Part 1: Input > Process > Output

In this secion you will write your first C programs, and learn about the differnt bits and pieces that are required to compile and run software written in C. By the end of this section you should be able to

- Write a simple, interactive, exectuable program.
- Understand essential C syntax
- Be aware of different data types in C

Hello World!

"Hello World" is typically the first program that students learn to write. It's origins are as old as C itself ...

```
#include <stdio.h>
#include <stdlib.h>

int main(){
  printf("Hello World");

  return 0;
}
```

Comments

Single line comments

```
//this is a single line comment.
```

Multi line comments span two or more lines.

```
/*
This is a
multiline
comment
*/
```

Escape Characters

sequence	output
`\a`	alert
`\n`	new line
`\t`	tab

Many More!

Conversion Characters

Data Type	Character
String	%s
int	%d
float	%f
pointer	%p

It is also possible to output a specific number of digits from a float like so:

```
int main(){
  printf("I ate some %f", 3.141593);
  printf("I ate some %.4f", 3.141593);
  printf("I ate some %.2f", 3.141593);

  return 0;
}
```

Variables

A variable is simply a placeholder where we are going to store some information in the computer's memory. There are three things you need to do to create a variable in your program. First you need to tell the compiler what **type** of data you'll be working with - i.e. is it a number or letters etc. Then you need to **name** your variable, so that it can refer to it elsewhere in your code. Finally you need to give your variable a **value**. The following code snippet has improved the previous example by including variables:

```
int main(){

  //declare a floating point variable and name it "pi"
  float pi;

  //assign the pi variable a value of 3.141593
  pi = 3.141593;

  printf("I ate some %f", pi);
  printf("I ate some %.4f", pi);
  printf("I ate some %.2f", pi);

  return 0;
}
```

To make your life easier you can **declare** and **initialise** a variable in one line like so:

```
float pi = 3.141593;
```

Arithmetic

You will recall that a computer is defined as a machine that can perform arithmetic and logic operations. So far we've done neither, so we'll try a bit of arithmetic.

```
int main(){

  //declare an integer variable
  int a = 7;

  //return the int squared
  printf(a*a);

  return 0;
}
```

Great... let's try some more. You code will execute from top to bottom, so new values can be assigned "on-the-fly":

```
int main(){

  int currentYear = 2015;
  int zuckerBorn = 1984;
  int babyGates = 1955;
  int age;

  //calculates how old Mark Zuckerberg is:
  age = currentYear - zuckerBorn;
    printf("Mark Zuckerberg is %d years old \n", age);

  //here we reuse the age variable to compute how old Bill Gates is.
  age = currentYear - babyGates;
    printf("Bill Gates is %d years old \n", age);

    return 0;

}
```

Types of Variables:

Integer Types

Integer data types are for storing whole numbers. There are signed and unsigned variants of these, where the allocated size is the same but the range oof possible values is changed.

Variable Name	C Identifier	Size	Range
Character	char	8 bit	0 - 255
Integer	int	16 bit	pm 32,000
Short	short	16 bit	pm 32,000
Long	long	32 bit	pm 2 billion

Floating Point Types

Floating point data types allow increasing levels of presicsion for calculations by providing more and more decimal places. For most engineering applications 15 decimal places will usually suffice.

Variable Name	C Identifier	Size	Precision
Float	float	32 bit	6 decimal places
Double	double	64 bit	15 decimal places
Long	long	80 bit	19 decimal places

Working with Strings

Unlike other (primative) variable types, strings are instantiated using `String` with a capital S. This is because a string is actually an array of `chars` - in other words a collection of characters.

String Terminator

When you go to run your code the C compiler needs to know where every string ends, or terminates. To do this, the compiler parses your program and everywhere it finds closing quotation marks it insterts a character known as the **string terminator**. The string terminator is simply `\0` and takes up one additional byte of memory. This means that if you create a String variable to store your name, and if your name is 8 characters long, the name variable will actually take up 9 bytes of memory.

Overwriting Strings

To overwrite an existing string use the `strcpy()` function. This function takes two **arguments** or parameters. Remember, the `strcpy()` function can be used to overwrite *any* string, so the first thing you need to tell is is what string to replace, then you need to tell it what to replace it with.

Arrays.

Let's explore arrays in a bit more detail. As mentioned already, an array is simply a collection of data. The only syntactic difference between a primative data type and an array is an extra set of square brackets e.g.`int integerArray[];` Each new element of an array is seperated by a comma. For example, you might use an array to store your lotto numbers: `int luckyNums[] = [4,8,15,16,23,42];`

Indexing

What is we want to know,for example, what the third element of an array is? Well, we can look it up using it's **index**. Just like with a book, an index is used to look up information you want to find. Now intuitively you might think that the third element of an array would be found at index 3 - however C, like that vast majority of programming languages, start indexing arrays at 0. This means the the first elements is at index 0, the second is at index 1 and so on. So to print our third lottery number to the console we could write:

```
printf("%d", luckyNums[i]);
```

Preprocessor Directives

Up until now we haven't really paid any great attention to the `#include` code at the beginning of our programs. The `.h` files that are included are known as header files or preprocessor directives. This is because the contents of these files are called *before* the main.c is compiled.

Writing a simple header.

Lets create a header with some commonly used mathematical constants

```
DEFINE PI = 3.141593
DEFINE EULER = 1.6
```

If you're using an IDE you'll see that this new .h file is saved in a new folder called "headers". Typically your compiler will expect your headers to be in a predefined location —— and this is implied in the code by using brackets `#include <someHeader.h>` however, our hearder is stored in the same directory as our source code so acccess it we use quotation marks instead: `#include "myHeader.h"`

Input!!

So far we've been manipulating predefined data - this is all well and good but it doesn't make for particularly interactive programs. In this section we will look at the `scanf()` function, which is used for reading (or scanning!) in data from the console.

scanf() syntax

Both `scanf()` and `printf()` are part of the `<stdio.h>` header, so hopefully they will look quite similar to you. As we saw before, when printing data we needed to tell the function both the **type** of data we are working with and a **value** for that data to have. Take a look at the following simple example:

```
int myInt;

scanf("%d", myInt);
```

If you build and run the previous example you will just see a console with a blinking cursor - not particularly intuitive for the end user.

```
int age;

printf("What year were you born in ? \n");
scanf("%d", age);
```

Challenge

Improve the snippet above so that it asks the user for their year of birth and then returns their (approximate) age to them.

stretch goal use system time and users D.o.B to get exact age, or to create a birthday countdown.