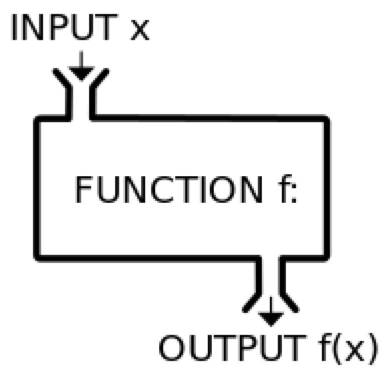
1- Introduction

This book will try to teach you the basics of the C programming language. It does not assume you have any programming experience. It is helpful to have taken enough math to know what a function is (https://en.wikipedia.org/wiki/Function_(mathematics)). You do not need any extra software or other books.

All the exercises in chapter 1 can be done with pen and paper. You can check your answers here. We'll start programming on the computer in Chapter 2.

Functions

Mathematical functions take some input and produce an output:



Below is a mathematical function.

$$f(x) = x + 1 $$

It takes some input number, x, and returns some other number, $x+1$. Using the function is just a matter of doing the math: $f(5)= 5 + 1 =6$ and $f(-1)= -1 + 1 = 0$. f is just a name, which could be anything we like. We could write the same function as

$$PlusOne(x) = x + 1$$

But its still the same function.

Programming in C or any other language can mostly be thought of as giving the computer some input and having it calculate (compute) some output. The input can come from the user typing at their keyboard, using their mouse, or using reading some file on their computer or downloaded from the internet.

> Exercise 1 (written):
>
> 1. Write a mathematical function which returns twice the input
> 2. Write a mathematical function which return the negative of its input

Answers

If we wanted to write the mathematical function above as a function in the C language it would look like this:

```
int PlusOne(int x)
{
    return x + 1;
}
```

This might look a like hieroglyphics at first so we are going to break it down one piece at a time.

In `C` we always tell the computer the type of data we are dealing with. `int` above stands for *integer* which means a positive or negative whole number. -1, 0, 1, and 1234567 are integers. 0.123 is not an integer. (We'll talk about other types later). In line 1 the first thing written is the type of data the function will output, which in this case is `int`.

Next we have at least one space, then the function's name; in this case `PlusOne`. The name can be almost anything but can't start with a number and can't include a few other special characters like spaces. **The name does not have any effect on the function's behavior**. I called this function `PlusOne` because that is a good description of what it does, but if I called it `MultiplyBy100`, or `AAA` the computer would still do the same calculations. It is a good idea to choose a name that gives some idea of what the function does; this will make using it later a lot easier.

After the name, in the `( )` parentheses we put the function *inputs*. There can be 0, 1 or more as shown below. This function has a single `int`eger input.

Thats all for line 1!

Next we enclose the part of the function where the *work* happens with `{ }` curly brackets. These brackets don't *do* anything except tell the computer where the start and end of the function is.

Inside the brackets are statements. Statements tell the computer how to product the output from the inputs. Statements always end with a `;` (semicolon). The `;` does not *do* anything. It just helps the computer to know the statement is over. This function has a single `return` statement. The statement:

```
    return <whatever>;
```

means whatever follows `return` is the output.

Because we told the computer that `PlusOne` will output an `int` in line 1, that has to be the type of data that follows the word `return`. If we tried to `return x+0.1` that would be an error, because if x is an integer then `x+0.1` is not an integer (not a whole number). If we forget a return statement that is also an error. I'll explain how the computer will let you know you've made an error later.

You may also notice we put several spaces before the `return` statement (it is *indented*). These are optional, but most people think it makes it easier to read. The placement of `{}` on lines by themselves is optional too. The program above can as easily be written

```
int PlusOne(int x){return x+1;}
```

but most people find this harder to read. Another similar convention you will see in almost all `C` code to make it easier to read is to indent a few spaces when we are inside a `{ }`.

THe technical term for *using* a function is calling the function. To call our function above we would write `PlusOne(5)`. Any place you write `PlusOne(5)` in our program it is basically the same as inserting a `6`.

> Exercise 2: What value is output (returned) by `PlusOne(-1)`?

Answers

Function Inputs

Our first function above has a single `int`eger as input. Some functions will have no inputs:

```
int ReturnFive()
{
    return 5;
}
```

Some functions will have more than one input

```
int Sum(int input1, int input2)
{
    return input1 + input2;
}
```

you just separate each input with a comma.

> Exercise 3:
>
> 1. Write a `C` function which takes a input a a single integer parameter returns twice the value.
> 2. Write a `C` function which returns takes a single integer input parameter returns the negative of it.
> 3. Write a `C` function which takes two integer parameters and returns the first minus the second

Answers

Variables

Lets look at another function

```
int AddThreeTimes(int x)
{
    return x + 5 + 5 + 5;
}
```

Hopefully it is clear what this function is doing. But what if we decide to revise our function later and wanted to change the number we adding from `5` to something else? We'd have to remember everywhere we wrote `5` in our program and change each one. We can write this function differently to avoid this hazard.

```
int AddThreeTimes(int x)
{
    int addMe = 5;
    return x + addMe +addMe + addMe;
}
```

Now the function has one extra statement on line 3 that defines a new variable called `addMe`. With this line we are telling the computer, `I want to store an integer value in the name addMe`. Now later if we want to use that value we can write `addMe` and will get the same result as putting that value there.

The rules for valid names of variables are basically are the same as functions. That is: names can contain all letters, some numbers and symbols, but can't start with a number. Just like with function names, variable names Names don't effect how they are used. Name things whatever you like but try to make the name meaningful.

You can change the value stored in a variable at any time in another statement as below:

```
int AddThreeTimes_v2(int x)
{
    int addMe = 5;
    addMe = 10;
    return x + addMe +addMe + addMe;
}
```

> Exercise 4: What is the value returned by `AddThreeTimes_v2(11)`?

Types

Whether we are store values in variables, or are defining function's inputs or outputs we always need to tell `C` what kind of data we are working with, this is called the data's type. Some simple types are described below, we'll cover more complex types later.

`int`

This is the one we have seen already. `int`s are positive or negative whole numbers. The largest and smallest numbers that can be stored in an `int` will depend on the computer you are working with, but a range of *approximately* -2^{32} (min) to 2^{32} (max) is common.

`float` and `double`

To use decimals, we need something other than integers. Another numeric type is `float`. This stand for `floating point`, a fancy way of saying a number that *might* include a decimal point. 1,2,3 are floats, but so are `1.1` and `2.1` and `3.1` and `1,234.5`. Floating point numbers are often written in **exponential form** like `334e6`, which is an easy way of writing $334 * 10^6$.

We can only store up to a certain precision of number in a float. For example, you can't store the exact value of π in a `float` because it would require it to store infinitely many digits and we'd run out of memory. (we'll learn more on computer memory in a future section)

More often than `float` you will see and use the type `double` which is just a `float` that uses **double** the memory but has **double** the precision.

Some examples of functions that use `double`:

```
double PlusOnePointFive(double x)
{
    return x + 1.5;
}

double Square(double x)
{
    return x * x;
}

double Divide(double x, double y)
{
    return x / y;
}
```

> Exercise 5: Write a function that takes a `double` named `radius` as input and returns a double which is the area of a circle with that radius. (hint: you can approximate π as 3.1415)

`char`

`char` is short for **character**. Most of the letters, numbers, or symbols on your keyboard can be stored as a `char`.

```
char GiveMeAnE()
{
    return 'E';
}
```

Sets of characters can be used to display messages to the user. I'll show you how to do this in the next chapter.

Summary

- `C` functions take some data input and produce some output.
- You always tell `C` the type of data we are working with
- You the names of functions and variables will not affect the behavior
- Some common types are `int`s like 5, `double`s like `3.1412` and `char`s like `'Z'`

To Chapter 2