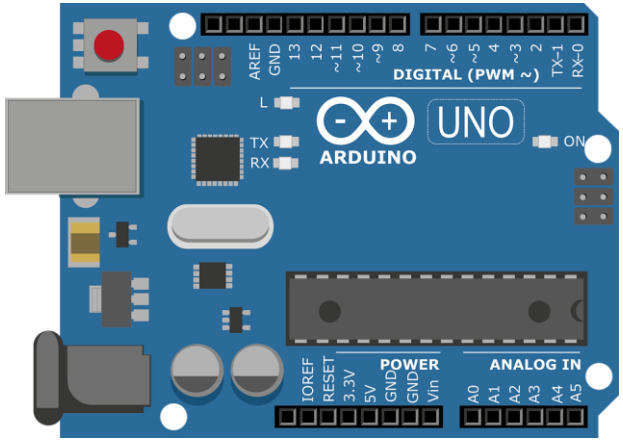
MICROCONTROLADORES
Arduino



Lendo um Botão

- Lendo um botão
  - Para ler um botão basta ligá-lo em uma porta digital.
  - Para que um circuito com botão funcione adequadamente, ou seja, sem ruídos, é necessário o uso de resistores *pull-down* ou *pull-up*.
  - Os resistores *pull-down* e *pull-up* garantem que os níveis lógicos estarão próximos às tensões esperadas.

Resistor Pull-Up e Pull-Down

Em praticamente todo projeto, é necessário que alguma leitura digital seja feita. E é extremamente ruim ler valores errados, e isso pode ocorrer por causa do pino não estar conectado a nada em determinado momento. Portanto, é necessário usar um ***resistor pull-up ou pull-down*** para resolver o problema.

Resistor Pull-Up e Pull-Down

O resistor pull-up ou pull-down é basicamente um resistor que fica ligado junto ao sinal que você deseja ler. Ele serve para garantir que um determinado sinal, ou uma determinada tensão, seja lida enquanto o pino não recebe nenhum sinal.

Um conceito importante é o de “pino flutuando”, que é o estado que o pino fica sem receber nada. Isso ocorre, pois o pino não está ligado a nada e não há como saber qual sinal ele está recebendo, se é sinal alto(2~5v) ou sinal baixo (0~1v).

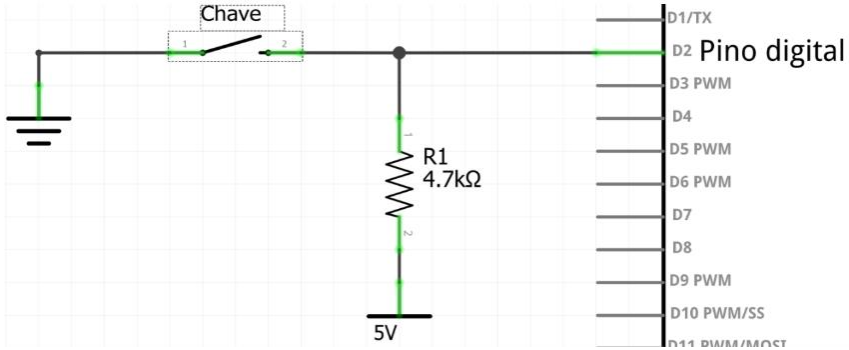
Resistor Pull-Up e Pull-Down

Observe que, enquanto a chave está aberta, não tem como sabe qual valor está sendo lido no pino D2.

Resistor Pull-Up e Pull-Down

- **Pull-up**

O resistor ***pull-up*** garante que o sinal lido seja de nível alto (5v) enquanto o botão ou a chave não forem pressionados. Ele faz isso da seguinte forma:

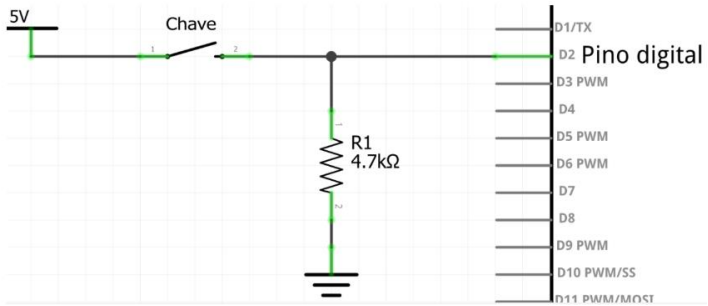


Enquanto a chave está aberta, a tensão em cima do pino é a tensão de 5v menos a tensão em cima do resistor, que é de aproximadamente 5v. Ou seja, o pino reconhece sinal de nível alto. Porém, quando a chave se fecha, o terra (GND) fica ligado diretamente no pino, e isso faz com que D2 leia sinal de nível baixo. E é claro, haverá uma corrente fluindo do 5v para o terra.

Resistor Pull-Up e Pull-Down

- **Pull-down**

O funcionamento é igual ao resistor pull-up, porém o que muda é a forma como a chave e o resistor são alimentados. Assim, o resistor ***pull-down*** garante que o sinal lido seja de nível baixo (GND) enquanto o botão ou a chave não forem pressionados.

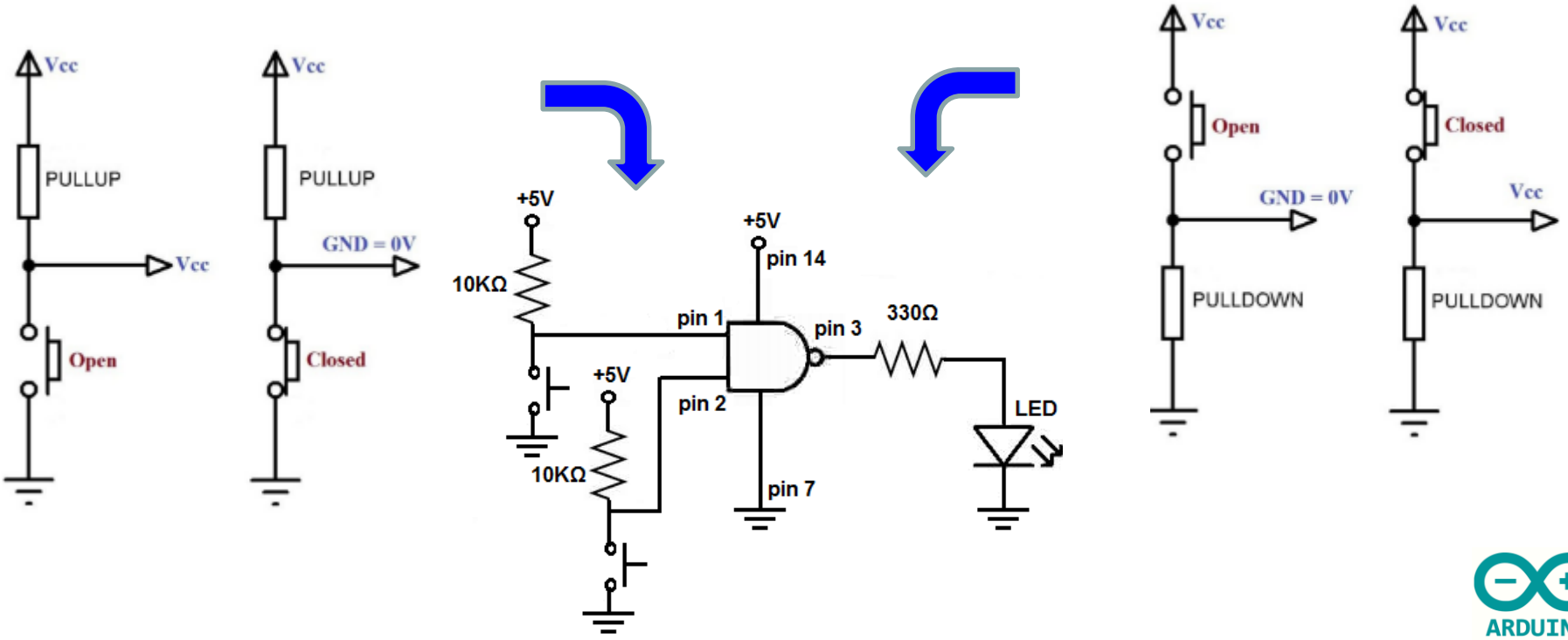


A ideia é a mesma: enquanto a chave está aberta, a tensão em cima do pino será 0v(GND). E, dessa forma, o pino reconhece sinal de nível baixo. Mas quando a chave se fecha, o 5v fica ligado diretamente no pino, e isso faz com que D2 leia sinal de nível alto. E da mesma forma, existe uma corrente fluindo do 5v para o terra.

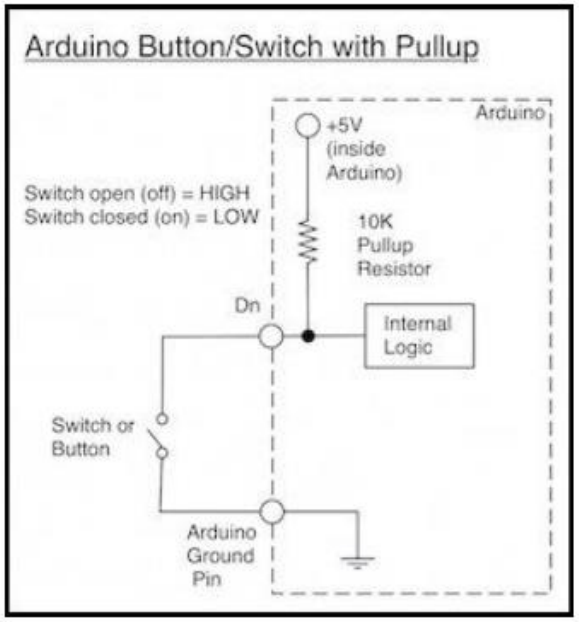
Resistor Pull-Up e Pull-Down

⇘**Resistor de pull-up:** Assegura em uma entrada (que pode ser compartilhada) de uma porta lógica o nível lógico 1.

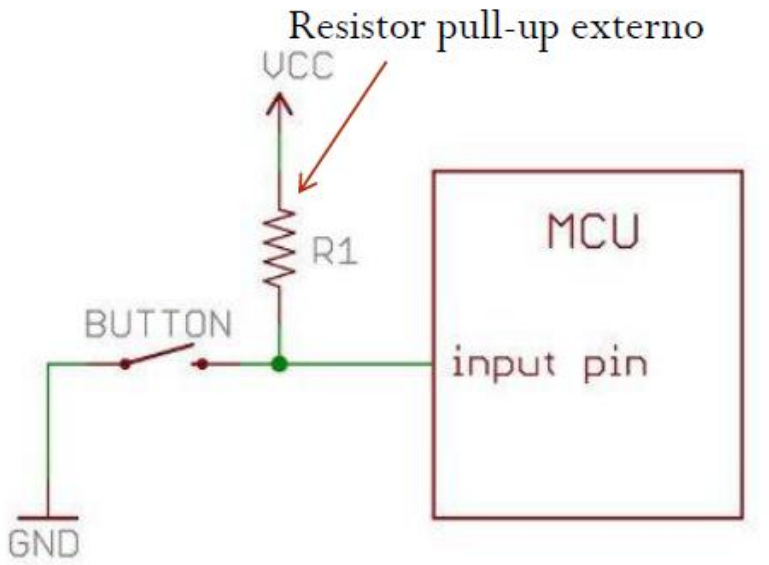
⇘**Resistor de pull-down:** Assegura em uma entrada (que pode ser compartilhada) de uma porta lógica o nível lógico 0.



Resistor Pull-Up e Pull-Down



a) pull-up interno
(necessita habilitar dentro da função setup() utilizando digitalWrite())



b) pull-up externo
(necessita conectar R1=10K Ohms, mas não necessita habilitar na função setup())

Leitura de Botão

- Lendo um botão
  - Para ler um botão basta ligá-lo em uma porta digital.
  - Para que um circuito com botão funcione adequadamente, ou seja, sem ruídos, é necessário o uso de resistores *pull-down* ou *pull-up*.
  - Os resistores *pull-down* e *pull-up* garantem que os níveis lógicos estarão próximos às tensões esperadas.

Leitura de Botão

- Exemplo: ativando o resistor *pull-up* de uma porta digital
  - Nota:
    - O Arduino possui uma constante chamada *INPUT_PULLUP* que define que a porta será de entrada e o resistor *pull-up* da mesma será ativado.
    - Exemplo:

```
void setup()
{
  pinMode(10, INPUT_PULLUP);
}
```

Define a porta 10 como entrada de dados e ativa o resistor pull-up.

Leitura de Botão

```
1) void setup()
2) {
3)          pinMode(3, OUTPUT);
4)          pinMode(10, INPUT_PULLUP);
5)   }
5) void loop()
6)          {
7)               if (!digitalRead(10)==0) //se o botão for acionado vai a 1
8)                    Comando de seleção simples
9)               {
10)                    digitalWrite(3,HIGH);
11)                    delay(1000);
12)                    digitalWrite(3,LOW);
13)                    delay(1000);
14)               }
16)         }
```

Comando de Seleção

- Comando de seleção simples
  - Um comando de seleção simples avalia uma condição, ou expressão, para executar uma ação ou conjunto de ações.
  - No Arduino o comando de seleção simples é:

```
if (expr) {
   cmd
}
```

  - onde:
    - *expr* – representa uma expressão a ser avaliada que pode ser do tipo lógica, relacional ou aritmética. O resultado da avaliação de uma expressão é sempre um valor lógico.
    - *cmd* – comando(s) a ser executado.

- Comentários
  - Muitas vezes é importante comentar alguma parte do código do programa.
  - Existem duas maneiras de adicionar comentários a um programa em Arduino.
  - A primeira é usando //, como no exemplo abaixo:
    - // Este é um comentário de linha
  - A segunda é usando /* */, como no exemplo abaixo:
    - /* Este é um comentário de bloco. Permite acrescentar comentários com mais de uma linha */
  - **Nota:**
    - Quando o programa é compilado os comentários são automaticamente suprimidos do arquivo executável, aquele que será gravado na placa do Arduino.

Comando de Seleção

```
1  //O Botão aciona o Led por dois segundos
2
3  void setup()
4  {
5
6      pinMode(3, OUTPUT);
7      pinMode(10, INPUT_PULLUP);
8  }
9
10  void loop()
11 {
12
13     if (!digitalRead(10)==0) //se o botão for acionado vai a 1
14
15         {
16         digitalWrite(3,HIGH);
17         delay(1000);
18         digitalWrite(3,LOW);
19         delay(1000);
20         }
21 }
22
```

Comando de Seleção

- Comando de seleção composta
  - Um comando de seleção composta é complementar ao comando de seleção simples.
  - O objetivo é executar um comando mesmo que a expressão avaliada pelo comando *if (expr)* retorne um valor falso.
  - No Arduino o comando de seleção composta é:

```
if (expr) {
    cmd;
}
else {
    cmd;
}
```

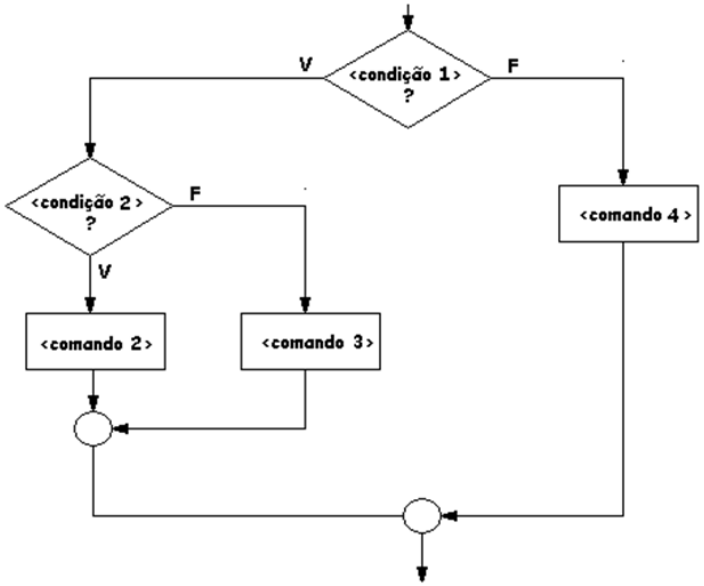
  - onde:
    - ***expr*** – representa uma expressão a ser avaliada que pode ser do tipo lógica, relacional ou aritmética. O resultado da avaliação de uma expressão é sempre um valor lógico.
    - ***cmd*** – comando(s) a ser executado.

Comando de Seleção

```
24 //Comando if-else
25
26 #define Led_VM 3 // Dizendo que o LED VERMELHO está ligado no pino 3
27 #define BotaoLiga 10 //Dizendo que o BotaoLiga está ligado no pino 7
28
29
30 void setup()
31 {
32   pinMode(BotaoLiga,INPUT); //utiliza o pullup externo
33   pinMode(Led_VM,OUTPUT);
34
35 }
36
37 void loop()
38 {
39   if (digitalRead(BotaoLiga)){
40         digitalWrite(Led_VM,HIGH);
41   }
42
43     else
44     {
45         digitalWrite(Led_VM, LOW);
46
47     }
48 }
```

Encadeamento de comando if-else

Problemas em que é necessário se estabelecerem verificações de condições sucessivas(teste, comparações), onde uma determinada ação poderá ser executada se um conjunto anterior de instruções ou condições for satisfeito. Isto implica em utilizar uma condição dentro de outra condição



Fluxograma

```
Início
    se< Condição 1 > então
        se< Condição 2 >
então
            <comando 2>
        senão
         < Comando 3 >
        Fim se
    senão
     <comando 4>
    Fim se
Fim
```

Pseudocódigo

Comando de Seleção

- Comando de seleção de múltipla escolha
  - Na seleção de múltipla escolha é possível avaliar mais de um valor.
  - No Arduino o comando de seleção de múltipla escolha é:

```
switch (valor) {
    case x: cmd1;
            break;
    case y: cmd2;
            break;
    default:  cmd;
}
```

  - onde:
    - ***valor*** – é um dado a ser avaliado. É representado por uma variável de memória.
    - cmd_x – comando a ser executado.
    - ***case***– indica a opção a ser executada.
    - ***default*** – comando padrão que deverá ser executado se nenhuma outra escolha (***case***) tiver sido selecionada.

Operadores

- Operadores
  - Em uma linguagem de programação existem vários operadores que permitem operações do tipo:
    - Aritmética
    - Relacional
    - Lógica
    - Composta

Operadores

- Operadores aritméticos

Símbolo	Função
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão inteira)

Operadores

- Operadores relacionais

Símbolo	Função
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
==	Igual
!=	Diferente

Operadores

- Operadores lógicos

Símbolo	Função
&&	E (and)
||	OU (or)
!	Não (not)

Operadores

- Operadores compostos

Símbolo	Função
++	Incremento
--	Decremento
+=	Adição com atribuição
-=	Subtração com atribuição
*=	Multiplicação com atribuição
/=	Divisão com atribuição

Funções

Tem como objetivo controlar a placa Arduino e executar cálculos

analogRead ()

Lê o valor do pino analógico especificado. As placas Arduino contêm um conversor analógico-digital de 10 bits e multicanal. Isso significa que ele mapeará as tensões de entrada entre 0 e a tensão de operação (5V ou 3,3V) em valores inteiros entre 0 e 1023. Em uma UNO Arduino, por exemplo, isso gera uma resolução entre leituras de: 5 volts / 1024 unidades ou 0,0049 volts (4,9 mV) por unidade.

```
int analogPin = A3; //terminal do meio do potenciometro conectado ao pino analógico 3
                    // terminais extremos ligados ao terra e ao +5V
int val = 0; // variável para armazenar o valor lido

void setup() {
Serial.begin(9600); // configuração do serial
 }

void loop() {
 val = analogRead(analogPin); // lê o pino de entrada
Serial.println(val); // mostra o valor lido na tela
 }
```

Funções

Tem como objetivo controlar a placa Arduino e executar cálculos

analogWrite ()

Grava um valor analógico (onda PWM) em um pino. Pode ser usado para acender um LED com brilho variável ou acionar um motor em várias velocidades. Após uma chamada para analogWrite(), o pino gerará uma onda retangular constante do ciclo de serviço especificado até a próxima chamada para analogWrite()(ou uma chamada para digitalRead()ou digitalWrite()) no mesmo pino.

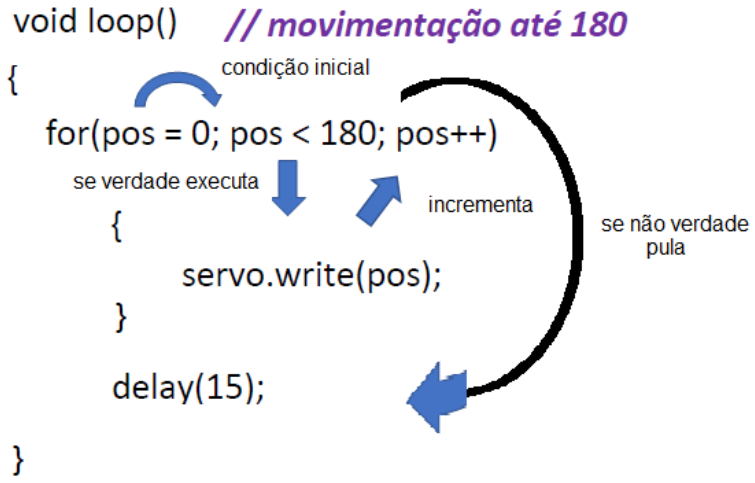
```
int ledPin = 9; // LED conectado ao pino digital 9
int analogPin = 3; // potenciômetro onectado ao pino analógico 3
int val = 0; // variável a ser armazenada com o valor lido

 void setup() {
pinMode(ledPin, OUTPUT); // configura o pino 9 como saída
}
void loop()
{ val = analogRead(analogPin); // lê o valor do pino 3 e armazena em val
analogWrite(ledPin, val / 4); // o valor de analogRead varia entre 0 e 1023,
                              //o valor de analogWrite varia 0 a 255
 }
```

Estrutura de Controle Laços - Instrução For

A sentença ***for*** é utilizada para repetir um bloco de código delimitado por chaves. Um contador com incremento normalmente é usado para controlar e finalizar o loop. A sentença ***for*** é útil para qualquer operação repetitiva, e é frequentemente usada com arrays (matrizes) para operar em conjuntos de dados ou de pinos.

for (int variavel=<valor inicial>; <condição final>; <passo>)

⇨ **Exemplo:**



Instrução map()

Realiza o mapeamento de um número de uma faixa para outra, conforme sintaxe

map(value, fromLow, fromHigh, toLow, toHigh)

- ***value***: o valor a ser mapeado
- ***fromLow***: o menor valor da faixa corrente
- ***fromHigh***: o maior valor da faixa corrente
- ***toLow***: o menor valor da faixa de destino
- ***toHigh***: o maior valor da faixa de destino

Instrução map()

Realiza o mapeamento de um número de uma faixa para outra, conforme sintaxe

map(value, fromLow, fromHigh, toLow, toHigh)

Sintaxe: map(***pos***, ***0, 1023***, ***0, 180***);

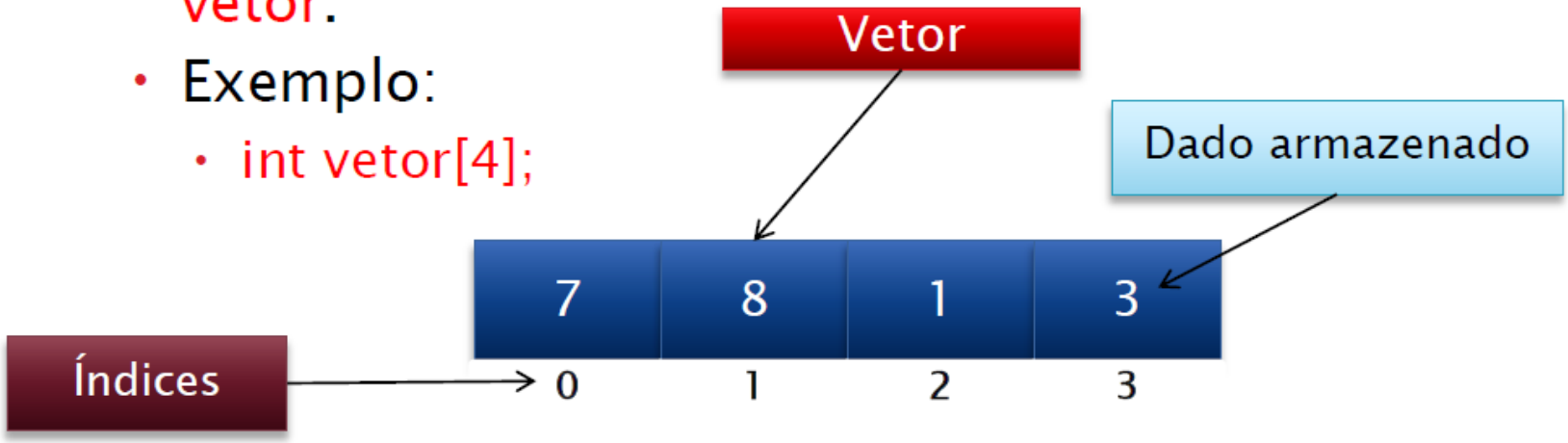
⇨ os valores de 0 até 1023, serão mapeados entre 0 e 180.

Vetores e Matrizes

- Vetores e matrizes
  - Uma variável escalar pode armazenar muitos valores ao longo da execução do programa, porém não ao mesmo tempo.
  - Existem variáveis que podem armazenar mais de um valor ao mesmo tempo. Essas variáveis são conhecidas como "variáveis compostas homogêneas".
  - No Arduino é possível trabalhar com dois tipos de variáveis compostas homogêneas, vetores e matrizes.

Vetores e Matrizes

- Vetores e matrizes
  - Vetor
    - A declaração de um vetor é feita da mesma maneira que uma variável escalar, entretanto é necessário definir a quantidade de itens do vetor.
    - Exemplo:
      - int vetor[4];



- Vetor com 4 (quatro) elementos do tipo inteiro.

Vetores e Matrizes

- Vetores e matrizes
  - Vetor

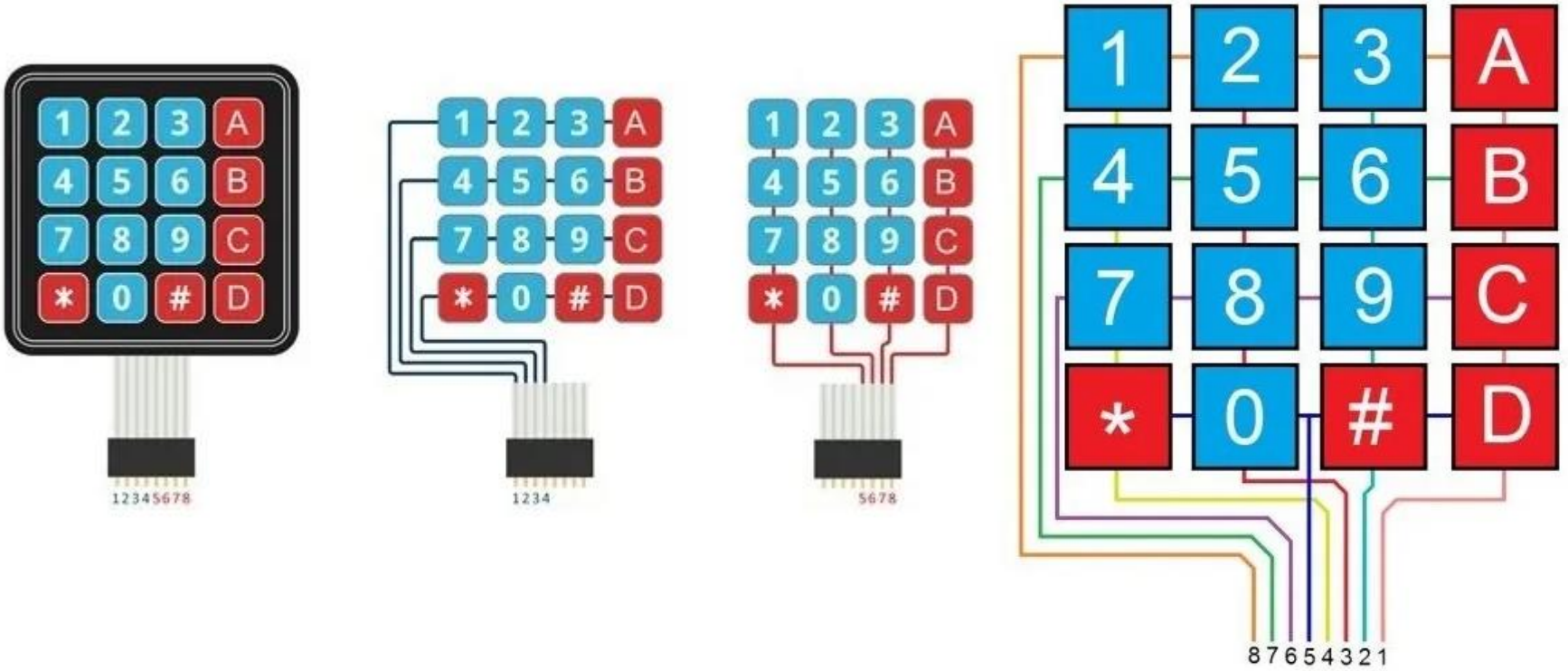
- Para atribuir um valor a uma determinada posição do vetor, basta usar o índice, ou seja, a posição onde o valor será armazenado no vetor.
- Exemplo:
  - vetor[0] = 7;
  - Atribui o valor 7 a posição 0 do vetor.

Vetores e Matrizes

- Vetores e matrizes
  - Vetor
    - Exemplo: acendendo e apagando leds cujas portas estão definidas em um vetor

```
int leds[5] = {2, 3, 4, 5, 6};  // Define as portas onde estão os leds

void setup()
{
  int i;
  for (i = 0; i < 5; i++) {
    pinMode(leds[i], OUTPUT);  // Define as portas como saída
  }
}

void loop()
{
  int i;
  for (i = 0; i < 5; i++) {
    digitalWrite(leds[i], HIGH);  // Acende os leds
    delay(1000);
  }
  for (i = 0; i < 5; i++) {
    digitalWrite(leds[i], LOW);  // Apaga os leds
    delay(1000);
  }
}
```

Teclado de Membrana Matricial



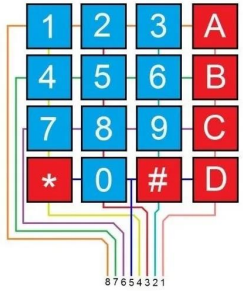
Teclado de Membrana Matricial

```
#include <Keypad.h> // BIBLIOTECA PARA O FUNCIONAMENTO DO TECLADO DE MEMBRANA
```

Insere a biblioteca da do teclado

```
char* senha = "123"; //SENHA CORRETA PARA DESTRANCAR A FECHADURA
int posicao = 0; //VARIÁVEL PARA LEITURA DE POSIÇÃO DA TECLA PRESSIONADA
const byte LINHAS = 4; //NUMERO DE LINHAS DO TECLADO
const byte COLUNAS = 4; //NUMERO DE COLUNAS DO TECLADO
char Teclas[LINHAS][COLUNAS] =
{
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};  //DECLARAÇÃO DOS NUMEROS, LETRAS E CARACTERES DO TECLADO
```

Configura a Matriz que representará o tipo de membrana



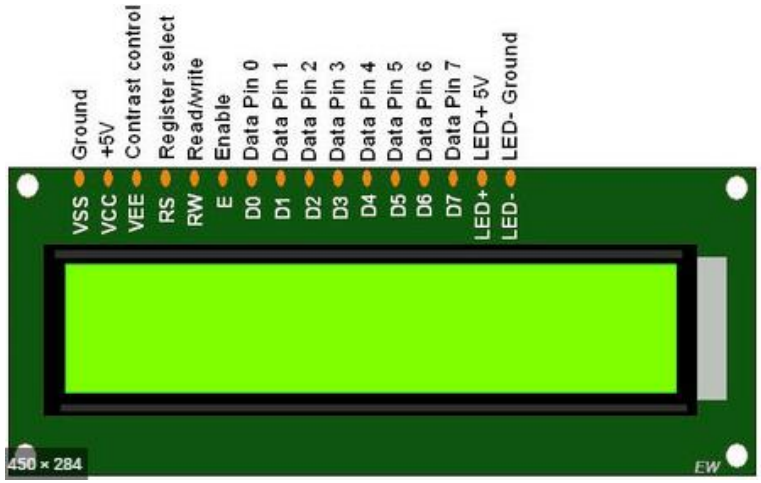

```
byte PinosLinhas[LINHAS] = { 8, 7, 6, 9 }; // PINOS DE CONEXAO DAS LINHAS DO TECLADO
byte PinosColunas[COLUNAS] = { 5, 4, 3, 2 }; //PINOS DE CONEXAO DAS COLUNAS DO TECLADO
Keypad Teclado = Keypad( makeKeymap(Teclas), PinosLinhas, PinosColunas, LINHAS, COLUNAS );//AS VARIAVEIS PinosLinhas E PinosColunas
 //RECEBERÃO O VALOR DE LEITURA DOS PINOS DAS LINHAS E COLUNAS RESPECTIVAMENTE
```

Configura quais pinos serão utilizados para receber a Matriz da membrana


```
char key = Teclado.getKey(); //LEITURA DAS TECLAS PRESSIONADAS
```

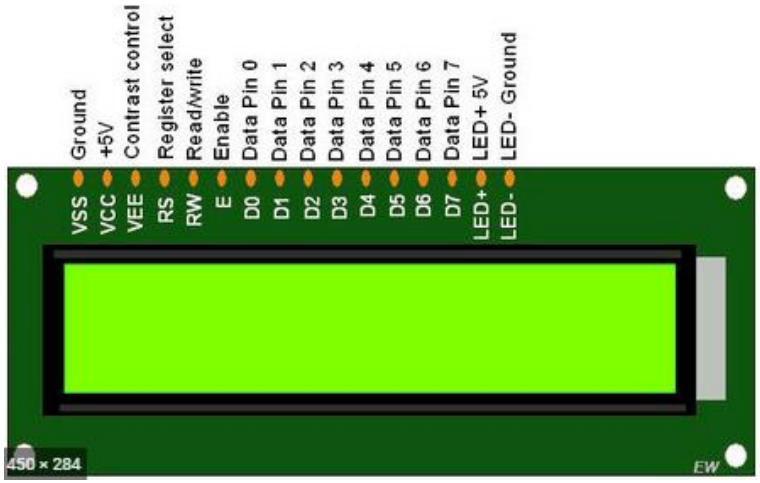
Armazena na variável "char key" o resultado da tecla identificada pelo cruzamento das linhas e colunas

LCD – Liquid Crystal Display



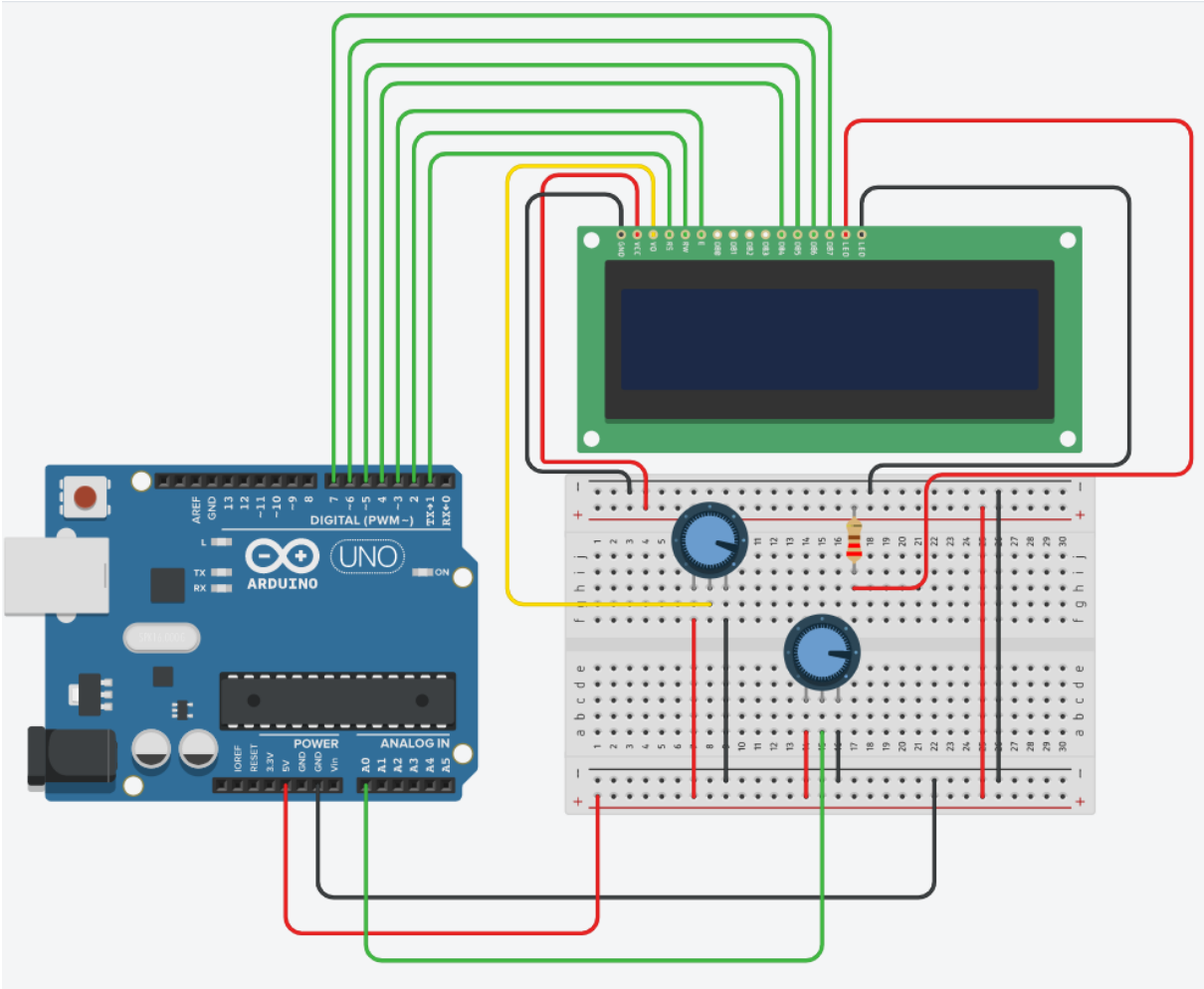
- pino 1 – **VSS** – Pino de alimentação (zero volts – GND)
- pino 2 – **VDD** – Pino de alimentação de +5V
- pino 3 – **VO** – Pino de ajuste do contraste do LCD
  - depende da tensão aplicada (ajustável)
- pino 4 – **RS** – Seleção de Comandos (nível 0) ou Dados (nível 1)
- pino 5 – **R/W** – Read(leitura – nível 1) / Write (escrita – nível 0)
- pino 6 – **E** – Enable (Ativa o display com nível 1 ou Desativa com nível 0)
- pino 7 – **D0** – data bit 0 (usado na interface de 8 bits)
- pino 8 – **D1** – data bit 1 (usado na interface de 8 bits)
- pino 9 – **D2** – data bit 2 (usado na interface de 8 bits)
- pino 10 – **D3** – data bit 3 (usado na interface de 8 bits)
- pino 11 – **D4** – data bit 4 (usado na interface de 4 e 8 bits)
- pino 12 – **D5** – data bit 5 (usado na interface de 4 e 8 bits)
- pino 13 – **D6** – data bit 6 (usado na interface de 4 e 8 bits)
- pino 14 – **D7** – data bit 7 (usado na interface de 4 e 8 bits)
- pino 15 – **A** – Anodo do LED de iluminação (+5V CC)
- pino 16 – **K** – Catodo do LED de iluminação (GND)

LCD – Liquid Crystal Display



Pino	Símbolo	Função
1	VSS	GND(Alimentação)
2	VDD	5V(Alimentação)
3	V0	Ajuste de Contraste
4	RS	Habilida/Desabilita Seletor de Registrador
5	R/W	Leitura/Escrita
6	E	Habilita Escrita no LCD
7	DB0	Dado
8	DB1	Dado
9	DB2	Dado
10	DB3	Dado
11	DB4	Dado
12	DB5	Dado
13	DB6	Dado
14	DB7	Dado
15	A	5V(Backlight)
16	K	GND(BackLight)

LCD – Liquid Crystal Display



LCD – Liquid Crystal Display

Parametrização para configurar o LCD

```
 //CONVERSOR A/D - Sensor de Temperatura TMP36

#include <LiquidCrystal.h>
float Temperatura;
#define Sensor A0
#define rs 1
#define rw 2
#define en 3
#define d4 4
#define d5 5
#define d6 6
#define d7 7
LiquidCrystal lcd(rs,rw,en,d4,d5,d6,d7);//Configura os pinos do Arduino para se comunicar com o LCD
```

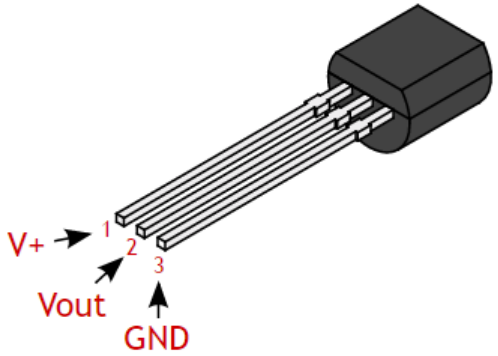
```
void setup()
{
  lcd.begin(16,2);//Inicia o LCD com dimensões 16x2(Colunas x Linhas)
  lcd.setCursor(1,0);//Posiciona o cursor na segunda coluna(1) e na primeira linha(0) do LCD
  lcd.print("Conversor  A/D");//Escrever no LCD "Conversor  A/D"
  lcd.setCursor(2,1);//Posiciona o cursor na terceira coluna(2) e linha 1
  lcd.print("TEMP = ");//Escreve no LCD "TEMP = "
}
```

Sensor de Temperatura

O **Sensor de Temperatura TMP36** é um circuito integrado medidor de temperatura que possui encapsulamento TO-92(aparência de um transistor).

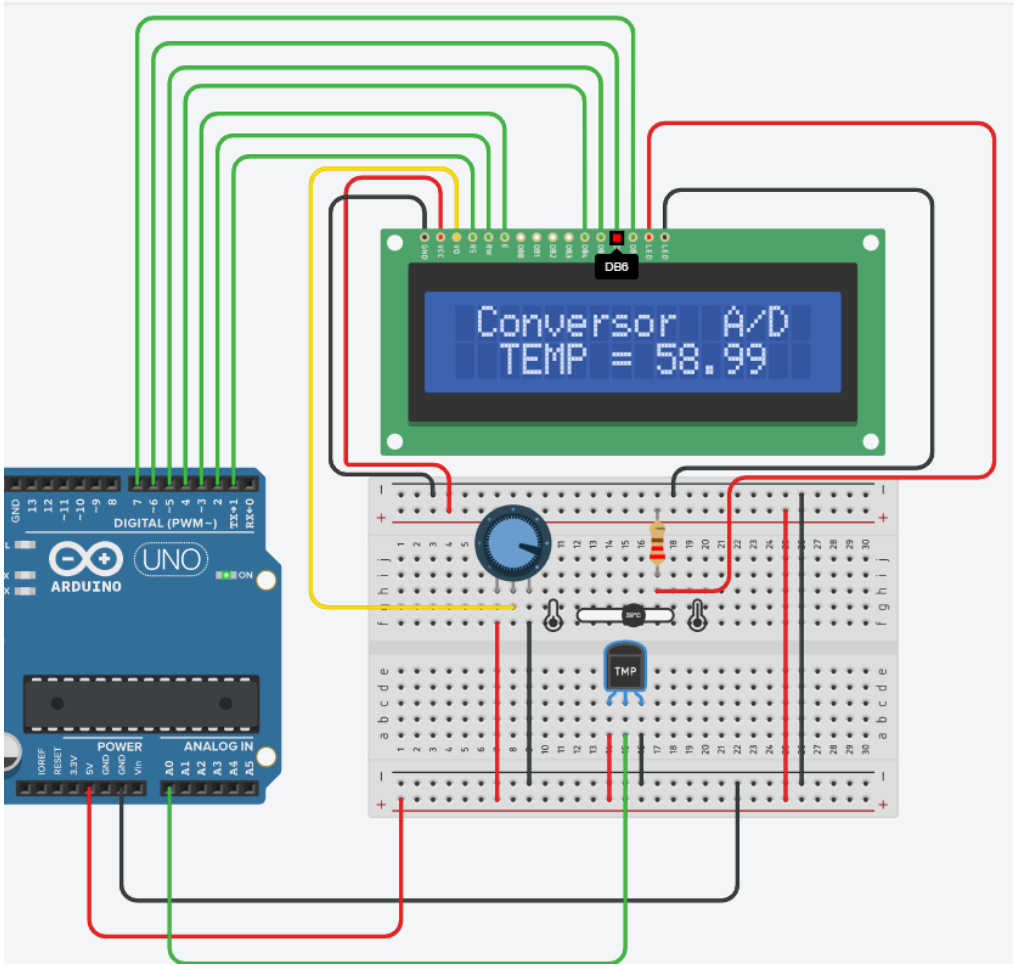
Possui alta precisão, funcionando na faixa de 2.7V a 5.5VDC. Além disso, o sensor fornece uma saída de tensão linearmente proporcional a temperatura em graus Celsius não necessitando de calibração externa para fornecer uma leitura com precisão de 1º a 25ºC e ±2° para a faixa de -40° a 125°C.

O sinal de saída (OUT) do **Sensor de Temperatura TMP36** é analógico e cada 10mV de tensão representa 1ºC.



Sensor de Temperatura

Ligação no Arduino



Sensor de Temperatura

Instrução para converter ***mV*** gerados pelo sensor LM36 em valor digital a ser mostrado em ***graus Celsius***

```
void loop()
{
  Temperatura=(float(analogRead(Sensor))*5/(1023)-0.5)/0.01;
  //Armazena o valor lido e convertido pelo sensor TMP. O 0.5
  //é para cobrir a faixa de -40°C à 125°C
```

Sensor de Temperatura

```
 //CONVERSOR A/D - Sensor de Temperatura TMP36

#include <LiquidCrystal.h>
float Temperatura;
#define Sensor A0
#define rs 1
#define rw 2
#define en 3
#define d4 4
#define d5 5
#define d6 6
#define d7 7
LiquidCrystal lcd(rs,rw,en,d4,d5,d6,d7);//Configura os pinos do Ard



void setup()
{
  lcd.begin(16,2);//Inicia o LCD com dimensões 16x2(Colunas x Linha
  lcd.setCursor(1,0);//Posiciona o cursor na segunda coluna(1) e na
  lcd.print("Conversor  A/D");//Escrever no LCD "Conversor  A/D"
  lcd.setCursor(2,1);//Posiciona o cursor na terceira coluna(2) e l
  lcd.print("TEMP = ");//Escreve no LCD "TEMP = "
}
void loop()
{
  Temperatura=(float(analogRead(Sensor))*5/(1023)-0.5)/0.01;
  //Armazena o valor lido e convertido pelo sensor TMP. O 0.5
  //é para cobrir a faixa de -40°C à 125°C
  lcd.setCursor(9,1);//posiciona o cursor na Decima coluna(9) e lin
  lcd.print(Temperatura);//ecreve o valor armazeanado em valor_conv
  lcd.print("     ");//mostra o valor lido

  delay(250);
}
```

FIM