What are unsigned integers?

Sometimes you'll see another integer type in C++ code, "unsigned" integer, that cannot represent negative values. Instead, unsigned integers have an increased upper positive value range compared to signed integers of the same memory usage. (Signed integers devote a bit of memory just to be able to represent a negative sign.) Unsigned integers are sometimes used in special cases to make use of memory extremely efficiently; for example, a data storage format might use them in order to be more compact. But mixing unsigned and signed integers in your code can cause unexpected problems. Let's walk through a few examples with unsigned integers.

You can download a source code version of this reading here:

unsigned-ints-example.zip

Unsigned type syntax

The normal "int" syntax creates a signed int by default. But, you could also write "signed int" or "signed" to get the same type:

```
1  int a = 7;
2  // signed int a = 7;
3  // signed a = 7;
```

If you write "unsigned int" or "unsigned", you can create a variable with the unsigned integer type:

```
1  unsigned int b = 8;
2  // unsigned b = 8;
```

Issues with unsigned arithmetic

Unsigned ints can't represent negative values. Instead, they have a higher upper positive limit to the value range that they can represent. This can cause strange behavior if you mix signed and unsigned ints. If you do arithmetic between signed and unsigned ints, then you can get undesired results if negative values should logically have arisen