Computer Excercise 1.1.21(a)

The following code will analyze the extent to how floating point values are rounded by comparing different display formats.

```
% part a

x=0.1;
y=0.01;
z = x-y;
p = 1.0/3.0;
q = 3.0*p;
u = 7.6;
v = 2.9;
w = u - v;

%display using default format
fprintf('x = %f, y = %f, z = %f, p = %f, q = %f, u = %f, v = %f, w = %f \n',x,y,z,p,q,u,v,w)
```

```
x = 0.100000, y = 0.010000, z = 0.090000, p = 0.333333, q = 1.000000, u = 7.600000, v = 2.900000, w = 4.700000
```

These values appear to be reasonable and consistent with the intended values assigned.

```

%display using extremely large format field
fprintf(['large format field:\n x = %30.20f,\n y = %30.20f,\n z = %30.20f,\n p = %30.20f,\n '
    'q = %30.20f,\n u = %30.20f, \n v = %30.20f,\n w = %30.20f \n'],x,y,z,p,q,u,v,w)
```

```
large format field:
 x =         0.10000000000000000555,
 y =         0.01000000000000000021,
 z =         0.09000000000000001055,
 p =         0.33333333333333331483,
 q =         1.00000000000000000000,
 u =         7.59999999999999964473,
 v =         2.89999999999999991118,
 w =         4.69999999999999928946
```

Well, well well.. the true colors are revealed. Unveiling more decimal places indicates that a lot more is going behind the scenes than what the first set of displayed values would suggest. This demonstrates that we should be very carefuly when working floating point arithmetic, ESPECIALLY if we want lots of precision. For example, something whacky is going on with the last five digits of 'w'; it was supposed to be just a simple subtraction of $7.6 - 2.9 = 4.7$. In fact, something crazy is going on with all of the variables except 'q' (but maybe if we displayed more digits, 'q' might also reveal its true colors).

Computer Excercise 1.1.21(b)

The following code uses different functions that map $n \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \subseteq \mathbb{N}$ onto the real numbers and stores them in x, y, and z.

```
%part b
for n = 1:10
    x = (n-1)/2;
    y = (n^2)/3.0;
    z = 1.0 + 1/n;
    fprintf('n = %d: x = %f, y = %f, z = %f \n\n',n,x,y,z)
end
```

```
n = 1: x = 0.000000, y = 0.333333, z = 2.000000

n = 2: x = 0.500000, y = 1.333333, z = 1.500000

n = 3: x = 1.000000, y = 3.000000, z = 1.333333

n = 4: x = 1.500000, y = 5.333333, z = 1.250000

n = 5: x = 2.000000, y = 8.333333, z = 1.200000

n = 6: x = 2.500000, y = 12.000000, z = 1.166667

n = 7: x = 3.000000, y = 16.333333, z = 1.142857

n = 8: x = 3.500000, y = 21.333333, z = 1.125000

n = 9: x = 4.000000, y = 27.000000, z = 1.111111

n = 10: x = 4.500000, y = 33.333333, z = 1.100000
```

Analysis of part (b)

Let's try to reduce the amount of floating point digits in the function expressions to see if we get the same results:

```
for n = 1:10
    x = (n-1)/2;
    y = (n^2)/3; %changed 3.0 to 3
    z = 1 + 1/n; %changed 1.0 to 1
    fprintf('n = %d: x = %f, y = %f, z = %f \n\n',n,x,y,z)
end
```

```
n = 1: x = 0.000000, y = 0.333333, z = 2.000000

n = 2: x = 0.500000, y = 1.333333, z = 1.500000

n = 3: x = 1.000000, y = 3.000000, z = 1.333333

n = 4: x = 1.500000, y = 5.333333, z = 1.250000

n = 5: x = 2.000000, y = 8.333333, z = 1.200000

n = 6: x = 2.500000, y = 12.000000, z = 1.166667

n = 7: x = 3.000000, y = 16.333333, z = 1.142857

n = 8: x = 3.500000, y = 21.333333, z = 1.125000

n = 9: x = 4.000000, y = 27.000000, z = 1.111111

n = 10: x = 4.500000, y = 33.333333, z = 1.100000
```

The values displayed here are consistent with the ones above, so at least to the extent of the digits displayed, the values we acquired for these expressions are reliable.