ARITHMETIC AND THE MATH LIBRARY

Today

- Recap arithmetic
- Arithmetic with mixed variable types
- Math library functions

  This is a bit of a miscellaneous collection of the things we didn't yet cover from Chapters 1 & 2.

Arithmetic

- The mathematical operators in C++ are:

$+$	unary plus
$-$	unary minus
$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
$\%$	modulo

- We also have `++`, `--`, `+=`, `-=`, `*=` and `/=`.

- Given:

```
int x;
int y;
int z;
```

we can write, for example:

```
z = -x;
z = +y;
z = x + y;
z = x - y;
z = x * y;
z = x / y;
z = x % y;
```

- Because `x` and `y` are integers, when we do division it is integer (elementary school) division.
- So:

```
x = 13;
y = 3;
z = x/y;
```

  makes `z` equal to 4.

- Similarly:

```
x = 13;
y = 3;
z = x % y;
```

  makes `z` equal to 1.

- We can write complex arithmetic expressions, like:

```
int x, y, z;
int u, v, w;

x = y + z * u - v / w;
```

- What sum does this do?
- My advice: use parentheses, so if you want:

```
x = (((y + z) * u) - v) / w;
```

  then write that.

- If you don't do what I advise, then C++ uses some (kind of complex) rules to figure out what to do.

- It uses *precedence* rules to decide which things to do first.
- For example, it does `*` and `/` before `+` and `-`
- There are also *associativity* rules, which say how to order things when the precedence rules don't help.
- For example is:

  `x / y * z`

  the same as

  `(x / y) * z`

  or

  `(x / (y * z)`?
- Precedence and associativity rules are in the textbook, page 65.

Arithmetic with mixed variables

- In the arithmetic we have seen before, everything is an integer.
- That's why division was odd (and we needed %) — there was no way to represent fractions.
- If we want to be able to handle fractions, we use `double`-valued variables

```
double x = 13;
double y = 4;
double z

z = x/y;
```

makes `z` equal to 3.25

- If all the variables are `double` then things work as you'd expect.
- You can combine fractions and get fractions as answers.
- Things can be odd if you mix `doubles` and `ints`.

```
double x = 13;
double y = 4;
int z

z = x/y;
```

  makes `z` equal to 3
- This happens because you can't store the fractional bit in `z`, so it just gets truncated.

- Perhaps stranger is:

```
int x = 13;
int y = 4;
double z

z = x/y;
```

  makes `z` equal to 3

- This happens because `x/y` has been evaluated to 4 (as a result of integer division) before it is assigned to `z`.

- However:

```
double x = 13;
int y = 4;
double z

z = x/y;
```

  makes `z` equal to 3.25

- This happens because having one of the variables in the division be a `double` forces the whole division to be done as if all the values were `doubles`.

Math library

- In the roombaGame, let's imagine we want to see how far the roomba is from the dirt.
- We have `x` and `y` which give us the roomba's position.
- We have `dirtX` and `dirtY` which give us the position of the dirt.
- The distance between them is:

$$distance = \sqrt{(x - dirtX)^2 + (y - dirtY)^2}$$

- How can we compute this?

- The squares are easy enough to compute.

  `(x - dirtX) * (x - dirtX)`

  and

  `(y - dirtY) * (y - dirtY)`

- For the square root we can use the *math library function* `sqrt`.

```
distance = sqrt(((x - dirtX) * (x - dirtX))
                + ((y - dirtY) * (y - dirtY)));
```

- To use the math library, we need to add in

  `#include<cmath>`

  at the start of the program.

- The math library contains a bunch of other functions:
- `double pow( double x, double y )`
- `double sin( double x )`
- `double cos( double x )`
- `double tan( double x )`
- `double asin( double x )`
- `double acos( double x )`
- `double atan( double x )`

Summary

- This lecture covered a couple of things related to arithmetic and other mathematics in C++.
- First we recapped the arithmetic operators.
- Then we looked at different aspects of arithmetic, especially what happens when you have complex expressions, and when you mix `int`s and `double`s.
- Finally, we looked at using the math library.