Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Practica 4. Paradigma Lógico.

Alumno: Ana Karen Ruiz Avila

Matrícula: 369435

Grupo: 941

Asignatura: Paradigmas de la Programación

Docente: Carlos Gallegos

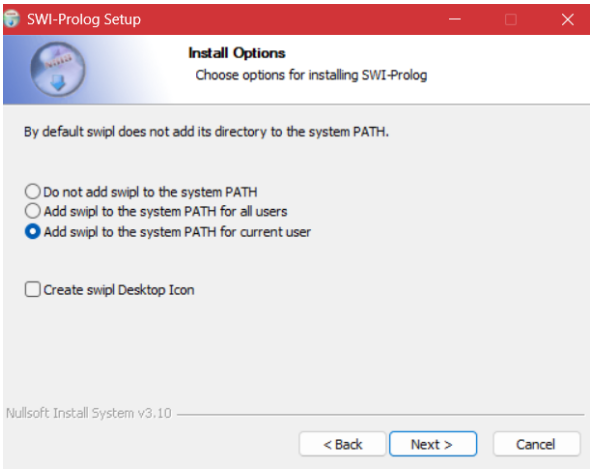
Ensenada, B. C. a 29 de mayo de 2025

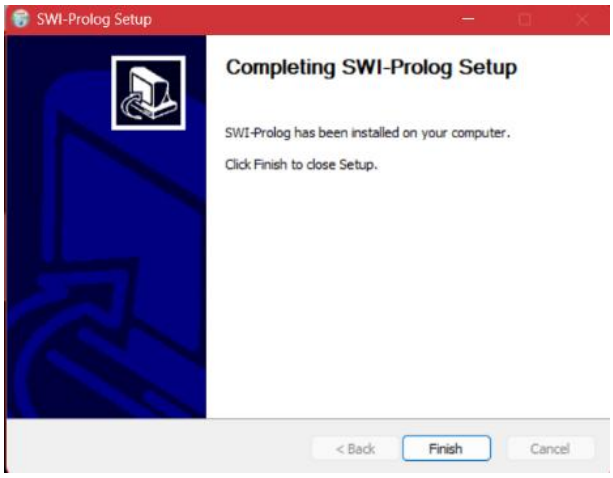
Instalación de Prolog.

Para instalar prolog, accedimos a la página de swi-prolog y descargaremos el instalador.

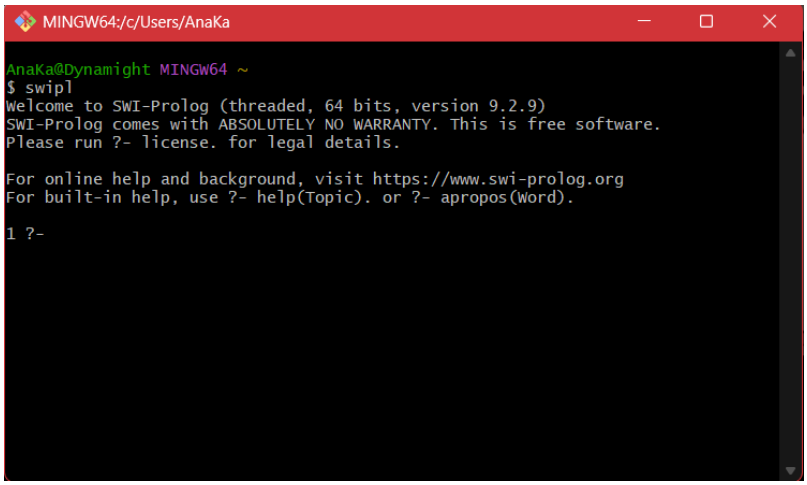
Binaries		
	14,331,585 bytes	SWI-Prolog **9.2.9-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft Windows 64-bit editions. **SHA256**: 0e6dbf5f4bb245344a257f2715f5d793d17870dee9eea1735ccb67b35f1e037c

Una vez descargado, ejecutaremos el instalador y en una de las opciones que nos ofrece antes de instalar nos saldrá si deseamos aplicar el sistema PATH para todos los usuarios, para el usuario actual o no aplicarlo. Seleccionamos la opción que aplica al usuario actual y completamos la instalación.





Una vez instalado, se comprobó en bash que la instalación fue realizada correctamente iniciándolo con el comando “swipl”.

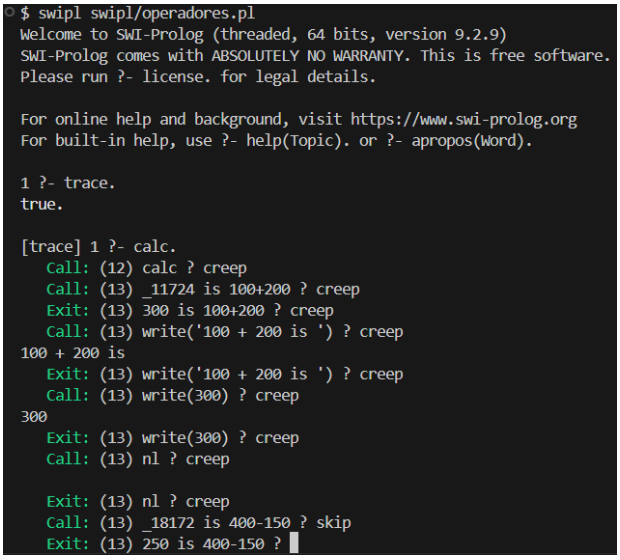


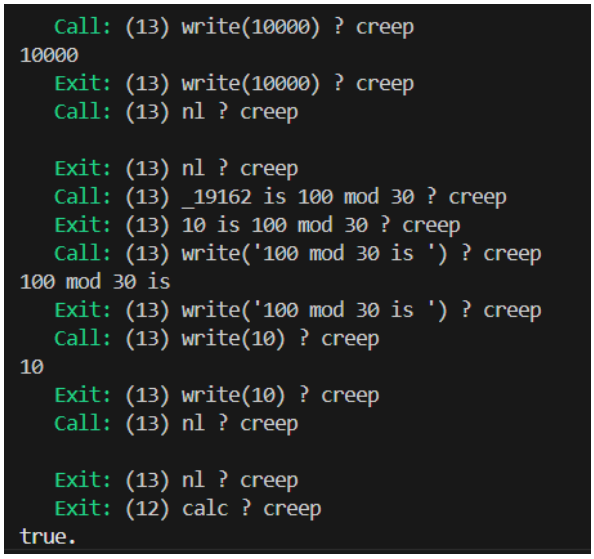
Consulta en archivos.

Se realizaron distintas consultas en diferentes archivos para comprobar la funcionalidad de trace. Esta función activa un modo de depuración en Prolog, por lo que podemos consultar el paso a paso de como se intenta resolver una consulta en prolog.

Objetos de datos (operadores.pl):

```
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.
```





Bucles (Loop.pl):

```prolog
count_to_10(10) :- write(10),nl.
count_to_10(X):-
    write(X),nl,
    Y is X + 1,
    count_to_10(Y).
```

```
[trace] 2 ?- count_to_10(1).
   Call: (12) count_to_10(1) ? creep
   Call: (13) write(1) ? creep
1
   Exit: (13) write(1) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Call: (13) _19330 is 1+1 ? creep
   Exit: (13) 2 is 1+1 ? creep
   Call: (13) count_to_10(2) ? creep
   Call: (14) write(2) ? creep
2
   Exit: (14) write(2) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Call: (14) _24972 is 2+1 ? creep
   Exit: (14) 3 is 2+1 ? creep
```

```
   Call: (22) write(10) ? creep
10
   Exit: (22) write(10) ? creep
   Call: (22) nl ? creep

   Exit: (22) nl ? creep
   Exit: (21) count_to_10(10) ? creep
   Exit: (20) count_to_10(9) ? creep
   Exit: (19) count_to_10(8) ? creep
   Exit: (18) count_to_10(7) ? creep
   Exit: (17) count_to_10(6) ? creep
   Exit: (16) count_to_10(5) ? creep
   Exit: (15) count_to_10(4) ? creep
   Exit: (14) count_to_10(3) ? creep
   Exit: (13) count_to_10(2) ? creep
   Exit: (12) count_to_10(1) ? creep
true .
```

Toma de decisions (If-else.pl):

```prolog
% If-Then-Else statement

gt(X,Y) :- X >= Y,write('X is greater or equal').
gt(X,Y) :- X < Y,write('X is smaller').
% If-Elif-Else statement

gte(X,Y) :- X > Y,write('X is greater').
gte(X,Y) :- X =:= Y,write('X and Y are same').
gte(X,Y) :- X < Y,write('X is smaller').
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- trace.
true.

[trace] 1 ?- gt(5, 3).
   Call: (12) gt(5, 3) ? creep
   Call: (13) 5>=3 ? creep
   Exit: (13) 5>=3 ? creep
   Call: (13) write('X is greater or equal') ? creep
X is greater or equal
   Exit: (13) write('X is greater or equal') ? creep
   Exit: (12) gt(5, 3) ? creep
true .
```

```
[trace] 2 ?-
gt(2, 10).
   Call: (12) gt(2, 10) ? creep
   Call: (13) 2>=10 ? creep
   Fail: (13) 2>=10 ? creep
   Redo: (12) gt(2, 10) ? creep
   Call: (13) 2<10 ? creep
   Exit: (13) 2<10 ? creep
   Call: (13) write('X is smaller') ? creep
X is smaller
   Exit: (13) write('X is smaller') ? creep
   Exit: (12) gt(2, 10) ? creep
true.
```

Rango con bucles (between.pl):

```
count_down(L, H):-
    between(L, H, Y),
    Z is H - Y,
    write(Z), nl.

count_up(L, H):-
    between(L, H, Y),
    Z is L + Y,
    write(Z), nl.
```

```
[trace] 2 ?- count_up(1, 5).
   Call: (12) count_up(1, 5) ? creep
   Call: (13) between(1, 5, _3162) ? creep
   Exit: (13) between(1, 5, 1) ? creep
   Call: (13) _4792 is 1+1 ? creep
   Exit: (13) 2 is 1+1 ? creep
   Call: (13) write(2) ? creep
2
   Exit: (13) write(2) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) count_up(1, 5) ? creep
true .
```

```
[trace] 3 ?- count_down(3, 1).
   Call: (12) count_down(3, 1) ? creep
   Call: (13) between(3, 1, _1512) ? creep
   Fail: (13) between(3, 1, _1512) ? creep
   Fail: (12) count_down(3, 1) ? creep
false.
```

Conjunciones y disyunciones (conj_disj.pl):

```
parent(jhon,bob).
parent(lili,bob).
male(jhon).
female(lili).
% Conjunction Logic
father(X,Y) :- parent(X,Y),male(X).
mother(X,Y) :- parent(X,Y),female(X).
% Disjunction Logic
child_of(X,Y) :- father(X,Y);mother(X,Y).
```

```
[trace] 1 ?- father(X, bob).
   Call: (12) father(_10612, bob) ? creep
   Call: (13) parent(_10612, bob) ? creep
   Exit: (13) parent(jhon, bob) ? creep
   Call: (13) male(jhon) ? creep
   Exit: (13) male(jhon) ? creep
   Exit: (12) father(jhon, bob) ? creep
X = jhon .
```

```
[trace] 3 ?- child_of(X, bob).
   Call: (12) child_of(_238, bob) ? creep
   Call: (13) father(_238, bob) ? creep
   Call: (14) parent(_238, bob) ? creep
   Exit: (14) parent(jhon, bob) ? creep
   Call: (14) male(jhon) ? creep
   Exit: (14) male(jhon) ? creep
   Exit: (13) father(jhon, bob) ? creep
   Exit: (12) child_of(jhon, bob) ? creep
X = jhon ;
   Redo: (14) parent(_238, bob) ? creep
   Exit: (14) parent(lili, bob) ? creep
   Call: (14) male(lili) ? creep
   Fail: (14) male(lili) ? creep
   Fail: (13) father(_238, bob) ? creep
   Redo: (12) child_of(_238, bob) ? creep
   Call: (13) mother(_238, bob) ? creep
   Call: (14) parent(_238, bob) ? creep
   Exit: (14) parent(jhon, bob) ? creep
   Call: (14) female(jhon) ? creep
   Fail: (14) female(jhon) ? creep
   Redo: (14) parent(_238, bob) ? creep
   Exit: (14) parent(lili, bob) ? creep
   Call: (14) female(lili) ? creep
   Exit: (14) female(lili) ? creep
   Exit: (13) mother(lili, bob) ? creep
   Exit: (12) child_of(lili, bob) ? creep
X = lili.
```

Listas (list_basic.pl, list_repost.pl, list_misc.pl):

```
1   list_member(X,[X|_]).
2   list_member(X,[_|TAIL]) :- list_member(X,TAIL).
3   list_length([],0).
4   list_length([_|TAIL],N) :- list_length(TAIL,N1), N is N1 + 1.
5   list_concat([],L,L).
6   list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
7   list_append(A,T,T) :- list_member(A,T),!.
8   list_append(A,T,[A|T]).
9   list_delete(X, [X], []).
10  list_delete(X,[X|L1], L1).
11  list_delete(X, [Y|L2], [Y|L1]) :- list_delete(X,L2,L1).
12  list_insert(X,L,R) :- list_delete(X,R,L).
```

```
[trace] 1 ?- list_member(3, [1,2,3,4]).
   Call: (12) list_member(3, [1, 2, 3, 4]) ? creep
   Call: (13) list_member(3, [2, 3, 4]) ? creep
   Call: (14) list_member(3, [3, 4]) ? creep
   Exit: (14) list_member(3, [3, 4]) ? creep
   Exit: (13) list_member(3, [2, 3, 4]) ? creep
   Exit: (12) list_member(3, [1, 2, 3, 4]) ? creep
true .
```

List_repost.pl:

```
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).
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).
list_shift([Head|Tail],Shifted) :- list_concat(Tail, [Head],Shifted).
list_order([X, Y | Tail]) :- X =< Y, list_order([Y|Tail]).
list_order([X]).
list_subset([],[]).
list_subset([Head|Tail],[Head|Subset]) :- list_subset(Tail,Subset).
list_subset([Head|Tail],Subset) :- list_subset(Tail,Subset).
list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).
list_union([X|Y],Z,W) :- list_member(X,Z),list_union(Y,Z,W).
list_union([X|Y],Z,[X|W]) :- \+ list_member(X,Z), list_union(Y,Z,W).
list_union([],Z,Z).
list_intersect([X|Y],Z,[X|W]) :- list_member(X,Z), list_intersect(Y,Z,W).
list_intersect([X|Y],Z,W) :- \+ list_member(X,Z), list_intersect(Y,Z,W).
list_intersect([],Z,[]).
```

```
[trace] 1 ?- list_order([1,2,3,4]).
   Call: (12) list_order([1, 2, 3, 4]) ? creep
   Call: (13) 1=<2 ? creep
   Exit: (13) 1=<2 ? creep
   Call: (13) list_order([2, 3, 4]) ? creep
   Call: (14) 2=<3 ? creep
   Exit: (14) 2=<3 ? creep
   Call: (14) list_order([3, 4]) ? creep
   Call: (15) 3=<4 ? creep
   Exit: (15) 3=<4 ? creep
   Call: (15) list_order([4]) ? creep
   Exit: (15) list_order([4]) ? creep
   Exit: (14) list_order([3, 4]) ? creep
   Exit: (13) list_order([2, 3, 4]) ? creep
   Exit: (12) list_order([1, 2, 3, 4]) ? creep
true .
```

List_misc.pl:

```
1   list_even_len([]).
2   list_even_len([Head|Tail]) :- list_odd_len(Tail).
3   list_odd_len([_]).
4   list_odd_len([Head|Tail]) :- list_even_len(Tail).
5   list_divide([],[],[]).
6   list_divide([X],[X],[]).
7   list_divide([X,Y|Tail], [X|List1],[Y|List2]) :- list_divide(Tail,List1,List2).
8   max_of_two(X,Y,X) :- X >= Y.
9   max_of_two(X,Y,Y) :- X < Y.
10  list_max_elem([X],X).
11  list_max_elem([X,Y|Rest],Max) :- list_max_elem([Y|Rest],MaxRest), max_of_two(X,MaxRest,Max).
12  list_sum([],0).
13  list_sum([Head|Tail], Sum) :- list_sum(Tail,SumTemp), Sum is Head + SumTemp.
```

```
[trace] 1 ?- list_even_len([1,2,3,4]).
   Call: (12) list_even_len([1, 2, 3, 4]) ? creep
   Call: (13) list_odd_len([2, 3, 4]) ? creep
   Call: (14) list_even_len([3, 4]) ? creep
   Call: (15) list_odd_len([4]) ? creep
   Exit: (15) list_odd_len([4]) ? creep
   Exit: (14) list_even_len([3, 4]) ? creep
   Exit: (13) list_odd_len([2, 3, 4]) ? creep
   Exit: (12) list_even_len([1, 2, 3, 4]) ? creep
true .

[trace] 2 ?- list_even_len([1,2,3]).
   Call: (12) list_even_len([1, 2, 3]) ? creep
   Call: (13) list_odd_len([2, 3]) ? creep
   Call: (14) list_even_len([3]) ? creep
   Call: (15) list_odd_len([]) ? creep
   Fail: (15) list_odd_len([]) ? creep
   Fail: (14) list_even_len([3]) ? creep
   Fail: (13) list_odd_len([2, 3]) ? creep
   Fail: (12) list_even_len([1, 2, 3]) ? creep
false.
```

Conclusiones.

La actividad dejo como resultado el como podemos manejar distintas aplicaciones en Prolog para resolver problemas lógicos en estructuras de datos. Se implemento en los diversos programas operaciones como búsqueda, longitud, suma, permutaciones y subconjuntos que pudimos analizar a través de trace. Con trace pudimos observar el flujo de ejecución de cada uno de los programas y entender el razonamiento lógico que utiliza prolog.

Repositorio.

https://github.com/demoncybor/Practica4