Name – Indranil Bain DEPT. – Computer Science & Technology
Enrolment No. – 2020CSB039, Assignment - 02
Subject – Artificial Intelligence LAB

---

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]}

-
  ```
  duplicate_helper(A, 1, [A]).
  duplicate_helper(A, X, L) :- X>1, X1 is X-1, duplicate_helper(A, X1, L2), L=[AIL2] •
  duplicate(1],-,|).
  duplicate(IH|T], X, L2) :- duplicate_helper(H, X, L3), duplicate(T, X, L4), append(L3,
  L4, L2).
  ```

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.

```
     ➢ pare(_,_,_,0).
compare([],[],_,_).
 compare([L1H|L1T], [L2H|L2T], 1, Len1) :- L1H = L2H, New_Len is Len1-1,
compare(L1T, L2T, 1, New_Len).
compare(L1, [L2H|L2T], Pos, Len1) :- Pos2 is Pos-1, Pos2 >= 1, compare(L1, L2T,
Pos2, Len1).
check_sub_list(L1, L2, Len1, Len2, Pos) :- Pos+Len1 =< Len2+1, compare(L1, L2, Pos,
Len1) ; Pos2 is Pos+1, Pos =< Len2, check_sub_list(L1, L2, Len1, Len2, Pos2).
sub_list(L1, L2) :- length(L1, Len1), length(L2, Len2), check_sub_list(L1, L2, Len1,
Len2, 1)
```

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

```
     ➢ union([X], L2, L3) :- not(sub_list([X], L2)), L3=[X|L2]; L3=L2.
     union([L1H|L1T], L2, L3) :- union(L1T, L2, L4), not(sub_list([L1H], L4)),
     L3=[L1H|L4]; union(L1T, L2, L4), L3=L4.
```

```
     ➢ intersection([], _, []).
intersection([L1H|L1T], L2, L3) :- sub_list([L1H], L2), intersection(L1T, L2, L4),
append([L1H], L4, L3); not(sub_list([L1H], L2)), intersection(L1T, L2, L3).
```

```
    ➢ difference([], _, []).
    difference([L1H|L1T], L2, L3) :- not(sub_list([L1H], L2)), difference(L1T, L2, L4),
    append([L1H], L4, L3); sub_list([L1H], L2), difference(L1T, L2, L3).

symm_diff_helper([], _, []).
symm_diff_helper([L4H|L4T], L5, L3) :- not(sub_list([L4H],L5)),
symm_diff_helper(L4T, L5, L6), append([L4H], L6, L3); sub_list([L4H],L5),
symm_diff_helper(L4T, L5, L3).
    symmetric_diff(L1, L2, L3) :- union(L1, L2, L4), intersection(L1, L2, L5),
    symm_diff_helper(L4, L5, L3).
```

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)]

```
    ➢ transpose([], [], []).
transpose([L1H|L1T], [L2H|L2T], L3) :- transpose(L1T, L2T, L4),
append([(L1H,L2H)], L4, L3).
```

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]

```
    ➢ split_list([], _, [], []).
split_list([L1H|L1T], Pos, L2, L3) :- Pos > 0, Pos2 is Pos-1, split_list(L1T, Pos2, L4,
L3), append([L1H], L4, L2); Pos =< 0, split_list(L1T, Pos, L2, L5), append([L1H], L5,
L3).
```

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]

```
    ➢ slice(L, Pos1, Pos2, L1) :- Pos3 is Pos1-1, split_list(L, Pos3, L2, L3), Pos4 is
      Pos2-Pos1+1, split_list(L3, Pos4, L4, L5), L1=L4.
```

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];
...
```

```
➢ combinations(1, [L1H|L1T], L2) :- L2=[L1H]; combinations(1, L1T, L2).
combinations(X, [L1H|L1T], L2) :- X>1, X2 is X-1, combinations(X2, L1T, L3),
append([L1H], L3, L2); X>1, combinations(X, L1T, L2).
```

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

```
➢ bubble_swap([X], [X]).
bubble_swap([X,Y|List], L2) :- X>Y, bubble_swap([X|List], L3), L2=[Y|L3].
bubble_swap([Z|List1], [Z|List2]) :- bubble_swap(List1, List2).
bubble_sort_helper(List1, List2, X) :- X>1, bubble_swap(List1,
L2),format('~w~n',[L2]), X2 is X-1, bubble_sort_helper(L2, List2, X2).
bubble_sort_helper(L2,L2,1).
bubble_sort(List1, List2) :- length(List1, Len), bubble_sort_helper(List1, List2, Len).
```

```
➢ swap([X,Y|List], [Y,X|List]) :- X>Y.
swap([Z|List], [Z|List1]) :- swap(List, List1).
insertion_sort(List1, List2) :- swap(List1, L2), format('~w~n',[L2]),
insertion_sort(L2, List2).
insertion_sort(L, L).

divide([],[], []).
divide([A], [A], []).
divide([A,B|L], [A|L1], [B|L2]) :- divide(L, L1, L2).

➢ merge(A, [], A).
merge([], B, B).
merge([A|L1], [B|L2], [A|L3]) :- A =< B, merge(L1, [B|L2], L3).
merge([A|L1], [B|L2], [B|L3]) :- A > B, merge([A|L1], L2, L3).

merge_sort([],[]).
merge_sort([A],[A]).
merge_sort([A,B|L], L2) :- divide([A,B|L], L3, L4), format('Divide: ~w ~w~n', [L3,
L4]), merge_sort(L3, S1), merge_sort(L4, S2), merge(S1, S2, L2), format('Merge:
~w~n', [L2]).
```

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]]
```

```
    ➢ pack([],[]).
pack([A],[[A]]).
pack([A,A|L1],[[A|L3]|L2]) :- pack([A|L1], [L3|L2]).
pack([A,B|L1], [[A]|L2]) :- dif(A,B), pack([B|L1], L2).

do_encoding([[A|L1]], L2) :- length(L1, Len1), Len2 is Len1+1, L2=[[Len2, A]].
do_encoding([[A|L1]|L2], L3) :- do_encoding(L2, L4), length(L1, Len1), Len2 is
Len1+1, append([[Len2,A]],L4,L3).
encode(L1, L2) :- pack(L1, L3), do_encoding(L3, L2).
```

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.

```
➢ 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, Emp, Smo), length(Smo, 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
```

```
➢ contains(X, [smoothie(X,_,_)|L]).
contains(X, [smoothie(Y,_,_)|L]) :- dif(X,Y), contains(X, L).
exists(X) :- store(_, _, Smo), contains(X, Smo).
```

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

```
➢ ratio(X, R) :- store(X, Emp, Smo), length(Emp, Len1), length(Smo, Len2), R is Len1/Len2.
```

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

```
➢ total_cost([smoothie(X,_,Cost)], Cost).
total_cost([smoothie(X,_,Cost)|L], Sum) :- total_cost(L, Sum2), Sum is Sum2+Cost.
average(X, A) :- store(X, _, Smo), total_cost(Smo, Sum), length(Smo, Len), A is Sum/Len.
```

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
```

```
    ➢ get_smoothies([smoothie(X,_,_)], [X]).
get_smoothies([smoothie(X,_,_)|L1], L2) :- get_smoothies(L1, L3), append([X], L3,
L2).
smoothies_in_store(X, L) :- store(X, _, Smo), get_smoothies(Smo, L).
```

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

```
    ➢ check_fruit([smoothie(_,Fruits,_)], F) :- sub_list([F], Fruits).
check_fruit([smoothie(_,Fruits,_)|L], F) :- sub_list([F], Fruits), check_fruit(L, F).
fruit_in_all_smoothies(X, F) :- store(X, _, Smo), check_fruit(Smo, F).
```