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>` library, or header file, 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.