ENGG1003 - PASS Session 4

Mitchell Deltoer

Monday	**14-15**	**ES238**
Wednesday	**12-13**	**ES238**
Thursday	**10-11**	**MCLG42**

Switch, Case, Break - Optional

A *switch statement* is another kind of *flow control* that allows a variable to be tested for equality against a list of values. Each value is called a *case*, and the variable or expression being switched on is checked for each *case*. The standard syntax of a *switch statement* is

```
switch(expression){
  // you can have any number of case statements
  case constant1:
    statement(s);
    break; // optional
  case constant2:
    statement(s);
    break; // optional
  default: // optional (this is executed if no other case is true)
  statement(s);
}
```

If the `break` statements are omitted, the program will continue executing each *case* that follows the first valid check.

A *switch statement* can be used in place of an `else if` chain for allowing a program to choose between several different options, especially where there is a large number of options to choose from. They are never absolutely necessary, it's your choice whether you include them in your program over an `else if` chain.

Functions in C

A *function* is a block of code which can be called multiple times. Functions have a function name, an optional *return value* (output) and optionally one or more *arguments* (inputs). The standard structure of a *function prototype* is

```
  return_type function_name(arguments);
```

where `return_type` is the data type of the *return value* (`void` for no output), and `arguments` is either `void` for no arguments or a list of arguments in the form

```
  arg1_type arg1_name, arg2_type arg2_name, ... , argN_type argN_name
```

where `arg_type` is the data type of each argument and `arg_name` is the name of each variable within the function.

Like variables, functions need to be defined before they are used. The *function prototype* should be included before the `main` function. The *function definition* (where the actual function code goes) should be included after the `main` function.

Functions are very useful when it comes to breaking down a problem into several sub-problems, and when a particular feature of your code is very cumbersome to read (more than 10-20 lines).

Variable Scope and Persistence

Whenever a variable is declared within a *code block* (between any pair of curly braces), i.e. in a `while` loop, `if` code block or a *function*, the variable's "existence" is limited to this code block. Where a variable exists is called a variable's *scope*.

Both the value and the definition of a variable are lost when the program is outside of a variable's scope. However, the value of a variable can be retained if it is declared as `static` but their *scope* is still limited.

These concepts are critical for programming in C, especially when it comes to writing *functions*.

Practice Programming

Some things to consider when using functions within your program:

- What *arguments* (inputs) are there to the function? What are their data types?
- What is the *return value* (output) of the function? What is its data type?
- Have I remembered to (correctly) include the *function prototype* and the *function definition*?

Task 1: Addition function

Write a C program that reads two numbers and returns their sum. The addition should be done within a function called `addition`, and the main function should then print the result to the console window.

Task 2: Pythagoras function

Write a C program that reads the lengths of two sides of a right-angled triangle and returns the hypotenuse of the triangle. The calculation should be done within a function called `pythagoras`, and the main function should then print the result to the console window.

Task 3: Maximum function

Write a C program that reads two numbers and returns the largest number. The calculation should be done in a function called `maximum`, and the main function should then print the result to the console window.

Task 4: Factorial function

Write a C program that reads an integer and returns the factorial of that number. The factorial should be computed within a function called `factorial`, and the main function should then print the result to the console window.

Task 5: Print function

Write a C program that displays a block-letter of the first letter of your name using $\star$ characters. It should be printed through a function called `print_letter`. Use the main function to call this function to view your result. The following shows a few examples of sample output (however, feel free to get creative).

```
**    **     ******     ******      *****     **  **
***  ***       **         **       **         **  **
********       **         **       **         ******
** ** **       **         **       **         **  **
** ** **     ******       **        *****     **  **
```

Task 6: Temperature conversion function

1. Write a C program that reads in a temperature in Celsius and returns the temperature in Fahrenheit. The conversion should be computed within a function called `temp_convert`, and the main function should then print the result to the console window.

$$\boxed{F = \frac{9}{5}C + 32, \qquad\qquad C = \frac{5}{9}(F - 32)}$$

2. Extend the previous program's functionality to instead read in a temperature in **either** Celsius or Fahrenheit, and also a character which will determine the units of their temperature input. The function should convert from C to F if their input is 'C' or from F to C if the input is 'F'. The main function should then print the result to the console window.