User's Manual and Tutorial

Table of Contents:

Introduction

This project is a simple List Processing Library written in Prolog. Querying the various functions in the program perform unique modifications on given lists. In order to run the program, it must be loaded into *swipl*. Query the program with lists for information using the functions below, descriptions of functions are also included.

Functions available to User:

- **write_list(List)**
    - *Parameter(s): List – list of objects*
    - When given a list of objects, this function iterates recursively through the list. While printing objects until it reaches the end of the list.
- **write_list_reversed(List)**
    - *Parameter(s): List – list of objects*
    - When given a list of objects, which iterates recursively through the list until it reaches the end of the list. Then prints the objects in reverse order while it steps out.
- **first(List, First)**
    - *Parameter(s): List – list of objects ; First – first object in list*
    - When given a List of objects this function, this function tries to match the first object with the First variable, returning true/false.
- **rest(List, Rest)**
    - *Parameter(s): List – list of objects ; Rest – all objects in list, excluding the first*
    - When given a List of objects this function, this function tries to match the list, excluding the first object, with the Rest variable returning true/false.
- **last(List, Last)**
    - *Parameter(s): List – list of objects ; Last – last object in the list*
    - When given a List of objects this function, this function tries to match the last object in the list, with the Last variable returning true/false.

- **nth(N, List, NthElement)**
    - *Parameter(s): N – index position in list ; List – list of objects ; NthElement – object at position N in list*
    - When given a List of objects this function, this function traverses the list until it reaches the object at the position N, matching it with the NthElement returning true/false.
- **item(N, List, Item)**
    - *Parameter(s): N – index position in list ; List – list of objects ; Item – object in position N in the list*
    - When given a List of objects this function, this function traverses the list until it reaches the object at the position N, matching it with the Item returning true/false.
- **member(X, List)**
    - *Parameter(s): X – object ; List – list of objects*
    - When given a List of objects this function, this function traverses the list until it reaches the object at the position N, matching it with the Item returning true/false.
- **count(Value, List, Count)**
    - *Parameter(s): Value – object in list ; List – list of objects ; Count – number of times Value appears in list*

- When given a List of objects this function, this function traverses the list matching Value with the objects. Matching Count with number of times Value is repeated in list.

- **append(List1, List2, Result)**
  - *Parameter(s): List1 – 1st List of objects ; List2 – 2nd list of objects ; Result – result of List1 and List2*
  - When given two lists of objects, List1 and List2, this function traverses List1 creating a new list of objects, Result, adding everything in List1 to it. Then traverses List2 adding objects to Result.
- **remove(N, List, NewList)**
  - *Parameter(s): N – index value ; List – list of objects ; NewList – result of object at index value removed*
  - When given an index value N, this function traverses the given list adding objects from List and adding it to NewList, skipping the object at index position N.
- **replace(N, Object, NewList)**
  - *Parameter(s): N – index value ; Object – single object ; NewList – result of object at index value replaced with Object*
  - When given an index value N, this function traverses the given list and adds objects from given list to NewList, replacing the value at that position with Object.

- **make_list(N, Object, NewList)**
    - *Parameter(s): N – number of iterations ; Object – single object ; NewList – list of Object, N times*
    - When given a value N, and Object, this function adds Object to NewList N times.
- **reverse(List, ReversedList)**
    - *Parameter(s): List – list of objects ; ReversedList – List reversed*
    - When given a List, this function iterates through that list recursively, adding the objects to ReversedList in reverse order.
- **add_first(E, List, NewList)**
    - *Parameter(s): E – object ; List – list of objects ; NewList – result of object E inserted at beginning of the List*
    - When given an object E and a List, this function recursively iterates through List adding objects where they were in original list, but adding E to the beginning.
- **pick(List, Item)**
    - *Parameter(s): List – list of objects ; Item – object*
    - When given a List, this function randomly chooses Item within the List.
- **add_last(E, List, NewList)**
    - *Parameter(s): E - object ; List – list of objects ; NewList – result of adding E to the end of List*
    - When given an object E, and List, this function adds all objects to NewList placing object E at the end.

- **take(List, O, NewList)**
    - *Parameter(s): List – list of objects ; O - object ; NewList – result of object O taken from List*
    - When given a List, and object O, this function iterates through List skipping over the first object matching O adding objects to NewList.
- **iota(N, IotaK)**
    - *Parameter(s): N – numeric value ; IotaK – numeric list from 0-N*
    - When given a number N, this function matches IotaK with an ordered list 0-N.
- **sum(List, Sum)**
    - *Parameter(s): List – numeric list ; Sum – result of all numbers in List added up*
    - When given a List, assuming it's numeric, this function traverses the list adding all numbers to equal Sum
- **product(List, Product)**
    - *Parameter(s): List – numeric list ; Product – result of all numbers in List multiplied*
    - When given a List, assuming it's numeric, this function traverses the list multiplying all numbers to equal Product
- **factorial(N, Factorial)**
    - *Parameter(s): N – numeric value ; Factorial – result of numbers 1-N added up, if N is 0 then Factorial is 1*
    - When given a numeric value N, this function recursively adds numbers 1-N to Factorial.

- **min(List, Minimum)**
    - *Parameter(s): List – numeric list ; Minimum – smallest number in List*
    - When given a List, assuming it's numeric, this function traverses all numbers in list finding the smallest number.
- **max(List, Maximum)**
    - *Parameter(s): List – numeric list ; Maximum – smallest number in List*
    - When given a List, assuming it's numeric, this function traverses all numbers in list finding the biggest number.
- **sort_inc(List, SortedList)**
    - *Parameter(s): List – numeric list ; SortedList – List sorted in order of smallest to largest*
    - When given a List, assuming it's numeric, this function traverses the numbers sorting them into SortedList in order of smallest to largest.
- **sort_dec(List, SortedList)**
    - *Parameter(s): List – numeric list ; SortedList – List sorted in order of largest to smallest*
    - When given a List, assuming it's numeric, this function traverses the numbers sorting them into SortedList in order of largest to smallest.
- **alist(List1, List2, AList)**
    - *Parameter(s): List1 – list of objects ; List2 – list of objects ; AList – list of objects in pairs*

    - When given a List1, and List2, assuming they are of equal length this function traverses both lists adding paired values to AList.

- **assoc(List, Key, Value)**
    - *Parameter(s): List – list of paired values ; Key – given Key ; Value – given Value*
    - When given a List of paired values, this function matches the Value to the pair with the given Key.
- **freq_count(List, FCList)**
    - *Parameter(s): List – list ; FCList – result of List objects paired with the number of times they appear in List*
    - When given a List, this function matches FCList to paired values with their frequency.
- **make_set(List, Set)**
    - *Parameter(s): List – list of objects ; Set – set of objects*
    - When given a List, this function adds all objects into Set except for duplicates.
- **flatten(List, FlattenedList)**
    - *Parameter(s): List – list of objects ; FlattenedList – list of objects*
    - When given a List, this function takes all nested lists and adds only the objects into FlattenedList.