|  |  |
| --- | --- |
|  | **PRÁCTICA 2: PROGRAMACIÓN**  **EN PROLOG**    **Participantes:**  Ciprian Ilut 160348  Carlos Mateos Martín 160185  Javier del Río García 160451 |

Código

%%%%%%%%%%%%%%%%%ALUMNOS DEL GRUPO%%%%%%%%%%%%%%%%%

alumno\_prode('Ilut','xxxxxx','Ciprian',160348).

alumno\_prode('Mateos','Martin','Carlos',160185).

alumno\_prode('del Rio','Garcia','Javier',160451).

%%%%%%%%%%%%%%%%%ALUMNOS DEL GRUPO%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%FUNCIONES AUXILIARES%%%%%%%%%%%%%%%%

var(X):-X=f(a). % Predicado que se verifica si X es una variable libre

nonvar(X):-not(var(X)). % Predicado que se verifica si X no es una variable libre

not(Goal) :- call(Goal), !, fail. % Negación por fallo

not(Goal).

addend([],X, [X]). % Predicado que se usa para añadir un elemento como último de la lista

addend([C|R],X, [C|R1]):-

addend(R,X, R1).

addhead(X, L, [X|L]). % Predicado que se usa para añadir un elemento como cabeza de la lista

unaCabeza([Cab|Lista],Cab,Lista). % Predicado utilizado para obtener el primer elemento de una lista

%%%%%%%%%%%%%%%%FUNCIONES AUXILIARES%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%MENOR%%%%%%%%%%%%%%%%

menor(A,B,Comp,M):-

functor(X,Comp,2), % Compone X con el predicado Comp de dos argumentos

arg(1,X,A), % Se introduce A en el primer argumento

arg(2,X,B), % Se introduce B en el segundo argumento

(call(X),M=A); % Se comprueba si la comparación es verdadera. De ser así el valor devuelto será el de A

M=B. % En caso contrario será B el valor devuelto

%%%%%%%%%%%%%%%%MENOR%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%MENOR\_O\_IGUAL%%%%%%%%%%%%%%%%

menor\_o\_igual(X,Y):-

(var(X),nonvar(Y) ; var(Y),nonvar(X)); % Uno de los elementos es 0

functor(X,F1,A1), % Se compone X con el predicado F1 de argumentos A1

functor(Y,F2,A2), % Se compone Y con el predicado F2 de argumentos A2

(

(F1@<F2;A1<A2); % Comprobamos el tamaño de la variable en si

(F1==F2,@<(X, Y)) % Se comparan tanto la aridad como los nombres y se deduce si es menor

);

X==Y. % En caso contrario se comprueba si X e Y son iguales

%%%%%%%%%%%%%%%%MENOR\_O\_IGUAL%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%LISTA\_HOJAS%%%%%%%%%%%%%%%

lista\_hojas(Lista,Hojas):-

lista\_hojas\_aux(Lista,[],Hojas). % Se llama al predicado auxiliar

lista\_hojas\_aux([],Hojas,Hojas). % Resolución para el caso de lista vacía

lista\_hojas\_aux(Lista,Hojas,Salida):-

unaCabeza(Lista,Cab,ListaCopia), % Se extrae el primer elemento de la lista

addend(Hojas,tree(Cab,void,void),HojasCopia), % Se añade a Hojas como último elemento de Hojas con la forma (extraido,void,void)

lista\_hojas\_aux(ListaCopia,HojasCopia, Salida). % Se vuelve a llamar a la función con la lista resultante de eleminar el primer elemento

%%%%%%%%%%%%%%%%LISTA\_HOJAS%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%HOJAS\_ARBOL%%%%%%%%%%%%%%%

hojas\_arbol(Hoja,Comp,Arbol):-

(

%En caso de que hay un solo elemento en la lista

unaCabeza(Hoja,Hoja1,HojaCopia),

ArbolAux=Hoja1 % Se almacena el primer elemento de la lista en la variable ArbolAux

),

hojas\_arbol\_aux(HojaCopia,Comp,ArbolAux,Arbol). % Se llama al predicado auxiliar con una lista sin el primer elemento

hojas\_arbol\_aux([],Comp,Arbol,Arbol). % Caso base

hojas\_arbol\_aux(Hoja,Comp,ArbolAux,Arbol):-

(

unaCabeza(Hoja,Hoja1,HojaCopia), % Se extrae el primer elemento de la lista

arg(1,Hoja1,Indice1), % Se extrae el valor de Hoja1 y se almacena en Indice1

arg(1,ArbolAux,IndiceArbol), % Se extrae el valor de ArbolAux y se almacena en IndiceArbol

menor(Indice1,IndiceArbol, Comp, Menor), % Se comprueba cuál es el menor de los dos

(

(Menor=IndiceArbol,ArbolAux1=tree(Menor,ArbolAux,Hoja1)); % Si el menor es IndiceArbol, se devuelve un tree con su nodo correspondiente y después el de Hoja1

(Menor=Indice1,ArbolAux1=tree(Menor,Hoja1,ArbolAux)) % Si el menor es Indice1, se devuelve un tree con su nodo correspondiente y después el de ArbolAux

)

),

hojas\_arbol\_aux(HojaCopia,Comp,ArbolAux1,Arbol). % Por último, se vuelve a llamar al predicado auxiliar con el nuevo árbol generado

%%%%%%%%%%%%%%%%HOJAS\_ARBOL%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%ORDENACION%%%%%%%%%%%%%%%

ordenacion(Arbol, Comp, Orden):-

arg(1,Arbol,A1), % Se extrae el primer nodo del árbol

arg(2,Arbol,A2), % Se extrae el segundo nodo del árbol

(% Si solo había un elemento en el árbol, se devuelve una lista con A1

A2==void,

addend([],A1,Orden)

);

(

arg(2,Arbol,A2),

arg(3,Arbol,A3), % Se extrae el tercer nodo del árbol

addhead(A3, [], Hojas), % Se añade el nodo como primer elemento de la lista

ordenacion\_aux(A2,Comp,[],Orden,Hojas) % Se llama al predicado auxiliar con el segundo nodo del árbol

).

ordenacion\_aux(Arbol,Comp, OrdenAux,Orden,Hojas):-

arg(1,Arbol,A1), % Se extrae el primer nodo del árbol secundario

arg(2,Arbol,A2), % Se extrae el segundo nodo del árbol secundario

( % Si solo hay un solo elemento en el árbol con el que se ha llamado al predicado auxiliar, se reflota el árbol

A2==void,

addend(OrdenAux,A1,OrdenAux1),

reflotar(Hojas,Comp,OrdenAux1,Orden)

);

arg(2,Arbol,A2), % Se extrae el segundo nodo del árbol

arg(3,Arbol,A3), % Se extrae el tercer nodo del árbol

addhead(A3, Hojas, Hojas1), % Se añade el nodo como primer elemento de la lista

ordenacion\_aux(A2,Comp,OrdenAux,Orden,Hojas1). % Se procede a llamar al predicado auxiliar de nuevo con el segundo nodo del árbol

reflotar([],Comp,Orden,Orden). % Caso base

reflotar(Hojas,Comp,OrdenAux,Orden):- % El nodo hoja se elimina sustituyéndolo por un árbol vacío

hojas\_arbol(Hojas,Comp,Arbol),

ordenacion\_aux(Arbol,Comp,OrdenAux,Orden,[]).

%%%%%%%%%%%%%%%%ORDENACION%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%ORDENAR%%%%%%%%%%%%%%%

ordenar(Lista,Comp,Orden):- % Predicado que llama utiliza los otros predicados vistos anteriormente para ordenar los elementos de una lista

lista\_hojas(Lista,Hojas),

hojas\_arbol(Hojas,Comp,Arbol),

ordenacion(Arbol,Comp, Orden).

%%%%%%%%%%%%%%%%ORDENAR%%%%%%%%%%%%%%%

Pruebas realizadas

%%%%%%%%%%%%%%%%%%%%%%%%%%%PRUEBAS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%MENOR%%%%%%%%%%%

menor(1,2,<,M),M==1,

menor(2,1,<,M),M==1,

menor(2,2,=<,M),M==2,

menor(4,2,>,M),M==4,

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%MENOR\_O\_IGUAL%%%%%%%

menor\_o\_igual([a,a,a],[a,a,a,a,a]),%menor

menor\_o\_igual(p(a,X),p(Z,m)),%son iguales

menor\_o\_igual(p(...),p(...)),%es menor p que q

menor\_o\_igual(p(a,b,c),p(a,a,a,a,a)),%es menor

menor\_o\_igual(p(a,X,c,a),p(a,b,c,d)),%es menor

menor\_o\_igual(p(q(r(a))),p(r(a))),%es menor

menor\_o\_igual([a,b],[a,b,c,d]),%es menor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%LISTA\_HOJAS%%%%%%%%%%%%%

lista\_hojas([1,2,3],Hojas), Hojas = [tree(1,void,void),tree(2,void,void),tree(3,void,void)],

lista\_hojas([1,2,3,4,5],Hojas), Hojas =[tree(1,void,void),tree(2,void,void),tree(3,void,void),tree(4,void,void),tree(5,void,void)],

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%ORDENAR%%%%%%%%%%%%%%%

%%Usa todos los apartados del punto tres

ordenar([1,2,3,4,5],>,Orden),Orden==[5,4,3,2,1],

ordenar([1,3,2,5,4,6,7,4],<,Orden),Orden== [1,2,3,4,4,5,6,7],

ordenar([1,3,2,5,4,6,7,4],>,Orden),Orden== [7,6,5,4,4,3,2,1],

ordenar([1,a,4,6,7,4],@>,Orden),Orden == [a,7,6,4,4,1],

ordenar([1,a,4,b,a,7,4],@>,Orden),Orden== [b,a,a,7,4,4,1].