Constraint Logic Programming (CLP)
Read Chapter 14 of Bratko

A very simple introduction to CLP(R) is in
http://www.sju.edu/~jhodgson/clp/howto_clpr.html

The problem is to use CLP to solve the following self-referential puzzle:
Given an integer \( n \), determine a list \( Z \) containing \( n \) elements (positive integers), i.e., \( Z = [X_1, X_2, X_3, \ldots, X_n] \) such that \( Z \) satisfies the properties:

(a) \( X_1 + X_2 + X_3 + \ldots + X_n = 2n \)
(b) \( 0*X_1 + 1*X_2 + 2*X_3 + \ldots + (n-1)*X_n = n*(n+1)/2 \)
(c) The integers \( X_1, X_2, \ldots X_n \) in \( Z \) are such that the number of 1’s in \( Z \) occurs \( X_1 - 1 \) times, the number of 2’s in \( Z \) occurs \( X_2 - 1 \) times and the number \( n \) occurs \( X_n - 1 \) times. In other words, \( X_i \) represents the number of elements \( i \) in \( Z \) plus one.

For example when \( n=7 \) your program should generate the list:
\( Z = [4, 3, 2, 2, 1, 1, 1] \)
the sum of elements of \( Z \) is \( 2*7 = 14 \) satisfying property (a);
also: \( 4*0 + 3*1 + 2*2 + 2*3 + 1*4 + 1*5 + 1*6 = 28 = 7*8/2 \) satisfying property (b);
notice that in \( Z \) there are 3 one’s, 2 two’s, one three, one four, no fives sixes or sevens and therefore the elements of \( Z \) are \( [4, 3, 2, 2, 1, 1, 1] \) satisfy property (c).

Your program should try to determine \( Z \) for \( n = 1,2,3,4,5,\ldots20 \)

An interesting pattern should emerge. Describe that pattern as \( n \) increases. Is there a number theory conjecture about it?
**Hints:**
You should use strict equality, and *do not utilize the Prolog “is”*. Your solution should include the predicates:

- **length(L, N)** - the length of list L is N (do not use “is”).
- **sum(L, S)** - the sum of the elements (integers) in list L is S (see property (a)).
- **wsum (L, WS)** – WS is the weighted sum of the elements of L (see property (b))
- **count (L, C)** – the list C is such that its first element is the number of ones in L plus 1, the second element is the number of two’s in L plus 1 and so forth (see property (c))

You may also want to use **member (X, [0,1,2,…])** to non-deterministically generate integers that are candidates for the lists you are determining.