# ENGG1003 - Lab 2

Brenton Schulz

## 1 C Summary

This section will be included in all future lab documents and lists a summary of C language features taught prior to the lab session. It will grow each week.

Not everything listed in this section is required to complete a particular lab.

### 1.1 Basic Structure

```c
#include <stdio.h>
int main() {
   // Your program starts here
   return 0;
}
```

### 1.2 Comments

```c
// This is a comment to end of line

/* this is a block comment
   which could span
   multiple
   lines
   */
```

### 1.3 Operators

| Operation | C Symbol |
|---|---|
| Addition | + |
| Subtraction | - |
| Multiplication | * |
| Division | / |
| Modulus | % |
| Increment | ++ |
| Decrement | -- |
| Less than | < |
| Less than or equal to | <= |
| Greater than | > |
| Greater than or equal to | >= |
| Equal to | == |
| Not equal to | != |

Table 1: Arithmetic operators in C

### 1.4 Operator Shorthand

Many arithmetic operators support the following shorthand syntax. The left and right columns present equivalent statements.

| | |
|---|---|
| `x = x + y;` | `x += y;` |
| `x = x - y;` | `x -= y;` |
| `x = x * y;` | `x *= y;` |
| `x = x / y;` | `x /= y;` |

### 1.5 Data Types

| Type | Bytes | Value Range |
|---|---|---|
| `char` | 1 | -128, +127 |
| `unsigned char` | 1 | 0, 255 |
| `short` | 2 | -32768, 32767 |
| `unsigned short` | 2 | 0, 65535 |
| `int` | 4 | $\approx \pm 2.1 \times 10^9$ |
| `unsigned int` | 4 | 0, 4294967296 |
| `long` | 8 | $\approx \pm 9.2 \times 10^{18}$ |
| `unsigned long` | 8 | $0, 1.8 \times 10^{19}$ |
| `float` | 4 | $1.2 \times 10^{-38}$ to $3.4 \times 10^{38}$ |
| `double` | 8 | $2.3 \times 10^{-308}$ to $1.7 \times 10^{308}$ |

### 1.6 Standard i/o

Read a single variable from `stdin` with `scanf();`
`scanf("format specifier", &variable);`

Write a single variable to `stdout` with `printf();`
`printf("format specifier", variable);`

You can use `printf();` *without* a newline (`\n`) to create an input prompt:

```c
printf("Enter a number: ");
scanf("%d", &variable);
```

This prints:

```
Enter a number:  _
```

where _ indicates the terminal prompt (ie: where typed characters will appear).

## 1.7 Format Specifiers

The following table is woefully incomplete. The compiler *may* generate warnings if `%d` is given something other than `int` and `%f` is given something other than `float`. An attempt will be made to ensure these are sufficient.

| Data Type | Format Specifier |
|---|---|
| Integers | `%d` |
| Floating point | `%f` |
| Float with `n` decimal places | `%.nf` |

Table 2: Basic format specifiers

## 1.8 Type Casting - Advanced

Placing the syntax `(type)` before a variable name performs a type cast (ie: data type conversion).

eg: convert `float` to an `int` prior to using its value. This forces a rounding-down to the nearest integer.

```
float a;
// ...
y = (int)a * z;
```

**NB:** This does **not** modify the original variable.

Data type "upgrades" are done automatically by the compiler but sometimes it is desired to downgrade or force esoteric behaviour. Adding it unnecessarily doesn't have any negative impact. Applications in ENGG1003 will be limited but it comes up regularly in embedded systems and nobody else explicitly teaches type casting. I have used it extensively in the low-level art of *bit banging*: manual manipulation of binary data. This is, unfortunately, beyond ENGG1003.

## 1.9 Flow control

Flow control allows selected blocks of code to execute multiple times or only under a specified condition.

### 1.9.1 `if()`

The `if()` statement executes a block of code only if the *condition* is true. The condition is an arithmetic statement which evaluates to either zero (false) or non-zero (true).

Syntax:

`if(condition) {/* other code */}`

Full example:

```
if(x > 10) {
   // Do stuff
}
```

Condition Examples:

- `if(x) // if(x is not zero)`
- `if(x+y) //if((x+y) is not zero)`
- `if(y >= 5)`
- `if(1) // Always executes`
- `if(0) // Never executes`
  - Can be used for debugging. Might be easier than a block comment /* */

**NB:** *NEVER* place a semicolon after an `if()`, that stops it from having any effect. The block after it will always execute. This bug can take days to find.

### 1.9.2 `while()`

The `while()` flow control statement executes a block of code so long as a condition is true. The condition is checked before the block is executed and before every repeated execution.

The condition rules and examples are the same as for those listed under the `if()` statement.

Syntax:

`while(condition) {/* other code */}`

Example:

Evaluate the infinite sum:

$$\sum_{n=0}^{\infty} \frac{1}{n^2} \tag{1}$$

to a precision of $1 \times 10^{-6}$

```
float sum = 0.0;
int x = 0;
while(1/(x*x) > 1e-6) {
   sum = sum + 1/(x*x);
   x++
}
```