Name: Gourav Kumar Shaw

Enrollment No. : 2020CSB010

Subject : Artificial Intelligence Laboratory.

Semester: 8th

Assignment-1

1. To find last element of a list.

```
% 1. To find last element of a list.
last_element([X],X).
last_element([Y|T],X):-last_element(T,X).
```

Output:

1 ?- last_element([1,2,3,4],X).

X = 4

2 ?- last_element([2,3,4,7],X).

X = 7

3 ?- last_element([42], X).

X = 42

4 ?- last_element([1, [2, 3], 4], X).

X = 4

2. To append two lists in a third list.

```
% 2. Concatenating two lists
concatenate_lists([], List, List).
concatenate_lists([Head|Tail1], List2, [Head|Result]) :-
    concatenate_lists(Tail1, List2, Result).
```

Output:

1 ?- concatenate_lists([1, 2, 3], [4, 5, 6], Result).

Result = [1, 2, 3, 4, 5, 6]

2 ?- concatenate_lists([1,2],[3,4,5],Result).

Result = [1, 2, 3, 4, 5]

3. To reverse a list in another list.

```
% 3. Reversing a list
reverse_list([], []).
reverse_list([Head|Tail], Reversed):-
    reverse_list(Tail, ReversedTail),
    append(ReversedTail, [Head], Reversed).
```

Output:

1 ?- reverse_list([1, 2, 3, 4, 5], Reversed).

Reversed = [5, 4, 3, 2, 1]

4. To determine whether a list is a palindrome.

The structure of predicate:

Palindrome(L)

```
% 4. To determine whether a list is a palindrome.
is_palindrome(List) :-
    reverse(List, List).

% Alternative implementation without using reverse/2
% is_palindrome(List) :-
%     palindrome_check(List, []).

% palindrome_check([], List).
% palindrome_check([X|Xs], Acc) :-
%     palindrome_check(Xs, [X|Acc]).
```

Output:

1 ?- is_palindrome([1, 2, 3, 2, 1]).

true.

2 ?- is_palindrome([a, b, c, c, b, a]).

true.

3 ?- is_palindrome([1, 2, 3, 4, 5]).

false.

5. To find the kth element of a list.

Example:

?- element_at(X, [a,b,c,d,e], 3).

X=c

```
% 5. To find the kth element of a list.

element_at([X|_], 1, X).
element_at([_|T], K, X) :-
    K > 1,
    K1 is K - 1,
    element_at(T, K1, X).
```

Output:

1 ?- element_at([a,b,c,d,e],3,X).

X = c .

2 ?- element_at([1,2,2,5,6,7],4,X).

X = 5 .

3 ?- element_at([a,b,c,d,e],6,X).

false.

6. To find the sum and average of all elements of a list using sum and length.

```
% 6. To find the sum and average of all elements of a list using sum and
length.

sum_and_average(List, Sum, Average) :-
    sum_list(List, Sum),
    length(List, Length),
    Average is Sum / Length.
```

Output:

1 ?- sum_and_average([1, 2, 3, 4, 5], Sum, Average).

Sum = 15,

Average = 3.

2 ?- sum_and_average([10, 20, 30], Sum, Average).

Sum = 60,

Average = 20.

7. To find gcd of two integers.

```
% 7. % Base case: GCD of a number and 0 is the number itself.
gcd(X, 0, X) :- X > 0.
gcd(0, Y, Y) :- Y > 0.
% Recursive case: GCD using the Euclidean algorithm.
gcd(X, Y, GCD) :-
    X > 0,
    Y > 0,
    X >= Y,
    Z is X mod Y,
    gcd(Y, Z, GCD).

gcd(X, Y, GCD) :-
    X > 0,
    Y > 0,
    X < Y,
    gcd(Y, X, GCD).
```

Output:

1 ?- gcd(48, 18, Result).

Result = 6 .

2 ?- gcd(35, 21, Result).

Result = 7 .

8. To determine whether a given integer number is prime or not.

Example:

?- is_prime(7).

True

```
% 8. To determine whether a given integer number is prime or not.
is_prime(2).
is_prime(3).
is_prime(P) :-
    P > 3,
    P mod 2 =\= 0,
    \+ has_factor(P, 3).

has_factor(N, Factor) :-
    Factor * Factor =< N,
    (N mod Factor =:= 0 ; NextFactor is Factor + 2, has_factor(N,
NextFactor)).
```

Output:

1 ?- is_prime(10).

false.

2 ?- is_prime(7).

true.

9. To determine the prime factors of a given positive integer.

Construct a flat list containing the prime factors in ascending order.

Example:

?- prime_factors(315, L).

L = [3,3,5,7]

```
is_prime(N) :- N > 1, is_prime_helper(N, 2).

is_prime_helper(N, I) :- I > sqrt(N), !.
```

```
is_prime_helper(N, I) :- N mod I =\= 0, I_next is I + 1, is_prime_helper(N,
I_next).

prime_factors(N, L) :- prime_factors_helper(N, 2, L).

prime_factors_helper(1, _, []) :- !.

prime_factors_helper(N, I, [I|L]) :- is_prime(I),  N mod I =:= 0, N1 is N //
I, prime_factors_helper(N1, I, L).

prime_factors_helper(N, I, L) :- I_next is I + 1, prime_factors_helper(N,
I_next, L).
```

Output:

1 ?- prime_factors(315, L).

L = [3, 3, 5, 7] .

2 ?- prime_factors(212,L).

L = [2, 2, 53] .

10. To determine Goldbach's conjecture

Goldbach's conjecture says that every positive even number greater than 2 is the sum

of two prime numbers. Example: 28 = 5 + 23. It is one of the most famous facts in

number theory that has not been proved to be correct in the general case. It has been

numerically confirmed up to very large numbers. Write a predicate to find the two prime

numbers that sum up to a give even integer.

Example:

?- goldbach(28, L).

L = [5, 23]

```
is_prime(N) :- N > 1, is_prime_helper(N, 2).

is_prime_helper(N, I) :- I > sqrt(N), !.

is_prime_helper(N, I) :- N mod I =\= 0, I_next is I + 1, is_prime_helper(N,
I_next).

goldbach(N, L) :- goldbach_helper(N, 2, 0, L).

goldbach_helper(1, _, 0, []) :- !.

goldbach_helper(N, I, C, [I | L]) :- C < 2, C1 is C + 1, is_prime(I), I1 is N
- I, is_prime(I1), goldbach_helper(N, I1, C1, L), !.

goldbach_helper(N, I, C, L) :- I >= N, !.

goldbach_helper(N, I, C, L) :- I_next is I + 1, goldbach_helper(N, I_next, C,
L).
```

Output:

1 ?- goldbach(28, L).

L = [5, 23|_]

2 ?- goldbach(30,L).

L = [7, 23|_]

11. To generate first N Fibonacci numbers.

```
fib(0, [0]).
fib(1, [0, 1]).
fib(2, [0, 1, 1]).

fib(N, L) :-
    N > 2,
```

```
    N1 is N - 1,
    N2 is N - 2,
    fib(N1, F1),
    last(F1, L1),
    fib(N2, F2),
    last(F2, L2),
    L_new is L1 + L2,
    append(F1, [L_new], L).
```

Output:

1 ?- fib(5,L).

L = [0, 1, 1, 2, 3, 5] .

2 ?- fib(7,L).

L = [0, 1, 1, 2, 3, 5, 8, 13]

12. Consider a database of facts that describe parent rela4onships as well as gender

rela4onships. The predicate parent(john,ann) is interpreted as: "John is a parent of

Ann". The predicate male(john) is interpreted as: "John is a man". The predicate

female(ann) is interpreted as: "Ann is a woman". So an example database of facts is:

parent(john,ann).

parent(jim,john).

parent(jim,keith).

parent(mary,ann).

parent(mary,sylvia).

parent(brian,sylvia).

male(keith). male(jim).

female(sylvia).

female(ann).

male(brian).

Note that some things are not specified in the database above (e.g., male(john)).

a) Write a Prolog predicate uncle(X,Y) that is true if X is Y's uncle. Note that we are

not considering uncles "by marriage", meaning that for X to be Y's uncle the two

must be related by blood. For instance (user input is in red):

?- uncle(keith,ann).

Yes

?- uncle(ann,mary).

No

?- uncle(keith,X).

X = ann ;

No

?- uncle(john,ann).

No

?- uncle(X,Y).

X = keith

Y = ann;

No

b) Write a Prolog predicate halfsister(X,Y) that is true if X is Y's half-sister. For

instance (user input is in red):

?- halfsister(ann,sylvia).

Yes

?- halfsister(X,sylvia).

X=ann ;

No

?- halfsister(X,Y).

X=ann

X=sylvia ;

X=sylvia

X=ann ;

No

```
parent(john,ann).
parent(jim,john).
parent(jim,keith).
parent(mary,ann).
parent(mary,sylvia).
parent(brian,sylvia).
male(keith).
male(jim).
female(sylvia).
female(ann).
male(brian).

uncle(X, Y) :- male(X), parent(Z, Y), parent(A, Z), parent(A, X).

half_sister(X, Y) :- female(X), parent(Z, X), parent(Z, Y), parent(A, X), A \=
Z, parent(B, Y), B \= Z, A \= B.
```

Output:

1 ?- uncle(keith,ann).

true .

2 ?- uncle(ann,mary).

false.

3 ?- uncle(keith,X).

X = ann .

4 ?- uncle(john,ann).

false.

5 ?- uncle(X,Y).

X = keith,

Y = ann .

1 ?-half_sister(ann,sylvia).

true .

2 ?- half_sister(X,sylvia).

X = ann .

3 ?- half_sister(X,Y).

X = sylvia,

Y = ann .