Write a program to flatten a list. The result is a new list just like the old one but without the internal structure. For example,

\[ L = [ [3; 5]; 2; [5; [7]; [nil]] ] \]
\[ L' = [3; 5; 2; 5; 7] \]

Your program may employ a test \( Li: int \) to see if an item is an integer or a list.

Define \( flat L \) to mean that list \( L \) is flat.

\[ flat L = \forall i: 0,..\#L: Li: int \]

or more succintly

\[ flat L = L: [*int] \]

Define \( sim L M \) to mean that lists \( L \) and \( M \) have the same items in the same order, though they may have different internal structure. Formally, it is easier to define \( sim \) for all strings. Let \( s, t, \) and \( u \) be strings, and let \( i \) and \( j \) be integers.

\[ sim s s = sim t s \]
\[ sim s t \land sim t u \Rightarrow sim s u \]
\[ sim s nil = s=nil \]
\[ sim (i; s) (j; t) = i=j \land sim s t \]
\[ sim (s; [t]; u) (s; t; u) \]

Define specifications \( P \) and \( Q \) as

\[ P = flat L' \land sim L L' \]
\[ Q = L'[0;..k]=L[0;..k] \land flat (L'[k;..#L']) \land sim (L[k;..#L]) \land (L'[k;..#L']) \]

Then the refinements are

\[ P \iff k:=0. \ Q \]
\[ Q \iff \text{if } k=\#L \text{ then } ok \]
\[ \text{else if } Lk: int \text{ then } k:=k+1. \ Q \]
\[ \text{else } L:=L[0;..k]+Lk+L[k+1;..#L]. \ Q \times fi \]