Simple Calculator


```
void setup()
{
  Serial.begin(9600);
}
```

void setup(): This is the setup function in Arduino, which is called once when the board starts or is reset.

Serial.begin(9600);: This line initializes the serial communication at a baud rate of 9600. It allows communication between the Arduino board and a computer through the Serial Monitor.

```
void loop() {
  int num1, num2;
  char operatorSymbol;
```

void loop(): This is the main loop function in Arduino, which runs repeatedly after the setup() function.

int num1, num2;: These are integer variables to store the two numbers entered by the user.

char operatorSymbol;: This is a character variable to store the arithmetic operator entered by the user.

```
Serial.println("Enter the first number:");
while(!Serial.available());
num1 = Serial.parseInt();
```

Serial.println("Enter the first number:");: This line prints a message to the Serial Monitor asking the user to enter the first number.

while(!Serial.available());: This is a while loop that waits until data is available on the Serial port (until the user enters something and sends it to the Arduino).

num1 = Serial.parseInt();: This line reads the number entered by the user and stores it in the variable num1.

```
Serial.println("Enter an arithmetic operator (+, -, *, /):");
while(!Serial.available());
operatorSymbol = Serial.read();
Serial.println("Enter the second number:");
while(!Serial.available());
num2 = Serial.parseInt();
```

int result;: This is an integer variable to store the result of the arithmetic operation.

switch (operatorSymbol) { ... }: This is a switch statement that checks the value of operatorSymbol, which was entered by the user, and performs the corresponding arithmetic operation.

case '+': result = num1 + num2; break;: If the user entered '+', the code inside this case block adds num1 and num2 and stores the result in result.

case '-': result = num1 - num2; break;: If the user entered '-', the code inside this case block subtracts num2 from num1 and stores the result in result.

case '*': result = num1 * num2; break;: If the user entered '*', the code inside this case block multiplies num1 and num2 and stores the result in result.

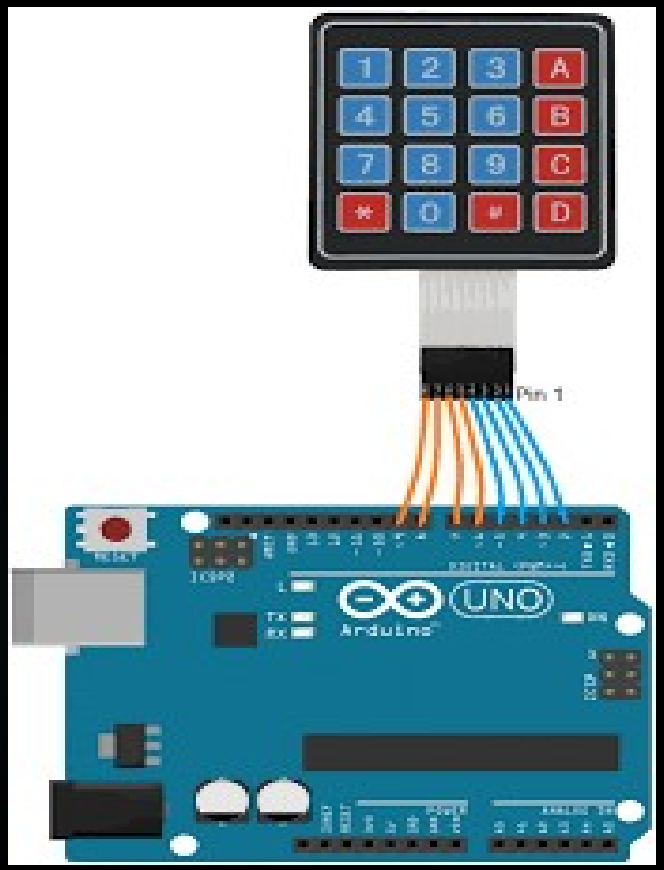
case '/': result = num1 / num2; break;: If the user entered '/', the code inside this case block divides num1 by num2 and stores the result in result.

default: Serial.println("Invalid operator!"); return;: If the user entered an invalid operator, the code inside this default block prints an error message to the Serial Monitor and exits the loop.

Serial.print("Result: ");: This line prints the text "Result: " to the Serial Monitor.

Serial.println(result);: This line prints the value of the result variable (the calculated result) to the Serial Monitor.

4X4 Keypad with Arduino



To create a simple calculator using an Arduino and a 4x4 hex keypad, you'll need to connect the keypad to the Arduino and write a program to read input from the keypad and perform basic arithmetic operations.

Components needed:

- Arduino board (e.g., Arduino UNO)
- 4x4 hex keypad
- Jumper wires

Wiring connections:

- **Connect the 4x4 hex keypad to the Arduino as follows:**
  - **Connect ROW0 to digital pin 2**
  - **Connect ROW1 to digital pin 3**
  - **Connect ROW2 to digital pin 4**
  - **Connect ROW3 to digital pin 5**
  - **Connect COL0 to digital pin 6**
  - **Connect COL1 to digital pin 7**
  - **Connect COL2 to digital pin 8**
  - **Connect COL3 to digital pin 9**

```
#include <Keypad.h>
```

The program uses the Keypad library to read input from the 4x4 hex keypad. When you press an operator (+, -, *, /) or the equals (=) key, it will perform the corresponding operation and display the result on the Serial Monitor.

Make sure to install the Keypad library in the Arduino IDE by going to "Sketch" -> "Include Library" -> "Manage Libraries" and then search for "Keypad" and install it.

```
//define the number of rows and columns of the keypad
const byte ROWS = 4;
const byte COLS = 4;
// Define the keypad layout, It maps each button's value to its corresponding position on the keypad. It represents a 4x4 matrix where each cell contains a character representing the button's value.
char keys[ROWS][COLS] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```

```
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
```

//These arrays define the connections of the keypad to the Arduino. The rowPins array contains the Arduino's digital pins connected to the keypad's rows, and the colPins array contains the Arduino's digital pins connected to the keypad's columns.

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

//This line initializes the Keypad object keypad with the defined keys array and the pin connections rowPins and colPins. It also specifies the number of rows and columns in the keypad.

```
void setup() { Serial.begin(9600); }
void loop() {
  char key = keypad.getKey();
  if (key) {
    handleKeyPress(key);
  }
}
```

It reads a character from the keypad using the getKey() function. If a key is pressed (the key variable is not NULL), it calls the handleKeyPress function to handle the keypress.

```
void handleKeyPress(char key) {
  if (key == 'A' || key == 'B' || key == 'C' || key == 'D') {
    // Handle operator keys (+, -, *, /)
    operatorSymbol = key;
    operatorPressed = true;
    operand1 = inputStr.toDouble();
    inputStr = "";
  } else if (key == '#') {
    // Handle the equals key (=)
    operand2 = inputStr.toDouble();
    performOperation();
    inputStr = String(operand1);
  } else {
    // Append the digit to the input string
    inputStr += key;
  }
  Serial.println(inputStr);
}
```

The handleKeyPress function processes the key pressed by the user. If the key is an operator key (A, B, C, D representing +, -, *, /), it sets the operatorSymbol, indicates that an operator is pressed (operatorPressed), converts the current input string to the first operand (operand1), and clears the inputStr.

If the key is the equals key (#), it converts the current input string to the second operand (operand2), calls the performOperation function to execute the arithmetic operation, and stores the result back into the inputStr.

If the key is a digit, it appends the digit to the inputStr.

```
void performOperation() {
  switch (operatorSymbol) {
    case 'A':
      operand1 += operand2;
      break;
    case 'B':
      operand1 -= operand2;
      break;
    case 'C':
      operand1 *= operand2;
      break;
    case 'D':
      operand1 /= operand2;
      break;
  }
  operatorPressed = false;
}
```

Finally, it prints the inputStr to the Serial Monitor.

The performOperation function is called when the equals key (#) is pressed. It performs the arithmetic operation based on the operatorSymbol value and updates the operand1 variable with the result. It then resets the operatorPressed flag to false.

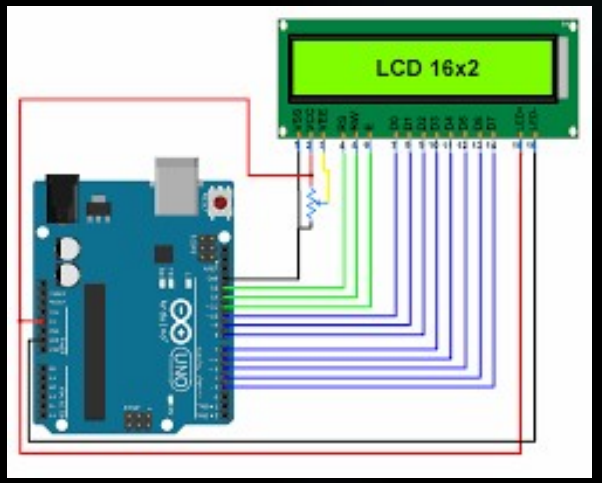
Interfacing 16X2 LCD Display

Interfacing an LCD (Liquid Crystal Display) with an Arduino Uno involves connecting the LCD module to the Arduino and writing code to control the display.

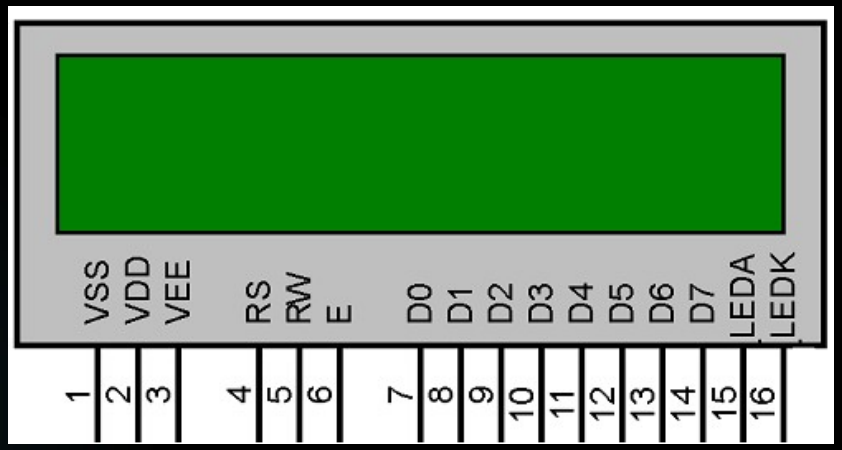
Below is a step-by-step explanation of how to interface an LCD with an Arduino Uno:

Components Required:

- Arduino Uno board
- LCD module (typically 16x2 or 20x4 characters)
- Potentiometer (for contrast adjustment)
- Breadboard and jumper wire



16X2 LCD Display Pin Diagram



Pin Connection:

- Connect the pins of the LCD module to the Arduino Uno as follows:
  - Connect VCC (LCD's power) to +5V on Arduino.
  - Connect GND (LCD's ground) to GND on Arduino.
  - Connect VEE or VO (LCD's contrast control) to the middle pin of the potentiometer.
  - Connect RW (LCD's Read/Write pin) to GND (to set the LCD in write mode).
  - Connect RS (LCD's Register Select pin) to a digital pin (e.g., Pin 12) on Arduino.
  - Connect E (LCD's Enable pin) to a digital pin (e.g., Pin 11) on Arduino.
  - Connect D4-D7 (LCD's data pins) to digital pins (e.g., Pins 5-8) on Arduino.

- **Potentiometer Setup:**
  - The potentiometer is used to adjust the contrast of the LCD. Connect its outer pins to +5V and GND, and the middle pin to VEE on the LCD.

- **Library Installation:**
  - You need the "LiquidCrystal" library to control the LCD. Install it via the Arduino IDE: **Sketch > Include Library > LiquidCrystal**.

Initialization:

In your Arduino sketch, include the LiquidCrystal library and initialize the LCD with the appropriate parameters:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 6, 7, 8); // RS, E, D4, D5, D6, D7
```

Setup Function:

- In the setup() function, set up the LCD:

```
void setup()
{
   lcd.begin(16, 2); // Set the LCD size (columns x rows)
   lcd.clear();      // Clear the display
   lcd.print("Hello, World!");
}
```

- Remember to connect everything correctly and upload the code to your Arduino Uno. With this setup, the LCD should display the text "Hello, World!" on the first line and "Arduino Uno" on the second line, with a delay and clearing effect in the loop as described.

- **Writing Text:**
  - You can use functions like lcd.print() and lcd.setCursor() to write text and position the cursor.
- **Other Functions:**
  - Explore the library for additional functions like scrolling, custom characters, and more.

Loop Function:

- In the loop() function, you can continuously update the display content:

```
void loop()
{
   lcd.setCursor(0, 1); // Set cursor to the second line
   lcd.print("Arduino Uno");
   delay(1000); // Wait for a second
   lcd.clear(); // Clear the display
   delay(500);  // Wait for half a second
}
```

Temperature Converter


```
void setup() {
  Serial.begin(9600);
}
void loop() {
  float celsius, fahrenheit;
  Serial.println("Enter temperature in Celsius:");
  while(!Serial.available()); // Wait for user input
  celsius = Serial.parseFloat(); // Read temperature in Celsius
  fahrenheit = (celsius * 9.0 / 5.0) + 32.0; // Celsius to Fahrenheit
conversion formula
  Serial.print("Temperature in Fahrenheit: ");
  Serial.println(fahrenheit);
}
```

Display the temperature conversion results on a 16x2 LCD:


```
#include <LiquidCrystal.h>

// Initialize the LCD with the appropriate pins
LiquidCrystal lcd(12, 11, 5, 6, 7, 8);

void setup() {
  lcd.begin(16, 2); // Initialize  LCD:16 col and 2 rows
  Serial.begin(9600); // Initialize serial communication
}

void loop() {
  float celsius, fahrenheit;

  lcd.clear(); // Clear the LCD screen
  lcd.setCursor(0, 0); // Set the cursor to the first line
  lcd.print("Enter temp (C):");
```

```
while (!Serial.available()); // Wait for user input
  celsius = Serial.parseFloat(); // Read temp in Celsius

  fahrenheit = (celsius * 9.0 / 5.0) + 32.0; // Cel to Fahr

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temp (C): ");
  lcd.print(celsius);
  lcd.setCursor(0, 1); // Set the cursor to the secondline
  lcd.print("Temp (F): ");
  lcd.print(fahrenheit);
  delay(5000); // Wait for 5 seconds before clearing
the display
}
```

LED Brightness Control with Potentiometer


```
const int potPin = A0; // Analog input pin for the potentiometer
const int ledPin = 9;  // PWM output pin for the LED
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  int potValue = analogRead(potPin); // Read potentiometer value (0-1023)
  int brightness = map(potValue, 0, 1023, 0, 255); // Map the value to LED
brightness range (0-255)
  analogWrite(ledPin, brightness); // Set LED brightness using PWM
  Serial.print("Potentiometer Value: ");
  Serial.print(potValue);
  Serial.print(", Brightness: ");
  Serial.println(brightness);
  delay(100); // Small delay for stability
}
```

Display the corresponding brightness value on LCD

Do it yourself

Ultrasonic sensor to measure distances and display the distance on an LCD or through the serial monitor.

HC SR-04

- The HC-SR04 sensor consists of an ultrasonic transmitter (transducer) and a receiver (transducer).
- The transmitter emits a short burst of ultrasonic waves, which are sound waves with frequencies above the upper limit of human hearing (typically above 20 kHz).
- These ultrasonic waves travel through the air until they encounter an object in their path.
- **Distance Calculation:**

Distance = (Speed of Sound * Time) / 2.

Pins of HC SR04

- **VCC (Voltage Supply):**
  - The VCC pin is used to provide power to the sensor.
  - Connect this pin to the +5V or +3.3V pin on your microcontroller board (such as Arduino) to supply the required operating voltage.
- **Trig (Trigger) Pin:**
  - The Trig pin is used to initiate the ultrasonic pulse transmission.
  - To start the measurement, you need to send a high-to-low pulse (at least 10 microseconds long) to this pin.
  - This pulse prompts the sensor to emit an ultrasonic burst.
- **Echo Pin:**
  - The Echo pin is used to receive the ultrasonic waves that bounce back after hitting an object.
  - The sensor sends out the ultrasonic pulse and then waits for the echo signal.
  - The Echo pin goes high when the sensor receives the reflected signal and stays high for a duration proportional to the time it takes for the signal to travel to the object and back.
- **GND (Ground):**
  - The GND pin is the ground reference for the sensor.
  - Connect this pin to the GND (ground) pin on your microcontroller board.

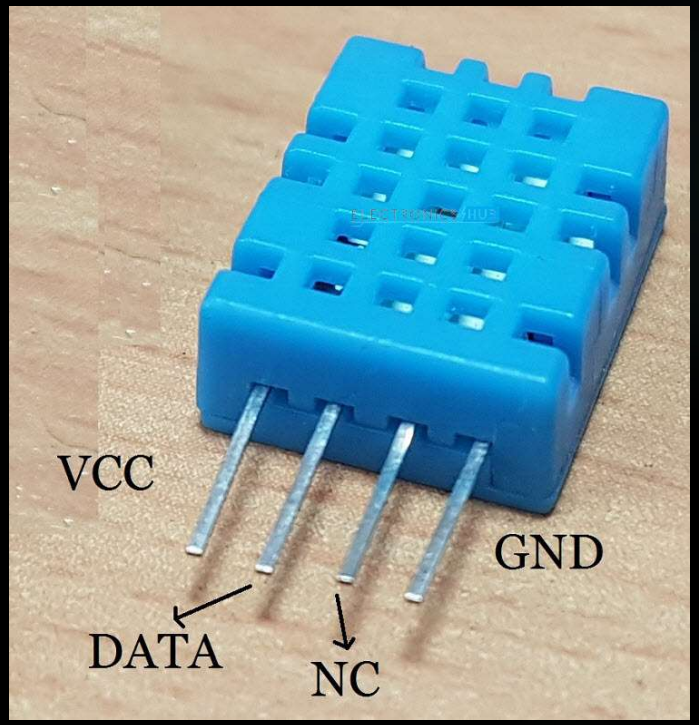
Step-by-step procedure to interface the HC-SR04 ultrasonic distance sensor with an Arduino:

- **Components Needed:**
  - Arduino board (e.g., Arduino Uno)
  - HC-SR04 ultrasonic sensor
  - Breadboard and jumper wires
- **Wiring:**
  - Connect VCC on the HC-SR04 to +5V on the Arduino.
  - Connect GND on the HC-SR04 to GND on the Arduino.
  - Connect TRIG on the HC-SR04 to a digital pin (e.g., Pin 9) on the Arduino.
  - Connect ECHO on the HC-SR04 to another digital pin (e.g., Pin 10) on the Arduino.

```
const int trigPin = 9;
const int echoPin = 10;

void setup() {
  Serial.begin(9600);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  int distance = duration * 0.0343 / 2; // Calculate distance in centimeters

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(1000); // Wait before the next measurement
}
```

DHT11 sensor

- The DHT11 sensor is a widely used digital temperature and humidity sensor that is commonly used in various electronics projects and applications.
- It's a simple sensor that provides accurate temperature and humidity readings.
- **Sensing Elements:** The DHT11 sensor contains two main sensing elements: a humidity sensor and a thermistor (temperature sensor).



Humidity Sensing:

- The humidity sensor in the DHT11 is based on the capacitive sensing principle.
- It consists of two conductive plates with a moisture-absorbing material between them.
- As the surrounding air's humidity changes, the moisture content of the material changes, causing a change in capacitance between the plates.
- The sensor's electronics measure this change in capacitance and convert it into a digital humidity value.

Temperature Sensing:

- The temperature sensing element in the DHT11 is a thermistor—a type of resistor whose resistance changes with temperature.
- The thermistor's resistance decreases as the temperature increases and vice versa.
- The sensor circuitry measures this resistance change and converts it into a digital temperature value.

Signal Processing and Communication:

- The DHT11 sensor has an integrated microcontroller that processes the signals from the humidity and temperature sensing elements.
- It converts the analog sensor outputs into digital values, and these values are then packaged into a simple digital signal format.

Timing and Data Format:

- The DHT11 sensor sends data in a specific format: it starts by sending a low-to-high signal to signal the beginning of data transmission.
- It then sends 40 bits of data (8 bits for integral humidity, 8 bits for decimal humidity, 8 bits for integral temperature, 8 bits for decimal temperature, and 8 bits for the checksum).

Example

Consider the data received from the DHT11 Sensor is

00100101 00000000 00011001 00000000 00111110.

This data can be separated based on the above mentioned structure as follows

00100101	00000000	00011001	00000000	00111110
High Humidity	Low Humidity	High Temperature	Low Temperature	Checksum (Parity)

Correctness of data received

- In order to check whether the received data is correct or not, we need to perform a small calculation.
- Add all the integral and decimals values of RH and Temperature and check whether the sum is equal to the checksum value i.e. the last 8 – bit data.

00100101 + 00000000 + 00011001 + 00000000 = 00111110

- This value is same as checksum and hence the received data is valid.

- Now to get the RH and Temperature values, just convert the binary data to decimal data.

RH = Decimal of 00100101 = 37%

Temperature = Decimal of 00011001 = 25^0C

Digital Output:

- The DHT11 sensor communicates with the outside world using a single-wire digital communication protocol.
- It sends out a series of pulses to transmit data. Each bit of data is represented by the duration of a specific pulse.

Temperature Alert System


```
const int temperatureThreshold = 30;
const int fanPin = 9;
const int buzzerPin = 10;
void setup() {
  pinMode(fanPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  int temperature = readTemperature();
  if (temperature > temperatureThreshold &&
temperature < 100)
 {
    digitalWrite(fanPin, HIGH);
    digitalWrite(buzzerPin, HIGH);
    Serial.println("Temperature is too high! Cooling
and sounding the alarm.");
  }
else
{
digitalWrite(fanPin, LOW);
    digitalWrite(buzzerPin, LOW);
  }
  delay(1000); // Delay to avoid rapid checks
}
int readTemperature() {
  // Replace this function with your actual
temperature reading code
  // For simplicity, we return a constant value here.
  return 25;
}
```

- pinMode(buttonPin, INPUT_PULLUP);: This line configures the buttonPin as an input pin with the internal pull-up resistor enabled. The pull-up resistor ensures that the button reads HIGH when not pressed and LOW when pressed.

Reading Analog Input


```
int analogPin = A0;
int threshold = 500;

void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(analogPin);

  while (sensorValue > threshold) {
    Serial.println("Object detected!");
    delay(1000);

    // Read the sensor value again
    sensorValue = analogRead(analogPin);
  }
}
```