More about numeric datatypes.

<u>"Closure" properties</u> (under arithmetic operations).

Say $\boxed{?}$ is any of +, −, *, /.

datatype of int $\boxed{?}$ int is <u>always</u> int.

Note: when mixed length integers are combined, result will have same length as max length of the operands. E.g., $\underset{\substack{\uparrow \\ \text{4 bytes}}}{\text{int}}\ \boxed{?}\ \underset{\substack{\uparrow \\ \text{8 bytes}}}{\text{long}}$ is a long

(See also "principle of least surprise")

Similarly, float $\boxed{?}$ float $\longrightarrow$ float.

+ float $\boxed{?}$ double $\longrightarrow$ double.

what about mixing integers w/ floating point?

int $\boxed{?}$ double $\longrightarrow$ double.

Some say result "promoted". (I say "contaminated")

Note: x ^ y is bitwise XOR, NOT exponentiation. (C/C++ don't have this built in)

However, x % y gives the remainder of x/y.

Could be useful for example to check even/oddness of an integer:

$$x \text{ is even} \iff (x \% 2 == 0)$$

Boolean datatype

Holds true/false values.

Note: in C, you just use an integer, under the convention

$$0 \equiv \text{false}$$

$$\text{anything else} \equiv \text{true}$$

C++ also uses this convention!

Typecasting

You can explicitly ask the compiler to treat a variable as if it had some other type.

Syntax: (desired type) expression...

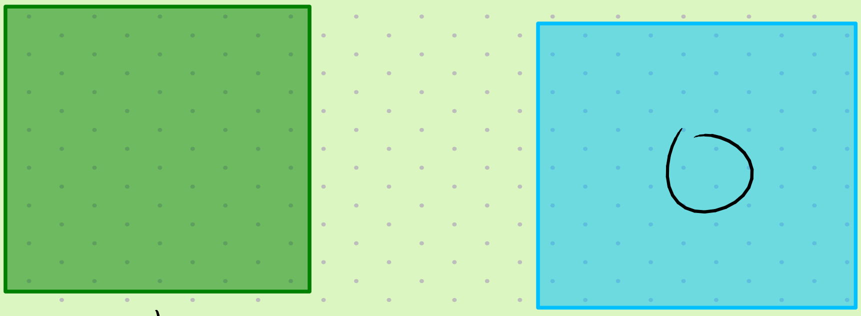
e.g. say x, y of type int, but want to divide & see fractional part:

```
cout << (double) x/y << "\n";
```

(≠ (double)(x/y) !)

Computing a sum...

Strategy: use <u>two</u> variables / post-it notes.



new #

sum of all #s so far

Start blue = 0.

Listen, writing on green.

Add blue + green, writing result back on blue...

Repeat until no more #s...