University of York
Department of Electronics

Programming the qbcan and Sensing the World

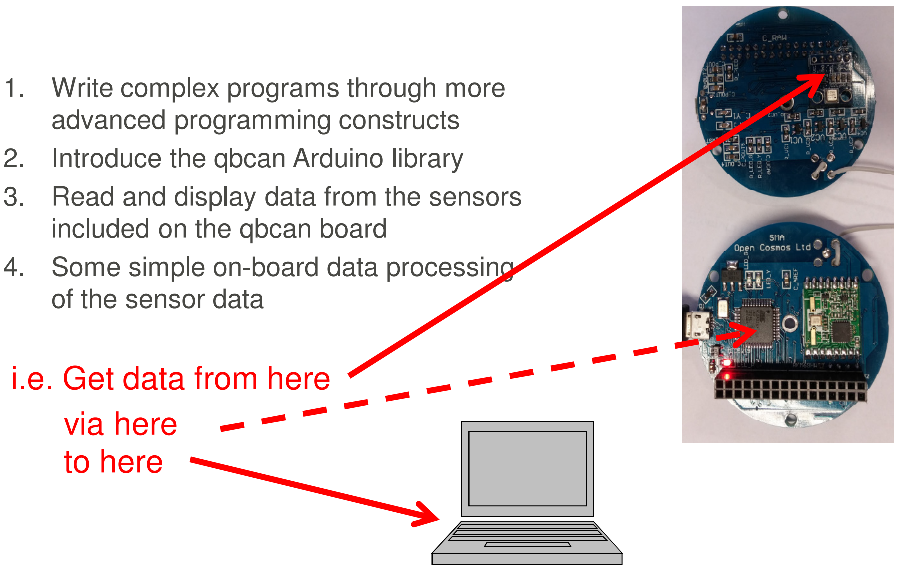
CanSat Workshop 2016

Matthew Rowlings

Outline

University of York
Department of Electronics

1. Write complex programs through more advanced programming constructs
2. Introduce the qbcan Arduino library
3. Read and display data from the sensors included on the qbcan board
4. Some simple on-board data processing of the sensor data



Programming

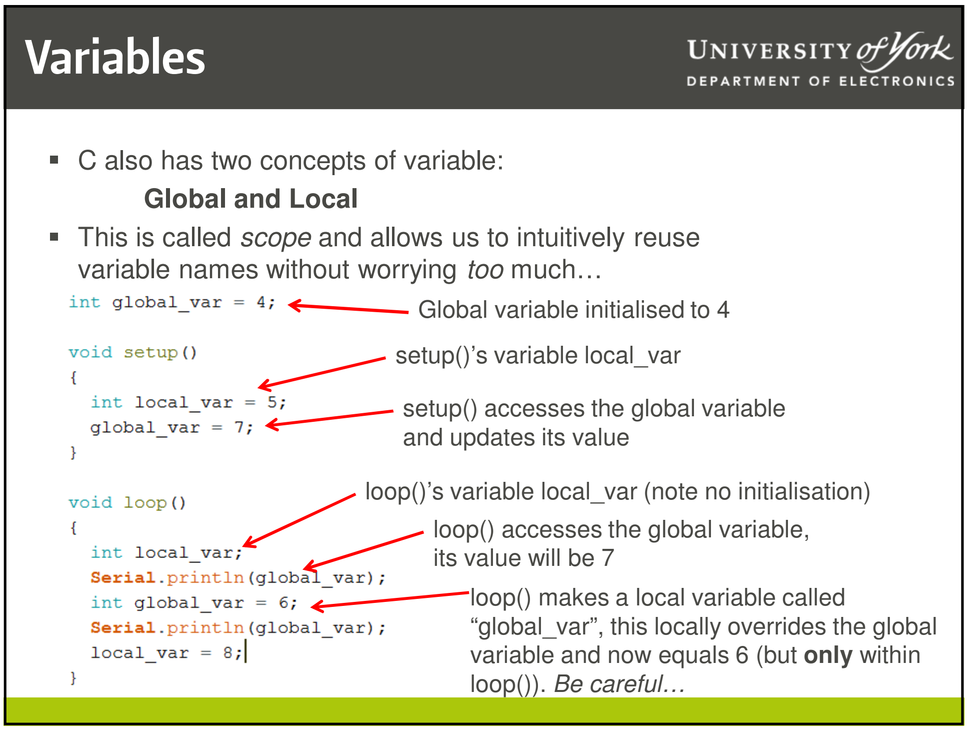
Variables

- C provides several “types” of variables:
  `char` – a “byte” of data, characters, integers -128 to 127
  `int` – integer numbers from -32,768 to 32,767
  `float` – decimal numbers (lots of them!)
  `double` – decimal numbers (even more than float!!)
  `bool` – boolean values (true/false)

```
char letter = 'a';
int number_of_results = 100;
float pi = 3.1415927;
double pi_2 = 3.141592653589793;
char string[] = "Hello_world!";
```

- Advanced users can make their own types from *structures* or *classes.*

```
#include <qbcan.h>
BMP180 bmp;
```



Variables

UNIVERSITY *of York*
DEPARTMENT OF ELECTRONICS

- C also has two concepts of variable:

  Global and Local
- This is called *scope* and allows us to intuitively reuse variable names without worrying *too* much…

```
int global_var = 4;

void setup()
{
  int local_var = 5;
  global_var = 7;
}

void loop()
{
  int local_var;
  Serial.println(global_var);
  int global_var = 6;
  Serial.println(global_var);
  local_var = 8;
}
```

Global variable initialised to 4

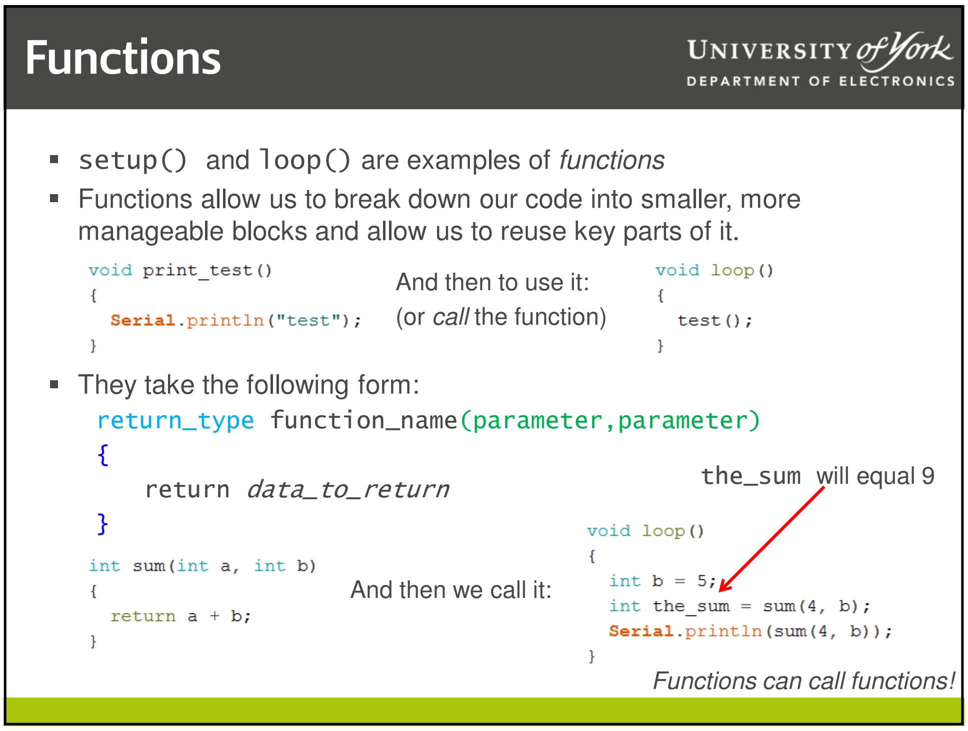
setup()’s variable local_var

setup() accesses the global variable and updates its value

loop()’s variable local_var (note no initialisation)

loop() accesses the global variable, its value will be 7

loop() makes a local variable called “global_var”, this locally overrides the global variable and now equals 6 (but **only** within loop()). *Be careful…*



Functions

UNIVERSITY *of York*
DEPARTMENT OF ELECTRONICS

- `setup()` and `loop()` are examples of *functions*
- Functions allow us to break down our code into smaller, more manageable blocks and allow us to reuse key parts of it.

```
void print_test()
{
  Serial.println("test");
}
```

And then to use it:
(or *call* the function)

```
void loop()
{
  test();
}
```

- They take the following form:

```
return_type function_name(parameter,parameter)
{
    return data_to_return
}
```

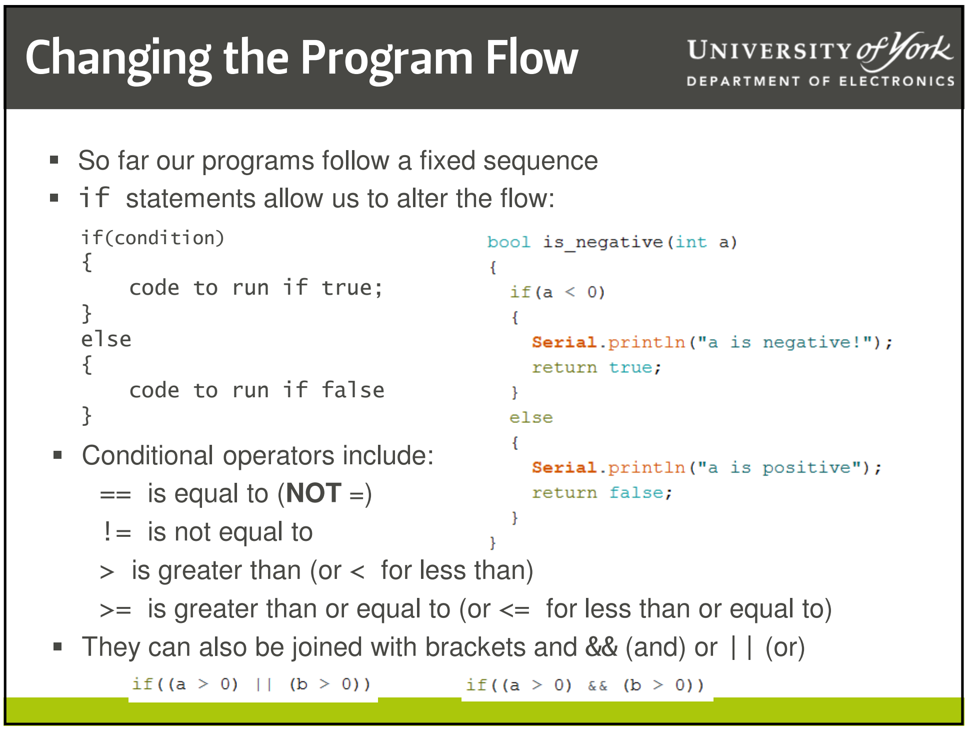
```
int sum(int a, int b)
{
  return a + b;
}
```

And then we call it:

```
void loop()
{
  int b = 5;
  int the_sum = sum(4, b);
  Serial.println(sum(4, b));
}
```

`the_sum` will equal 9

Functions can call functions!



Changing the Program Flow

UNIVERSITY *of York* DEPARTMENT OF ELECTRONICS

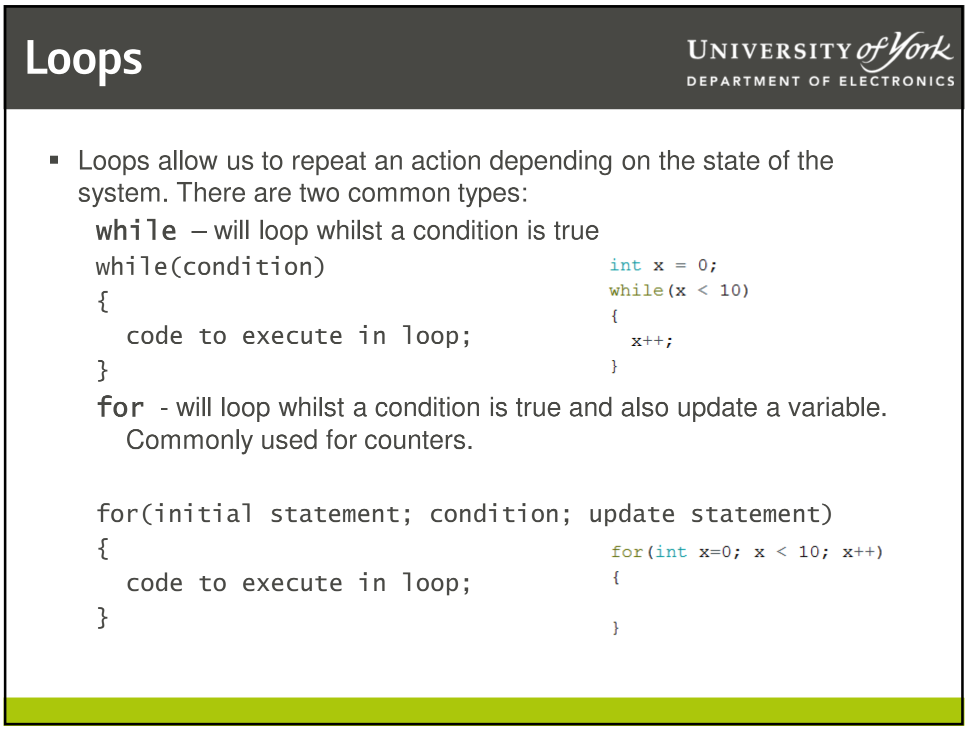
- So far our programs follow a fixed sequence
- `if` statements allow us to alter the flow:

```
if(condition)
{
    code to run if true;
}
else
{
    code to run if false
}
```

```
bool is_negative(int a)
{
  if(a < 0)
  {
    Serial.println("a is negative!");
    return true;
  }
  else
  {
    Serial.println("a is positive");
    return false;
  }
}
```

- Conditional operators include:
  - == is equal to (**NOT** =)
  - != is not equal to
  - > is greater than (or < for less than)
  - >= is greater than or equal to (or <= for less than or equal to)
- They can also be joined with brackets and && (and) or | | (or)

```
if((a > 0) || (b > 0))        if((a > 0) && (b > 0))
```



Loops

UNIVERSITY *of York* DEPARTMENT OF ELECTRONICS

- Loops allow us to repeat an action depending on the state of the system. There are two common types:

  `while` – will loop whilst a condition is true

```
while(condition)
{
  code to execute in loop;
}
```

```
int x = 0;
while(x < 10)
{
  x++;
}
```

  `for` - will loop whilst a condition is true and also update a variable. Commonly used for counters.

```
for(initial statement; condition; update statement)
{
  code to execute in loop;
}
```

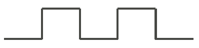
```
for(int x=0; x < 10; x++)
{

}
```

Serial Out

UNIVERSITY *of York*
DEPARTMENT OF ELECTRONICS

```
Serial.println("test");
```

- This function converts the argument into a binary data stream that we can send to a PC (or another device that is UART compatible). Wen

`Serial.print(5);` "5" -> 0101 $\longrightarrow$ $\boxed{\text{Serial UART}}$ 

- We then read this data using the "Serial Monitor" or another program (such as PuTTY)
- We do have to remember to set up the Serial port in the setup() function

`Serial.begin(9600);` This is the "baud rate" and is the speed that the data is sent at. Thus it is important that this value matches on both ends

- We can send many types of values with Serial.print()

```
Serial.print("Absolute pressure: ");
Serial.print(P,2);
Serial.println(" mb.");
Serial.print("Temperature: ");
Serial.print(T,2);
Serial.println(" deg C.");
```

```
Absolute pressure: 1234.56 mb.
Temperature: 78.90 deg C.
```

Serial Out

UNIVERSITY *of York*
DEPARTMENT OF ELECTRONICS

- We can send many types of values with Serial.print()

```
Serial.print("Absolute pressure: ");
Serial.print(P,2);
Serial.println(" mb.");
Serial.print("Temperature: ");
Serial.print(T,2);
Serial.println(" deg C.");
```

```
Absolute pressure: 1234.56 mb.
Temperature: 78.90 deg C.
```

- There is another way though:

```
char payload[50];
sprintf(payload,"T: %d C, P: %f mb.",(int)T, P);
Serial.println(payload);
```

- The % acts as a placeholder for the value, which follows after the string
- %d is a placeholder for a integer number
- %f is a placeholder for a floating point (i.e. decimal) number
- `payload` is an array of individual characters – this makes up the string

Comments and Macros

UNIVERSITY *of York* DEPARTMENT OF ELECTRONICS

- Comments are lines of code that are not compiled
- These are very, very, very, very helpful to assist others with understanding your code!

```
bmp.getData(T,P);
```

```
/*
 * These comments
 * can span
 * multiple
 * lines
 */
```

- Macros can also make your lives a bit easier
- These typically go at the top of the programming file and always start with a #

```
#include <qbcan.h>
```

The `#include` macro includes a code library that others have written or a file

```
#define CALIBRATION_VALUE 12.87
#define ME "Matt"
```

`#define` allows us to define constants that may occur many times in the file and this allows us to only have to change them in one place
Typically in CAPS to set them apart from variables, but this is not compulsory

UNIVERSITY *of York* DEPARTMENT OF ELECTRONICS

The qbcan Library

Qbcan Library

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

- OpenCosmos provide some support for the sensor and radio chips onboard the qbcan compact:
- To use the sensing chip (BMP180), they provide us with a custom *class* which contains functions which handle the communication with the sensors for us.

```
BMP180 bmp;
```

- To use it we have to initialise it. We only need to do this once, so it goes in the `setup()` function:

```
bmp.begin()
```

- This function in fact returns True or False depending on if the ProMicro could successfully connect to the BMP180. So we can test this value and report an error if need be: (error detection is very handy for debugging, especially if it only takes one wire to take out a sensor!)

```
if (bmp.begin())
  Serial.println("BMP180 init success");
else
{
  //In case of error let user know of the problem
  Serial.println("BMP180 init fail (disconnected?)\n\n");
  while(1); // Pause forever.
}
```

BMP180 Functions

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

- char begin();
- char startTemperature(void);
- char getTemperature(double &T);
- char startPressure(char oversampling);
- char getPressure(double &P, double &T);
- double sealevel(double P, double A);
- double altitude(double P, double P0);
- char getError(void);
- void getData(double &T, double &P);

RFM69 Functions

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

- There are several functions for operating the onboard radio, but T shall cover this later so this list is just for completeness.
- bool initialize(uint8_t freqBand, uint8_t ID, uint8_t networkID=1);
- void setAddress(uint8_t addr);
- void setNetwork(uint8_t networkID);
- bool canSend();
- void send(uint8_t toAddress, const void* buffer, uint8_t bufferSize, bool requestACK=false);
- bool sendWithRetry(uint8_t toAddress, const void* buffer, uint8_t bufferSize, uint8_t retries=2, uint8_t retryWaitTime=40);
- bool receiveDone();
- bool ACKReceived(uint8_t fromNodeID);
- bool ACKRequested();
- void sendACK(const void* buffer = "", uint8_t bufferSize=0);
- uint32_t getFrequency();
- void setFrequency(uint32_t freqHz);
- void encrypt(const char* key);
- void promiscuous(bool onOff=true);
- void setHighPower(bool onOFF=true);
- void setPowerLevel(uint8_t level);
- void sleep();

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

Reading Sensors

Sensor Reading Software

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

- So finally we can put this together to **read the cansat's sensors** and **send them down the serial line** to the PC!
- Even from just this one-line brief we now know that we need to include the Serial and the BMP180
- So don't forget to `#include` the qbcan library
- We need to make a variable for the BMP180 **and** access it from both `setup()` and `loop()` and so it should be a global variable
- We also need to remember to call the serial and BMP180 setup functions.
- Then we can finally read the sensors and write the data out within the `loop()` function.

- Exercise: go through the cheatsheat and sense the pressure and temperature of the room!

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

On-Board Data Processing

Why Do Data Processing?

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

- Some custom sensors you might connect for your secondary mission may output data in a “raw” form that needs processing to be useful.
- For example: a thermistor will just give you a voltage, you would need to convert that into a temperature by a calculation and possibly calibration
- Sensors are also noisy, whether you handle this on the ground or on the cansat is up to you.
- Was any of the sensor readings in the previous exercise noisy?

Data Processing Exercise

UNIVERSITY of York
DEPARTMENT OF ELECTRONICS

- Like all engineering projects, the spec for our cansat has changed:

  1. Now we need to report the temperature in degrees Kelvin.
  2. The pressure reading is too noisy. We need to average it to smooth it.

This exercise is up to you to do! But here are some hints:

- The Kelvin calibration is 273.15 degrees
  - But you should use a `#define` to implement it

```
#define KELVIN_CAL 273.15
```

- You will need to call `getData()` several times to read the pressure sensor and then average these values (3 – 5 readings) before printing the value