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

Paradigmas de Programación

Proyecto semestral de laboratorio

Versión Preliminar - al 26/03/2018

Durante el semestre se trabajarán distintos paradigmas de programación, los cuales serán abordados mediante cuatro lenguajes de programación en cuatro entregas de laboratorio. El tema del proyecto a desarrollar será un chatbot.

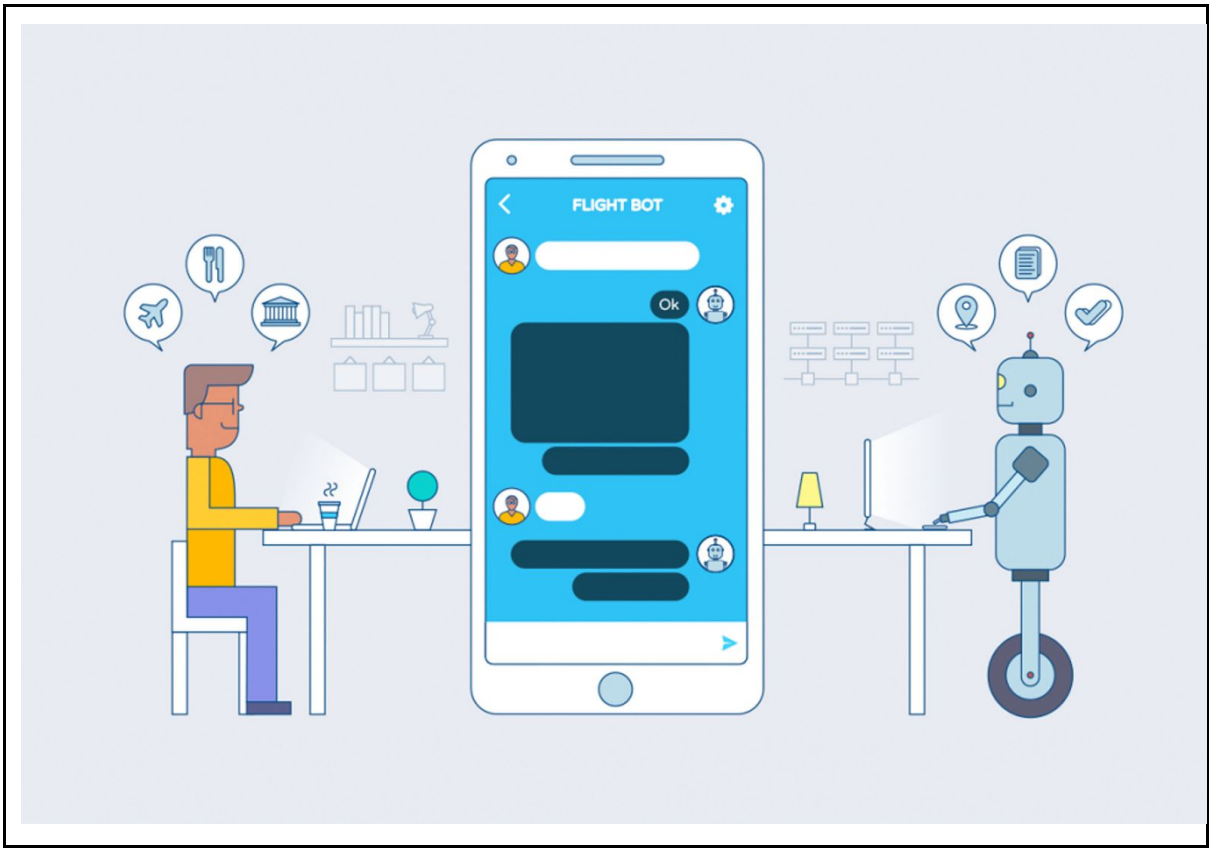

Figura 1. Ilustración chatbot. Fuente: http://www.tci.net.pe/sera-2018-ano-los-chatbots/

Descripción general

Varios aspectos del chatbot quedarán abiertos a la creatividad de cada estudiante, sin embargo se debe cumplir con requisitos mínimos los cuales se especifican en los enunciados específicos para cada laboratorio. Además se deben respetar los siguientes puntos.

Los primeros tres laboratorios tendrán interfaces claramente definidas (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 de laboratorio.**

Los chatbots se pueden definir como entidades conversacionales artificiales. Se trata de programas computacionales capaces de mantener una conversación con un ser humano, la que queda limitada por una serie de parámetros y contexto (Lectura recomendada: Turing Test y Turing Test 50 Años después). Los chatbots suelen ser utilizados ampliamente en la actualidad en los portales web de empresas y otras organizaciones que ofrecen algún tipo de atención al público/clientes. La comunicación de los chatbots puede ser a través de voz o texto.

Los chatbots pueden resultar de utilidad para manejar algunos aspectos protocolares de la conversación. Por ejemplo, saludos de bienvenida, solicitud de datos de identificación y proporcionar algunas respuestas elementales (principalmente factuales) a las preguntas del usuario.

A modo de ejemplo, considere la siguiente construcción de un chatbot simple en scratch: https://codeclubprojects.org/en-GB/scratch/chatbot/

A continuación se detallan las consideraciones que debe tener para la construcción de su chatbot.

- Defina un contexto/temática para el chatbot a implementar (comercio electrónico, video juegos, buscador de parejas, amigo virtual, atención al cliente, búsqueda de empleo, etc.) Este contexto/temática se debe mantener de forma consistente a lo largo de los cuatro proyectos.
- Diseñar flujos conversacionales protocolares (ej: saludos, bienvenidas, etc.) y otros de estilo libre. Todo debe quedar documentado y plasmado en su informe técnico.
- Considerar en su implementación del chatbot y diseño conversacional, componentes de respuesta empática para lograr una mejor conexión entre el usuario y el chatbot.
    - Ejemplo:
        - Chatbot: ¿cómo estás?
        - Usuario: mas o menos
        - Chatbot: Lo lamento. ¿Qué puedo hacer para ayudarte a estar mejor?
- Considere un amplio abanico de posibilidades para mantener una conversación activa a fin de cautivar la atención del usuario.
- Para evaluar el comportamiento de su chatbot debe implementar simulaciones de usuario que permitan iniciar distintos flujos de conversación con el chatbot.
- Dentro de las respuestas entregadas por el chatbot, considere la incorporación de historias que le permitan reaccionar a distintas situaciones.
- Considere que el usuario que conversara con su chatbot utiliza un lenguaje con estructuras gramaticales simple y vocabulario conocido por el chatbot. También se

omite la posibilidad de que existan faltas de ortografía por parte del usuario. Puede asumir que sus respuestas son bien estructuradas y que se ajustan de manera precisa al flujo conversacional que implementa su chatbot. (Ej: si el chatbot pregunta "¿cómo te llamas?", el usuario podría responder "Juan Saez") Esta decisión tiene por objetivo reducir la complejidad asociada al procesamiento del lenguaje natural. Usted incluso puede diseñar una conversación que dirija al usuario a responder de manera más precisa (sin caer en exageraciones, que terminen convirtiendo al chatbot en un formulario) con el fin de poder procesar de mejor manera la información procesada por éste (ej. en lugar de preguntar "¿cómo te llamas?", podría preguntar "¿cuáles son sus nombres?" y luego, "¿Cuáles son sus apellidos?").

- Su chatbot debe estar implementado únicamente para mantener una conversación en español.
- Además de la información recabada de la interacción del usuario, su chatbot debe recabar información del contexto (de forma simulada o real). Por ejemplo, se puede utilizar la hora del sistema para determinar el tipo de saludo que se dará al usuario (ej: "Buenos días", "Buenas tardes", "Buenas noches").
- El chatbot debería manejar un historial de la conversación con el fin de implementar memoria de corto, mediano y/o largo plazo. De esta manera el chatbot podrá "recordar" entre otras cosas el nombre del usuario para referirse a éste en términos más personalizados.
- Ante entradas desconocidas, el chatbot debe contar con un repertorio de respuestas que le permitan redirigir la conversación con el usuario (Ej: "disculpe, no entendí bien su respuesta. Me lo podría decir de otra forma").
- La memoria de largo plazo, permitirá al chatbot "aprender" de interacciones previas para variar dentro de ciertos parámetros la forma de responder al usuario. A modo de ejemplo, en un chatbot de atención comercial que maneja las preferencias de usuarios en la contratación de un servicio se podría dar un diálogo como el siguiente.
    - Ejemplo:
        - chatbot: ¿Cuál de nuestros servicios desea contratar?
        - usuario: cable
        - chatbot: Registrado ¿que hay de internet?
        - usuario: no
        - chatbot: Entendido ¿y telefonía?
        - usuario: si
        - chatbot: Perfecto, ya he registrado sus servicios.

  Supongamos por un momento que estas mismas dos opciones (cable y telefonía) son escogidas por la mayoría de los clientes de la compañía. Luego, en un futuro diálogo con otros clientes y apoyado en la "memoria de largo plazo" el diálogo podría cambiar a:

    - Ejemplo:
        - chatbot: ¿Cuál de nuestros servicios desea contratar?
        - usuario: cable
        - chatbot: Registrado ¿que hay de telefonía?
        - usuario: si

- chatbot: Entendido ¿e Internet?
- usuario: no gracias
- chatbot: Perfecto, ya he registrado sus servicios.

Esto implica que basado en los antecedentes de conversaciones previas, el chatbot "aprende" y sabe que la mayoría de las personas que contratan cable, a continuación debería preguntarles por telefonía y no por Internet.

Nótese que esto es solo un ejemplo de flujo de conversación. Usted puede implementar conversaciones más ricas que permitan al usuario expresarse con mayor libertad, sin limitarlo a respuestas atómicas.

Ejemplo:

- chatbot: ¿Cuál de nuestros servicios desea contratar?
- usuario: cable y telefonía
- chatbot: Registrado ¿que hay de Internet?
- usuario: no gracias
- chatbot: Listo, ya he registrado sus servicios.

Comodines: Durante el semestre cada alumno dispondrá de un total de **DOS comodines** 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 y/o manual de usuario) **es inferior a 4.0 pero mayor que 2.0**.

El comodín permitirá al alumno entregar lo necesario (requerimientos funcionales, no funcionales, informe y/o manual de usuario) para alcanzar la nota mínima requerida (4.0) en la correspondiente entrega de laboratorio para la aprobación al final del semestre.

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 junto a su laboratorio final.

Finalmente, por temas de tiempo considerando el avance del semestre, en el laboratorio final se podrá hacer un uso limitado de comodines (en caso de disponer de estos). 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 17 semanas lectivas.

Laboratorio 1 (Paradigma Funcional - Lenguaje Scheme)

Versión 1.1 (Borrador) Última Actualización el 16 de Abril de 2018.
Obs: Se añade una nueva función extra: displayLog

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

Fecha de Entrega: 23 de Abril de 2018, hasta las 23:55 a través de Usach Virtual

Entregables: Archivo ZIP con el siguiente nombre de archivo: lab2_rut_ApellidoPaterno.zip (ej: lab2_12123456_Perez). Notar que no incluye dígito verificador. Dentro del archivo se debe incluir:

1. Informe en formato Word o PDF.
2. Manual de Usuario en formato Word o PDF.
3. Carpeta con código fuente.
4. Archivo leeme.txt para cualquier instrucción especial para ejecución u otro, según aplique.

El nombre del archivo con el código fuente debe seguir el siguiente formato: NombreArchivo_rut_Apellidos.extension (ej: candyCrush_12123456_PerezPeña.rkt)

Objetivo del laboratorio: Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma de programación funcional usando el lenguaje de programación Scheme.

Resultado esperado: Programa que implementa un chatbot para un contexto/temática determinada bajo el paradigma de programación funcional usando el lenguaje de programación Scheme..

Informe del proyecto: Extensión no debe superar las 10 planas y debe incluir: portada, índice, introducción, descripción del problema abordado con el paradigma y lenguaje, análisis del problema, diseño detallado de la solución implementada, resultados obtenidos y conclusiones. **Contenido requerido y un formato tipo se encuentran disponible en UsachVirtual.**

Manual de Usuario: Extensión no debe superar las 10 planas y debe incluir: portada, índice, introducción, contenido central del manual de usuario. **Contenido requerido y un formato tipo se encuentran disponible en UsachVirtual.**

Evaluación: Tanto el informe (Inf), Manual (Man), como requerimientos funcionales (RF) y no funcionales (RNF) 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.

$$
\begin{aligned}
&\textit{if } (Inf >= 4.0 \;\&\&\; Man >= 4.0 \;\&\&\; (RF + 1.0) >= 4.0 \;\&\&\; (RNF + 1.0) >= 4.0) \textit{ then} \\
&\qquad NL = 0.05 \cdot Inf + 0.05 \cdot Man + 0.7 \cdot (RF + 1.0 + PtjeExtra) + 0.2 \cdot (RNF + 3.0) \\
&\textit{else} \\
&\qquad NL = min(Inf, Man, RF + 1.0, RNF + 1.0)
\end{aligned}
$$

Nota: Existe una calificación interna por ejecución. Si su programa no logra ser interpretado por el intérprete, se calificará con un 1.0 el resto de su trabajo. Si su trabajo es entregado de forma incompleta, procure comentar aquellos elementos que puedan entorpecer la ejecución de su programa.

Requerimientos No Funcionales obligatorios (0.8 pts. c/u).

1. La implementación debe ser en el lenguaje de programación Scheme (Racket 6.12).
2. Se deben utilizar funciones estándar de dicho lenguaje y no aquellas propias Racket. Regirse por standard R6RS (esto último es opcional si desea integrar con C# en el laboratorio 4).
3. No hacer uso de set! o funciones similares para emular el trabajo con variables.
4. 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. Usar comentarios llenando la siguiente plantilla
    a. Nombre función
    b. Firma: Dominio -> Recorrido
    c. Tipo de recursión.
5. Respetar la definición de función en términos de conjunto de salida (dominio) y llegada (recorrido). No implementar relaciones. Además respetar el uso de mayúsculas y minúsculas en los identificadores de las funciones obligatorias y extras, tal como se especifica a continuación.
6. Demostrar el uso de recursión natural y de cola. Al menos un uso concreto por cada uno. Si lo prefiere, puede reemplazar alguno de los tipos anteriores por recursión árborea.

Requerimientos Funcionales Obligatorios (4 pts)

Parámetros comunes: Las siguientes funciones operan sobre un conjunto de parámetros que son comunes a la mayoría de las funciones. Estos se especifican a continuación.

- chatbot: corresponde a una estructura con distintos parámetros que reflejan la personalidad que adoptará el chatbot (ej: coloquial, formal, agresivo, etc.), su vocabulario, evaluaciones de diálogos, etc.

- log: corresponde al registro histórico de conversaciones que ha sostenido el chatbot, el que puede servir como referencia para cambiar los cursos de las nuevas conversaciones. Las conversaciones deben destacar a través de metadatos la fuente de origen de los mensajes (i.e., usuario o chatbot), hora, fecha y otra información relevante que usted considere pertinente para el contexto de uso de su chatbot (ej.: probabilidades, estadísticas que sean de utilidad en el historial del chat). Para efectos de este proyecto el chatbot y el log histórico se manejan de forma independiente con el fin de que distintos chatbots puedan operar sobre un mismo historial.

- seed: corresponde a la semilla que normará el comportamiento de las funciones ante eventuales funciones pseudoaleatorias que sean implementadas para lograr la variabilidad en las respuestas que ofrezca el chatbot ante los mensajes del usuario.

1. **(1 pto)** Implementar abstracciones apropiados para el problema. Hacer uso de la estructura TDA de 6 capas: Representación e implementación del TDA a través de 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 del TDA. Solo implementar las funciones necesarias dentro de esta estructura. Dejar claramente documentado con comentarios en el código que corresponde a la estructura base del TDA.

2. **(0,25 pts)** *beginDialog*: Permite iniciar un nuevo diálogo con el chatbot, donde siempre el mensaje inicial es ofrecido por el chatbot (ej: mensaje de bienvenida, saludo inicial).

   El retorno de esta función es un log modificado que incorpora un delimitador (ej.: “BeginDialog”) que permite separar la conversación en curso de diálogos anteriores. Este delimitador debe registrar datos como la fecha y hora de inicio de la conversación, además de un identificador único que permita vincular las intervenciones que ha tenido un chatbot sobre una determinada conversación.

   (beginDialog chatbot log seed)

3. **(1 pto)** *sendMessage*: Permite realizar envío de mensajes desde el usuario al chatbot. La función recibe cuatro parámetros. Además de los parámetros comunes, msg corresponde a un string con el mensaje formulado por el usuario y el cual es usado por el chatbot en conjunto con el histórico, para definir una respuesta. Cada mensaje debe ir acompañado de metadatos como la fecha y hora.

   El retorno de esta función es un log modificado donde se añade el mensaje del usuario y la respuesta otorgada por el chatbot. Sin embargo debe demostrar en al menos uno de sus retornos el uso de evaluación perezosa. Respecto de esto último, considere por ejemplo que el usuario responde o solicita algo que al sistema le puede tomar tiempo o que no puede responder en ese instante. Ante estos escenarios el sistema devolverá como respuesta una promesa la que puede ser

forzada a ser cumplida por el usuario a través de un mensaje especial definido por usted (ej: “responder las preguntas sin responder”)

(sendMessage msg chatbot log seed)

4. **(0,25 pts)** *endDialog*: Permite dar cierre a un diálogo, donde siempre el mensaje final es ofrecido por el chatbot (ej.: “ha sido un placer ayudarlo”, “espero ayudarlo mejor en otra oportunidad”).

   El retorno de esta función es un log modificado que incluye un delimitador de cierre de la conversación sostenida con el usuario (ej.: “EndDialog”). Este delimitador debe registrar datos como la fecha y hora de término de la conversación, además de un identificador único que permita vincular las intervenciones que ha tenido un chatbot sobre una determinada conversación.

   (endDialog chatbot log seed)

5. **(0,5 pts)** *rate*: Permite evaluar el desempeño del chabot frente a la última conversación registrada en el log desde dos frentes. Primero, a través de un score (puntaje/calificación) entre 0 y 5 asignado por el usuario solo después de haber terminado una conversación. Esto quiere decir que la función rate no se puede aplicar si una conversación no ha sido cerrada a través de la función endDialog. Por otro lado, la función debe utilizar el log para realizar una autoevaluación a partir de una métrica f ingresada como función. La métrica la debe diseñar usted y debe entregar como resultado un valor entre 0 y 5. En ambos casos (evaluación del usuario y autoevaluación) los valores de 1 a 5 corresponde al nivel del servicio desde “muy malo” a “muy bueno” y 0 corresponde a que la evaluación no aplica o no se puede calcular.

   El retorno de esta función es un chatbot con un registro actualizado de sus evaluaciones, donde debe quedar un registro histórico de las distintas evaluaciones que ha ido recibiendo para cada una de las conversaciones. El vínculo entre la evaluación y la conversación queda plasmado a través del identificador único que se asigna a cada chat a través de la función *beginDialog*.

   (rate chatbot score f log)

6. **(1 pto)** *test*: Esta función permite ilustrar/simular el desarrollo de una conversación entre un usuario user y un chatbot sobre la base de un registro histórico de conversaciones log. En este caso, el parámetro user queda expresado como una lista de strings que contienen un conjunto de mensajes predefinidos con los que el usuario irá interactuando con el chatbot. Se asume en este caso que la lista de mensajes (lista de strings) proporcionadas en la lista son suficientes para comenzar, desarrollar y concluir una conversación. Además se debe garantizar que los mensajes provistos en la lista resultan propicios para un flujo normal en el diálogo. En este sentido, deberá proporcionar como ejemplo al menos tres listas definidas a través de los identificadores user1, user2, user3, …, userN, que permitan ilustrar distintos flujos de conversación.

El retorno de esta función es un log actualizado con el desarrollo completo de la conversación simulada.

(test user chatbot log seed)

Requerimientos extra (se consideran si y sólo si la evaluación de los requerimientos obligatorios es igual o superior a 4.0. La nota máxima que se puede alcanzar es un 7.0 en RF).

1. **(1 pto)** *analyse*: Función que permite realizar un análisis histórico del chatbot en conjunto con sus logs (recuerde vincular las intervenciones de un chatbot sobre un log a partir del identificador único generado para cada conversación) a partir de distintas funciones de evaluación las que se pasan a través de la función f. Para esta función debe proporcionar como complemento al menos 3 funciones que permitan por ejemplo: (1) contar cantidad total de mensajes intercambiados por el chatbot, (2) duración promedio de las conversaciones, (3) promedio de ratings, (4) desviación estándar de ratings, (5) moda de ratings, (6) mediana de ratings, (7) promedio de interacciones, (8) largo promedio de mensajes del usuario, entre otras.

   (analyse f chatbot log)

2. **(1 pto)** *chat>xml*: Función que recibe un chatbot y un log como parámetros de entrada con el objeto de producir una combinación estructurada de ambos en formato XML. Usted debe implementar la función de conversión y no usar funciones disponibles en Racket.

   El retorno de la función es un string que contiene toda la información combinada del chatbot y del log en formato XML.

   (chat>xml chatbot log)

3. **(1 pto)** *chat->json*: Función que recibe un chatbot y un log como parámetros de entrada con el objeto de producir una combinación estructurada de ambos en formato JSON. Usted debe implementar la función de conversión y no usar funciones disponibles en Racket.

   El retorno de la función es un string que contiene toda la información combinada del chatbot y del log en formato JSON.

   (chat->json chatbot log)

4. **(0,5 pto) displayLog :** Función que recibe un chatbot y un log e imprime en pantalla la información clave de estos (intercambio de mensajes) en un formato legible para el usuario. Esta es la única función en la que tiene permitido usar la función “display”.

   (displayLog chatbot log)

Laboratorio 2 (Paradigma lógico- Lenguaje Prolog)

Versión 1.0

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

Fecha de Entrega: 21 de mayo de 2017, hasta las 23:55 a través de Usach Virtual

Entregables: Archivo ZIP con el siguiente nombre de archivo: lab2_rut_ApellidoPaterno.zip (ej: lab2_12123456_Perez). Notar que no incluye dígito verificador. Dentro del archivo se debe incluir:

1. Informe en formato Word o PDF.
2. Manual de Usuario en formato Word o PDF.
3. Carpeta con código fuente.
4. Archivo leeme.txt para cualquier instrucción especial para ejecución u otro, según aplique.

El nombre del archivo con el código fuente debe seguir el siguiente formato: NombreArchivo_rut_Apellidos.extension (ej: chatbot_12123456_PerezPeña.pl)

Objetivo del laboratorio: Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma de programación lógico usando el lenguaje de programación Prolog.

Resultado esperado: Programa para la generación de tableros, validación de tableros y ayuda en su resolución.

Informe del proyecto: Extensión no debe superar las 10 planas y debe incluir: portada, índice, introducción, descripción del problema abordado con el paradigma y lenguaje, análisis del problema, diseño de la solución implementada, resultados obtenidos y conclusiones. **Contenido requerido y un formato tipo se encuentran disponible en UsachVirtual.**

Manual de Usuario: Extensión no debe superar las 10 planas y debe incluir: portada, índice, introducción, contenido central del manual de usuario. **Contenido requerido y un formato tipo se encuentran disponible en UsachVirtual.**

Evaluación: Tanto el informe (Inf), Manual (Man), como requerimientos funcionales (RF) y no funcionales (RNF) 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.

$$
\begin{aligned}
&\textit{if } (Inf >= 4.0 \;\&\&\; Man >= 4.0 \;\&\&\; (RF + 1.0) >= 4.0 \;\&\&\; (RNF + 1.0) >= 4.0) \textit{ then}\\
&\qquad NL = 0.05 \cdot Inf + 0.05 \cdot Man + 0.7 \cdot (RF + 1.0 + PtjeExtra) + 0.2 \cdot (RNF + 3,0)\\
&\textit{else}\\
&\qquad NL = min(Inf,\ Man,\ RF + 1.0,\ RNF + 1.0)
\end{aligned}
$$

Requerimientos No Funcionales obligatorios (1.33 pts. c/u).

1. La implementación debe ser en el lenguaje de programación Prolog (SWI-Prolog o P#).
2. Se debe documentar el código indicando una breve descripción del programa, reglas, hechos, parámetros de entrada y salida, consultas, etc.
3. Organización del código (orden y claridad).

Requerimientos Funcionales Obligatorios (5 pts en total)

Parámetros comunes: Las siguientes relaciones operan sobre un conjunto de parámetros que son comunes a la mayoría de las relaciones. Estos se especifican a continuación.

- Chatbot: corresponde a una estructura con distintos parámetros que reflejan la personalidad que adoptará el chatbot (ej: coloquial, formal, agresivo, etc.), su vocabulario, evaluaciones de diálogos, etc.

- InputLog: corresponde al registro histórico de conversaciones que ha sostenido el chatbot (Al igual que en el enunciado anterior)

- Seed: corresponde a la semilla que normará el comportamiento de las funciones ante eventuales funciones pseudoaleatorias que sean implementadas para lograr la variabilidad en las respuestas que ofrezca el chatbot ante los mensajes del usuario.

- OutputLog: Corresponde a la variable de salida de la mayoría de los predicados solicitados, contiene el log luego de haberle agregado algún mensaje a la conversación.

1. **(1 pto)** Implementar abstracciones apropiados para el problema. Hacer uso de la estructura TDA de 6 capas: Representación e implementación del TDA a través de 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 del TDA. Solo implementar las funciones necesarias dentro de esta estructura. Dejar claramente documentado con comentarios en el código que corresponde a la estructura base del TDA.

2. **(0,25 pts)** *beginDialog*: Permite iniciar un nuevo diálogo con el chatbot, donde siempre el mensaje inicial es ofrecido por el chatbot (ej: mensaje de bienvenida, saludo inicial).

   El retorno de este predicado es un log modificado que incorpora un delimitador (ej.: “BeginDialog”) que permite separar la conversación en curso de diálogos anteriores. Este delimitador debe registrar datos como la fecha y hora de inicio de la conversación, además de un identificador único que permita vincular las intervenciones que ha tenido un chatbot sobre una determinada conversación.

   beginDialog (Chatbot, InputLog, Seed, OutputLog).

3. **(1 pto)** *sendMessage*: Permite realizar envío de mensajes desde el usuario al chatbot. El predicado recibe cuatro parámetros y deja su salida en un quinto parámetro. Además de los parámetros comunes, msg corresponde a un string con el mensaje formulado por el usuario y el cual es usado por el chatbot en conjunto con el histórico, para definir una respuesta. Cada mensaje debe ir acompañado de metadatos como la fecha y hora.

   Este predicado deja su resultado en la variable OutputLog, el cual es un log modificado donde se añade el mensaje del usuario y la respuesta otorgada por el chatbot.

   sendMessage(Msg, Chatbot, InputLog, Seed, OutputLog).

4. **(0,25 pts)** *endDialog*: Permite dar cierre a un diálogo, donde siempre el mensaje final es ofrecido por el chatbot (ej.: “ha sido un placer ayudarlo”, “espero ayudarlo mejor en otra oportunidad”).

   Este predicado deja su resultado en OutputLog, el cual es un log modificado que incluye un delimitador de cierre de la conversación sostenida con el usuario (ej.: “EndDialog”). Este delimitador debe registrar datos como la fecha y hora de término de la conversación, además de un identificador único que permita vincular las intervenciones que ha tenido un chatbot sobre una determinada conversación.

   endDialog (Chatbot, InputLog, Seed, OutputLog).

5. **(0,5 pts)** *logToStr*: Predicado que permite obtener una representación entendible para un usuario de su TDA Log, este predicado debe entregar su resultado en la variable StrRep. Nota: No debe utilizar “display(term)” dentro de este predicado, pero sí debe ser posible usar display con el resultado que entrega este predicado.

   logToStr(Log, StrRep).

6. **(1 pto)** *test*: Este predicado permite ilustrar/simular el desarrollo de una conversación entre un usuario User y un Chatbot sobre la base de un registro histórico de conversaciones InputLog. En este caso, la variable User queda expresada como una lista de strings que contienen un conjunto de mensajes predefinidos con los que el usuario irá interactuando con el chatbot. Se asume en este caso que la lista de mensajes (lista de strings) proporcionadas en la lista son suficientes para comenzar, desarrollar y concluir una conversación. Además se debe garantizar que los mensajes provistos en la lista resultan propicios para un flujo normal en el diálogo. En este sentido, deberá proporcionar como ejemplo al menos tres listas definidas a través de los hechos User1, User2, User3, …, UserN, que permitan ilustrar distintos flujos de conversación.

   Este predicado debe entregar su resultado en la variable OutputLog, la cual es un log actualizado con el desarrollo completo de la conversación simulada.

   test(User, Chatbot, InputLog, Seed, OutputLog).

Requerimientos extra (se consideran si y sólo si la evaluación de los requerimientos obligatorios es igual o superior a 4.0. La nota máxima que se puede alcanzar es un 7.0 en RF).

1. **(0,5 pts)** *rate*: por definir

2. **(0,5 pts)** *possibleResponses*: Predicado que recibe un chatbot, un log y un mensaje como entrada y devuelve en una variable una lista de posibles respuestas que puede entregar el chatbot frente al mensaje ingresado. Por ejemplo si la variable Msg toma el valor “22 años”, en Responses deben estar todas las posibles respuestas que puede hacer el chatbot frente a tal mensaje de entrada, por ejemplo las posibles respuestas pueden ser:

[“me parece una buena edad. Ahora indiqueme su rut”, “es usted muy joven. ¿Me indica su rut por favor?”, “Aja! entonces usted nació en el año 1996, qué rut tiene?”]

possibleResponses(Msg, Chatbot, InputLog, Responses).

3. **(1 pto)** *chatToXml*: Predicado que recibe un chatbot y un log como variables de entrada con la finalidad de producir una combinación estructurada de ambos en formato XML. Usted debe implementar la función de conversión y no usar funciones disponibles en Swi-Prolog.

   El predicado debe dejar su resultado en la variable StrXML, el cual debe contener toda la información combinada del chatbot y del log en formato XML.

   chatToXml(Chatbot, InputLog, StrXML).

4. **(1 pto)** *chatToJson*: Predicado que recibe un chatbot y un log como variables de entrada con la finalidad de producir una combinación estructurada de ambos en formato JSON. Usted debe implementar la función de conversión y no usar funciones disponibles en Swi-Prolog.

   El Predicado debe dejar su resultado en la variable StrJSON, el cual debe contener toda la información combinada del chatbot y del log en formato JSON.

   chatToJson(Chatbot, Inputlogm, StrJSON).