UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Ingeniería en Software y Tecnologías Emergentes

Paradigmas de la programación

Práctica 4

ALUMNO: Cesar Alejandro Velazquez Mercado
MATRÍCULA: 372329

GRUPO: 941

PROFESOR: Carlos Gallegos

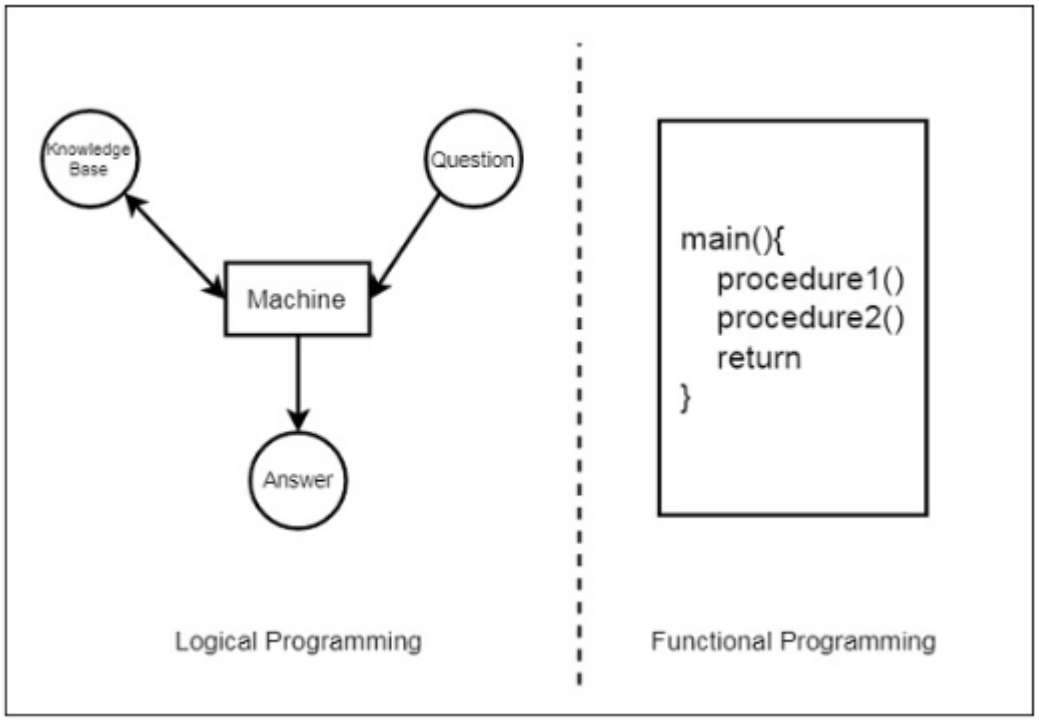
30 de mayo del 2024

1. Prolog - Home

a. Empezamos en “Home” donde se nos da una pequeña introducción al tutorial, aquí se nos explica que Prolog es un lenguaje de programación lógico y declarativo y un gran ejemplo de un lenguaje de cuarta generación, sumado a esto se nos dice que el tutorial fue hecho para principiantes con un conocimiento previo de programación.

2. Prolog - Introduction

a. Prolog escribe las reglas con cláusulas lógicas donde cabeza y cuerpo están presentes. Por ejemplo, H es cabeza y B1, B2, B3 son cuerpo.



Para esta imagen vemos que la programación funcional trabaja paso a paso para resolver los problemas, en cambio, la programación lógica trabaja usando su conocimiento base para responder preguntas

Diferencias:

i. Programación Funcional
   1. La Programación Funcional sigue la arquitectura de Von-Neumann o utiliza pasos secuenciales.
   2. La sintaxis es en realidad la secuencia de declaraciones como (a, s, l).
   3. La computación se lleva a cabo ejecutando las declaraciones secuencialmente.
   4. La lógica y los controles están mezclados.

ii. Programación Lógica

1. La Programación Lógica utiliza un modelo abstracto o trata con objetos y sus relaciones.
2. La sintaxis es básicamente las fórmulas lógicas.
3. Computa deduciendo las cláusulas.
4. Las lógicas y los controles pueden separarse.

3. Prolog - Environment Setup

a. Usaremos la versión 1.4.5 así que entramos a la página para descargarlo http://www.gprolog.org/ y esto es lo que veremos:

The GNU Prolog web site

Current stable version is gprolog-1.5.0

Table of contents

Bajamos hasta encontrar los links para descargar

Download

We provide both source and binary distributions for GNU Prolog.

Source distributions:

- the main source distribution gprolog-1.5.0.tar.gz.

Binary distributions:

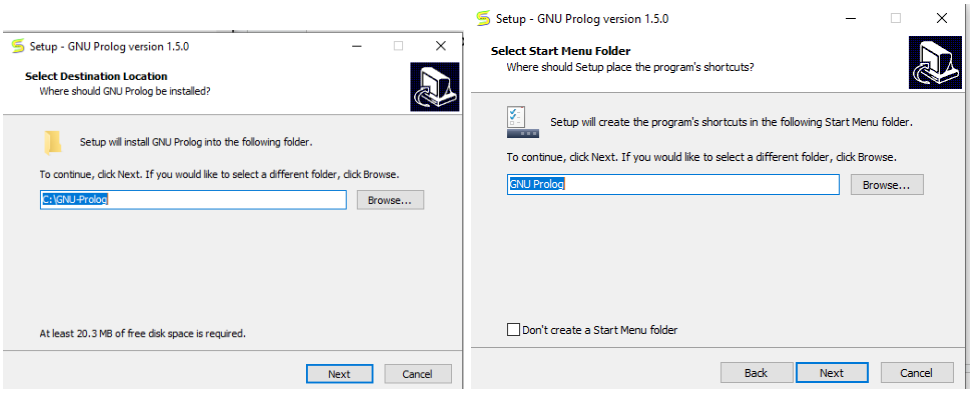
- Mac OS X installer package created on Big Sur using MacPorts by Paulo Moura (installs GNU Prolog in /opt/local/ and /opt/local/bin).

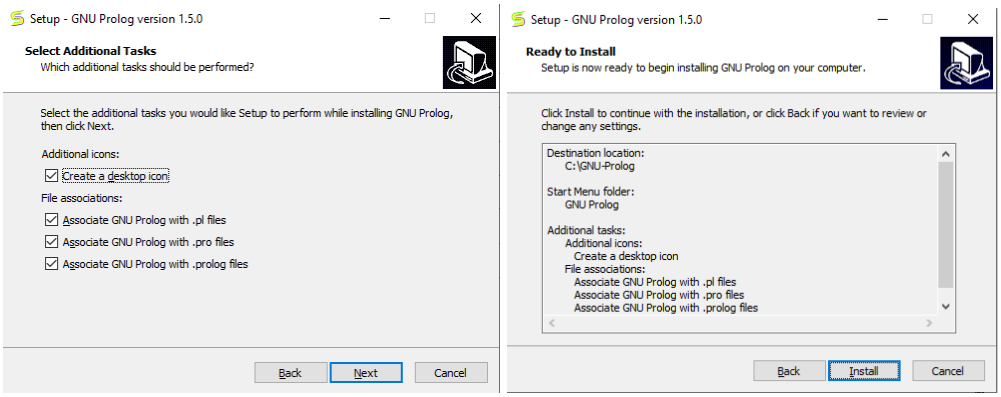
- Windows intel 32 bits auto-install setup (compiled under ix86 / Windows 10 with MSVC++).
- Windows intel 32 bits auto-install setup (compiled under ix86 / Windows 10 with MinGW gcc under MSys2).

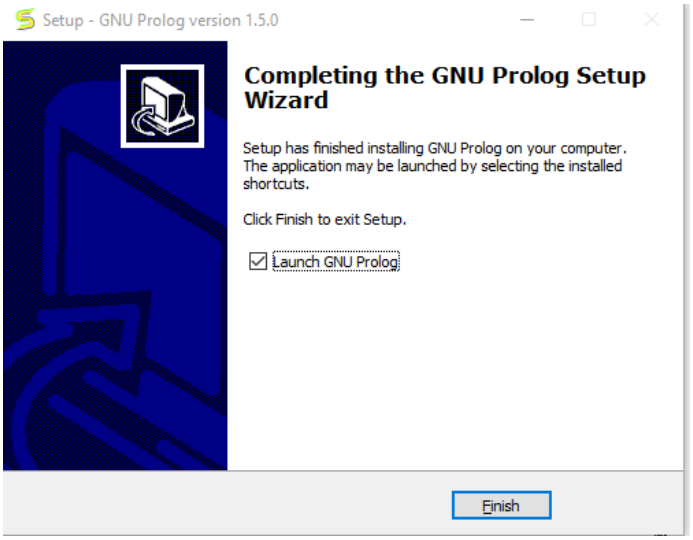
- Windows intel 64 bits auto-install setup (compiled under x86_64 / Windows 10 with MSVC++).
- Windows intel 64 bits auto-install setup (compiled under x86_64 / Windows 10 with MinGW64 gcc under MSys2).

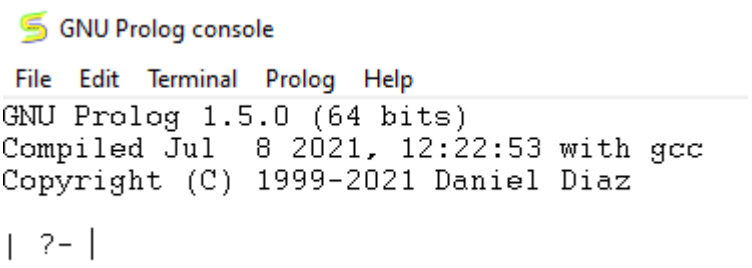
Other versions:

Descargamos y seguimos los pasos:





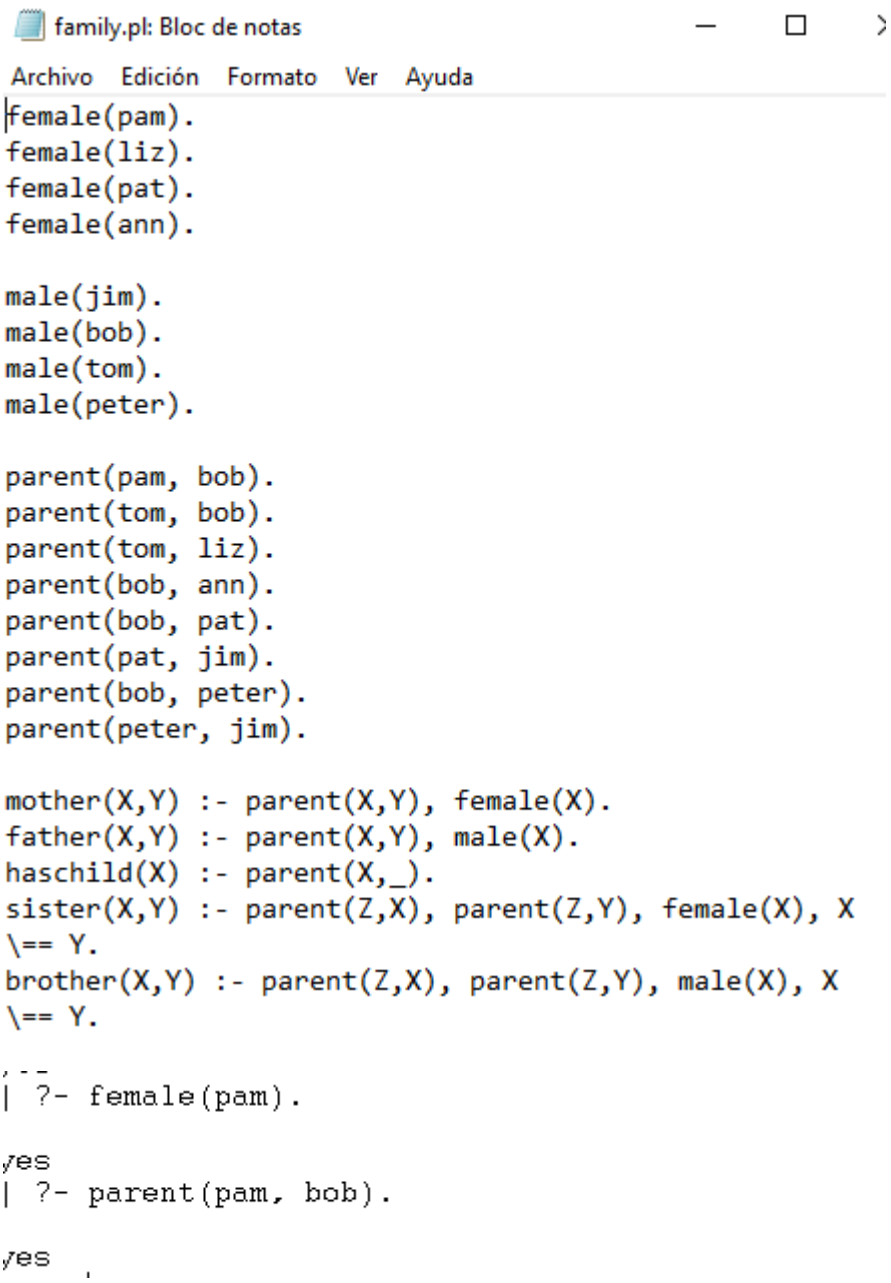




Y sé descargo.

4. Prolog - Hello World
    a. Comenzamos escribiendo “write('Hello World').” En la plataforma que nos despliega “Hello World”, ahora vamos a ver como correr el Prolog script file, la Prolog consola.

        i. Primero desde el programa vamos a File > Change Dir, Y hacemos click en el menú para después seleccionar un folder y presionar OK. esto nos despliega el mensaje: change_directory('C:/Users/costco/paradigmas/Practica 4').
        ii. Ahora vamos a crear una extensión para el archivo con estas líneas de código:
        main :- write('This is sample Prolog program'),
        write(' This program is written into hello_world.pl file').
        iii. Y lo corremos con [hello_world]
5. Progol - Basics
    a. Aquí vemos varios ejemplos de las bases de programación lógica como
        i. girl(priya).
        ii. girl(tiyasha).
        iii. girl(jaya).
        iv. can_cook(priya).
    b. Ahora tenemos un ejemplo de base de conocimiento
        i. girl(priya).
            1. yes
        ii. girl(tiyasha).
            1. no
        iii. girl(jaya).
            1. yes
        iv. can_cook(priya).
            1. no
6. Prolog - Relations
    a. En los programas Prolog se especifican las relaciones entre objetos y las propiedades de los objetos un ejemploo de relación es:
        i. Relación:
            1. Ambos son hombres
            2. Ambos tienen el mismo padre
        ii. Respuesta
            1. Padre(Sudip,piyus).
            2. Padre(sudip, raj).
            3. Hombre(piyus).
            4. Hombre(raj).
            5. Hermanos(X, Y) :- Padre(Z, X), Padre(Z, Y), Hombre(X), Hombre(Y)
    b. Ejemplo práctico



i.

```
| ?- female(pam).

yes
| ?- parent(pam, bob).
```

ii. yes

7. Prolog - Data Objects
    a. Aquí hablamos de “data objects in prolog” y sus categorías
        i. Este es un ejemplo de variables anónimas, se caracterizan por no tener nombres ejemplo:
            1. hates(jim,tom).
               hates(pat,bob).
               hates(dog,fox).

hates(peter,tom).

```
tell('C:/Users/costco/paradigmas/Practica4/hates.pl'),
write('hates(jim, tom).'), nl,
write('hates(pat, bob).'), nl,
write('hates(dog, fox).'), nl,
write('hates(peter, tom).'), nl,
told.
```

```
| ?- hates(X,tom).

X = jim ? ;

X = peter

(15 ms) yes
| ?-
```

2.

8. Prolog - Operators
    a. En esta sección veremos ejemplos de diferentes tipos de operadores en Prolog y comparaciones.
        i. Operator Meaning
        X > Y X is greater than Y
        X < Y X is less than Y
        X >= Y X is greater than or equal to Y
        X =< Y X is less than or equal to Y
        X =:= Y the X and Y values are equal
        X =\= Y the X and Y values are not equal
        ii. Ejemplo en la aplicación
            1.

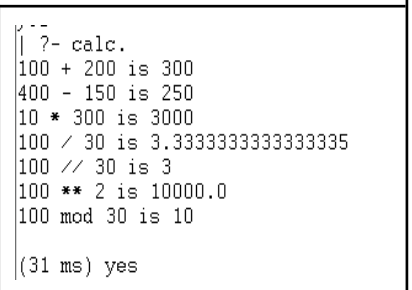
```
| ?- 1+2=:=2+1.

yes
| ?- 1+2=2+1.

no
| ?- 1+A=B+2.

A = 2
B = 1

yes
| ?- 5<10.
```

```
?- 1+2=:=2+1.

es
?- 1+2=2+1.

o
?- 1+A=B+2.

 = 2
 = 1

es
?- 5<10.
```

```
calc :- X is 100 + 200,write('100 + 200 is
'),write(X),nl,
        Y is 400 - 150,write('400 - 150 is
'),write(Y),nl,
```

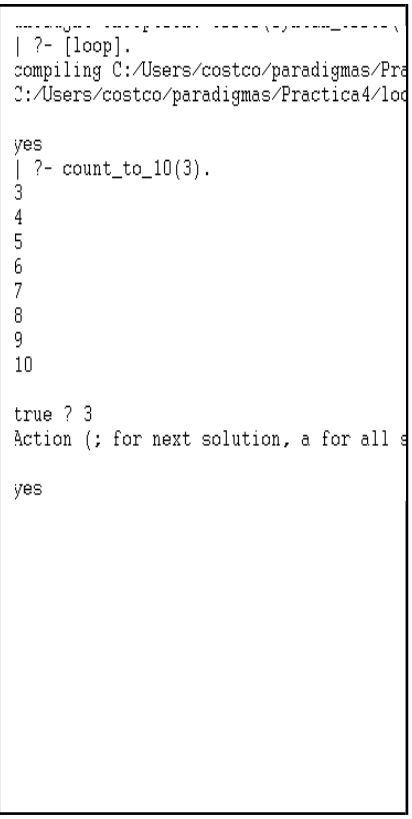
```
        Z is 10 * 300,write('10 * 300 is '),write(Z),nl,
        A is 100 / 30,write('100 / 30 is '),write(A),nl,
        B is 100 // 30,write('100 // 30 is '),write(B),nl,
        C is 100 ** 2,write('100 ** 2 is '),write(C),nl,
        D is 100 mod 30,write('100 mod 30 is '),write(D),nl.
```

```
| ?- calc.
100 + 200 is 300
400 - 150 is 250
10 * 300 is 3000
100 / 30 is 3.3333333333333335
100 // 30 is 3
100 ** 2 is 10000.0
100 mod 30 is 10

yes
```



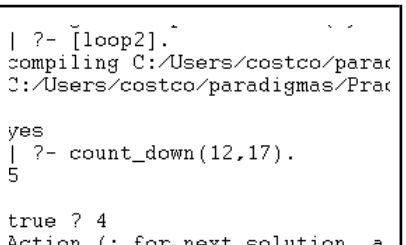
9. Loop & Decision Making
    1.

```
| ?- [loop].
compiling C:/Users/costco/paradigmas/Practica4/loop.pl for byte code...
C:/Users/costco/paradigmas/Practica4/loop.pl compiled, 4 lines read - 773 bytes written, 5 ms

yes
| ?- count_to_10(3).
3
4
5
6
7
8
9
```

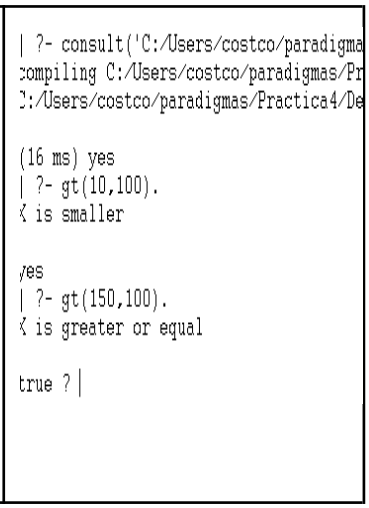


10	

2.

| ?- [loop2]. compiling C:/Users/costco/paradigmas/Practica4/loop2.pl for byte code... C:/Users/costco/paradigmas/Practica4/loop2.pl compiled, 8 lines read - 1097 bytes written, 6 ms yes | ?- count_down(12,17). 5 true ? ; 4	

3. Decision Making
    a. Las declaraciones de decisión son declaraciones If-Then-Else. Por lo tanto, cuando intentamos coincidir con alguna condición y realizar alguna tarea, utilizamos las declaraciones de toma de decisiones. El uso básico es el siguiente: If <condition> is true, Then <do this>, Else
    b.

| ?- consult('C:/Users/costco/paradigmas/Practica4/Decision_Making.pl'). compiling C:/Users/costco/paradigmas/Practica4/Decision_Making.pl for byte code... C:/Users/costco/paradigmas/Practica4/	

Decision_Making.pl compiled, 9 lines read - 1138 bytes written, 6 ms (16 ms) yes | ?- gt(10,100). X is smaller yes | ?- gt(150,100). X is greater or equal	

b. Prolog - Conjunctions & Disjunctions
   i. La conjunción (lógica AND) puede implementarse utilizando el operador coma (,). Entonces, dos predicados separados por coma se unen con una declaración AND.
   ii. La disyunción (lógica OR) se puede implementar utilizando el operador punto y coma (;). Por lo tanto, dos predicados separados por punto y coma se unen con una declaración OR.

```
[conj_disj].
uncaught exception: error(s
| ?- compiling D:/TP Prolog
uncaught exception: error(s
| ?- D:/TP Prolog/Sample_Co

yes
| ?- father(jhon,bob).
uncaught exception: error(s
| ?-
yes
| ?- child_of(jhon,bob).
uncaught exception: error(s
| ?-
true ?

yes
| ?- child_of(lili,bob).
uncaught exception: error(s
| ?-
yes
| ?-
```

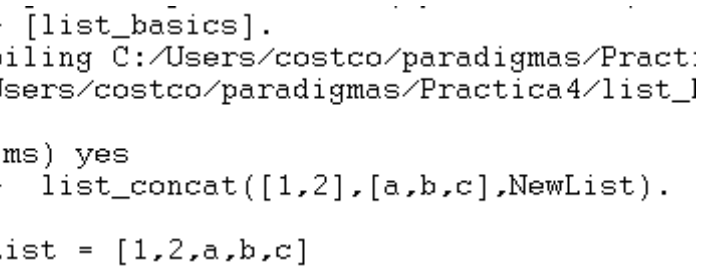
10. Prolog - Lists
    a. Las listas son una estructura que se puede usar en diferentes casos de programación no numérica.

i. Ejemplo: list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).

ii. Resultado:
[list_basics].
uncaught exception: error(syntax_error('user_input:106 (char:3) current operator needs brackets'),read_term/3)
| ?- compiling D:/TP Prolog/Sample_Codes/list_basics.pl for byte code…
list_member(b,[a,b,c]).
uncaught exception: error(syntax_error('user_input:108 (char:2) . or operator expected after expression'),read_term/3)
| ?-
true ?

b. Concatenation
   i. La concatenación de dos listas significa agregar los elementos de la segunda lista después de la primera. Entonces, si las dos listas son [a, b, c] y [1, 2], entonces la lista final será [a, b, c, 1, 2].
      1. [list_basics].
         compiling C:/Users/costco/paradigmas/Practica4/list_basics.pl for byte code...
         C:/Users/costco/paradigmas/Practica4/list_basics.pl compiled, 4 lines read - 871 bytes written, 10 ms
         (15 ms) yes
         | ?- list_concat([1,2],[a,b,c],NewList).

         NewList = [1,2,a,b,c]
         yes



      2. Ejemplo borrar desde lista: list_delete(X, [X], []).
         list_delete(X,[X|L1], L1).

```
list_delete(X, [Y|L2], [Y|L1]) :-
list_delete(X,L2,L1).
```

```
| ?- list_delete(a,[a,e,i,o,u],NewList).

NewList = [e,i,o,u] ?

yes
| ?- list_delete(a,[a],NewList).

NewList = [] ?

yes
| ?- list_delete(X,[a,e,i,o,u],[a,e,o,u]).

X = i ? ;

no
| ?-
```

3. Añadir dos listas significa combinarlas, o agregar una lista como un elemento. Ahora, si el elemento está presente en la lista, entonces la función de añadir no funcionará. Por lo tanto, crearemos un predicado llamado list_append(L1, L2, L3). A continuación se presentan algunas observaciones.
    a. Ejemplo: list_member(X,[X|_]).
    list_member(X,[_|TAIL]) :-
    list_member(X,TAIL).

    list_append(A,T,T) :- list_member(A,T),!.
    list_append(A,T,[A|T]).
    list_append(a,[e,i,o,u],NewList).

    NewList = [a,e,i,o,u]

    yes
    | ?- list_append(e,[e,i,o,u],NewList).

    NewList = [e,i,o,u]

yes

b. Insertar en la lista: esta función se usa para insertar X en la lista L y el resultado será R
   i. list_delete(X, [X], []).
      list_delete(X,[X|L1], L1).
      list_delete(X, [Y|L2], [Y|L1]) :-
      list_delete(X,L2,L1).

      list_insert(X,L,R) :- list_delete(X,R,L).
   ii. Resultado:
      list_insert(a,[e,i,o,u],NewList).
      NewList = [a,e,i,o,u] ? a
      NewList = [e,a,i,o,u]
      NewList = [e,i,a,o,u]
      NewList = [e,i,o,a,u]
      NewList = [e,i,o,u,a]
      NewList = [e,i,o,u,a]
      (15 ms) no

c. Permutation Operation: Esta operación cambiará las posiciones de los elementos de la lista y generará todos los resultados posibles.
      list_delete(X,[X|L1], L1).
      list_delete(X, [Y|L2], [Y|L1])
      :-list_delete(X,L2,L1).
      list_perm([],[]).
      list_perm(L,[X|P])
      :-list_delete(X,L,L1),list_perm(L1,P).
   i. Resultado: list_perm([a,b,c,d],X).
      X = [a,b,c,d] ? a
      X = [a,b,d,c]
      X = [a,c,b,d]
      X = [a,c,d,b]
      X = [a,d,b,c]
      X = [a,d,c,b]
      X = [b,a,c,d]
      X = [b,a,d,c]
      X = [b,c,a,d]
      X = [b,c,d,a]
      X = [b,d,a,c]
      X = [b,d,c,a]

```
X = [c,a,b,d]
X = [c,a,d,b]
X = [c,b,a,d]
X = [c,b,d,a]
X = [c,d,a,b]
X = [c,d,b,a]
X = [d,a,b,c]
X = [d,a,c,b]
X = [d,b,a,c]
X = [d,b,c,a]
X = [d,c,a,b]
X = [d,c,b,a]

(31 ms) no
```

ii. Operación de Reversión: Supongamos que tenemos una lista L = [a, b, c, d, e], y queremos revertir los elementos, por lo que la salida será [e, d, c, b, a]. Para hacer esto, crearemos una cláusula, list_reverse(List, ReversedList). A continuación se presentan algunas observaciones.

1.
```
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :-
list_concat(L1,L2,L3).

list_rev([],[]).
list_rev([Head|Tail],Reversed) :-
   list_rev(Tail, RevTail),list_concat(RevTail,
[Head],Reversed).
```
2. Resultado: yes
```
| ?- list_rev([a,b,c,d,e],NewList).

NewList = [e,d,c,b,a]

yes
| ?- list_rev([a,b,c,d,e],[e,d,c,b,a]).

yes
```

11. Prolog - Recursion and Structures
    a. Recursión es una técnica en la que un predicado se utiliza a sí mismo (quizás con otros predicados) para encontrar el valor de verdad.

b. Coincidencia en Prolog: La coincidencia se utiliza para verificar si dos términos dados son iguales (idénticos) o si las variables en ambos términos pueden tener los mismos objetos después de ser instanciadas. Veamos un ejemplo.
c. Árboles Binarios
   A continuación se muestra la estructura de un árbol binario utilizando estructuras recursivas:

   Árboles Binarios
   La definición de la estructura es la siguiente:node(2, node(1,nil,nil), node(6, node(4,node(3,nil,nil), node(5,nil,nil)), node(7,nil,nil)))

12. Prolog - Backtracking
    a. El backtracking es un procedimiento en el que Prolog busca el valor de verdad de diferentes predicados verificando si son correctos o no.
    b. Knowledge Base
       i.
          1. | ?- pay(X,Y).
             X = tom
             Y = alice ?
             (15 ms) yes
             | ?- pay(X,Y).
             X = tom
             Y = alice ? ;
             X = tom
             Y = lili ? ;
             X = bob
             Y = alice ? ;
             X = bob
             Y = lili
          2. yes
    c. Evitando el Backtracking: desventajas del backtracking. A veces escribimos los mismos predicados más de una vez cuando nuestro programa lo demanda, por ejemplo, para escribir reglas recursivas o para crear algunos sistemas de toma de decisiones. En tales casos, el backtracking no controlado puede causar ineficiencia en un programa. Para resolver esto, utilizaremos el Corte (*Cut*) en Prolog.
    d. f(X,0) :- X < 3. % Rule 1

       f(X,2) :- 3 =< X, X < 6. % Rule 2

```
f(X,4) :- 6 =< X.            % Rule 3
```

e. | ?- trace
.
The debugger will first creep -- showing everything (trace)
yes
{trace}
| ?- f(1,Y), 2<Y.
1 1 Call: f(1,_23) ?
2 2 Call: 1<3 ?
2 2 Exit: 1<3 ?
1 1 Exit: f(1,0) ?
3 1 Call: 2<0 ?
3 1 Fail: 2<0 ?
1 1 Redo: f(1,0) ?
2 2 Call: 3=<1 ?
2 2 Fail: 3=<1 ?
2 2 Call: 6=<1 ?
2 2 Fail: 6=<1 ?
1 1 Fail: f(1,_23) ?

f.
(46 ms) no

13. Prolog - Different and Not
    a. Aquí definiremos dos predicados: different y not. El predicado different verificará si dos argumentos dados son iguales o no. Si son iguales, devolverá falso; de lo contrario, devolverá verdadero.
        i. different(X, X) :- !, fail.
        different(X, Y).
        ii. yes
        | ?- different(100,200).
        yes
        | ?- different(100,100).
        no
        | ?- different(abc,def).
        yes
14. Prolog - Inputs and Outputs
    a. En este capítulo, veremos algunas técnicas para manejar entradas y salidas a través de Prolog. Utilizaremos algunos predicados incorporados para realizar estas tareas y también veremos técnicas de manejo de archivos.
        i. El predicado write()

1. Para escribir la salida, podemos usar el predicado write(). Este predicado toma el parámetro como entrada y escribe el contenido en la consola por defecto. write() también puede escribir en archivos. Veamos algunos ejemplos de la función write().
2. | ?- write(56).
   56
   yes
   | ?- write('hello').
   hello
   yes
   | ?- write('hello'),nl,write('world').
   hello
   world
   yes
   | ?- write("ABCDE")
   .
   [65,66,67,68,69]
   yes

ii. El predicado read(): El predicado read() se utiliza para leer desde la consola. El usuario puede escribir algo en la consola, que puede ser tomado como entrada y procesarlo.

1. cube :-
   write('Write a number: '),
   read(Number),
   process(Number).
   process(stop) :- !.
   process(Number) :-
   C is Number * Number * Number,
   write('Cube of '),write(Number),write(':
   '),write(C),nl, cube.
2. Ejemplo:
   | ?- cube.
   Write a number: 2.
   Cube of 2: 8
   Write a number: 10.
   Cube of 10: 1000
   Write a number: 12.
   Cube of 12: 1728

Write a number: 8.
Cube of 8: 512
Write a number: stop
.

3. (31 ms) yes

15. Prolog - Built-In Predicates
    a. En Prolog, hemos visto los predicados definidos por el usuario en la mayoría de los casos, pero también hay algunos predicados integrados.Hay tres tipos de predicados integrados como se indica a continuación:
        i. Identificación de términos
        ii. Descomposición de estructuras
        iii. Recopilación de todas las soluciones
    b. Ejemplo:
        i. | ?- var(X).
        yes
        | ?- X = 5, var(X).
        no
        | ?- var([X]).
        no
16. Tree Data Structure (Case Study)
    a. Ahora veremos un estudio de caso en Prolog. Veremos cómo implementar una estructura de datos de árbol usando Prolog y crearemos nuestros propios operadores. Así que empecemos con la planificación.

    Supongamos que tenemos un árbol como se muestra a continuación:

    Estructura de Datos de Árbol
    Tenemos que implementar este árbol usando Prolog. Tenemos algunas operaciones de la siguiente manera:

        op(500, xfx, 'is_parent').
        op(500, xfx, 'is_sibling_of').
        op(500, xfx, 'is_at_same_level').

    b. /* The tree database */
    :- op(500,xfx,'is_parent').
    a is_parent b. c is_parent g. f is_parent l. j is_parent q.
    a is_parent c. c is_parent h. f is_parent m. j is_parent r.
    a is_parent d. c is_parent i. h is_parent n. j is_parent s.

```
b is_parent e. d is_parent j. i is_parent o. m is_parent t.
b is_parent f. e is_parent k. i is_parent p. n is_parent u.
n
is_parent v.
/* X and Y are siblings i.e. child from the same parent */
:- op(500,xfx,'is_sibling_of').
X is_sibling_of Y :- Z is_parent X,
                     Z is_parent Y,
                     X \== Y.
leaf_node(Node) :- \+ is_parent(Node,Child). % Node
grounded
/* X and Y are on the same level in the tree. */
:-op(500,xfx,'is_at_same_level').
X is_at_same_level X .
X is_at_same_level Y :- W is_parent X,
                        Z is_parent Y,
                        W is_at_same_level Z.
```

c.
```
| ?- i is_parent p.
yes
| ?- i is_parent s.
no
| ?- is_parent(i,p).
yes
| ?- e is_sibling_of f.
true ?
yes
| ?- is_sibling_of(e,g).
no
```

d. Más sobre la Estructura de Datos de Árbol
   i. Aquí, veremos algunas operaciones adicionales que se realizarán en la estructura de datos de árbol dada anteriormente.