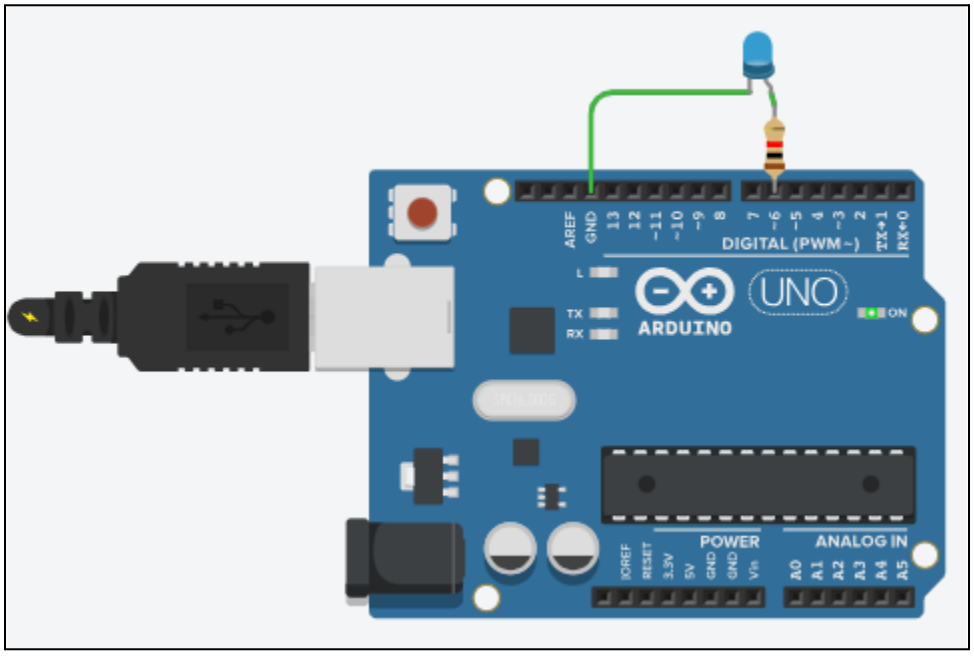
| | |
|---|---|
| **Program No:** | **02** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Introduction to Basic IoT Components** |
| **Objective :** | **Write a program to demonstrate fading effect of LED** |

**Source Code:**

```
// C++ code
//
int led = 6;
int brightness = 255;
int fadeamt = 100;
void setup()
{
  pinMode(led, OUTPUT);
}

void loop()

{
  analogWrite(led,brightness);
  brightness = brightness - fadeamt;
  if (brightness <=0  || brightness >= 255)
    fadeamt = -fadeamt;
 delay(2000);

}
```

**Output:**


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

| | |
|---|---|
| **Program No:** | **03** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Introduction to Basic IoT Components** |
| **Objective :** | **Write a program to generate different colors using rgb components interface with Arduino** |

**Theory:**

setup() - used to initialise variables, pinmodes, etc. called when a sketch starts , it will only run once, after each power up or the reset of the arduino board.

loop() - after creating a setup() function, which initialises and sets the initial values, the loop() function does exactly what its name suggests , and loops consecutively, allowing your program to change and respond

pinMode() - the pinMode() function is used to config a specific pin to behave either as an input or an output

Delay() - pauses the program for the amount of time specified as parameter

analogWrite() - Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

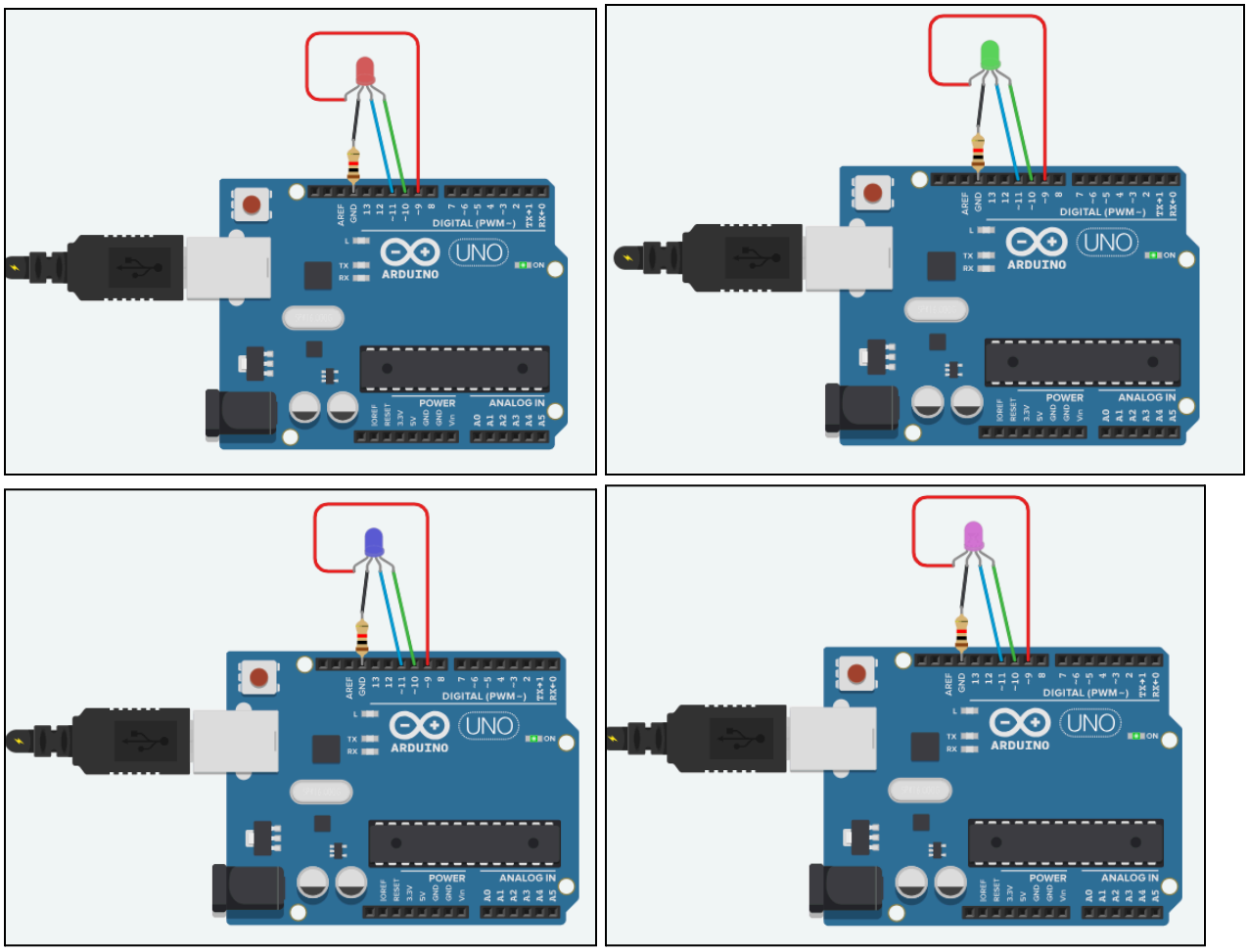
rbg() - RGB is also the term referring to a type of component video signal used in the video electronics industry. It consists of three signals—red, green, and blue—carried on three separate cables/pins

**Source Code:**

```
// C++ code
//
int red = 9;
int green = 10;
int blue = 11;
void setup()
{
```

```
  // pin are in order (9,10,11) therefore for loop is used
  for (int i=9; i<12; i++) {
        pinMode(i,OUTPUT);
  }
}

void loop()
{
  rgb(255,0,0);       //red
  delay(1000);
  rgb(0,255,0);       //green
  delay(1000);
  rgb(0,0,255);       // blue
  delay(1000);
}

void rgb(int redVal, int greenVal, int blueVal)
{
  analogWrite(red, redVal);
  analogWrite(green, greenVal);
  analogWrite(blue, blueVal);
}
```

## Output:

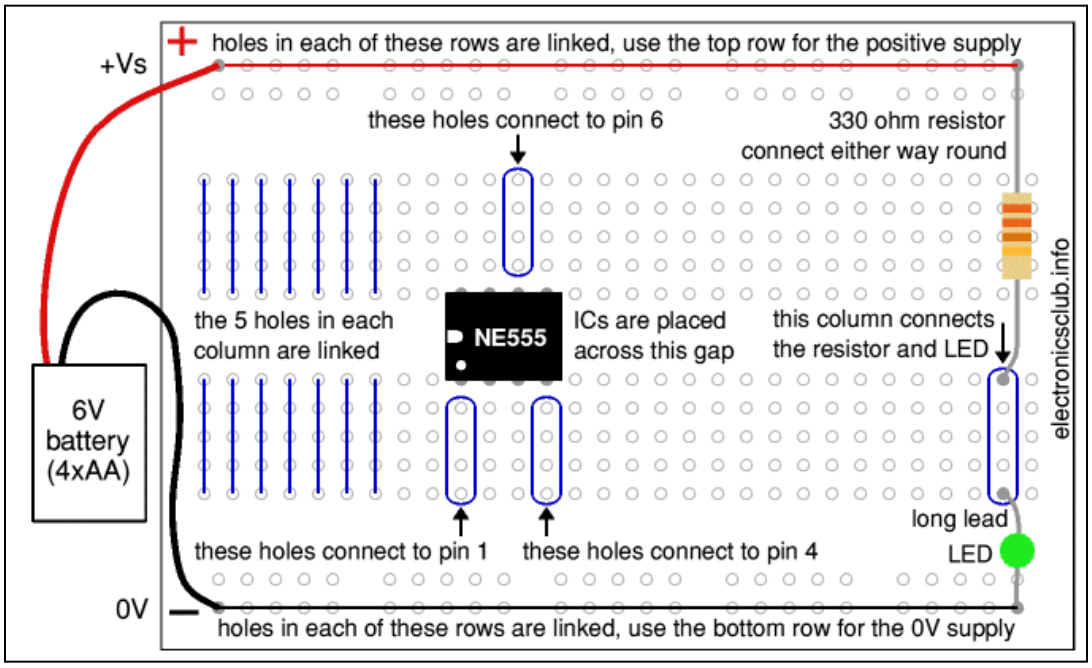


| | |
|---|---|
| **Program No:** | **4** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Using breadboard** |
| **Objective :** | **Interface six LED's using breadboard to Arduino and glow them in forward and bckward direction** |

breadboard - A breadboard provides an easy way to create electronic circuits. It is a plastic board with lines of holes for holding either electronic components or connecting wires (jumper wires).

Breadboards have many tiny sockets (called 'holes') arranged on a 0.1" grid. The leads of most components can be pushed straight into the holes. ICs are inserted across the central gap with their notch or dot to the left.

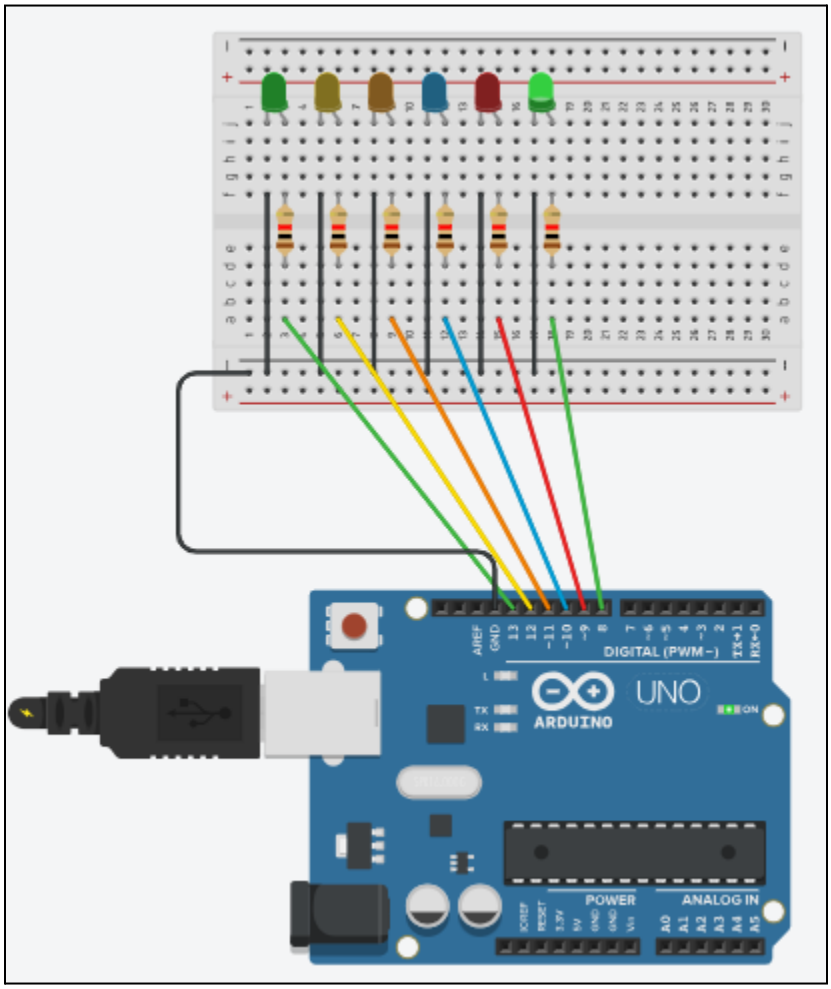
Wire links can be made with single-core plastic-coated wire of 0.6mm diameter (the standard size), this is known as 1/0.6mm wire. I suggest buying a pack with several colours to help identify connections, red for +Vs wires, black for 0V, and so on.




**Source Code:**

```
// C++ code
//
void setup()
{
  for(int i=8; i<=13; i++) {
        pinMode(i, OUTPUT);
  }
}

void loop()
{
  for(int i=13; i>=8; i--) {
        digitalWrite(i, HIGH);
    delay(500);
    digitalWrite(i, LOW);
    delay(500);
  }

  for(int i=8; i<=13; i++) {
        digitalWrite(i, HIGH);
    delay(500);
    digitalWrite(i, LOW);
    delay(500);
  }
}
```

**Output:**


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN

| | |
|---|---|
| **Program No:** | **5** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Using push button and piezo buzzor** |
| **Objective :** | **Interface pushbutton and buzzer with Arduino** |

**push button** - Push buttons allow us to power the circuit or make any particular connection only when we press the button. Simply, it makes the circuit connected when pressed and breaks when released. In this kit we've got one to use during building projects for enhancing our knowledge. It is a very useful IoT hardware component.

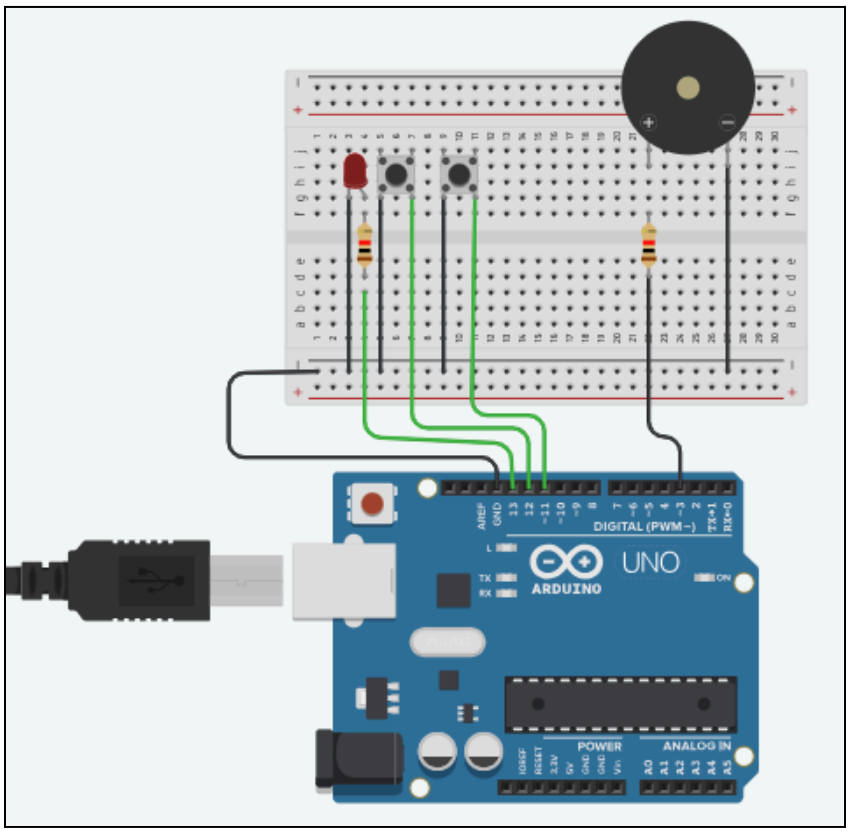
**piezo buzzer** - A piezo buzzer is a compact electronic device that produces audible alerts or notifications. It operates based on the piezoelectric effect, which allows certain materials to generate an electric charge under mechanical stress

**Source Code:**

```
const int button1 = 11;
const int button2 = 12;
const int buzzer = 3;
const int led = 13;
int buttonstate1 =0;
int buttonstate2=0;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer,OUTPUT);
  pinMode(button1,INPUT_PULLUP);
  pinMode(button2,INPUT_PULLUP);
  noTone(buzzer);
}
void loop()
{
  buttonstate1 = digitalRead(button1);
  buttonstate2 = digitalRead(button2);
  if(buttonstate1 == LOW)
```

```
 {
        digitalWrite(led, HIGH);
        delay(2000); // Wait for 1000 millisecond(s)
        digitalWrite(led, LOW);
        delay(1000); // Wait for 1000 millisecond(s)
 }
 if(buttonstate2==LOW)
 {
        tone(buzzer,500);
   delay(2000);
   noTone(buzzer);
   delay(2000); // Wait for 1000 millisecond(s)
 }
}
```

**Output:**



| | |
|---|---|
| **Program No:** | **6** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Slide switch** |
| **Objective :** | **Interface 2 LEDs and a slide switch with Arduino and glow respective LED on slide of switch** |

**What is Slide Switch?**

Definition of a slide switch is: It is a mechanical switch that is used to control the flow of current in a circuit by sliding the slider from the OFF (open) position to the ON (close) position known as a slide switch. This switch simply controls the current within a circuit without cutting a wire manually. These switches will stay in one position until changed into another position manually. The slide switch symbol is shown below.

**How Does Slide Switch Work?**

Slide switches work by using a slider to move from the OFF position to the ON position. They control the flow of current within a circuit in small projects. These switches are designed in two ways by using a metal slide and a metal seesaw.

The metal slides are used in most common designs that make contact through the flat metal parts on the switch. When the slider on the switch is moved, then metal slide contacts can move from one set of metal contacts to the other to activate the switch.

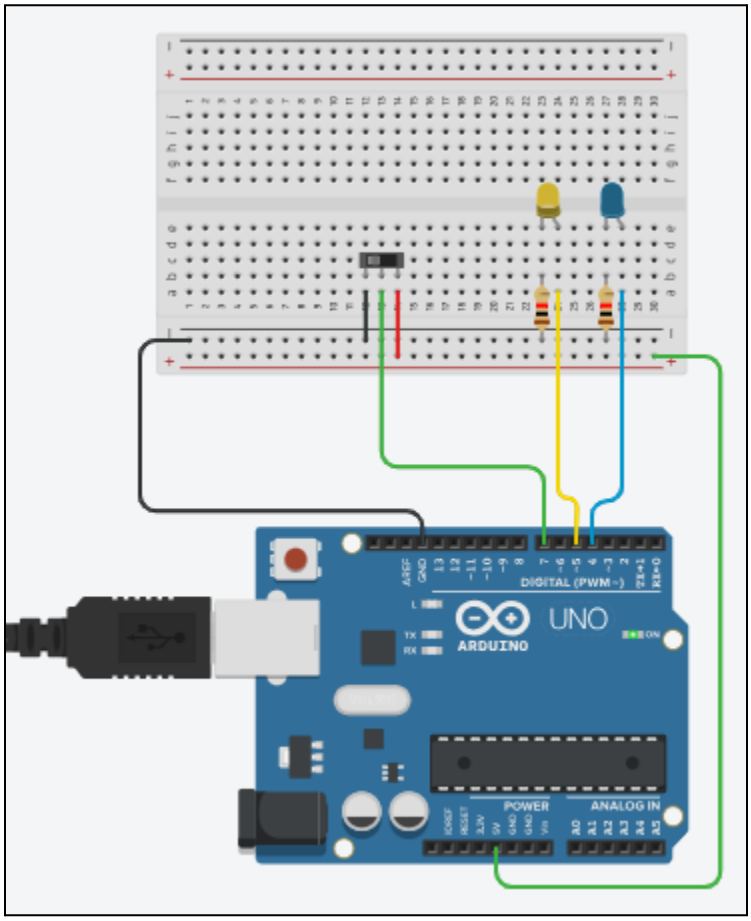
Similarly, the secondary design utilizes a metal seesaw. The slider of this switch includes a spring that pushes down on one face of the metal seesaw otherwise the other. These are maintained-contact switches, so they wait in one state until activated into the latest state and after that stay in that state until performed upon again.

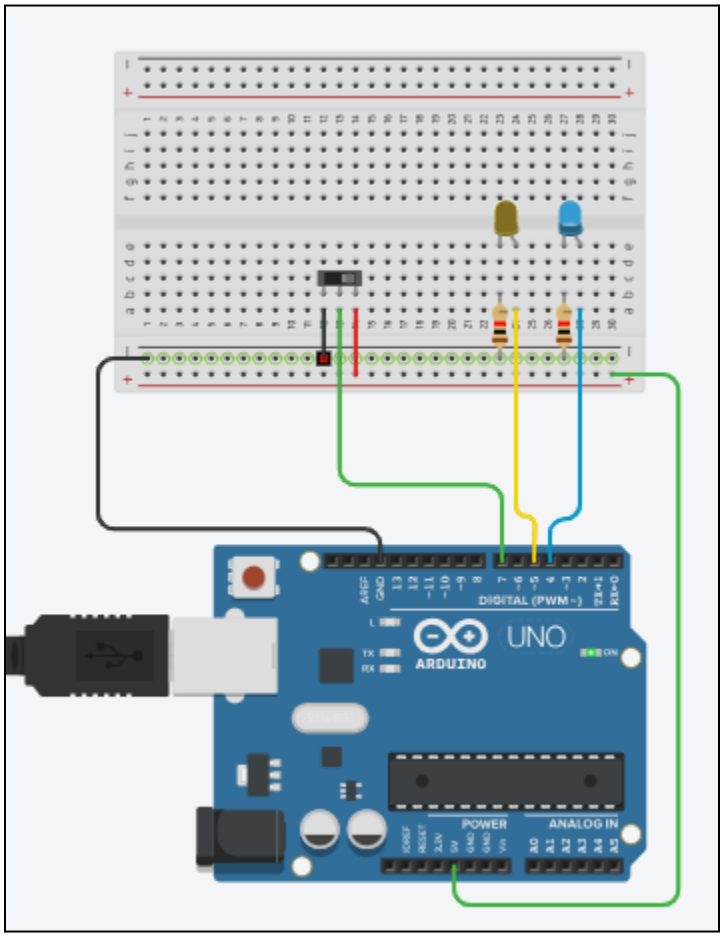
**Source Code:**

```
int butt = 7;
int led1 = 4;
int led2 = 5;
```

```
int reading ;
void setup()
{
  pinMode (led1, OUTPUT);
  pinMode (led2, OUTPUT);
  pinMode (butt, INPUT);
}

void loop()
{
  reading = digitalRead (butt);
  if (reading ==1)
  {
    digitalWrite (led1, HIGH);
    delay(500);
    digitalWrite (led2, LOW);
  }
  else
  {
    digitalWrite (led2, HIGH);
    delay(1000);
    digitalWrite (led1, LOW);
  }
}
```

## Output:

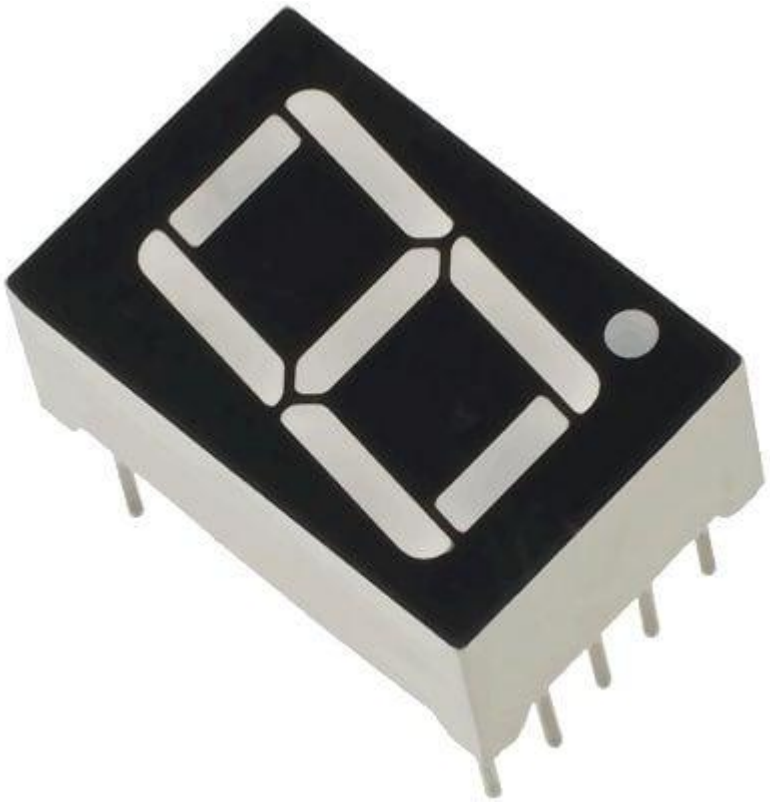



DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN

| | |
|---|---|
| **Program No:** | **7a & 7b** |
| **Roll No :** | **1545** |
| **Title of Program :** | **7 segment display** |
| **Objective :** | **Interface Seven Segment Display with Arduino and<br>(a) To blink it<br>(b) to build a step up counter and step down counter** |

## Theory:

**7-segment LED displays are formed by LED segments. It is basically used to display numerical values from 0 to 9. One more segment is also present there which is used as decimal point.**



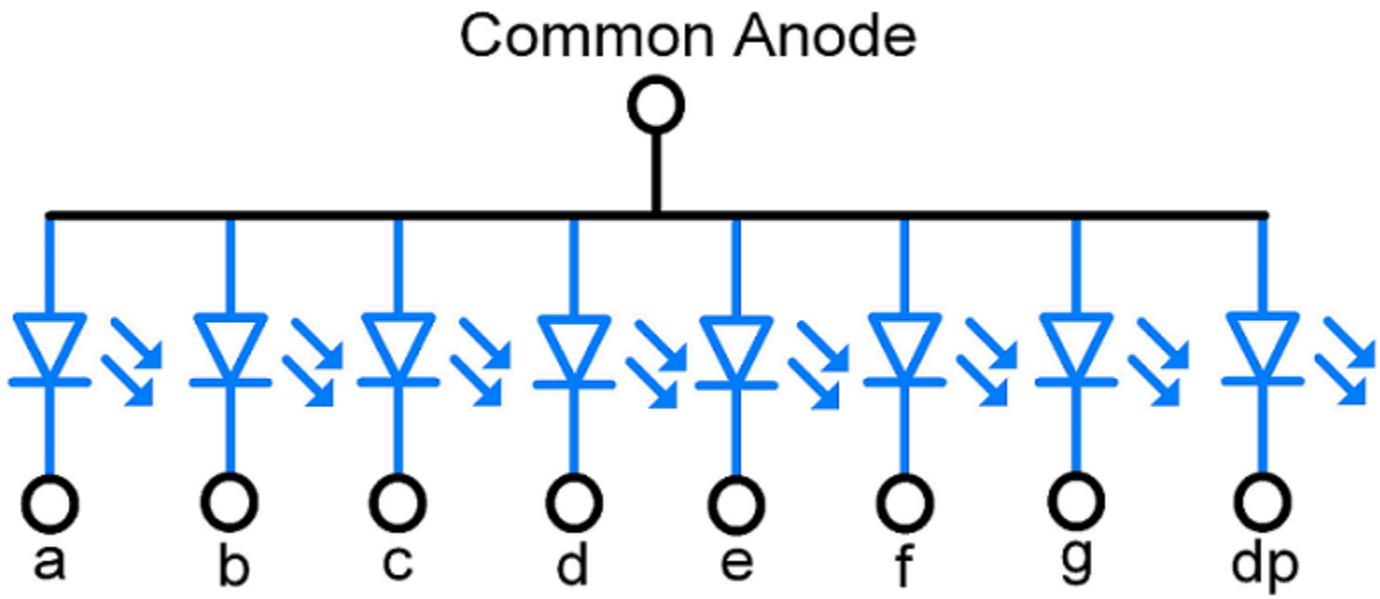
**7-Segment Display**

**In 7-segment displays there are two types, common anode and common cathode.**

**1. Common Anode (CA)**

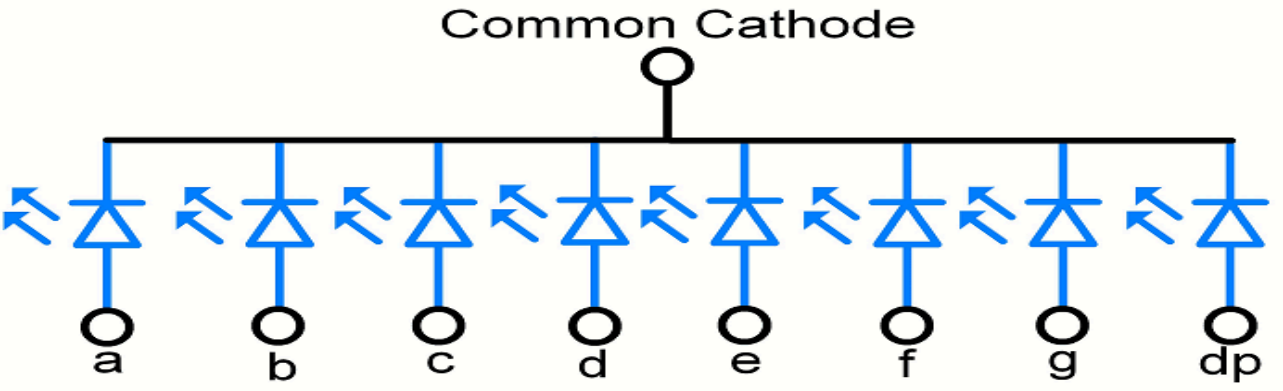
**In common anode display, all anode pins are connected together to VCC and LEDs are controlled via cathode terminals. It means to turn ON LED (segment), we have to make that cathode pin**

**logic LOW or Ground.**




**2. Common Cathode (CC)**

**In common cathode display, all cathode pins are connected together and led are controlled via anode terminal. It means to turn ON LED (segment), we have to apply proper voltage to the anode pin.**



## Resistor Connections

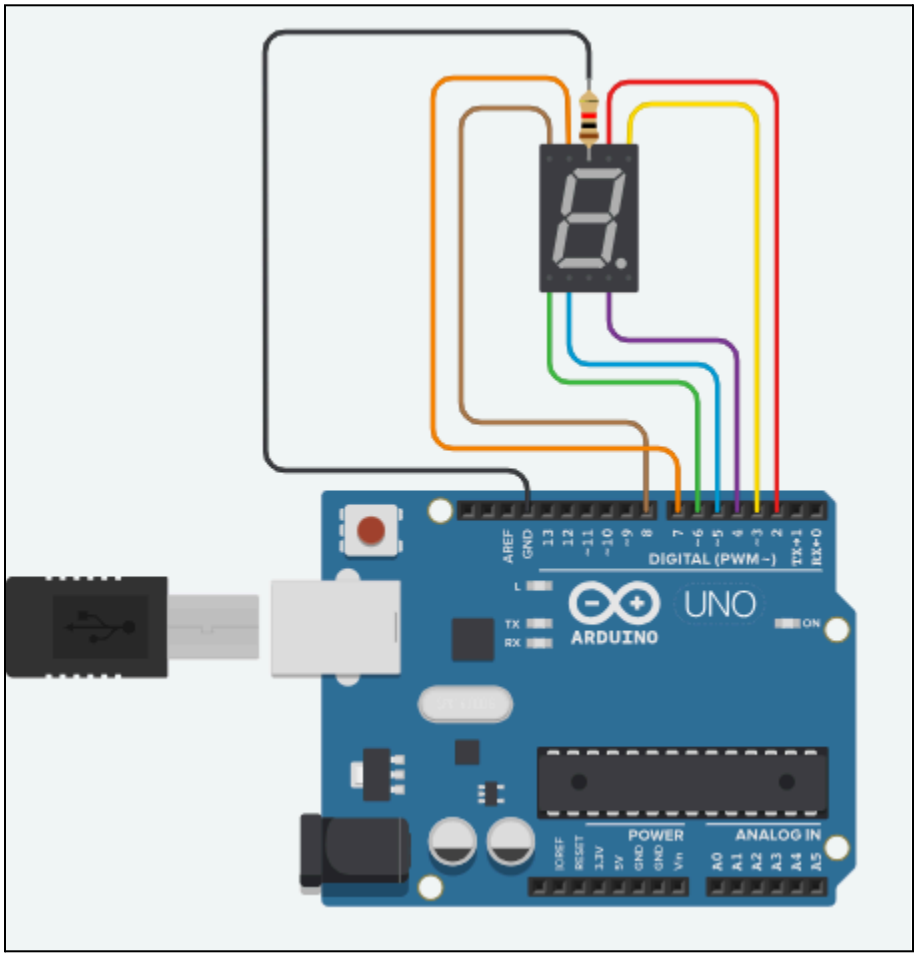
**We have to connect resistor to each segment individually.**
**Avoid LED's connection in parallel with one resistor, because each LED segment doesn't have exact same forward voltage drop.**
**If we connect one resistor to the parallel LEDs then some LEDs will glow and some will not. Because forward voltage drops are different for each LEDs, LEDs which are having lowest voltage drop across them will only glow. And though if we connect LEDs with ideally equal voltage drop, current will get divided and brightness will get affected each time while switching the LEDs.**

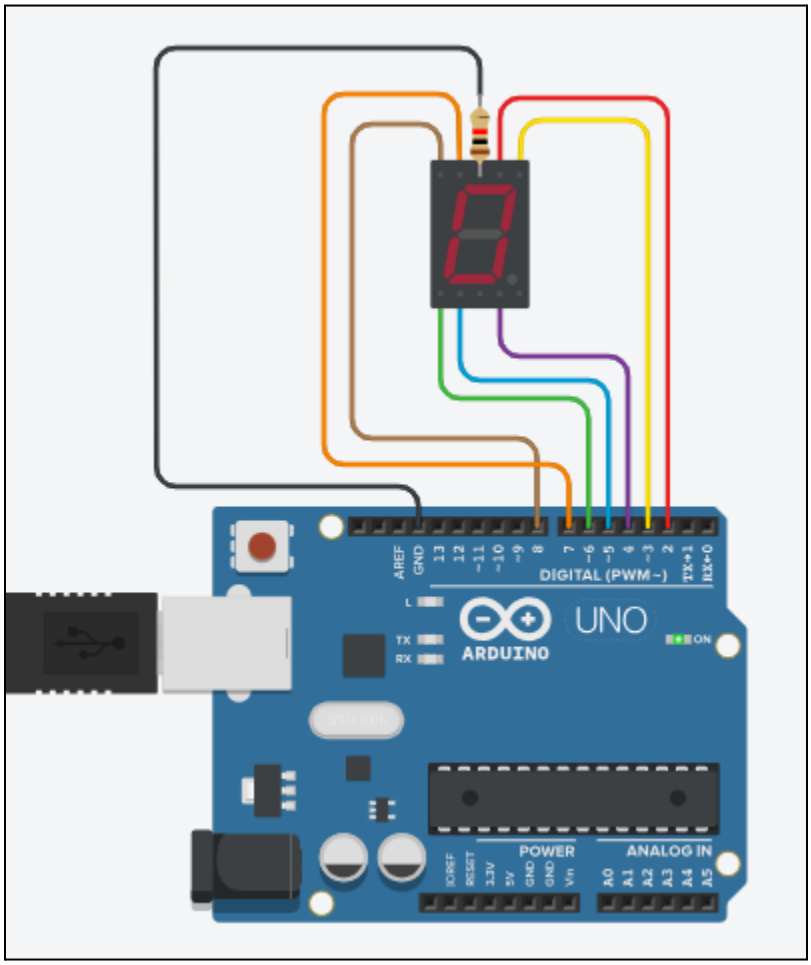
**Source Code:**

```
int A = 2;
int B=3;
int C=4;
int D=5;
int E =6;
int F=7;
int G=8;
int i;
void setup()
{
  for(i=2;i<9;i++)
        pinMode(i, OUTPUT);
}

void loop()
{
  digit(1,1,1,1,1,1,0);
  delay(200);
  digit(0,0,0,0,0,0,0);
  delay(500);
}

void digit(int a,int b,int c,int d,int e,int f, int g)
{
  digitalWrite(A,a);
  digitalWrite(B,b);
  digitalWrite(C,c);
```

```
  digitalWrite(D,d);
  digitalWrite(E,e);
  digitalWrite(F,f);
  digitalWrite(G,g);
}
```

**Output:**



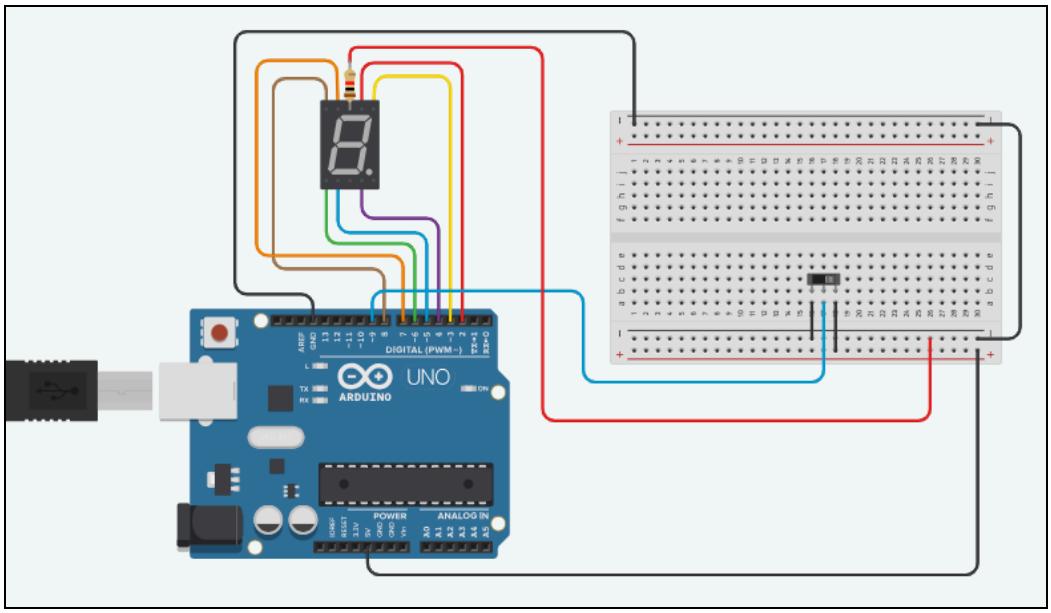

AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

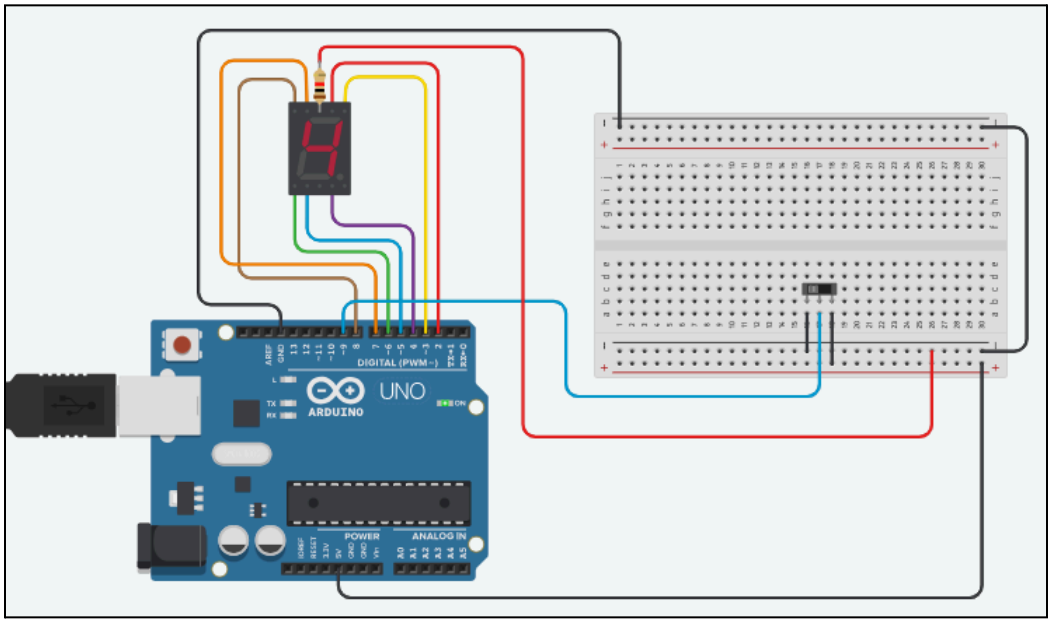
**Source code:**

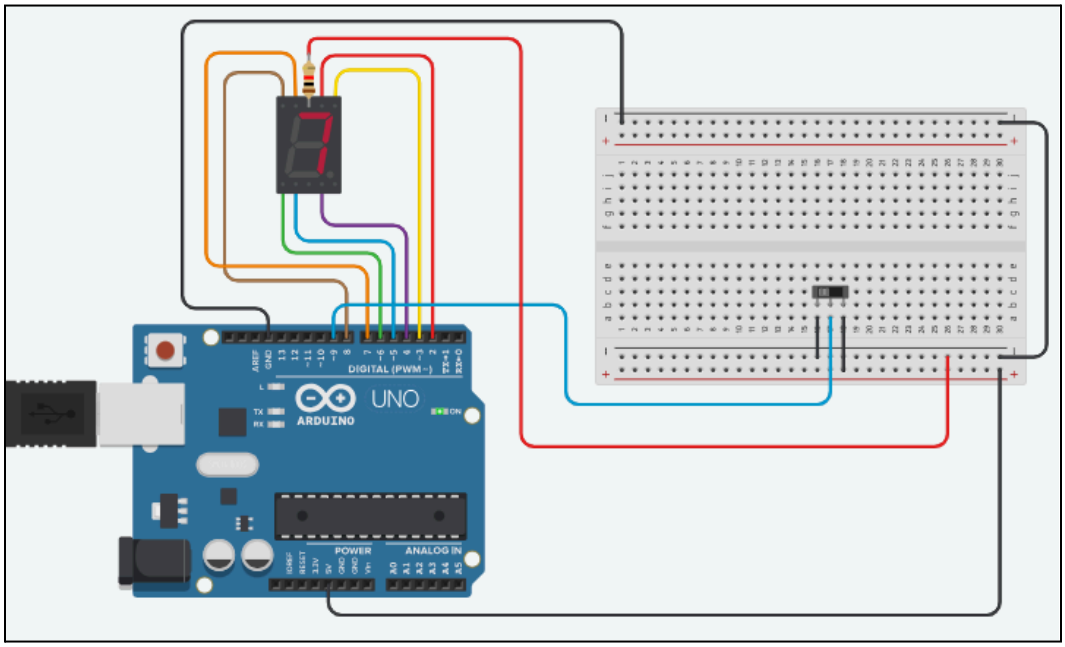
```
int A = 2;
int B = 3;
int C = 4;
int D = 5;
int E = 6;
int F = 7;
int G = 8;
int reading;
int butt = 9;
void setup()
{
  for(int i=2;i<9;i++)
          pinMode(i, OUTPUT);
    pinMode(butt , INPUT);
}

void loop()
{
  reading = digitalRead(butt);
  if (reading == 0) {
    digit(1,1,1,1,1,1,0); //0
    delay(200);
    digit(0,1,1,0,0,0,0); //1
    delay(500);
    digit(1,1,0,1,1,0,1); //2
    delay(500);
    digit(1,1,1,1,0,0,1); //3
    delay(500);
    digit(0,1,1,0,0,1,1); //4
    delay(500);
    digit(1,0,1,1,0,1,1); //5
    delay(500);
    digit(1,0,1,1,1,1,1); //6
    delay(500);
    digit(1,1,1,0,0,0,0); //7
    delay(500);
    digit(1,1,1,1,1,1,1); //8
    delay(500);
    digit(1,1,1,1,0,1,1); //9
    delay(2000);
  }
  else
  {
    digit(1,1,1,1,0,1,1);  //9
```

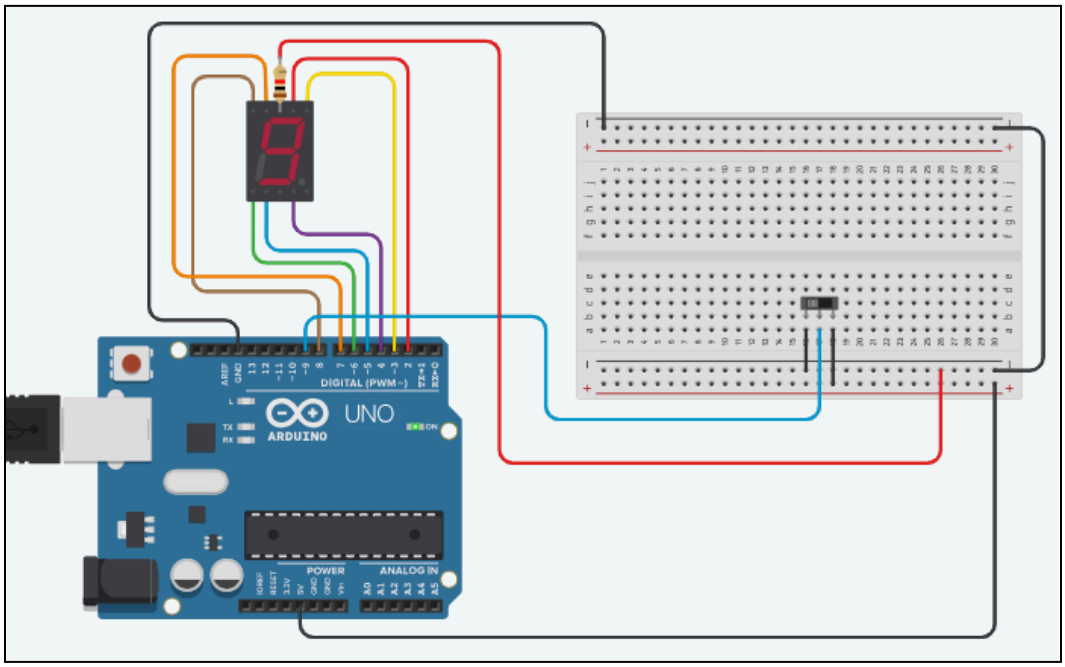
```
    delay(500);
    digit(1,1,1,1,1,1,1);
    delay(500);
    digit(1,1,1,0,0,0,0);
    delay(500);
    digit(1,0,1,1,1,1,1);
    delay(500);
    digit(1,0,1,1,0,1,1);
    delay(500);
    digit(0,1,1,0,0,1,1);
    delay(500);
    digit(1,1,1,1,0,0,1);
    delay(500);
    digit(1,1,0,1,1,0,1);
    delay(500);
    digit(0,1,1,0,0,0,0); //1
    delay(500);
    digit(1,1,1,1,1,1,0);  // 0
    delay(500);
  }

}

void digit(int a,int b,int c,int d,int e,int f, int g)
{
  digitalWrite(A,a);
  digitalWrite(B,b);
  digitalWrite(C,c);
  digitalWrite(D,d);
  digitalWrite(E,e);
  digitalWrite(F,f);
  digitalWrite(G,g);
}
```

**Output:**


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN

| | |
|---|---|
| **Program No:** | **8** |
| **Roll No :** | **1560** |
| **Title of Program :** | **Two 8 Digit SSD** |
| **Objective :** | **To design a 2 Digit Counter using 2 SSD and Arduino.** |

# Source Code:

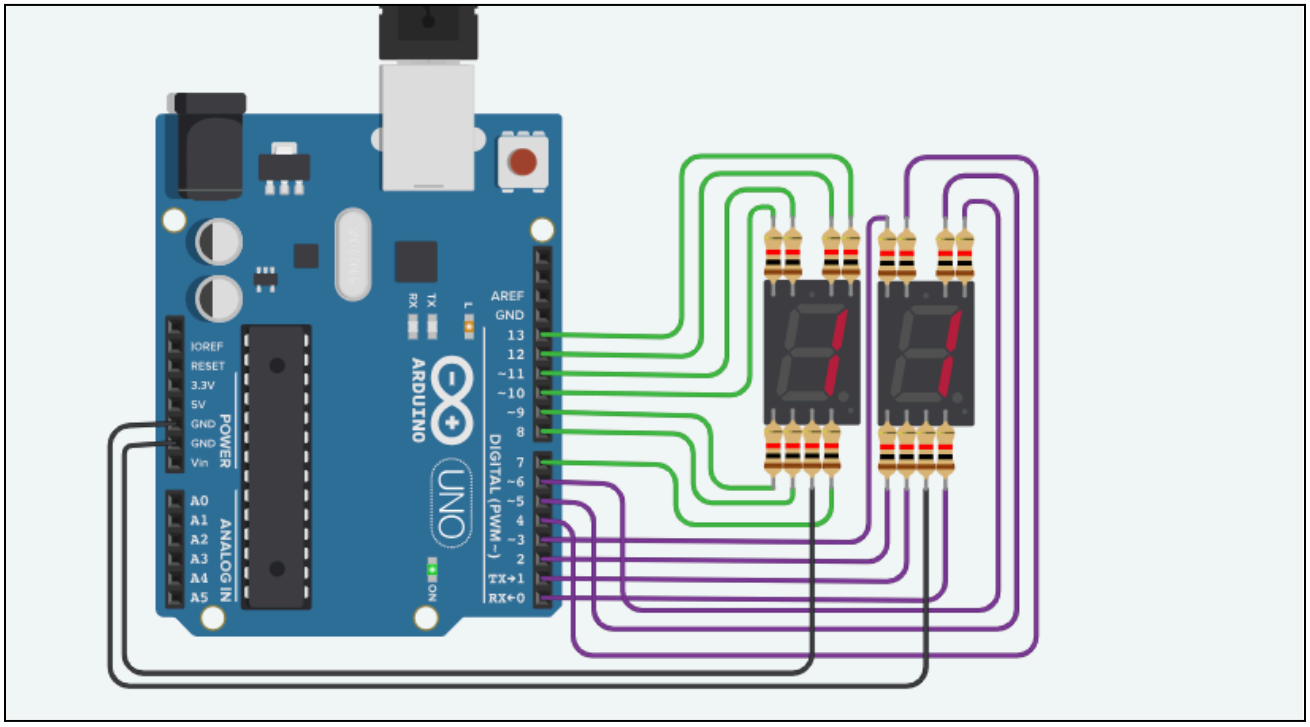
```
// C++ code
//
int a1 = 12;
int b1 = 13;
int c1 = 7;
int d1 = 8;
int e1 = 9;
int f1 = 11;
int g1 = 10;
int a2 = 5;
int b2 = 6;
int c2 = 0;
int d2 = 1;
int e2 = 2;
int f2 = 4;
int g2 = 3;

void setup()
{
  for(int i=0;i<=13;i++){
   pinMode(i, OUTPUT);
  }
}
int l1 = 0;
int l2 = 0;

void loop() {
 led1(l1);
 led2(l2);
 l2 = l2+1;
 l1 = (l1+(l2/10))%10;
```

```
 l2 = l2%10;
 delay(500);
}

void led1(int n){
 switch(n){
    case 0:
      display1(1,1,1,1,1,1,0);
      break;
    case 1:
      display1(0,1,1,0,0,0,0);
      break;
    case 2:
      display1(1,1,0,1,1,0,1);
            break;
    case 3:
      display1(1,1,1,1,0,0,1);
      break;
    case 4:
      display1(0,1,1,0,0,1,1);
            break;
    case 5:
      display1(1,0,1,1,0,1,1);
      break;
    case 6:
      display1(1,0,1,1,1,1,1);
            break;
    case 7:
      display1(1,1,1,0,0,0,0);
      break;
    case 8:
      display1(1,1,1,1,1,1,1);
            break;
    case 9:
      display1(1,1,1,1,0,1,1);
            break;
      }
}
void led2(int n){
```

```
 switch(n){
    case 0:
      display2(1,1,1,1,1,1,0);
      break;
    case 1:
      display2(0,1,1,0,0,0,0);
      break;
    case 2:
      display2(1,1,0,1,1,0,1);
            break;
    case 3:
      display2(1,1,1,1,0,0,1);
      break;
    case 4:
      display2(0,1,1,0,0,1,1);
            break;
    case 5:
      display2(1,0,1,1,0,1,1);
      break;
    case 6:
      display2(1,0,1,1,1,1,1);
            break;
    case 7:
      display2(1,1,1,0,0,0,0);
      break;
    case 8:
      display2(1,1,1,1,1,1,1);
            break;
    case 9:
      display2(1,1,1,1,0,1,1);
            break;
      }
}

void display2(int a, int b, int c, int d, int e, int f, int g){
      digitalWrite(a2, a);
      digitalWrite(b2, b);
      digitalWrite(c2, c);
      digitalWrite(d2, d);
```

```
      digitalWrite(e2, e);
      digitalWrite(f2, f);
      digitalWrite(g2, g);
}
void display1(int a, int b, int c, int d, int e, int f, int g){
      digitalWrite(a1, a);
      digitalWrite(b1, b);
      digitalWrite(c1, c);
      digitalWrite(d1, d);
      digitalWrite(e1, e);
      digitalWrite(f1, f);
      digitalWrite(g1, g);
}
```

## Output:



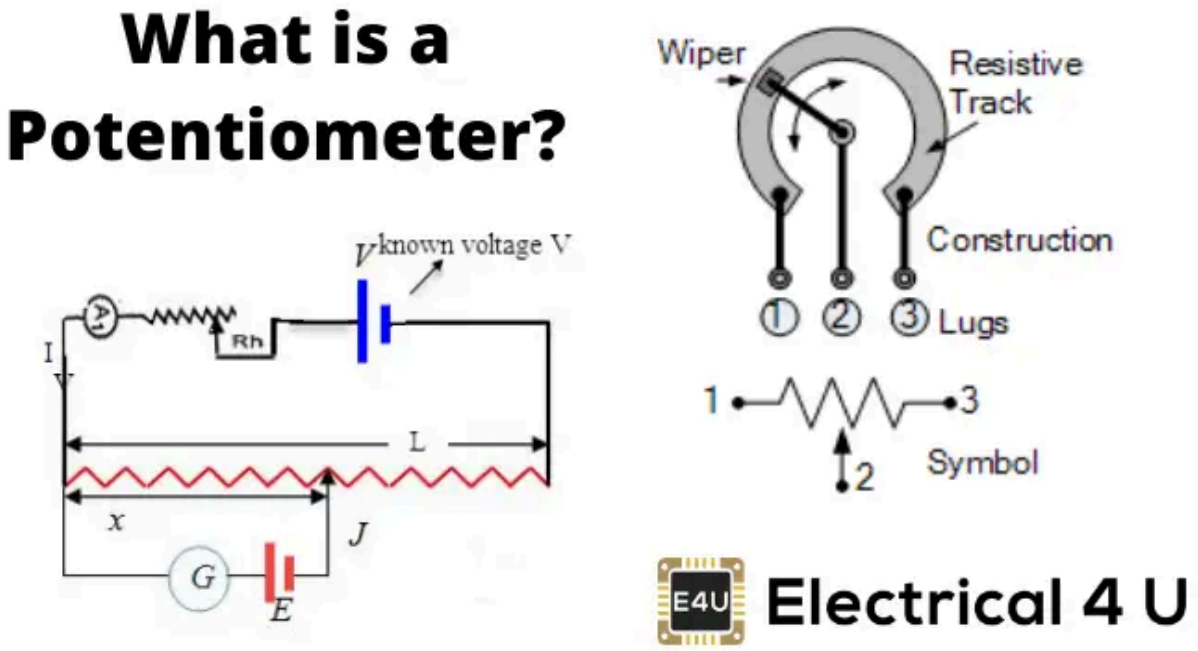
| | |
|---|---|
| **Program No:** | **10** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Using potentiometer and photoregistor** |
| **Objective :** | **a) LED with potentiometer**<br>**b) to increase brightness of LED based on LDR value** |

**A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.**

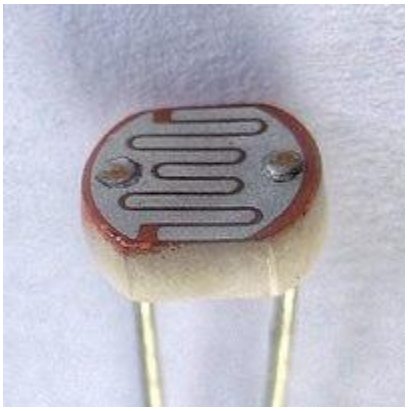
**If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.**

## Components of a Potentiometer:

**Resistive Track: A strip of resistive material that determines the overall resistance value of the potentiometer.**
**Wiper: A movable contact that slides along the resistive track, creating an adjustable voltage divider.**
**Terminals: Three terminals—two outer terminals connected to each end of the resistive track and a middle terminal connected to the wiper.**




**Photoresistor**



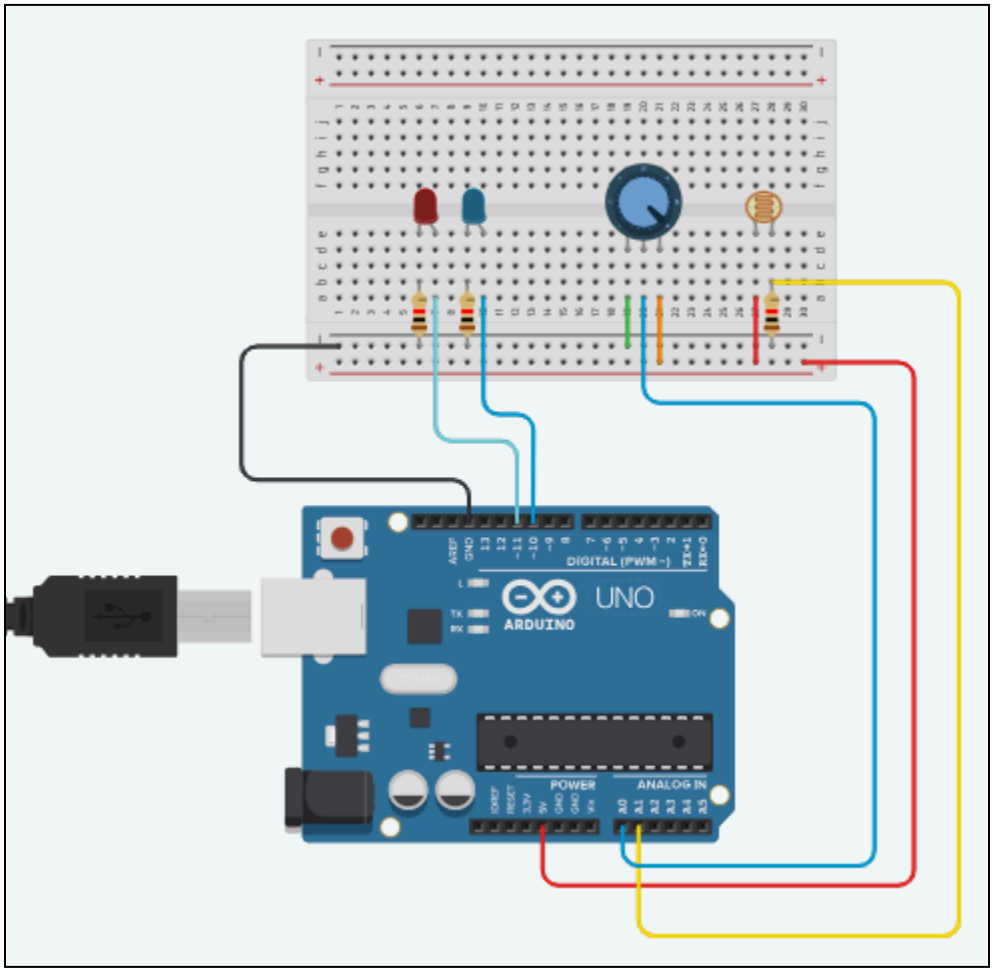
**A photoresistor (also known as a light-dependent resistor, LDR, or photo-conductive cell) is a passive component that decreases in resistance as a result of increasing luminosity (light) on its sensitive surface, in other words, it exhibits photoconductivity. A photoresistor can be used in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a semiconductor resistance**

## Source Code:

```
int potentval = 0;
int ldrval=0;
int pmap=0;
int ltrmap=0;
void setup()
{
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);

  Serial.begin(9600);
}

void loop()
{
  potentval=analogRead(A0);
```

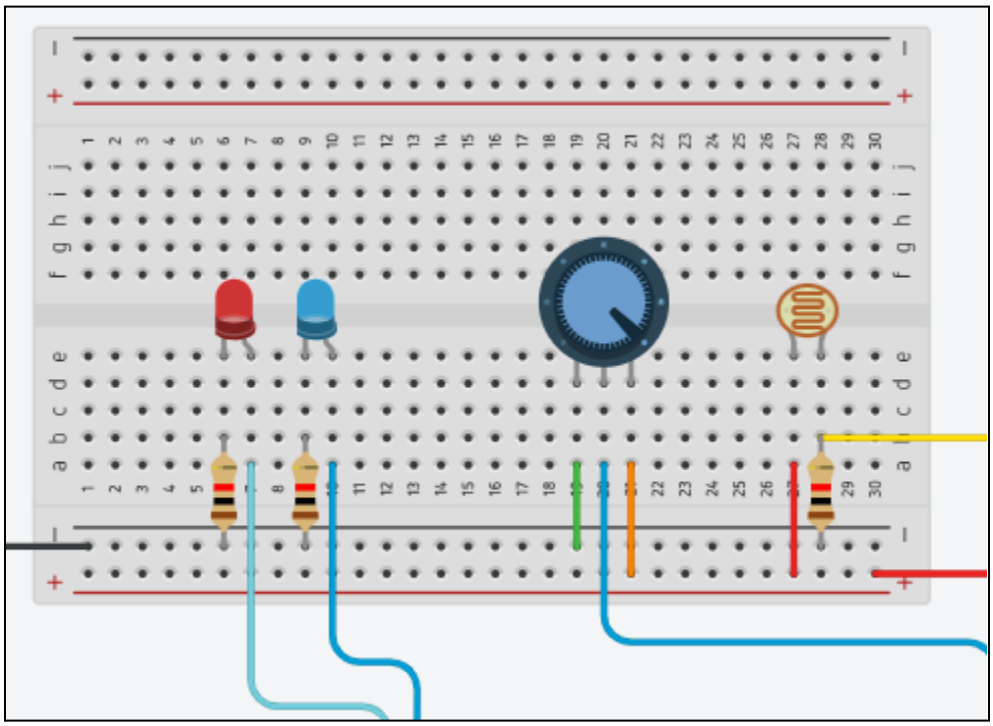
```
  ldrval=analogRead(A1);
  pmap= map(potentval,0,1023,0,255);
  ltrmap = map(ldrval, 6,679,0,255);
  analogWrite(11,pmap);
  analogWrite(10,255-ltrmap);
  //Serial.println(potentval);
  //Serial.println(mapval);
  //Serial.println(ltrmap);
}
```

**Output:**



Serial Monitor

```
1023
1023
1023
1023
1023
1023
1023
1023
1023
1023
1023
1023
```

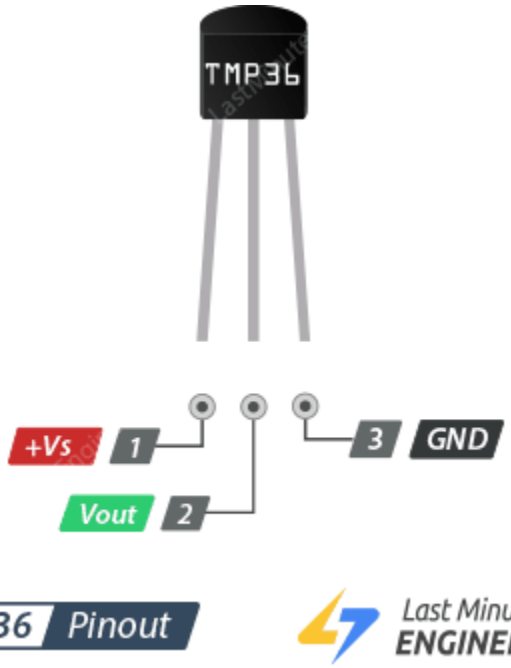


| | |
|---|---|
| **Program No:** | **11** |
| **Roll No :** | **1560** |
| **Title of Program :** | **LCD** |
| **Objective :** | **Display temperature using LCD and Temperature Sensore** |

**THEORY :**

## TMP36 Sensor Pinout

The TMP36 comes in three different form factors, but the most common type is the 3-pin TO-92 package, which looks just like a transistor. Let's take a look at its pinout.




TMP36 Pinout

Last Minute ENGINEERS.com

## Reading the Analog Temperature Data

As you can see in the wiring diagram above, the output of the TMP36 is connected to one of the analog inputs of the Arduino. The value of this analog input can be read with the `analogRead()` function.

However, the `analogRead()` function does not actually return the output voltage of the sensor. Instead it maps the input voltage between 0 and the ADC reference voltage (technically it is the operating voltage i.e. 5V or 3.3V unless you change it) to 10-bit integer values ranging from 0 to 1023. To convert this value back to the sensor's output voltage, use this formula:

Vout = (reading from ADC) * (5 / 1024)

This formula converts the number 0-1023 from the ADC into 0-5V

If you're using a 3.3V Arduino, you'll want to use this:

Vout = (reading from ADC) * (3.3 / 1024)

This formula converts the number 0-1023 from the ADC into 0-3.3V

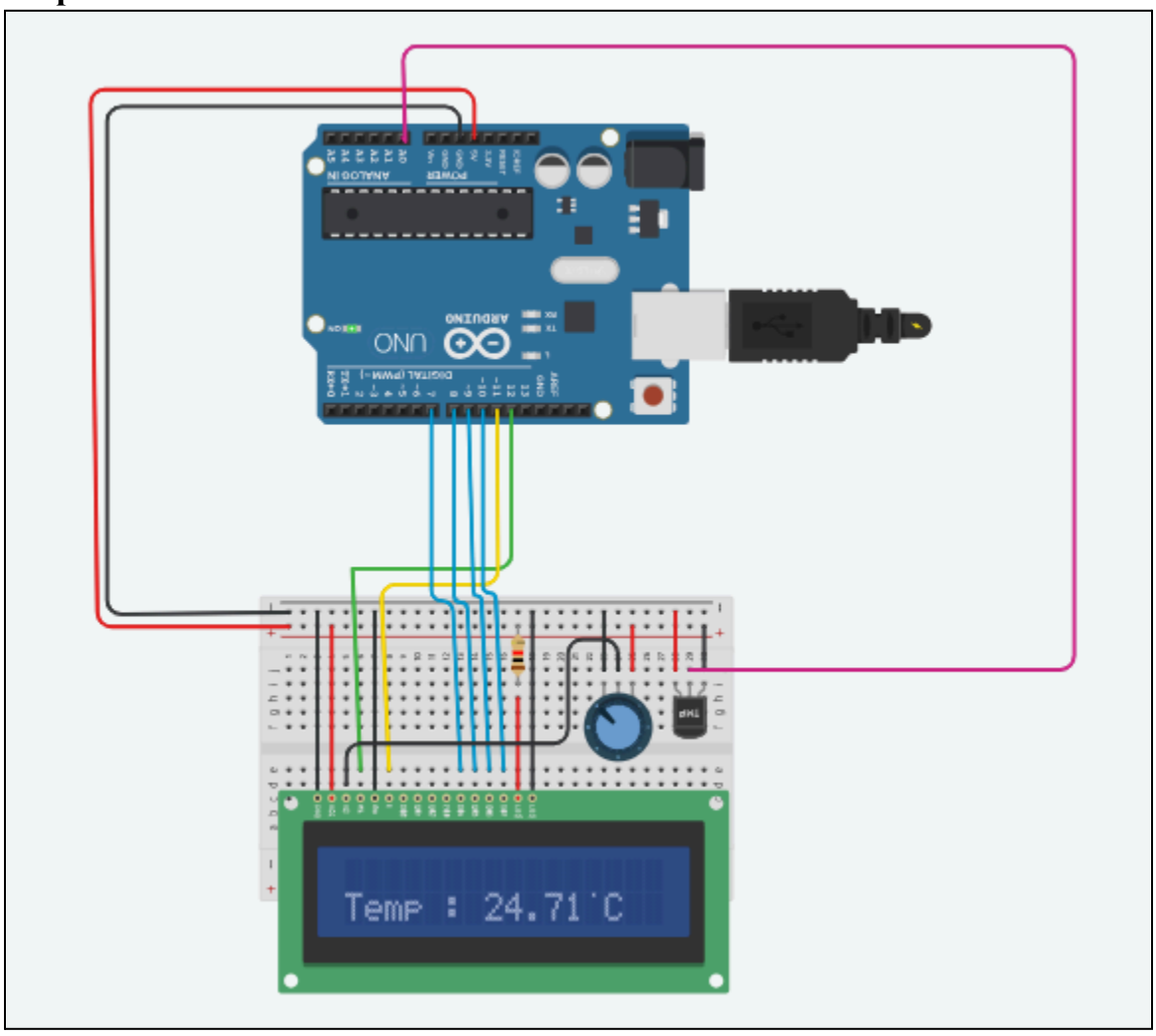
Then, to convert volts into temperature, use this formula:

## Source Code:

```
#include<LiquidCrystal.h>
const int rs = 12;
const int en = 11;
const int d4 = 7;
const int d5 = 8;
const int d6 = 9;
const int d7 = 10;

LiquidCrystal lcd(rs,en,d4,d5,d6,d7);

void setup()
{
 lcd.begin(16,2);
 pinMode(A0,INPUT);
 Serial.begin(9600);
}

void loop()
{
 int reading = analogRead(A0);
 Serial.println(reading);

  float voltage=reading*(5.0/1024.0);

  float temperatureC=(voltage-0.5)*100;
  Serial.println(reading);
  lcd.setCursor(0,1);

  lcd.print("Temp : ");
  lcd.print(temperatureC);
```

```
  lcd.print("\xB0");
  lcd.print("C");

  //for(int posCounter=0;posCounter <13; posCounter++)
//{
 //scroll one position left;
  //lcd.autoscroll();
  //lcd.scrollDisplayLeft();
//}
}
```

**Output:**

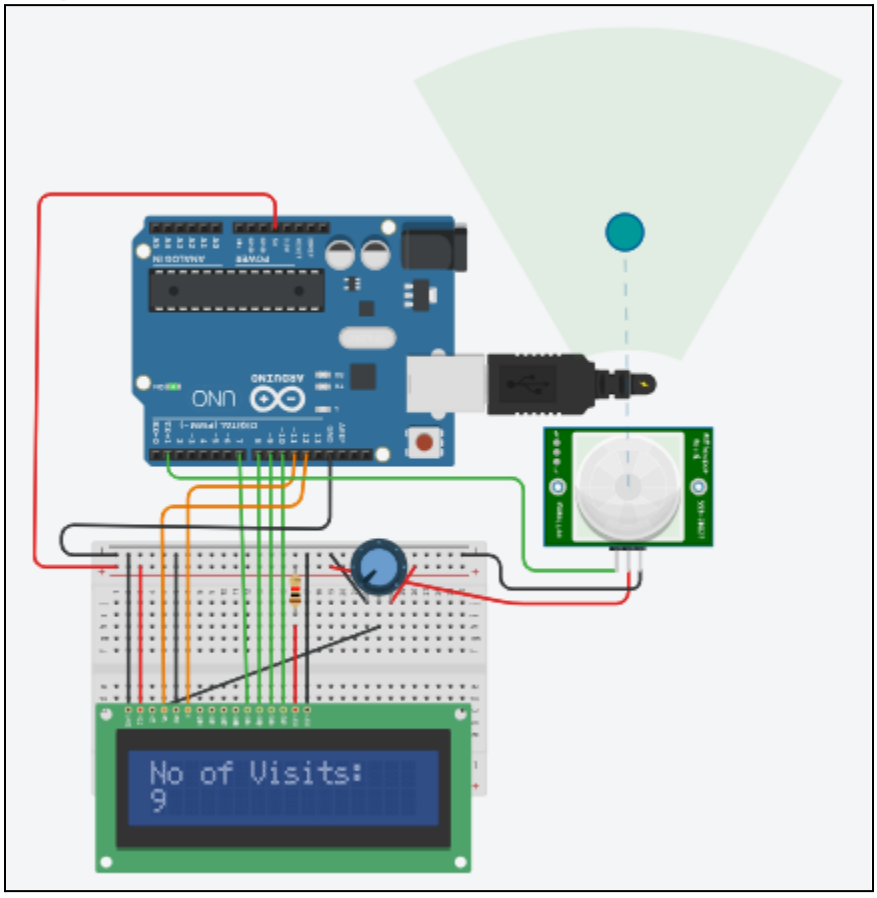


| | |
|---|---|
| **Program No:** | **15** |
| **Roll No :** | **1545** |
| **Title of Program :** | **LCD and PIR sensor** |
| **Objective :** | **Visitor counter using PIR sensor and LCD** |

**Source Code:**

```
#include <LiquidCrystal.h>

const int rs = 12;
const int en = 11;
const int d4 = 7;
const int d5 = 8;
const int d6 = 9;
const int d7 = 10;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
int visitct = 0;
int pir = 1;

void setup() {
  lcd.begin(16, 2); // Initialize LCD for 16 columns and 2 rows
  pinMode(pir, INPUT);
}

void loop() {
  lcd.setCursor(0, 0); // Move to the first row, first column
  lcd.print("No of Visits:"); // Print the label

  if (digitalRead(pir) == HIGH) {
    visitct++; // Increment visit count if PIR sensor detects motion
    delay(1000); // Delay to avoid multiple counts for one motion
  }

  lcd.setCursor(0, 1); // Move to the second row, first column
  lcd.print(visitct); // Print the visit count
}
```

**Output:**



| | |
|---|---|
| **Program No:** | **16** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Soil moisture sensor** |
| **Objective :** | **Soil sensor to detect moisture level and start dc motor** |

An IoT soil moisture sensor is a sensor designed to measure the water content in the soil. It consists of probes or sensors that are embedded in the soil and connected to a wireless module. The sensor collects data about the soil moisture level, and the IoT technology allows for the transmission of this data to a centralized system or cloud platform. This enables farmers and agricultural professionals to monitor and manage soil moisture remotely and make data-driven decisions.

IoT soil moisture sensors typically operate based on the principle of capacitance measurement. Capacitance-based sensors measure the dielectric constant of the soil, which is directly related to the soil moisture content. When the soil moisture changes, it affects the electrical properties of the soil, leading to a change in capacitance.

The sensor's probes or rods are inserted into the soil at different depths to capture moisture information from various soil layers. The probes emit electrical signals that interact with the surrounding soil, and the resulting capacitance is measured. This data is then converted into moisture readings and transmitted wirelessly to the IoT platform.

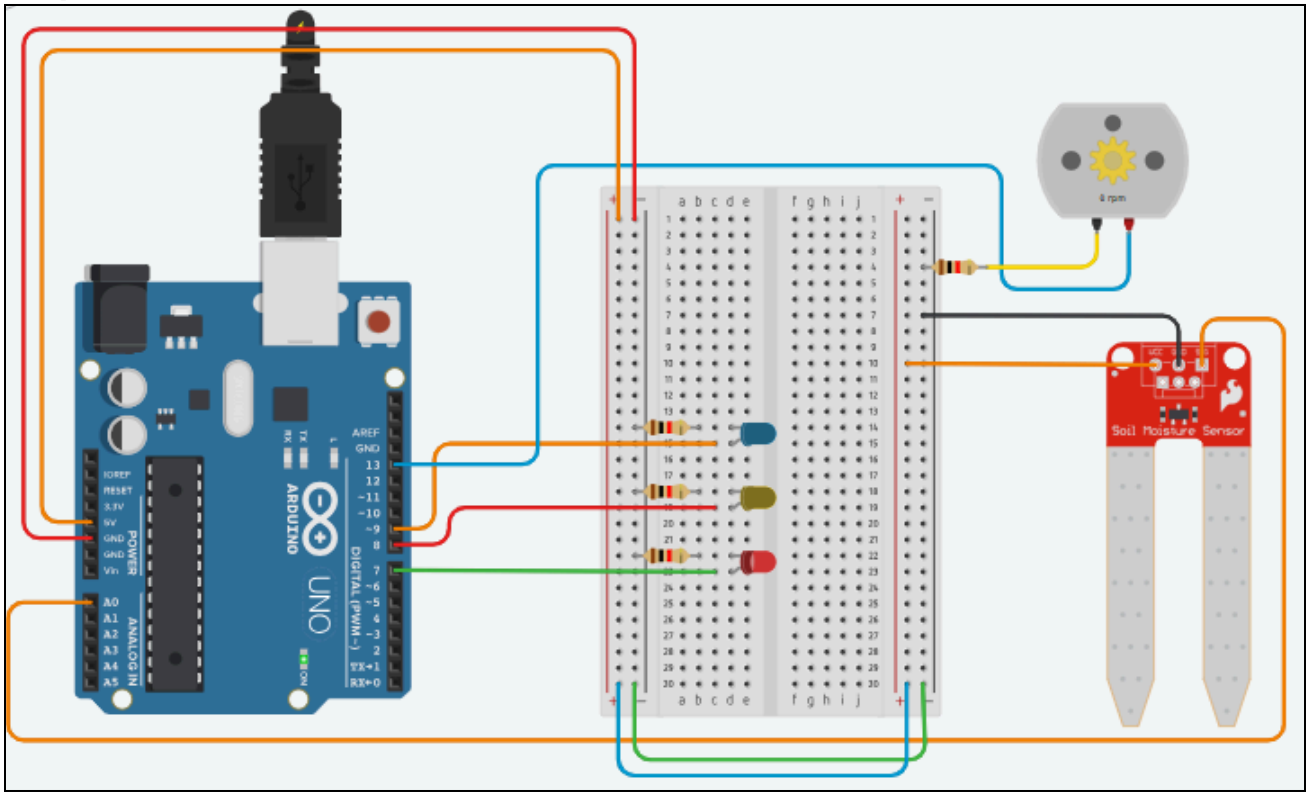
## Source Code:

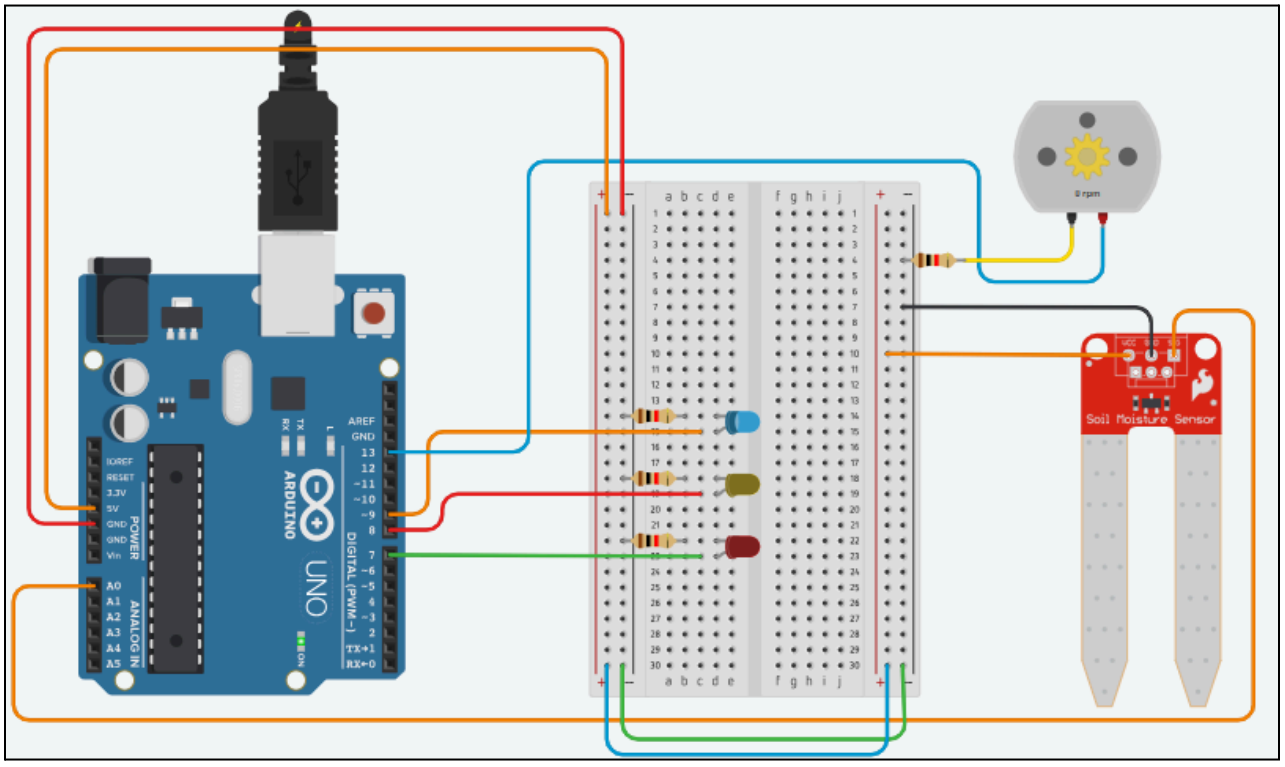
```
const int sensorpin = A0;
const int blue = 9;
const int yellow = 8;
const int red = 7;
const int dcmotor = 13;


int sensor;
int wet = 800;
```

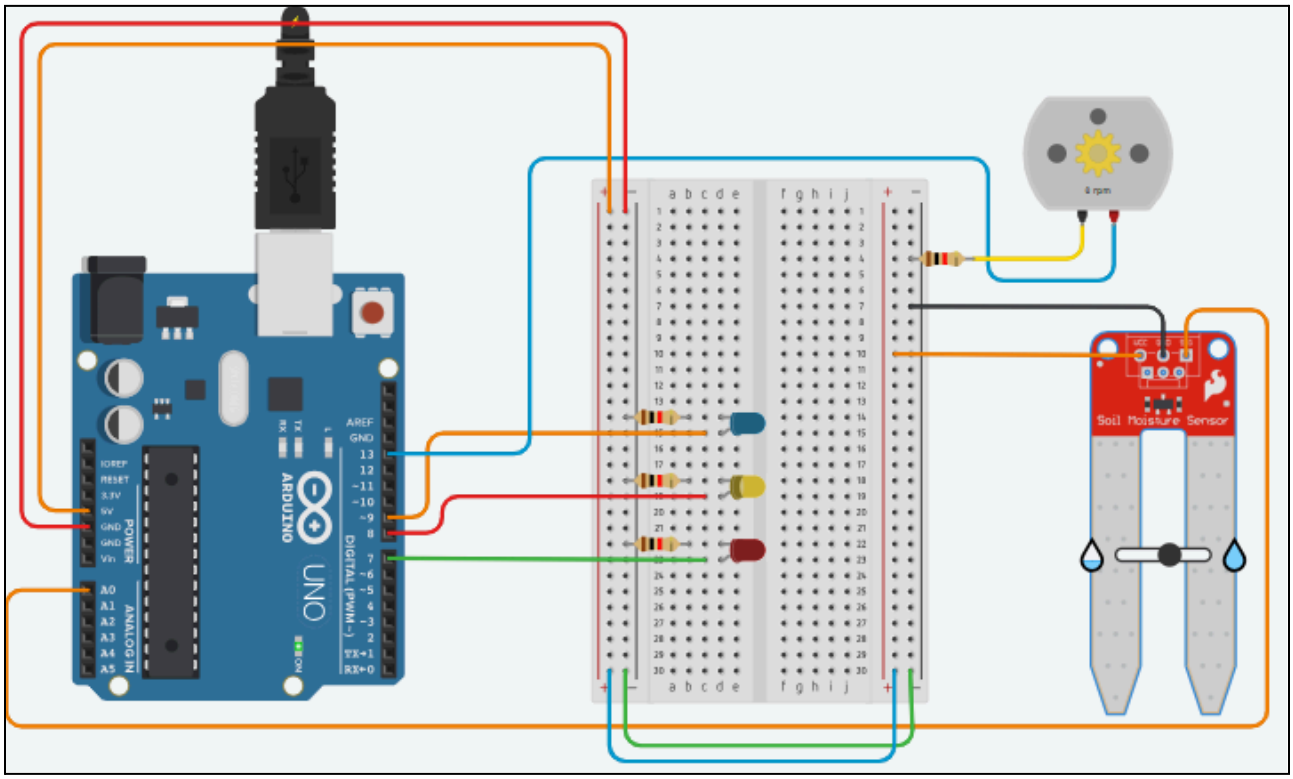
```
int dry = 500;

void setup()
{
  pinMode(red, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(dcmotor, OUTPUT);
  pinMode(sensorpin, INPUT);
  Serial.begin(9600);
}

void loop()
{
  sensor = analogRead(sensorpin);
  Serial.println(sensor);

  if (sensor > wet)
  {
        digitalWrite(red, LOW);
    digitalWrite(yellow, LOW);
    digitalWrite(blue, HIGH);
    digitalWrite(dcmotor, LOW);
  }
  else if (sensor < dry)
  {
        digitalWrite(red, HIGH);
    digitalWrite(yellow, LOW);
    digitalWrite(blue, LOW);
    digitalWrite(dcmotor, LOW);
  }
  else
  {
        digitalWrite(red, LOW);
    digitalWrite(yellow, HIGH);
    digitalWrite(blue, LOW);
    digitalWrite(dcmotor, LOW);
  }
  delay(1000);
```

```
}
```

**Output:**




Soil Moisture Sensor
ARDUINO
UNO


Soil Moisture Sensor
ARDUINO
UNO

MUMBAI EDUCATIONAL TRUST

MET INSTITUTE OF COMPUTER SCIENCE

| | |
|---|---|
| Program Number | 18 |
| Roll Number | 1525 |
| Title of program | Gas Sensor |
| Objective | To detect gas leakage using Gas Sensor |

## Theory : -

## Gas Sensor:

Gas sensors are generally understood as providing a measurement of the concentration of some analyte of interest, such as CO, CO2, NO*x*, SO2, without at this point dwelling on the plethora of underlying approaches such as optical absorption, electrical conductivity, electrochemical (EC), and catalytic bead (see Section 3). However, and as discussed in Section 2, many other gas sensors measure a physical property of the environment around them, such as simple temperature, pressure, flow, thermal conductivity, and specific heat, or more complex properties such as heating value, supercompressibility, and octane number for gaseous fuels. The latter may require capital-intensive (engines) or destructive testing, for example, via combustion, or involve the measurement of a number of parameters to serve as inputs to a correlation with the complex property of interest.

When the sensor provides a multiplicity of outputs, as with optical or mass spectrometers (MSs), we refer to it as a gas analyzer. Gas chromatography (GC), differential thermal analysis (DTA), ion mobility, and nuclear magnetic resonance

(NMR) are additional examples, some of which will be detailed in Section 4. Such analyzers, preferred by the author, should not be confused with sensor arrays, in which different sensing materials (typically polymers and metal oxides) are used on each element of the array, which then needs to conform to difficult-to-achieve stability requirements.
The performance of all of the above-mentioned sensors and analyzers may be characterized by their signal-to-noise (S/N) ratio, minimum detectable limit (MDL), selectivity, and response time. Increasingly, power consumption, size, and weight are becoming more important as interest and demand increases for handheld, battery-powered sensors, with or without wireless capability. These specifications may be viewed as simple performance parameters, because they are relatively simple to quantify.

Self-calibration, drift, S/N, and false alarm rate (FAR) (mainly for composition sensors or analyzers) require more sophisticated approaches, but are of increasing importance in all applications such as for medical, industrial, environmental, security, and first-responder use. Section 5 goes into the details of this subject.

Another classification of gas sensors and analyzers could be based on their sampling method: by diffusion, pumped transport, or via remote optical sampling to induce fluorescence, absorption, or scattering.
Researchers, designers, and planners continually face the need to make development or fabrication decisions before all the facts are available.

Therefore, there is a perennial need to generate estimates about the performance and sensitivity of devices, structure, and also sensor systems. This is where mathematical modeling, be it simple or complex or *ad hoc* (to simulate a specific sensor) or

multipurpose (such as ANSYS, FLUENT to simulate heat transfer or flow of a sensor within a given programmatic framework, which is adapted to individual geometries and conditions) can be of tremendous help. Rather than dedicating a section specifically to this subject, several examples will be discussed throughout this chapter.

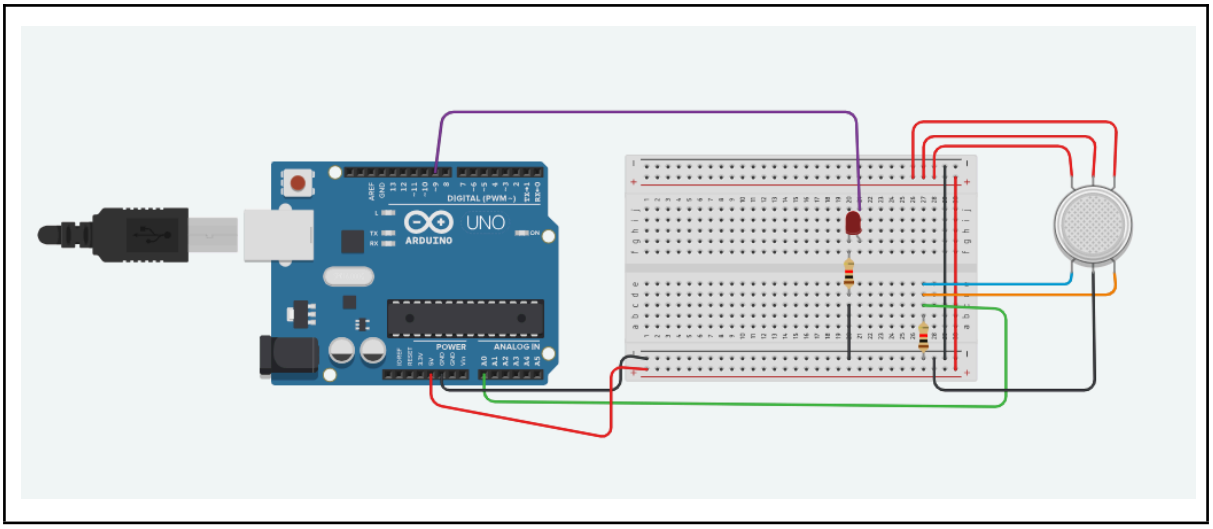
The intent of this chapter is not to provide an exhaustive review of the world of gas sensors, nor a history of their development, but to highlight and share selected gas-sensing approaches that impressed the author in meeting modern expectations for performance, features, and cost.

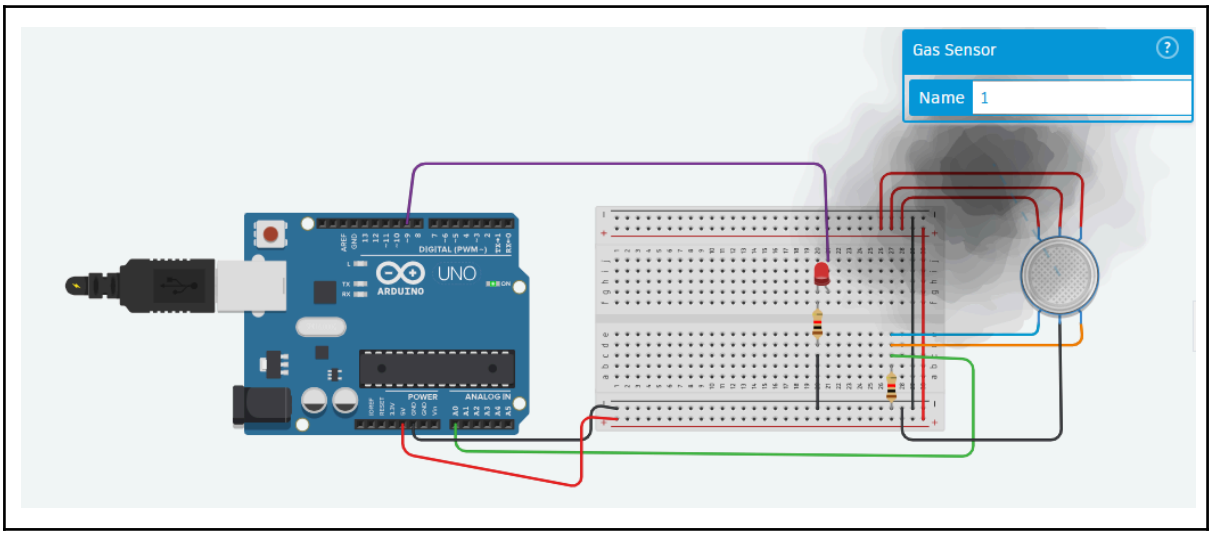
The issue of cost merits additional comments. Contrary to initial negative reactions one might have about it, because of its potential commercialism, the author subscribes to the view that cost just adds a tough professional challenge (conservation, sustainable development, affordability) to all the others related to achieving generic and useful sensor performance attributes mentioned above. In fact, many elegant sensing approaches are withering on the shelf because few potential users could afford to implement them.

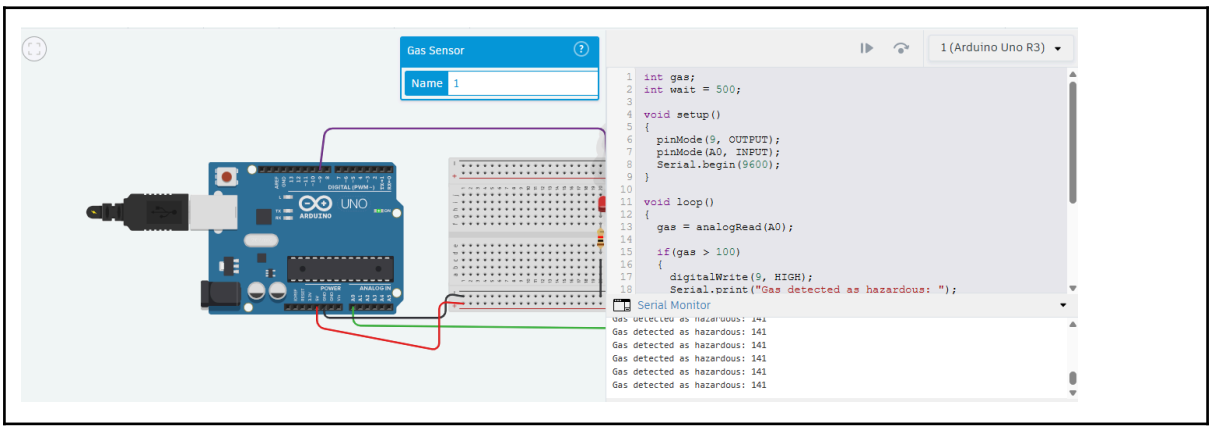
**Source Code : -**

```
int gas;
int wait = 500;

void setup()
{
  pinMode(9, OUTPUT);
  pinMode(A0, INPUT);
  Serial.begin(9600);
}

void loop()
{
  gas = analogRead(A0);

  if(gas > 100)
  {
    digitalWrite(9, HIGH);
    Serial.print("Gas detected as hazardous: ");
    Serial.println(gas);
    delay(wait);
  }
  else
  {
    digitalWrite(9, LOW);
    Serial.print("Gas detected as normal: ");
    Serial.println(gas); // Fixed "serial" to "Serial"
  }
}
```

# Output: -




Gas Sensor
Name 1


Gas Sensor
Name 1
1 (Arduino Uno R3)
int gas;
int wait = 500;
void setup()
{
pinMode(9, OUTPUT);
pinMode(A0, INPUT);
Serial.begin(9600);
}
void loop()
{
gas = analogRead(A0);
if(gas > 100)
{
digitalWrite(9, HIGH);
Serial.print("Gas detected as hazardous: ");
Serial Monitor
Gas detected as hazardous: 141
Gas detected as hazardous: 141
Gas detected as hazardous: 141
Gas detected as hazardous: 141
Gas detected as hazardous: 141
Gas detected as hazardous: 141

MUMBAI EDUCATIONAL TRUST

MET INSTITUTE OF COMPUTER SCIENCE

| | |
|---|---|
| Program Number | 19 |
| Roll Number | 1525 |
| Title of program | IR Sensor |
| Objective | Using Remote with IR sensor |

## Theory : -

## IR Remote:

# Introduction to IR Sensors

Infrared (IR) sensors are electronic devices that detect the presence of an object or measure its temperature. Generally, IR sensors operate by detecting thermal radiation. These radiations fall under the infrared region of the electromagnetic spectrum and are invisible to the human eye. Therefore, these radiations go unnoticed in our daily life.

IR sensors are available in different types and can be used for a variety of applications, such as robotics, security systems, and other automation projects.

# Working Principles of IR Sensors

The working principle of an IR sensor is based on the transmission and reception of infrared radiation. It consists of a transmitter that generates infrared radiation and a receiver that detects IR radiation.

It must be noted that the transmitter and receiver should have the same operational wavelength. This is because a receiver having a different operational wavelength will fail to detect the IR radiations emitted by the transmitter, and the system will not function as desired.

## Transmitter

The transmitter part consists of an infrared LED (light emitting diode) which produces infrared radiation when supplied with electric current. This radiation is then directed toward the object that needs to be detected.

## Receiver

The receiver part of an IR sensor consists of a photodiode, a semiconductor device that is sensitive to infrared radiation. When the infrared radiation from the LED strikes the photodiode, it produces an electric current which is then amplified and converted into a voltage signal. This voltage signal is then used to trigger the desired output.

# Types of IR Sensors

Based on the wavelength, size, voltage, etc., there are different types of IR sensors used for different applications. There are several types of IR sensors available on the market. The most commonly used types of IR sensors are active IR sensors and passive IR sensors.

## Active Infrared Sensors

Active IR sensors are the most commonly used IR sensors. They consist of an infrared LED and a phototransistor, as described above. These sensors are used to detect the presence of objects in their vicinity.

Active IR sensors are commonly found in daily household applications like TV remotes, break beam sensors, etc., where a source sends the IR signal and a receiver detects the signal and responds accordingly.

A Laser IR sensor is also a type of active infrared sensor that is used to detect the presence of objects over long distances. They use an infrared laser beam to detect the presence of objects, and they are commonly used in military applications.

## Passive Infrared Sensors

Passive IR sensors, on the other hand, only consist of the IR receiver and do not emit any radiation. Instead, they detect infrared radiation emitted by objects in their vicinity. These sensors are commonly used in security and safety systems.

There are two types of passive Infrared sensors: Thermal IR Sensors and Quantum IR Sensors.

## Thermal IR Sensors

**An IR temperature sensor is a type of infrared sensor that measures the temperature of objects. These sensors operate by detecting infrared thermal radiation emitted by objects in their vicinity. This radiation is then used to calculate the object's temperature.**

**An IR temperature sensor is commonly used to measure the temperature of objects without making physical contact with them. These sensors are commonly used in thermographic cameras, medical imaging, and industrial applications like temperature monitoring and flame detection.**

## Quantum IR Sensors

**A quantum IR (Infrared) sensor detects and measures infrared radiation by utilizing the quantum mechanical properties of molecules. It is used to measure environmental temperature, motion, and other physical properties.**

**When compared to thermal IR sensors, quantum IR sensors provide better accuracy over a wider range of temperatures, higher sensitivity, and the ability to detect IR radiations over a wider frequency range. This makes quantum IR sensors suitable for industrial applications where high accuracy and reliability are of utmost importance.**

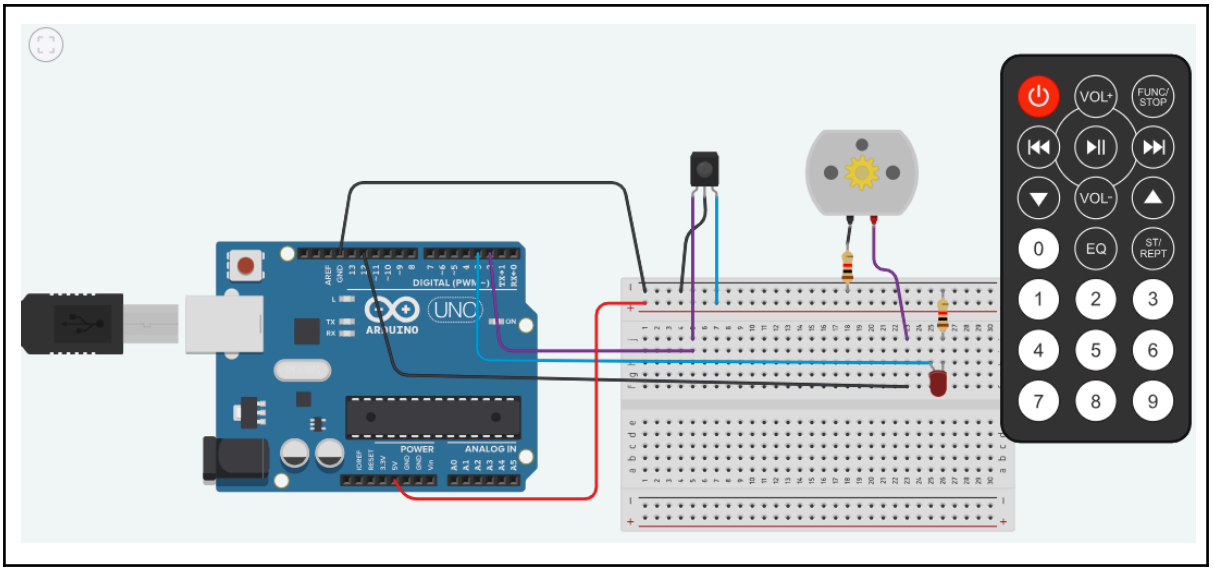
**Some of the popular applications of quantum IR sensors include medical imaging, industrial sensing, and remote sensing. In medical imaging, they are used to detect tumors and other abnormalities, while in industrial sensing, they are used to monitor temperature and motion. In remote sensing, they are used to measure environmental conditions, such as air temperature, humidity, and air pressure.**

## Source Code : -

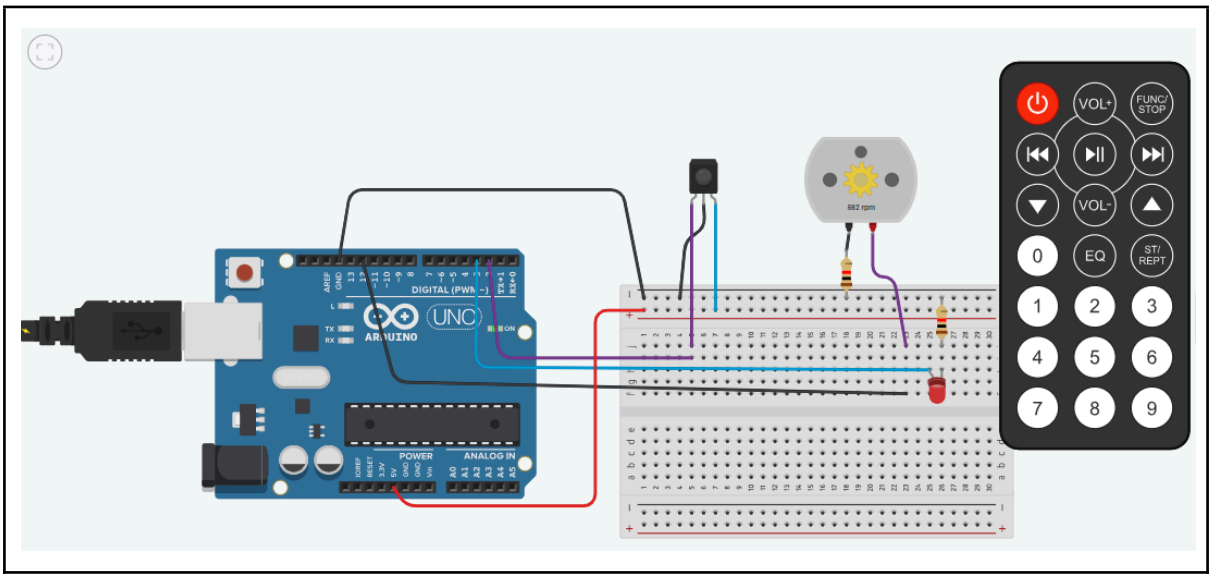
```
// C++ code
//
#include<IRremote.h>
int recv_pin = 2;
IRrecv irr(recv_pin);

void setup()
{
  Serial.begin(9600);
  pinMode(12,OUTPUT);
  pinMode(3,OUTPUT);
  irr.enableIRIn();
```

```
}

void loop()
{
  if(irr.decode())
  {
    switch(irr.decodedIRData.command)
    {
      case 0X10:
         digitalWrite(12,HIGH);
         digitalWrite(3,HIGH);
         Serial.println("START");
         break;
     case 0X11:
         digitalWrite(12,LOW);
         digitalWrite(3,LOW);
         Serial.println("STOP");
         break;
     default:
         Serial.println("Wrong I/P");
      break;
    }
    irr.resume();
  }
}
```

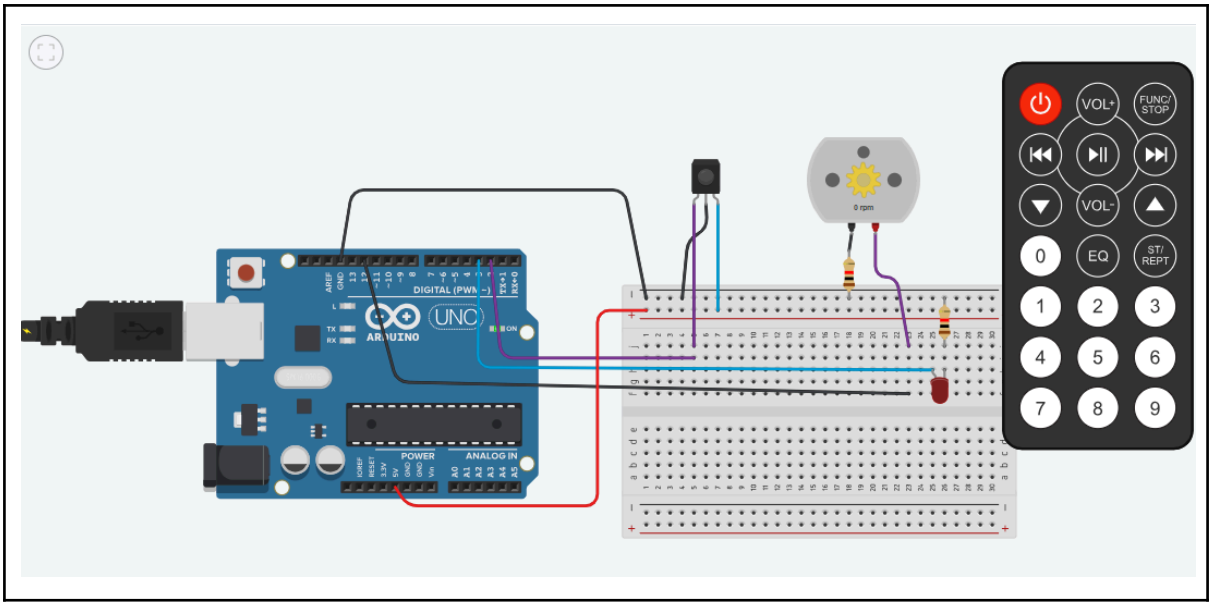
**Output: -**




**Press “1” of the IR Remote to Turn on the led and Motor.**




**Press “2” of the IR Remote to Turn off the led and Motor.**


VOL+
FUNC/
STOP
VOL-
0
EQ
ST/
REPT
1
2
3
4
5
6
7
8
9
POWER
ANALOG IN

| | |
|---|---|
| **Program No:** | **20** |
| **Roll No :** | **1545** |
| **Title of Program :** | **Basic Calculator** |
| **Objective :** | **Calculator using keypad** |

**Source Code:**

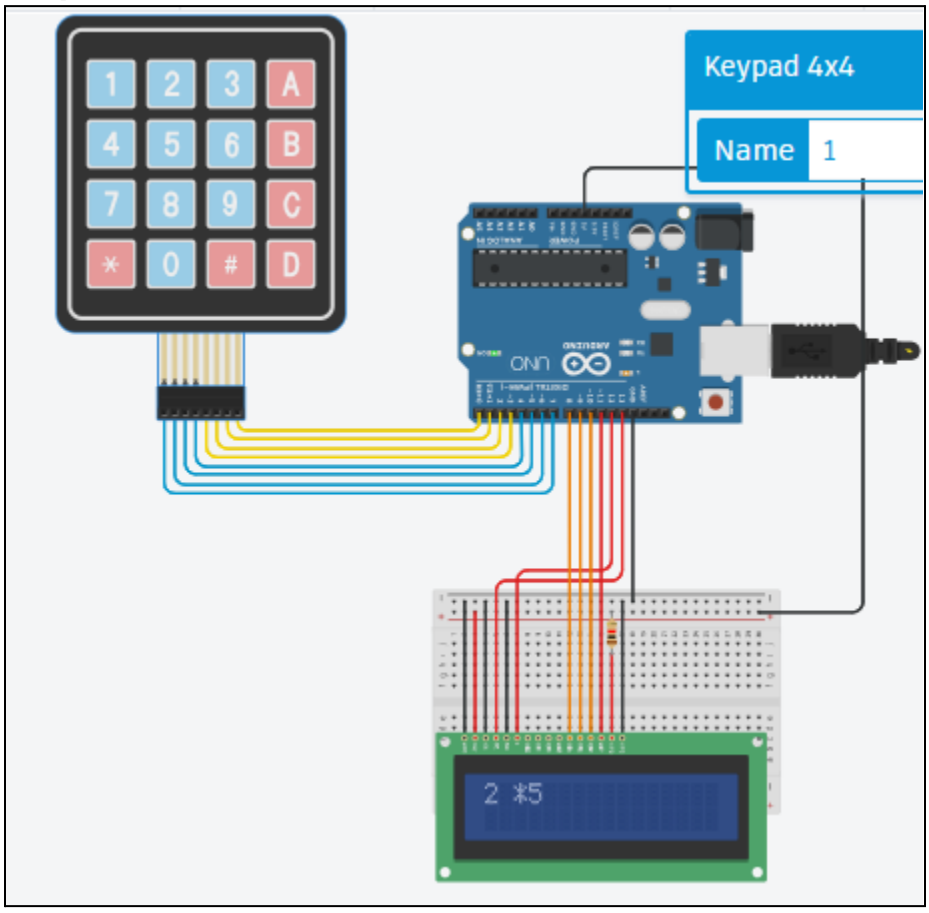
```
 // C++ code
//
#include<LiquidCrystal.h>
#include<Keypad.h>

int rs =13;
int en=12;
int d4=8;
int d5 = 9;
int d6=10;
int d7 = 11;
long first = 0;
long second =0;
double total = 0;
int posit = 0;
char customKey;
const byte ROWS =4;
const byte COLS=4;
char keys[ROWS][COLS]={
  {'1','2','3','/'},
  {'4','5','6','*'},
  {'7','8','9','-'},
  {'C','0','=','+'}
};

byte rowPins[ROWS]={7,6,5,4};
byte colPins[COLS]={3,2,1,0};

Keypad customKeypad =
Keypad(makeKeymap(keys),rowPins,colPins,ROWS,COLS);
```

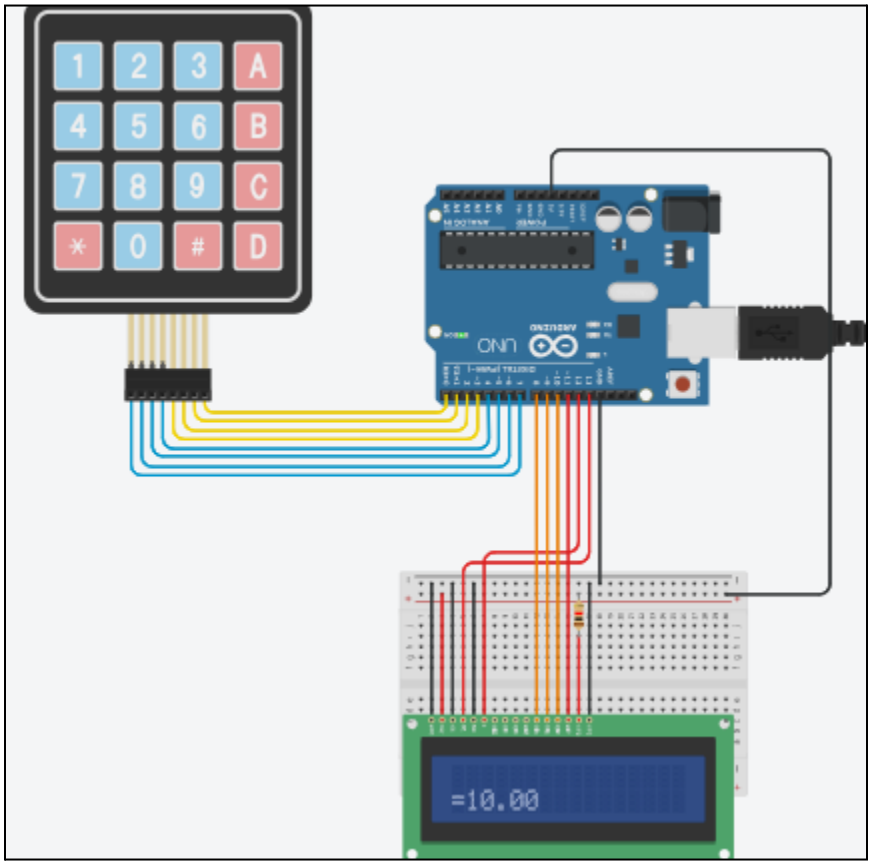
```
LiquidCrystal lcd(rs,en,d4,d5,d6,d7);

void setup()
{
  lcd.begin(16,2);
  lcd.setCursor(5,0);
  lcd.print("Basic calci");
  delay(2000);
  lcd.clear();
}

void loop()
{
  lcd.setCursor(0,1);
  customKey = customKeypad.getKey();
  switch(customKey){
        case '0' ... '9':
        lcd.setCursor(0,0);
        first=first * 10 + (customKey - '0');
        lcd.print(first);
        posit++;
        break;

    case '+':
    first = (total != 0 ? total : first);
    lcd.setCursor(posit + 1,0);
    lcd.print("+");
    posit+=2;
    second = secondNumber();
    total = first + second;
    lcd.setCursor(1,1);
    lcd.print(total);
    first =0, second =0;
    posit =0;
    break;

    case '-':
    first = (total != 0 ? total : first);
    lcd.setCursor(posit + 1,0);
```

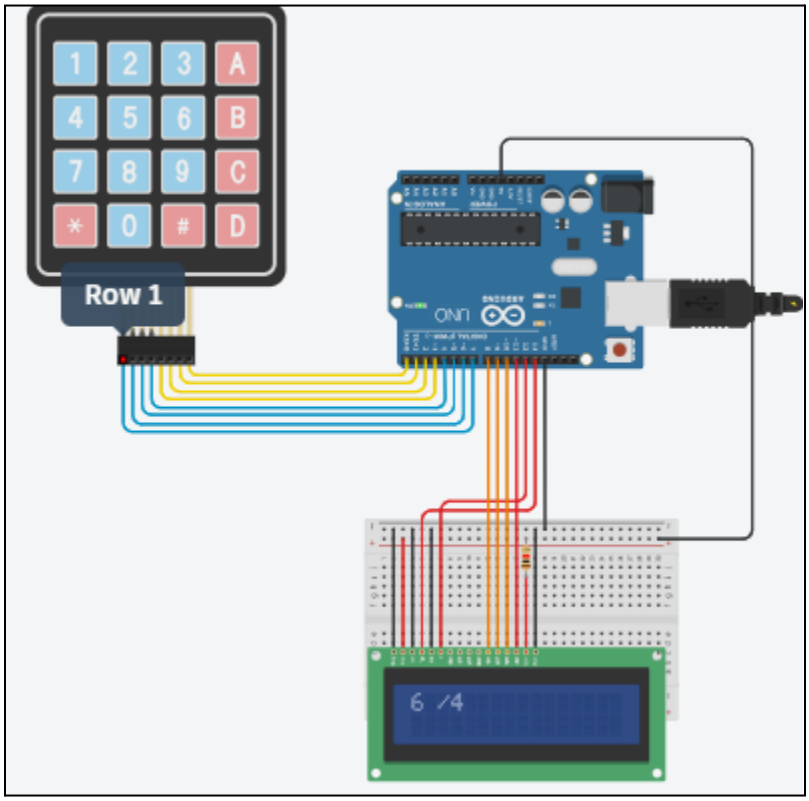
```
 lcd.print("-");
 posit+=2;
 second = secondNumber();
 total = first - second;
 lcd.setCursor(1,1);
 lcd.print(total);
 first =0, second =0;
 posit =0;
 break;

  case '*':
              first = (total != 0 ? total : first);
      lcd.setCursor(posit + 1,0);
      lcd.print("*");
      posit+=2;
      second = secondNumber();
      total = first * second;
      lcd.setCursor(1,1);
      lcd.print(total);
      first =0, second =0;
      posit =0;
 break;

 case '/':
 first = (total != 0 ? total : first);
 lcd.setCursor(posit + 1,0);
 lcd.print("/");
 posit+=2;
 second = secondNumber();
 lcd.setCursor(1,1);
 second = 0 ? lcd.print("Error") : total = (float)first/(float)second;
 lcd.print(total);
first =0, second =0;
 posit =0;
 break;

  case 'C':
 first =0;
 second =0;
```

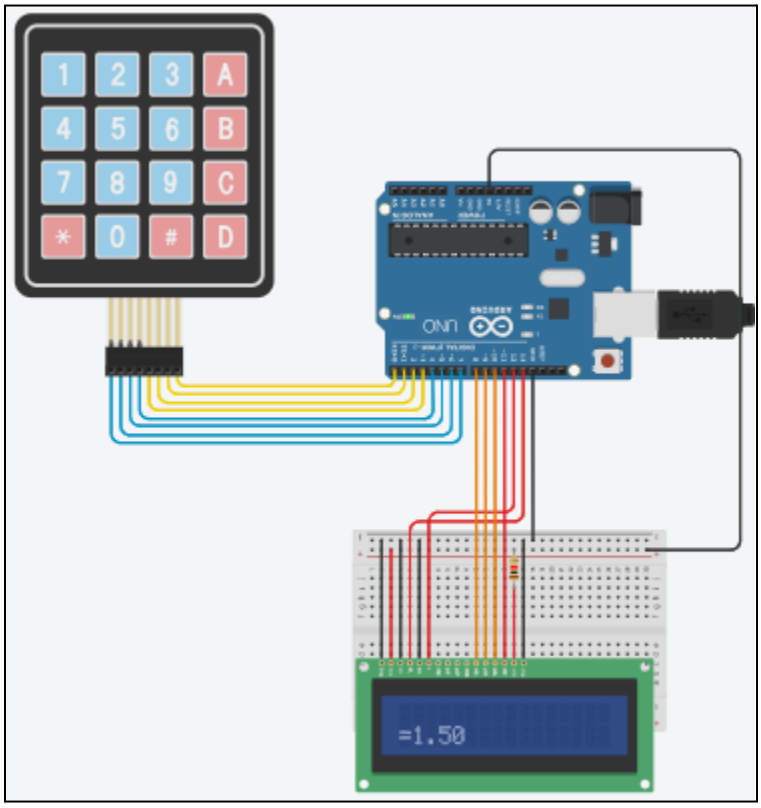
```
    posit =0;
    total = 0;
    lcd.clear();
    break;



  }


}

long secondNumber(){
  while(1){
        customKey  = customKeypad.getKey();
    if(customKey >= '0' && customKey <='9'){
        second = second*10 + (customKey - '0');
        lcd.setCursor(posit,0);
        lcd.print(second);
    }
    if(customKey == 'C'){
        total =0;
        first = 0;
              second =0;
        posit=0;
        lcd.clear();
        break;

    }
    if(customKey== '='){
        posit =total;
      lcd.clear();
      lcd.setCursor(0,1);
      lcd.print("=");
      break;
    }
  }
  return second;
}
```

**Output:**




1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
UNO
=10.00


1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
Row 1
UNO
6 /4

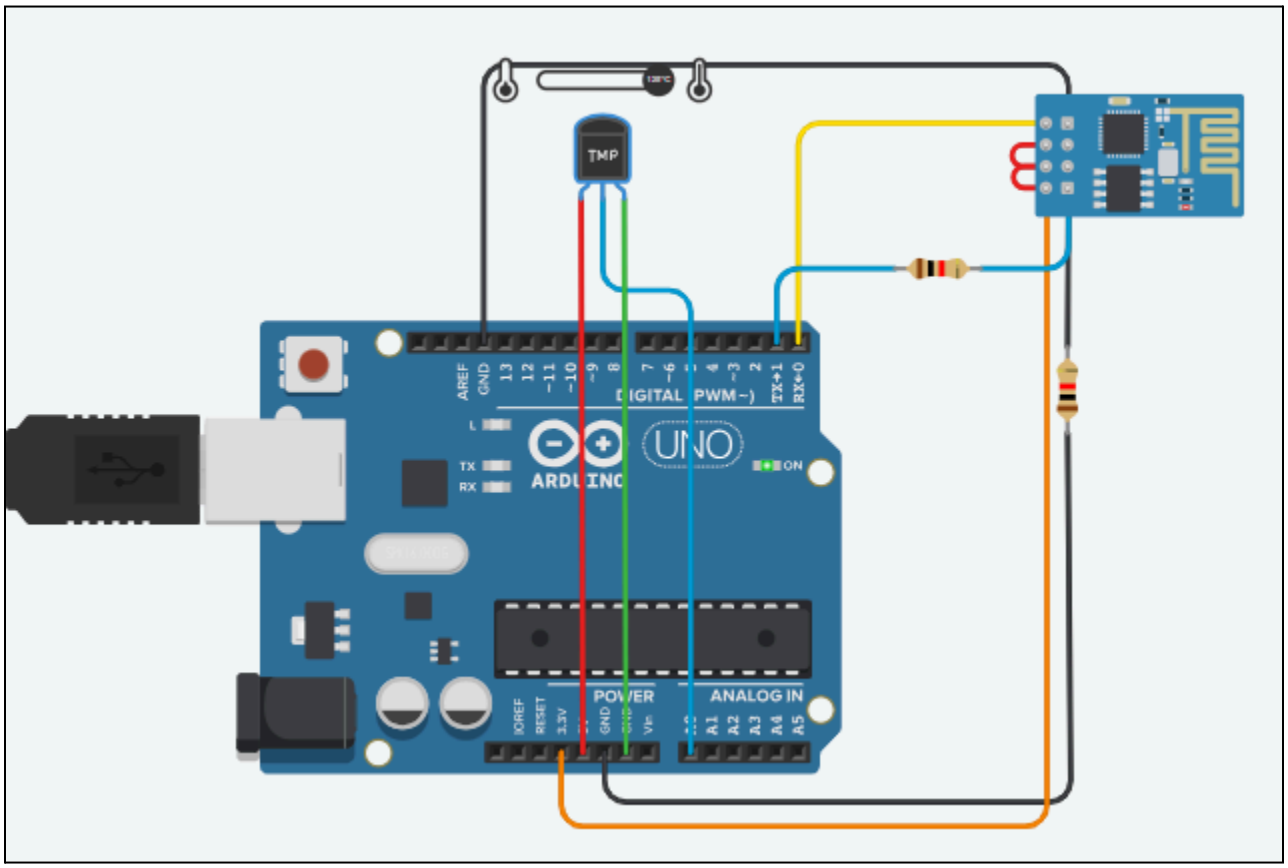

1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
=1.50

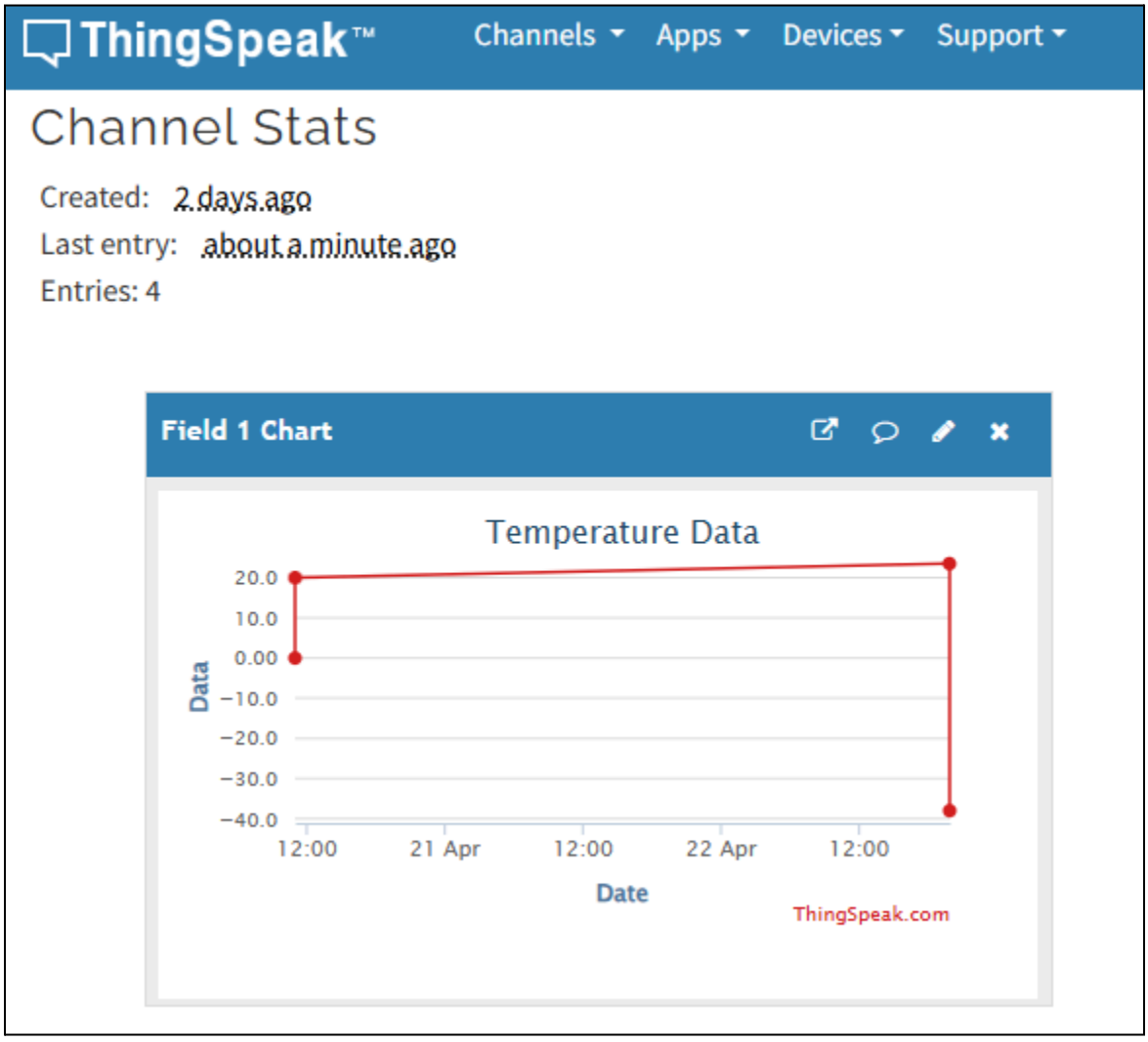
| | |
|---|---|
| **Program No:** | **21** |
| **Roll No :** | **1545** |
| **Title of Program :** | **To upload data on Thingspeak cloud manually.** |
| **Objective :** | **To send temperature data to thingspeak cloud and generate visualization** |

**Source Code:**

```
void setup()
{
  Serial.begin(115200);
  delay(1000);
  Serial.println("AT+CWJAP=\"Simulator Wifi\",\"\"\r\n"); delay(3000);
}
void loop()
{
  int sensorValue = analogRead(A0);
  float volt = (sensorValue/1020.0) * 4.9; //Volts float
  float tempC = (volt -0.5) * 100; //Celcius
  Serial.println(tempC);
  Serial.println("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80\r\n"); delay(5000);
  int len = 65; Serial.print("AT+CIPSEND=");
  Serial.println(len);
  delay(10);
  Serial.print("GET /update?api_key=OSVIQ23ON1D20JP9&field1=" +
String(tempC)
  +" HTTP/1.1\r\n");
  delay(100);
  Serial.println("AT+CIPCLOSE=0\r\n");
  delay(6000);
}
```

**Output:**


TMP
UNO
ON
TX
RX
POWER
ANALOG IN


ThingSpeak™
Channels
Apps
Devices
Support
Channel Stats
Created: 2 days ago
Last entry: about a minute ago
Entries: 4
Field 1 Chart
Temperature Data
20.0
10.0
0.00
−10.0
−20.0
−30.0
−40.0
Data
12:00
21 Apr
12:00
22 Apr
12:00
Date
ThingSpeak.com