Structs

Why Structures?

We've seen lots of examples so far of where we would want to store multiple pieces of information. Up until now we've always used an array, however the main drawback of using an array is that it can only store one **type** of information, like an array of ints or an array of chars. The main advantage of using a struct is that it allows you to use different types of data.

Convention

The convention in C is to create your structs in a seperate header `.h` file, but for the time being we'll create them in our main.c files. In this lesson we'll create a basic car structure, make a few different cars and then modify them using a few functions we'll create ourselves.

Creating a struct

In something as complex as car, there are lots of different properties that we might want to assign. We can use strings to represent things like the make, model, features etc. numbers to represent properties such as horse power or top speed and arrays to capture the variety of engine sizes available, or perhaps types of fuel. The properties of a struct are contained between curly braces, and just like when you initialise any other data type, you must end your declaration with a semicolon.

Using a Struct

Now that we've created a prototype for all cars in general, we can now go ahead and make specific cars by using our car struct.

The Dot Operator

The dot operator, `.`, is used to assess specific members (elements) of a struct. In the example of our car structure, it has properties, or elements corresponding to the make, model, horse power and, top speed. Each of these can be accessed using the syntax `name.element` like in the example below:

Typedef

With other data types

Example Code

Here is the complete code for this section:

And the code for changing values:

Using pointers to modify values globally, rather than confined to the scope of the `getTopSpeed()` function.