In the previous network, inputs were configured such that the operation to perform was the first element in the list, the first input the second element, and the second input the third. We represented $a + b$ as $[+, a, b]$, and $a - b$ as $[-, a, b]$.

Suppose we want to evaluate the expression $a + b + c - d$ in the same way that was done in the last network. Since the expression is evaluated using prefix notation, using this expression as it is as input would result in an error. Instead, we must transform the expression into this:

$$- + +abcd$$

Keep in mind, though, that the expression which would be passed into the eval() function would look like this:

$$(-(+(+ab)c)d)$$

since the LISP interpreter is incapable of inferring the order of evaluation of the expression without the help of parenteses.

First, let us clearly define the desired transformation

$$[a, +, b, +, c, -, d] \rightarrow [-, +, +, a, b, c, d]$$

Which we may represent more generally as

$$[e_1, e_2, e_3, e_4, e_5, e_6, e_7] \rightarrow [e_6, e_4, e_2, e_1, e_3, e_5, e_7]$$

or by a seventh order permutation matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we will generalize this permutation matrix so that we can generate one every time we need to reformat a simple mathematical expression into prefix notation.

Consider another transformation

$$[a, +, b, -, c] \rightarrow [-, +, a, b, c]$$

We can express it as

$$[e_1, e_2, e_3, e_4, e_5] \rightarrow [e_4, e_2, e_1, e_3, e_5]$$

Let us now define a function which, given an input of an element's position in the first expression, will output it's position in the resulting expression

$$f(p) = (n + \frac{n-1}{2}) + \sum_{i=1}^{n}(-1)^p * p$$

Where n is the number of elements in the input vector.

We can apply this function to the expression to get

$$[e_1, e_2, e_3, e_4, e_5] \rightarrow [f(1), f(2), f(3), f(4), f(5)] \rightarrow [3, 2, 4, 1, 5]$$

Finally, we can define our permutation matrix of dimensions $n * n$ such that each element $a_{r,c}$ is zero except for those where $f(r) = c$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,5} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,5} \\ a_{3,1} & a_{3,2} & \cdots & a_{3,5} \\ a_{4,1} & a_{4,2} & \cdots & a_{4,5} \\ a_{5,1} & a_{5,2} & \cdots & a_{5,5} \end{bmatrix}^T$$

$$a_{r,c} = \begin{cases} 0, & f(r) \neq c \\ 1, & f(r) = c \end{cases}$$

When we use this procedure to generate a permutation matrix of size $n = 5$, we get

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

When we apply P to the expression $[a, +, b, -, c]$, we get $[-, +, a, b, c]$
To further confirm that our algorithm works, lets generate a permutation matrix of size $n = 7$

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Which is identical to the matrix we got for the first transformation.

When we apply this matrix to the expression $[a, +, b, +, c, -, d]$, sure enough, we get $[-, +, +, a, b, c, d]$