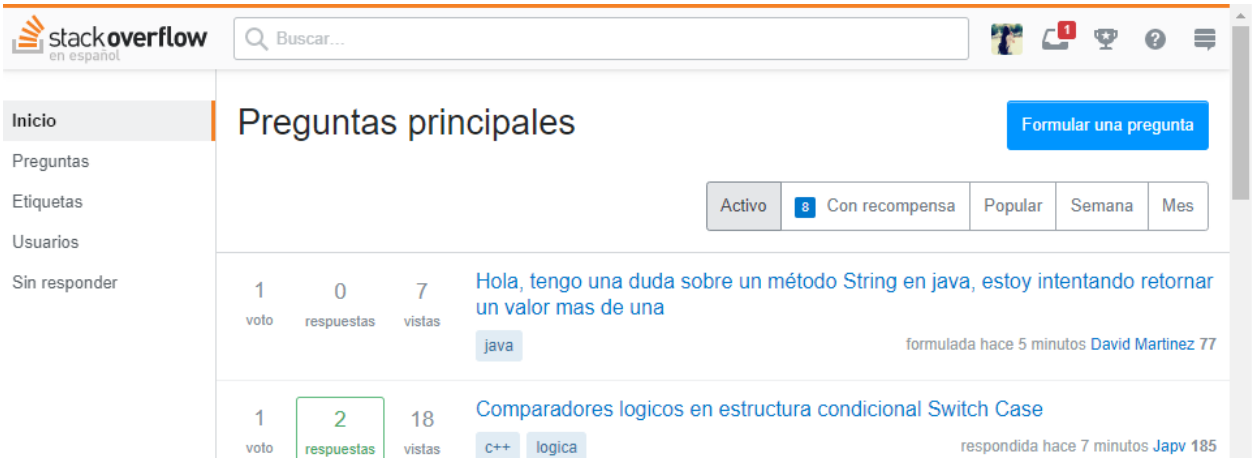
Los primeros tres laboratorios tendrán interfaces claramente definidas (esto es, formatos de lectura, cabeceras de funciones, etc). **El no respetar estas interfaces/prototipos implicarán la obtención de la nota mínima y por consecuencia la reprobación del laboratorio.**

Para este semestre se ha definido hacer una simulación de un software de sistema de foros, tomando como ejemplo la implementación de **Stack Overflow**. Al final del semestre los alumnos 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.

Stack Overflow es un foro que construye una base de conocimiento sobre programación, permitiendo plantear preguntas que luego pueden ser respondidas por otros usuarios.

- Stack Overflow tiene los siguientes elementos claves a considerar: Etiquetas, Preguntas, respuestas y Usuarios.

La página principal del sitio, una vez se inicia sesión, se visualiza de la siguiente manera:



1.1 Etiquetas:

"*Una etiqueta es una palabra clave o un rótulo que categoriza tu pregunta con otras preguntas similares. El uso de las etiquetas correctas facilita que otros encuentren una pregunta y la respondan*".

En Stack overflow se encuentran múltiples etiquetas, por ejemplo: php, javascript, python, etc…

Cada etiqueta posee una **descripción** sobre en qué consiste que el usuario puede ver al situar el puntero sobre ella. Además, un usuario puede seguir una etiqueta si esta le interesa, lo que le permite ver en su inicio preguntas y respuestas relacionadas a esta.

1.2 Preguntas:

Las preguntas publicadas por los usuarios en el sitio tienen las siguientes características principales:

- Votos
- Respuestas (answers)

- Visualizaciones
- Etiquetas
- Título
- Contenido
- Fecha de publicación
- Fecha de última actividad (respuesta, modificación, etc)
- Autor (usuario)
- Estado
- Recompensa (Opcional): El usuario que plantea una pregunta puede ofrecer una recompensa en puntos de reputación, los cuales se deducen de su propia reputación.

  Los usuarios pueden publicar una pregunta, editarla o eliminarla. Además, el usuario que publicó una pregunta puede seleccionar la que, a su gusto, es la mejor respuesta.

  Otros usuarios pueden votar (positiva o negativamente) tanto por la pregunta como por sus respuestas.

  - Las preguntas permiten ingresar texto con formato (negrita, cursiva), imágenes, enlaces y código.

1.3 Usuario y Reputación:

Otra de las partes esenciales de Stack Overflow son los usuarios. Estos son quienes plantean y responden las preguntas del sitio y, para asegurar la calidad del contenido, el sitio provee un sistema de reputación de usuarios.

Reputación: Se trata de una medida que representa cuánto confía la comunidad en un usuario en particular y se gana ofreciendo respuestas y haciendo preguntas útiles que contribuyan a generar conocimiento.

A continuación hay un listado de acciones en el sitio que afectan a la reputación del usuario:

Se gana reputación cuando:
- tu pregunta es votada a favor: +10
- tu respuesta es votada a favor: +10
- tu respuesta es marcada como "aceptada": +15 (y +2 para quien "acepta" la respuesta)
- una edición sugerida es aceptada: +2 (hasta +1000 en total por usuario)
- una recompensa es otorgada manualmente a tu respuesta: + el total del monto ofrecido como recompensa

Se pierde reputación cuando:

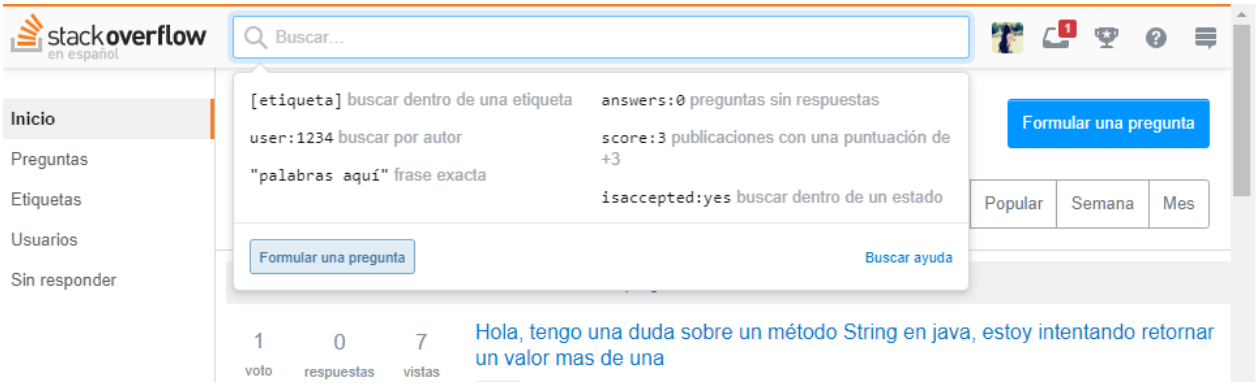
- tu pregunta es votada en contra: -2

- tu respuesta es votada en contra: -2
- tú votas en contra una respuesta de otro usuario: -1
- tú ofreces una recompensa sobre una pregunta: - el total del monto ofrecido como recompensa
- una publicación que has hecho recibe 6 reportes de spam u ofensivo: -100

1.4 Opciones de búsqueda:

Para buscar entre los posts de Stack Overflow, el buscador del sitio ofrece varias características interesantes que permiten encontrar información rápidamente.

- Buscar por etiquetas: Para buscar por etiquetas el usuario debe colocar la palabra entre corchetes. Ej: [javascript]
- Buscar por palabras exactas: Para buscar por palabras exactas el usuario debe colocar la palabra o frase a buscar entre comillas. Ej: “control de versiones”.
- Buscar por cantidad de respuestas: para buscar por cantidad de respuestas el usuario debe escribir la palabra “answers” seguida de dos puntos y la cantidad de respuestas que desea que tengan los resultados. Ej: answers:0
- Buscar por puntaje: para buscar por puntaje el usuario debe escribir la palabra “score” seguida de dos puntos y el puntaje que desea que tengan los resultados. Ej: score: 2



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.

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

En caso de requerir el uso de comodín, podrá hacer entrega del correspondiente laboratorio corregido al final del semestre. 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

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

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.

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**.

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.

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

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: 9-11-2020 (Vespertino)

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 un sistema de preguntas y respuestas como stackoverflow.

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 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.

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 register.

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. En particular, las funciones obligatorias asumen una estructura de repositorios lineal (solo master y por tanto sin ramas), mientras que algunas funciones extras se enfocan en el uso de ramas. 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 “stack” que corresponde al contenedor de preguntas, respuestas, usuarios y nombre del usuario con sesión activa.

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

   De igual forma, las preguntas, además de información relativa a fecha de publicación, ID único incremental (se va generando un correlativo a partir de la última pregunta registrada), debe contar con información relativa a votos a favor, votos en contra, recompensas, número de visualizaciones, estado (cerrada/abierta), listado de respuestas, reportes de ofensa y otros datos que considere relevante para abordar el problema.

   Para el caso de las respuestas, se registra el autor de la respuesta, fecha de respuesta, ID único incremental en el contexto de la pregunta (se va generando un correlativo a partir de la última respuesta registrada), votos a favor, votos en contra, estado de aceptación (si/no), reportes de ofensa y otros datos que considere relevante para abordar 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.
Ejemplo (Solo de referencia. Cada estudiante define sus propios TDAs)	*(stack …..) ; constructor de stack* *(questions stack) ;selector preguntas desde el stack* *(answers stack questionID) ;selector respuestas pregunta* *etc.*

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

Prerrequisitos para evaluación	Implementación de TDAs
Requisitos de implementación	- Emplear recursión natural
Dominio	stack X string X string
Recorrido	stack
Encabezado	*(register stack username password)*
Ejemplos	*(register stackOverflow “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 del stack. 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 stack actualizado. La actualización del stack incorpora al usuario autenticado en 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	- Usar recursión natural o de cola - 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) - Emplear recursión de cola
Dominio	stack X string X string X function
Recorrido **Recorrido final**	function stack
Encabezado	*(login stack username password operation)*
Ejemplos	*(login stackOverflow “user” “pass” ask)*

4. **(0.5 pts) ask:** Función currificada que permite a un usuario con sesión iniciada en la plataforma realizar una nueva pregunta. Cada pregunta registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación, la pregunta y 0 o más etiquetas (en este orden). El retorno final de la función es una versión actualizada del stack donde se registra la nueva pregunta y se elimina la sesión activa del usuario en el stack.

Prerrequisitos para evaluación	Implementación de TDAs, función login
Requisitos de implementación	- Esta función no arroja ningún cambio en stack 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 parámetro. El retorno para la evaluación sobre el parámetro stack es una función que opera sobre el resto de los argumentos. Retorno es una función currificada.
Dominio	stack
Recorrido **Recorrido final**	function: list ->stack X date X string X string list stack
Encabezado	*(ask stack)*
Ejemplos	*(((login stackOverflow “user” “pass” ask) (date 30 10 2020)) “mi pregunta” “e1” “e2 “e3”)*
Tips	Para la definición de la función más interna que retorna ask y que opera sobre los argumentos pregunta y etiquetas, considere la siguiente definición: *(lambda (question . labels) ……)*

6. **(0.5 pto) reward:** Función currificada que permite a un usuario con sesión iniciada en la plataforma ofrecer una recompensa para una determinada pregunta. La recompensa puede ir dirigida a preguntas propias o de terceros. Las preguntas se refieren en base a su identificador único. No se puede ofrecer recompensas mayores a la reputación con la que cuenta el usuario. Una vez ofrecida la recompensa, el puntaje ofrecido queda retenido temporalmente y descontado de la reputación actual. De esta forma se resguarda que el usuario no pueda ofrecer más recompensas de las que podrá eventualmente pagar. Una vez que una respuesta es aceptada, la recompensa se descuenta definitivamente de la reputación del autor y se traspasa al usuario que entregó la respuesta. El resultado de la función es un stack donde se refleja el cambio en el estado de pregunta (recompensa), retenciones en la reputación del usuario que ofrece la recompensa y se elimina la sesión activa del usuario en el stack.

Prerrequisitos para evaluación	Implementación de TDAs, función ask
Requisitos de implementación	- Esta función no arroja ningún cambio 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 tres primeros parámetros. El retorno para la evaluación sobre el parámetro stack es una función que opera sobre el resto de argumentos. Retorno es una función currificada.
Dominio	stack
Recorrido **Recorrido final**	function: integer (IDPregunta) X integer (recompensa) stack
Encabezado	*(reward stack)*
Ejemplos	*(((login stackOverflow “user” “pass” reward) 123) 300)*

7. **(0.5 pts) answer:** Función currificada que permite a un usuario con sesión iniciada en la plataforma responder una pregunta. Cada respuesta registra al autor de la misma (obtenido desde la sesión iniciada con login), se vincula al ID de la pregunta, incluye fecha de publicación, la respuesta y 0 o más etiquetas (en este orden). El retorno final de la función es una versión actualizada del stack donde se registra la nueva respuesta vinculada a la pregunta y se elimina la sesión activa del usuario en el stack.

Prerrequisitos para evaluación	Implementación de TDAs, función reward
Requisitos de implementación	- Esta función no arroja ningún cambio 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 tres primeros parámetros. El retorno para la evaluación sobre el parámetro stack es una función que opera sobre el resto de argumentos. Retorno es una función currificada.
Dominio	stack
Recorrido **Recorrido final**	function: list ->stack X date X string X string list stack
Encabezado	*(answer stack)*
Ejemplos	*((((login stackOverflow “user” “pass” answer) (date 25 10 2020)) 132) “mi respuesta” “e1” “e2 “e3”)*
Tips	Para la definición de la función más interna que retorna answer y que opera sobre los argumentos respuesta y etiquetas, considere la siguiente definición: *(lambda (answer . labels) ……)*

8. **(1 pts) accept:** Función currificada que permite a un usuario con sesión iniciada en la plataforma aceptar una respuesta a una de sus preguntas. La pregunta y la respuesta involucrada se refieren con sus respectivos IDs. Un usuario solo puede aceptar respuestas a sus propias preguntas. El resultado de la función es un stack donde se refleja el cambio en el estado de pregunta, respuesta, **actualización de puntajes** según indicaciones señaladas al comienzo de este enunciado (votos, recompensas vigentes, etc.) y se elimina la sesión activa del usuario en el stack. .

Prerrequisitos para evaluación	Implementación de TDAs, función answer
Requisitos de implementación	- Esta función no arroja ningún cambio si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario. - Función debe ser implementada de forma currificada
Dominio	stack
Recorrido **Recorrido final**	function: integer (IDPregunta) X integer (IDRespuesta) stack
Encabezado	*(accept stack)*
Ejemplos	*(((login stackOverflow “user” “pass” accept) 10) 5)*

9. **(0.5 pts) stack->string**: Función que recibe un stack y entrega una representación del mismo como un string posible de visualizar de forma comprensible al usuario. 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, funciones ask y answer.
Requisitos de implementación	- El string resultante al ser impreso por la función write o display debe ilustrar con claridad los elementos de un stack (i.e., preguntas con sus respuestas, usuarios, sus valoraciones, etc.). - El resultado de aplicar string->stack directamente sobre stack sin usar función login de por medio, imprime todo el stack. - El resultado de aplicar string->stack a través de la función login entrega como resultado todos los antecedentes (preguntas formuladas con sus respectivas respuestas) junto a los detalles del usuario que inició sesión .
Dominio	stack
Recorrido	string
Encabezado	(stack->string stack)
Ejemplos	*(login stackOverflow "user" "pass"* stack->string*)* *(*stack->string *stackOverflow)*

De las siguientes funciones implementar sólo 1 conforme al cumplimiento de los prerrequisitos. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.

10. **(2 pts) vote:** Función que permite a un usuario votar por una pregunta o respuesta. El voto puede ser positivo o negativo. Para esto se debe entregar la pregunta o respuesta sobre la que se desea efectuar el voto y el tipo de voto (positivo o negativo) y el usuario que efectúa el voto. Los puntajes a favor o en contra se deben ajustar a las indicadas presentadas al comienzo de este enunciado. El resultado de la función es un nuevo stack con los cambios producidos por el voto y se elimina la sesión activa del usuario en el stack.

Prerrequisitos para evaluación	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
Requisitos de implementación	- Para esta función debe implementar dos funciones complementarias (questions y answers). Estas funciones se emplean como argumento de vote para determinar donde se aplica el voto. - La función complementaria getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID. Estas funciones deben estar currificada para que sean empleadas internamente por vote. - La función vote debe implementarse de manera currificada
Dominio	stack
Recorrido **Recorrido final**	function: function X integer (ID pregunta o respuesta) X boolean (true: positivo | false: negativo) stack
Encabezado	(vote stack)
Ejemplos	*;voto positivo para la pregunta 25* *((((login stackOverflow "user" "pass"* vote*) getQuestion) 25) true)* *;voto negativo para la respuesta 4 de la pregunta 10* *((((login stackOverflow "user" "pass"* vote*) (getAnswer 10)) 4) false)*

11. **(2 pts) reportOffense:** Función que permite a un usuario reportar preguntas o respuestas que contengan contenido ofensivo. Este contenido puede encontrarse en preguntas o respuestas. La función recibe como entrada el stack y el ID de pregunta o el ID de pregunta y respuesta que se desea reportar. El retorno de la función corresponde a un stack con el reporte de ofensa en la pregunta o respuesta, ajuste de puntaje asociado a reputación (en caso de que se hayan alcanzado 6 o más votos de ofensa, en cuyo caso se aplica descuento según indicaciones provistas al comienzo del enunciado) y se elimina la sesión activa del usuario en el stack.

Prerrequisitos para evaluación	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
Requisitos de implementación	- Para esta función debe implementar dos funciones complementarias (questions y answers). Estas funciones se emplean como argumento de vote para determinar donde se aplica el voto. - La función complementaria getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID. Estas funciones deben estar currificada para que sean empleadas internamente por reportOffense. - La función vote debe implementarse de manera currificada
Dominio	stack
Recorrido **Recorrido final**	function: function X integer (ID pregunta o respuesta) stack
Encabezado	(reportOffense stack)
Ejemplos	*;reporte de ofensa para la pregunta 25* *(((login stackOverflow "user" "pass"* reportOffense*) getQuestion) 25)* *;reporte de ofensa para la respuesta 4 de la pregunta 10* *(((login stackOverflow "user" "pass"* reportOffense*) (getAnswers 10)) 4)*

12. **(2 pts) ranking:** Función que retorna un ranking de las preguntas o respuestas de una pregunta según votación, pudiendo ser el orden ascendente (menos votado al mas votado) o descendente (más votado al menos votado). La función se puede usar directamente sin pasar por login. El retorno es una lista de elementos (preguntas ordenadas.

Prerrequisitos para evaluación	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
Requisitos de implementación	- Para esta función debe implementar las siguientes funciones complementarias: questions, answers, users, asc, desc. Estas funciones se emplean como argumento de ranking para determinar que se quiere rankear y en qué orden. - La función complementaria questions permite obtener las preguntas dentro de un stack. Por otro lado, getAnswer permite seleccionar la respuesta para una pregunta a través de su ID. Por último, la función users permite obtener el listado de usuarios. Estas funciones deben estar currificada para que sean empleadas internamente por ranking. - La función ranking debe implementarse de manera currificada - Puede usar funciones de ordenamiento de listas que ya vienen implementadas en Scheme o Racket.
Dominio	stack
Recorrido **Recorrido final**	function: function X function list
Encabezado	(ranking stack)
Ejemplos	*;ranking ascendente de preguntas* *(((*ranking *stackOverflow) questions) asc)* *;ranking descendente de respuestas para pregunta 10* *(((login stackOverflow "user" "pass"* ranking*) (answers 10)) desc)* *;ranking ascendente de usuarios según reputación* *(((*ranking *stackOverflow) users) asc)*

13. **(2 pts) search:** Función que permite buscar preguntas o respuestas 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 la pregunta y/o respuestas incluyendo las etiquetas, o si la búsqueda será solo por etiqueta. La función se puede usar directamente sin pasar por login. El resultado de la búsqueda debe ser un listado de las preguntas y/o respuestas que cumplen con criterio de búsqueda.

Prerrequisitos para evaluación	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
Requisitos de implementación	- Para esta función debe implementar dos funciones complementarias (all y labels). Estas funciones se emplean como argumento de search para concentrar las búsquedas en todo el cuerpo de texto o solo en las etiquetas
Dominio	stack
Recorrido **Recorrido final**	function: function X string list
Encabezado	(search stack)
Ejemplos	*;búsqueda del texto "curry" en todo el cuerpo de texto* *;retorna preguntas o respuestas que contengan el substring "curry" (ej: currying, currificación)* *(((*search stackOverflow*) all) "curry")* *;búsqueda del texto "racket" en las etiquetas de preguntas y respuestas* *(((login stackOverflow "user" "pass"* search*) labels) "racket")*

14. **(2 pts) clean:** Función que automáticamente realiza un proceso de limpieza del stack borrando los N elementos según filtro. Los filtros pueden estar orientados a eliminar preguntas con baja votación (worstQuestions) y preguntas previas a una fecha F sin respuesta (before). Por otro lado, la función permite eliminar todas las preguntas reportadas como ofensivas (con más de 6 reportes de ofensa). La función clean se puede aplicar independientemente de login. El resultado de la función es un stack con el contenido eliminado.

Prerrequisitos para evaluación	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
Requisitos de implementación	- Para esta función debe implementar funciones complementarias que permitan aplicar el filtro.
Dominio	stack
Recorrido **Recorrido final**	function: function X string stack
Encabezado	(clean stack)
Ejemplos	*;Elimina las 5 preguntas con votación más baja* *(((clean stack) worstQuestions) 5)* *;Elimina las 8 preguntas sin respuestas antes del 5/12/2019* *(((clean stack) (before (date 5 12 2019))) 8)* *;Elimina todas las preguntas marcadas como ofensivas* *((clean stack) offenses)*

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 Vespertino: 30-11-2020

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 un sistema de preguntas y respuestas como stackoverflow.

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 *register*.

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. **(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**.

   Aplican los mismos requisitos del laboratorio anterior para su TDA.

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.
Ejemplo (Solo de referencia. Cada estudiante define sus propios TDAs)	*stack(…..SOut) ; constructor de stack* *stackQuestions(Stack, Questions) ;selector preguntas desde el stack* *stackAnswers(Stack, QuestionID, Answers) ;selector de respuestas a una pregunta* *etc.*

2. **(0.5 pts) register:** Predicado que permite consultar el valor que toma un Stack al ocurrir el registro de un nuevo usuario. Para esto se ingresa un stack inicial, nombre del usuario (identificador único, se debe verificar que no exista para su correcto registro), contraseña y el Stack 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 un Stack2 como variable sin valor como también teniendo un valor. ● En el caso de ser una variable esta se unifica con el Stack resultante luego de registrar al usuario. ● En caso de ser un valor concreto se verifica si el Stack2 cumple con ser el resultante luego de registrar al usuario.
Dominio	stack X string X string x stack
Encabezado	*stackRegister(Stack, Username, Password, Stack2).*
Ejemplos	*stackRegister(S1, “user”, “pass”, S2).*

3. **(0.5 pts) login:** 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 *register*
Requisitos de implementación	- Si “Stack2” es una variable, se debe unificar con el stack resultante luego de agregar una sesión activa con el usuario que realizó login. - Si el usuario y/o la contraseña con incorrectas se debe retornar *false*.
Dominio	stack X string X string X stack
Encabezado	*stackLogin(Stack, Username, Password, Stack2).*
Ejemplos	*stackLogin(Stack, “user”, “pass”, Stack2).*

4. **(0.5 pts) ask:** Predicado que permite a un usuario con sesión iniciada en la plataforma realizar una nueva pregunta. Cada pregunta registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación, la pregunta, un listado de etiquetas. El retorno es *true* si se puede satisfacer en "Stack2" el stack con la nueva pregunta incluida y sin la sesión activa del usuario que realizó la pregunta.

Prerrequisitos para evaluación	Implementación de TDAs, predicado login
Requisitos de implementación	-Debe funcionar cuando se entrega un Stack2 como variable sin valor como también teniendo un valor. ● En el caso de ser una variable esta se unifica con el Stack resultante luego de agregar la pregunta y eliminar la sesión del usuario activo. ● En caso de ser un valor concreto se verifica si el Stack2 cumple con ser el resultante luego de haber realizado la pregunta 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.
Dominio	stack X date X string X string list x stack
Encabezado	*ask(Stack, Fecha, TextoPregunta, ListaEtiquetas, Stack2).*
Ejemplos	*ask(Stack, "2020-10-30", "mi pregunta", ["e1" "e2 "e3"], Stack2).*

6. **(0.5 pto) reward:** Predicado que permite consultar el valor que toma un Stack al ofrecer una recompensa para una determinada pregunta. La recompensa puede ir dirigida a preguntas propias o de terceros. Las preguntas se refieren en base a su identificador único. No se puede ofrecer recompensas mayores a la reputación con la que cuenta el usuario. Una vez ofrecida la recompensa, el puntaje ofrecido queda retenido temporalmente y descontado de la reputación actual. De esta forma se resguarda que el usuario no pueda ofrecer más recompensas de las que podrá eventualmente pagar. Una vez que una respuesta es aceptada, la recompensa se descuenta definitivamente de la reputación del autor y se traspasa al usuario que entregó la respuesta. Para esto se ingresa un stack inicial, el id de la pregunta a la que ofrecer una recompensa, el monto de la recompensa y el Stack resultante luego de ofrecida la recompensa. El retorno del predicado es *true* en caso que se pudo satisfacer esta recompensa a la pregunta.

Prerrequisitos para evaluación	Implementación de TDAs, predicado ask
Requisitos de implementación	- Debe retornar *false* si no existe una sesión activa. en el stack. - Debe retornar false si la pregunta con ese id no existe. - Para los casos anteriormente descritos debe hacer uso de predicado de corte !.
Dominio	stack X number X number X stack
Encabezado	*reward(Stack, IdPregunta, Recompensa, Stack2).*
Ejemplos	*reward(Stack, 123, 300, Stack2).*

7. **(0.5 pts) answer:** Predicado que permite consultar el valor que toma un Stack al realizar la respuesta a una pregunta por cierto usuario que tenga sesión iniciada. Para esto se ingresa un stack inicial, la fecha de la respuesta, el id de la pregunta, el texto de la respuesta, un listado de etiquetas y el Stack resultante luego del registro. El retorno del

predicado es *true* en caso que se pudo satisfacer la creación de esta respuesta en el Stack resultante.

Prerrequisitos para evaluación	Implementación de TDAs, predicado reward
Requisitos de implementación	- Debe retornar *false* si no existe una sesión activa. en el stack. - Debe retornar false si la pregunta con ese id no existe. - Para los casos anteriormente descritos debe hacer uso de predicado de corte !. - La fecha puede estar representada como lo estime conveniente, pero debe indicarlo en los ejemplos de uso.
Dominio	stack X date X number X string X string list X stack
Encabezado	*answer(Stack, Fecha, IdPregunta, TextoRespuesta, ListaEtiquetas, Stack2).*
Ejemplos	*answer(Stack, “2020-10-31”, 132, “mi respuesta”, [“e1” “e2 “e3”]).*

8. **(1 pts) accept:** Predicado que permite consultar el valor que toma un Stack al aceptar una respuesta a una de sus preguntas. Para esto se ingresa un stack inicial, el id de una pregunta, el id de una respuesta y el Stack resultante luego de aceptar la respuesta. El retorno del predicado es *true* en caso que se pudo satisfacer el registro del usuario. Un usuario solo puede aceptar respuestas a sus propias preguntas. El stack resultante debe reflejar el nuevo estado de pregunta, respuesta, **actualización de puntajes** según indicaciones señaladas al comienzo de este enunciado (votos, recompensas vigentes, etc.) y la sesión activa del usuario en el stack debe desaparecer.

Prerrequisitos para evaluación	Implementación de TDAs, predicado answer
Requisitos de implementación	- Debe retornar *false* si no existe una sesión activa. en el stack. - Debe retornar *false* si la sesión iniciada no es del mismo usuario que hizo la pregunta a la que se está aceptando. - Debe retornar *false* si la pregunta con ese id no existe o si la respuesta con ese Id no existe. - Para los casos anteriormente descritos debe hacer uso de predicado de corte !.
Dominio	stack X number X number X stack
Encabezado	*accept(Stack, IdPregunta, IdRespuesta, Stack2).*
Ejemplos	*accept(Stack, 10, 5, Stack2).*

9. **(0.5 pts) stack->string**: Predicado que permite consultar el valor que toma un String al intentar representar en este cierto Stack. Para esto se ingresa un stack inicial y el String equivalente al Stack. El retorno del predicado es *true* en caso que se pudiera satisfacer esta equivalencia entre el stack y el string. En el String equivalente debe hacer uso del char '\n' para los saltos de línea. **No utilice las funciones write y display dentro de este predicado**, ya que si este string se pasa como argumento a los predicados "write" o "display" entonces este es visualizado de forma comprensible al usuario en la consola de prolog.

Prerrequisitos para evaluación	Implementación de TDAs, predicado login y ask.
Requisitos de implementación	- El string resultante al ser impreso por el predicado write o display debe ilustrar con claridad los elementos de un stack (i.e., preguntas con sus respuestas, usuarios, sus valoraciones, sesiones activas, etc.). - El resultado de aplicar stackToString directamente sobre stack sin usar función login de por medio, imprime todo el stack. - El resultado de aplicar stackToString con una sesión activa debe hacer que la unificación de la variable "StackStr" tenga solo las preguntas realizadas por el usuario con la sesión activa junto a los detalles de ese usuario.
Dominio	stack X string
Encabezado	stackTostring(Stack, StackStr).
Ejemplos	*stackLogin(Stack, "user", "pass", Stack2),* stackToString(Stack2, Str).

De los siguientes predicados, implementar sólo 1 conforme al cumplimiento de los prerrequisitos. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.

1. **(2 pts) vote:** Predicado que permite consultar el valor que toma un Stack al ocurrir un voto del usuario con sesión activa a una pregunta o respuesta. Para esto se ingresa un stack inicial, un TDA pregunta o respuesta, el voto que puede ser positivo o negativo y el Stack resultante luego de realizado el voto y sin la sesión activa del usuario que realizó el voto. Los puntajes a favor o en contra se deben ajustar a las indicadas presentadas al comienzo de este enunciado. El retorno del predicado es *true* en caso que se pudo satisfacer el voto del usuario.

Prerrequisitos para evaluación	Implementación de TDAs, predicados ask, reward, answer, accept y stackTostring
Requisitos de implementación	- Para este predicado debe implementar dos predicados complementarios (getQuestion y getAnswer). Estos predicados se emplean para determinar el TDA pregunta o respuesta donde se aplica el voto. - El predicado complementario getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID. El predicado vote debe ser capaz de determinar si su segundo argumento corresponde a una pregunta o una respuesta. - Debe retornar *false* si el voto va para una pregunta que no es del usuario que tiene sesión activa en el stack.
Dominio	stack X Pregunta o Respuesta X boolean X stack
Encabezado	vote(Stack, TdaPreguntaORespuesta, Voto, Stack).
Ejemplos	*;voto positivo para la pregunta 25* *getQuestion(Stack, 25, Preg),* vote(Stack*, Preg, true, Stack2).* *;voto negativo para la respuesta 4 de la pregunta 10* *getAnswer(Stack, 10, 4, Resp), vote(Stack, Resp, false, Stack2).*

(2 pts) reportOffense: En desarrollo...