Tenets of Precision and Uncertainty

1. Integers are assumed to be exact representations
2. The precision indicates the number of significant decimal digits in the representation of the value. We specify precision of a double as the power of 10 of the least significant digit.
3. Doubles with an uncertainty of 0. are assumed to be exact representations if the value is an integer between –2^53 and +2^53 (9007199254740992).
4. A 15-digit decimal number converted to a double and back is guaranteed to result in the same value. Most 16-digit decimal numbers and some 17-digit decimal numbers will obey this rule. Similarly, a double converted to a 17-digit number and converted back to a double will always result in the same value. (This is not guaranteed to be true if fewer than 17 digits are used.)
5. The maximum relative error in representing a value via double format is 2^-53, which is 1.1102230246251567e-16. This means the actual value can be greater or less than the represented value by 2^-53 \* value. Thus, the minimum uncertainty of an inexact number represented via a double is 2^-53 \* value.

Ran a series of tests as follows:

double x13 = 1. + 1.e-12;

double x14 = 1. + 1.e-13;

double x15 = 1. + 1.e-14;

double x16 = 1. + 1.e-15;

double x17 = 1. + 1.e-16;

printf("Output 17 digits after the decimal point.\n");

printf(" 1.12345678901234567\n");

printf("13 significant digits: %19.17e\n", x13);

printf("14 significant digits: %19.17e\n", x14);

printf("15 significant digits: %19.17e\n", x15);

printf("16 significant digits: %19.17e\n", x16);

printf("17 significant digits: %19.17e\n", x17);

printf("Output 16 digits after the decimal point.\n");

printf(" 1.12345678901234567\n");

printf("13 significant digits: %18.16e\n", x13);

printf("14 significant digits: %18.16e\n", x14);

printf("15 significant digits: %18.16e\n", x15);

printf("16 significant digits: %18.16e\n", x16);

printf("17 significant digits: %18.16e\n", x17);

printf("Output 15 digits after the decimal point.\n");

printf(" 1.12345678901234567\n");

printf("13 significant digits: %17.15e\n", x13);

printf("14 significant digits: %17.15e\n", x14);

printf("15 significant digits: %17.15e\n", x15);

printf("16 significant digits: %17.15e\n", x16);

printf("17 significant digits: %17.15e\n", x17);

printf("Output 14 digits after the decimal point.\n");

printf(" 1.12345678901234567\n");

printf("13 significant digits: %16.14e\n", x13);

printf("14 significant digits: %16.14e\n", x14);

printf("15 significant digits: %16.14e\n", x15);

printf("16 significant digits: %16.14e\n", x16);

printf("17 significant digits: %16.14e\n", x17);

The following output is generated:

Output 17 digits after the decimal point.

1.12345678901234567

13 significant digits: 1.00000000000100009e+00

14 significant digits: 1.00000000000009992e+00

15 significant digits: 1.00000000000000999e+00

16 significant digits: 1.00000000000000111e+00

17 significant digits: 1.00000000000000000e+00

Output 16 digits after the decimal point.

1.12345678901234567

13 significant digits: 1.0000000000010001e+00

14 significant digits: 1.0000000000000999e+00

15 significant digits: 1.0000000000000100e+00

16 significant digits: 1.0000000000000011e+00

17 significant digits: 1.0000000000000000e+00

Output 15 digits after the decimal point.

1.12345678901234567

13 significant digits: 1.000000000001000e+00

14 significant digits: 1.000000000000100e+00

15 significant digits: 1.000000000000010e+00

16 significant digits: 1.000000000000001e+00

17 significant digits: 1.000000000000000e+00

Output 14 digits after the decimal point.

1.12345678901234567

13 significant digits: 1.00000000000100e+00

14 significant digits: 1.00000000000010e+00

15 significant digits: 1.00000000000001e+00

16 significant digits: 1.00000000000000e+00

17 significant digits: 1.00000000000000e+00

We observe the following:

1. The 17th significant digit is ignored (i.e., 1. + 1.e-16 = 1.) Even when x17 is increased to 1. + 8.e-16, the value is still 1.
2. The 18th significant digit is noise.
3. The 17th significant digit is within 1.
4. The 16 and 15 digit representations are correct.

When 1. Is changed to 9., the following output is generated:

Output 17 digits after the decimal point.

1.12345678901234567

13 significant digits: 9.00000000000100009e+00

14 significant digits: 9.00000000000009948e+00

15 significant digits: 9.00000000000001066e+00

16 significant digits: 9.00000000000000178e+00

17 significant digits: 9.00000000000000000e+00

Output 16 digits after the decimal point.

1.12345678901234567

13 significant digits: 9.0000000000010001e+00

14 significant digits: 9.0000000000000995e+00

15 significant digits: 9.0000000000000107e+00

16 significant digits: 9.0000000000000018e+00

17 significant digits: 9.0000000000000000e+00

Output 15 digits after the decimal point.

1.12345678901234567

13 significant digits: 9.000000000001000e+00

14 significant digits: 9.000000000000099e+00

15 significant digits: 9.000000000000011e+00

16 significant digits: 9.000000000000002e+00

17 significant digits: 9.000000000000000e+00

Output 14 digits after the decimal point.

1.12345678901234567

13 significant digits: 9.00000000000100e+00

14 significant digits: 9.00000000000010e+00

15 significant digits: 9.00000000000001e+00

16 significant digits: 9.00000000000000e+00

17 significant digits: 9.00000000000000e+00

We observe the following:

1. The 15 digit representation is still correct.
2. The 16 digit representation is off by up to 2 in the 16th digit.
3. The 17th digit has become noise. (This is likely the reason the error in the 1. + 1.e-15 case is 2 rather than 1. The 17 digit representation shows 18 as the last 2 digits. The 8 causes the 16 digit representation to round to 2 instead of 1.)

When 1. Is changed to a value from 2 through 17, the 16 digit representation is correct. For 8 and 9, there are errors up to 2 in the 16th digit.

When 1.e-i is changed to 2.e-i, all 9 are correct to 16 digits.

When every tenth is run from 1.0 to 9.9, the values from 1.0 to 4.0 have no errors, from 4.1 to 7.9 have errors up to 1 in the 16th digit, and from 8.0 to 9.9 have errors up to 2 in the 16th digit.

The above is sensible, given that the minimum 4.0 \* minimum fractional uncertainty is 4.44e-16 and 4.1 \* minimum fractional uncertainty is 4.55e-16. Thus, 4.1 is the smallest value we tried where the uncertainty rounds to 5e-16, so could product a value with a 5 in the 17th digit. This is enough to affect the 16th digit by 1 when rounded to 16 digits for output.

To attempt to confirm this, compute the smallest value that results in an uncertainty of 4.5e-16, by dividing 4.5e-16 by minimum fractional uncertainty. The result is 4.0082. I ran the above from 4.0 to 4.09 by 0.01, and there were errors at 4.02 and 4.06. I then ran the above from 4.000 to 4.009 by 0.001, and there were errors at 4.004 and 4.007.

Further experimentation indicated that an error in the 16th digit could be produced with values < 3, e.g., 2.9991 + 1.e-14. As such, adopt the following approach

**Use 16 digits of decimal precision for all doubles. This is achieved by multiplying the value by 1.e-15, then determining the next lower power of 10, unless the result of the multiplication is exactly a power of 10.**

**Set the minimum fractional uncertainty to 2.e-16. This value allows an error of 5.e-16 beginning with a value of 2.5, which would round to an error of 1 in the 16th digit. At a value of 7.5, the error is 1.5e-16, which rounds to an error of 2 in the 16th digit.**