

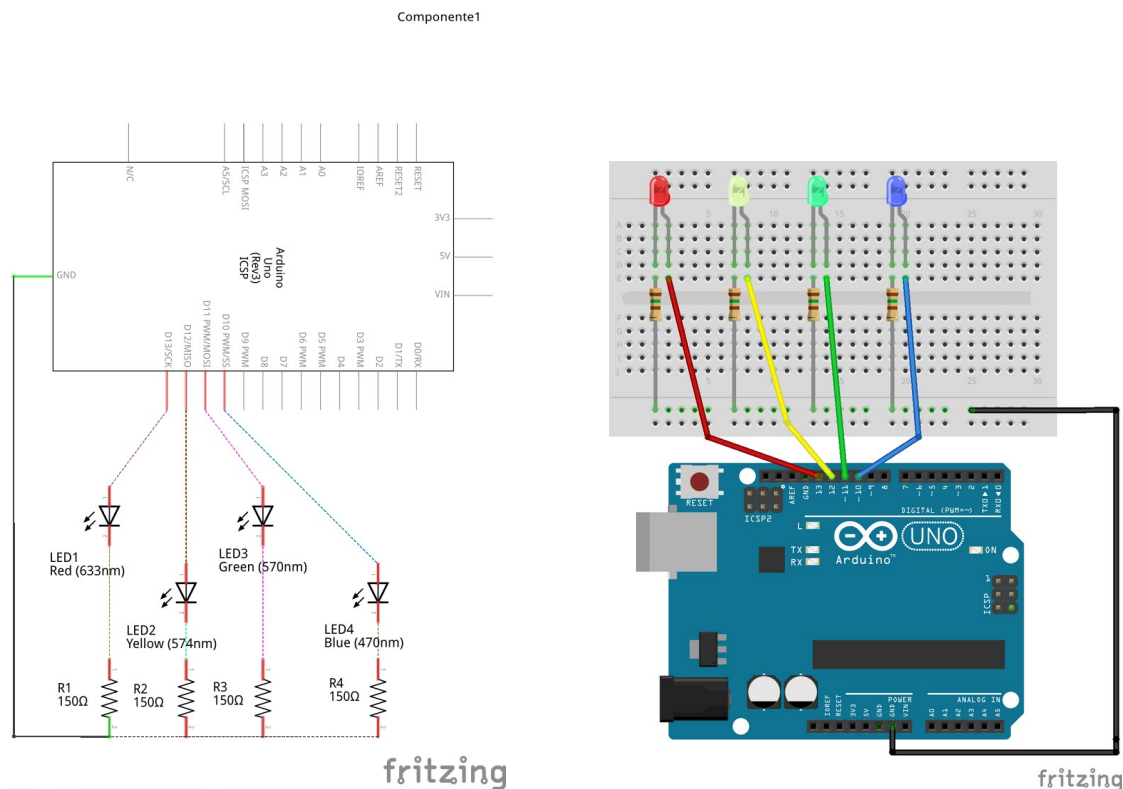
## Comandar leds usando teclado

No experimento desta tarde, vamos usar o teclado para controlar o que o Arduino fará com alguns LEDs.

Vamos usar o Monitor Serial não somente para ver o que o Arduino imprime, mas também para informá-lo do que fazer.

O circuito elétrico será o seguinte:

E a ligação física poderá ser assim:



Primeiramente, vamos ver o circuito, continuando nossos estudos de eletrônica básica.

O símbolo do resistor é (ou, ).

Para o exemplo, foram utilizados resistores com valor de 150  $\Omega$  (lê-se cento e cinquenta ohms), que tem, segundo o código internacional, as cores Marrom, Verde, Marron (Marrom = 1, Verde = 5, o que dá 15; com mais um Marrom, que corresponde a 1 zero, temos 150. A unidade é Ohm, abreviado pela letra grega ômega maiúscula:  $\Omega$ ).

O valor do resistor é calculado em função da corrente necessária para o funcionamento do circuito. Para o caso dos LEDs, utilizamos o resistor para limitar a corrente e, ao mesmo tempo, provocar uma queda na tensão aplicada. Já falamos disto, mas ainda não fizemos o cálculo. Vamos aprender agora.

Os LEDs funcionam com **tensões** que dependem do material do qual foram fabricados (o que também implica nas suas cores). Temos, como regra geral, o seguinte: LED infravermelho 1,1V (um vírgula um Volt) a 1,5V; LED vermelho 1,7V ou 1,8V; LED amarelo 2,0V; LED laranja 2,0V; LED verde 2,1V; LED azul 3,1V; LED branco 3,1V a 4,0V; LED ultravioleta 3,3V. Para termos certeza, é sempre bom verificar com o fabricante (na folha de dados, *datasheet*).

Já a **corrente** de funcionamento varia entre 0,006A (Ampere) até 0,020A, ou, mais comumente utilizada como sendo de 6mA (seis miliamperes) até 20mA (vinte miliamperes, vinte milésimos de um Ampere – notem que a letra ‘A’ é maiúscula, e a letra ‘m’ é minúscula).

Notem a importância de levarmos em consideração os dois valores conjuntamente: limitar a tensão e a corrente de funcionamento!

Por exemplo, para um LED vermelho, dependendo do fabricante, podemos ter como tensão de funcionamento 1,8V e corrente máxima de 20mA (e, talvez, mínima de 6mA para que ele funcione adequadamente). Se variarmos estes limites de corrente estaremos variando o brilho do LED. Se exagerarmos, ele queima. Se exagerarmos demais, o Arduino também queima.

Devemos lembrar que o Arduino UNO, que estamos utilizando, funciona com 5V (cinco volts).

Assim, por exemplo, se quisermos ligar um LED vermelho, de 1,8V, com corrente de 10mA (0,01A), o que corresponde a uma luminosidade média, teremos que colocar um resistor limitador, que vai permitir a passagem somente da corrente desejada e provocar uma queda na tensão de alimentação. Ele terá o valor de:

$$\text{Valor do Resistor} = (\text{Tensão do Arduino} - \text{Tensão do LED}) / \text{Corrente no LED}$$

{o valor do resistor será dado em ohms, as tensões serão em volts e a corrente em amperes}

$$\text{Valor do Resistor (R)} = (5V - 1,8V) / 0,01A$$

$$\text{Valor do Resistor (R)} = (3,2V) / 0,01A$$

$$R = 320 \Omega$$

Nem todos os valores calculados existem comercialmente. Neste caso, utilizamos valores próximos. Um valor próximo do valor calculado (320Ω, trezentos e vinte ohms) seria o de 330Ω (trezentos e trinta ohms, com as cores: laranja, laranja, marrom).

Devemos observar que os resistores possuem uma **tolerância** quanto ao valor. A tolerância pode ser conhecida por meio de uma faixa colorida metálica ao final das faixas de cores do resistor. A faixa dourada indica 5% e a prateada 10% (são as mais comuns, mas existem outras).

Por exemplo, se tivermos uma tolerância de 5%, isto significa que poderemos ter um valor entre 5% a menos ou a mais do valor nominal (o valor indicado pelas cores).

Calculamos 5% dividindo 5 por 100 e multiplicando pelo valor nominal. Ou seja:

$$\text{Valor da tolerância} = 5 / 100 * \text{Valor do Resistor}$$

$$\text{Valor da tolerância} = 0,05 * 330\Omega = 16,5\Omega$$

Isto quer dizer que o valor real do resistor de 330Ω poderá estar entre

$$330 \text{ menos } 5\% = 330\Omega - 16,5\Omega = 313,5\Omega$$

$$330 \text{ mais } 5\% = 330\Omega + 16,5\Omega = 346,5\Omega$$

Ou seja, o valor real de um resistor de 330Ω (que é o valor nominal) com +/- 5% de tolerância poderá estar entre 313,5Ω e 346,5Ω, sem que o componente esteja ruim ou estragado. Encontramos comumente no comércio resistores com tolerâncias de (sempre +/-, ou seja, para mais e para menos): 1%, 2%, 5%, 10% e 20%.

Conceitos: valor nominal é o valor pelo qual o componente é comprado (por exemplo, um resistor de 330Ω). Valor real é o valor que o componente realmente apresenta durante sua utilização (por

exemplo, um resistor de  $330\Omega \pm 5\%$  pode ter um valor de  $320\Omega$ , e isto está dentro de sua tolerância de  $\pm 5\%$ , que vai de  $313,5\Omega$  até  $346,5\Omega$ ).

Mas, usamos no exemplo resistores de  $150\Omega$  (que era o que eu tinha disponível). Como ficam as correntes e tensões então? Vamos calcular?

Para o LED vermelho  $\rightarrow 150\Omega = (5V - 1,8V) / \text{Corrente no LED} \Rightarrow 21\text{mA}$

Para o LED amarelo  $\rightarrow 150\Omega = (5V - 2,0V) / \text{Corrente no LED} \Rightarrow 20\text{mA}$

Para o LED verde  $\rightarrow 150\Omega = (5V - 2,0V) / \text{Corrente no LED} \Rightarrow 19\text{mA}$




Para o LED azul  $\rightarrow 150\Omega = (5V - 2,0V) / \text{Corrente no LED} \Rightarrow 13\text{mA}$

Como os componentes possuem, todos, tolerâncias, estamos dentro delas.

**Observação: a corrente máxima indicada para cada saída do Arduino UNO é de 40mA por pino! E, não devemos exceder 200mA na soma do total de pinos utilizados, sob pena de queimar a plaquinha!**

E, ainda, nunca usar mais do que 500mA da fonte para alimentar outras cargas, tais como motores e módulos em geral!

Agora, vamos falar um pouquinho da polaridade dos LEDs.

O símbolo do LED é  ( ou  ou  )

Veja em relação ao aspecto físico:

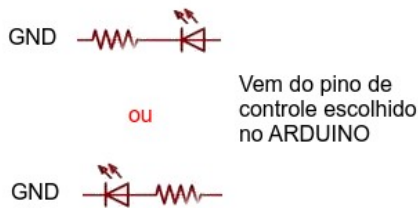


Diferente dos resistores, que podem ter seus terminais ligados indistintamente em quaisquer posições relativamente às tensões positiva e negativa da alimentação, os LEDs são componentes polarizados. Significa que deveremos respeitar sua polaridade nas ligações dos terminais, sob pena de: queimar o LED; queimar outros componentes; não queimar, mas não funcionar.

Assim, sempre ligaremos o cátodo do LED na parte mais negativa ou no 0V (zero volts, GND), e, o ânodo do LED na parte mais positiva do circuito.

A ligação do LED ficará assim:

Notem que tanto faz ligar o resistor antes do LED ou depois dele. Porém a polaridade do LED tem que ser respeitada!



Agora, finalmente, vamos ver o código desta semana.  
(adaptado de código obtido na internet)

Primeiramente, nas linhas de 1 a 4, temos:

```
1 const int ledVermelho = 13;  
2 const int ledAmarelo = 12;  
3 const int ledVerde = 11;  
4 const int ledAzul = 10;
```

Isto fará com que possamos usar 4 variáveis, identificadas pelos nomes ledVermelho (que será ligado no pino 13 – que, por sinal, já tem um led conectado na placa), ledAmarelo no pino 12, ledVerde no pino 11 e ledAzul no pino 10 (veja a figura na página 1 deste texto, e elmbre-se de que você pode modificar as ligações físicas, basta, depois, alterar os números de pinos no código).

Nas linhas 6 a 11 do código-exemplo temos nossa função de *setup()*:

```
6 void setup(){  
7   Serial.begin(9600);  
8   pinMode(ledVermelho, OUTPUT);  
9   pinMode(ledAmarelo, OUTPUT);  
10  pinMode(ledVerde, OUTPUT);  
11  pinMode(ledAzul, OUTPUT);  
12 }
```

Usaremos a comunicação serial; portanto, ela foi iniciada (na linha 7, com o comando `Serial.begin`, e inicializada com uma velocidade de 9600bps – bits por segundo).

Os quatro LEDs foram ligados a pinos do Arduino que deverão se comportar como uma saída. Portanto, nas linhas 8 até 11 usamos o comando `pinMode` para explicar que aquele pino corresponde a uma saída (OUTPUT).

Na linha 14 inicia-se a função `loop()` (e termina na linha 89, não mostrada a seguir), que funciona enquanto houver energia. Para facilitar a compreensão dos comandos usados nesta semana, a função foi resumida conforme o que segue (na sequência teremos o exemplo completo do código)

```

14 void loop(){
15   if (Serial.available())
16   {
17     switch(Serial.read())
18     {
19       case 't':
20         digitalWrite(ledAmarelo, 0);
21         digitalWrite(ledVermelho, 0);
22         digitalWrite(ledVerde, 0);
23         digitalWrite(ledAzul, 0);
24         Serial.println("Todos apagados");
25         break;

```

A linha 15 realiza um teste: SE (if) a comunicação serial tiver um dado disponível (Serial.available), então será executado um comando.

O comando inicia-se na linha 16 (com o abre chaves, {) e termina somente na linha 88, com o fecha chaves (}) do comando if.

Na linha 17 inicia-se um novo comando, o switch (que pode ser traduzido como selecione, ou escolha). Este comando termina somente na linha 87 do código. O teste realizado dentro do comando switch é dado pelo retorno da função Serial.read. Isto quer dizer que a seleção que vem na sequência será realizada em função do que for lido (read).

Assim, na linha 19 testaremos, caso (case) o valor lido for ‘t’ (letra t minúscula), serão executados todos os comandos que estiverem antes do comando de parada (break), que são:

```

digitalWrite(ledAmarelo, 0);
digitalWrite(ledVermelho, 0);
digitalWrite(ledVerde, 0);
digitalWrite(ledAzul, 0);
Serial.println("Todos apagados");

```

Ou seja, para todas as saídas, correspondentes aos LEDs amarelo, vermelho, verde e azul, foi enviado um comando de escrever o valor 0 (zero, ou desligado, ou LOW). Depois, foi informado no monitor serial, por meio do comando Serial.println (imprima e pule uma linha, ln vem de line), que todos os LEDs foram apagados.

Portanto, digitar ‘t’ faz com que todos os LEDs apaguem.

Na sequência, temos:

```

26   case 'T':
27     digitalWrite(ledAmarelo, 1);
28     digitalWrite(ledVermelho, 1);
29     digitalWrite(ledVerde, 1);
30     digitalWrite(ledAzul, 1);
31     Serial.println("Todos acesos");
32     break;

```

Agora o teste da função case será quanto à letra ‘T’ T maiúscula. Serão executados todos os comandos que estiverem antes do comando de parada (break), que são:

```
digitalWrite(ledAmarelo, 1);  
digitalWrite(ledVermelho, 1);  
digitalWrite(ledVerde, 1);  
digitalWrite(ledAzul, 1);  
Serial.println("Todos acesos");
```

Ou seja, para todas as saídas, correspondentes aos LEDs amarelo, vermelho, verde e azul, foi enviado um comando de escrever o valor 1 (um, ou ligado, ou HIGH). Depois, foi informado no monitor serial, por meio do comando `Serial.println` (imprima e pule uma linha, `ln` vem de `line`), que todos os LEDs foram acessos.

Portanto, digitar ‘**T**’ faz com que **T**odos os LEDs apaguem.

Notem a diferença: a letra minúscula é diferente da letra maiúscula. EM linguagem de programação chamamos esta diferença de *case sensitive*.

Continuando, temos:

```
33     case 'v':  
34         digitalWrite(ledVermelho, !digitalRead(ledVermelho));  
35         if(digitalRead(ledVermelho) == 1)  
36             Serial.println("Led Vermelho Aceso");  
37         else  
38             Serial.println("Led Vermelho Apagado");  
39         break;
```

O código utilizado entre as linhas 33 e 39 refere-se ao LED vermelho, mas para os demais será de comportamento idêntico.

Após testar, na linha 33, para a existência da letra ‘v’ (letra vê minúscula), serão executados todos os comandos que estiverem antes do comando de parada (`break`), que são:

```
digitalWrite(ledVermelho, !digitalRead(ledVermelho));  
if(digitalRead(ledVermelho) == 1)  
    Serial.println("Led Vermelho Aceso");  
else  
    Serial.println("Led Vermelho Apagado");
```

Na linha 34 temos o comando ‘`digitalWrite(ledVermelho, ,`’ como usual. Mas, na segunda parte do comando, ao invés de colocarmos somente uma informação LOW/ HIGH (baixo/ alto), temos ‘`!digitalRead(ledVermelho)`’. O comando `digitalRead` faz a leitura de uma porta digital (de um dos pinos de conexão do Arduino, configurado como saída, OUTPUT, pelo comando `pinMode`). Neste caso, a leitura do pino correspondente ao LED vermelho. Mas, também foi incluído o operador lógico ‘`!`’, que corresponde à operação de **negação**.

Negar, em termos lógicos, quer dizer inverter: se estiver ligado a negação leva ao desligado; se estiver desligado a negação leva ao ligado. Ou seja, não ligado = desligado, não desligado = ligado.

O que este trecho de código (`!digitalRead(ledVermelho)`) faz é INVERTER o estado do LED, ao mesmo tempo em que lê seu estado e utiliza-o para o test da linha seguinte. Ou seja, a cada vez que o comando for executado ele inverte o valor (estado) do LED (se estiver aceso, apaga; se estiver apagado, acende).

Na linha 35 é efetuado um teste lógico, por meio do operador ou comando 'if'. O teste pergunta se o valor lido referente ao LED vermelho é igual ('==') a 1 (que corresponde a ligado, a HIGH). SE for igual a 1, o código a ser executado é o da linha seguinte, linha 36, que informa que o LED está ligado. SENÃO (else), se não for igual a 1, o código a ser executado será o da linha que segue o 'else', a linha 38, que informa que o LED está apagado.

Observação: na linguagem C, e na C++, que estamos utilizando (assim como em outras), o operador '=' é o de atribuição, o '==' é o de igualdade, e o '!=' é o de desigualdade. Desta forma, temos, por exemplo:

```
variavel = 2    // a variavel recebe o valor 2
variavel == 2   // testa se a variavel é igual a 2
variavel != 2   // testa se a variavel é diferente (não igual) de 2
```

Prosseguindo no código, teremos, entre as linhas 40 e 46 o mesmo código acima, mas para o LED amarelo, testando com 'a' (letra 'a' minúscula). Entre as linhas 47 e 53 o código aplicado ao LED verde, testando com 'v' (letra 'a' maiúscula), e entre as linhas 54 e 60 o código voltado ao LED azul, testando com 'A' (letra 'A' maiúscula).

Portanto:

- digitar 'a' faz com que o LED **a**marelo tenha seu estado invertido.
- digitar 'v' faz com que o LED **v**erde tenha seu estado invertido.
- digitar 'A' faz com que o LED **A**zul tenha seu estado invertido.

Poderiam ter sido utilizados números (ou outras letras).

```
61     case '?':
62     {
63         Serial.println("Condicao dos leds:");
64         int contador = 0;
65         if(digitalRead(ledVermelho) == 1){
66             Serial.println("\tLed Vermelho Aceso");
67             contador++;
68         }
69         if(digitalRead(ledAmarelo) == 1){
70             Serial.println("\tLed Amarelo Aceso");
71             contador++;
72         }
73         if(digitalRead(ledVerde) == 1){
74             Serial.println("\tLed Verde Aceso");
75             contador++;
76         }
77         if(digitalRead(ledAzul) == 1){
78             Serial.println("\tLed Azul Aceso");
79             contador++;
80         }
81         if(contador == 0)
82             Serial.println("\tTodos os leds apagados");
83
84         Serial.print("\n");
85         break;
86     }
```

Na linha 61 testamos pela presença da letra (ou símbolo) '?'. Caso ela seja encontrada, testaremos todas as variáveis correspondentes aos LEDs para saber quais estão acesos e quais estão apagados. Isto ocorrerá por meio de testes utilizando o comando 'if', nas linhas 65, 69, 73 e 77.

A cada teste em que for encontrado um LED ligado, a variável `contador` (que foi declarada na linha 64, e inicializada com o valor 0) será incrementada de 1 (`contador++`). Na linha 81, caso o contador (que foi inicializado na linha com o valor 0) continue com o valor 0, será exibida a mensagem de que todos os LEDs estão apagados.

O comando da linha 84, `Serial.print("\n")`, simplesmente pula uma linha na exibição ('\n' é o código para uma nova linha, `new line`; e, a propósito, o comando '\t' é um comando de tabulação – ele foi usado nos 'prints' das linhas 66, 70, 74 e 78).

E, fim de código.