SER 502

Mini Assignment-1 Prolog

P01 (*) Find the last element of a list.

Code:

```
isLast(X, [X]).

isLast(X, [_|L]):- isLast(X, L).
```

Sample Run1:

isLast(X, [1,2,3,4,5,6]).

Output: X = 6

isLast(X, [1,2,3,4]).

Output: X = 4

P02 (*) Find the last but one element of a list.

Code:

```
Previous_list_element(X, [X, _]).

Previous_list_element(X, [_|T]) :- previous_list_element(X,T).
```

Sample Run1:

previous_list_element(X, [1,3,5,7,9]).

Output: 7

previous_list_element(4, [1,2,3,4,5]).
Output: true

Sample Run2:

P03 (*) Find the K'th element of a list.

Code:

```
element_at(X,1,[X|_]).

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

Sample Run1:

element_at(X,4, [1,4,5,6,2,3,8]).

Output: X = 6

Sample Run 2:

element_at(X,3, [1,4,5,6,2,3,8]).

Output: X = 5

P04 (*) Find the number of elements of a list.

Code:

```
no_of_elements(0,[]).

no_of_elements(N,[_ |T]):- no_of_elements(N1, T), N is N1 + 1.
```

Sample Run1:

no_of_elements(X,[2,5,7,3,1,6]).

Output:

X = 6

Sample Run2:

no_of_elements(X,[2,5,7,3,1,6,7,8,9,5]).

Output:

X = 10

P05 (*) Reverse a list.

Code:

```
my_reverse(X, R) :- my_reverse_(X, R, []).

my_reverse_([],R,R).
```

my_reverse_([X|X_], R, A):- my_reverse_(X_, R, [X|A]).

Sample Run1:

reverse([1,2,3,4], X).

Output:

X = [4, 3, 2, 1]

Sample Run2:

reverse([a,b,c,d], X).

Output:

X = [d, c, b, a]

P06 (*) Find out whether a list is a palindrome.

Code:

is_palindrome([]).

is_palindrome([_]).

is_palindrome([H|T]) :-append(Mid, [H], T),is_palindrome(Mid).

Sample Run1:

is_palindrome([r,a,c,e,c,a,r]).

Output:

true

Sample Run2:

is_palindrome([r,o,h,a,n]).

Output:

False

P07 (**) Flatten a nested list structure.

Code:

```
my_flatten(X, [X]) :- \+ is_list(X).

my_flatten([], []).

my_flatten([X|Xs], Y) :- my_flatten(X, X_), my_flatten(Xs, Xs_), append(X_, Xs_, Y).
```

Sample Run1:

my_flatten([a,b,[c,[d]],e], X).

Output:

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

Sample Run2:

my_flatten([a,b,[c,[d],e],f], X).

Output:

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

P08 (**) Eliminate consecutive duplicates of list elements.

Code:

```
compress([], []).

compress([X], [X]).

compress([X,X|X_], Y) :- compress([X|X_], Y).

compress([X,Z|X_], [X|Y]) :- X \= Z, compress([Z|X_], Y).
```

Sample Run1:

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

Output:

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

Sample Run2:

compress([1,1,1,2,3,3,3,3,4,4,5], X).

Output:

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

P09 (**) Pack consecutive duplicates of list elements into sublists.

Code:

```
pack_duplicates([], []).

pack_duplicates([X], [[X]]).

pack_duplicates([X|Xs], [[X,X|P]|T]) :- pack_duplicates(Xs, [[X|P]|T]).

pack_duplicates([X|Xs], [[X],[Y|P]|T]) :- pack_duplicates(Xs, [[Y|P]|T]), X \= Y.
```

Sample Run1:

pack_duplicates([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).

Output:

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

Sample Run2:

pack_duplicates([a, a, a, 1, 1, 1, 1, c, c, 2, 2, d, e, e, e, e], X).

Output:

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

P10 (*) Run-length encoding of a list.

Code:

```
encode([], []).

encode([X], [[1, X]]).

encode([X|Xs], [[N_, X]|E]) :- encode(Xs, [[N, X]|E]), N_ is N + 1.

encode([X|Xs], [[1, X], [N, Y]|E]) :- encode(Xs, [[N, Y]|E]), X \= Y.
```

Sample Run1:

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

Output:

X = [[3, a], [2, b], [2, c], [1, d], [2, b], [3, e]]

Sample Run2:

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

Output:

X = [[3, a], [2, b], [2, c], [1, d], [2, b], [3, c]]