CADA GRUPO DEBE COPIAR Y PEGAR LA SIGUIENTE PREGUNTA

Grupo: nombre1, nombre2, etc

(40 min) - Hasta 8.50

RECUERDEN: IMPLEMENTEN SUS PROPIAS REGLAS/PREDICADOS PARA LA MANIPULACIÓN DE LISTAS

Pregunta 1 (solo hacer esta pregunta) Ahora hay permisos de edición

Considere un carrito de compras de supermercado, que se conforma de artículos. Un artículo se compone de su nombre, precio y cantidad de elementos a llevar. En base a esto implemente en pseudo prolog las siguientes consultas.

ejemplo: articulo: tomate, 1500, 3 , [tomate, 1500, 3]
carrito de compras: lista de articulos -> [articulo1, articulo2, articulo3, etc]

Documente todas las cláusulas implementadas.

Implemente todas las cláusulas que operan con listas. No utilice reglas propias de Prolog. Debe implementarlas.

1. ¿Está el carrito de compras vacío? estaCarritoVacio?

TDAs:
carrito compra: [A1, A2, A3]
Articulo: nombre, precio, cantidad
articulo(Nombre, Precio, Cantidad, [Nombre, Precio, Cantidad].
carrito(A1, A2, A3, [A1,A2,A3]).

% es verdadero que carrito de compra esta vacio si la lista esta vacio

estaCarritoVacio([A1,A2,A3]) -> false
estaCarritoVacio([]) -> true

estaCarritoVacio([ ]).

2. Agregar un artículo al carrito. agregaArticulo

% agregarElemento(Articulo, [ ], [Articulo]).
% agregarElemento ([tomate, 1000, 1], [], [[tomate,1000,1]]).
% agregarElemento ([tomate, 1000, 1], [[palta, 4000, 5]], [[tomate,1000,1], [palta, 4000, 5]).

agregarElemento(Articulo, [ ], [Articulo]).

agregarElemento(Articulo, [Primero|Resto], [Articulo|Primero|Resto]).

agregarElemento(Articulo, [Primero|Resto], [Articulo|Primero]).

agregarElemento(Articulo, Lista, [Articulo|Lista]).

Profe podría hacer por ejemplo algo que agregue elemento solo con dar elemento , y dentro este el agregarElemento()

Por ejemplo:
AgregarElemento(Artículo):-
AgregarAlPrincipio(Lista, Artículo).

AgregarElemento ([tomate,100,1]).

3. Obtener el **precio total a pagar por el carrito de supermercado**

articulo(Nombre, Precio, Cantidad, [Nombre, Precio, Cantidad]).
carrito(A1, A2, A3, [A1,A2,A3]).

%sumarPrecios(Carrito, PrecioTotal).

%sumarPrecio ([[tomate, 100, 1], [palta, 200, 5]] , PrecioTotal). % PrecioTotal = 100 + 200 = 300

[EsumarPrecios([ ], 0).

sumarPrecios([Elemento|Resto] , PrecioTotal) :-
articulo(_, Precio, _, Elemento]), % Elemento: [tomate, 100, 1] Resto: [[palta,200,5]]
PrecioTotal is Precio + PrecioTotal, %PrecioTotal is 100 + PrecioTotal
sumarPrecios(Resto, PrecioTotal).

1. **% PrecioTotal is 100 + PrecioTotal**
2. **% PrecioTotal is 200 + PrecioTotal**
3. **% PrecioTotal is 0**

2. % PrecioTotal is 200 + 0 = 200
    1. PrecioTotal is 100 + 200 = 300

sumarPrecios([Elemento|Resto] , PrecioTotal) :-
articulo(_, Precio, Cantidad, Elemento]), % Elemento: [tomate, 100, 1] Resto: [[palta,200,5]]
PrecioTotal is Cantidad*Precio + PrecioTotal, %PrecioTotal is 100 + PrecioTotal
sumarPrecios(Resto, PrecioTotal).

```
sumarCantidades [Elemento|Resto] , CantidadTotal) :-
  articulo(_, _, Cantidad, Elemento]),
  CantidadTotal is Cantidad + CantidadTotal,
  sumarCantidades(Resto, CantidadTotal).
```

4. Obtener el primer elemento del carrito de supermercado

```
primerElemento( [Primero| _ ] , Primero).  %car
```

5. Obtener el resto de los elementos del carrito.

```
restoElementos( [ _ | Resto ] , Resto).  %cdr
```

6. Obtener el largo total del carrito de compras (len) *repositorio
7. Seleccionar todos los artículos cuyo precio sea mayor a 1000 *repositorio

8. Realizar una consulta que permite añadir 4 artículos al carrito de supermercado, seleccionar artículos cuyo precio es mayor a 100 y finalmente entregar el precio total a pagar por todos los artículos seleccionados

```
agregarElemento(Articulo1, CarritoInicial, Carrito2),
agregarElemento(Articulo2, Carrito2, Carrito3),
agregarElemento(Articulo3, Carrito3, Carrito4),
agregarElemento(Articulo4, Carrito4, CarritoFinal),
seleccionarArticulosPrecioMayor100(CarritoFinal, ArticulosSelecionados),
sumarPrecios(ArticulosSeleccionados, PrecioTotal).
```

Es que cuando piden en el lab que demos el largo de una lista solo con dar la lista en el dominio

```
largo(Lista) : numero
Prolog: len -> len(Lista)

%dom: lista
largoLista(Lista):
   calcularLargo(Lista, 0).  %wrapper, envoltorio

calcularLargo( [ ], 0).
calcularLargo([Primero|Resto], LargoTotal) :-
   LargoTotal is LargoTotal + 1
   calcularLargo(Resto, LargoTotal).
```

Grupo 1:

```
estaCarritoVacio(Carrito) :- false.
estaCarritoVacio([]):- true.

agregarArticulo(Articulo, [], [Articulo]).
agregarArticulo(Articulo, [], [Articulo, Lista]).

precioTotal([],0).
precioTotal(Lista,acum):-
```

list

CAR

```
obtenerPrimerElemento([A1,A2,A3], A1].
obtenerPrimerElemento([A90,A11,A3, A93, A666], A90].

obtenerPrimerElemento([Primero| _ ], Primero]).

obtenerResto([A90,A11,A3, A93, A666], [A11, A3, A93, A666]].

obtenerResto( [ _| Resto ], Resto]).
```

Grupo 2:

```
1-
estaCarritoVacio([]).
2-
agregaArticulo( Articulo, [], [Articulo] ).
agregaArticulo( Articulo, Carrito, [Articulo|Carrito] ).

3-
totalArticulo([_,Precio, Cantidad], PrecioTotalArticulo):-
    PrecioTotalArticulo is Precio * Cantidad.

totalCarrito([], 0).
totalCarrito([Primero|Resto], PrecioTotalCarrito):-
    totalCarrito(Resto, AcumuladoPrecioCarrito),
    totalArticulo(Primero, PrecioTotalArticulo),
    PrecioTotalCarrito is AcumuladoPrecioCarrito + PrecioTotalArticulo.

4.-
%obtenerPrimerElemento([A1,A2,A3],A1)
obtenerPrimerElemento([Primero|_], Primero).
```

5.-
%obtenerRestoElemento([A1,A2,A3],[A2,A3])
obtenerRestoElemento([_|Resto], Resto).

6-
largoCarrito([], 0).
largoCarrito([_|Resto], Largo):-
largoCarrito(Resto, Largoacumulado),
largo is Largoacumulado + 1.

¿Está el carrito de compras vacío? estaCarritoVacio?

% es verdadero que el carrito de compras es vacio si la lista es vacia

estaCarritoVacio([A1,A2,A3]. %false
estaCarritoVacio([]). %verdadero

estaCarritoVacio([ ]).

Obtener el precio total a pagar por el carrito de supermercado

Carrito: [A1,A2,A3]
carrito(A1, A2, A3, [A1, A2, A3]).
Articulo:
articulo(Nombre, Precio, Cantidad, [Nombre, Precio, Cantidad]).

%sumarPrecio(Carrito, PrecioTotal).
%sumarPrecio([ ], 0).

% sumarPrecio([[tomate, 100, 1], [palta, 5000, 3]], PrecioTotal).
% PrecioTotal: 5100

sumarPrecio([ ], 0).
sumarPrecio(**[Primero|Resto]**, PrecioTotal) :-
articulo(_, Precio, _, Primero),
PrecioTotal is Precio + PrecioTotal, %PrecioTotal is 100 + PrecioTotal
sumarPrecio(Resto, PrecioTotal).

% Carrito = [A1, A2, A3, An]
Carrito = [Primero|Resto]:
Primero = A1 car (selector)

Resto = [A2, An] cdr

% Carrito = [A2, A3, An]
Carrito = [Primero|Resto]:
Primero = A2 car (selector)
Resto = [A3, An] cdr

PrecioTotal is 100 + PrecioTotal
PrecioTotal is 5000 + (100 + 0) = 5100
PrecioTotal is 0

Pregunta extra

1. La Universidad Lógica cuenta con una base de datos en Prolog de sus estudiantes, las asignaturas que ofrecen, los profesores que las imparten a través de distintas secciones, las asignaturas que toman sus estudiantes y las calificaciones que estos obtienen en los cursos que inscriben. La base de datos en términos de hechos se expresa a través de los siguientes predicados.

```
carrera(CodigoCarrera,NombreCarrera)

estudiante(RutEstudiante,Nombre,Apellido,CodigoCarrera).

asignatura(CodigoAsignatura,Nombre,Nivel,CodigoPrerrequisito,CodigoCarrera).

profesor(RutProfesor,Nombre,Apellido,Experiencia)

seccion(CodigoSeccion,RutProfesor,CodigoAsignatura,Capacidad).

inscripcion(RutEstudiante,CodigoSeccion)

calificacion(CódigoSeccion,RutEstudiante,Nota)
```

   A partir de estos antecedentes escribir consultas y/o reglas que permitan determinar:

   1) Estudiantes de la carrera de “Ingeniería Civil en Informática”.
   2) Profesores que dictan clases de la carrera de “Administración de Empresas”
   3) Si un estudiante puede inscribir una determinada asignatura considerando los prerrequisitos de ésta.
   4) Si dos estudiantes están cursando una misma asignatura (en cualquiera de sus secciones).
   5) El Rut del estudiante cuya calificación sea la más alta en una sección.

Pregunta extra

Considere la especificación e implementación parcial del TDA documento, el cual forma parte de un sistema tipo GoogleDocs.

a. retorna la **primera palabra** de un documento.
dom: document ; rec: word
firstWord(document):word

b. retorna un **documento con las siguientes palabras** después de la primera.
dom: document ; rec: document
nextWords(document):document

c. localiza las **posiciones donde se ubica una palabra en el documento**. El retorno corresponde a una lista con todas las posiciones donde figura la palabra.
dom: document X word ; rec: position List
find(document,word):position List

d. reemplaza **todas las ocurrencias de la palabra especificada en word**, por newWord. El retorno es un nuevo documento con los cambios efectuados.
dom: document X word X word, rec: document
replaceAll(document,word,newWord):document

e. contabiliza la cantidad de caracteres en el documento sin considerar los espacios (se asume que entre palabras existen espacios simples).
dom: document ; rec:integer+{0}
charactersCountWithoutSpace(document):integer+{0}

f. busca frases (ej: “esta es una frase”) en un documento. El retorno es una lista con las posiciones en la lista donde comienzan las frases que coinciden.
dom:document X string , rec: position List
findPhrase(document,phrase):position List