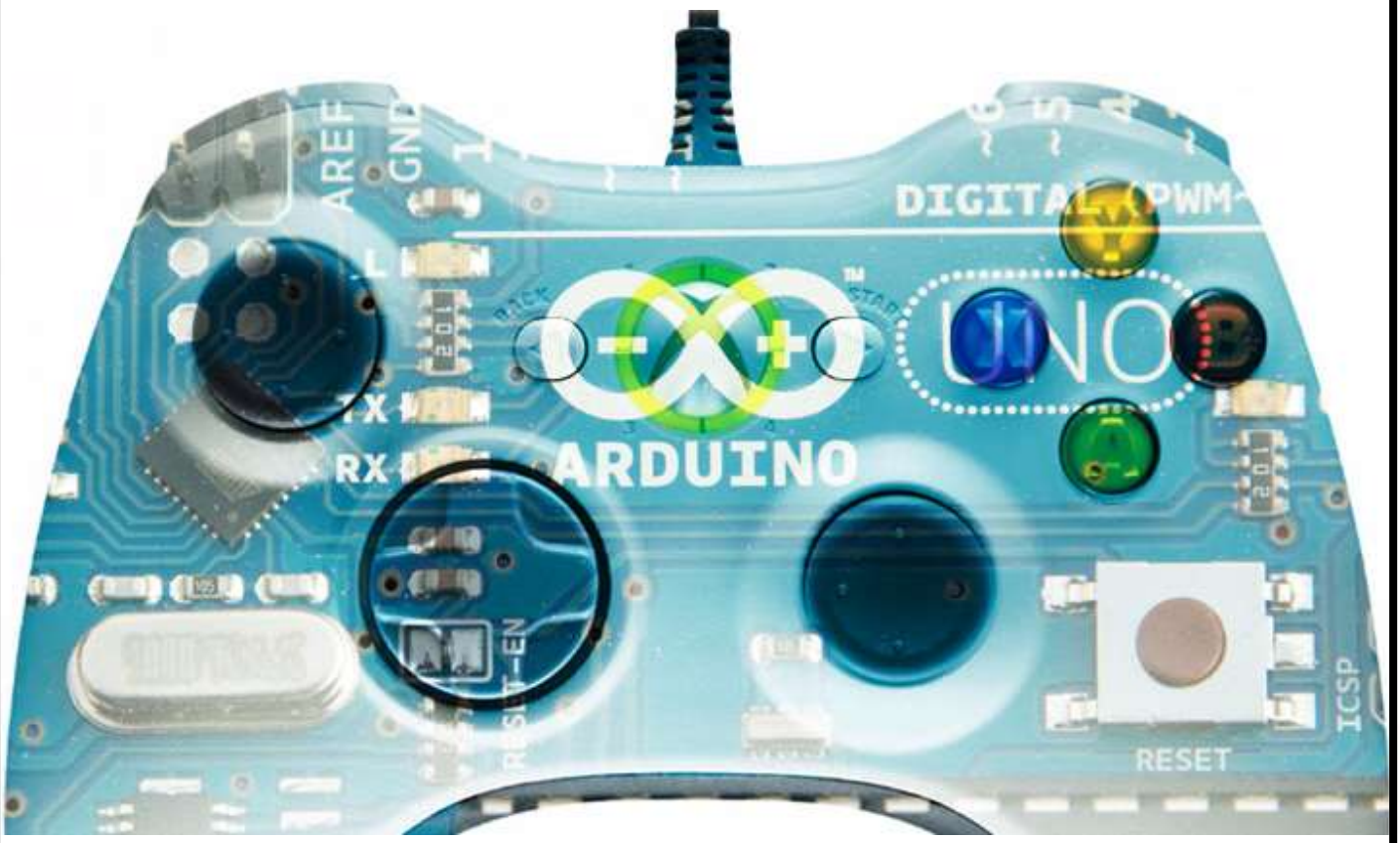
HIGHER ENGINEERING SCIENCE

Programmable Control



Name _____

Class _____

Teacher _____

Ellon Academy
Technical Faculty

Learning Intentions

- I will gain the ability to design and evaluate solutions to engineering problems in a range of contexts
- I will gain knowledge and understanding of key concepts related to electronic and microcontroller-based systems, and their application
- To know what a microcontroller is and how its used
- I will know advantages and disadvantages of microcontroller based control systems compared to a hard-wired electronic equivalent
- To construct flowcharts showing solutions to complex control programs, involving time delays, continuous and fixed loops

Success Criteria

I can develop programmable control systems for mechatronic systems by:

- Designing and simulating high-level programs to monitor digital inputs and initiate digital outputs
- Designing and simulating high-level programs to make decisions using arithmetic and logic functions
- Testing and evaluating programs against a specification

To access video clips that will help on this course go to
www.youtube.com/MacBeathsTech

A free website that you can use to simulate breadboarding and Arduino.
http://123d.circuits.io/

What is Programmable Control?

The automation of machines, process control and conveyor lines have resulted in the ever-increasing consistency of quality, speed and cost savings within complex processes. Consumers have come to expect high standards of quality in the manufactured goods they use, but to an engineer these are the challenges that make the profession interesting.

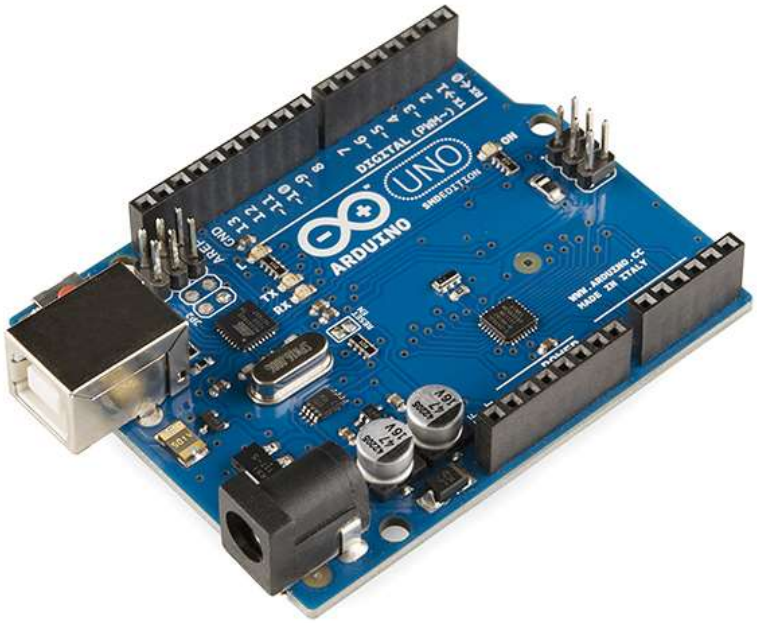
Programmable Control is a computer control system that continuously monitors the state of input devices and makes decisions based upon a custom program to control the state of output devices.

Almost any production line, machine function, or process can be greatly enhanced using this type of control system

This unit will build on skills learned in National 5 Engineering Science in order to further deepen your understanding of Programmable Control to enable you to design your own working systems and give you a deeper insight into a possible career route.

Arduino

Arduino I san open source Electronics prototyping platform that can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language. This language is frequently used in industry today due to its Simple, clear programming environment, its ability to work cross platforms, and its Open source and extensible software



The Microcontroller

A micro controller is often described as a **'computer on a chip'.** Microcontrollers have a controller and memory all built into a single chip. As they are **small, inexpensive** they can easily be built into other devices to make these products more intelligent and easier to use.

Microcontrollers are usually programmed for a specific electronic product – for instance, a microwave oven may use a single microcontroller to process information for all its electronic devices. By altering the microcontroller program the same 'brand' of chip can do many different tasks.

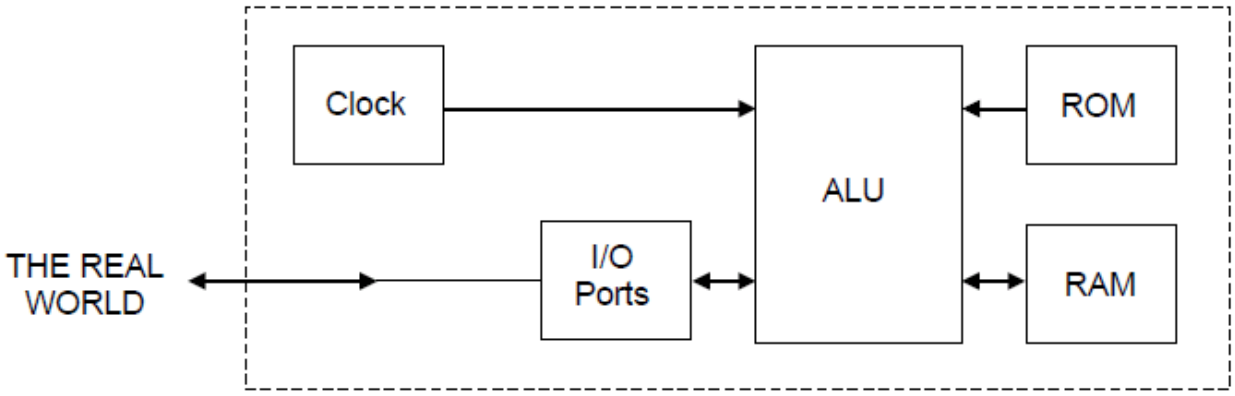
Advantages of using microcontrollers in a product design are:

- Increased reliability and reduced quantity of stock (as one microcontroller replaces several parts)
- Simplified product assembly and smaller end products
- Greater product flexibility and adaptability since features are programmed into the microcontroller and not built into the electronic hardware
- Rapid product changes or development by changing the program and not the electronic hardware

Inside a Microcontroller

The 'brain' of the stamp controller system is the 18 pin micro controller in the centre of the board. Although micro controllers are relatively cheap, they are very complex devices containing many thousands of transistors, resistors and other electronic components.

The main features of the micro controller are shown in the block diagram;



http://www.bbc.co.uk/schools/gcsebitesize/design/systemscontrol/electronicsrev3.shtml

ROM

The ROM (read only memory) contains the operating instructions for the micro controller. The ROM is 'programmed' before the micro controller is installed in the target system, and the memory retains the information even when the power is removed.

RAM

The Ram (random access memory) is 'temporary' memory used for storing information whilst the program is running. This is normally used to store mathematical answers that the micro controller comes out with as it is working. This memory is **'volatile'**, which means that as soon as the power is disconnected the contents of the memory are lost.

ALU

The processing unit (full name arithmetic and logic unit – ALU) is the 'control centre' of the micro controller. It operates by reading instructions from the ROM and then carrying out the mathematical operations for each instruction.

Clock

The clock circuit controls the speed at which these operations occur. The **clock circuit** within the micro controller 'synchronises' all the internal blocks (ALU, ROM, RAM, etc) so that the whole system works correctly.

Buses

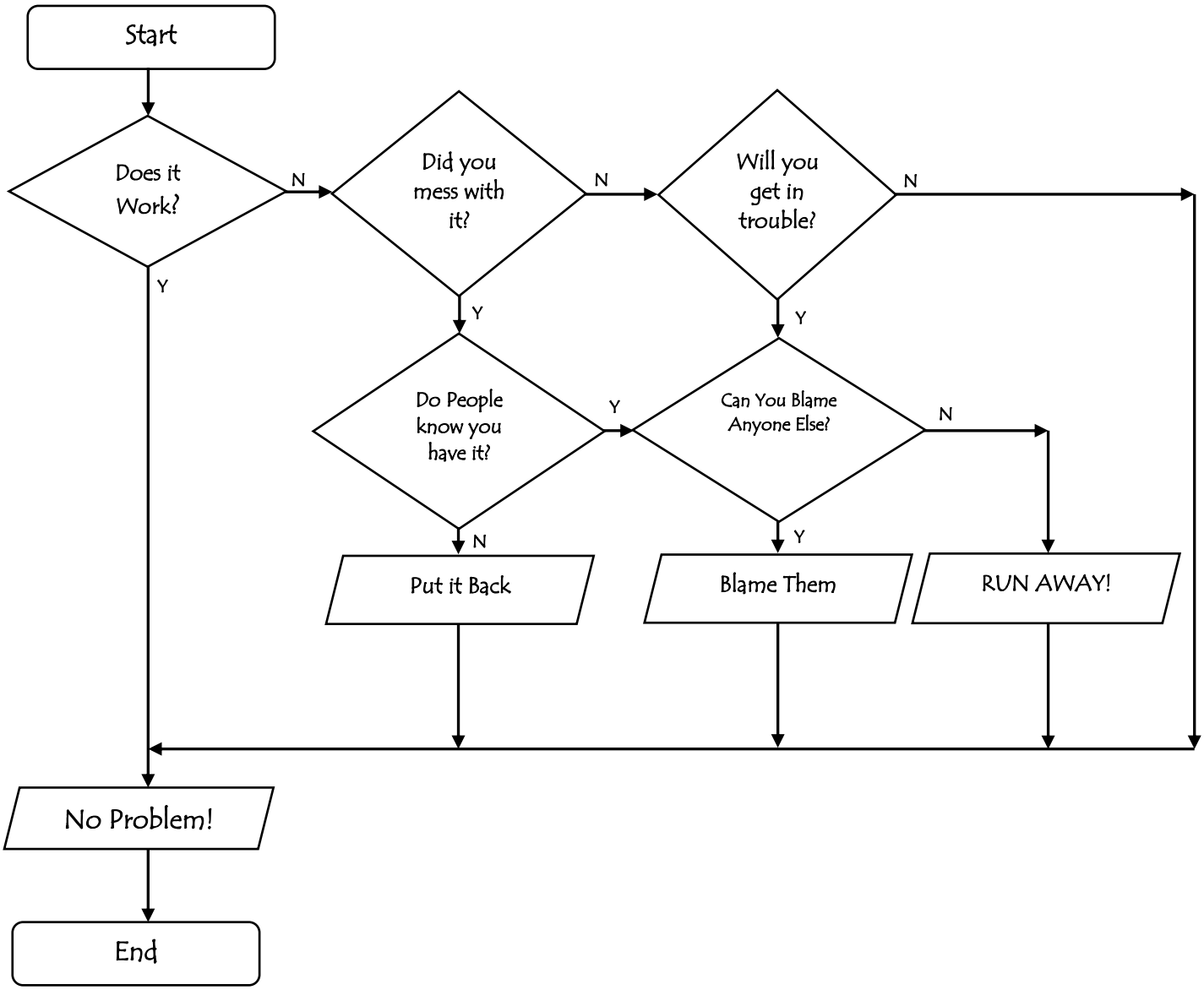
Information is carried between the various blocks of the micro controller along 'groups' of wires called **buses**. The 'data bus' carries data between ALU and RAM and the 'program bus' carries the program instructions from the ROM to the ALU.

EEPROM

EEPROM (Electrically Erasable Programmable Read Only Memory) is a type of memory that can be reprogrammed when desired, but it also keeps the program when the power supply is removed. This means the stamp controller will start to run the program currently in the memory whenever the power supply is connected.

Flowcharts

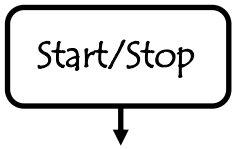
Flowcharts are commonly used to explain how a program works. As flowcharts are drawn graphically they often make programs easier to understand. A flowchart should be drawn for each program you develop.



Symbols

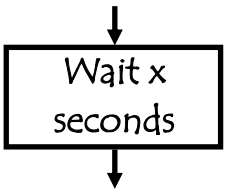
Start/stop symbol

The start/stop symbol shape is a rectangle with rounded ends. Each flow chart must contain one start and usually one stop symbol unless it is a continuous loop.



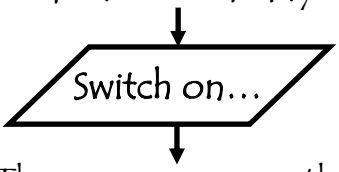
Wait symbol

The 'wait' symbol is a rectangle. The text inside the symbol explains how long the time delay is.



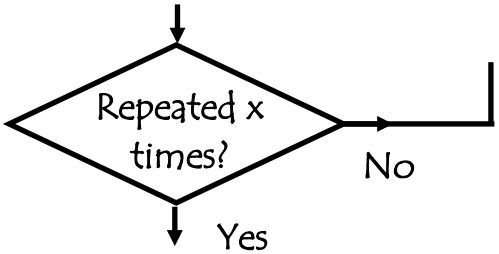
Outputs symbol

The 'outputs' symbol is a parallelogram. The text inside the symbol explains which output pins are switched on or off at any time.



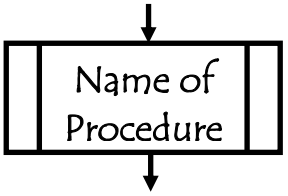
Decision Box

The Decision box is a diamond. The program uses this to check if something has been completed.



Sub Procedure

A **sub-procedure** is a small section of code that can be 'called' from a different part of the program. After the sub-procedure is finished program flow moves back to the original section of the program.



Continuous loop

A lot of the times it is necessary to create programs that has an infinite loop. This means it will loop' forever', and instead of a 'stop' symbol it will have a feedback loop. This is the most common type of programming as you would not want your code to end once it has completed the task only once. An example would be traffic lights – you would not want the programme controlling them to end as soon as it has went through each light once – you want it to last forever!

Task 1

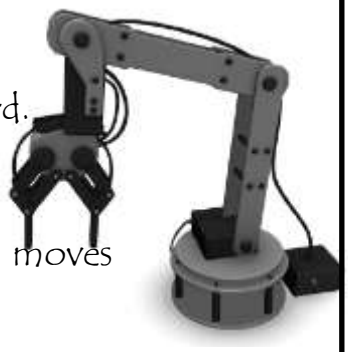
A set of temporary traffic lights is required for a system of road works.

RED	12s
RED & AMBER	3s
GREEN	15s
AMBER	3s

Draw a flow chart for the lights sequence shown above by one set of traffic lights. Use the times given in the table.

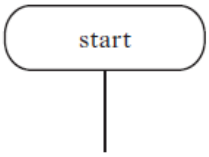
Task 2

A microcontroller is used to operate a robotic arm.
The sequence of operation for the robotic arm is shown below.



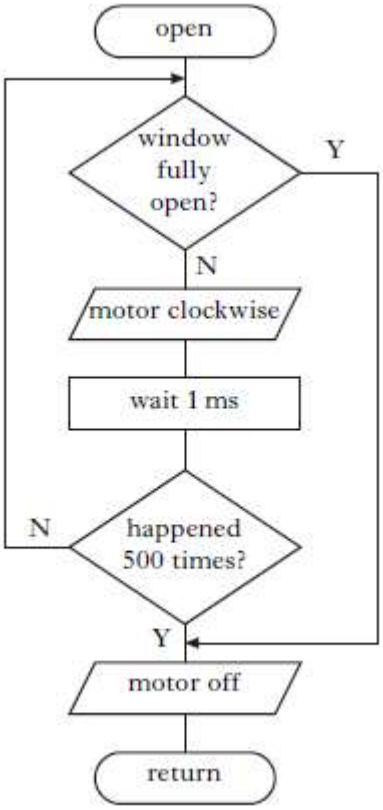
Step 1: Check if start switch is pressed.
Step 2: When start switch is pressed arm moves forward.
Step 3: After two seconds the arm stops.
Step 4: Jump to the sub-procedure "gripper".
Step 5: Once sub-procedure "gripper" is complete arm moves backward.
Step 6: After two seconds the arm stops.
Step 7: Repeat 200 times steps two to six before resetting to step one.

Complete the flowchart for the robotic arm operation.



Task 3

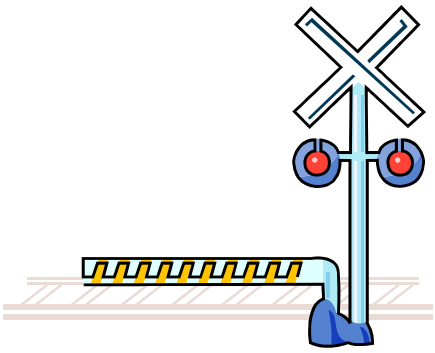
A microcontroller-based system controls the temperature in a greenhouse by opening and closing a window. Below shows the flowchart for the sub-procedure *open*. A motor rotates clockwise to open the greenhouse window.



Explain how the sub-procedure *open* controls the movement of the window

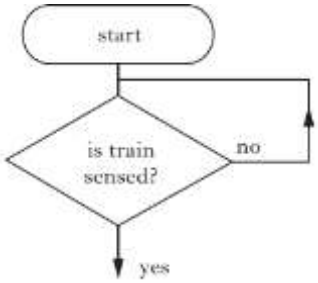
Task 4

A model railway uses a microcontroller to operate a barrier. The operation of the system is shown below.



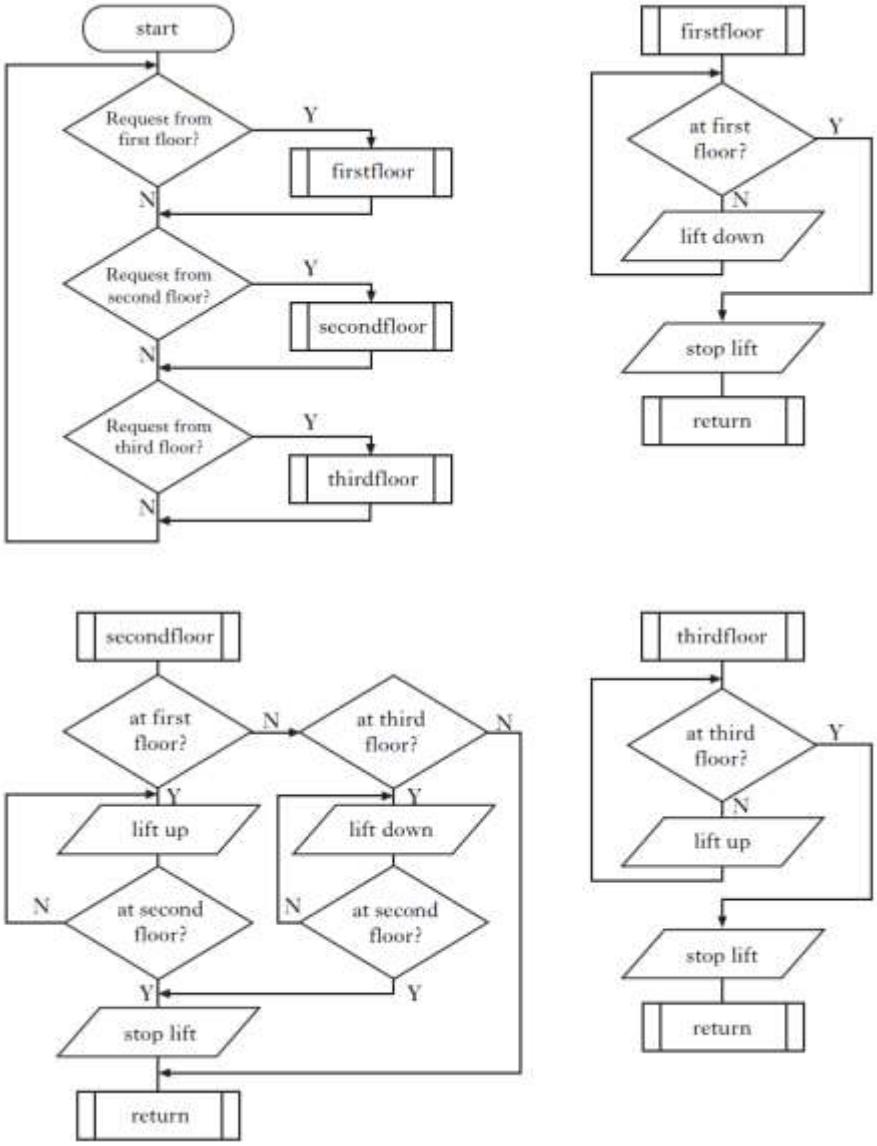
- When a train is sensed the barrier should lower.
- When a limit switch is pressed the barrier will stop.
- When the train is no longer sensed, wait for ten seconds.
- Barrier will rise.
- After three seconds the barrier will stop.
- Sequence will then repeat.

Complete the flowchart for the barrier operation



Task 5

A lift operates between 3 floors of a building, and is controlled by a microcontroller. Request buttons on the three floors are used to call the lift. The flowcharts below show the control sequence for the movement of the lift to the appropriate floor.

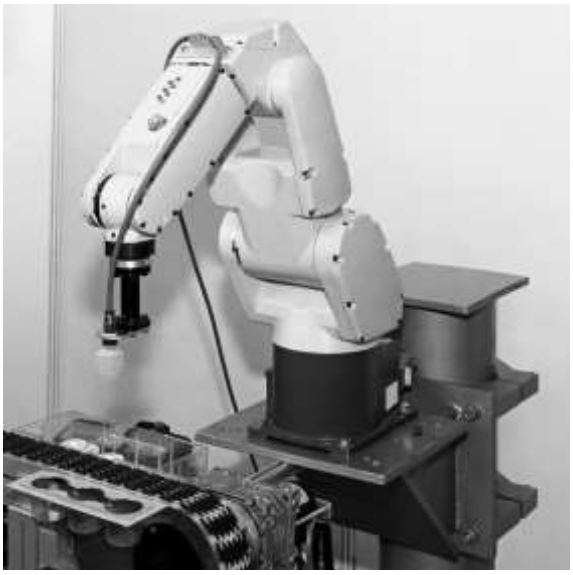


a) State the circumstances under which the lift will move upwards.

b) State the condition for which the lift will not move if a request is received for the second floor.

Task 6

The pick-and-place robotic arm lifts components from a conveyor and places them onto a rack. The arm is controlled by a microcontroller.



A sub-procedure *place* picks up one component at a time and places it onto the rack. A pre-written sub-procedure *moverack* repositions the rack after each component has been placed.

When 18 components have been placed, the rack is full.

The specification for sub-procedure *place* is as follows:

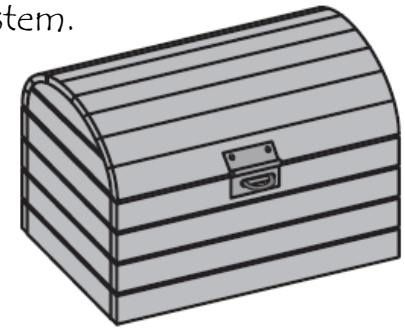
- Close gripper until limit-switch is high
- Arm motor forwards for 3·2 seconds
- Open gripper until limit-switch is low
- Arm motor backwards for 3·2 seconds
- Call sub-procedure *moverack*
- If 18 components have been placed, sub-procedure ends.

Task 6 (continued)

Draw a flowchart to represent the sub-procedure *place*.

Task 7

The research and development department of a toy company has appointed an engineering team to redesign the wooden treasure chest shown. The new version will have an electronically controlled locking system.



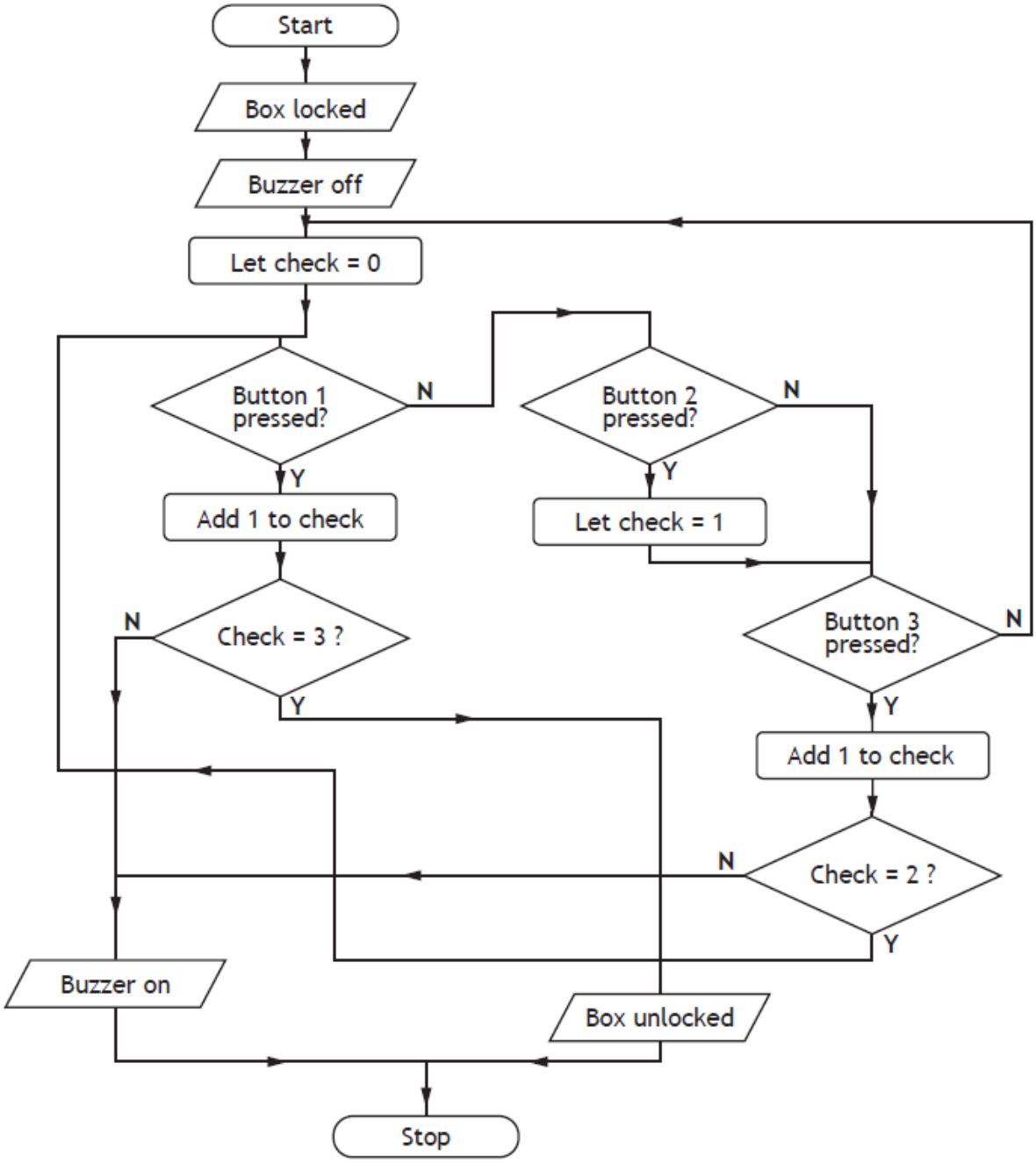
The prototype uses a microcontroller to process a three digit code to control the locking system. Each button can only be pressed once; when pressed, it latches on and stays high.

The partly completed table below lists all possible codes that could be entered. Only one of these codes will open the box without sounding the buzzer.

Code	Buzzer sounds	Box unlocked	Correct code?
123	✓	✗	no
132			
213	✓	✗	no
231			
312			
321	✓	✗	no

Use the flowchart on the following page to complete the table above, and so identify the correct code to open the box without sounding the buzzer.

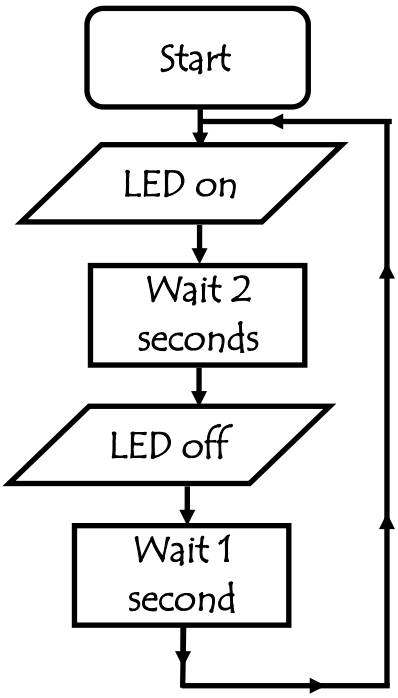
Task 7 (continued)



Writing a Programme

Once you have worked out your flowchart it is necessary to convert it into the Arduino programming language. As you know from your N5 course this is called a 'sketch'

To remind you of the language we will do a sketch of a basic flowchart



The first thing we have to do is name the Sketch. This can be any name whatsoever but it must start with a /* and end with a */

For example:

/*
Arduino is Cool!
I love Engineering Science!!!!
*/

Your next step is to set up integers and tell the Arduino what pins you're your inputs and outputs will connected to. In this case we are going to hook a LED up to pin 12 so we need to use this command

int ledPin = 12;

We now have to set up 2 parts to the program that must ALWAYS be there. Even if we do not want to use one section we put it in, we just don't put a program inside. These are the *void setup* and *void loop* commands

Void setup commands are ones that will run once the Arduino first switches on. This is where you tell your board what pins are to be treated as inputs, and which ones as outputs.

Void loop commands are one that will run continuously in a loop forever while the Arduino is on. This is where your main program will be written.

Within the *void setup* we need to set our pin 12 as an output. To do this we have to initialise the pins. You will need to use the *pinMode* command to focus on our specific pin and set it as output.

```
void setup()
{
  pinMode(ledPin, OUTPUT);
}
```

We now need to work on the writing of our program, and since it is in a continuous loop it needs to be put in the *void loop* area.

To do this we use the **DigitalWrite** command. This needs 2 pieces of information – which pin you use (or what you have named it previously), and whether the pin will be HIGH or LOW.

HIGH = on
LOW = off

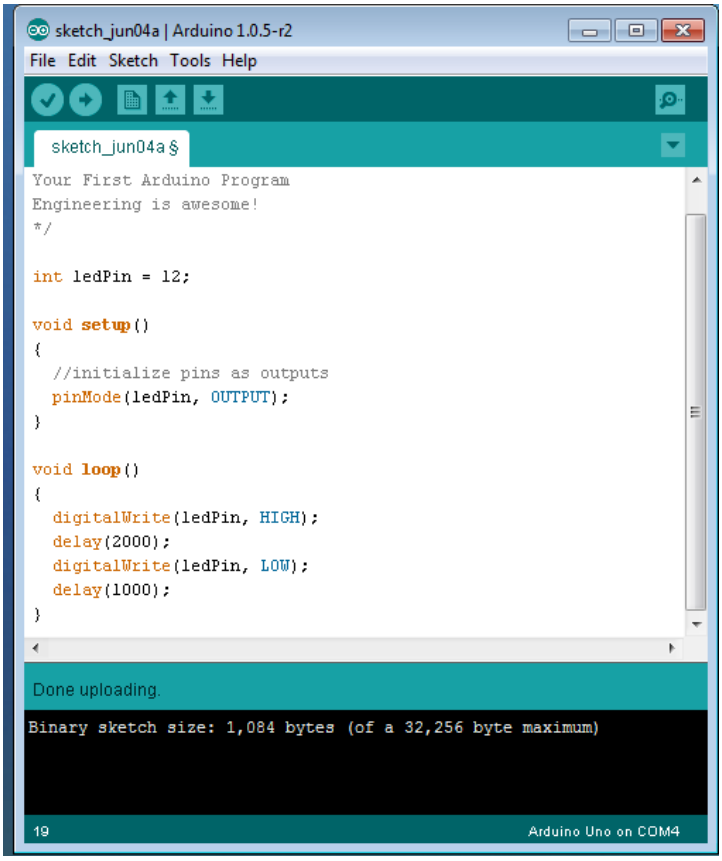
```
void loop()
{
  digitalWrite(ledPin, HIGH);
```

We then have to put in the delay using the **delay** command. This puts a pause in the program. This delay is measured in milliseconds.

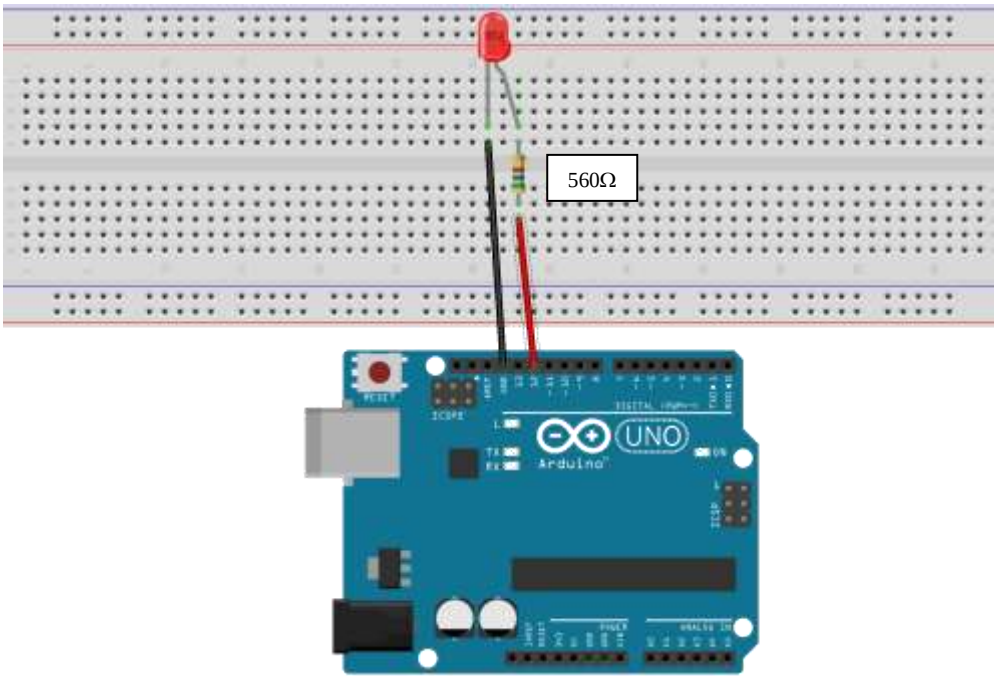
e.g. 1 second = 1000
2 seconds = 2000….etc

```
void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(2000);
```

We then repeat the same process for switching the pin off and putting the next delay.



Before we test the sketch we have to hook up the LED to the breadboard, using the skills you have learned in previous units.



*One wire has to come from Pin 12, the other from the ground. Make sure you also hook the LED properly! Long leg should be on the + side, short leg on the ground)

Understanding the Sketch

As you have discovered when writing a program it can be confusing at times reading each line and then knowing what it does. To solve this program we can use // at the end of a line of programming. By using // it tells the Arduino software to stop reading and move onto the next line.

For example instead of just having…

```
void loop()
{
  digitalWrite(redledPin, HIGH);
  delay(10000);
  digitalWrite(redledPin, LOW);
```

we could say…

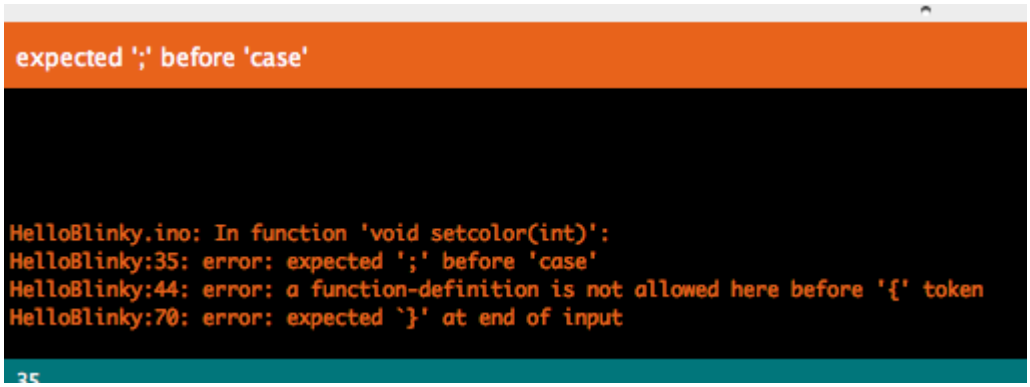
```
void loop()
{
  digitalWrite(redledPin, HIGH);          //  Red LED switched on
  delay(10000);                           // pause 10 seconds
  digitalWrite(redledPin, LOW);           // Red LED switched off
```

This lets us know that the LED will be switched on and this command will stay the same for 10 seconds. After this the LED will be switched off.

It is good practice to do this in case you make a mistake as your error will be seen easier.

Error Checking

As we are getting more complex in our commands it might be necessary to check our sketch to see if there are any errors within it.



As Im sure you have already noticed if you made a mistake in your syntax the Arduino program tells you (as above) and even tells you where the mistake is. For example above it says

HelloBlinky:35: error: expected ';' before 'case;

This says the error is in line 35 and what the problem is – in this case you have not put a ; at the end of the previous line.

Another thing we can do is check that the computer is understanding what we are asking it to do, and get it to tell us what it is doing. We do this with the Serial.Print Command.

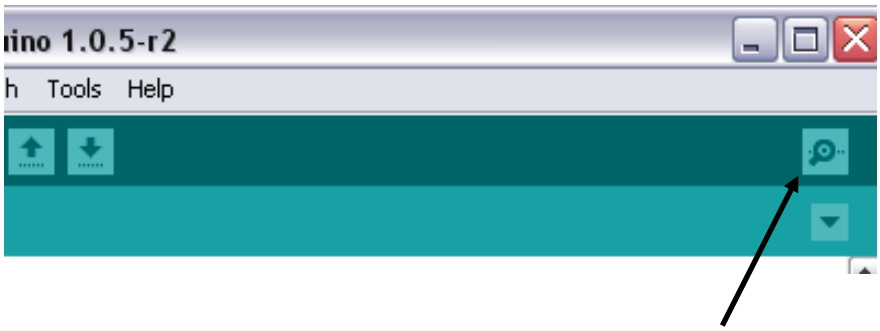
By putting ' Serial.begin(9600); ' in to our void setup() loop this tells the microcontroller to open the serial port. After this we can use the Serial.Print command within the void loop() after a command.

For example if we want to check an LED has went on we would put the serial.begin(9600); into our void setup and insert lines into our viod loop. For example to flash and LED instead of writing..

```
void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(2000);
  digitalWrite(ledPin, LOW);
  delay(5000);
}
```

You could write

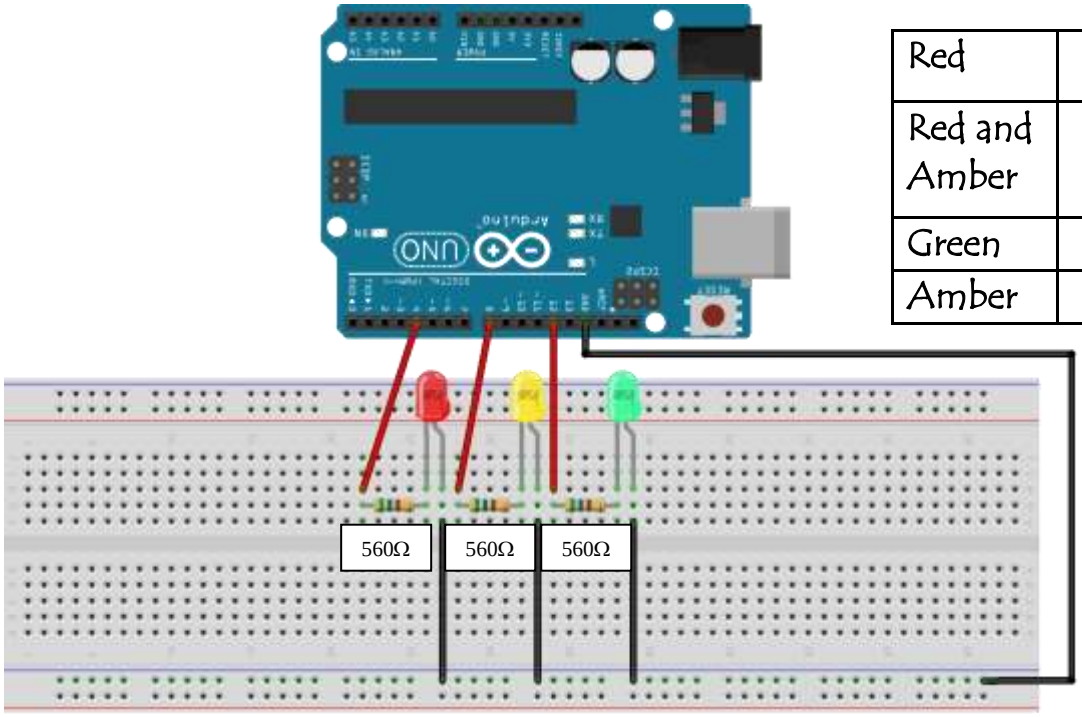
```
void loop()
{
  digitalWrite(ledPin, HIGH);
        serial.print("switch on LED")
  delay(2000);
digitalWrite(ledPin, LOW);
        serial.print("switch off LED")
  delay(5000);
}
```



When it has uploaded I can press the Serial monitor button, which opens a new dialogue box and "prints" what I put in the serial print boxes once it has completed

Task 8

A set of temporary traffic lights are required for a system of road works. Wire up the circuit as shown below, then write a program that could be used for the system. Use the times given in the table.



Red	12s
Red and Amber	5s
Green	15s
Amber	4s

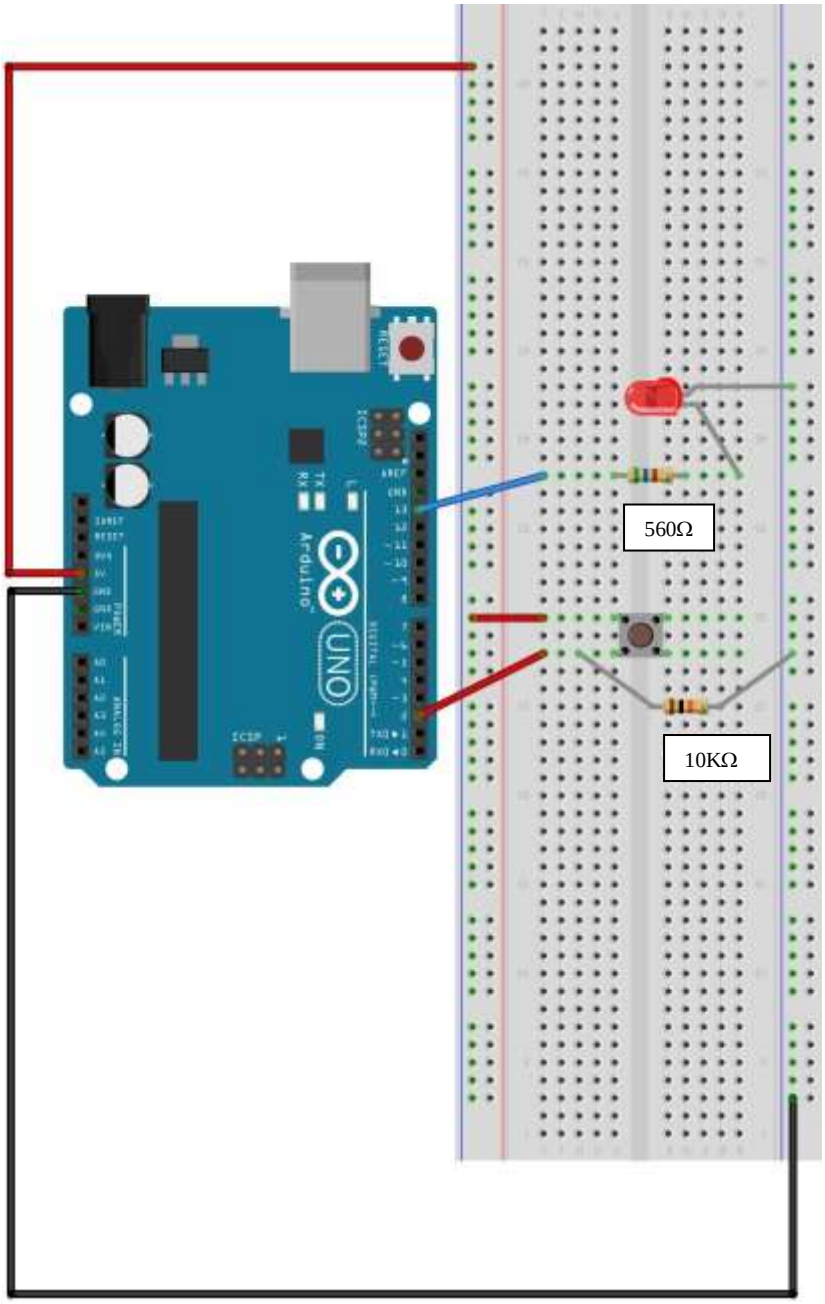
Alter the program so it includes serial print commands, so you can check your programming

Counter

It is often useful to repeat the same part of a program a number of times, for instance when flashing an LED. Instead of constantly pressing it, we could press it once and it would follow a sequence to flash a certain amount of times.

Task 9

a) Build up the circuit as shown below.

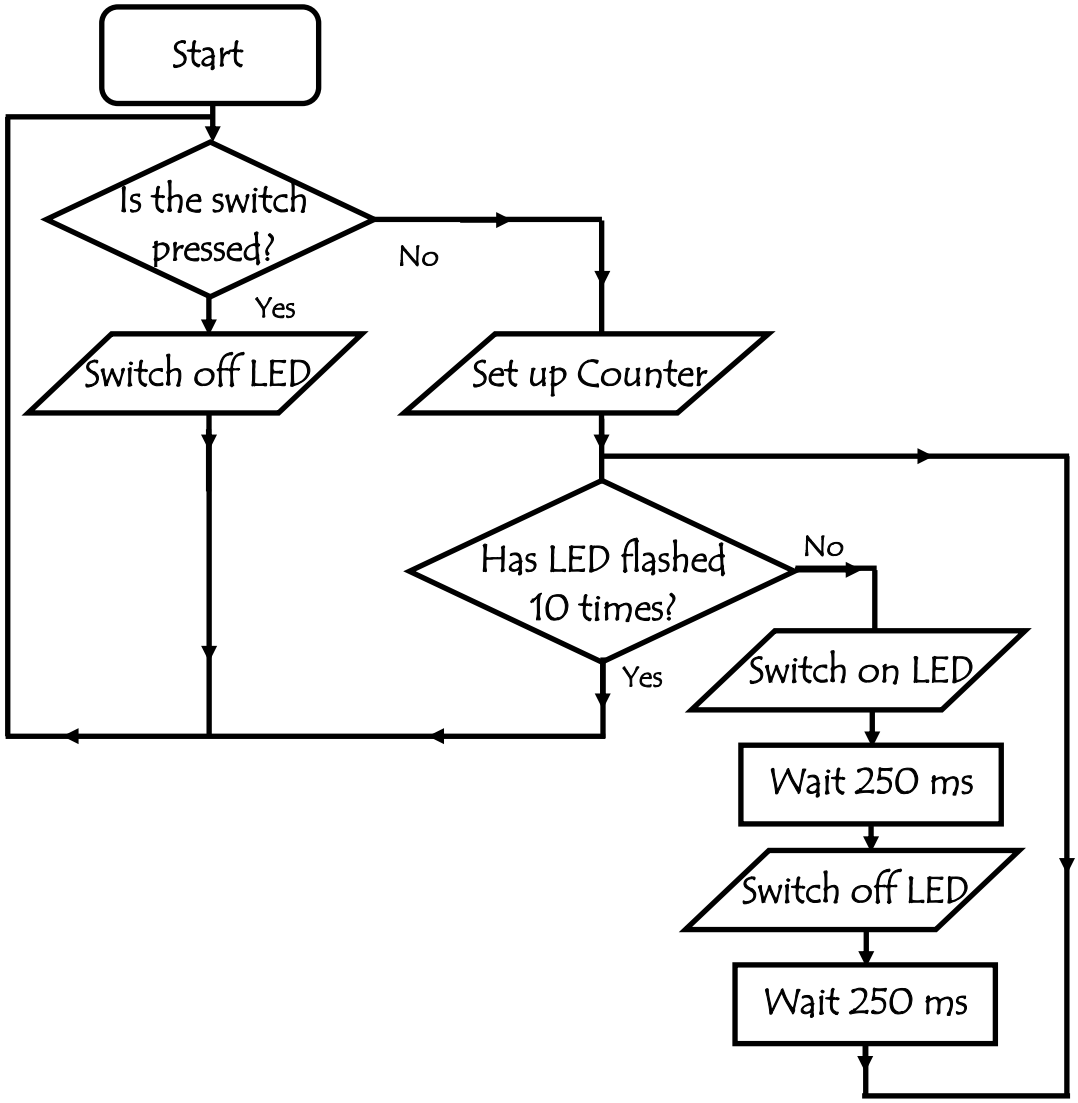


Task 9 (continued)

As you can see the switch is set up to pin 2, and the LED is set up to pin 13.

b) Write down how we would write **the beginning** of this program, assigning these pins to the components and telling the Arduino board if they are inputs or outputs.

As you can see in the flowchart we want the LED to flash 10 times when the button is pressed.



What we have to do first is set up the program and use the **if…else** command to see if the switch is pressed.

```
void loop()
{
  if (digitalRead(switchPin) == HIGH)
```

This 'reads' your circuit to see if the button has been pressed, or in other words, if switch pin = 1

Later we will add the else part of this command to see what will happen if it has not been pressed.

Our next block of code is setting up the counter as shown below.

{ for (int counter = 1 ; counter <= 10; counter = counter ++)

There are three separate statements in this command that are separated by a semicolon. The first statement is the initialization of the counter variable.

int counter = 1

This tells the Arduino board to start at 1 when completing counting. This is important as it gives the Board a starting point for the counter. This may seem obvious, but firstly a computer does not have common sense like we do – it only knows EXACTLY what we tell it. Secondly, it may have an old counter program still in the EEPROM so this makes sure it does not use that.

The next part of the command tells the computer the number of times it has to repeat this process. In this case it has to count to 10.

counter <= 10

The final command in this line of programming tells the computer how much to add to the value of the pin each time it repeats. In other words, every time it the program completes a loop add and additional loop until it reaches the previously set limit.

counter = counter ++

After this command is written the 10x flashing LED aspect of the program can be written. This has to be contained in { } brackets. The reason for this is that the Arduino reads the program one line at a time. Without the brackets to tell it that this is a block of code, it will only do the next line. If the brackets are there is recognises there is a block of code to read and will continue to complete the actions until the end of brackets.

```
void loop()
{
 if (digitalRead(switchPin) == HIGH)
   { for (int counter = 1 ; counter <= 10; counter = counter ++)
    {  digitalWrite(ledPin, HIGH);
       delay (250);
       digitalWrite( ledPin, LOW);
       delay (250);}

 }
```

We now have to add the else part of the command from our first decision box. According to our flowchart this switches off the LED and loops back to the beginning.

```
 else
   {
   digitalWrite( ledPin, LOW);
   }
}
```

Our finished program should now look like this.

```
/*
Counter Sketch
*/

int ledPin = 13;
int switchPin = 2;

void setup()
{
  pinMode(ledPin, OUTPUT);    // initialize the LED pin as an output
  pinMode(switchPin, INPUT);  // initialize the pushbutton pin as an input
}

void loop()
{
  if (digitalRead(switchPin) == HIGH)
    { for (int counter = 1 ; counter <= 10; counter = counter ++)
      {  digitalWrite(ledPin, HIGH);
         delay (250);
         digitalWrite( ledPin, LOW);
         delay (250);}
  }

  else
    {
    digitalWrite( ledPin, LOW);
    }
}
```

Upload your completed sketch to the Arduino and see if it works.

Labels

Another way we could have written this sketch is by using labels. This breaks the program into branches so that it is easier to understand.

```
/*
Counter Sketch
*/

int ledPin = 13;
int switchPin = 2;

void setup()
{
  pinMode(ledPin, OUTPUT);    // initialize the LED pin as an output
  pinMode(switchPin, INPUT);  // initialize the pushbutton pin as an input
}

void loop()
{
MAIN_LABEL;                   //This is a label I can now refer to in my program

  if (digitalRead(switchPin) == LOW)
         {
         goto MAIN_LABEL;          //This will create a loop that will 'goto' the
                                   //label Ive created until the input condition
                                   //changes
         }

    for (int counter = 1 ; counter <= 10; counter = counter ++)
     {  digitalWrite(ledPin, HIGH);
        delay (250);
        digitalWrite( ledPin, LOW);
        delay (250);
  }

  else
    {
    digitalWrite( ledPin, LOW);
    }
}
```

Using Labels and the "goto" command within our coding it can simplify certain programs and make it easier to understand.

Task 10

As part of a Christmas decoration, a lighting sequence is to be controlled by a microcontroller. The output connections are shown below.

Green LED 1: Pin 1
Green LED 2: Pin 13
Red LED 1: Pin 2
Red LED 2: Pin 12
Yellow LED: Pin 5

The red and green lights should come on together and stay on for 5 seconds. Then they both go off and the yellow light should come on for 8 seconds. The Yellow light should then go off and the red light should flashes on and off 6 times (the 'on' and 'off' times being 0.5 seconds each). The sequence then repeats itself.

a) Draw a flowchart to represent this programme.

Task 10 (continued)

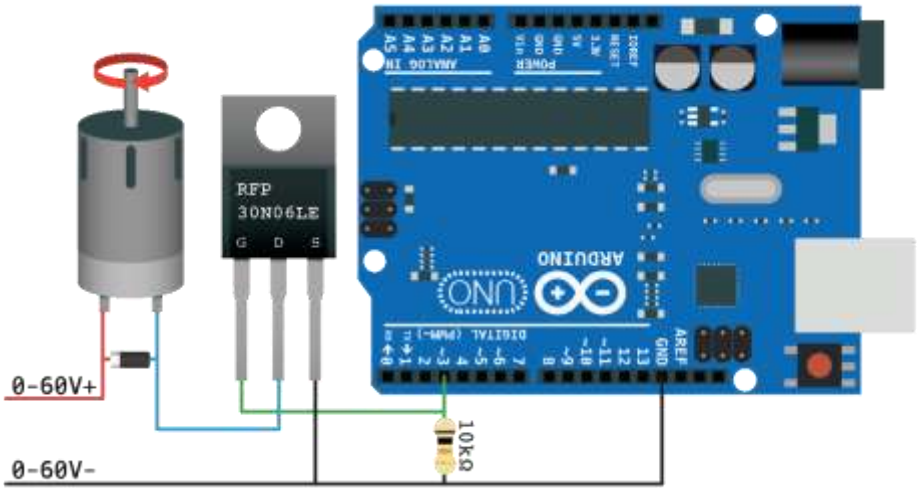
b) Write a program for this sequence. Insert serial print commands to check your programming at appropriate places. Write the program below.

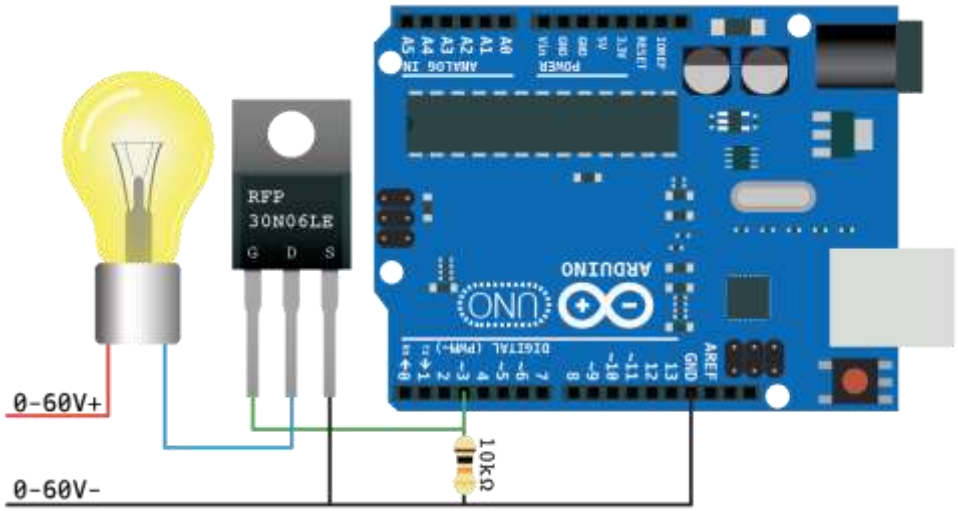
c) Take a photo of you model and stick below

High Power Control

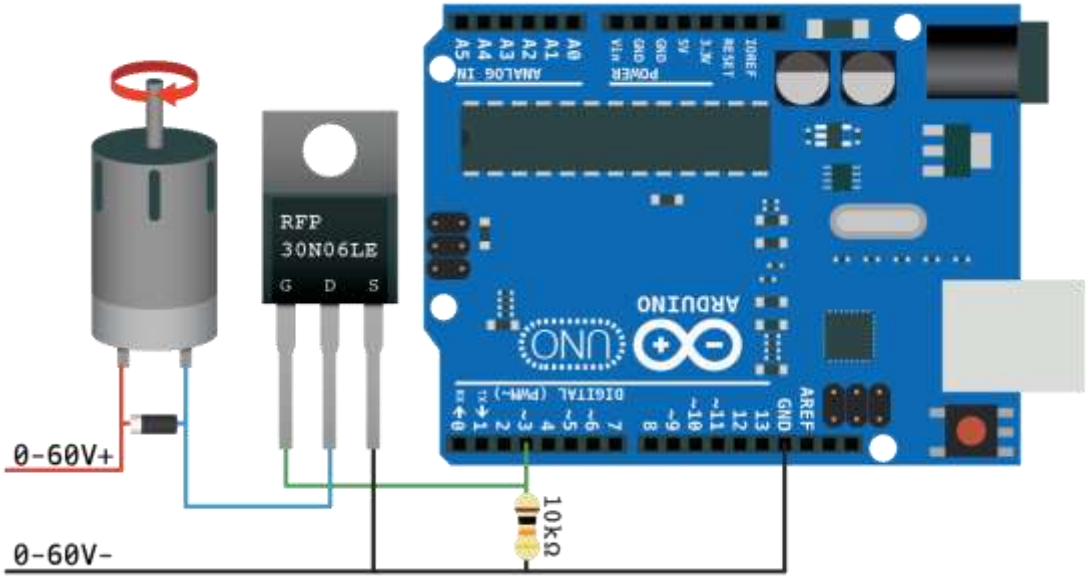
As you know the Arduino board is powered by a 5v supply by a USB port, but what if we want to power something more powerful like a motor? We can do this by using a MOSFET. As you know a MOSFET is essentially like a transistor. It has 3 legs - you have an **In** called the **Source**, an **Out** called the **Drain**, and a **Control** called the **Gate**. When you send a HIGH signal to the gate (control pin), the transistor switches and allows current to flow from the source (in) to the drain (out).

So, we can connect it so that our high powered device is connected to V+ but not ground (V-). Ground is connected to the transistor's drain. When our Arduino sends a HIGH signal to the transistor's gate, it switches the transistor (connecting the drain and source) and completes the circuit for the motor, solenoid, or light.





This circuit is pretty simple. The only part that looks funny is the resistor. This is a pull-down resistor. The resistor holds the gate low when the Arduino does not send a high signal. This is here in case the Arduino comes loose, or the wiring is bad it will default to off. You don't want this pin to ever be floating as it will trigger on and off.

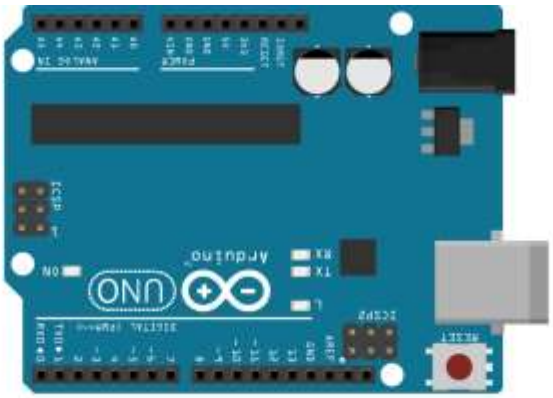


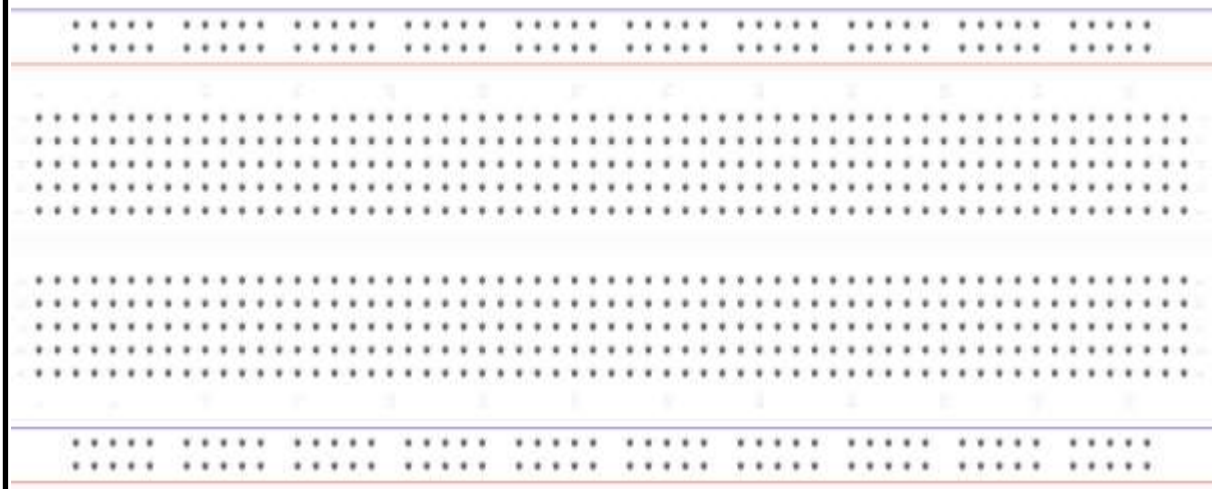
You can see that in the motor diagram, there is a diode parallel to the motor. Any time you are powering a device with a coil, such as a relay, solenoid, or motor, you need this. When you stop powering the coil, a reverse voltage (EMF), up to several hundred volts, spikes back. This only lasts a few microseconds, but it is enough to kill our MOSFET. So this diode (only allows current to pass one way) is normally facing the wrong direction and does nothing. But when that voltage spikes comes flowing the opposite direction, the diode allows it to flow back to the coil and not the transistor.

Just make sure that protection diode is facing the correct way (stripe facing the V+ of device). If it is facing the wrong direction, the device you are trying to power will not work as the diode will just allow the current to bypass it.

Task 11

a) Plan out how you would build this circuit using the breadboard below. Connect the circuit to pin 9.



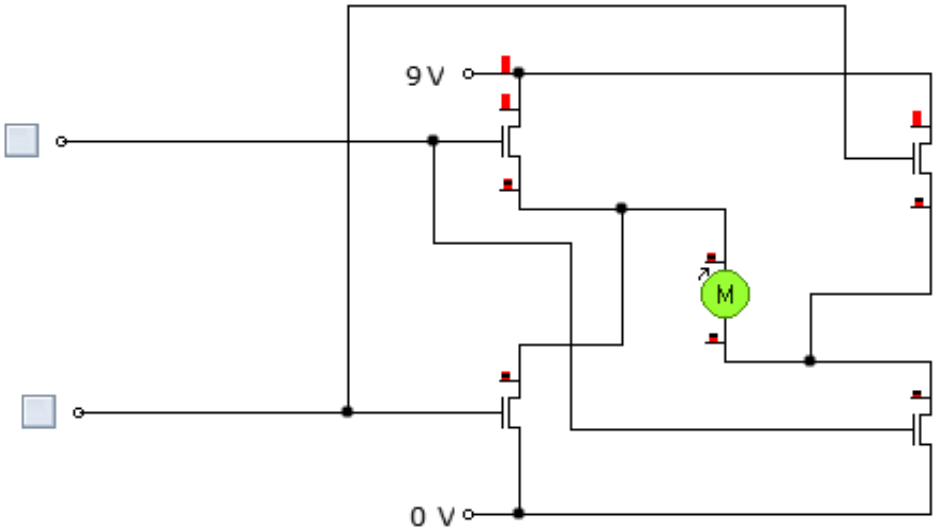


b) Build and test this using the Arduino kits. Make a simple program that switches the pin on for 5 seconds then switches off for 10.
Attach a photo below.

Motor Control

Within the National 5 course we used a motor as one of our outputs, and this was controlled using an H-Bridge chip. This allowed us to change the direction of the motor by essentially changing the polarity of the voltage running through the motor.

We are now going to control it the same way, but instead of using the H bridge chip, we are going to build an H bridge using MOSFETs. As you know a MOSFET can be used as a switch and to do this, you have to have its gate voltage (Vgs) higher than the source. If you connect the gate to the source (Vgs=0) it is turned off.



By using 4 different MOSFETs we can change the direction of the motor. By making one switch high, it makes the motor turn in one direction. If we change it so the other switch is high, the motor will turn in the other direction.

Task 12

a) Using Yenka, build and test this circuit. There is a component missing from this circuit – what is it?

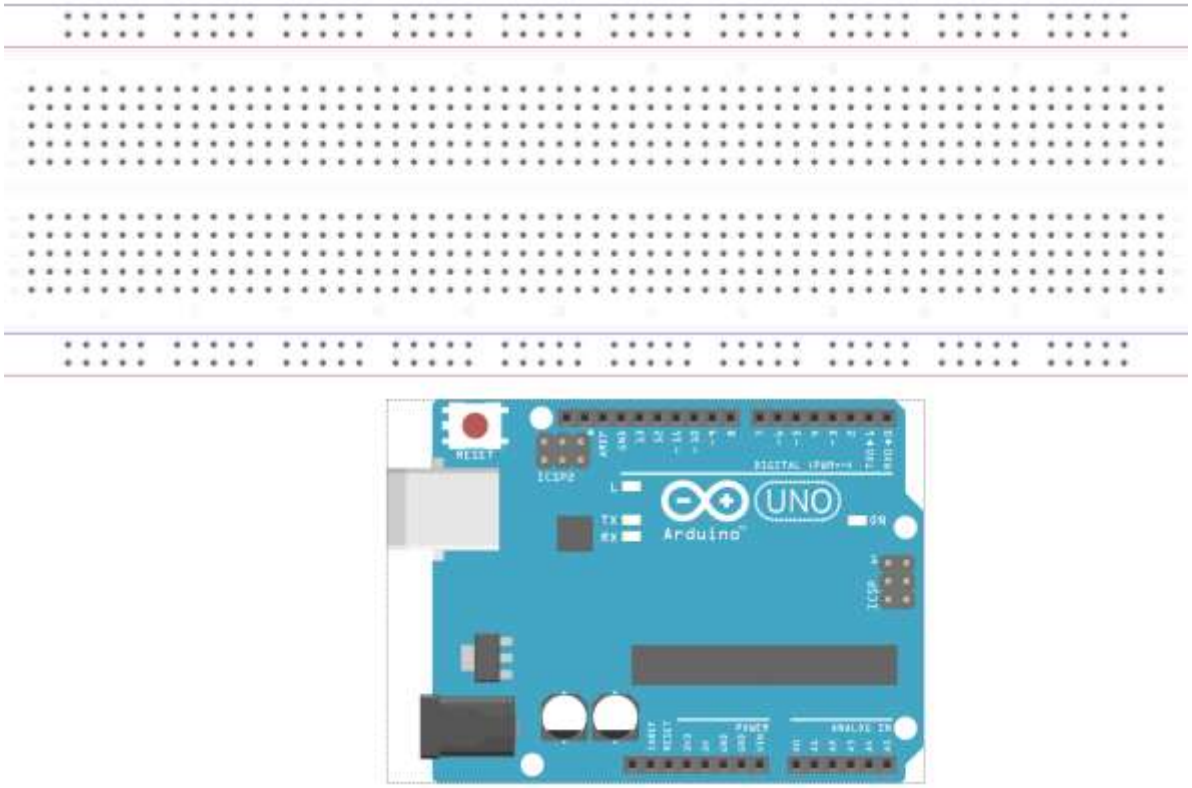
Add this component.

b) Alter this circuit by connecting it to a PIC microcontroller. Build a flowchart and circuit that will rotate it one way for 5 seconds. Then rotate it in the other direction for 5 seconds.

Print your circuit and glue below.

Task 12 (continued)

c) Using the breadboard below, plan out how you would wire this up to your Arduino board.



d) Take a photo of your completed circuit and attach below.

Task 13

Design a program that could be used to power the drive control of a remote control boat. When one switch is pressed the boat drives forward, when another is pressed it will reverse.

a) Draw the flowchart for this below

Task 13 (Continued)

b) Write the program for this below

c) Simulate this at http://123d.circuits.io/. Once you have got it working attach an image of your circuit below.

Defining Commands

When you are going to do something in our sketch that involves a few lines of instructions and it will be used several times throughout a program a useful command to use is #DEFINE.

For example when rotating a motor instead of constantly repeating

```
digitalWrite(motor1APin, LOW);
digitalWrite(motor2APin, HIGH);
```

within my beginning initialisation I could write

```
#DEFINE MOTOR_LEFT digitalWrite (Motor1APin, LOW);
                                digitalWrite(Motor2APin, HIGH)
```

And then within my void Loop Command I would just have to write the command

```
MOTOR_LEFT
```

Task 14

Change your program for the Remote control boat to involve define commands.

The While Command

Our program for the remote control car still does not work as well as it could do, because the commands only work when we have our finger on the button – as soon as it's released it stops that command.

The way to resolve this is using the While command. The **While command created loops within the program that** will loop continuously, and infinitely, until the command inside its brackets () becomes false.

For example, within my program I may have set up a define command that will turn my car left.

```
MOTOR_LEFT;                 //right  motor turns anticlockwise to make car drive left
   while (!END1_HIT)        // wait here while switch is NOT pressed
                            // a ! inverts it
         {
          delay(10);         // do nothing, just delay a very tiny amount
         }
```

There is also a delay of 10ms inserted after this – this command does nothing except create a slight pause between it changing directions with a new command. This is good practice to stop it damaging the motor.

Task 15

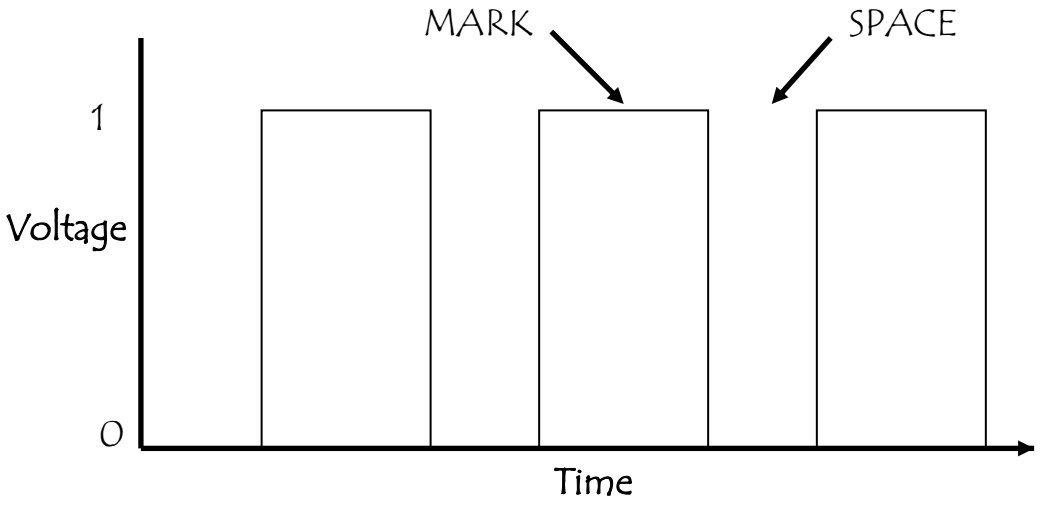
Change your program for the Remote control boat to involve the while commands.

Speed Control of D.C. Motors

There are two ways to control the speed of a D.C. motor. The simplest is to vary the voltage applied to the motor. If, for instance, 3 V is applied to a small D.C. motor it will rotate at a lower speed than if 5 V were applied. Unfortunately the 'turning power' (torque) of the motor will also drop, which means the whole motor system will be less powerful.

The second way to control the motor is to always apply the full voltage across the motor, but then to switch the power supply on and off rapidly. As the power supply is off some of the time, the motor does not receive as much power and so the motor turns more slowly. The advantage of this system is that the torque remains quite high.

This system is called **Pulse-Width Modulation** (PWM). The time that the power supply is switched on is called the *mark* time, and the time that the motor is switched off is called the *space* time. By varying the on (mark)-to-off (space) ratio, the speed of the motor can be varied.



Pulse-Width Modulation Programming

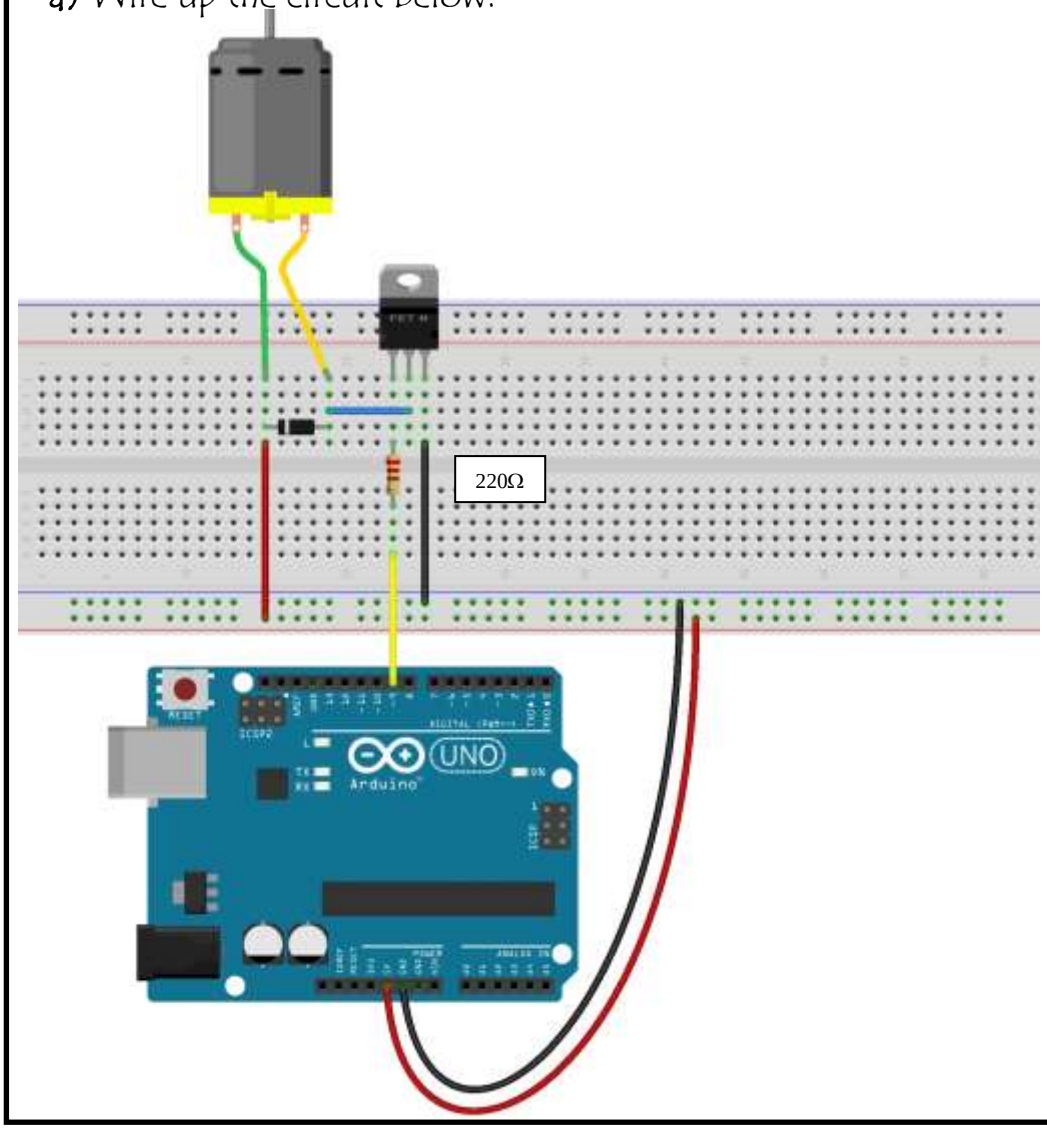
As you know Pulse Width Modulation is a technique for getting analogue results by digital means.

PWM Method 1

The first one is by switching the motor on, putting in the wanted delay, switching it off and once again putting in a delay. This would then be repeated for however long it is needed. It should be noted that anytime we are using Pulse Width Modulation we should be using the pins marked with a ~.

Task 16

a) Wire up the circuit below.



Task 16 (Continued)

b) Upload this program into you Arduino board and see what happens.

```
/* PWM program */

int LED = 9; // the pin for the LED

void setup()
{
pinMode(LED, OUTPUT); // tell Arduino LED is an output
}

void loop()
{
  digitalWrite(LED, HIGH);
  delay(100);
  digitalWrite(LED, LOW);
  delay(100);
}
```

The motor should switch on and turn at full speed, then stop.

c) (i) Change the first delay to 25, the second to 75 and observe the speed. What happens?

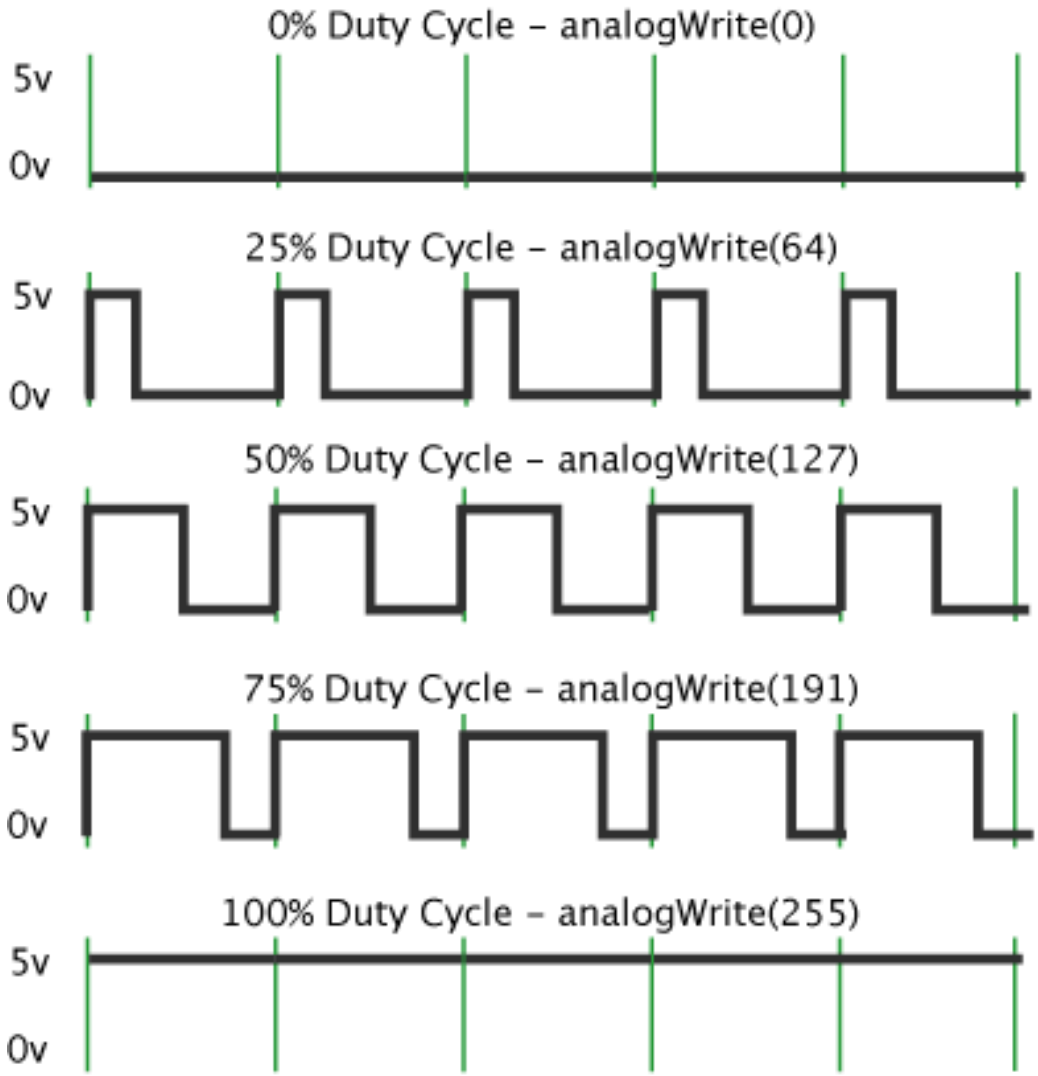
(ii) Change the first delay to 50, the second to 50 and observe the speed. What happens?

(iii) Change the first delay to 75, the second to 25 and observe the speed. What happens?

PWM Method 2

The second is by using the command **analogWrite()** will allow you to create the speed you need, using it on a scale of 0 – 255. We use these numbers because a byte is a unit of storage in a computer, and this contains 8-bits and can store 256 different values: 0 to 255.

This means that analogWrite(255) would request a 100% duty cycle (meaning its always on), and analogWrite(64) is a 25% duty cycle (on half the time). Using maths we can get the motor to go at any speed we like.



Task 17

Keep the same circuit you used in task 16.

a) Upload this program into you Arduino board and see what happens.

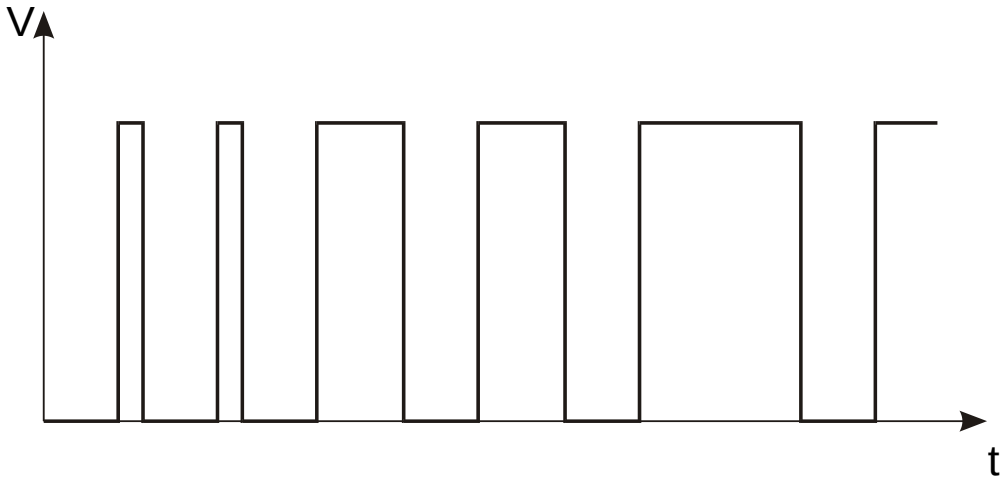
```
/* PWM program */

int MOTOR = 9; // the pin for the LED

void setup()
{
pinMode(MOTOR,OUTPUT); // tell Arduino LED is an output
}

void loop()
{
 analogWrite(MOTOR, 255);  // Sets the PWM mark:space ratio.

}
```

The Motor should turn at full speed.

b) (i) Change the delay to 127 and observe the speed. What happens compared to the full speed? Draw the mark space ratio graph for this.

(ii) Change the delay to 64 and observe the speed. What happens compared to the full speed? Draw the mark space ratio graph for this.

Soft Start of DC Motors



In some devices, such as electric drills, it is desirable for the motor to start rotating slowly and then build up speed, rather than rapidly 'accelerating' up to full speed. This is called **'soft starting'** the motor, and the use of PWM is often appropriate in these situations. The motor is started at a low speed and then gradually accelerated by varying the mark to space ratio over a period of time.

We do this to help protect the motor and any mechanical aspects that would be attached to it.

Task 18

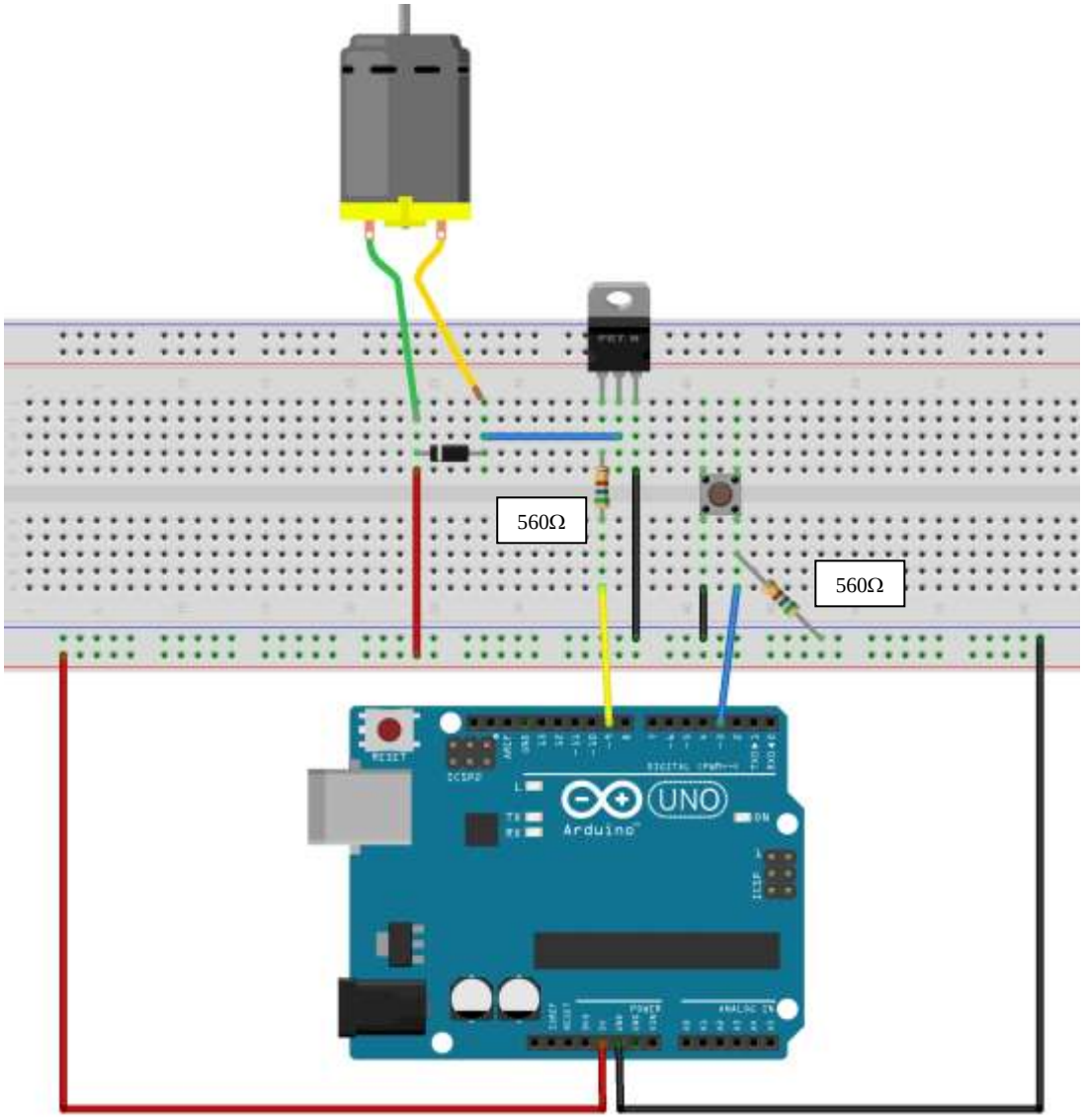
Keep the same circuit you have been using in task 16 and 17, but add a switch to pin 13.

a) Upload this program into you Arduino board.

```
/* Soft Start Control. */

int switchPin = 13;
int motor = 9;

void setup()
{
pinMode(switchPin, INPUT);
pinMode(motor, OUTPUT);
}

void loop()
{
START:
if (digitalRead(switchPin) == HIGH)
                {
                analogWrite(motor, 64); // sets speed to ratio 1:4
                delay(100);              //slight delay in between each step
                analogWrite(motor, 127); // sets speed to ratio 1:2
                delay(100);
                analogWrite(motor, 191); // sets speed to ratio 3:4
                delay(100);
                analogWrite(motor, 255); // sets speed to ratio 1:1
                goto START;
                }
}
```

What happens?

Task 19



a) Create this circuit using our breadboard and Arduino kit

Task 19 (Continued)

b) Upload this program into you Arduino board and see what happens

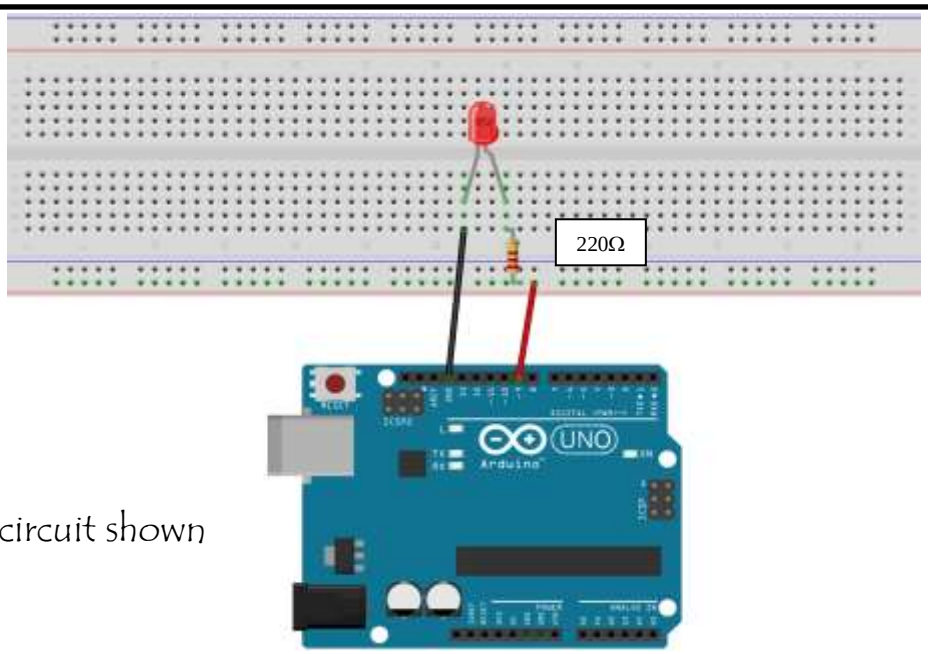
```
/* Soft Start Control. */

int switchPin = 3;
int motor = 9;


void setup()
{
pinMode(switchPin, INPUT);
pinMode(motor, OUTPUT);

}

void loop()
{
START;
        if (digitalRead(switchPin) == HIGH)
                {
                analogWrite(motor, 64); // sets speed to ratio 1:4
                delay(100)
                analogWrite(motor, 127); // sets speed to ratio 1:2
                delay(100)
                analogWrite(motor, 191); // sets speed to ratio 3:4
                delay(100)
                analogWrite(motor, 255); // sets speed to ratio 1:1
                goto START;
                }
}
```

Describe what happens with this program

Fading

Although PWM is mainly used for motor control it can be used for other output devices. We can even use counters to increase the output levels by set values.

Task 20



a) Build the circuit shown

b) Write this program into the arduino and test

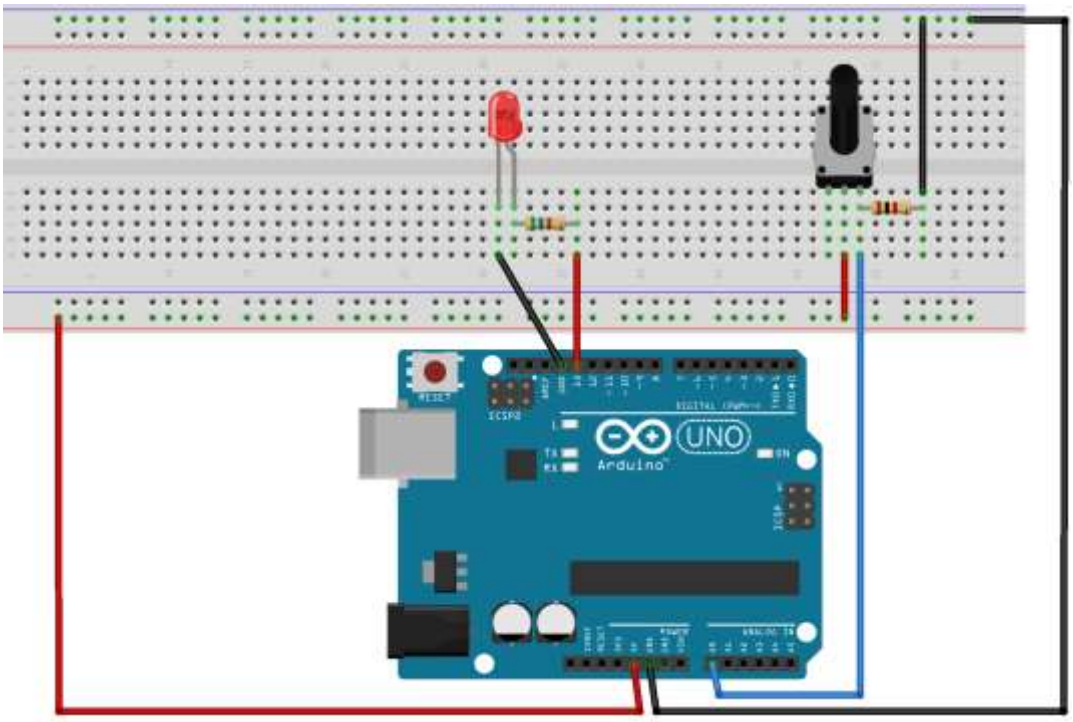
```
*/Fade Value/*
int ledPin = 9;                  // LED connected to digital pin 9

void setup()
{
pinMode(ledPin, OUTPUT);
}

void loop()
{
  for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5)
                                   // fade in from min to max in increments of 5 points
                                   // sets the value (range from 0 to 255)
          {
           analogWrite(ledPin, fadeValue);
           delay(30);              // wait for 30 milliseconds to see the dimming effect
            }

  for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5)
                                   // fade out from max to min in increments of 5 points:
          {
           analogWrite(ledPin, fadeValue);
             delay(30);             // wait for 30 milliseconds to see the dimming effect
           }
}
```

Analogue Input

The Arduino board can also be used to read inputs that are analogue, and this is done by the **analogRead** command.

Task 21

a) Build the voltage divider circuit with a potentiometer as shown



b) Write this program into the arduino and test

```
*/ Analogue reading/*

int sensorPin = A0;    // select the analogue input pin 0 for the potentiometer
int ledPin = 13;      // select  pin 13 for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  sensorValue = analogRead(sensorPin);
  digitalWrite(ledPin, HIGH);            // turn the ledPin on
  delay(sensorValue);              // stop the program for <sensorValue> milliseconds
  digitalWrite(ledPin, LOW);              // turn the ledPin off:
  delay(sensorValue);             // stop the program for for <sensorValue> milliseconds
}
```

Task 21 (continued)

c) Experiment with the analogRead command. Design a circuit that will fade an LED dependant on the incoming light level.

Draw a circuit diagram for this circuit below.

d) Write the program you have created below