SYS2/CCCP: Data Types and Structures

For Java and Python Programmers

Dr David Griffin

Data Types

C has two fundamental data types

Integers
Floats

C also has a lot of ways of looking at Integers

Different sized integers
Signed / Unsigned Numbers
As ASCII characters

But just don't be surprised that some strange things work

Like you can add two letters together. Please don't.

Defining a Variable

Defining a variable works by specifying it's type, name and optionally it's default value.

```
int x = 5;
```

```
float y = 0.4;
```

```
int z;
```

If you don't specify a default value, it's undefined. That normally means "0", but this isn't guaranteed.

Basic Integer Types

`char, short, int, long`

A `char` is the smallest data type, a `long` is the biggest

Fun fact: C does **not** specify how big they are! C only defines the order of the sizes i.e. a `char` is smaller than an `int`.

Normally though, the sizes are defined like this.

But they don't have to be like this. One well known textbook defined char = 6 bits.

If you want to know exactly how big something is, C provides the `sizeof` function which tells you how much memory a variable occupies

```
char normally_1_byte_int = 0;

short normally_2_byte_int = 0;

int normally_4_byte_int = 0;

long normally_8_byte_int = 0;
```

Specific Sized Integers

For simplicities sake, this course will assume that the sizes of integer types are the "normal" sizes

But if you really want to be sure, modern C provides the following types in stdint.h

If your C compiler has stdint.h, you can use these types to be explicit about the sizes of integer variables. However, older C compilers might not have it.

stdint.h has a lot of other types in it, like "fastest" or "at least this big" types. These can help the compiler optimise

```
#include <stdint.h>


int8_t 8_bit_int = 0;

int16_t 16_bit_int = 0;

int32_t 32_bit_int = 0;

uint8_t unsigned_8_bit_int = 0;

uint16_t unsigned_16_bit_int = 0;

uint32_t unsigned_32_bit_int = 0;
```

Signed and Unsigned Integers, Overflow and Underflow

- C integers can be signed or unsigned. By default they are signed.
- Signed integers can hold negative numbers, Unsigned integers cannot, but can store larger positive numbers.
- This is important because C does not check overflow and underflow
  - If you do the following:

```
unsigned short x = 0;
x = x - 1;
```

  - `x` becomes `65535`
- A compiler might warn you about this if it can work it out, but it doesn't have to.

Booleans

C also has a `bool` data type for truth values

Don't let the name fool you. It's just a `char`.

In C, anything with a value of `0` is `False`

Anything with a value which is not `0` is `True`

Because `bools` are actually `chars`, you can do silly things like add two `bools` together. Don't do this. There are better ways to manipulate `bools`.

But using the `bool` type correctly can make your code easier to read.

Floats

`float, double`

A `float` is a 32-bit data type that can store non-integer numbers

A `double` is a 64-bit version of a float

Without hardware acceleration, float and doubles are slow

So you might want to check what your system provides hardware acceleration for

You might also have acceleration for floats but not doubles

Arrays

Arrays can be defined by adding a subscript to the definition

```
int x[3] = {1, 2, 3}; // initialises an array of 3 integers
char name[16] = "Fred"; // initialises 16-char array to null terminated string "Fred"
float coords[2]; // defines an array of 2 floats
```

Subscripts are used to access elements in the array

```
x[0] == 1; // true
name[3] == 'f'; // false - note that single quotes mean single character
coords[0] = 2.3; // set first coord to 2.3
```

Note: You cannot assign multiple elements in C

`char` arrays are often used to represent ASCII strings

- C assumes ASCII strings end with the `null` character, so always allocate one more `char` than you need for your data
- C has a datatype `wchar` for Unicode, but that's beyond the scope of this course

Be careful! It's perfectly valid in C to go past the end of an array!

- This will almost certainly cause "undefined behavior" i.e. bugs
- Cybersecurity people will sometimes call bugs like this a buffer overflow

Manipulating Strings

- In C you cannot use a lot of normal operators on strings.
  - The reasons for this will be explained later
- You must use string manipulation functions from `<string.h>`
- `<string.h>` has other functions in at as well, but not all of them are a good idea to use
  - For example, `strcopy` doesn't have the maximum number of characters to copy, which makes it easy to overflow the size of `str_1`
- Note that string manipulation functions tend to manipulate strings in place – they do not return a string

```
void strncopy(char[] str_1, char[] str_2, int n)
```

- Copies at most `n` characters from `str_2` to `str_1`

```
void strncat(char[] str_1, char[] str_2, int n)
```

- Appends at most `n` characters for `str_2` to `str_1`

```
bool strcmp(char[] str_1, char[] str_2)
```

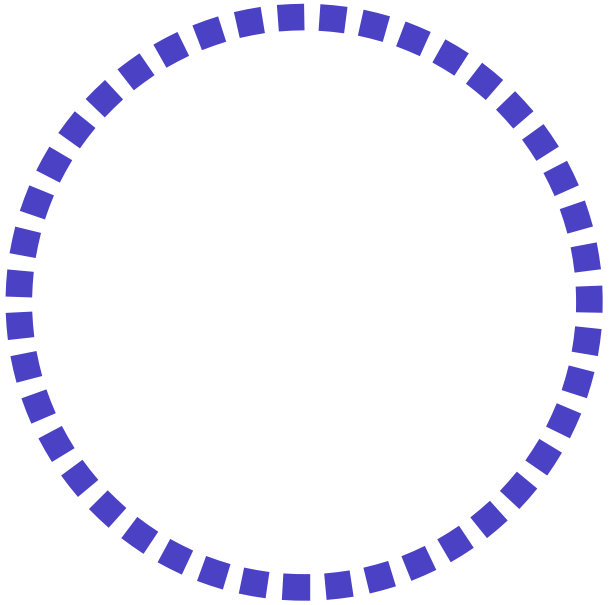
- Compares if two strings are equal

```
int strlen(char[] str)
```

- Returns an integer for the length of the string

void

- **void** is a special variable type that indicates the absence of a type
- It can be used either when
  - A function returns nothing through the normal function return mechanism (like Java)
  - The type of something isn't known
    - But you can't allocate a variable whose type isn't known. We'll come back to this use later.



Other things you might see

Type Qualifiers: `const`, `volatile`

- `const` tells the compiler that the variable should never be modified
  - This might sound strange, but it's very good for finding bugs. We'll come back to this when we look at functions in more depth.
- `volatile` tells the compiler the variable might be modified by something not in the program
  - For example, hardware sensors
  - This will slow down the variable, as each time it's accessed the program must fetch it from main memory.

Storage Qualifiers: `extern`, `register`, `static`

- `extern` tells the compiler the variable is defined in another file
- `register` suggests that the compiler puts the variable into a CPU register for fast access
  - Use this one sparingly! Normally the compiler is better at optimising than you are!
- `static` makes the variable live forever
  - This is useful as a more controlled form of global variable
  - e.g. a static variable in a function won't be reallocated on the next function call – the value will be the same as when the function last ran.

Structures

A structure is a set of variables that are grouped together

They're useful for

- Organising code
- Passing lots of logically linked variables to functions
- Hiding messy bits of detail the user of an API doesn't need to see
  - For example, a UI toolkit will normally gives you access to structures that represent state, but it doesn't tell you what's in them

```
struct report {
    char   name[50];
    char   author[50];
    char   department[100];
    int    id;
};

struct report r;
r.id = 1;
```

Structures

C will normally work out the best memory layout for a structure itself

It optimises mainly for speed

If storage is at a premium, or you're reading an obscure file format, you can specify "packed" structures

You don't have to give a struct a name

You can declare a variable of this struct in the definition

If you don't give the struct a name, you should probably do this!

```
struct {
    unsigned int beep:1;
    unsigned int light:1;
    unsigned int msg_id:6;
} global_flags;

global_flags.msg_id = 4;
```

Conclusions

Everyone here should have learned about the

basic data types in C	how to use arrays	how to manipulate strings with string manipulation functions	how structures can be used to group variables together

There's still a lot to cover about C, but variables are the basic building block of just about everything else

Assignment

The lectures assignment looks at data types, ASCII strings and structures

It focuses on highlighting some of the features of C

As well as some the pitfalls

By the end you should be able to create and use variables, structures and strings.

- Hopefully in a way that doesn't cause the program to crash