UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Informe de Laboratorio 2: Creación de un Simulador de Sistema de Chatbots Simplificado en Prolog

Integrantes: Agustín Vera
Profesor: Edmundo Leiva
Asignatura: Paradigmas de programación

Tabla de contenido

1.- Introducción

En el presente informe correspondiente al curso Paradigmas de Programación. Se expondrá una solución propuesta a un problema planteado. La solución se planteará desde la perspectiva del paradigma lógico, mediante el lenguaje de programación Prolog, haciendo uso del IDE SWI-Prolog, y será explicada en el siguiente informe. El informe posee la siguiente estructura: Introducción con sus subcategorías: descripción del problema, descripción del paradigma, objetivos del proyecto, después se tiene, desarrollo, con sus subcategorías correspondientes: análisis del problema, diseño de la solución, aspectos de implementación, ejemplos de uso, resultados y autoevaluación, finalizando se encuentra una conclusión al trabajo realizado, la bibliografía consultada y anexos.

1.1.- Descripción del problema

Se desea construir un sistema para la creación, despliegue y administración de chatbots simplificados, como por ejemplo, los chatbots de ITR (Respuesta de Interacción a Texto). Estos sistemas permiten a sus usuarios entrar a un sistema mediante un registro en concreto, para luego poder interactuar con el sistema de chatbots. Las interacciones que realizan los usuarios tienen una respuesta concreta en texto del sistema de chatbots. Los chatbots dentro del sistema pueden tener relaciones de flujo entre ellos para lograr una interacción más complementaria a lo que el usuario desee, además los usuarios pueden visualizar su historial de interacciones con el sistema.

1.2.- Descripción del paradigma

El paradigma lógico es un paradigma declarativo, su funcionamiento se basa hechos y diferentes reglas lógicas definidas. Además, se posee una base de conocimiento la cual es un archivo que contiene los diferentes predicados que se le asignen a un programa, estos se definen como lo que se quiere explicar y contienen diferentes hechos y reglas para ese fin.

Átomo: Son los elementos en los que se basa el conocimiento que se desea explicar. Su primera letra debe ser minúscula y debe ser una letra en lugar de cualquier otro elemento.

Consulta (?-): Son las consultas que se realizan al programa con el fin de encontrar respuestas. Estas pueden ser verdaderas (en caso de encontrar respuesta) y false (en caso de no encontrar respuesta), también se pueden entregar variables (se definen con primera letra mayúscula) dentro de las consultas con el fin de obtener un valor que satisfaga una consulta. Las consultas se realizan en la consola de SWI-Prolog y las respuestas son buscadas en la base de conocimientos del programa.

Predicado: Son aquellas cosas que se quieren explicar, representan propiedades o relaciones.

Hecho: Son aquellas cosas que se definen como verdades, establecen relaciones entre dominios. Son las bases de los predicados.

1.3.- Objetivos

Como objetivos del proyecto se tiene aprender y entender el paradigma lógico, mediante el lenguaje Prolog mediante el uso del IDE SWI-Prolog, además, la aplicación de estos conocimientos para dar solución a la problemática planteada como enunciado del laboratorio.

2.- Desarrollo

2.1.- Análisis de problema

Se tiene la entidad de sistema de chatbots, el cual almacenará los diferentes chatbots que el sistema necesite para su funcionamiento, los diferentes usuarios con sus respectivos historiales de interacción con el sistema de chatbots. Los chatbots en cuestión, pueden ser agregados al sistema, también pueden interactuar entre sí, esto lo realizan mediante una serie de flujos que estos poseen y permiten la interacción. Los predicados que debe aceptar el sistema para la simulación son:

option, flow, flowAddOption, chatbot, chatbotAddFlow, system, systemAddChatbot, systemAddUser, systemLogin, systemLogout, systemTalkRec, systemSynthesis, systemSimulate.

También se solicita que cada TDA utilizado este en un archivo propio.

2.2.- Diseño de solución

Al diseñar la solución se utilizaron diferentes tipos de datos: datos nativos del lenguaje y se crearon seis TDA que serán utilizados para la abstracción y simplificación del problema, estos TDA son:

TDA System:

- **Representación:** Una lista que posee el nombre del sistema como un string, un número con el ID del chatbot inicial, la lista de chatbots, la lista de users agregados al sistema, la lista de chatHistorys de los usuarios, dos números que representan el currentChabotID y el currentFlowID, finalmente se tiene un string con la fecha de creación del sistema.

- **Constructor:** Se entregan todos los elementos antes mencionados y una variable que almacenará la lista System, si solo se está creando un system por primera vez, solo se introduce el nombre, el ID del chatbot inicial y los chatbots que se deseen (pueden ser 0 o muchos), si se entregan cero chatbots, se entregará una lista vacía.

- Predicados: Anexo 1

TDA Option:

- **Representación:** Una lista, donde que posee un número que representa el ID único de la option (code), un string con el mensaje de la opción, dos números con el ID del chatbot al cual apunta dicha opción y el ID del flujo inicial de dicho chabot, finalmente se posee una lista con keywords que representan el distintivo de la option.

- **Constructor:** Se entregan todos los elementos antes mencionados y una variable que almacenará la lista Option, pero si no se desea agregar keywords, se entregan todos los elementos menos la lista de keywords antes mencionada, en lugar de ello se entrega una lista vacía.

- Predicados: Anexo 2

TDA Flow:

- **Representación:** Una lista, la cual posee un número correspondiente al ID único del flujo, un string correspondiente al nombre del flujo con su respectivo mensaje sobre las opciones que posee, finalmente una lista de options, las cuales representan las diferentes opciones que tiene un usuario para escoger mientras interactúa.

- **Constructor:** Se entregan todos los elementos mencionados anteriormente y una variable que almacenará la lista Flow, si solo se esta creando un flujo sin opciones, se ingresan todos los elementos mencionados anteriormente pero no se ingresan las opciones correspondientes al flujo, se ingresa una lista vacía.

- Predicados: Anexo 3.

TDA Chatbot:

- **Representación:** Una lista, la cual posee un número que representa el ID único del chatbot, dos strings que representan el nombre del chatbot y el mensaje de bienvenida del chatbot, un número que representa al flujo inicial del chabot, finalmente se tienen todos los flujos que posee el chatbot en cuestión.

- **Constructor:** Se entregan todos los elementos mencionados anteriormente y una variable que almacenará la lista Chatbot, si se crea un chatbot sin flujos, se ingresan todos los elementos mencionados anteriormente pero no se entregan los flujos del chatbot, se entrega una lista vacía.

- Predicados: Anexo 4.

TDA User:

- **Representación:** Una lista, donde el primer elemento es un string que representa el nombre del usuario, el segundo elemento es un int que representa el estatus del usuario en el sistema, si se encuentra iniciado o si su sesión está cerrada.

- **Constructor:** Se entrega el nombre del usuario que se desea agregar al sistema, un int representando el estado del usuario y una variable que almacenará la lista User.

- Predicados: Anexo 5.

TDA ChatHistory:

- **Representación:** Una lista, donde el primer elemento es un string que representa el nombre del usuario que interactúa con el sistema de chatbots, el segundo elemento es un string que representa el diálogo con fechas que posee el usuario con el sistema de chatbots.

- **Constructor:** Se entrega el nombre del usuario que interactuará con el sistema de chatbots, la interacción de este con el sistema y una variable que almacenará la lista ChatHistory.

- Predicados: Anexo 6.

Con la creación de las estructuras mencionadas anteriormente, se crea el sistema de chatbots. Cada estructura cumple un rol dentro del sistema, interactuando unas con otras para poder complementarse y generar el sistema de chatbots, para ver la relación entre las diferentes estructuras ver anexo 7.

Para agregar opciones a los flujos, flujos a los chatbots y chatbots al sistema, se utilizaron técnicas como: la recursión de cola, esto debido a la facilidad que está brinda para recorrer las diferentes listas.

2.3.- Aspectos de implementación

Para este laboratorio el compilador utilizado es SWI-Prolog, en versiones 9.0 o superiores. Se utilizan algunos predicados estándar brindados por el lenguaje y los predicados propios creados.

Se trabaja en su mayoría con listas, pero se crean diferentes predicados para poder acceder a estas. El programa creado es indiferente frente a mayúsculas y minúsculas, para el programa representan lo mismo, ósea el string “Museo” es igual a “MUSEO”, esto con el fin comparar case insensitive como es pedido.

El programa funciona sin validación de entradas de usuario, es decir, se asume que el usuario ingresará entradas válidas para el funcionamiento del programa.

Para llevar un mayor orden se distribuyen todos TDAs en archivos separados donde cada uno posee sus predicados correspondientes, para que los distintos archivos puedan tener acceso entre sí, se utiliza “**:-include(nombreArchivo).**” (el nombre debe ir en minúsculas y sin extensión) para importar y exportar predicados. También se tiene el archivo main que contiene todos los requerimientos funcionales implementados y el scriptDePruebas que es un archivo que posee los ejemplos de utilización de los requerimientos funcionales.

Los archivos mencionados anteriormente son: “main.pl”, “TDASystem.pl”, “TDAOption.pl”, “TDAFlow.pl”, “TDAChatbot.pl”, “TDAUser.pl”, “TDAChathistory.pl” y el “scriptDePruebas.pl” (con las identificaciones pedidas y en minúsculas).

2.4.- Ejemplos de uso

Se debe verificar que se tengan todos los archivos mencionados anteriormente, el archivo main necesita a todos los archivos TDAs implementados.

El programa creado cuenta con un formato especifico para poder se usado de manera correcta, sin que este presente errores, para ello, los usuarios deben usar de manera correcta los requerimientos implementados. El programa implementado no distingue entre mayúsculas y minúsculas, el texto es el mismo, además, lo usuarios deben ingresar entradas correctas debido a que no se realizan validaciones de entradas de usuario.
Para ver un ejemplo de uso del programa, ver anexo 8.

2.5.- Resultados y autoevaluación

Los resultados obtenidos son los esperados para la mayoría de los requerimientos implementados, logrando hacer un programa que cumple de manera correcta lo pedido. Programa en el cual se colocan diferentes entradas posibles, o utilizando predicados cuando no corresponde y el programa responde de manera correcta exceptuando los requerimientos que no fueron implementadas.

Se logran completar 13 requerimientos funcionales propuestos para el proyecto.

La autoevaluación radica entre valores de 0 (no implementado) hasta 1 (implementado y sin errores) aumentando en una escala de 0.25 y se puede observar en el anexo 9.

Se les asigna 1 a los requerimientos que trabajan de manera correcta, y se les asigna 0.5 a los requerimientos que trabajan de manera no completa (cumplen su objetivo, sin embargo, no realizan todos los cambios que deberían o no cumplen todos los requisitos pedidos)

3.- Conclusión

Una vez terminado el proyecto de laboratorio, se puede concluir que se cumple con los objetivos principales de este, debido a que se logra entender de mejor manera el paradigma lógico, así como también se aprende a utilizar Prolog mediante SWI-Prolog para dar solución a la problemática planteada como enunciado del proyecto.

Durante la realización del laboratorio una de las complicaciones más grandes fue que al estar trabajando con un nuevo paradigma había cosas que se realizaban de manera distinta a paradigmas y lenguajes vistos anteriormente, como el paradigma funcional. Una de ellas es la recursión, la forma en la que debe implementarse en Prolog es poco intuitiva, se define que cada predicado es un condicional para la recursión. Se tuvieron variadas dificultades a la hora de recorres las diferentes listas con recursión, el programa entregaba false en reiteradas ocasiones, por ello se debió realizar la traza del programa en aquellas situaciones, sin embargo, esto fue lo que ayudó a entender de mejor manera el paradigma y el lenguaje.

Otra complicación que se tuvo es que a diferencia de la programación funcional en Scheme, al realizar recursiones, en caso de haber un error este entregaba el error en cuestión, en cambio la programación lógica en Prolog entrega False al no poder encontrar respuesta. Esto llevo a utilizar bastante tiempo en verificar errores y entender la lógica del lenguaje.

4.- Bibliografía

Flores, V. (2023). *Paradigmas de Programación Proyecto Semestral de Laboratorio.* Recuperado de https://docs.google.com/document/d/1QCk3k-HCShNSByEZHoEIFbrxoOJSnPblyjiJDj3Q9Jk/edit

Flores, V. (2023). "4 - P. Lógico". Paradigmas de Programación. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=16

5.- Anexos

1.- Predicados implementados del TDA System (Los requerimientos funcionales se dejan dentro del archivo “main.pl” como es pedido):

Tipo de Predicado	**Nombre**	**Descripción**
Constructor	system	Crea un System
Selector	getSystemChatbots	Obtiene la lista de Chatbots del System
Selector	getSystemUsers	Obtiene la lista de Users del System
Selector	getSystemChatHistorys	Obtiene la lista de ChatHistorys del System
Selector	getSystemCurrentChatbotID	Obtiene el CurrentChatbotID del System
Selector	getSystemCurrentFlowID	Obtiene el CurrentFlowID del System
Selector	getSystemInitialChatbotCodeLink	Obtiene el InitialChatbotCodeLink del System
Modificador	setSystemStartFlowID	Modifica el StartFlowID del System
Modificador	setSystemChatbots	Modifica los Chatbots del System
Modificador	setSystemUsers	Modifica los Users del System
Modificador	setSystemChatHistorys	Modifica los Chathistorys del System
Modificador	setSystemChathistorysAnd CurrentFlowAndChatbotIDs	Modifica el CurrentFlowID, CurrentChatbotID y ChatHistorys del System
Modificador	systemAddChatbot	Agrega un Chatbot a un System
Modificador	systemAddUser	Agrega un User a un System
Modificador	systemLogin	Inicia la sesion de un User dentro de un System
Modificador	systemLogout	Cierra la sesion de un User dentro de un System
Modificador	systemTalkRec	Permite al User interactuar con un Chatbot
Otros predicados	addChatbotToChatbots	Agrega un Chatbot a una lista de Chatbots
Otros predicados	addUserToUsers	Agrega un User a una lista de Users
Otros predicados	addChatHistoryToChatHistorys	Agrega un Chathistory a una lista de Chathistorys
Otros predicados	makeNewInteraction	Crea el mensaje de interaccion entre el usuario y el sistema
Otros predicados	systemSynthesis	Obtiene el Historial de interacciones de un usuario y el sistema

2.- Predicados del TDA Option:

Tipo de Predicado	Nombre	Descripción
Constructor	option	Crea una Option
Selector	getOptionCode	Obtiene el Code de la Option
Selector	getOptionKeyword	Obtiene las Keywords de la Option
Selector	getOptionChatbotCodeLink	Obtiene el ChatbotCodeLink de la Option
Selector	getOptionInitialFlowCodeLink	Obtiene el InitialFlowCodeLink de la Option
Selector	getOptionMessage	Obtiene el Message de la Option
Selector	getChatbotAndFlowIDsByOption	Obtiene el ChatbotID y FlowID de la Option
Selector	getOptionsCodes	Obtiene todos los Codes de una lista de Options
Selector	getOptionByMessage	Obtiene una Option dada una lista de Options y un Message
Selector	getChatbotAndFlowCodeLinks ByMessage	Obtiene el ChatbotCodeLink y InitialFlowCodeLink de una Option
Selector	getOptionsMessages	Obtiene todos los Message de una lista de Options
Pertenencia	codeExist	Verifica que los codes dentro de una lista sean unicos
Pertenencia	uniqueOptions	Verifica que todas las Options de una lista sean distintas
Pertenencia	optionExist	Verifica si una Option existe en una lista
Otros predicados	keywordsToLowerKeywords	Transforma todas las Keywords a minusculas
Otros predicados	optionToString	Transforma una lista de Strings (Messages de Option) a un String con un formato
Otros predicados	optionsToString	Transforma una lista de Options a un String con un formato

3.- Predicados del TDA Flow:

Tipo de Predicado	Nombre	Descripción
Constructor	flow	Crea un Flow
Selector	getFlowOptions	Obtiene las Options del Flow
Selector	getFlowID	Obtiene el ID del Flow
Selector	getFlowNameMsg	Obtiene el NameMsg dek Flow
Selector	getFlowsIDs	Obtiene todos los IDs de una lista de Flows
Selector	getFlowByID	Obtiene un Flow dada una lista de Flows y un ID para identificarlo
Pertenencia	uniqueFlows	Verifica si una lista posee Flows unicos
Pertenencia	flowIdExist	Verifica si una lista posee IDs unicos
Pertenencia	myMember	Verifica si un Elemento esta dentro de una Lista
Pertenencia	flowExist	Verifica si un flow existe en una lista de flows
Modificador	setFlowOptions	Modifica la lista de Options de un Flow
Modificador	flowAddOption	Agrega una Option a un Flow
Otros predicados	addOptionToOptions	Agrega una Option a una lista de Options

4.- Predicados del TDA Chatbot:

Tipo de Predicado	Nombre	Descripción
Constructor	chatbot	Crea un Chatbot
Selector	getChatbotID	Obtiene el Id del Chatbot
Selector	getChatbotName	Obtiene el Name del Chatbot
Selector	getChatbotFlows	Obtiene los Flows del Chatbot
Selector	getChatbotStartFlowID	Obtiene el StartFlowID del Chatbot
Selector	getChatbotsIDs	Obtiene todos los ID de los Chatbots dentro de una lista de Chatbots
Selector	getStartFlowIDBy InitialChatbots	Obtiene el IntialFlowID dada una lista de Chatbots y un InitialChatbotID
Selector	getChatbotByID	Obtiene un chatbot dado su ID y una lista de chatbots
Pertenencia	uniqueChatbots	Verifica si hay un Chatbot repetido en una lista de Chatbots
Pertenencia	chatbotIdExist	Verifica si hay un ID repetido en una lista de IDs
Pertenencia	chatbotExist	Verifica si un Chatbot existe en una lista de Chatbots
Modificador	setChatbotFlows	Modifica la lista de Flows de un Chatbot
Modificador	chatbotAddFlow	Agrega un Flow a un Chatbot
Otros predicados	addFlowToFlows	Agrega un Flow a una lista de Flows
Otros predicados	isInitialChatbot	Verifica si un Chatbot corresponde al Chatbot asociado a el InitialChatbotCodeLink

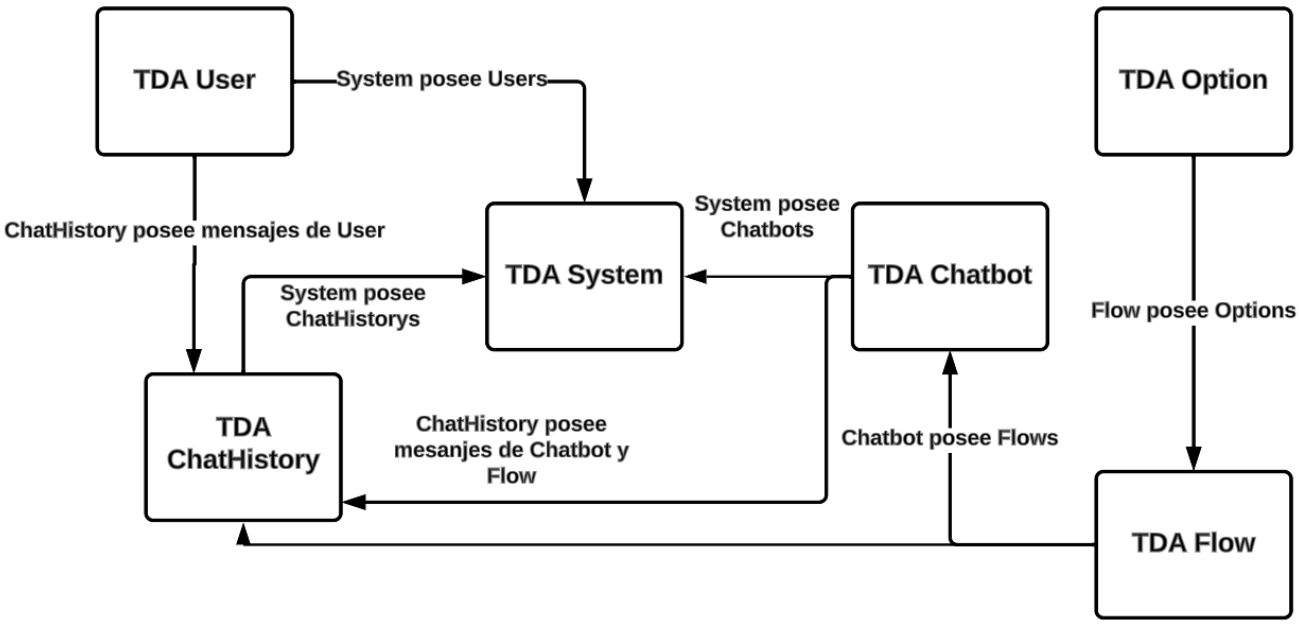
5.- Predicados del TDA User:

Tipo de Predicado	Nombre	Descripción
Constructor	user	Crea un usuario
Selector	getUserName	Obtiene el Name del User
Selector	getUserStatus	Obtiene el estatus del User
Selector	getUsersIDs	Dada una lista de Users obtiene todos los nombres de los Users
Selector	getUsersStatus	Dada una lista de Users obtiene todos los estatus de los Users
Selector	getUserLogged	Dada una lista de User obtiene al User con sesión iniciada
Pertenecia	userExist	Verifica si un User existe dentro de una lista de Users
Pertenecia	usersLogged	Verifica si hay usuarios iniciados
Modificador	changeUserStatus	Cambia el estatus de un User correspondiente
Modificador	changeUsersStatus	Dada una lista de User cambia el estatus de un User en concreto

6.- Predicados del TDA ChatHistory:

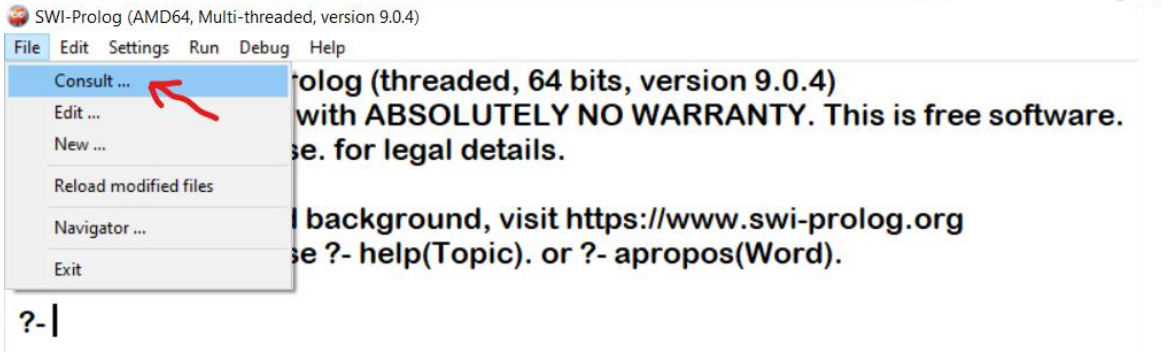
Tipo de Predicado	Nombre	Descripción
Constructor	chatHistory	Crea un ChatHistory
Selector	getChathistoryUser	Obtiene el Username del Chathistory
Selector	getChathistoryHistory	Obtiene el Historial del Chathistor
Selector	getHistoryByUsername	Obtiene un Chathistory dado un Username y una lista de Chathistorys
Modificador	updateChatHistory	Actualiza un Chathistory cambiando el Historial del Chathistory
Modificador	changeChathistorys	Actualiza el Chathistory asociado a un User agregando contenido al Historial

7.- Relación entre estructuras:



8.- Ejemplos de uso:

8.1. Abrir el archivo con el IDE SWI-Prolog para realizar consultas.



presionar el archivo: pruebas.pl

8.2. Ingresar la primera consulta:

set_prolog_flag(answer_write_options,[max_depth(0)]).

Se realiza está consulta para que el IDE muestre el contenido completo de las listas.

8.3. Realizar las consultas correspondientes a la simulación, por ejemplo:

option(1, "1) Viajar", 1, 1, ["viajar", "turistear", "conocer"], OP1),

option(2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"], OP2),

flow(1, "flujo1", [OP1], F10),

flowAddOption(F10, OP2, F11),

chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F11], CB0), %solo añade una ocurrencia de F11

system("Chatbots Paradigmas", 0, [CB0], S0),

systemAddUser(S0, "user1", S01),

systemLogin(S01, "user1", S02),

systemTalkRec(S02, "hola", S03),

systemTalkRec(S03, "1", S04),

systemSynthesis(S04, "user1", History),

write(History).

```
?- option(1, "1) Viajar", 1, 1, ["viajar", "turistear", "conocer"], OP1),
option(2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"], OP2),
flow(1, "flujo1", [OP1], F10),
flowAddOption(F10, OP2, F11),
chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F11], CB0), %solo añade una ocurrencia de F11
system("Chatbots Paradigmas", 0, [CB0], S0),
systemAddUser(S0, "user1", S01),
systemLogin(S01, "user1", S02),
systemTalkRec(S02, "hola", S03),
systemTalkRec(S03, "1", S04),
systemSynthesis(S04, "user1", History),
| | | | | | | | | | write(History).|
```

(Ignorar los “|”).

Esto entregará como respuesta:

```
1699750937.244862 - user1: hola
1699750937.244862 - Inicial: flujo1
1) Viajar
2) Estudiar

1699750937.24488 - user1: 1
1699750937.24488 - Inicial: flujo1
1) Viajar
2) Estudiar

OP1 = [1,1) Viajar,1,1,[viajar,turistear,conocer]],
OP2 = [2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]],
F10 = [1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]]]],
F11 = [1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]],
CB0 = [0,Inicial,Bienvenido
¿Qué te gustaría hacer?,1,[[1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]]],
S0 = [Chatbots Paradigmas,0,[[0,Inicial,Bienvenido
¿Qué te gustaría hacer?,1,[[1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]]]],[],[],
0,1,1699750937.24483],
S01 = [Chatbots Paradigmas,0,[[0,Inicial,Bienvenido
¿Qué te gustaría hacer?,1,[[1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]]]],[[user
1,0]],[[user1,]],0,1,1699750937.24483],
S02 = [Chatbots Paradigmas,0,[[0,Inicial,Bienvenido
¿Qué te gustaría hacer?,1,[[1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]]]],[[user
1,1]],[[user1,]],0,1,1699750937.24483],
S03 = [Chatbots Paradigmas,0,[[0,Inicial,Bienvenido
¿Qué te gustaría hacer?,1,[[1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]]]],[[user
1,1]],[[user1,1699750937.244862 - user1: hola
1699750937.244862 - Inicial: flujo1
1) Viajar
2) Estudiar
```

```
]],0,1,1699750937.24483],
S04 = [Chatbots Paradigmas,0,[[0,Inicial,Bienvenido
¿Qué te gustaría hacer?,1,[[1,flujo1,[[1,1) Viajar,1,1,[viajar,turistear,conocer]],[2,2) Estudiar,2,1,[estudiar,aprender,perfeccionarme]]]]]],[[user
1,1]],[[user1,1699750937.244862 - user1: hola
1699750937.244862 - Inicial: flujo1
1) Viajar
2) Estudiar

1699750937.24488 - user1: 1
1699750937.24488 - Inicial: flujo1
1) Viajar
2) Estudiar

]],0,1,1699750937.24483],
History = 1699750937.244862 - user1: hola
1699750937.244862 - Inicial: flujo1
1) Viajar
2) Estudiar

1699750937.24488 - user1: 1
1699750937.24488 - Inicial: flujo1
1) Viajar
2) Estudiar

.
```

Existen más ejemplos de uso en el archivo “Pruebas.pl” donde se puede apreciar como hacer un correcto uso del programa.

9.- Autoevaluación requerimientos funcionales y no funcionales:

Autoevaluación	Puntaje
TDA	1
option	1
flow	1
flowAddOption	1
chatbot	1
chatbotAddFlow	1
system	1
systemAddChatbot	1
systemAddUser	1
systemLogin	1
systemLogout	1
systemTalkRec	1
systemSynthesis	1
systemSimulate	0

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Autoevaluación	Puntaje
Autoevaluación	1
Lenguaje	1
Versión	1
Standar	1
Documentación	1
Organización	1
Historial	1
Script de pruebas	1
Prerrequisitos	1