**Introducing Functions**

We’ve already been using some functions!

- **main** is a function
- **printf** and **scanf** are also functions

**What is a function?**

- A separate piece of code identified by a name
- It has inputs and an output
- If we “call” a function it will run the code in the function

---

**Function Syntax**

We write a function with (in order left to right):

- An output (known as the function’s type)
- A name (for humans to remember)
- Zero or more input(s) (also known as function parameters)
- A body of code in curly brackets

```c
// a function that adds two numbers together
int add(int a, int b) {
    return a + b;
}
```

---

**Return**

An important keyword in a function

- **return** will deliver the output of a function
- **return** stops the function running, and returns to where it was called from
- **return** must be executed before function end reached
  - except **void** functions (explanation to come)

---

**How is a function used?**

If a function already exists (like printf)

We can use a function by calling it by name

And providing it with input(s) of the correct type(s)

```c
// using the add function
int first_number = 4;
int second_number = 6;
int total;
total = add(first_number, second_number);
printf("%d\n", total);
return 0;
```
Compilers and Functions

How does our main know what our function is?
A compiler will process our code, line by line, from top to bottom.
If it has seen something before, it will know its name.

// An example using variables
int main(void) {
    // declaring a variable means it's usable later
    int number = 1;
    // this next section won't work because the compiler
    // doesn't know about other_number before it's used
    int total = number + other_number;
    int other_number = 5;
}

Functions and Declaration

We need to declare a function before it can be used
// a function can be declared without being fully
// written (defined) until later
int add(int a, int b);
int main(void) {
    int first_number = 4;
    int second_number = 6;
    int total = add(first_number, second_number);
    return 0;
}

// the function is defined here
int add(int a, int b) {
    return a + b;
}

Void Functions

We can also run functions that return no output
- We can use a void function if we don’t need anything back from it
- The return keyword won’t have a variable or value if it is used at all

// a function of type "void"
// It will not give anything back to whatever function
// called it, but it might still be of use to us
void print_add(int a, int b) {
    int total = a + b;
    printf("The total is \%d ", total);
}

Function Example - Cubes

double cube(double x);
int main(void) {
    printf("42 cubed is %lf\n", cube(42.03));
    double a = 2;
    double b = cube(a);
    printf("2 cubed is %lf\n", b);
    return 0;
}

double cube(double x) {
    double result;
    result = x * x * x;
    return result;
}

source code for cube.c
Summary of Function Properties

- Functions have a type - the type of the value they return
  - Type `void` for functions that return no value
- Functions take 0 or more parameters
  - `void` used to indicate function takes no parameters
- Functions have their own variables
  - Created when function called
  - Destroyed when function returns
  - Not accessible outside the function
- Parameters are also variables - initialized with the value supplied by the caller
  - Changed in the function have no effect outside the function
- `return` statement stops execution of a function
- `return` statement specifies value to return
  - Unless function is of type `void`
- Run-time error if end of non-`void` function reached without `return`

Library functions

- Over 700 functions are defined in the C standard library.
- You’ll need to use less than 20 of these in COMP1511
- The C compiler needs to see a prototype for these functions before you use them.
- You do this indirectly with `#include` line
- For example, `stdio.h` contains declarations for `printf` and `scanf` so:

```c
#include <stdio.h>

int main(void) {
    printf("Andrew Rocks!\n");
}
```

Scanf return value

- Scanf() is a function we’ve used already
  - It returns the number of values that it has successfully read and stored
  - This means we can use the return value of the function
  - It will tell us whether or not we’ve received a valid input

Function Example - Quadratic

```c
double evaluate_quadratic(double a, double b, double c, double x);
int main(int argc, char *argv[]) {
    double x;
    printf("Enter x: ");
    scanf("%lf", &x);
    double value = evaluate_quadratic(2.0, -3.0, 4.0, x);
    printf("2x^2 - 3x + 4 = %lf\n", value);
    return 0;
}
```

```c
double evaluate_quadratic(double a, double b, double c, double x) {
    return (a * x * x) + (b * x) + c;
}
```

Source code for `quadratic.c`
double quadratic_root(double a, double b, double c);
double evaluate_quadratic(double a, double b, double c, double x);
int main(int argc, char *argv[]) {
    double a, b, c;
    printf("Enter a: ");
    scanf("%lf", &a);
    printf("Enter b: ");
    scanf("%lf", &b);
    printf("Enter c: ");
    scanf("%lf", &c);
    double root = quadratic_root(a, b, c);
    printf("Calculated root is %lf\n", root);
    double value = evaluate_quadratic(a, b, c, root);
    printf("The value of the quadratic for %lf is %lf\n", root, value);
    return 0;
}

double quadratic_root(double a, double b, double c) {
    return (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
}

// given x calculate value of polynomial with coefficients a, b, c
double evaluate_quadratic(double a, double b, double c, double x) {
    return (a * x * x) + (b * x) + c;
}