Representation of real numbers

For this first section, use our signed six decimal-digit floating-point representation.

1. What are the following numbers in our representation?

$$9.6249193 \quad 16.2549 \quad -132.5499 \quad 1.6945 \times 10^{-19}$$

Answer: `+499625 +501625 -511325 +301694`

2. What real numbers do the following numbers represent?

`+488713 -527326 +536317 -834800`

Answer: 0.8713, –7326, 63170 or preferably 6.317×10^4, -4.800×10^{34}

3. Why is 6.317×10^4 preferable to 63170 when describing what is represented by `+536317`?

The latter potentially suggests five significant digits, meaning that the representation is representing all numbers on the range [63169.5, 63170.5], but in fact represents numbers on the much larger range (63165, 63175), recalling that 63165 would be rounded to 6.316×10^4 and 63175 would be rounded to 6.318×10^4.

4. What is the philosophy for not having a division-by-zero automatically return an error. For example, the following will result in the program terminating in C++:

```
#include <iostream>
int main();
int main() {
      int n;   // To be assigned by the user
      std::cout << "Enter an integer: "; // Assume the user enters 0
      std::cin  >> n;
      std::cout << (1/n) << std::endl;

      return 0;
}
```

Answer: +0.000 does not represent a true zero, but rather, it represents all numbers smaller than the smallest number that can be represented as a floating-point number; that is, all numbers less than or equal to 0.0005×10^{-49}. Thus, 1.0 divided by a very small number is actually a very large number, so it is better to represent this as infinity.

5. Why do we require that the most significant digit of the significand is, in general, not equal to zero?

Answer: This ensures that the same number does not have more than one representation. For example, the following could all be used to represent 1.0: `+491000 +500100 +510010 +520001`

6. What are the benefits of a floating-point representation of real numbers given a fixed number of digits that may be stored?

Answer:

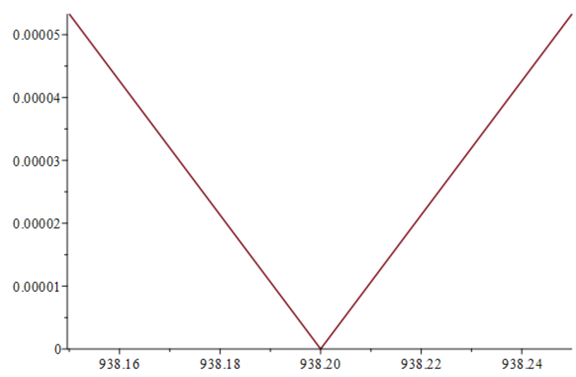
a. It can represent a vast range of real numbers.
b. It represents all those numbers to approximately the same relative error.

7. What numbers does `+519382` represent, and what is the maximum percent relative error of this representation?

Answer: This number is 9.382×10^2 or 938.2 and thus represents all numbers on the range [938.15, 938.25], because each of these end-points, rounded to four significant digits is 938.2. The relative error of each of the end-points is $\frac{0.05}{938.15} \approx 0.00005330$, or 0.005330% relative error or $\frac{0.05}{938.25} \approx 0.00005329$, or 0.005329%. The first is larger, and because the absolute error of any other number in the interval is less than 0.05, any other number would have a smaller relative error.

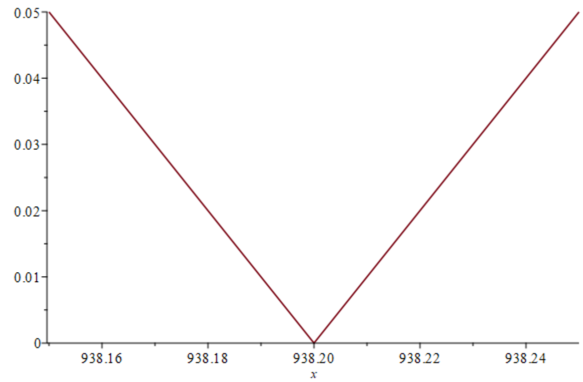
8. What is the plot of the relative error of the representation `+519382` for numbers on the range [938.15, 938.25]? Is this a piecewise linear function?

Answer: The plot is



showing a minimum relative error of zero for $x = 938.20000$, and then an apparent linear increase outward; however, these are not straight lines, for the actual function is $\frac{|x-938.2|}{|x|}$, and so therefore the denominator will be ever so slightly changing on the interval in question.

The plot of the absolute error $|x-938.2|$ would be piecewise linear:



9. Add the numbers represented by +499625 and +488713 return the result to this representation.

Answer: +501050

10. Add the numbers represented by +974913 and +977812 return the result to this representation.

Answer: +981272

11. Add the numbers represented by +134913 and +137822 return the result to this representation.

Answer: +141274

12. In which order would you add these three numbers to get the best approximation of the actual answer?

+253253, +297931 and +254135

Answer: Adding either of the two smaller numbers to the larger number results in the larger number unchanged, but adding the two smaller numbers together first, and then adding that to the larger number produces +297932. This latter representation is closer to the exact answer.

13. Which sum is likely to be closer to the actual sum? Recall that $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$.

```
#include <iostream>
#include <cmath>
int main();
int main() {
    double sum_frwd{ 0.0 };
    double sum_bkwd{ 0.0 };

    for ( unsigned int k{1}; k <= 25000000; ++k ) {
        sum_frwd += 1.0/std::pow( k, 2 );
    }

    for ( unsigned int k{25000000}; k > 0; --k ) {
        sum_bkwd += 1.0/std::pow( k, 2 );
    }

    std::cout.precision( 16 );
    std::cout << sum_frwd << std::endl;
    std::cout << sum_bkwd << std::endl;
    // The finite sum printed to 20 significant digits
    std::cout << "1.6449340568482264865" << std::endl;
    std::cout << (M_PI*M_PI/6.0) << std::endl;

    return 0;
}
```

Answer: Try this yourself but consider first which should give a better approximation.

14. What is the reciprocal of the reciprocals of each of the following numbers if the reciprocal is first stored as a floating-point number? Should the reciprocal of the reciprocal of x equal x?

`+495457 +497125 +498414 +499574`

Answer: `+495456 +497123 +498418 +499585`

For real numbers, $1/(1/x) = x$, but this is not true when performing floating-point calculations.

15. We have that sin(`+491000`) equals `+488415` and sin(`+491001`) equals `+488420`. Is it therefore fair to say that sin(1.001) – sin(1) equals `+455000` as the second number minus the first equals 5.000×10^{-4}? What is the percent relative error of this approximation?

Answer: No, as the result is a consequence of subtractive cancellation, so we have actually lost three significant digits. The actual answer, to four significant digits, is 0.0005399, so the precent relative error is 7.387%, which is orders of magnitude larger than what the answer should be if we were able to successfully calculate 0.0005399.

16. Note that $\sin(1 + h) = \sin(1) + \cos(1)h - ½\ \sin(\xi)h^2$ according to a first-order Taylor's series, so $\sin(1 + h) - \sin(1) \approx \cos(1)\ h$. What is this value if we were to calculate everything using our floating-point representation?

Answer: cos(`+491000`) = `+485403` and this multiplied by `+461000` yields `+455403` which is much closer to the ideal answer of 0.0005399 or `+455399`, which has a percent relative error of only 0.07752%. Thus, the Taylor series formula provides a much better approximation of the value than the actual calculation.

17. Multiply the numbers represented by `+502180` and `-484750`; `-501480` and `-488625`; and return these results to this representation.

Answer: `-501036` and `+501276`

18. Multiply the numbers represented by `+974913` and `+977812` and return the result to this representation.

Answer: `+990000` (this is infinity)

19. Multiply the numbers represented by `+234913` and `-237822` and return the result to this representation.

Answer: `-000038` (this is a denormalized number, as -3.843×10^{-51} cannot be represented using our format as a normalized number, so we must represent -0.038×10^{-49}.

20. Sort these numbers based on the values that they represent:

`+860534 -816415 +170465 -484459 -056869 +684442 -285840`

Answer:: `-816415 -484459 -285840 -056869 +170465 +684442 +860534`

21. Did you need to convert, for example, `-484459` to –0.4459 when you sorted the numbers in Question 20?

Answer: Hopefully no…

For this first section, use the binary double-precision floating-point representation.

22. Add the following two double-precision floating-point numbers and write the result in that format:

```
0 10010110011 0010110000000000000000000000000000000000000000000000
```

```
0 10010110101 1101110000000000000000000000000000000000000000000000
```

Answer:

```
0 10010110110 0001001110000000000000000000000000000000000000000000
```

23. Add the following two double-precision floating-point numbers and write the result in that format:

```
0 01111111111 0101010101010101010101010101010101010101010101010101
0 10000000000 0101010101010101010101010101010101010101010101010101
```

Answer:

```
0 10000000001 0000000000000000000000000000000000000000000000000000
```

24. Add the following two double-precision floating-point numbers and write the result in that format:

```
0 11110100101 0100000100010000010000001100000111000100100000000101
0 00010110100 1001100000000100001100000010100100000100011000000010
```

Answer:

```
0 11110100101 0100000100010000010000001100000111000100100000000101
```

25. Calculate the reciprocal of this number and write the result in that format:

```
0 10000000010 0000000000000000000000000000000000000000000000000000
```

Answer:

```
0 01111111100 0000000000000000000000000000000000000000000000000000
```

26. Multiply the following two double-precision floating-point numbers and write the result in that format:

```
0 10000000010 0010100000000000000000000000000000000000000000000000
0 01111111101 1011000000000000000000000000000000000000000000000000
```

Answer:

```
0 10000000000 1111001110000000000000000000000000000000000000000000
```

27. Sort the following seven double-precision floating-point numbers:

```
0 11101100110 0110000100000000111111111010100001111010001111110010
1 10110001011 1110111101000011001101001000100000101101010010001101
0 11101100110 0110000111101111100101100110001110101100001110111110
1 01100101110 0010011010111100001111001111000000001101011011101100
0 11011010110 1111001011100001010110100111100001100001010110101000
1 10110101011 1011000111011011001001011110001110000110011011011110
0 10111001111 0001101001101000101010000110111001101011101000110000
```

Answer:

```
1 10110101011 1011000111011011001001011110001110000110011011011110
1 10110001011 1110111101000011001101001000100000101101010010001101
1 01100101110 0010011010111100001111001111000000001101011011101100
0 10111001111 0001101001101000101010000110111001101011101000110000
0 11011010110 1111001011100001010110100111100001100001010110101000
0 11101100110 0110000100000000111111111010100001111010001111110010
0 11101100110 0110000111101111100101100110001110101100001110111110
```

Acknowledgement: Chinemerem Chigbo found an error in Question 19 and Pavan Jayasinha found an error in Question 11.