Instituto Tecnológico de Costa Rica

Área Ingeniería en Computadores

Lenguajes, compiladores e intérpretes.

CE - 3104

Tarea 2

Paradigma de programación lógico

Profesor:

Marco Rivera Meneses

Estudiantes:

Isaac Araya Solano - 2018151703

Jose Pablo Fuentes Apú - 2018112078

Brayan León Urbina - 2018234632

I Semestre

2020

1. **Reglas implementadas**

1.1 leer(Persona): Regla que tiene como objetivo leer los datos que el usuario escribe en la consola para convertirlos en en una lista que posee las palabras escritas. El parámetro persona es la lista de los atributos del personaje que se está construyendo.

1.2 leer(Persona,Sus,Atri): Regla que tiene como objetivo leer los datos que el usuario escribe en la consola para convertirlos en en una lista que se utilizan en las preguntas de “Si o no”. El parámetro sustantivo se refiere al sustantivo o característica del personaje a tratar, mientras que el parámetro Atri se refiere al valor del atributo de la pregunta.

1.3 afirmar ([A|_]: Regla que detecta una respuesta afirmativa. Se basa en el hecho afirmaciones.

1.4 limpiar_oraciones([ ], [ ],: Regla que recibe una lista de palabras en el primer parámetro y una lista vacía en el segundo.

1.5 lectura(Personas, Datos, Sustantivos, Atri): Regla que divide los datos escritos por el usuario en adjetivos, verbos y sustantivos. Construye una lista de los atributos del personaje que se está tratando de adivinar. El parámetro persona es una lista de la persona que se está construyendo. Sustantivo, es la característica del personaje al que se refiere la pregunta y atri, es el valor del atributo de la pregunta.

1.6 sacar_adjetivos ([ ], [ ]): Crea una lista de adjetivos encontrados que coinciden con la base de datos del programa . La primera entrada es una lista de palabras y la segunda, una lista de adjetivos.

1.7 sacar_verbos ([ ], [ ]): Regla que crea una lista de verbos encontrados, a partir de una lista de palabra,s que coincidan con la base de datos del programa. Recibe como primer entrada una lista de palabras y como segunda, una lista de verbos.

1.8 sacar_sustantivos ([ ], [ ]): Regla que crea una lista de verbos encontrados, a partir de una lista de palabra,s que coincidan con la base de datos del programa. Recibe como primer entrada una lista de palabras y como segunda, una lista de sustantivos

1.9 miembro (X, [X|_]) Indica si el elemento X que recibe como parámetro pertenece a la lista que recibe como segundo parámetro.

1.10 longitud ([ ], 0) Determina la longitud de la lista que recibe como primer parámetro. El segundo parámetro se usa para indicar la longitud de la lista.

1.11 obtener_pos(0, [A|_], A): Obtiene el elemento de una lista. Recibe como entradas el índice del elemento, la lista de datos en la que se quiere buscar el elemento y el valor del elemento en la posición indicada.

1.12 agregar (X,L, [X|L]): Agrega la entrada X a la lista L que recibe como entradas. La tercer entrada es la lista L con el elemento X agregado.

1.13 inversa (L1, L): Invierte la lista L1 que recibe como entrada. La segunda entrada L es la lista invertida.

1.14 cambiar_dato (0, A, [_ | D], [A | C]) : Tiene como objetivo cambiar de posición un elemento. La primera entrada es la nueva posición del elemento que recibe en la segunda. La tercera entrada es la lista en donde se desea modificar la posición del elemento y la cuarta, es la lista con el elemento ya cambiado.

1.15 lista_x_ele_repetidos (X, B, E). Crea una lista con de n cantidad de un solo elemento. X es el largo de la lista. E: es la lista creada de n cantidad de elementos. B es el elemento que se re repite.

1.16 posiciones_dato (A, L, Ln): Obtiene las posiciones en las que se encuentra un dato en una lista. A es el dato a buscar, L es el la lista donde se desar buscar el dato y Ln es la lista con el número de las poscioes en las que se encuentra el dato.

1.17 obtener_verb (Verb, C) Obtiene la referencia al que está ligado el verbo. Verb es el verbo del que se obtiene la referencia y C es la referencia a ese mismo verbo.

1.18 obtener_sin (Sin, C): Obtiene la referencia al que está ligado un sustantivo. Sin es el sustantivo del que se obtiene la referencia y C es la referencia a ese mismo sustantivo.

1.19 obtener_ad (Adj, C): Obtiene la referencia al que está ligado un adjetivo. Adj es el adjetivo del que se obtiene la referencia y C es la referencia a ese mismo adjetivo.

1.20 verificar(Persona,_,_, [ ]): Tiene como objetivo ir armando el personaje que desea conocer el personaje. La primer entrada es una lista de atributos en las que se van a agregar los atributos para encontrar al personaje. La segunda entrada es la lista de adjetivos a los que se refieren los adjetivos. La tercer entrada es la lista de verbos que se refieren a sustantivos y la cuarta, es la lista de adjetivos a comparar y encontrar similitud con los personajes de la base de datos.

1.21 verificar_verb (Persona, [ ], Adjetivos, Persona, Adjetivos): Verifica la lista de verbos convirtiéndolos en sustantivos encontrados en la lectura de la oración con los adjetivos encontrados. Persona es la lista con las características del personaje que se está construyendo. La segunda entrada es la lista de adjetivos restantes a los que no se le encontró referencia.

1.21 verificar_sin (Persona, [ ], Adjetivos, Persona, Adjetivos): Verifica la lista de sinónimos encontrados en la lectura de la oración con los adjetivos encontrados. Persona es la lista con las características del personaje que se está construyendo. La segunda entrada es la lista de sustantivos a los que se refieren los adjetivos. Adjetivos es la lista de adjetivos a comparar y encontrar similitud en la base de datos.

1.22 comprobar(Persona, [Sin|Rest], Sustantivo, [Adj | _], Adj, Newp): Su tarea es buscar y comparar todos los adjetivos encontrados con los personajes establecidos para construir al personaje pensado. Persona se refiere a la lista con las características del personaje que se está construyendo. [Sin|Rest] es la lista de sustantivos restantes, Sustantivo es el sustantivo a evaluar con la lista de adjetivo.

1.23 comprobacion_listas(P): Comprueba si el personaje construido concuerda completamente con alguno de los personajes de la base de datos. En caso de que alguno concuerde, se indica cual ese. La entrada P se refiere a la lista con las características del personaje que se está construyendo.

1.24 comprobacion_individual([D|C],[D|L]): Su tarea es derterminar si los atributos de un personaje pertenecen a algún personaje establecido en la base de datos.

1.24 obtener_dato_random(Ln,Dato): Saca un dato aleatorio de la lista. Ln es la lista de donde obtiene el dato y dato es donde se guarda el dato de la lista.

1.24 preguntar(Persona): Realiza una pregunta normal al usuario para ir descubriendo al personaje. Persona es la lista con los datos de la persona que se va descubriendo.

1.25 jugar: Función a llamar para empezar el juego

2. Estructuras de datos desarrolladas.

La estructura de datos que se desarrolló es muy sencilla pero efectiva, primero se definió a los personajes como:

personaje(Nombre,[atributo1,atributo2….]).

De esta manera siempre los datos van a tener un mismo orden, lo cual va a ser necesario para los algoritmos de solución, luego se define un nombre clave para cada posición de los atributos, junto la posición en la que se encuentran:

atributo_posicion(clave_atributo1,0).

ya con ello existe una alta facilidad para acceder a los atributos directamente. después de esto se definen palabras de referencia para los tres conjuntos grandes de palabras que se definen en la progra, los adjetivos (son los valores de los atributos de cada personaje, pueden o no ser adjetivos como tal), los sustantivos

(son la palabra clave de cada atributo, y se encuentran en atributo_posicion) y los verbos (estos son los verbos que dan lugar a ciertas claves)

referencia_X(clave,[sinonimo1,sinonimo2]).

con ello ya sin importar la oración se puede sacar las palabras claves de lo que el usuario escribió y para finalizar con las estructuras de datos desarrolladas para la lógica los verbos son convertidos a sustantivos, para manejarlos como como tal en la lógica, para ello se le dan ciertas referencias a palabras claves y tiene la forma de:

referencia_atributos(verbo,[sustantivo1,sustantivo2….).

para las preguntas se usan estas 2 estructuras, una para elegir el tipo de pregunta que se le va a realizar al usuario con la forma

pregunta(sustantivo,[pregunta1,pregunta2]).

y para las boolenas se debe también elegir una característica del sustantivo al que referir en la pregunta:

adj_preguntas(sustantivo1,[atributo11,atributo12]).

y con esto se puede aplicar los algoritmos desarrollados en la solución del problema.

3.Descripción de los algoritmos utilizados.

se desarrolló un algoritmo lógico y recursivo donde se realizan varias pasadas de lectura de oraciones en consola, con un personaje vacío (["","","",""]), luego se crea una lista de las palabras dichas por el usuario y se manda a corroborar su español con el bnf, después se separan las palabras de acuerdo a las estructuras de datos mostradas anteriormente en verbos,

sustantivos y adjetivos, que ya han sido explicadas en estructuras de datos también, con ello en verificar_sin() primero se agarra la persona y se le añade al primer sustantivo de la lista cada adjetivo y se va probando coincidencias con los personajes, hasta encontrar alguna o ninguna, si se encuentra, se elimina el adjetivo de la lista y pasa al siguiente sustantivo, hasta terminar con todos los sustantivos y en consecuencia continúa con los verbos en verificar_verb() el cual unicamente convierte los verbos en sustantivos y vuelve a llamar a verificar_sin() para volver a realizar los mismos pasos.

Si no se encuentra sustantivos para todos los adjetivos se da como erróneo el personaje creado y se vuelve a iniciar de nuevo desde el principio, si no se manda a preguntar otra vez al usuario, con la persona construida en esta pasada, hasta tener todos los atributos de la persona y que coincida con alguno de los personajes de la base de datos, en este punto se le indicará al usuario el personaje en que estaba pensado y se iniciara el juego otra vez.

4.Problemas sin solución.

Sin duda alguna, un evidente problema es el hecho de que no se podrá incluir el lenguaje español en su totalidad y se tiene que incluir una forma muy básica del lenguaje. Esto se debe a la complejidad del español y su enorme cantidad de conjugaciones para cada uno de los verbos del idioma. A esto se suma también, la gran cantidad de de proposiciones que posee el idioma.

5.Plan de actividades.

Feature	Feature	Semana	Tareas	Horas
1	Investigación	1	Investigar a posibles personaje que tienen influencia en la población costarricense	2
2	Establecer datos	1	Establecer cada uno de los atributos que tendra cada personaje de la base de datos adaptandose a la investigación realizada previamente	2
3	Creación de la base de datos	2	Una vez realizada la investigacion, procede a la realización de la estructura de datos que se va a utilizar como base.	3
4	Establecer sinónimos	2	Se establece una estructura de datos en las que se colocan sinonimos de los atributos de cada personaje y de diferentes verbos, sutantivos y proposiciones	2
5	Leer datos enviados por el usuario	1	Hacer que el programa reconozca los datos que el usario porne en consola	3
6	Identificar el personaje	2	Realizar la lógica que permita identificar el usuario a partir de los atributos que ingreso el usuario.	5
7	Definir preguntas	2	Establecer la lógica para que el programa formule preguntas al usuario basadas en los atributos de los personajes.	4
8	Investigacion	1	Investigar sobre la gramatica libre de contexto en español y utilizando el lenguaje PROLOG	2
9	Definir BNF	2	Implemetacion del BNF en el archivo PROLOG	4
10	Estructura de oraciones	2	Establecer la estructura de las oraciones. Para esto se requería implementar el uso de sujetos y predicados asi como los verbos, adverbios, proposiciones y adjetivos	6

Persona	Tareas
Jose	1,2,3,4
Isaac	8,9,10
Brayan	5,6,7

Horas	
Brayan	12
Jose	9
Isaac	12

6.Problemas encontrados.

Los problemas encontrados se basan principalmente en el BNF y la capacidad de entender lo que el usuario escribió. Esto se debe a, como se anteriormente, el lenguaje español posee una gran cantidad de verbos (con muchas formas de conjugarlos), proposiciones y ambigüedades. Este tipo de problemas se pudieron resolver agregando más vocabulario (más verbos, más proposiciones, adverbio, etc) a la base de datos del juego.

El programa no detectaba cuando el usuario escribía dos oraciones usando una coma, la solución más práctica fue quitandolas por medio de una comparación usando la regla limpiar_oraciones([ ]).

7.Conclusiones y recomendaciones

A pesar de los problemas y la complejidad del lenguaje español, se desarrolla una aplicación que se comporta como un sistema Experto utilizando el lenguaje de programación PROLOG.

Para su realización se aplican los fundamentos de lógica basada en las cláusulas de Horn, así como las instrucciones definidas por el lenguaje de paradigma lógico PROLOG.

Como parte de las recomendaciones, se promueve a crear una extensa base de datos con la mayor cantidad de verbos, sustantivos, adverbios y preposiciones, así como diversas estructuras de oraciones para que el sistema le sea más facil emitar una conversación fluida con el usuario,

8.Bibliografía

https://www.swi-prolog.org/pldoc/man?section=strings

https://www.swi-prolog.org/pldoc/man?predicate=random/3

https://www.swi-prolog.org/pldoc/doc/_SWI_/library/readln.pl

https://www.swi-prolog.org/pldoc/man?predicate=write/1

9. Bitácora

• 21/05/2020: Isaac inicia la investigación sobre la gramática libre de contexto en prolog.

• 21/05/2020: Jose inicia la investigación sobre los personajes que conforman nuestra base de datos.

• 22/05/2020:Brayan empieza a realizar la lógica para leer los datos que el usuario escribe en la consola

• 26/05/2020:Jose establece los datos que llevará cada uno de los personajes que integran la base de datos

• 27/05/2020:Bryan Empieza a realizar la lógica para definir al personaje a partir de los atributos que el usuario coloca en consola.

• 28/05/2020:Isaac luego de la investigación implementa BNf utilizando el lenguaje de programación PROLOG.

• 29/05/2020: Jose crea la base de datos final, con la que Brayan se basa para el reconocimiento final de los personajes..

•29/05/2020: Isaac empieza a definir la estructura de las oraciones .

•30/05/2020: Jose amplía la base de datos para que el sistema reconozca aún más sinónimos.

• 31/05/2020: Se empieza a unir cada una de las partes para que el programa funcione de la manera esperada

• 1/06/2020: Se empieza a depurar en conjunto cada uno de los errores del programa.