# Lecture 10 Notes

Derek Harter

2015-04-07

## 1 First Session (11 - 11:40)

### 1.1 Structures in C

- `struct` are **aggregrate** data types - they can be built using elements of primitive types.
- `struct` are basically a **record**, a row of a database or table.
- `struct` are an example of a **user defined type**. Like `enum` they allow.
- In this course we don't cover object oriented analysis and design, but basically classes in C++ are `struct` user defined types with associated methods that operate on the new type.

### 1.2 Defining a struct

```
struct Trial
{
   string name;
   string gender;
   float reactionTime; // ms
   int numberOfPresses;
}; // don't forget the semicolon
```

- By convention, always use an initial upper case letter for user defined types. In the previous example, `Trial` is a user defined type (an experimental trial record for a single participant).
- Usually a `struct` definition should be done globally, it usually doesn't make sense to have a new data type that is only defined inside of a single local function scope.

### 1.3 Allocating and accessing a struct

- Declare a variable (or array) of the new type, just as you would any built-in type.
- Use . with name of field to access fields for reading or writing to fields in an allocated structure.
- Can use a comma separated list initializer, as we with arrays.

### 1.4 Good Programming Practice enum

- We haven't used it a lot, but we actually have seen an example of user defined type before, using enum (coin, Heads or Tails).
- Gender is a qualitative variable (it takes on named values, not numbers). Qualitative variables always only take on a discrete set of allowable values. You should always define an enum type when using a discrete qualitative variable, for readability and to reduce errors.

## 2 Second Session (11:45 - 12:30)

### 2.1 Using structures with functions

- Structure elements can be accessed individually if needed, and used or passed to function separately.
- However, a `struct` is a new data type, so we can create functions that take the `struct` as a parameter.
- We can even have functions that return `struct` as result, thus this is one way to create functions that return more than 1 value (e.g. to return a whole record of values).
- **NOTE**: `struct` can potentially be very large, so can be inefficient to pass stucture by value (which is copied). So sometimes might want to pass by reference instead when large.

## 3 Third Session (12:40 - 1:40)

### 3.1 Arrays of structures

- It is useful to create arrays of structures, this represents a basic table of records that we can process.

- You can create an array of any user defined type, just as you would an array of integers, or of floats, etc.

- Likewise, you can pass arrays of structures to functions as we have done with other types. The arrays will be passed by reference, just as arrays of built-in types are passed by reference.