Name: Gourav Kumar Shaw
Enrolment No.: 2020CSB010
Subject: Artificial Intelligence Laboratory.
Semester: 8th

Assignment - 2

1. To duplicate the elements of a list, a given number of times.

Example:

?- duplicate([a,b,c],3,X).

{X = [a, a, a, b, b, b, c, c, c]}

Code:

```
append_lists([], L, L).
append_lists([H|T1], L2, [H|Result]) :-
append_lists(T1, L2, Result). append_K_times(_,0,[]).
append_K_times(H,N,[H|R]):- N>0, N1 is N-1,
append_K_times(H,N1,R).
duplicate([],_,[]). duplicate([H|R],N,X):-
    duplicate(R,N,Remains),
    append_K_times(H,N,HeadNtimes),
    append_lists(HeadNtimes,Remains,X).
```

Output:

1 ?- duplicate([1,2,3],2,Res).

Res = [1, 1, 2, 2, 3, 3] .

2 ?- duplicate([a,b,c],3,X).

X = [a, a, a, b, b, b, c, c, c] .

2. To determine whether a list is a sub list of another list. A list is a sub list of another list

if it's elements are present in another list consecutively and in the same order.

Code:

```
is_sub_list([],_).
is_sub_list([X|R1], [X|R2]):-
 check_sublist(R1,R2).
is_sub_list([X|R1],[_|R2]):-
  is_sub_list([X|R1],R2).

check_sublist([],_).
check_sublist([X|R1],[X|R2]):-
  check_sublist(R1,R2).
```

Output:

1 ?- is_sub_list([1,2,3], [a,b,c,1,2,3,d,e,f]).

true .

2 ?- is_sub_list([2,4],[a,b,c,d,1,9,3]).

false.

3. To determine intersection, union, difference, symmetric difference of two sets.

Code:

```
union([],L,L).
union([X|R],L,Res):-
  member(X,L),
  union(R,L,Res).
union([X|R], L,Res):-
  \+ member(X,L),
   union(R,L,R1),
   append([X],R1, Res).
```

```
intersection([],_,[]).
intersection([X|R],L, Res):-
   member(X,L),
   intersection(R,L,Res1),
   append([X],Res1,Res).
intersection([X|R],L,Res):-
   \+ member(X,L),
    intersection(R,L,Res).
subtract([],_,[]).
subtract([H|R],I,Res):-
   member(H,I),
   subtract(R,I,Res).

subtract([H|R],I,Res):-
 \+ member(H,I),
  subtract(R,I,Res1),
  append([H],Res1,Res).

sym_dif(L1,L2,Res):-
 union(L1,L2,U),
 intersection(L1,L2,I),
 subtract(U,I,Res).
```

Output:

1 ?- union([1,2,4,5,6],[2,3,4,6,7],Res).

Res = [1, 5, 2, 3, 4, 6, 7]

2 ?- intersection([1,2,4,5,5,6],[2,3,4,6,7,7],Res).

Res = [2, 4, 6] .

3 ?- subtract([1,2,4,5,6],[2,3,4,6,7],Res).

Res = [1, 5]

4 ?- sym_dif([1,2,4,5,6],[2,3,4,6,7],Res).

Res = [1, 5, 3, 7]

4. Transpose L1, L2 into L. That is, if L1 = [a, b, c] and L2 = [1, 2, 3], then L = [(a, 1), (b,

2), (c, 3)]

Code:

```
transpose([],[],[]).
transpose([H1|R1],[H2|R2],Res) :-
    transpose(R1,R2,Res1),
    append([(H1,H2)],Res1,Res).
```

Output:

1 ?-transpose([1,2,3],[a,b,c],Res).

Res = [(1, a), (2, b), (3, c)].

5. To split a list into two parts; the length of the first part is given.

Example:

?- split([a, b, c, d, e, f, g, h, i, j, k], 3, L1, L2).

L1 = [a, b, c], L2 = [d, e, f, g, h, i, k]

Code:

```
split(L, 0, [], L).
split([X|Xs], N, [X|L1], L2) :-
    N > 0,
    N1 is N - 1,
    split(Xs, N1, L1, L2).
```

Output:

1 ?- split([a, b, c, d, e, f, g, h, i, j, k], 3, L1, L2).

L1 = [a, b, c],

L2 = [d, e, f, g, h, i, j, k]

6. To extract a slice from a list. Given two indices, I and K, the slice is the list containing

the elements between the Ith and Kth element of the original list (both limits included).

Start counting the elements with 1.

Example:

?- slice([a, b, c, d, e, f, g, h, i, j, k], 3, 7, L).

L = [c, d, e, f, g]

Code:

```
/* slice(List,Start,End,Res) */
slice([H|_],1,1,[H]).
slice([H|R],1,End,Res):-
    End2 is End-1,
    slice(R,1,End2,Res2),
    append([H],Res2,Res).
slice([_|R], Start,End,Res):-
    Start >1,
    Start2 is Start-1,
    End2 is End-1,
    slice(R,Start2,End2,Res).
```

Output:

1 ?- slice([a,b,c,d,e,f,g,h,i,j],4,8,Res).

Res = [d, e, f, g, h]

7. Generate the combinations of K distinct objects chosen from the N elements of a list.

In how many ways can a committee of 3 be chosen from a group of 12 people? We all

know that there are C(12, 3) = 220 possibilities (C(N, K) denotes the well-known

binomial coefficients).

Example:

?- combinations(3, [a, b, c, d, e, f], L).

L = [a, b, c];

L = [a, b, d];

L = [a, b, e];

Code:

```
combinations(0,_,[]).
combinations(N,[_|R], Res):-
    N>0,
    combinations(N,R, Res).
combinations( N,[H|R], Res):-
    N>0,
    N2 is N-1,
    combinations(N2, R,Res1),
    append([H],Res1, Res).
```

Output:

1 ?- combinations(3,[a,b,c,d,e],Res).

Res = [c, d, e] ;

Res = [b, d, e] ;

Res = [b, c, e] ;

Res = [b, c, d] ;

Res = [a, d, e] ;

Res = [a, c, e] ;

Res = [a, c, d] ;

Res = [a, b, e] ;

Res = [a, b, d] ;

Res = [a, b, c] ;

8. Implement Bubble Sort, Insertion Sort, and Merge Sort.

Code :

```
/* bubble sort */
getHead([H|_], H).
getRest([_|R], R).
bubble_sort(List, Res):-
  bub_sort(List,List, Res).

bub_sort([],R,R).
bub_sort([_|R],List,Result):-
    bubble(List,Res1),
    bub_sort(R,Res1,Result).

bubble([H],[H]).
bubble([H|R],Result):-
    getHead(R,H2),
    H<H2,
    bubble(R,Res1),
    append([H],Res1, Result).
bubble([H|R], Result):-
    getHead(R,H2),
    H>=H2,
    getRest(R,Rest),
append([H], Rest, R1),
bubble(R1, Res1),
append([H2],Res1, Result).
```

Output:

1 ?- bubble_sort([7, 2, 5, 9, 1, 8, 3, 6, 4],Res).

Res = [1, 2, 3, 4, 5, 6, 7, 8, 9]

```
 /*
Insertion sort
for(int i=1;i<n; i++)
{
    int num = nums[i];
    int j = i-1;
    while(j>=0 && nums[j]>num)
    {
        nums[j+1] = nums[j];
        j--;
    }
       nums[j+1] = num;
    }

insertion_sort(List,sortedList)
*/
insertion_sort(List, Result):-
    i_sort(List,[], Result).
i_sort([],L, L).
i_sort([H|R],SortedList, Result):-
    insert_in_sorted(H,SortedList,NewSortedList),
    i_sort(R,NewSortedList,Result).

insert_in_sorted(E,[],[E]).
insert_in_sorted(E,[H|R],Res):-
    H<E,
    insert_in_sorted(E,R,Res1),
    append([H],Res1,Res).
insert_in_sorted(E,[H|R],Res):-
    H >= E,
    append([E],[H|R],Res).
```

Output:

1 ?- insertion_sort([7, 2, 5, 9, 1, 8, 3, 6, 4],Res).

Res = [1, 2, 3, 4, 5, 6, 7, 8, 9]

```
/* Implement merge_sort(Lis, Res) */
split(L,0,[],L).
split([H|R],N, L1,L2):-
 N2 is N-1,
 split(R,N2,Tl1,L2),
 append([H], Tl1,L1).

merge_sort([H],[H]).
merge_sort(List, Res):-
  length(List,Len),
  Len>1,
  Len2 is integer(Len/2),
  split(List,Len2, L1, L2),
  merge_sort(L1,Res1),
  merge_sort(L2,Res2),
  merge(Res1,Res2,Res).

merge([],L,L).
merge(L,[],L).
merge([H1| R1], [H2| R2], Res):-
    H1 =< H2,
    merge(R1, [H2|R2], Res1),
    append([H1],Res1, Res).
merge([H1|R1], [H2|R2], Res):-
    H2 <H1,
    merge([H1|R1], R2, Res1),
    append([H2], Res1, Res).
```

Output:

1 ?- merge_sort([7, 2, 5, 9, 1, 8, 3, 6, 4],Res).

Res = [1, 2, 3, 4, 5, 6, 7, 8, 9]

9. Pack consecutive duplicates of list elements into sub lists. If a list contains repeated

elements they should be placed in separate sub lists. Also, consecutive duplicates of

elements are encoded as terms [N, E] where N is the number of duplicates of the

elements E.

Example:

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

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

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

X = [[4, a], [1, b], [2, c], [2, a], [1, d], [4, e]]

Code:

```
/*
pack([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).
X = [[a, a, a, a], [b], [c, c], [a, a], [d], [e, e, e, e]]
pack(L,X)
*/
getHead([H|_], H).
get_all_consecutive(_,[],[], []).
get_all_consecutive(E,[E|R],Res, Rem):-
    get_all_consecutive(E,R,Res1,Rem),
    append([E],Res1,Res).
get_all_consecutive(_,L,[], L).
pack([],[]).
pack([H|R],Res):-
    get_all_consecutive(H,[H|R],Chs,Rem),
    pack(Rem,Res1),
    append([Chs],Res1,Res).

enc(Cnt,E,[Cnt,E]).
encode([],[]).
encode([H|R],Res):-
  length(H,Cnt),
  getHead(H,E),
  enc(Cnt,E, EncodedH),
```

```
  encode(R,Res1),
  append([EncodedH],Res1,Res).
```

Output:

1 ?- pack([a,a,a,b,c,c,a,a,d,d,d],Res).

Res = [[a, a, a], [b], [c, c], [a, a], [d, d, d]] .

2 ?- encode([[a, a, a], [b], [c, c], [a, a], [d, d, d]],Res).

Res = [[3, a], [1, b], [2, c], [2, a], [3, d]].

10. Consider a database of smoothie stores. Each store has a name, a list of employees,

and a list of smoothie that can be purchased in the store, which are encoded in a

store predicate. Each smoothie is defined by a name, a list of fruits, and a price,

which are encoded in a smoothie predicate. For example, here are three predicates

defining three different smoothie stores:

store(best_smoothies, [alan,john,mary],

[smoothie(berry, [orange, blueberry, strawberry], 2),

smoothie(tropical, [orange, banana, mango, guava], 3),

smoothie(blue, [banana, blueberry], 3)]).

store(all_smoothies, [keith,mary],

[smoothie(pinacolada, [orange, pineapple, coconut], 2),

smoothie(green, [orange, banana, kiwi], 5),

smoothie(purple, [orange, blueberry, strawberry], 2),

smoothie(smooth, [orange, banana, mango],1)]).

store(smoothies_galore, [heath,john,michelle],

[smoothie(combo1, [strawberry, orange, banana], 2),

smoothie(combo2, [banana, orange], 5),

smoothie(combo3, [orange, peach, banana], 2),

smoothie(combo4, [guava, mango, papaya, orange],1),

smoothie(combo5, [grapefruit, banana, pear],1)]).

The first store has three employees and sells three different smoothies, the second

store has two employees and sells four different smoothies, and the third store has

three employees and sells five different smoothies.

You can assume that there are no duplicates (pineapple is not listed twice in any

ingredient list, mary is not listed twice in any employee list, the same smoothie

specificaEon is not listed twice in any store menu, etc.). Given a database of

smoothie store facts, the quesEons below have you write predicates that implement

queries to the database.

a) Write a Prolog predicate more_than_four(X) that is true if store X has four or more

smoothies on its menu. For instance:

?- more_than_four(best_smoothies).

No

?- more_than_four(X).

X = all_smoothies ;

X = smoothies_galore ;

No

b) Write a Prolog predicate exists(X) that is true if there is a store that sells a

smoothie named X. For instance:

?- exists(combo1).

Yes

?- exists(slimy).

No

?- exists(X).

X = berry ;

X = tropical <enter>

Yes

c) Write a Prolog predicate ratio(X,R) that is true if there is a store named X, and if R is

the raEo of the store's number of employees to the store's number of smoothies on

the menu. For instance:

?- ratio(all_smoothies,R).

R = 0.5 ;

No

?- ratio(Store,R).

Store = best_smoothies

R = 1 ;

Store = all_smoothies

R = 0.5 ;

Store = smoothies_galore

R = 0.6 ;

No

Hint you may need to define a helper predicate to implement ratio

d) Write a Prolog predicate average(X,A) that is true if there is a store named X, and if

A is the average price of the smoothies on the store's menu. For instance:

?- average(best_smoothies,A).

A = 2.66667 ;

No

Hint you may need to define mulEple helper predicates to implement average

e) Write a Prolog predicate smoothies_in_store(X,L) that is true if there is a store

named X, and if L is the list of smoothie names on the store's menu. For instance:

?- smoothies_in_store(all_smoothies,L).

L = [pinacolada, green, purple, smooth] ;

No

?- smoothies_in_store(Store,L).

Store = best_smoothies

L = [berry, tropical, blue] ;

Store = all_smoothies

L = [pinacolada, green, purple, smooth] ;

Store = smoothies_galore

L = [combo1, combo2,

combo3, combo4, combo5] ;

No

Hint you may need to define a helper predicate to implement smoothies_in_store

f) Write a Prolog predicate fruit_in_all_smoothies(X,F) that is true if there is a fruit

F that is an ingredient of all smoothies on the menu of store X. For instance:

?- fruit_in_all_smoothies(Store,orange).

Store = all_smoothies ;

No

Hint you may need to define mulEple helper predicates to implement

fruit_in_all_smoothies

Code:

```
* Database Predicates */

store(best_smoothies, [alan,john,mary],
[ smoothie(berry, [orange, blueberry, strawberry], 2),
smoothie(tropical, [orange, banana, mango, guava], 3),
smoothie(blue, [banana, blueberry], 3) ]).
 store(all_smoothies, [keith,mary],
[ smoothie(pinacolada, [orange, pineapple, coconut], 2),
 smoothie(green, [orange, banana, kiwi], 5),
 smoothie(purple, [orange, blueberry, strawberry], 2),
smoothie(smooth, [orange, banana, mango],1) ]).
 store(smoothies_galore, [heath,john,michelle],[
smoothie(combo1, [strawberry, orange, banana], 2),
smoothie(combo2, [banana, orange], 5),
 smoothie(combo3, [orange, peach, banana], 2),
 smoothie(combo4, [guava, mango, papaya, orange],1),
smoothie(combo5, [grapefruit, banana, pear],1) ]).

  /* a) Write a Prolog predicate more_than_four(X) that is
true if store X has four or more smoothies on its menu.
For instance: ?- more_than_four(best_smoothies). No ?-
more_than_four(X). X = all_smoothies ; X =
smoothies_galore ; No */

more_than_four(X):-
store(X,_,Smoothies),
length(Smoothies,Len),
Len>=4.
/* b) Write a Prolog predicate exists(X) that is true if
there is a store that sells a smoothie named X. For
instance: ?- exists(combo1). Yes ?- exists(slimy). No ?-
exists(X). X = berry ; X = tropical <enter> Yes */

exists(X):-
store(_,_,Smoothies),
```

```
member(smoothie(X,_,_),Smoothies).

/* c) Write a Prolog predicate ratio(X,R) that is true if
there is a store named X, and if R is the ratio of the
store's number of employees to the store's number of
smoothies on the menu. For instance: ?-
ratio(all_smoothies,R). R = 0.5 ; No ?- ratio(Store,R).
Store = best_smoothies R = 1 ; Store = all_smoothies R =
0.5 ; Store = smoothies_galore R = 0.6 ; No Hint you may
need to define a helper predicate to implement ratio */

ratio(X,R):-
store(X,Emplist,Smoothielist),
length(Emplist, No_of_Employees),
length(Smoothielist,No_of_Smoothies),
 R is No_of_Employees/No_of_Smoothies.

/* d) Write a Prolog predicate average(X,A) that is true
if there is a store named X, and if A is the average price
of the smoothies on the store's menu. For instance: ?-
average(best_smoothies,A). A = 2.66667 ; No */
price(smoothie(_,_,Price),Price).
sumup([],0). sumup([H|R], Sum):-
price(H,Price),
sumup(R,Sum2),
 Sum is Price + Sum2.
    average(X,A):-
    store(X,_,SmoothieList),
    sumup(SmoothieList,Sum),
length(SmoothieList,No_of_smoothies),
A is Sum/No_of_smoothies.

/* e) Write a Prolog predicate smoothies_in_store(X,L)
that is true if there is a store named X, and if L is the
list of smoothie names on the store's menu. For instance:
?- smoothies_in_store(all_smoothies,L). L = [pinacolada,
green, purple, smooth] ; No ?-
smoothies_in_store(Store,L). Store = best_smoothies L =
[berry, tropical, blue] ; Store = all_smoothies L =
[pinacolada, green, purple, smooth] ; Store =
smoothies_galore L = [combo1, combo2,
combo3, combo4, combo5] ; No */
```

```
getName(smoothie(Name,_,_),Name).
getSmoothieNames([],[]).
getSmoothieNames([H|R], Res):-
 getName(H,Name),
 getSmoothieNames(R,Res1),
 append([Name],Res1,Res).
smoothies_in_store(X,L):-
     store(X,_,SmoothieList),
getSmoothieNames(SmoothieList,L).

/* f) Write a Prolog predicate fruit_in_all_smoothies(X,F)
that is true if there is a fruit F that is an ingredient
of all smoothies on the menu of store X. For instance: ?-
fruit_in_all_smoothies(Store,orange). Store =
all_smoothies ;
No */

is_in(F,smoothie(_,Lof,_)):-
member(F,Lof).
is_present(_,[]).
is_present(F,[H|R]):-
     is_in(F,H),
     is_present(F,R).
fruit_in_all_smoothies(X,F):-
     store(X,_,SmoothieList),
     is_present(F,SmoothieList).
```

Output:

?- more_than_four(X).
X = all_smoothies ;
X = smoothies_galore.

```
?- exists(X).
X = berry ;
X = tropical ;
X = blue ;
X = pinacolada ;
X = green ;
X = purple ;
X = smooth ;
X = combo1 ;
X = combo2 ;
X = combo3 ;
X = combo4 ;
X = combo5.
?- ratio(Store,R).
Store = best_smoothies,
R = 1 ;
Store = all_smoothies,
R = 0.5 ;
Store = smoothies_galore,
R = 0.6.
```

?- average(best_smoothies,A).
A = 2.6666666666666665.
?- average(S,A).
S = best_smoothies,
A = 2.6666666666666665 ;
S = all_smoothies,
A = 2.5 ;
S = smoothies_galore,
A = 2.2.
?- smoothies_in_store(X,L).
X = best_smoothies,
L = [berry, tropical, blue] ;
X = all_smoothies,
L = [pinacolada, green, purple, smooth] ;
X = smoothies_galore,
L = [combo1, combo2, combo3, combo4, combo5].
?- fruit_in_all_smoothies(S,F).
S = all_smoothies,
F = orange ;
false