MORE CONTROL STRUCTURES AND SOME MATHEMATICS

Today

- The `for` statement
- 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.

The `for` loop statement

- We use the `for` structure in the ant-game program:

```
for(turns = 8; turns >= 0 ; turns--)
{
       :
  <move the ant>
       :
}
```

- We use it to give us just 8 turns to get the ant to its home.

The `for` loop statement

- General structure:

```
for(<start>; <true or false> ; <change>)
{

    <some instructions>

}
```

- This works as follows

The `for` loop statement

- At the start of the loop, the instruction in `<start>` is carried out.
- We usually use this to set the value of a counter.
- Then `<true or false>` is tested to see if it is true or false.
- This is usually a test on the counter.
- If it is false, the program will skip to the } that marks the end of the control structure.
- If it is true the `<some instructions>` are executed.
- Once they are done, the instruction in `<change>` is executed.
- This is usually something that changes the value of the counter.
- Then `<true or false>` is tested again.
- Thus `<some instructions>` will be repeatedly executed until `<true or false>` becomes false.

Examples

- In the antworld, we might have:

```
int myCount;

for(myCount = 1 ; myCount <= 5 ; myCount++)
{

    goNorth();
    goEast();

}
```

- This would make the ant go 5 steps north and five steps east.

- While:

```
int myCount;

for(myCount = 10 ; myCount > 5 ; myCount--)
{

    goNorth();
    goEast();

}
```

would do the same, but with different values of `myCount`.

- What would

```
int myCount;

for(myCount = 2 ; myCount < 8 ; myCount+=2)
{

    goSount();
    goWest();

}
```

do?

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 integter (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 `double`s and `int`s.

```
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 `double`s.

Math library

- In the ant-game, let's imagine we want to see how far the ant is from home.
- We have `x` and `y` which give us the ant's position.
- We have `homeX` and `homeY` which give us the position of the home.
- The distance between them is:

$$distance = \sqrt{(x - homeX)^2 + (y - homeY)^2}$$

- How can we compute this?

- The squares are easy enough to compute.

  ```
  (x - homeX) * (x - homeX)
  ```

  and

  ```
  (y - homeY) * (y - homeY)
  ```

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

  ```
    distance = sqrt(((x - homeX) * (x - homeX))
                  + ((y - homeY) * (y - homeY)));
  ```

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

  ```
  #include<cmath>
  ```

  at the start of the program.

- See the ant game for an example of this.

- 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 number of slightly unrelated things.
- We looked at `for` loops.
- 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.