Function Prototypes

In this assignment you are asked to code several functions and only given the function prototype, also known as the function signature, for instance:

```
int getSomething(struct Something *);
```

This prototype is not a complete function definition, it is simply the signature of the function. It is used by the compiler to do a first check of the application code to ensure that all components are being used correctly. The compiler checks each use of **getSomething** to ensure that it passes the address of a **Something**, and returns an **int** value. The compiler does not care whether or not the return value is used by the caller.

If the usage test indicates that all functions and variables are being used correctly, it continues and completes the full compilation of your code. If the usage test fails (function is not being used correctly) the compilation stops and an error message is posted.

Function Definition

The definition expands upon the prototype to tell the compiler **what** to do when a function is called (provides the function logic with the necessary instructions on **how** it is to be done).

Thus: `int getSomething(struct Something *)`

Becomes:

```
int getSomething(struct Something * meaningfullParameterName)
{
      int result = 0;
      //implement functionality here
      //result may be changed to another value
      return result;
}
```

Note:

It is common in industry to see function prototypes missing parameter identifiers (only shows the data type as in the example above). However, it is considerably more meaningful and developer-friendly to include self-described names for each parameter to help convey the function requirements.

This is particularly important when there are multiple parameters of the same data type and the order in which the arguments are given is important – the identifiers (names) will qualify what is expected.

Header Files

A header file may contain structure declarations, as you have already seen in MS2, and function prototypes. The functions and structures declared or prototyped must all relate to each other (highly cohesive), this is a best practice. So, for instance, in assignment 1, the file **`contacts.h`** only contains declarations and prototypes for variables and functions related to a Contact. This is not the place to include a definition of an unrelated struct or a function such as **`int getNumberOfOranges(struct Orange *orange)`**.

Function Definition File

Functions are not defined (the function logic/body) in the header file, they are defined in another source file. In this case, since the header file is named contacts.h, the source file is named contacts.c. Again, only functions related to Contacts may be defined here, this is a best practice.

The contacts header file must be included at the beginning of the contacts source file:

```
#include “contacts.h”
```

The quotation marks indicate that the header file is in the same folder as the source file.

Best Practice

Commonly observed coding behaviours that other programmers expect and recognize. In this course, and other courses in the C/C++ programming stream, failure to abide by best practices may result in loss of marks. In real life, it may result in loss of job!