```
      AREA=0.
      X = 1.
      DELTX=0.1
 9    Y=X**2+2.*X+3.
      X=X+DELTX
      YPLUS=X**2+2.*X+3.
 10   AREA=AREA+(YPLUS+Y)/2.*DELTX
      IF(X-10.)9,15,15
 15   WRITE(2,7)AREA
 7    FORMAT(E20.8)
      STOP
      END
```

This should evaluate the function for X=1.0, 1.1, until X is 10.0, should it not? But try it, and you will discover that on many machines it in fact does an extra evaluation, the last one at X equal to 10.09999.... The reason is simple: "0.1" is not an exact fraction in a binary machine (in much the same way that 1/3 is not an exact fraction in a decimal world); its nearest representation in most machines happens to be slightly less than 0.1. Thus 10 times "0.1" is not 1.0000..., but is 0.9999..., and by extension, when "0.1" is added to 1.0 ninety times, the result is not 10.000..., but 9.999.... The test that terminates the loop is sensitive to the difference, and gives us an extra trip around.

The value of the integral is too high by over two percent for this function and range. This error could have been readily caught, since the function can be integrated by hand. The discrepancy might then have led to further analysis of the program. (People who worry about computing efficiency might also notice that the function is evaluated twice as often as it need be. Since the answers are wrong, however, this seems unimportant.)

The moral? Floating point numbers should never be used for counting. If you want intervals of 0.1, do it this way:

```
      AREA = 0.0
      X = 1.0
      Y = X**2 + 2.0*X + 3.0
      DELTX = 0.1
C                         STEPS OF 0.1 FROM 1.1 TO 10.0
      DO 10 I = 11,100
         X = FLOAT(I)/10.0
         YPLUS = X**2 + 2.0*X + 3.0
         AREA = AREA + DELTX*(YPLUS+Y)/2.0
         Y = YPLUS
   10 CONTINUE
      WRITE(2,20) AREA
   20 FORMAT(1PE20.8)
      STOP
      END
```

This also eliminates the redundant function evaluations.

---

*10.0 times 0.1 is hardly ever 1.0.*

---