Nested Radicals

Here is an infinite nested sequence of radicals.

$$\sqrt{6+\sqrt[3]{-7-\sqrt[4]{3-\sqrt{6+\sqrt[3]{-7-\sqrt[4]{3-\cdots}}}}}}$$

To evaluate it, we can turn it into a recurrence relation:

$$x_0 = 0$$

$$x_{n+1} = \sqrt{6+\left(-7-(3-x_n)^{1/4}\right)^{1/3}}$$

The sum key can be used as a loop control to iterate the recurrence. Each term of the "sum" can update x_n and keep the value in a register. At the end, we ignore the sum of the terms and just look at the final x_n.

The main problem is in computing the cube roots, since the arguments are negative. For example, computing $(-8)^{1/3}$ by using the $\mathrm{y^x}$ key to directly do a 1/3 power,

−8, enter, 1, enter, 3, /, $\mathrm{y^x}$

doesn't work. We would want the cube root of −8 to be −2 for this calculation, but when the exponent is not an integer, the calculator evaluates it as

$$y^x = e^{x\ \ln(y)}$$

But $y = -8$ means $\ln(y)$ is not a real number. Moving to the complex function screen doesn't solve the problem, since in complex mode we get the principal cube root of −8 in the first quadrant.

$$(-8)^{1/3} = 1 + \sqrt{3}\, i$$

A solution that works for this recurrence is to see that whenever a cube root is computed, the argument is always negative, so we can use $a^{1/3} = -\left(-a\right)^{1/3}$.

f3: 1, func, chs, 1, enter, 3, /, $\mathrm{y^x}$, chs

Now define f1 to do one iteration of the recurrence. The previous x_n will be in register 1. f1 will compute x_{n+1} and put that back into register 1.

f1: 1, func, 1, rcl, chs, 3, +, $\sqrt{\mathrm{x}}$, $\sqrt{\mathrm{x}}$, chs, 7, −, 3, $\mathrm{f_n}$, 6, +, $\sqrt{\mathrm{x}}$, 1, sto

To apply the recurrence 10 times, first initialize $x_0 = 0$ in register 1, use the sum key, then recall x_n.

0, enter, 1, sto,   30, fix, 7, func, 1, enter, 10, enter, 1, enter, 1, sum, 1, rcl

This gives 1.99999999999999999999999981684391, so it seems the nested radical expression equals 2.

The f3 function above for the cube root assumed its input was negative. How could we generalize the cube root to work for any input? An even better solution would be a function that computes $f(x,p,q) = x^{p/q}$ for any real x, where p and q could be any integers.

Calc-50 is not designed for complicated programming. It doesn't have explicit loops or if statements, but as we did with the sum function to get a loop, we can sometimes use the select function like an if statement.

Define f3 to compute $x^{p/q}$ when p and q are integers.
It will give $x^{p/q}$ directly if $x \geq 0$.
It will give unknown if $x < 0$ and q is even (in lowest terms of p/q).
It will give $-(-x)^{p/q}$ if $x < 0$ and q is odd.
Use the select (sel) function, with f3 storing x in register 1, p in register 2, q in register 3.

f3: 1, func, 3, sto, roll, 2, sto, roll, 1, sto, 0, enter, e99, e^{x}, 7, func, 4, sel

f4: 1, func, 1, rcl, 2, rcl, 3, rcl, /, y^{x}

f5: 2, func, 2, rcl, 3, rcl, gcd, 4, sto, 2, rcl, 4, rcl, /, 2, sto, 3, rcl, 4, rcl, /, 3, sto, 6, f_{n}

f6: 2, func, 3, rcl, 2, mod, 7, func, 0, enter, 0, enter, 7, sel

f7: 1, func, -1, $\sqrt{\mathrm{x}}$

f8: 1, func, 1, rcl, chs, 2, rcl, 3, rcl, /, y^{x}, chs

f3 stores the input arguments x, p, q in registers 1,2,3, then calls f4 if $x \geq 0$ or f5 if $x < 0$.
"e99, e^{x}" computes +overflow as the upper limit for the interval where f4 is called.

f4 computes $x^{p/q}$, used when $x \geq 0$

f5 reduces p/q to lowest terms by factoring out their gcd, then calls f6

f6 calls f7 if q is even, or f8 if q is odd

f7 returns unknown (by computing $\sqrt{-1}$)

f8 computes $-(-x)^{p/q}$

Test f3 for several different cases:

-8, enter, 1, enter, 3, enter, 3, f_{n}	$(-8)^{1/3} = -2.00000000000000000000000000000000$
8, enter, 1, enter, 3, enter, 3, f_{n}	$(+8)^{1/3} = 2.00000000000000000000000000000000$
-8, enter, 2, enter, 6, enter, 3, f_{n}	$(-8)^{2/6} = -2.00000000000000000000000000000000$
-8, enter, 1, enter, 2, enter, 3, f_{n}	$(-8)^{1/2} =$ unknown
8, enter, 1, enter, 2, enter, 3, f_{n}	$(+8)^{1/2} = 2.82842712474619009760337744841939$