COMP1917: 11 Structs, Debugging and Other Topics

Sim Mautner

s.mautner@unsw.edu.au

August 23, 2016

Table of Contents

This set of lecture slides contains notes on:

- Structs
- Typedef
- Header files
- Debugging with GDB
- `ctype.h` library

Structs: Introduction

Until now we have seen:

- Variables which hold single values. (`int`, `char`, `double`)
- Variables which hold multiple values of the same type. (arrays)

Now we want to be able to use:

- Variables which hold multiple values of different types.

Structs

For Example:
Until now, if we wanted to keep track of numerous number plates, we can use an array. But if we want to keep track of speeding tickets, we need the number plate (a string) and the speed that the vehicle was going at (a `double`). To do this, we use a struct:

```
struct _speedingTicket {
   char numberPlate[PLATE_SIZE+1];
   double speed;
};
```

This creates for us a new data type, so instead of being limited to `char`, `int`, `float`, `double` etc and arrays of such data types, we now also can define something as a `struct _speedingTicket`.

Speeding Ticket Exercise

- Ex 1: Write an application which uses a struct to store the details of a speeding ticket. Write the corresponding functions to create and print speeding tickets.
- Ex 2: Update the speeding ticket exercise to include the date of the speeding ticket. Discuss different ways of implementing the date in the struct.

Typedef

Used to simplify type names:

```
typedef struct _speedingTicket SpeedingTicket;
```

Or all in one with the struct definition:

```
typedef struct _speedingTicket {
   char numberPlate[PLATE_SIZE+1];
   double speed;
} SpeedingTicket;
```

Now instead of:

```
struct _speedingTicket ticket = createTicket();
```

We can use:

```
SpeedingTicket ticket = createTicket();
```

Speeding Ticket Exercises

- Ex 3: Update the speeding ticket exercise to use typedefs to simplify the declaration of speeding ticket and date variables.

Header Files

Purpose:

- As code becomes longer, breaking it down into multiple files makes it easier to locate sections of code.
- Separates code into modular pieces for re-use.

- Ex 4: Update the speeding ticket exercise so that the `date` functionality is in a separate file.

Debugging with GDB

- Compile with: -g (-gdwarf-2 on cse machines).
- Run with: gdb a.out
- `run` or `r` - Start running the application.
- `where` or `backtrace` or `bt` - Print stack trace (which function called which function to get to the current place in execution).
- `break n` - Stop executing when line `n` is reached.
- `break function_name` - Stop executing when `function_name` is entered.
- `continue` or `c` - Continue execution until the next breakpoint.
- `step` or `s` - Execute the next line of code.
- `quit` or `q` - Quit gdb.

`ctype.h`

This library includes functions such as:

- `isdigit`
- `islower`
- `isspace`
- `ispunct`
- and many others

Don't forget to `#include <ctype.h>` to get it to work!!