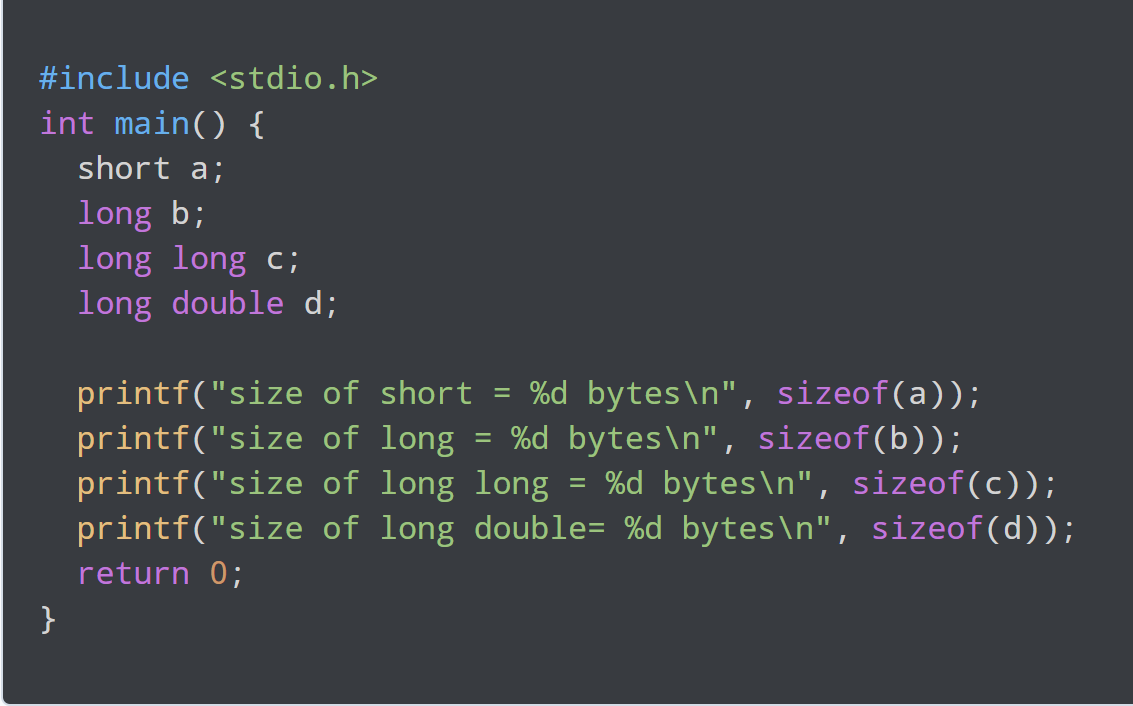
Lab

# Datatypes in C

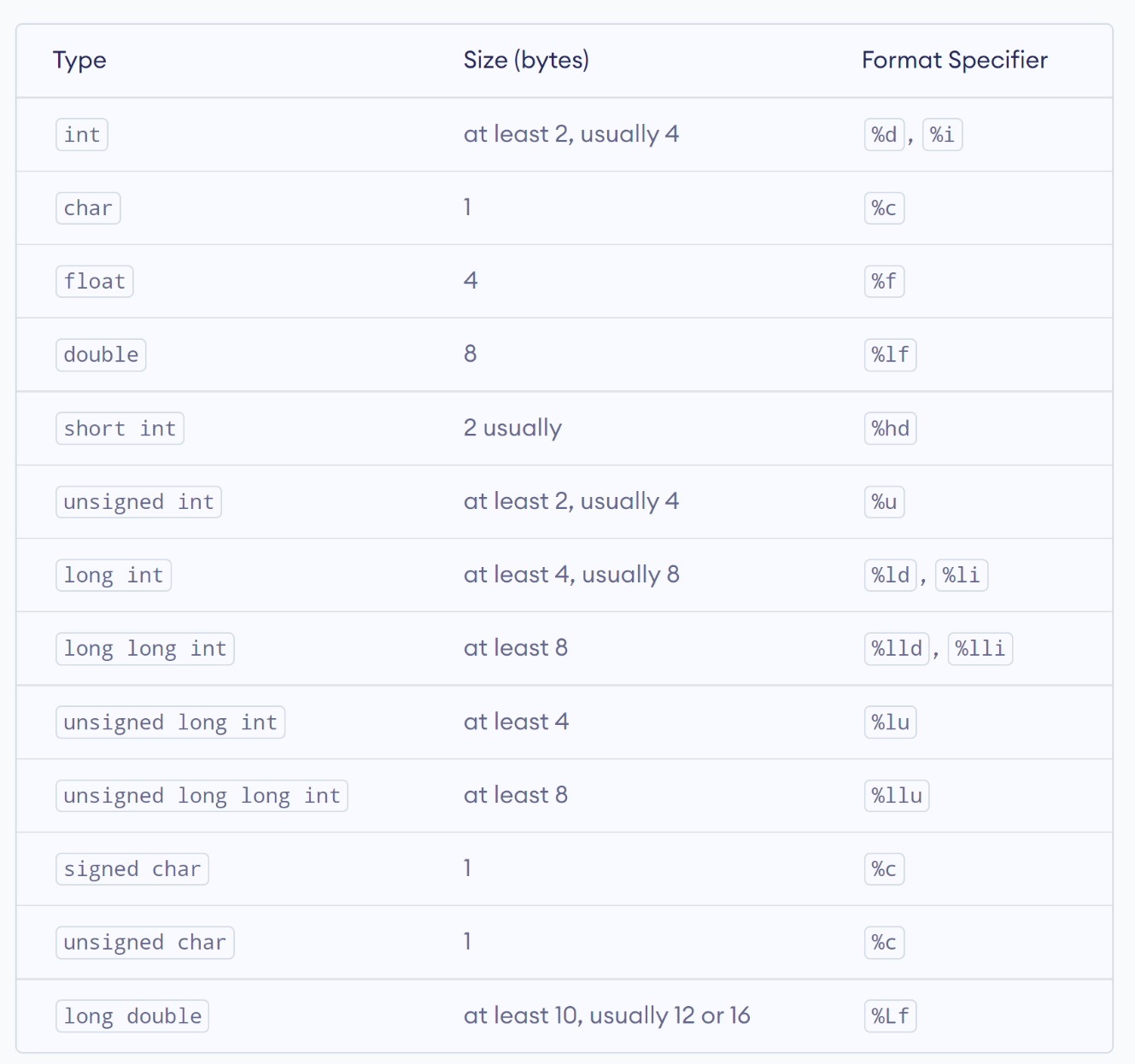
Please type out the code in this program to see the size of different data types on your computer.



Exercise

1. Use a similar printf statement to display the size of a float.

# Datatypes Table



# Floating Point Numbers

Floating point numbers have three components:

* Sign bit – whether the number is positive or negative
* Exponent bits – the magnitude of the number
* Mantissa bits – the fractional bits

32 bit floats use **1 bit** for **sign**, **8 bits** for **exponent** and **23 bits** for **mantissa**. Whatever number is encoded in the exponent bits, you subtract 127 to get the actual exponent, meaning the exponent can be from -126 to +127.

Note the following: which means that 256 numbers can be represented by the 8 bits. Floats are signed by default which means that half the numbers are positive and the other half are negative. But half of 256 is 128. However, from 1 to 128 there are 129 numbers therefore to get 128 numbers the positive numbers stop at 127. The negative numbers cannot stop at -128 or 127 either. Can you see why? Well, from -1 to -127 there are 128 numbers but we also need to cater for 0. Hence, the range of negative numbers starts from -126.

64 bit doubles use **1 bit** for **sign**, **11 bits** for **exponent** and **52 bits** for **mantissa**. Whatever number is encoded in the exponent bits, you subtract 1023 to get the actual exponent, meaning the exponent can be from -1022 to +1023.

# Calculating the precision for a specified number.

The choice of data type limits the precision (number of decimal points) you can expect in calculation. Suppose that you need to estimate the precision when the values you are going to deal with range from 10 to 1000. The least precision will be experienced at the largest number which is 1000 in this case. 1000 is between and . To calculate the highest precision for 1000 as a float, do the following:

1. Calculate the value of 2 raised to the number of mantissa bits i.e.
2. Get the difference between and i.e. 1024 – 512 = 512. This is referred to as the *range*.
3. The precision will therefore be
4. We can conclude that:

# When do you start to hit precision issues?

Suppose I have a mathematical problem where I need the precision to be 0.0333. At what value should I expect to start getting precision issues?

Rewriting the equation in the section above gives: .

This gives:

It is around this value that precision issues will arise if you are using the float data type.

# Exercise

Suppose I have a mathematical problem where the precision needed is 0.0025. What is the maximum value that can be represented with the right precision using float data type?

# Round Off Error

Run the program below. Can you explain what is happening?

