CS 257: Non-Imperative Programming: Scheme! Project Warm Up (Spring '05)

The following problems are warm up exercises for the final project, a BASIC to Scheme translator. Renumbering BASIC programs and eliminating *for* and *next* statements by translating them into *let*'s, *if-then*'s and *goto*'s will make the job of writing your translator significantly easier.

1. Write a function called *renumber* which takes a BASIC program represented as a list of numbered statements, *prog*, an initial line number, *start*, and an increment, *step* as arguments. It returns a program where the initial line number is *start* and where the difference between successive line numbers is *step*. Note that all line numbers, and all references to those line numbers in *goto*, *gosub*, and *if-then* statements, must be consistently renumbered. For example,

```
> (define quadratic2
'((100 input "What is the value of A" a)
  (110 input "What is the value of B" b)
  (120 input "What is the value of C" c)
  (130 let d = ((b * b) - (4 * (a * c))))
  (140 if (d < 0) then 190)
  (150 gosub 210)
  (160 print "The 1st root is: " (((-1 * b) + s) / (2 * a)))
  (170 print "The 2nd root is: " (((-1 * b) - s) / (2 * a)))
  (180 end)
  (190 print "Imaginary roots.")
  (200 end)
  (210 let s = 1)
  (220 for i = 1 to 10)
  (230 let s = ((s + (d / s)) / 2))
  (240 next i)
  (250 return)))
> (renumber quadratic2 1000 10)
((1000 input "What is the value of A" a)
 (1010 input "What is the value of B" b)
 (1020 input "What is the value of C" c)
 (1030 let d = ((b * b) - (4 * (a * c))))
 (1040 if (d < 0) then 1090)
 (1050 gosub 1110)
 (1060 print "The 1st root is: " (((-1 * b) + s) / (2 * a)))
 (1070 print "The 2nd root is: " (((-1 * b) - s) / (2 * a)))
 (1080 end)
 (1090 print "Imaginary roots.")
 (1100 end)
 (1110 let s = 1)
 (1120 for i = 1 to 10)
 (1130 let s = ((s + (d / s)) / 2))
 (1140 next i)
 (1150 return))
>
```

Note: Line feeds and spaces have been added to the above to improve clarity. Your function need not add line feeds or spaces. You may assume that the lines of the input program are numbered in order and that there are no duplicate line numbers. You may not assume that the line numbers increase uniformly–that is the point of renumbering.

2. Write a function called *transform-for-stmt* which takes a *for* statement from a BASIC program, *stmt*, and returns a *let* statement which will initialize the *for* statement's loop variable. For example,

```
> (transform-for-stmt '(10 for i = 1 to k))
((10 let i = 1))
>
```

3. Write a function called *transform-next-stmt* which takes a *next* statement, *stmt*, and a BASIC program, *prog*, as input. It returns a list of three BASIC statements. The first is an *if-then* statement, the second is a *let* statement and the third is a *goto* statement. These statements do the following

   - Compare the loop variable current value to its final value and jump to the line following the *next* statement if it exceeds it.
   - Increment the loop variable

- Jump to the line following the corresponding *for* statement.

For example,

```
> (define bar '((10 for i = 1 to k) (20 print i) (30 next i) (40 end)))
> (transform '(30 next i) bar)
((30 if (i >= k) then 40) (30.1 let i = (i + 1)) (30.2 goto 20))
>
```

4. Write a function called *transform* which renumbers a BASIC program with a starting line number of 0 and an increment of 1 and eliminates all *for* and *next* statements by translating them into *let*'s, *if-then*'s, and *goto*'s. For example,

```
> (define foo
'((10 for i = 1 to k)
  (20 print i)
  (30 next i)
  (40 end)))
> (transform foo)
((1 let i = 1)
 (2 print i)
 (3 if (i >= 10) then 6)
 (4 let i = (i + 1))
 (5 goto 2)
 (6 end))
>
```

Here is a second example:

```
> (define bar
'((10 for i = 1 to 10)
  (20 for j = 1 to 10)
  (30 print (i * j))
  (40 next j)
  (50 next i)
  (60 end)))
> (transform bar)
```

```
((1 let i = 1)
 (2 let j = 1)
 (3 print (i * j))
 (4 if (j >= 10) then 7)
 (5 let j = (j + 1))
 (6 goto 3)
 (7 if (i >= 10) then 10)
 (8 let i = (i + 1))
 (9 goto 2)
 (10 end))
>
```