

Exercise 1

Let us denote the basis in which we are representing the numbers by a subscript. For example $(3)_{10}$ means that 3 is written in the decimal basis.

Let us first consider number of the type $(1.X)_{10}$. In the binary basis these numbers are of the type $(1.Y)_2$, i.e. only one bit is used for the integer part, which is always 1.

In the IEEE-754 standard representation one has 23 bits available for the mantissa when using single precision and 51 when using double precision.

Since the exponent in this representation for these numbers is fixed, one has that all the possible combination of numbers within 1 and 2 are 2^{23} for the single precision case and 2^{51} for the double precision case.

On the other side numbers of the type $(255.X)_{10}$ have a binary representation of the type $(1111111.Y)_2$, which becomes $(1.111111Y \times 2^Z)_2$ when written in the IEEE-754 format. The operation of shifting the point put constraints on the first 7 digits of the mantissa, that is all the numbers of the type $(255.X)_{10}$ have the same first 7 digits in the mantissa when represented in the IEEE-754 standard. Since the number of digits available to represent the mantissa does not change we are now left with only $23 - 7 = 16$ usable digits for the single precision and $51 - 7 = 44$ for the double precision. This means that in the given interval we can represent only 2^{16} numbers in the single precision format and 2^{44} in the double precision format.

Exercise 2

1

This is an automatic cast problem. In the case of y , when performing the division $i/2$, we are dividing two integers. By convention the type of the result is considered int itself, so that the decimal part is truncated.

When the result of the multiplication by 2 is stored into y the result is casted to double, but the decimal part was lost before.

In the case of z , when performing the division $i/2$, we are dividing a integer by a floating point number, and this conventionally is stored in a floating point variable. When storing the result of the multiplication by 2 the number is casted to double, which increase the precision for successive operation but has no influence of what happened before i.e. at that point the number still have single precision.

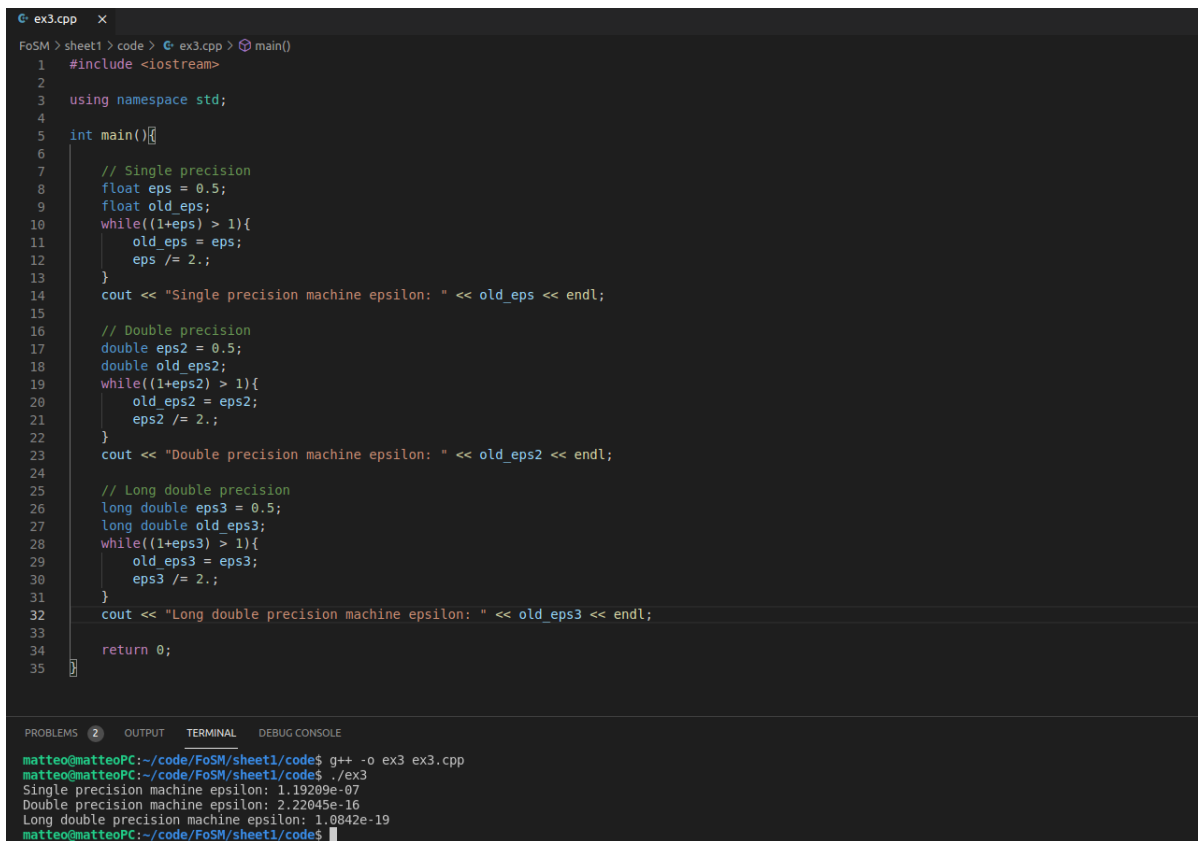
2

The correct one is x . In the case of y , when performing $b + c$, we are small summing a number (c) to very big number (b) and in particular the value of c is so small compared to b that the result of the sum is still b , i.e. we don't have enough resolution to store the variation caused by c .

3

The number $x * x$ exceeds the maximum representable number in the floating point format which is $2^{128} \simeq 3.4 \times 10^{38}$. Hence an overflow happens and the number is stored as $+inf$.

Exercise 3



The image shows a code editor window with a file named `ex3.cpp`. The code is a C++ program that calculates the machine epsilon for single, double, and long double precision. It uses a while loop to repeatedly add the current epsilon to 1 until the result is no longer greater than 1, then divides by 2. The results are printed to the console.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6
7     // Single precision
8     float eps = 0.5;
9     float old_eps;
10    while((1+eps) > 1){
11        old_eps = eps;
12        eps /= 2.;
13    }
14    cout << "Single precision machine epsilon: " << old_eps << endl;
15
16    // Double precision
17    double eps2 = 0.5;
18    double old_eps2;
19    while((1+eps2) > 1){
20        old_eps2 = eps2;
21        eps2 /= 2.;
22    }
23    cout << "Double precision machine epsilon: " << old_eps2 << endl;
24
25    // Long double precision
26    long double eps3 = 0.5;
27    long double old_eps3;
28    while((1+eps3) > 1){
29        old_eps3 = eps3;
30        eps3 /= 2.;
31    }
32    cout << "Long double precision machine epsilon: " << old_eps3 << endl;
33
34    return 0;
35 }
```

The terminal output shows the execution of the program:

```
matteo@matteoPC:~/code/FoSM/sheet1/code$ g++ -o ex3 ex3.cpp
matteo@matteoPC:~/code/FoSM/sheet1/code$ ./ex3
Single precision machine epsilon: 1.19209e-07
Double precision machine epsilon: 2.22045e-16
Long double precision machine epsilon: 1.0842e-19
matteo@matteoPC:~/code/FoSM/sheet1/code$
```

Figure 1: Program that evaluates machine ϵ for single, double and long double precision