Prolog 1

CSE 4102 Project Homework 5, Spring 2018

Bryan Arnold

4/26/2018

Section: 001

Instructor: Jeffrey A. Meunier

## Introduction

For this assignment, I wrote a few Prolog predicates, or theorems. These are short predicates just to get used to using GNU as well as coding in Prolog.

## Output

| ?- consult('C:/GNU-Prolog/Homework 5/homework5.pro').

compiling C:/GNU-Prolog/Homework 5/homework5.pro for byte code...

C:/GNU-Prolog/Homework 5/homework5.pro compiled, 68 lines read - 5885 bytes written, 15 ms

yes

| ?- compress([], X).

X = []

yes

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

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

yes

| ?- my_flatten([], X).

X = []

yes
| ?- my_flatten([a, [b, [c, d], e]], X).

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

yes
| ?- pack([], X).

X = []

yes
| ?- 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]]

yes
| ?- rlencode([], O).

O = []

yes
| ?- rlencode([a, a, a, a, d, c, c, a, a, d, e, e, e, e], O).

O = [[a,4],[d,1],[c,2],[a,2],[d,1],[e,4]]

yes
| ?- rldecode([], O).

O = []

yes
| ?- rldecode([[a,4],[d,1],[c,2],[a,2],[d,1],[e,4]], O).

O = [a,a,a,a,d,c,c,a,a,d,e,e,e,e]

yes
| ?- range(2, 2, L).

L = [2]

yes
| ?- range(2, 10, L).

L = [2,3,4,5,6,7,8,9,10]

yes

## Source Code

```
/* Prolog 1 */
/* CSE 4102 Project Homework 5, Spring 2018 */
/* Bryan Arnold */
/* 4/26/2018 */
/* Section: 001 */
/* Instructor: Jeffrey A. Meunier */

/* compress Predicate */
```

```
/* This predicate is responsible for taking a query that is a */
/* list of consecutive ground terms and collapses them. */
/* Example: [a, a, b, c, c, a, a] would be the query, and the response */
/* would be [a, b, c, a]. Use this predicate when you want to collapse */
/* successive ground terms into just one. */
compress([], []).
compress([X | [X | Ys]], Z) :- !, compress([X | Ys], Z).
compress([ X | Xs], [X | Ys]) :- compress(Xs, Ys).

/* my_flatten Predicate */
/* This predicate is responsible for taking a query of a list of lists */
/* and responding with one list with the same ground terms that were in the */
/* original query. Example: [a, [b, [c, d], e]] response is */
/* [a, b, c, d, e]. Use this predicate when you want to turn multiple lists */
/* within one list into a single list with the same ground terms. */
my_flatten([], []) :- !.
my_flatten([X | Xs], Y) :- !, my_flatten(X, A1), my_flatten(Xs, A2), append(A1, A2, Y).
my_flatten(X, [X]).

/* pack Predicate */
/* This predicate takes a query of a list of ground terms and */
/* returns a grouping of lists within a list of consecutive grounds terms. */
/* Example: [a, a, b, c, c, a, a] would respond with */
/* [[a, a], [b], [c, c], [a, a]]. Use this predicate when you want to group */
/* consecutive ground terms in a list together. */
pack([], []).
pack([X], [[X]]) :- !.
pack([X | [X | Xs]], [[X | Ys] | Z]) :- !, pack([X | Xs], [Ys | Z]).
pack([X | [Y | Ys]], [[X] | Zs]) :- pack([Y | Ys], Zs).
```

```
/* rlencode Predicate */
/* This predicate takes a query of a list and responds with a run-length */
/* encoding of a list with sequences of repetition by utilizing the pack predicate. */
/* So, basically the same as pack, but instead of the groupings, the ground term */
/* with the number of terms grouped together are returned as the sublists. */
/* Example: [a, a, b, c, c, a, a] responds with [[a, 4], [b, 1], [c, 2], [a, 2]]. */
/* Use this predicate when you want to know how many times each grouping of ground terms */
/* within a list occur together. */
rlencode(X, Y) :- pack(X, Z), rlencode2(Z, Y).
rlencode2([], []).
rlencode2([[X | Xs] | Ys], [[X, Z] | Zs]) :- length([X | Xs], Z), rlencode2(Ys, Zs).

/* rldecode Predicate */
/* This predicate takes a query of the format that rlencode responds with, and returns */
/* the original list, so the reverse of rlencode. Example: [[a, 4], [b, 1], [c, 2], [a, 2]] */
/* responds with [a, a, b, c, c, a, a]. Use this predicate to get the original list that */
/* rlencode altered. Note: in the assignment, this predicate was said to utilize pack and */
/* rlencode, but I don't see how. The query is formatted like rlencode result, so using */
/* rlencode and pack within it wouldn't work, as it would be 3-laters deep of lists. */
rldecode([], []).
rldecode([[X, 1] | Y], [X | Z]) :- rldecode(Y, Z).
rldecode([[X, A] | Y], [X | Z]) :- B is A-1, rldecode([[X, B] | Y], Z), !.
rldecode([X | Y], [X | Z]) :- rldecode(Y, Z).

/* range Predicate */
/* This predicate creates a list of integers ranging from a lower number to a higher number */
/* (is assumed that the lower number cannot be greater than the higher number). Example: */
/* range(1, 3, L) responds with L = [1, 2, 3]. Use this predicate to see what numbers are */
/* between the upper and lower limit and put them into a list. */
range(X, X, [X]) :- !.
```

```
range(X, Y, [X | Zs]) :- !, A is X + 1, range(A, Y, Zs).
```