PES, Section 2.8
Rounding and Overflow

1. The following program was used in Section 2.8 of PES to illustrate the problem of rounding, which occurs in Fahrenheit-to-Celsius conversion:

```
#include "RIMS.h"
unsigned char C2F_uc(unsigned char C) {
   unsigned char F;
   F = (9/5)*C + 32;
   return F;
}
void main() {
   while (1) {
      B = C2F_uc(A);
   }
}
```

The following Table shows the actual Fahrenheit values (real numbers), the closest integer approximation (i.e., round-up, round-down), the results computed by the above program, and a variation that performs division later: F = (9*C)/5 + 32.

Celsius	Fahrenheit (actual)	Fahrenheit (integer)	Fahrenheit (from above program)	Fahrenheit (late division)
0	32	32	32	32
1	33.8	34	33	33
2	35.6	36	34	35
3	37.4	37	35	37
4	39.2	39	36	39
5	41	41	37	41
6	42.8	43	38	42
7	44.8	45	39	44
8	46.4	46	40	46
9	48.2	48	41	48

Rewrite the function C2F_uc to compute the values in the table listed as Fahrenheit (integer).

```
unsigned char C2F_uc(unsigned char C) {
   unsigned char F;
   F = (9*C)/5 + 32;
   if( (9*C)%5 >= 3)
      F++;
   return F;
}
```

2. In the preceding example, we replaced F = (9/5)*C + 32 with F= (9*C)/5 + 32. Making this change could introduce a new type of error that would not have occurred originally. What is the error?

9*C could overflow, depending on the value of C.

(9/5)*C could not overflow, because (9/5) simplifies to 1, and 1*C does not overflow (presuming that C is an unsigned char to begin with).

3. When computing (a + b + c)/3, the possibility of integer overflow occurs. One proposed remedy was to compute the division earlier, i.e., (a/3) + (b/3) + (c/3). Give an example, assuming that overflow does not occur, where (a + b + c)/3 computes a more accurate result than (a/3) + (b/3) + (c/3).

a = b = c = 1

(a + b + c)/3 = (1 + 1 + 1)/3 = 3/3 = 1

(a/3) + (b/3) + (c/3) = (1/3) + (1/3) + (1/3) = 0 + 0 + 0 = 0