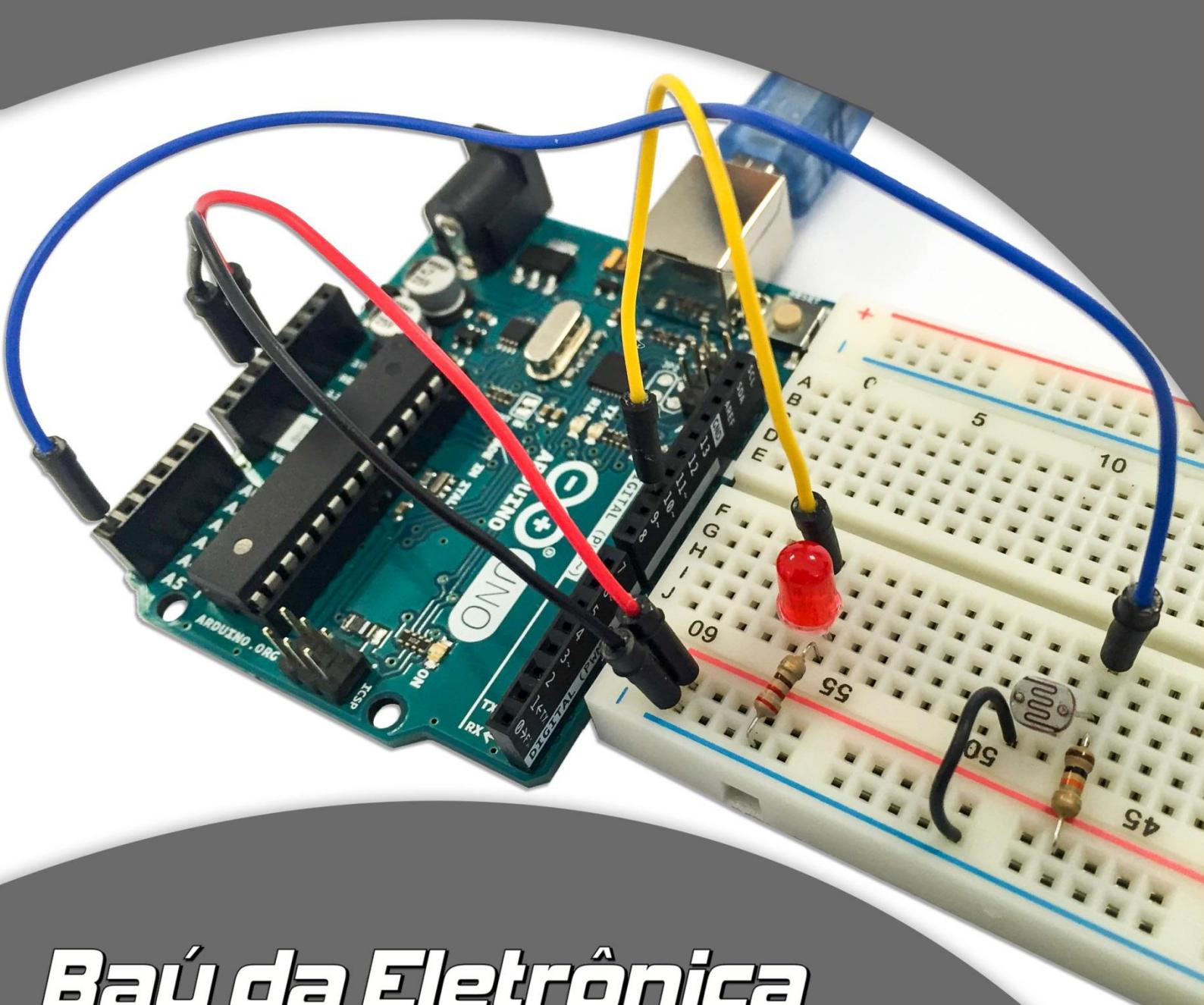


Kit Iniciante Para Arduino

Guia prático



Baú da Eletrônica
Componentes Eletrônicos

www.baudaelectronica.com.br

Este documento foi elaborado com base no conhecimento e tecnologia atuais. Consequentemente, as edições subsequentes podem levar a declarações diferentes. Como resultado, a possibilidade de interpretações equivocadas ou erros nos dados técnicos não pode ser descartada.

Prefácio

Devido a ser livremente programável e possuir projeto open source, o Arduino permite a criação de diferentes soluções para uma ampla gama de aplicações. Com seus kits didáticos o Baú da Eletrônica não só busca ampliar seu portfólio de produtos, como também criar uma forma de seus clientes obterem um pacote completo, de produto mais conhecimento para uma experiência mais aprofundada acerca dessas inúmeras possibilidades.

Este material didático fornece uma visão geral sobre os conceitos básicos da placa Arduino Uno, do software de programação IDE, dos componentes do kit e da interligação de sua placa com outros componentes e equipamentos.

Sumário

1 – O que é Arduino.....	9
2 – Conhecendo o software Arduino IDE	10
3 – Instalando o Software Arduino IDE	11
3.1 - Instalação através do arquivo .exe.....	11
3.2 - Instalação através da pasta de arquivos.	14
3.3 - Instalação do Driver Arduino.....	16
4 – Conhecendo sua placa Arduino UNO	18
5 – Diferença de pino digital e pino analógico	18
4.1 - Pinos Digitais	19
4.2 - Pinos Analógicos.....	20
4.3 – Como alimentar sua placa Arduino	21
5 – Conhecendo seu Material Didático	22
5.1 Resistor	22
5.2 - Led	24
5.3 - Led RGB	24
5.4 - Potenciômetro.....	25
5.5 - Chave Táctil de 4 terminais ou Botão.....	25
5.6 - Sensor óptico TCRT5000.....	25
5.7 - Sensor de luminosidade LDR.....	26
5.8 - Sensor de temperatura NTC.....	26
5.9 - Sensor de temperatura linear LM35	26
5.10 - Buzzer	27
5.11 - Display BCD de 7 segmentos	27
5.12 - Circuito Integrado 4511.....	27
5.13 - Display LCD 16x2	28
5.14 - Motor DC.....	28
5.15 - Circuito integrado L293D	28
5.16 - Microservo 9g.....	29
5.17 - Módulo Relé	29
5.18 - Cápsula Piezoelétrica ou piezo buzzer	29
5.19 - Protoboard	30
6 – A linguagem de programação	31

6.1 - Sintaxe	31
6.1.1 – Chaves { }	31
6.1.2 – Ponto e Vírgula ;	32
6.1.3 – Blocos de Comentários /* ... */ ;	32
6.1.4 – Linhas de Comentários // ;	32
6.2 – Funções Principais	33
6.2.1 - Função setup().....	33
6.2.2 - Função loop()	34
6.3 - Funções	34
7 – Variáveis	35
7.1 - Declaração de variáveis.....	36
7.2 - Escopo de Variáveis.....	36
7.3 - Tipos de Variáveis.....	37
7.3.1 - byte.....	37
7.3.2 - int	38
7.3.3 – unsigned int	38
7.3.4 - char.....	38
7.3.5 – word.....	39
7.3.6 – long	39
7.3.7 – unsigned long	39
7.3.8 – float.....	39
8 - Aritmética	40
9 – Operadores de comparação	41
10 – Operadores lógicos	41
11 – Constantes	42
11.1 – true / false	42
11.2 – HIGH / LOW	42
11.3 – input / output	42
12 – Estruturas de controle	42
12.1 – if	43
12.2 – if... else	43
12.3 – for	44
12.4 – while	45

12.5 – do - while	46
12.6 – switch case.....	47
13 – Funções para pinos digitais	47
13.1 - pinMode().....	48
13.2 - digitalWrite()	48
13.3 - digitalRead()	48
14 – Funções para pinos analógicos	49
14.1 - analogRead()	49
14.2 - analogWrite()	50
15 – Funções avançadas	51
15.1 - tone()	51
15.2 - noTone()	52
16 – Funções de tempo	53
16.1 - millis()	53
16.3 - micros ().....	53
16.4 - delay ().....	53
16.5 - delayMicroseconds ()	54
17 – Exemplo de um código fonte.....	55
18 – Projetos.....	60
18.1 – Projeto 1 (Led Piscante)	60
18.1.1 – Componentes necessários.....	60
18.1.2 – Montando o circuito	61
18.1.3 – Código fonte	64
18.2 – Projeto 2 (Acendendo o LED com um botão)	65
18.2.1 – Componentes necessários.....	65
18.2.2 – Montando o circuito	66
18.2.3 – Código fonte	67
18.3 – Projeto 3 (Semáforo)	68
18.3.1 – Componentes necessários.....	68
18.3.2 – Montando o circuito	69
18.3.3 – Código fonte	70
18.4 – Projeto 4 (Controlando o LED RGB)	71
18.4.1 – Componentes necessários.....	71

18.4.2 – Montando o circuito	71
18.4.3 – Código fonte	73
18.5 – Projeto 5 (Detector de objetos)	75
18.5.1 – Componentes necessários.....	75
18.5.2 – Montando o circuito	76
18.5.3 – Código fonte	77
18.6 – Projeto 6 (Piano)	79
18.6.1 – Componentes necessários.....	79
18.6.2 – Montando o circuito	80
18.6.3 – Código fonte	81
18.7 – Projeto 7 (Medindo Temperatura com Sensor NTC)	83
18.7.1 – Componentes necessários.....	83
18.7.2 – Montando o circuito	84
18.7.3 – Código fonte	85
18.8 – Projeto 8 (Alarme de temperatura)	87
18.8.1 – Componentes necessários.....	87
18.8.2 – Montando o circuito	88
18.8.3 – Código fonte	89
18.9 – Projeto 9 (Medindo Temperatura com LM35)	90
18.9.1 – Componentes necessários.....	90
18.9.2 – Montando o circuito	91
18.9.3 – Código fonte	92
18.10 – Projeto 10 (Controle de iluminação automatizado)	94
18.10.1 – Componentes necessários.....	94
18.10.2 – Montando o circuito	95
18.10.3 – Código fonte	96
18.11 – Projeto 11 (Display BCD de 7 segmentos)	98
18.11.1 – Componentes necessários.....	99
18.11.2 – Montando o circuito	100
18.11.3 – Código fonte	101
18.12 – Projeto 12 (Display BCD de 7 segmentos com 4511)	104
18.12.1 – Componentes necessários.....	104
18.12.2 – Montando o circuito	105

18.12.3 – Código fonte	106
18.13 – Projeto 13 (Escrevendo no display LCD 16x2)	108
18.13.1 – Componentes necessários.....	108
18.13.2 – Montando o circuito	109
18.13.3 – Código fonte	110
18.14 – Projeto 14 (Enviando mensagens para o LCD 16x2).....	113
18.14.1 – Componentes necessários.....	113
18.14.2 – Montando o circuito	114
18.14.3 – Código fonte	115
18.15 – Projeto 15 (Emitindo sons com o Piezo Buzzer)	116
18.15.1 – Componentes necessários.....	116
18.15.2 – Montando o circuito	116
18.15.3 – Código fonte	117
18.16 – Projeto 16 (Sensor de toque com o Piezo Buzzer)	118
18.16.1 – Componentes necessários.....	118
18.16.2 – Montando o circuito	119
18.16.3 – Código fonte	120
18.17 – Projeto 17 (Controlando o Motor DC)	121
18.17.1 – Componentes necessários.....	121
18.17.2 – Montando o circuito	122
18.17.3 – Código fonte	125
18.18 – Projeto 18 (Movimentos com Micro Servo 9g)	127
18.18.1 – Componentes necessários.....	127
18.18.2 – Montando o circuito	127
18.18.3 – Código fonte	128
18.19 – Projeto 19 (Automação com módulo Relé)	130
18.19.1 – Componentes necessários.....	130
18.19.2 – Montando o circuito	131
18.19.3 – Código fonte	132
19 – Próximos passos	133

1 – O que é Arduino

Arduino é um projeto que engloba as diversas placas da plataforma (como o versão UNO, MEGA, DUE entre outras), e o software de programação IDE. Analogamente uma placa Arduino nada mais é que uma espécie de um computador, que ao invés de um sistema operacional (Windows, Linux ou MAC OS) , executa o código fonte elaborado por você, possui portas de comunicação, uma entrada para fonte de alimentação, microcontrolador, memória RAM, memória Flash, etc.

Esse módulo do nosso curso é baseado na placa Arduino UNO, que é a placa mais simples e didática da família Arduino, o microcontrolador é o modelo ATmega328P. Dispõe de 14 pinos digitais de entrada / saída (de 6 que podem ser utilizados como saídas PWM), 6 entradas analógicas, um cristal de quartzo 16 MHz, a conexão USB, um plug P4 de alimentação, pinos para ICSP e um botão de reset.

Na placa contém tudo o que é necessário para o funcionamento do microcontrolador. Para utilizá-la basta conectá-la a um computador com um cabo USB ou ligá-la a uma fonte de energia como uma fonte de tomada ou uma bateria. Você pode mexer com o seu Arduino UNO sem medo de fazer algo errado, pois na pior das hipóteses, você ainda pode substituir o microcontrolador ATMega328P que possui soquete para fácil substituição e um preço relativamente baixo no mercado.

2 – Conhecendo o software Arduino IDE

O software Arduino IDE (Integrated Development Environment), é o ambiente onde criamos, compilamos e realizamos o upload do nosso código fonte. O software é gratuito e bastante simples de utilizar. No módulo básico do nosso curso vamos aprender a utilizar as ferramentas, botões e funções listadas a seguir.

Verificar

Verifica no código fonte digitado se o mesmo está correto, caso esteja 100% correto o código é compilado e caso encontre algum erro informa na caixa de diálogo.

Descarregar

Executa todas as funções do botão verificar e caso o código esteja correto, realiza o upload do mesmo na placa Arduino.

Abrir

Abre um programa salvo anteriormente.

Monitor Serial

Ferramenta para ler e enviar dados para sua placa Arduino.

Novo

Abre um novo documento em branco.

Salvar

Salva as alterações realizadas.
Obs. Quando o código é compilado, as alterações são salvas automaticamente.

Ambiente de programação

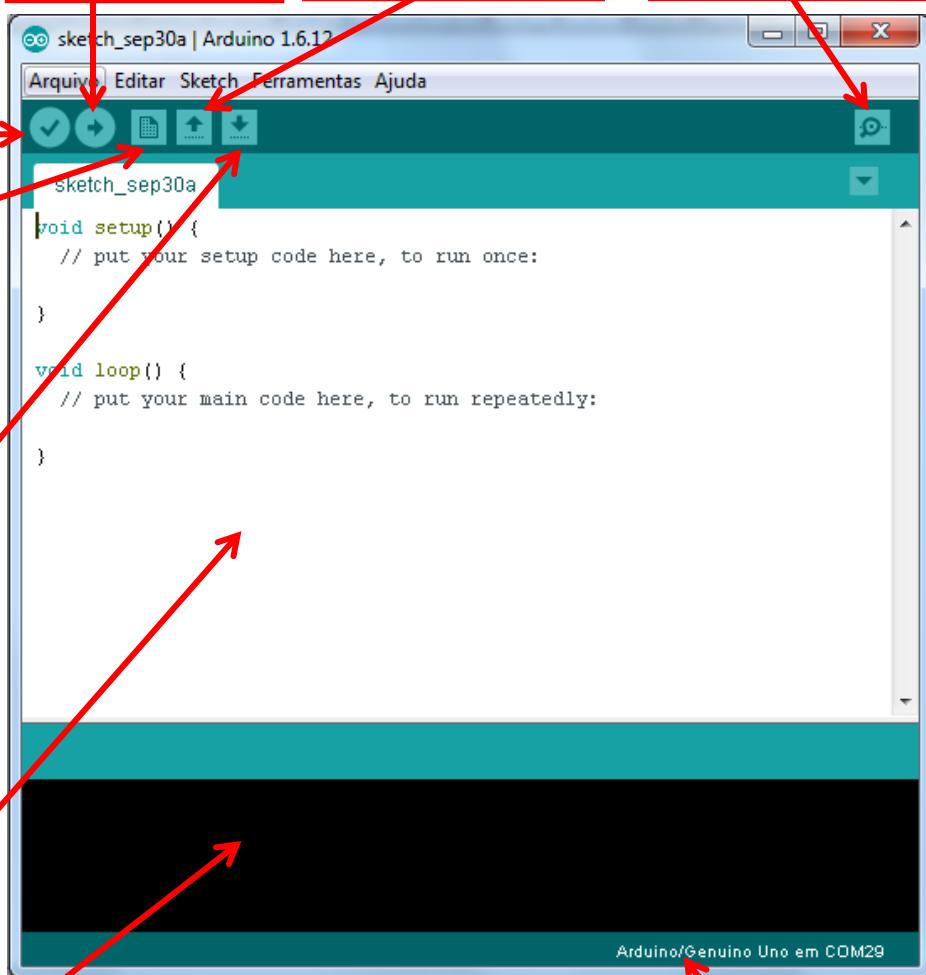
Área para criação e escrita do código fonte.

Caixa de diálogo

Área onde são exibidas mensagens sobre o código fonte. como por exemplo se o mesmo possui erros, onde encontram-se os erros, se o código foi compilado, o tamanho do programa em KB, etc.

Hardware configurado

Este campo mostra qual placa Arduino você vai utilizar.

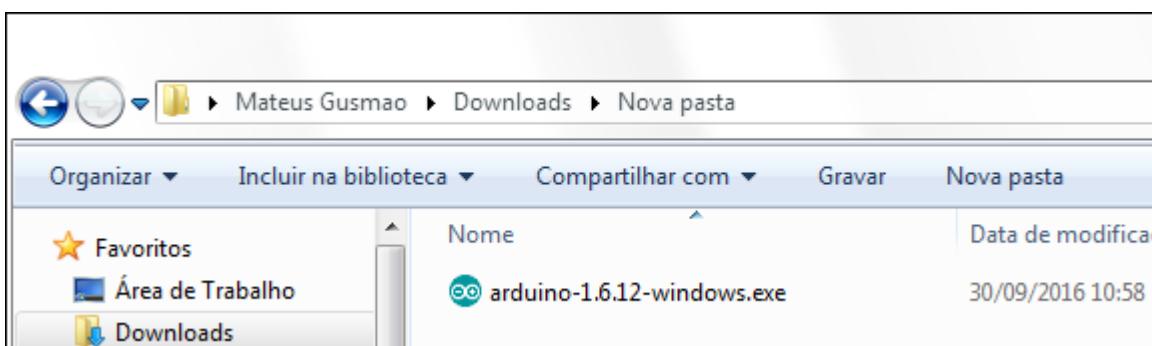


3 – Instalando o Software Arduino IDE

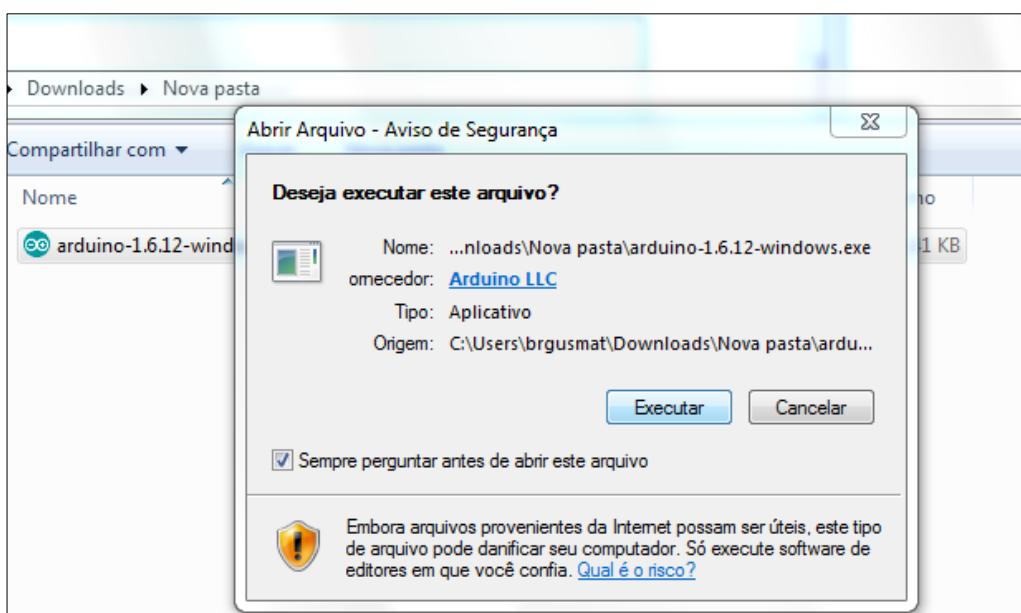
O software Arduino IDE pode ser instalado através do arquivo executável ".exe" ou através da pasta de arquivos para usuários sem permissão de administrador. Vamos abordar aqui os dois métodos e assim você poderá então optar pelo que melhor te atender.

3.1 - Instalação através do arquivo .exe

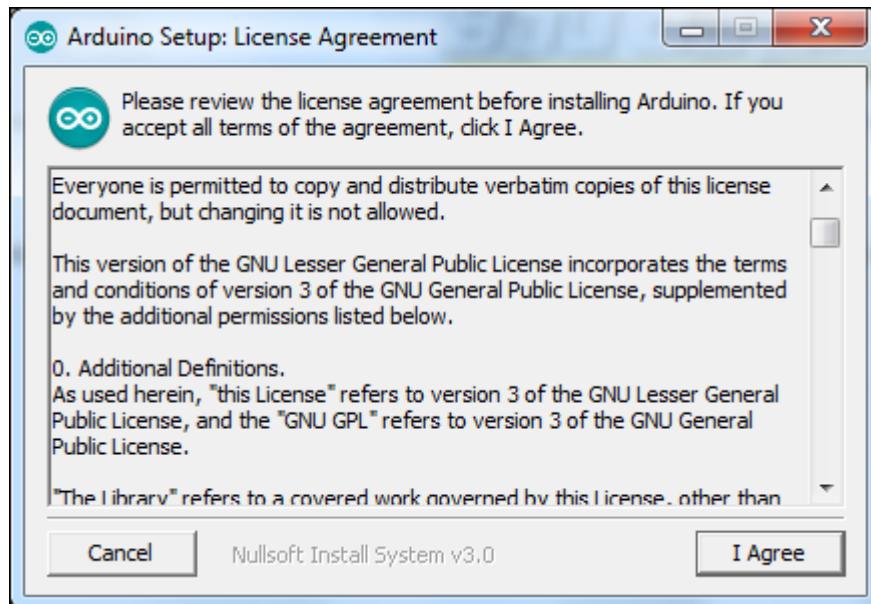
Passo 1 - Para instalação através do Instalador ".exe" primeiramente você deve realizar o download do arquivo de instalação selecionando a opção "**Windows Installer**" na página <https://www.arduino.cc/en/Main/Software> e salvando em uma pasta de sua preferência;



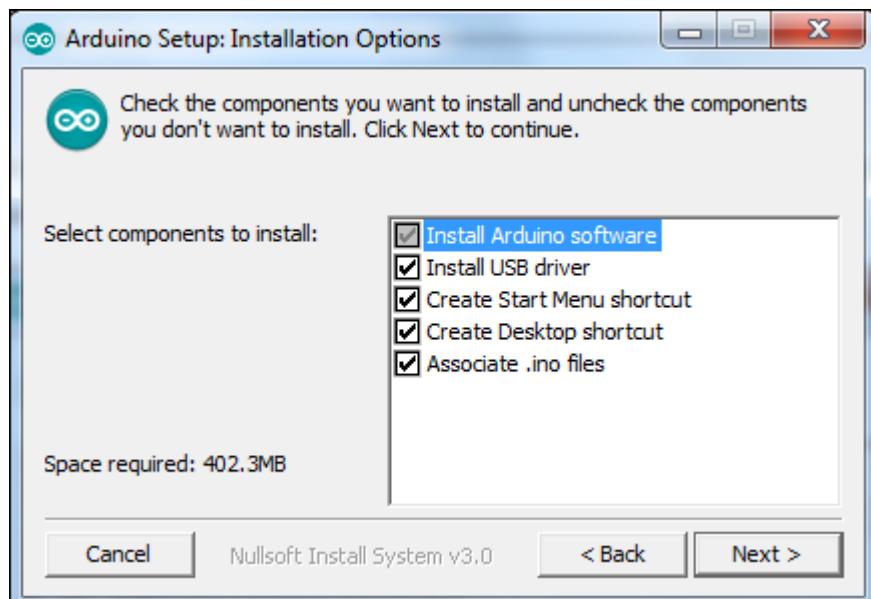
Passo 2 - Já com o arquivo salvo em sua pasta, execute-o, depois clique no botão executar;



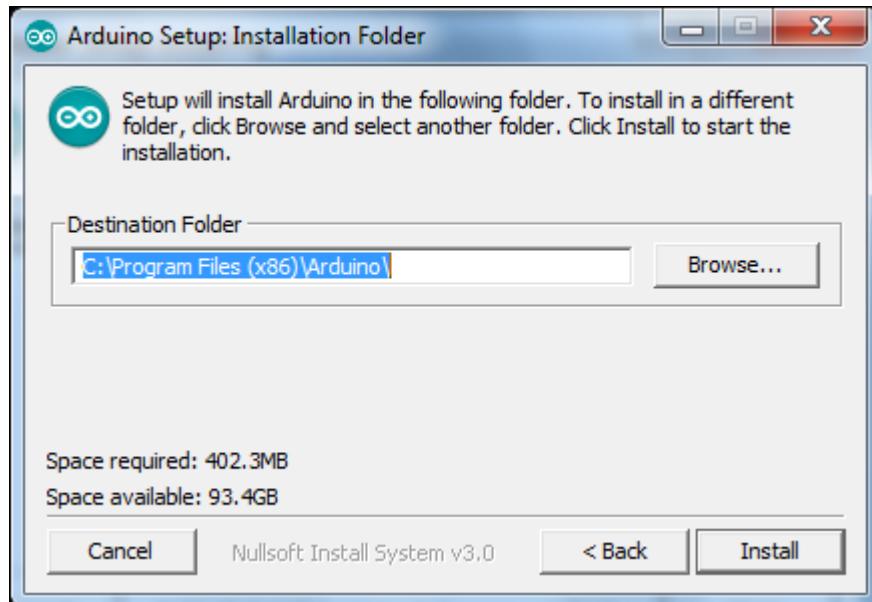
Passo 3 - Clique no botão "I Agree" na parte inferior direita da janela para aceitar os termos da licença de utilização do software;



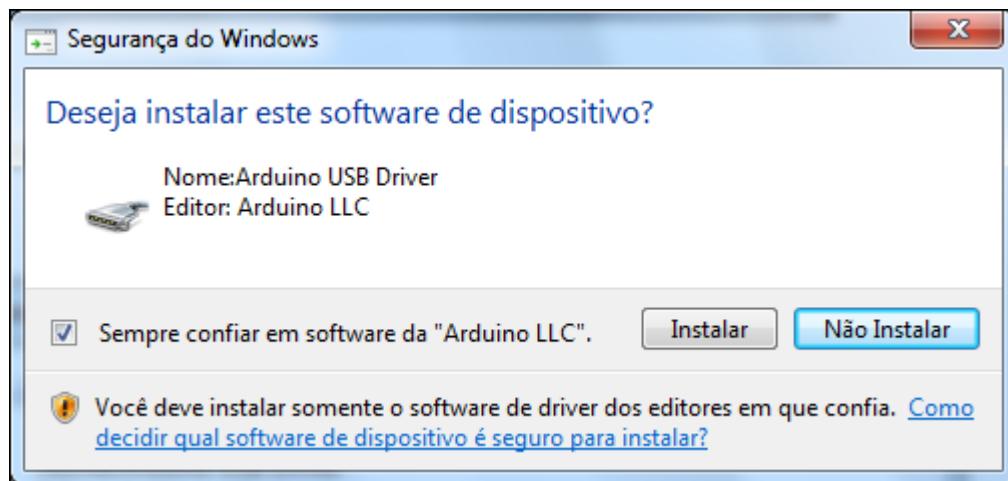
Passo 4 - Certifique que todas as opções estão selecionadas e clique em "Next";



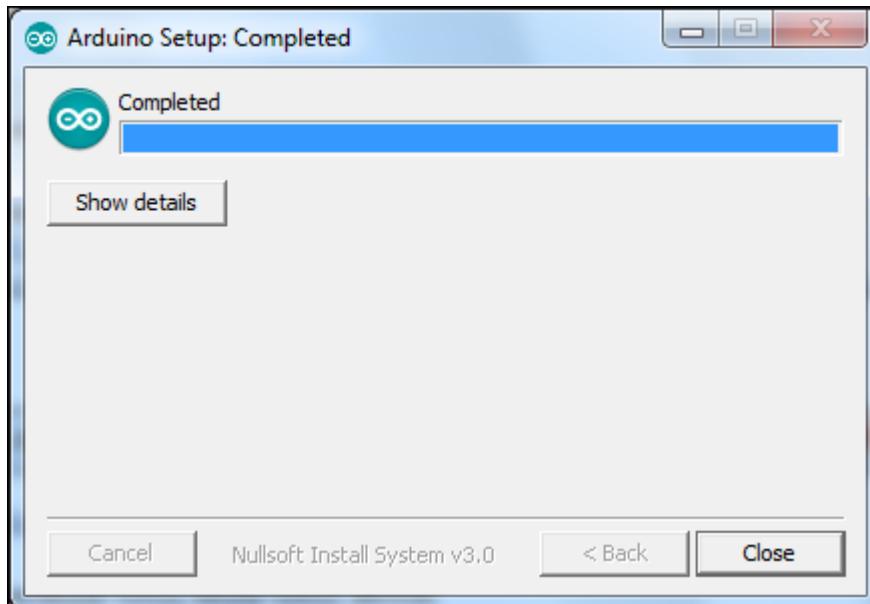
Passo 5 - Clique no botão "Install";



Passo 6 - Aguarde, e ao final do processo de instalação abrirá a janela a seguir, selecione a opção "Sempre confiar em software da "Arduino LLC" e clique no botão "Instalar";



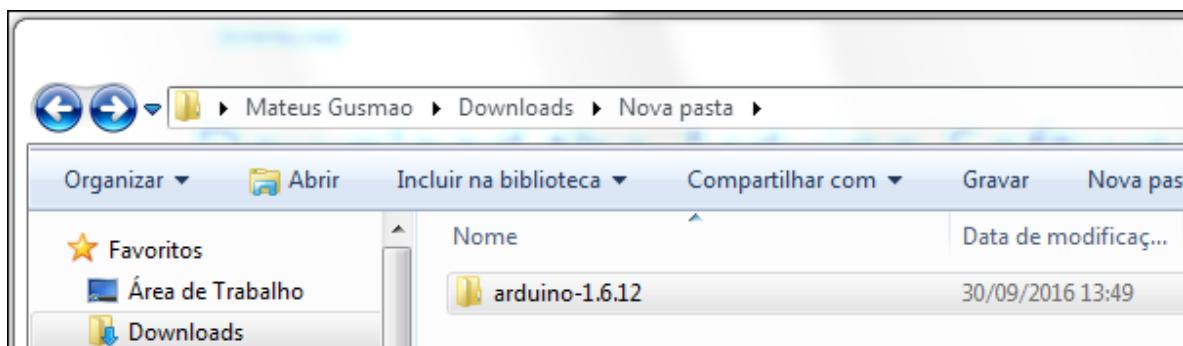
Passo 7 - Aguarde, alguns segundos até que o botão "Close" apareça e clique nele.



Pronto, o software está instalado.

3.2 - Instalação através da pasta de arquivos.

Passo 1 - Para instalação através da pasta de arquivos, você deve realizar o download da pasta de arquivos de instalação selecionando a opção "**Windows ZIP file for non admin install**" na página <https://www.arduino.cc/en/Main/Software> e salvando em uma pasta de sua preferência;

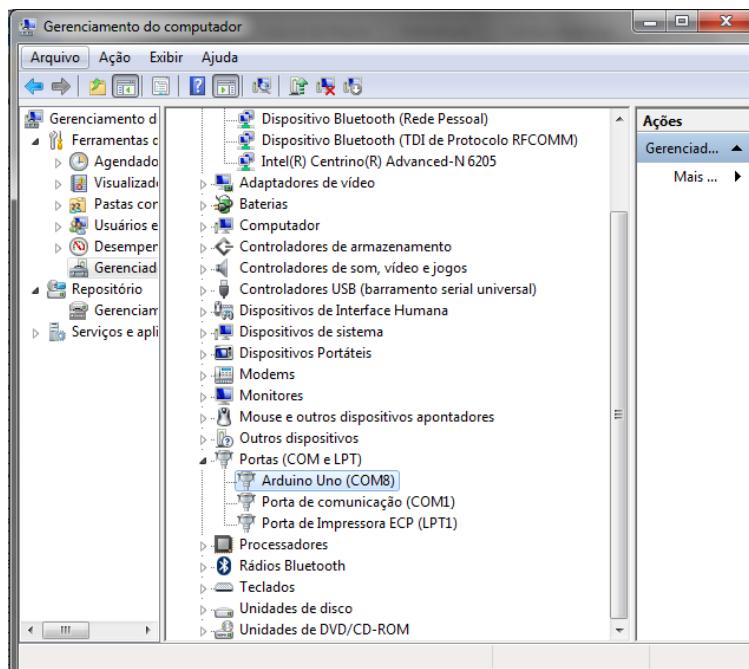


Pronto, o software está instalado, para iniciar o programa, basta executar o arquivo arduino.exe.

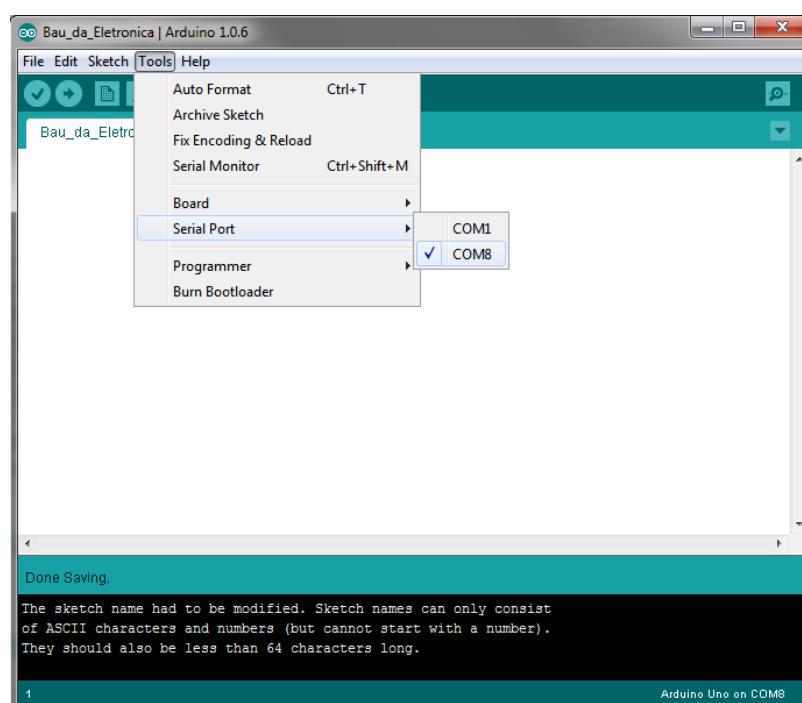
Downloads ▶ Nova pasta ▶ arduino-1.6.12 ▶			
Nome	Data de modificaç...	Tipo	Tamanho
drivers	30/09/2016 13:49	Pasta de arquivos	
examples	30/09/2016 13:49	Pasta de arquivos	
hardware	30/09/2016 13:49	Pasta de arquivos	
java	30/09/2016 13:52	Pasta de arquivos	
lib	30/09/2016 13:52	Pasta de arquivos	
libraries	30/09/2016 13:53	Pasta de arquivos	
reference	30/09/2016 13:53	Pasta de arquivos	
tools	30/09/2016 13:55	Pasta de arquivos	
tools-builder	30/09/2016 13:55	Pasta de arquivos	
arduino.exe	21/09/2016 12:13	Aplicativo	393 KB
arduino.l4j.ini	21/09/2016 12:13	Parâmetros de co...	1 KB
arduino_debug.exe	21/09/2016 12:13	Aplicativo	391 KB
arduino_debug.l4j.ini	21/09/2016 12:13	Parâmetros de co...	1 KB
arduino-builder.exe	21/09/2016 12:12	Aplicativo	3.797 KB
libusb0.dll	21/09/2016 12:12	Extensão de aplica...	43 KB
msvcp100.dll	21/09/2016 12:12	Extensão de aplica...	412 KB
msvcr100.dll	21/09/2016 12:12	Extensão de aplica...	753 KB
revisions.txt	21/09/2016 12:12	Documento de Te...	78 KB

3.3 - Instalação do Driver Arduino

O hardware Arduino é Plug & Play, o que significa que ao ser conectado à uma porta USB de seu computador o mesmo será automaticamente reconhecido e instalado. Após instalado ele aparecerá pra você com uma porta COM. Como no exemplo a seguir que foi reconhecido instalado como Arduino Uno (COM8).

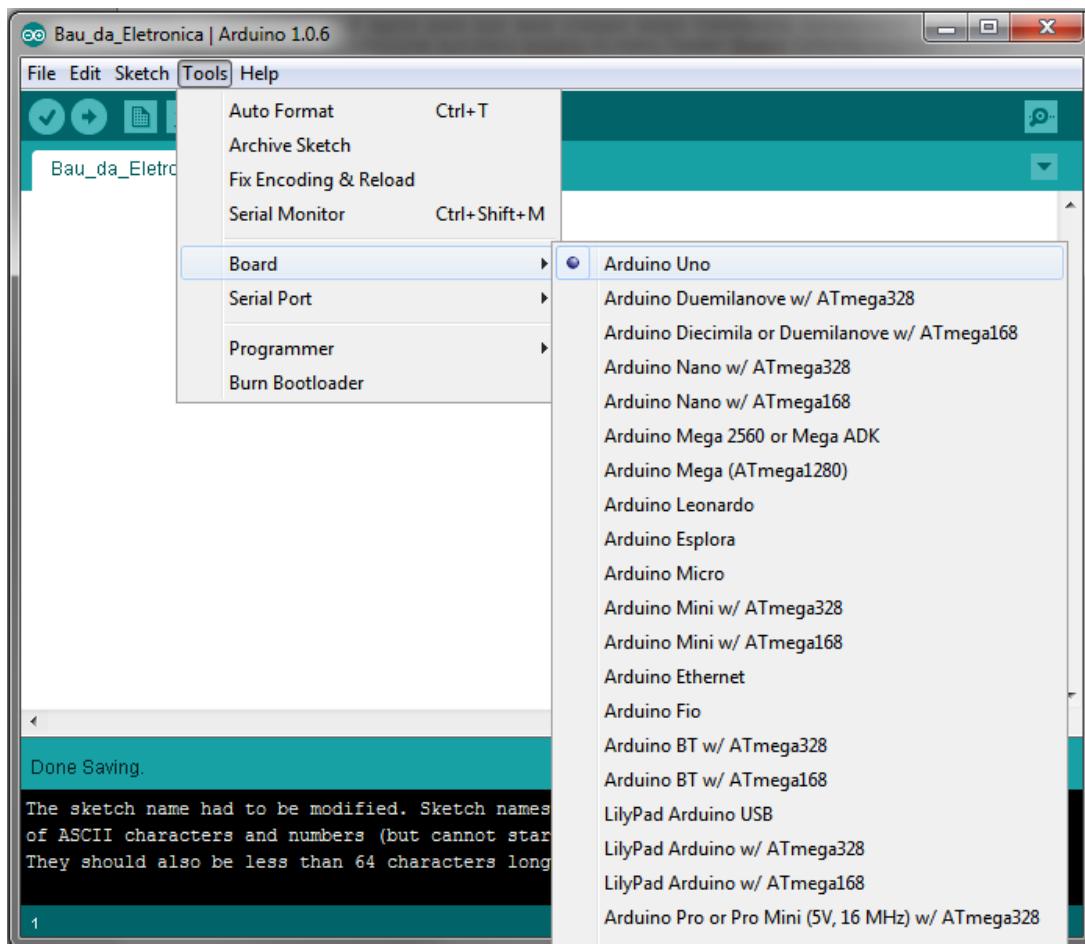


Agora selecione a porta em que está sua placa Arduino no software de programação. Para isso clique em **Tools> Serial Port> COMx** conforme exemplo a seguir:



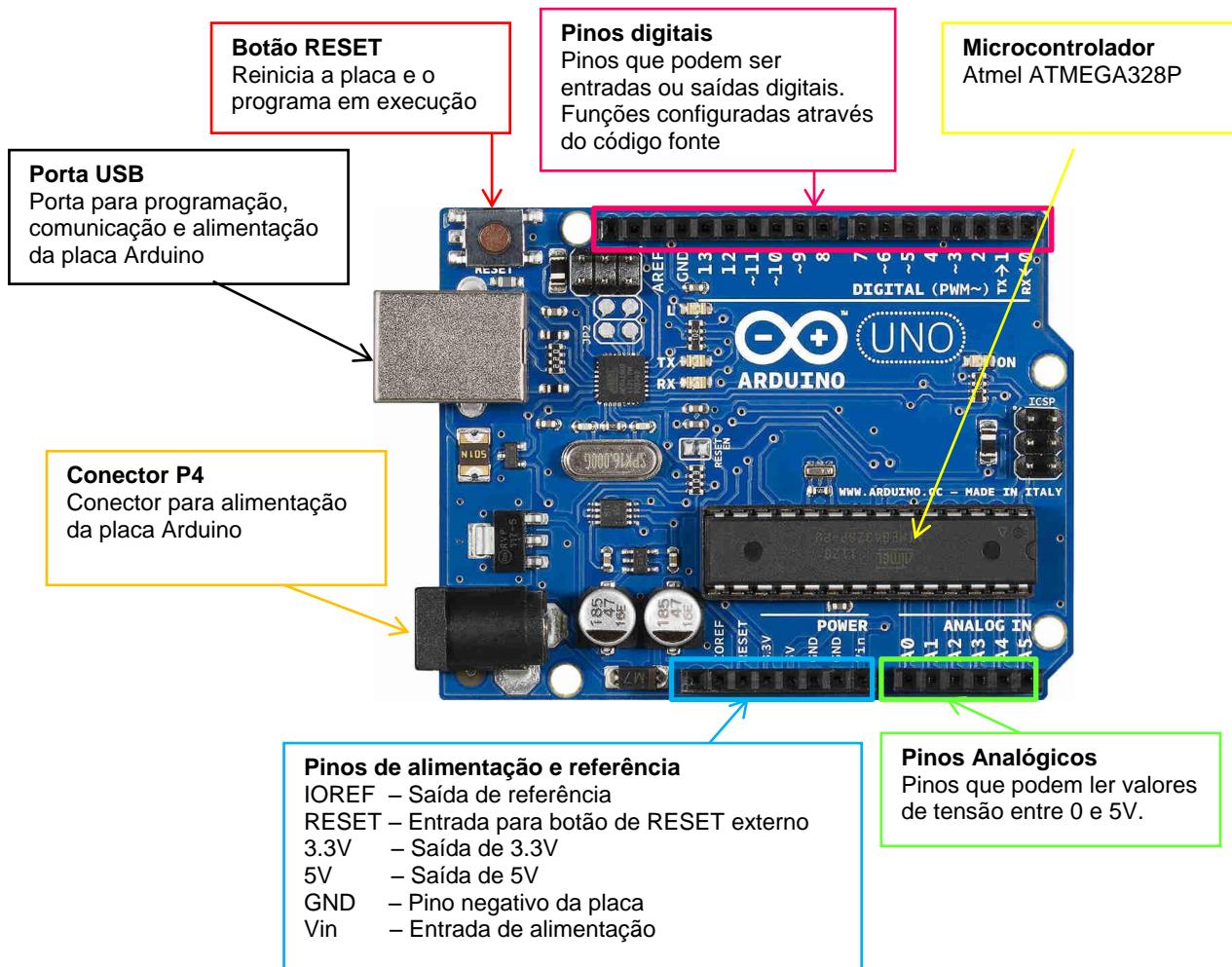
Após a instalação, o número da porta COM que aparecerá em seu computador não necessariamente será 8 como no exemplo acima. Este número vai mudar de acordo com cada computador.

E agora para que seus códigos sejam transferidos corretamente é necessário selecionar sua placa Arduino no menu **Tools> Board** conforme exemplo a seguir:

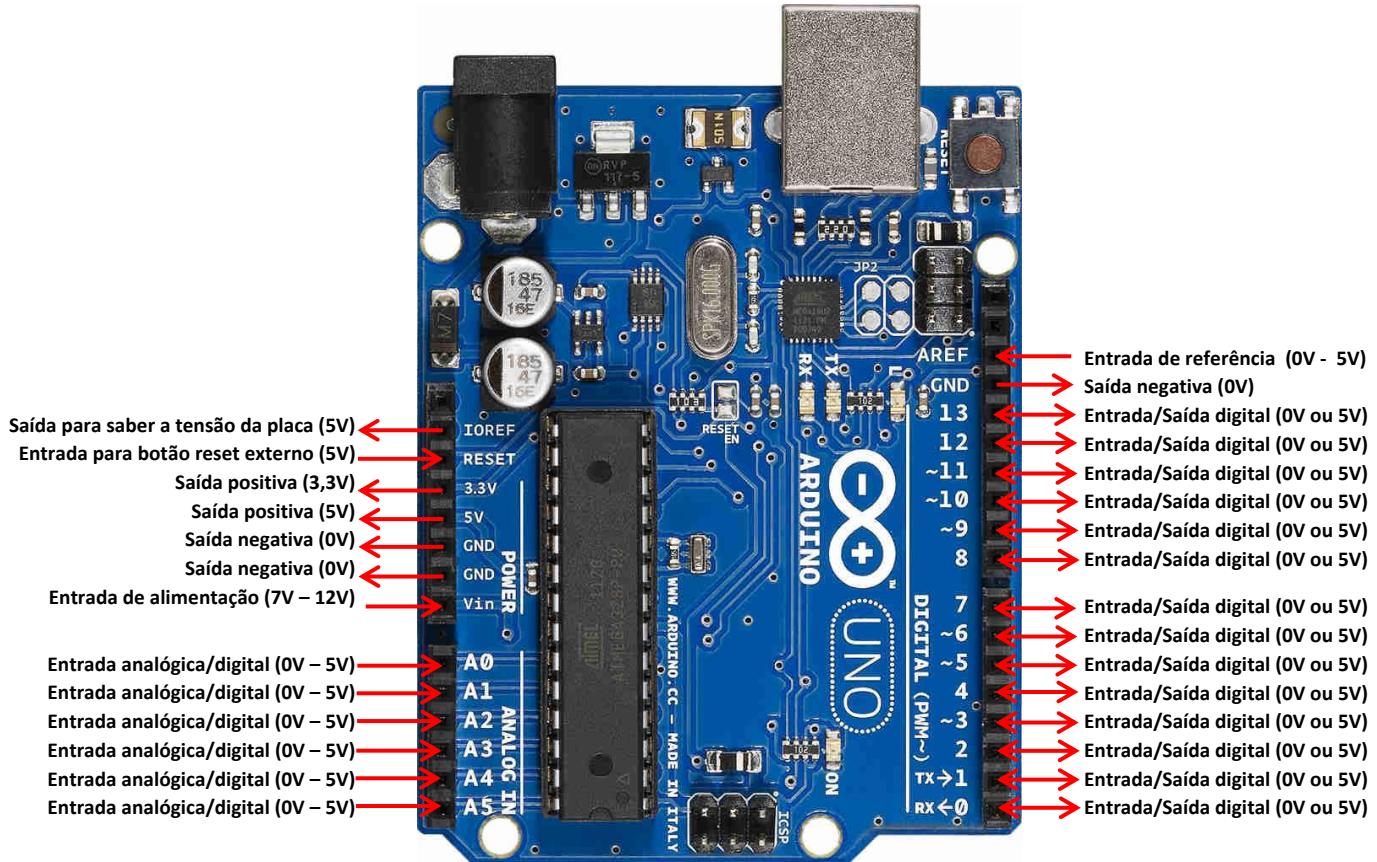


4 – Conhecendo sua placa Arduino UNO

Sua placa Arduino UNO é composta basicamente pelas partes a seguir:



A imagem a seguir mostra a função de cada pino da placa Arduino individualmente se é entrada ou saída. Você deve consulta-la sempre que tiver dúvida sobre algum pino, se o mesmo é entrada ou saída e qual valor de tensão máxima e mínima este pino suporta.



Como podemos ver, em nossa placa Arduino existem dois tipos de pinos, pinos digitais e pinos analógicos, veja a seguir as diferenças entre eles.

4.1 - Pinos Digitais

Pinos digitais são muito mais fáceis de entender pois possuem apenas dois estados, ON ou OFF. Em termos de Arduino, um estado ON é conhecido como HIGH (5V) e do estado OFF é conhecido como LOW (0V).

É importante fixar em sua mente que um pino digital pode ter dois estados:

- **HIGH = ON** – possui 5V presente no pino;
- **LOW = OFF** – Possui 0V presente no pino ou não possui tensão alguma;

Os pinos digitais do Arduino podem ser configurados como entradas ou saídas. Vamos explorar o funcionamento dos pinos em ambos os modos. Neste tópico estamos abordando os pinos digitais,

mas é importante ter em mente que todos os pinos analógicos da nossa placa Arduino, podem ser configurados e utilizados, exatamente da mesma maneira que os pinos digitais.

Pinos digitais são por padrão configurados como entradas, por isso eles não precisam ser explicitamente declarados como entradas com a função `pinMode ()` quando você quiser usa-los como entradas. Pinos configurados como entradas são colocados em estado de alta impedância.

Pinos configurados como saída com a função `pinMode ()` são colocados em estado de baixa impedância. Isto significa que eles podem fornecer uma quantidade substancial de corrente para outros circuitos. Os pinos digitais podem fornecer até 40 mA (miliampères) de corrente para outros dispositivos / circuitos. Esta corrente é o suficiente para acender um LED, ou ler vários sensores por exemplo, mas não o suficiente atual para ligar relés, solenóides ou motores.

Cuidado: *Curtos-circuitos nos pinos do Arduino, ou a tentativa de ligar dispositivos de alta corrente como relés, solenóides ou motores, pode danificar os transistores internos do pino, ou danificar o chip ATmega inteiro. Muitas vezes, isso irá resultar em um pino "morto" no microcontrolador mas o chip restante continuará a funcionar adequadamente. Por esta razão, é uma boa ideia sempre utilizar resistores de 470Ω a 1k para conectar pinos de saída a outros dispositivos.*

4.2 - Pinos Analógicos

Um pino analógico pode assumir qualquer número de valores, ao contrário do sinal digital, que como vimos há pouco tem apenas dois valores (LOW=0V e HIGH=5V). Para medir o valor de sinais analógicos, o Arduino utiliza de um conversor analógico-digital (ADC) interno. O ADC transforma a tensão analógica num valor digital. A função que você usa para obter o valor de um sinal analógico é `analogRead ()`. Esta função converte o valor da tensão num pino de entrada analógica e te fornece um valor de 0 a 1023, proporcional ao valor de referência. A referência é 5V na maioria dos Arduinos, 7V no Arduino Mini e Nano, e 15V em Arduino mega. Para usar esta função basta inserir o número do pino utilizado dentro dos parênteses assim `analogRead (nº do pino)`.

Apenas a título de informação, o Arduino não possui um conversor de digital para analógico (DAC), mas ele pode modular por largura de pulso (PWM) um sinal digital para alcançar algumas das funções de uma saída analógica. A função utilizada para emitir um sinal PWM é `analogWrite (pino, valor)`. Pino é o número do pino usado como saída PWM e valor é um número proporcional ao ciclo de trabalho do sinal. Quando o valor = 0, o sinal é sempre desligado. Quando o valor = 255, o sinal é sempre ligado. Na maioria das placas Arduino, a função PWM está disponível nos pinos 3, 5, 6, 9, 10 e 11. A frequência do sinal PWM na maioria dos pinos é de aproximadamente 490 Hz. Nas placas Uno e similares, pinos 5 e 6 têm uma frequência de aproximadamente 980 Hz. Pinos 3 e 11 no Leonardo também executar em 980 Hz.

4.3 – Como alimentar sua placa Arduino

O Arduino pode ser alimentado de três maneiras, diretamente pela porta USB, através do conector P4 e através dos pinos Vin e GND.

Para alimentar através da porta USB basta utilizar um cabo USB padrão e conectá-lo em sua placa e numa porta USB de um PC ou laptop. Esta maneira é mais utilizada durante a fase de criação do código fonte pois é mais prático já que a placa deve estar conectada ao PC para descarregar os seus sketches.



As outras duas maneiras são mais utilizadas quando o projeto já está finalizado ou case necessite de alguma mobilidade como o caso de robôs ou projetos portáteis. Tanto através do conector P4 como através dos pinos Vin e GND a tensão de alimentação deve estar entre 7 e 12V.

Para alimentar através do conector P4 você pode utilizar uma bateria 9V, case para 6 pilhas de 1,5V, uma fonte de tomada 9V, enfim, qualquer fonte de energia CC que possua tensão entre 7 e 12V.



Caso você alimente a placa com menos de 7V, no entanto, o pino de 5V pode fornecer menos do que cinco volts e a placa pode apresentar funcionamento instável. E caso você utilize de mais do que 12 V, o regulador de tensão pode superaquecer e danificar a placa. ***"Lembre-se o valor seguro e recomendado é de 7 a 12 volts".***

5 – Conhecendo seu Material Didático



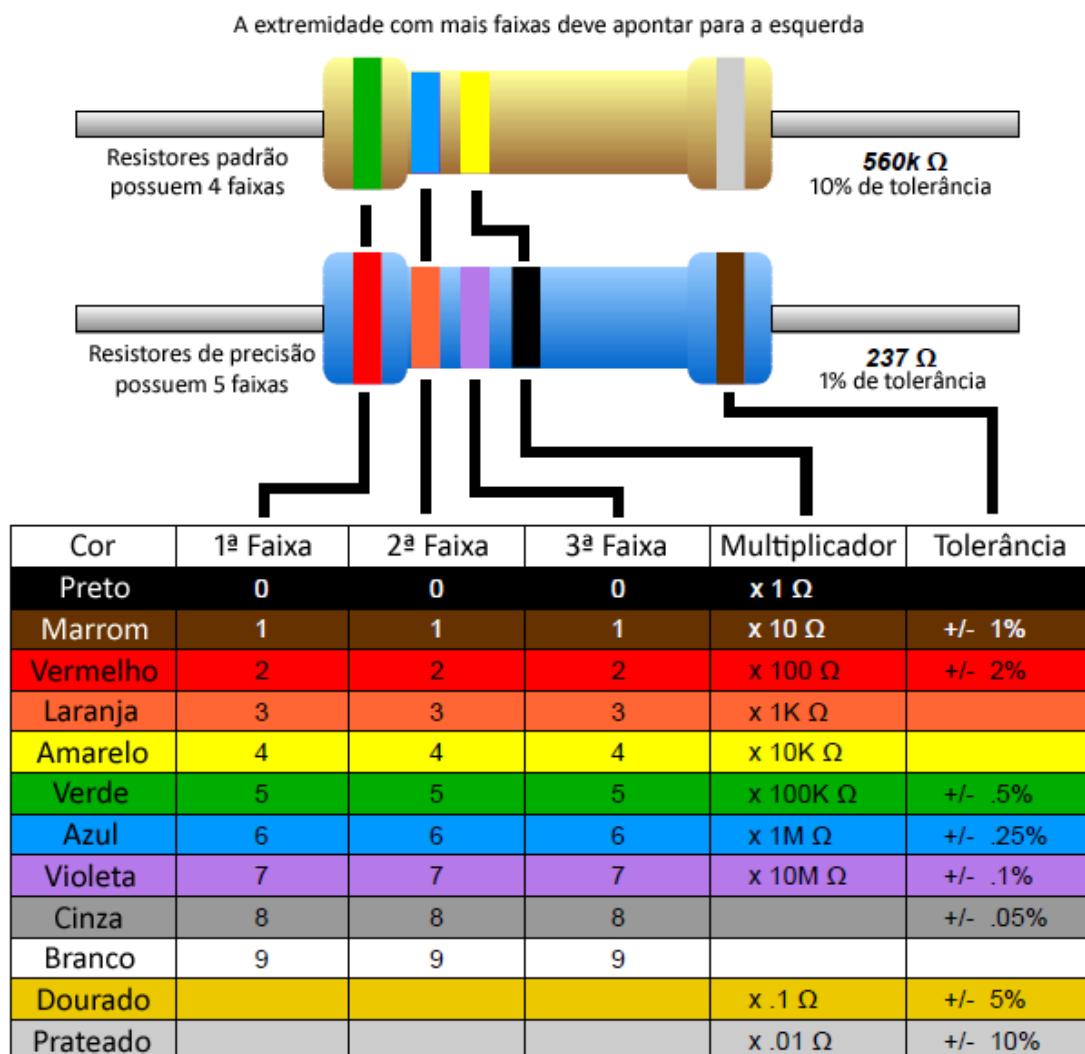
5.1 Resistor



Este componente serve para limitar a corrente elétrica que passa pelo circuito. A quantidade de corrente limitada depende do valor de resistência do resistor. Você pode pensar em um resistor como um cano de água muito mais fino do que o cano conectado a ele. Conforme a água (ou a corrente elétrica) entra no resistor, o cano se torna mais fino e o volume da água (corrente) saindo na

outra ponta é, dessa forma, reduzido. Você utiliza resistores para diminuir a voltagem ou a corrente para outros dispositivos.

O valor da resistência de um resistor é apresentado por faixas coloridas conhecidas como código de cores de resistores. Cada faixa de cor representa um algarismo. Veja a seguir a tabela de código de cores usada para a leitura destes resistores.



Em seu material didático existem três valores de resistores diferentes, 220Ω, 300 Ω e 10KΩ, que poderão ser padrão ou de precisão, tente identificar qual é qual através da leitura do código de cores. Em nossas experiências vamos utilizar diferentes valores de resistores e é importante que você saiba como identifica-los.

5.2 - Led



Este componente também é conhecido como diodo emissor de luz. Ele funciona igual a uma pequena lâmpada que emite luz quando uma corrente elétrica flui através de seus terminais. A intensidade do feixe de luz depende da quantidade de corrente que circula através de seus terminais.

Este componente possui polaridade, isso significa que ele possui um terminal positivo e um terminal negativo, a inversão destes polos pode ocasionar a queima do seu componente, portanto fique atento à polaridade antes de ligá-lo. O terminal maior é o positivo e é chamado Anodo e o terminal menor é o negativo chamado de Catodo, portanto a corrente deve circular do terminal maior para o menor.

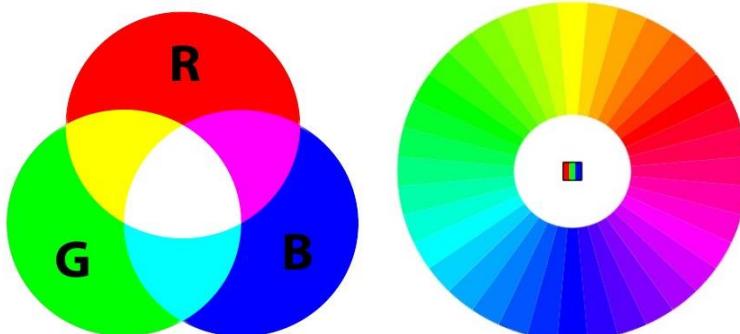
Para funcionar um LED precisa ter sobre ele 2V e 20mA. Como vamos liga-lo sempre a um pino de nossa placa Arduino que possui 5V, devemos fazer um divisor de tensão com um resistor. Abordaremos este assunto mais detalhadamente em nosso primeiro experimento.

5.3 - Led RGB



O LED RGB é a junção de três Leds em um só, ele é formado por um vermelho (R de red), um verde (G de green) e um azul (B de blue). Associando as cores dos três Leds é possível se obter um verdadeiro show de luzes utilizando apenas um led.

Veja a seguir as possibilidades de cores que podem ser obtidas.



5.4 - Potenciômetro



Este componente é um resistor variável. Sua resistência varia conforme a haste central é rotacionada. Ele possui três terminais e a resistência varia sempre entre o terminal do meio e uma das extremidades.

5.5 - Chave Táctil de 4 terminais ou Botão

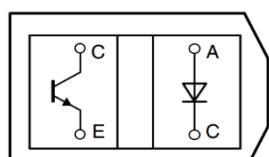


Este componente é uma chave aberta que quando pressionada, fecha e interliga seus dois terminais fazendo com que a corrente elétrica circule através do circuito.

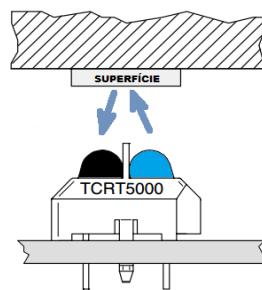
5.6 - Sensor óptico TCRT5000



Este componente é um sensor constituído de dois Leds, um emissor e outro receptor de raios infravermelhos. Seu funcionamento é bastante simples, consiste em emitir um feixe de raios infravermelhos através do emissor para que este ao tocar uma superfície reflita de volta para o receptor e através deste processo se possa detectar uma variação de cor ou tipo da superfície. Esta é apenas umas das diversas aplicações para este sensor.



Vista



É importante que você saiba identificar os terminais do LED emissor e do transistor receptor, pois uma inversão poderá acarretar a queima deste sensor.

5.7 - Sensor de luminosidade LDR



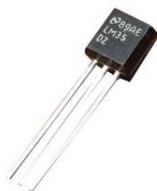
O LDR é um sensor de luminosidade. Ele varia sua resistência de acordo com a luminosidade que incide sobre sua superfície. Onde pouca luminosidade a resistência aumenta e alta luminosidade, a resistência diminui. Este componente é um resistor variável, por este motivo, assim como o resistor ele não possui polaridade, ou seja, seus terminais podem ser ligados de forma indiferente, pois não há definição de positivo e negativo.

5.8 - Sensor de temperatura NTC



O NTC é um sensor de Temperatura. De forma similar ao LDR, ele varia sua resistência de acordo com a temperatura que incide sobre sua superfície. Onde pouca temperatura a resistência aumenta e em alta temperatura, a resistência diminui. Este componente é um resistor variável, por este motivo, assim como o resistor ele não possui polaridade, ou seja, seus terminais podem ser ligados de forma indiferente, pois não há definição de positivo e negativo.

5.9 - Sensor de temperatura linear LM35



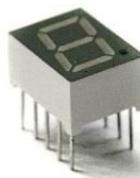
O sensor LM35 é um sensor de temperatura de precisão, que apresenta uma saída de tensão linear relativa à temperatura em que ele se encontrar no momento em que for alimentado por uma tensão de 4-20Vdc e GND, tendo em sua saída um sinal de 10mV para cada Grau Celsius de temperatura, sendo assim, apresenta uma boa vantagem com relação aos demais sensores de temperatura. Ele pode fornecer com exatidão, valores temperatura com variações de 0,25°C dentro da faixa de temperatura de -55°C à 150°C.

5.10 - Buzzer



O Buzzer é um emissor de som que se assemelha a uma campainha. Quando uma corrente circula através de seus terminais ele emitem um som de intensidade proporcional à corrente elétrica.

5.11 - Display BCD de 7 segmentos



O Display BCD de 7 segmentos é bem conhecido por ser largamente utilizado em painéis de senhas e relógios eletrônicos. Este componente nos permite formar números e letras através da combinação de suas linhas que por sua vez são formadas por Leds com seus terminais negativos interligados entre si. Para entender melhor seu funcionamento consulte o **Projeto 11** do Guia Pratico que acompanha seu material didático.

5.12 - Circuito Integrado 4511



Este componente converte um código binário em informações para o display de 7 segmentos. O mesmo será utilizado para facilitar na elaboração e entendimento do código fonte da experiência e para economizar pinos da placa Arduino.

5.13 - Display LCD 16x2



Este componente permite que nós seres humanos possamos ler dados que estão dentro do Arduino. Ele possui 2 linhas de que comportam até 16 caracteres que podem ser preenchidos por letras e números. Para liga-lo use como referência a ligação utilizada no projeto 13 do **Guia Prático** que acompanha seu material didático.

5.14 - Motor DC



Este componente converte energia elétrica em energia mecânica. É um motor com dois terminais que ao aplicar 5V ele gira o eixo. Para inverter seu sentido de rotação basta inverter o positivo e o negativo em seus terminais.

5.15 - Circuito integrado L293D



O L293D é um circuito integrado para controle de motores DC, ele é comumente chamado de ponte-H. Ele permite controlar motores de até 36V de alimentação. Cuidado! O L293D suporta corrente constante de 600mA e corrente pico em torno de 1.2A. Portanto não o utilize para controlar motores DC que exijam mais do que 600mA.

5.16 - Microservo 9g



O micro servo também é um motor, mas só realiza movimentos de 180º com seu eixo nos dois sentidos, ele é bastante utilizado em robótica para posicionamento e para movimentar articulações de robôs e braços robóticos. Apesar de possuir uma extensa e complexa teoria acerca de seu funcionamento, sua utilização com o Arduino é bastante simples.

5.17 - Módulo Relé



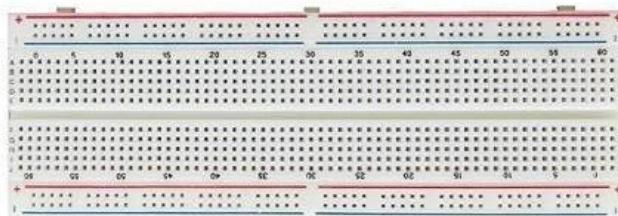
O módulo relé é uma interface para adaptar os pinos do Arduino para ligar outros aparelhos com uma tensão maior que 5V. Ele permite por exemplo que você ligue ou desligue uma lâmpada 220V com sua placa Arduino que é alimentada com apenas 5V.

5.18 - Cápsula Piezoelétrica ou piezo buzzer

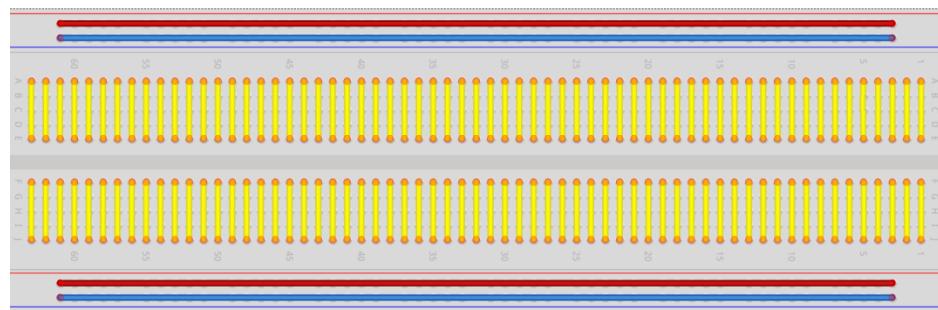


A capsula piezoelétrica é semelhante à um buzzer, porém possui outro formato e serve tanto para emitir sons quanto como um sensor de toque. Ele é capaz de gerar corrente elétrica a partir de uma pressão física em seu corpo, ou seja, através desta corrente, é possível detectar se existe ou não algo sobre o mesmo.

5.19 - Protoboard



O protoboard é uma placa para montagem de seus circuitos eletrônicos. Ele possui diversos furos para inserção de componentes e estes furos por sua vez estão interligados de forma a facilitar as conexões entre os componentes. Os furos das barras das extremidades identificados pelas linhas azul e vermelha estão interligados na horizontal e os furos da parte central estão interligados na horizontal conforme esquema a seguir onde cada fio indica a ligação interna correspondente.



6 – A linguagem de programação

Para programar o Arduino, utilizamos uma linguagem de programação baseada em C e C++ e geralmente o código segue a sequência a seguir:

- Funções principais;
- Variáveis;
- Operadores booleanos, de comparação e aritméticos;
- Estrutura de controle;
- Funções digitais e analógicas.

6.1 - Sintaxe

Cada uma dessas partes do código fonte, são compostas por comandos escritos que possuem uma maneira correta de serem escritos, a isso damos o nome de sintaxe. E para que você entenda a sintaxe é importante você conhecer a função dos elementos que compõe essa sintaxe.

Neste material vamos abordar as principais funções e necessárias para que você explore todo o seu material didático.

6.1.1 – Chaves {}

As chaves definem o início e o fim de um bloco de função ou bloco de declaração como por exemplo a função **void loop()**, ou as declarações **for** e **if**.

```
void setup()
{
    pinMode(pino, OUTPUT); //Configura um pino como saída
}
```

Uma chave aberta deve sempre ser seguida de uma chave fechada. Chaves abertas que não são fechadas podem acarretar erros ocultos e impenetráveis no compilador que às vezes podem ser difíceis de encontrar. O ambiente de programação do Arduino possui uma funcionalidade que verifica se as chaves abertas foram fechadas corretamente.

6.1.2 – Ponto e Vírgula ;

O ponto e vírgula deve ser usado para finalizar uma declaração e separar elementos do programa.

```
int x=13 //Declara variável X como o inteiro 13
```

Importante: Esquecer de finalizar uma linha com o ponto e vírgula vai resultar em um erro no compilador. O texto do erro pode ou não ser óbvio e falar que falta um ponto e vírgula. Caso você tenha um erro impenetrável ou sem lógica no compilador, uma das primeiras coisas que você deve desconfiar é que falta um ponto e vírgula próximo de onde o compilador informar o erro.

6.1.3 – Blocos de Comentários /* ... */ ;

Blocos de comentários ou comentários multilinhas, são áreas de texto ignoradas pelo programa, e são usadas para descrever partes do código ou comentários que possam auxiliar outras pessoas a entender o código. Eles começam com /* e terminam com */ e podem ter múltiplas linhas.

```
/* este é um bloco de comentário  
não esqueça de fechar o comentário */
```

Como os comentários são ignorados pelo programa e não utilizam nenhum espaço na memória, eles podem ser utilizados à vontade.

6.1.4 – Linhas de Comentários // ;

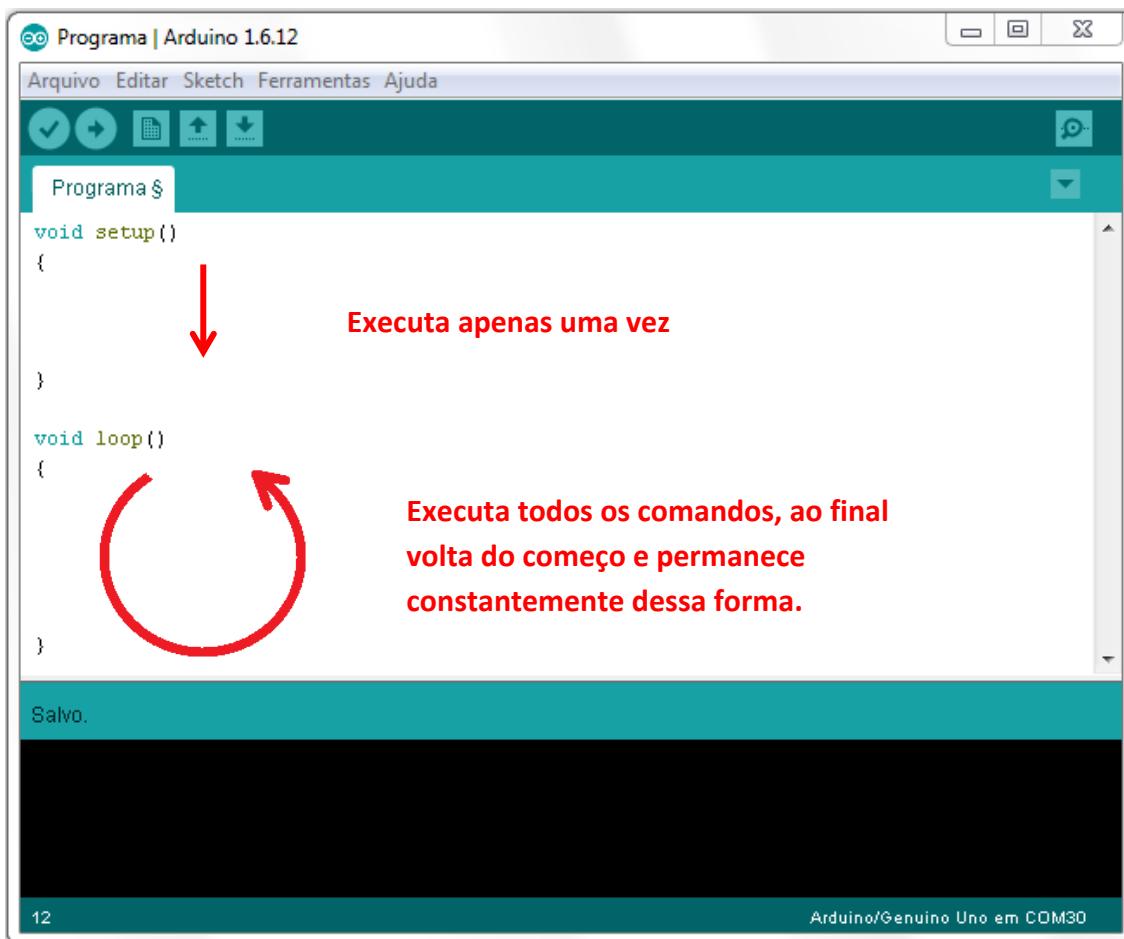
Linhos únicas de comentários começam com // e terminam ao final dessa mesma linha. Linhas de comentários também são ignoradas pelo programa e não ocupam espaço na memória.

```
// este é uma linha de comentário
```

Linhos de comentários são comumente utilizadas após uma instrução válida para fornecer mais informações sobre o que aquela instrução faz ou servir de lembrete futuro.

6.2 – Funções Principais

As funções principais são as que compõem a estrutura da linguagem de programação Arduino. São duas as funções principais, `setup()` e `loop()`.



6.2.1 - Função `setup()`

A função `setup()` é executada quando um esboço começa. A função `setup()` será executada apenas uma vez após cada energização ou reset da placa Arduino e ela é usada para inicializar variáveis, configurar pinos, começar a usar bibliotecas, etc..

```
void setup()
{
    instruções;
}
```

6.2.2 - Função loop()

A função `loop()` faz exatamente o que seu nome sugere, ela fica em loop constantemente, permitindo que o seu programa execute tarefas continuamente. É nessa parte que inserimos o conteúdo ativo (ligar e desligar pinos, ler sensores e valores externos e tomar decisões) de nosso programa.

```
void loop()
{
    digitalWrite(pino, HIGH); //Liga um pino
    delay(1000);             //Aguarda 1 segundo
    digitalWrite(pino, LOW);  //Desliga um pino
    delay(1000);             //Aguarda 1 segundo
}
```

6.3 - Funções

Uma função é um bloco de código que possui um nome e um bloco de declarações que são executadas quando a função é chamada. As funções "`void setup()`" e "`void loop()`" já foram explicadas acima e demais funções embutidas serão explicadas futuramente.

Funções personalizadas podem ser criadas para executar tarefas repetitivas e manter o código organizado. Funções são declaradas primeiramente declarando-se o tipo de função. Este é o tipo de valor que deve ser retornado pela função como por exemplo "`int`" para um tipo de função inteira. Caso a função não deva retornar nenhum valor o tipo de função deve ser "`void`". Após o tipo, declare o nome dado à função e em parênteses qualquer parâmetro que deva ser passado para a função.

```
tipo nomeDaFunção(parâmetros)
{
    instruções;
}
```

A seguinte função inteira `delayVal()` é usada para criar uma contagem de tempo ou um atraso em um programa através da leitura de um potenciômetro. Ela primeiro declara uma variável local "`v`", atribui a `v` o valor lido no potenciômetro que pode variar de 0 a 1023, então divide o valor por 4 para um resultado final entre 0 e 255, e finalmente retorna o resultado para o programa principal.

```
int delayVal()
{
    int v;                      //cria uma variável temporária "v"
    V = analogRead(pot);        //Lê o valor do potenciômetro
    V /= 4;                     //divide o valor por 4
    return v;                   //retorna o resultado
}
```

7 – Variáveis

Uma variável é uma maneira de dar nome e armazenar um valor numérico para ser usado posteriormente no programa. Como o próprio nome sugere, variáveis são números que podem ser constantemente modificados, o oposto das constantes que seus valores nunca mudam. Uma variável precisa ser declarada e opcionalmente atribuída a ela um valor para ser armazenado. O código a seguir declara uma variável chamada **variavelEntrada** e atribui a ela o valor obtido na entrada analógica 2:

```
int variavelEntrada = 0;           //Declara uma variável  
                                    //atribui o valor 0 a ela  
variavelEntrada = analogRead(2); //escreve nessa variável o  
                                //valor da entrada analógica 2
```

- **variavelEntrada** é a variável em si. A primeira linha declara que nela vai conter uma **int** (valor inteiro). A segunda linha escreve na variável o valor obtido na entrada analógica 2. Desta forma o valor da entrada analógica dois fica armazenado e acessível em outro lugar do programa.

Uma vez que um valor foi atribuído a uma variável, ou reescrito, você pode testar este valor para ver se ele atende a alguma condição, ou você pode usar esse valor diretamente. Como exemplo para ilustrar três operações úteis com variáveis, o código a seguir testa se a "**variavelEntrada**" é menor que 100, se for verdade atribui o valor 100 a "**variavelEntrada**", e então estabelece um delay baseado na "**variávelEntrada**" que agora é 100:

```
if (variavelEntrada < 100) //Testa se a variável é menor que 100  
{  
    variavelEntrada = 100; //Se verdadeiro, atribui o valor 100  
}  
delay(variavelEntrada);    //Usa a variável como delay
```

Importante: Variáveis devem receber nomes descritivos, para tornar o código mais fácil de ser lido e interpretado. Nomes de variáveis como "tiltSensor" ou "pushButton", ajudam o programador e outras pessoas a ler o código e entender o que a variável representa. Já variáveis com nomes como "var" ou "valor", não ajudam muito a entender o código. Uma variável pode usar qualquer palavra como nome desde que não seja nenhuma da palavras chave da linguagem Arduino.

7.1 - Declaração de variáveis

Todas as variáveis devem ser declaradas antes de serem usadas. Declarar uma variável significa definir seu tipo (qual tipo de valor será armazenado nela), como int, long, float, etc., dar um nome, e opcionalmente atribuir um valor inicial. Isso só precisa ser feito uma vez em um programa, mas o valor da variável pode ser modificado a qualquer momento utilizando aritmética e varias outras atribuições.

O exemplo a seguir declara que "variavelEntrada" é uma int, ou tipo inteira, que que seu valor inicial é igual a zero. Isto é chamado de atribuição simples.

```
int variavelEntrada = 0;
```

Uma variável pode ser declarada em diversos pontos ao longo do programa e onde essa declaração estiver determina quais partes do programa podem usar a variável.

7.2 - Escopo de Variáveis

Uma variável pode ser declarada no início do programa antes da função void setup(), localmente dentro de funções, e às vezes em declarações como por exemplo em um loop "for". Onde a variável é declarada determina o escopo da variável ou a capacidade de certas partes do programa acessar e usar a variável.

Uma variável global é uma que pode ser vista e utilizada por todas as funções e declarações em um programa. Esta variável é declarada no início do programa, antes da função void setup().

Já uma variável local é uma que é definida dentro de uma função ou como parte de um loop for. Ela só é visível e só pode ser usada dentro da função na qual ela foi declarada. Portanto é possível se ter duas ou mais variáveis com o mesmo nome em diferentes partes de um mesmo programa que contenham valores diferentes. Garantindo que somente uma função tem acesso à suas próprias variáveis simplifica o programa e reduz a possibilidade de erros de programação.

O exemplo a seguir mostra como declarar alguns tipos diferentes de variáveis e demonstra a visibilidade de cada variável:

```
int valor;           // "variável" valor é visível
                     // a todas as funções
void setup()
{
    // não necessário usar o setup
}

void loop()
{
    for (int i=0; i<20;) // "i" só é visível
    {
        // dentro do loop for
        i++;
    }
    float f;           // "f" só é visível
    // dentro do loop()
```

7.3 - Tipos de Variáveis

Ao declarar uma variável devemos informar o tipo de variável. Isso significa informar que tipo de dados serão armazenados nessa variável. Para cada tipo de dado existe uma variável específica para armazená-lo. Isso existe para que possamos otimizar nosso código fonte e utilização do microcontrolador já que maiores valores ou dados mais complexos como valores negativos, ou valores decimais além de ocuparem maior espaço de memória para armazená-los, consomem mais ciclos de processamento. Dessa forma, se temos um valor pequeno ou simples utilizamos uma variável que consome menos de nosso microcontrolador fazendo com que nosso programa apresente um melhor desempenho.

7.3.1 - byte

Um byte armazena um número de 8 bits sem sinal, de 0 a 255. Ou seja uma variável do tipo byte só pode armazenar valores de 0 até 255.

```
byte valor = 0;
```

7.3.2 - int

Inteiro é o tipo de dado primário para o armazenamento de números. No Arduino Uno (e outras placas baseadas em ATmega), um int armazena um valor de 16 bits (2 bytes) ou de -32768 a 32767. Ou seja, uma variável do tipo int pode armazenar valores de -32768 à 32767.

```
int valor = 0;
```

7.3.3 – unsigned int

Unsigned ints (inteiros sem sinal) são os mesmos que ints em que armazenam um valor de 2 bytes ou 16 bits, só que ao invés de armazenar números negativos no entanto, eles armazenam apenas valores positivos, de 0 a 65.535.

```
unsigned int valor = 0;
```

7.3.4 - char

Um tipo de dado que ocupa 1 byte de memória e armazena o valor de um caractere ASCII. Caracteres literais são escritos entre aspas.

O tipo de dados que ocupa 1 byte de memória que armazena um valor de caractere. Literais de caracteres são escritos em aspas simples, como este: 'A' (para caracteres múltiplos - sequências de caracteres - use aspas: "ABC").

No entanto, os caracteres são armazenados como números. Você pode ver a codificação específica na tabela ASCII. Isso significa que é possível fazer aritmética em caracteres, nos quais o valor ASCII do caractere é usado (por exemplo, 'A' + 1 tem o valor 66, já que o valor ASCII da letra maiúscula A é 65).

```
char meuChar = 'A';
```

Ou ainda

```
char meuChar = 65;
```

Os dois são equivalentes

7.3.5 – word

Uma variável do tipo word armazena dados de 16 bits ou dois bytes sem sinal.

```
word valor = 10000;
```

7.3.6 – long

As variáveis do tipo long são variáveis de tamanho estendidas para armazenamento de valores de 16 bits com sinal, ou seja de -2.147.483.648 a 2.147.483.647.

```
long velocidadeDaLuz = 186000;
```

7.3.7 – unsigned long

As variáveis do tipo unsigned long (long sem sinal) são variáveis de tamanho estendidas para armazenamento de números de 32 bits sem sinal, ou seja de 0 a 4.294.967.295.

```
unsigned long velocidadeDaLuz = 5000000;
```

7.3.8 – float

Variáveis no tipo float são utilizadas para armazenar números de ponto flutuante, um número que tem um ponto decimal. Números de ponto flutuante são freqüentemente usados para aproximar valores analógicos e contínuos porque eles têm maior resolução do que números inteiros. Os números de ponto flutuante podem ser tão grandes que são armazenados como 32 bits (4 bytes) de informações.

A matemática de ponto flutuante também é muito mais lenta do que a matemática de número inteiro na execução de cálculos como explicado no início deste capítulo, portanto, deve ser evitada se, por exemplo, um loop tiver que ser executado na velocidade máxima para uma função cronológica crítica. Programadores muitas vezes vão para converterem cálculos de ponto flutuante para inteiros para aumentar a velocidade de execução do programa.

Ao realizar cálculos matemáticos com variáveis float, você precisará adicionar um ponto decimal, caso contrário ele será tratado como um int.

```
float offset = 1.145;
```

Importante: Quando o valor de uma variável exceder sua capacidade máxima ocorre o estouro dessa variável, ou seja, ela retorna ao seu valor mínimo. Por exemplo, no caso de uma variável do tipo **byte** que está com valor igual a 255, se a incrementarmos em 1 o seu valor será 0. Já no caso de uma variável do tipo **int** com valor igual a 32767, se a incrementarmos em 1, o seu valor será -32768.

8 - Aritmética

Operadores aritméticos incluem adição, subtração, multiplicação e divisão. Eles retornam a soma, a diferença, o produto ou o quociente (respectivamente) de dois valores ou variáveis.

```
y = y + 3;  
x = x - 7;  
i = j * 6;  
r = r / 5;
```

A operação é conduzida usando o tipo de dados dos operandos, então por exemplo, $9 / 4$ resulta em 2 ao invés de 2,25 já que 9 e 4 são ints e por esse motivo são incapazes de usar casas decimais. Isso também significa que uma operação pode dar overflow caso o resultado seja maior que o que pode ser armazenado no tipo de variável.

9 – Operadores de comparação

Comparações de uma variável ou constante com outras são bastante usadas em declarações do tipo “if” para testar se uma condição específica é verdadeira.

```
x == y          // Compara se x é igual a y  
x != y          // Compara se x é diferente de y  
x < y          // Compara se x é menor que y  
x > y          // Compara se x é maior que y  
x <= y         // Compara se x é menor ou igual a y  
x >= y         // Compara se x é maior ou igual a y
```

10 – Operadores lógicos

Operadores lógicos são usualmente a maneira de comparar duas expressões e retornar um TRUE (verdadeiro) ou FALSE (falso), dependendo do operador. Existem três operadores lógicos, AND (E), OR (OU) e NOT (NÃO), que são bastante usados em declarações:

Lógica AND (E):

```
if (x > 0 && x < 5)      //Verdadeiro somente se ambas as comparações  
                         // forem verdadeiras
```

Lógica OR (OU):

```
if (x > 0 || y > 0)      //Verdadeiro se qualquer uma das comparações  
                         // for verdadeira
```

Lógica NOT (NÃO):

```
if (!x > 0)              //Verdadeiro somente se a comparação  
                         // for falsa
```

11 – Constantes

A linguagem Arduino possui alguns valores pré-definidos, que são chamados de constantes. Eles são usados para tornar os programas fáceis de ler. Constantes são classificadas em grupos.

11.1 – true / false

Estas são constantes que definem níveis lógicos. FALSE é facilmente definido como 0 (zero) enquanto TRUE é frequentemente definido como 1, mas também pode ser qualquer valor diferente de zero. Desta forma, -1, 2 e -200 são todos definidos como TRUE.

```
if (b == true);           // Se b for verdadeiro
{
    doSomething;         // faça algo
}
```

11.2 – HIGH / LOW

Estas são constantes definem níveis lógicos de pinos como HIGH (1) ou LOW (0) e são usados ao escrever ou ler os pinos digitais. HIGH é definido como nível lógico 1 ou LIGADO ou 5V enquanto LOW é definido como nível lógico 0 ou DESLIGADO ou 0 volts.

```
digitalWrite(13, HIGH);
```

11.3 – input / output

Estas são constantes são usadas com a função pinMode() para definir o modo de operação de um pino digital como entrada (INPUT) ou saída (OUTPUT).

```
pinMode(13, OUTPUT);
```

12 – Estruturas de controle

Uma estrutura de controle é um bloco de função que analisa variáveis e toma uma decisão com base nos parâmetros dados.

12.1 – if

Declarações if testam se uma certa condição foi atendida, tal como um valor analógico estar acima de um certo número, e executa qualquer declaração dentro das chaves se a declaração for verdadeira (true). Se for falsa, o programa pula essa declaração. O formato para um teste if é:

?? Pode ser:
== - Igual a
!= - Diferente de
> - Maior que
< - Menor que
<= - Menor ou igual a
>= - Maior ou igual a

```
if (algumaVariavel ?? valor);  
{  
    executeAlgo;  
}
```

O exemplo anterior compara “algumaVariavel” com “valor”, que pode ser uma variável ou constante. Se a comparação ou condição em parênteses for verdadeira, as declarações dentro das chaves são executadas. Se não, o programa pula e continua após as chaves.

IMPORTANTE: Cuidado para não utilizar acidentalmente “=”, como por exemplo “if (x=10), pois enquanto tecnicamente válido, define a variável x com o valor de 10 e o resultado desta operação é sempre verdadeiro (true). Ao invés disso use “==”, como por exemplo “if (x==10)”, que testa apenas se x é ou não igual a 10. Pense em ‘=’ como “igual” e ‘==’ como sendo “é igual a”.

12.2 – if... else

If... else permite se tomar decisões. Por exemplo, se você quer verificar o estado de uma entrada digital e fazer uma coisa caso ela esteja em HIGH e outra coisa caso esteja em LOW, você deve escrever o seguinte código:

```
if (pino == HIGH);           // Se o pino digital estiver em HIGH  
{  
    executeCoisaA;          // Executa função A  
}  
else                         // Caso contrário  
{  
    executeCoisaB;          // Executa função B  
}
```

'Else' pode ainda vir antes de um if, de modo que múltiplos testes exclusivos possam ser executados ao mesmo tempo. Desta forma cada verificação irá prosseguir para a próxima até que uma delas seja verdadeira. Quando uma verificação é verdadeira, o bloco de código associado (que está entre as chaves logo abaixo) é executado, e o programa, em seguida, pula para a linha após o fechamento de chaves executado. Se nenhum teste for verdadeiro e existir um bloco else , o bloco else é executado.

Observe que um bloco else if pode ser usado com ou sem um bloco de terminação else e vice-versa. Um número ilimitado de 'else if' é permitido.

```
if (valor < 500);           // Se valor for menor que 500
{
    executeCoisaA;        // Executa função A
}
else if (valor >= 1000);   // Se valor for maior ou igual a 1000
{
    executeCoisaB;        // Executa função B
}
else                      // Caso contrário
{
    executeCoisaC;        // Executa função C
}
```

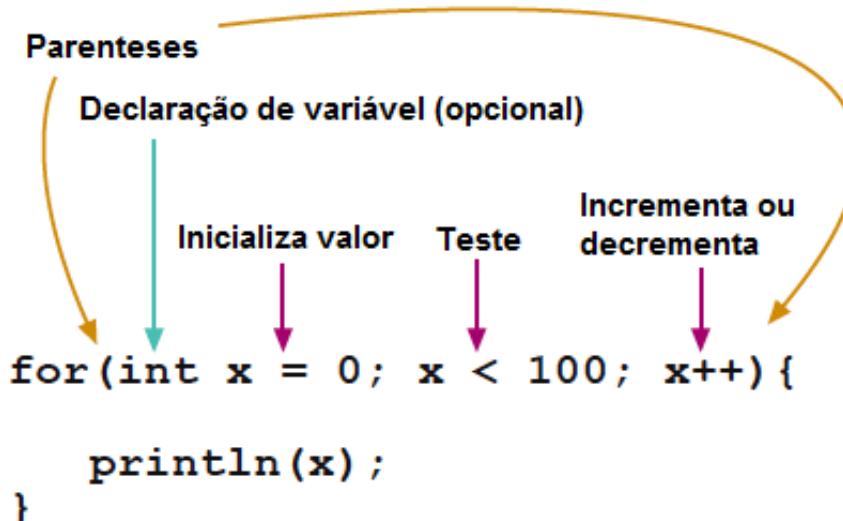
Importante: Uma declaração if simplesmente testa se a condição entre parênteses é verdadeira ou falsa. Esta condição pode ser qualquer de declaração válida em linguagem C, como no primeiro exemplo *if (pino == HIGH)* que a declaração if apenas verifica se de fato a entrada digital está em nível lógico HIGH ou com 5V.

12.3 – for

A declaração for é usada para repetir um bloco de declarações fechado por chaves por uma determinada quantidade de vezes. Um contador de incrementos é utilizado para incrementar e terminar o loop. O comando for é composto por três partes, separadas por ponto e vírgula:

```
for (inicialização; condição; expressão)
{
    executeAlgo;
}
```

Exemplo: O exemplo a seguir imprime o valor de 0 a 99.



A inicialização acontece primeiro e apenas uma vez. Cada vez através do loop, a condição é testada; Se for verdade, o bloco de instrução e o incremento são executados, então a condição é testada novamente. Quando a condição se torna falsa, o loop termina.

12.4 – while

Loops while são executados continuamente, e infinitamente, até que a expressão dentro do parêntese, () torne-se falsa. Algo deve mudar a variável testada, ou o loop while nunca sairá. Isso pode estar em seu código, como uma variável incrementada ou uma condição externa, como testar um sensor.

```

while (algumaVariável ?? valor)
{
    executeAlgo;
}

```

Exemplo: O exemplo a seguir testa se ‘variável’ é menor que 200 e se for verdadeira executa as declarações entre chaves e permanece executando até que “variável” não seja mais menor que 200.

```

while (Variavel < 200) // Testa se variável é menor que 200
{
    executeAlgo; // Executa função
    variável++; // Incrementa variável em 1
}

```

12.5 – do - while

O loop do funciona da mesma maneira que o loop while com a exceção de que a condição é testada no final do loop, dessa forma o loop sempre será executado pelo menos uma vez.

```
do
{
    executeAlgo;
} while (algumaVariável ?? valor)
```

Exemplo: O exemplo a seguir atribui ‘leituraSensor()’ à variável ‘x’, pausa por 50 milisegundos, e então permanece executando o loop até que ‘x’ não seja mais menor que 100:

```
do
{
    x = leituraSensor();      // Atribui o valor de
                            //leituraSensor() a x
    delay(50);               //pausa 50 milisegundos
} while (x < 100)           //permanece em loop enquanto
                            //x for menor que 100
```

12.6 – switch case

Da mesma forma que as declarações do tipo 'if', o 'switch ... case' controla o fluxo de programas permitindo que os programadores especifiquem códigos diferentes que devem ser executados em várias condições. Em particular, uma instrução switch compara o valor de uma variável com os valores especificados em instruções case. Quando uma instrução case é encontrada cujo valor coincide com o da variável, o código nessa instrução case é executado.

O comando 'break' é utilizado para sair de um 'switch' e normalmente é usada no final de cada case. Sem um comando break, a instrução switch continuará executando as seguintes expressões até que um break seja executado ou o fim da instrução switch seja atingido.

```
switch (variavel) {  
    case 1:  
        // executa algo se variavel for igual a 1  
        break;  
    case 2:  
        // executa algo se variavel for igual a 2  
        break;  
    default:  
        // se não for igual a nenhum valor dos cases acima  
        // executa o default  
        // a utilização do default é opcional  
        break;  
}
```

13 – Funções para pinos digitais

Estas funções são basicamente as funções que nos permite ligar ou deligar um pino de saída digital e ler o estado de um pino de entrada digital.

São elas:

- pinMode()
- digitalWrite()
- digitalRead()

13.1 - pinMode()

O comando pinMode() serve para informar para nosso microcontrolador qual a função de um determinado pino da placa Arduino, se o mesmo irá operar com uma entrada ou como uma saída. O comando pinMode() deve ser usado da seguinte maneira:

```
pinMode (número do pino, função do pino);
```

Exemplos: `pinMode(13, OUTPUT); //Configura pino 13 como saída`
`pinMode(13, INPUT); //Configura pino 13 como entrada`

13.2 - digitalWrite()

O comando digitalWrite() serve para ligar ou desligar um pino digital da nossa placa Arduino. O comando digitalWrite() deve ser usado da seguinte maneira:

```
digitalWrite (pino, tarefa);
```

Exemplo: `digitalWrite(13, HIGH); //Liga o pino 13`
`digitalWrite(13, LOW); //Desliga o pino 13`

13.3 - digitalRead()

O comando digitalRead() serve para ler o estado de um pino digital da nossa placa Arduino. Através dessa leitura podemos ler dois estados HIGH (+5V) ou LOW (0V).

```
variavel = digitalRead (pino);
```

Exemplo: `valor = digitalRead(13); //lê o valor do pino 13 e armazena`
`//na variável 'valor'`

Veja a seguir um exemplo de aplicação das três funções para pinos digitais. Neste exemplo temos um botão ligado a uma entrada digital e um led ligado a uma saída digital. Então o programa faz a leitura do estado da entrada digital e o estado lido é enviado para a saída digital, dessa forma, se o botão for pressionado o led acende e se o botão for liberado o led se apaga. Vejamos:

```
int pinoLed = 13;           // dá o nome pinoLed ao pino digital 13
int pinoBotao = 7;          // dá o nome pinoBotao ao pino digital 7
int estado = 0;             // Cria a variável estado de valor inicial 0

void setup()
{
    pinMode(pinoLed, OUTPUT);      // configura pino 13 como saída
    pinMode(pinoBotao, INPUT);     // configura pino 7 como entrada
}

void loop()
{
    estado = digitalRead(pinoBotao); // lê o estado do pino 13 e armazena na
                                     // variável estado
    digitalWrite(pinoLed, estado);   // envia para o pino 13 o mesmo estado
                                     // lido no pino 7
}
```

14 – Funções para pinos analógicos

Estas funções como as dos pinos digitais servem para controle de pinos analógicos, ler o valor de um pino analógico e escrever valores analógicos nas saídas PWM.

São elas:

- analogRead()
- analogWrite()

14.1 - analogRead()

O comando analogRead() serve para ler o valor de tensão de uma entrada analógica da placa Arduino. A placa Arduino possui um conversor analógico para digital de 10 bits. Isto significa que mapeará tensões de entrada entre 0 e 5 volts em valores inteiros entre 0 e 1023. Isso produz uma resolução entre leituras de: 5 volts / 1024 unidades ou .0049 volts (4,9 mV) por unidade.

Para facilitar basta ter em mente que 5V é lido como 1023 e 0V é lido como 0 e desta forma proporcionalmente os valores entre 0 e 5V, como por exemplo 2,5V que será lido como 511 e assim sucessivamente.

Demora cerca de 100 microssegundos (0,0001 s) para ler uma entrada analógica, então a taxa de leitura máxima é cerca de 10.000 vezes por segundo.

```
variavel = analogRead (pinoAnalogico);
```

Exemplo: valor = analogRead(A0); //lê o valor do pino A0 e armazena //na variável 'valor'

14.2 - analogWrite()

Escreve um valor analógico (onda PWM) em uma saída PWM da placa Arduino. Pode ser usado para acender um LED em variações de luminosidade ou conduzir um motor a várias velocidades. Depois de executar um comando analogWrite(), o pino irá gerar uma onda quadrada constante do ciclo de trabalho especificado e permanecerá até que receba um comando diferente no mesmo pino. A freqüência do sinal PWM é de aproximadamente 490 Hz. No Arduino Uno e semelhantes, os pinos 5 e 6 têm uma freqüência de aproximadamente 980 Hz. Os pinos 3 e 11 no Leonardo também funcionam a 980 Hz.

Na maioria das placas Arduino (aqueles com o ATmega168 ou o ATmega328), esta função funciona nos pinos 3, 5, 6, 9, 10 e 11. No Arduino Mega, funciona nos pinos 2 ao 13 e 44 ao 46. Os Arduino mais antigos que utilizam o ATmega8 só suportam analogWrite() nos pinos 9, 10 e 11.

O Arduino Due suporta analogWrite() nos pinos 2 a 13, mais pinos DAC0 e DAC1. Diferentemente dos pinos PWM, DAC0 e DAC1 são conversores Digital para Analógico e atuam como verdadeiras saídas analógicas.

De forma bastante similar ao comando analogRead() que possui resolução de 10 bits como vimos anteriormente, o comando analogWrite() possui uma resolução de 8 bits, ou seja, devemos enviar valores entre 0 e 255 para a saída PWM, onde 255 corresponde a 5V, 0 a 0V e demais valores de 0 a 5V proporcionalmente de 0 a 255. Como por exemplo para escrever 2,5V em uma saída PWM devemos enviar o valor 127 no comando analogWrite().

```
analogWrite (pino, valor entre 0 e 255);
```

Exemplo: analogWrite(9, 127); // envia o valor de 2,5V para o pino 9

Veja a seguir um exemplo de aplicação das duas funções para pinos analógicos. Neste exemplo temos um potenciômetro conectado a uma entrada analógica e um led conectado a uma saída PWM. O programa então realiza a leitura do valor da entrada analógica (ajustado pelo potenciômetro) e envia o mesmo valor proporcional para a saída PWM (led). Dessa forma o potenciômetro controla a luminosidade do led do mínimo ao máximo. Vejamos:

```
int pinoLed = 9;           // Led conectado ao pino 9
int pinoAnalogico = 3;     // potenciometro conectado ao pino 3
int valor = 0;             // variável valor para armazenar valor analógico
                           // do potenciometro
void setup()
{
    pinMode(pinoLed, OUTPUT); // configura pino digital 9 como saída
}

void loop()
{
    valor = analogRead(pinoAnalogico); // lê o valor do pino analógico
    analogWrite(pinoLed, valor / 4);   // envia para a saída PWM o valor
                                       // analógico lido, dö que dividido
                                       // por 4 pois a entrada analógica
                                       // lê de 0 a 1023 e a saída PWM
                                       // aceita valores de 0 a 255
                                       // (4 vezes menor)
}
```

15 – Funções avançadas

Das funções avançadas veremos as funções para gerar frequência através de pinos digitais.

- tone();
- noTone();

15.1 - tone()

Gera uma onda quadrada da freqüência especificada (e 50% de ciclo de trabalho) em um pino. Uma duração pode ser especificada, caso contrário, a onda continua até que um comando noTone () seja executado. O pino pode ser conectado a um buzzer, capsula piezoelétrica ou outro alto-falante para tocar tons.

Somente um tom pode ser gerado de cada vez. Se um tom já estiver tocando em um pino diferente, o comando tone() não terá efeito. Se o tom estiver tocando no mesmo pino, a chamada ajustará sua frequência para a frequência solicitada. Isso significa que se você quiser jogar diferentes tons em vários pinos, você precisa executar noTone() em um pino antes de executar tone() no próximo pino.

O uso da função tone() interferirá com a saída PWM nos pinos 3 e 11 (em placas diferentes do Mega).

Placa	Frequencia mínima (Hz)	Frequencia Máxima (Hz)
Uno, Mega, Leonardo e outras placas AVR	31	65535
Due	Função não implementada	Função não implementada

```
tone (pino, frequência em Hz);
```

ou ainda

```
tone (pino, frequência em Hz, duração em ms);
```

Exemplo: `tone(9, 1000); // envia 1000Hz ou 1KHz para o pino 9`

```
tone(9, 1000, 1000); // envia 1000Hz ou 1KHz para o pino 9 por  
// 1000 milissegundos ou 1 segundo
```

15.2 - noTone()

Para a geração de uma onda quadrada desencadeada pela função tone(). Não tem efeito se nenhum comando tone() estiver sendo executado.

```
noTone (pino);
```

Exemplo: `noTone(9); // interrompe a geração de frequência no pino 9`

16 – Funções de tempo

As funções de tempo são funções que nos permitem realizar a contagem de tempo ou ainda gerar atrasos ou paradas em na execução de nosso programa.

- millis();
- micros();
- delay();
- delayMicroseconds();

16.1 - millis()

Retorna o número de milissegundos desde que a placa Arduino começou a executar o programa atual. Esse número irá voltar a zero, após aproximadamente 50 dias.

Observe que o valor resultado da função millis () é um unsigned long, erros de lógica podem ocorrer se um programador tentar fazer aritmética com tipos de dados menores, como int por exemplo.

```
variavel = millis();
```

16.3 - micros ()

Retorna o número de microsegundos desde que a placa Arduino começou a executar o programa atual. Este número irá voltar a zero, após aproximadamente 70 minutos.

```
variavel = micros();
```

16.4 - delay ()

Pausa o programa por uma quantidade de tempo (em milissegundos) especificado dentro do parênteses.

```
delay(tempo em milisegundos);
```

16.5 - delayMicroseconds ()

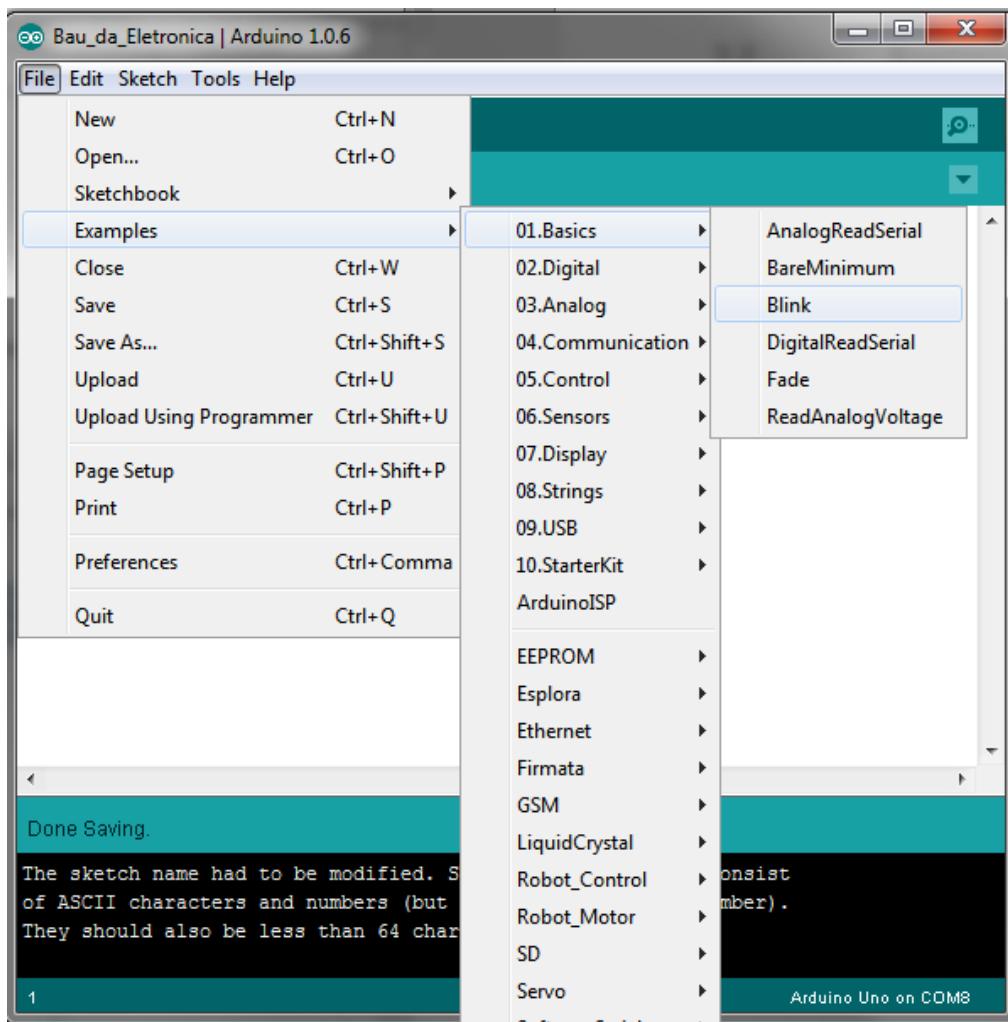
Assim como o delay() , o delayMicroseconds() pausa o programa por uma quantidade de tempo especificado dentro dos parênteses, só que o tempo deve ser expresso em microsegundos. Atualmente, o maior valor que produzirá um atraso preciso é 16383. Para atrasos maiores do que alguns milhares de microsegundos, você deve usar delay () em vez disso.

```
delayMicroseconds (tempo em microsegundos) ;
```

17 – Exemplo de um código fonte

O software IDE de programação do Arduino possui uma série de exemplos prontos para que possamos utilizar como referência. Para te proporcionar um primeiro contato com um código fonte e realizarmos a leitura e entendimento do mesmo, vamos utilizar um dos exemplos prontos do próprio software do Arduino, o exemplo BLINK que serve para piscar o LED da placa Arduino.

Para acessá-lo clique em FILE > EXAMPLES > 1.BASICS > BLINK como mostrado na figura abaixo:



Feito isto, o código fonte do programa irá aparecer na tela do ambiente de desenvolvimento. É interessante que você analise o programa para tentar compreendê-lo. Para tanto, iremos colocar abaixo todo o programa, assim como você deve estar vendo na tela do ambiente de desenvolvimento, para analisá-lo com você.

Código Fonte

```
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
Most Arduinos have an on-board LED you can control. On the Uno and
Leonardo, it is attached to digital pin 13. If you're unsure what
pin the on-board LED is connected to on your Arduino model, check
the documentation at http://arduino.cc
This example code is in the public domain.
modified 8 May 2014
by Scott Fitzgerald
*/

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(13, HIGH);      // turn the LED on (HIGH is the voltage level)
    delay(1000);                // wait for a second
    digitalWrite(13, LOW);       // turn the LED off by making the voltage LOW
    delay(1000);                // wait for a second
}
```

Para iniciar o entendimento do código, vamos primeiramente verificar o que são comando e o que são comentários, lembre-se que para fazer um comentário quer irá se desenvolver por mais de 1 linha, devemos usar os caracteres /* para começar um comentário e os caracteres */ para finalizar os comentários que foram feitos anteriormente. E para fazer um comentário em 1 linha apenas, devemos podemos utilizar //. Outra maneira simples de verificar isso é pela cor das letras. Por padrão os comentários são em cinza.

Separando o que é comentário do que é comando, podemos verificar que o código do BLINK está escrito em apenas 9 linhas. Vamos agora te acompanhar na leitura e entendimento do código BLINK linha a linha explicando cada linha do código, para que você possa já treinar e fixar alguns dos comandos básicos antes de iniciar seu primeiro projeto.

```
void setup() {  
    // initialize digital pin 13 as an output.  
    pinMode(13, OUTPUT);  
}
```

void setup() : Declaração que irá começar o Setup do programa. Sempre aberto com uma “{“ e fechada, no fim da declaração, por uma “}”.

// initialize the digital pin as an output. : Comentário dizendo que o pino digital será inicializado como uma saída.

pinMode(13, OUTPUT); : Escolha do modo do pino, se é entrada (INPUT) ou saída (OUTPUT).

Como neste caso queremos acender um led, a corrente elétrica irá sair do pino e não entrar. Logo, setamos o ledPin (que tinha o valor 13, por causa do pino digital 13) como saída.

Por fim, neste programa, iremos analisar a estrutura Loop:

```
void loop() {  
    digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)  
    delay(1000); // wait for a second  
    digitalWrite(13, LOW); // turn the LED off by making the voltage LOW  
    delay(1000); // wait for a second  
}
```

void loop() : De modo análogo ao setup, com o comando ao lado dizemos que irá começar o loop do programa, ou seja, o programa principal que ficará rodando por tempo indeterminado. Também é aberto com uma “{“ e fechado com uma “}”.

digitalWrite(13, HIGH); : Escrita digital. Por tratar-se de um pino digital, ou você terá nível lógico 1 ou terá nível lógico 0, no caso de um led, ou teremos led acesso (1) ou teremos led apagado (0). O comando então liga o led, ou seja, envia 1 para o pino 13.

delay(1000); : Delay (atraso/tempo) é mais uma função pronta de seu arduino. O número que for inserido entre os parêntesis será o valor, em milissegundos, que o Arduino irá esperar para seguir para a próxima instrução. No caso, temos um delay de 1000 milissegundos, ou seja, uma espera de 1 segundo para executar a próxima instrução.

```
digitalWrite(13, LOW);
```

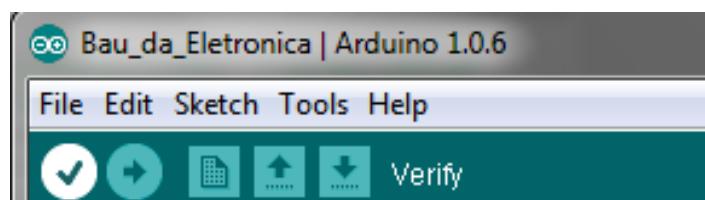
`delay(1000);` : Estes dois comandos são similares aos dois vistos acima, com a única diferença que a escrita digital escreverá um 0 no pino do led, ou seja, um nível lógico baixo: o led apagará e o Arduino espera 1 segundo para fazer a próxima instrução que, no caso, volta a ser o `digitalWrite(13, HIGH);`.

Agora que entendemos o código, podemos compilar o mesmo e fazer o upload para nossa placa Arduino para vê-lo em execução.

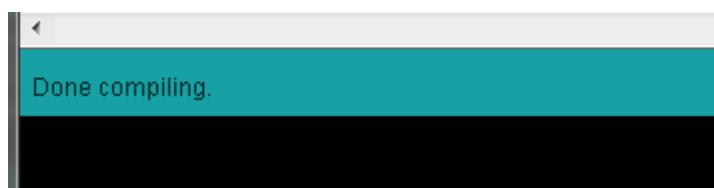
Conecte sua placa Arduino ao seu computador através de uma porta USB.

Agora compile o programa clicando no botão Verify (Verificar) do ambiente de desenvolvimento, para ver se não existe nenhum erro de código.

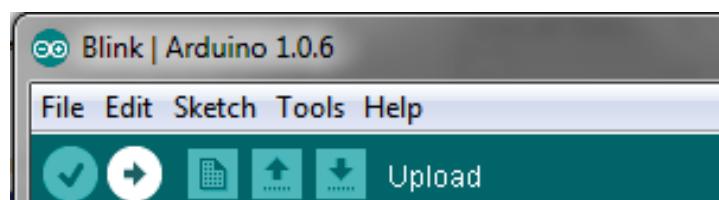
O botão é o seguinte:



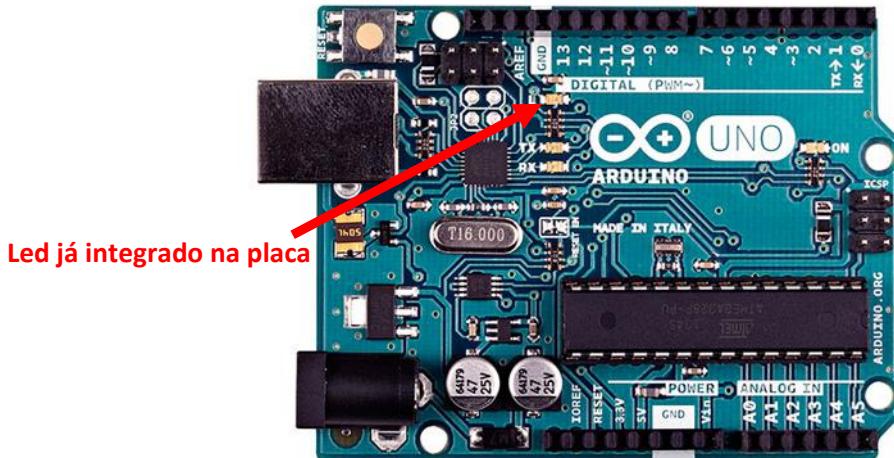
Se na barra inferior aparecer a mensagem: Done Compiling (Compilação Efetuada), o programa está pronto para ser enviado ao Arduino.



Para tanto, basta clicar no botão Upload que é o seguinte:



Espere então o upload ser completado e pronto. Você deverá ver o led da placa piscando com intervalos de 1 segundo.



Vista toda esta explicação, agora sim podemos começar a estudar a primeira experiência deste material.

18 – Projetos

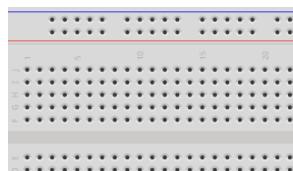
Vamos finalmente aprender exatamente como o hardware e o software de cada projeto funcionam. O aprendizado será logicamente partindo dos projetos mais simples com menor nível de dificuldade para os projetos mais complexos. Dessa forma a cada projeto você se aprofundará um pouco mais em eletrônica e codificação.

18.1 – Projeto 1 (Led Piscante)

Neste primeiro projeto iremos basicamente fazer um LED piscar mas dessa vez, no entanto, você conectará um LED a um dos pinos digitais em vez de utilizar o LED 13, soldado na placa.

18.1.1 – Componentes necessários

Protopboard



LED



Resistor 220R



Jumpers

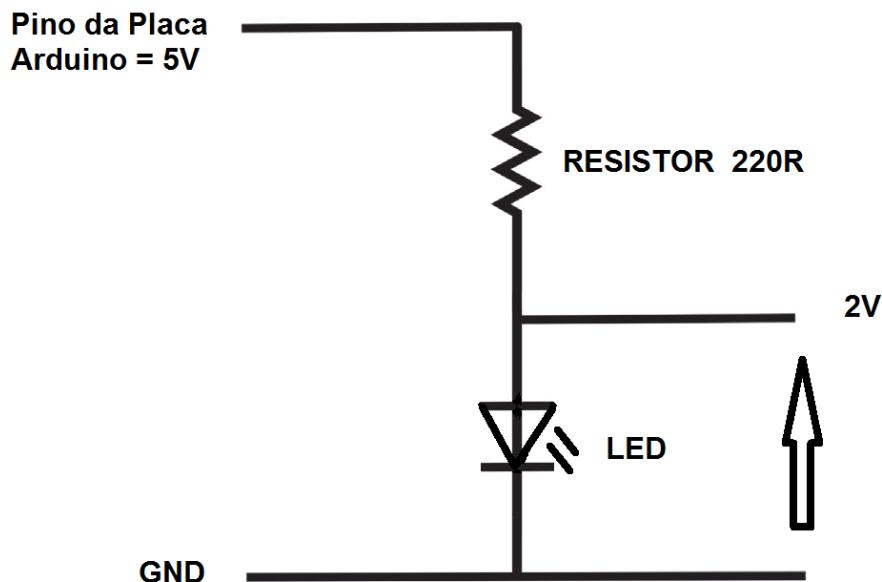


18.1.2 – Montando o circuito

Antes de iniciarmos a montagem de nosso circuito, vamos aprender um conceito muito importante que é o do divisor de tensão, este conceito deve ficar claro para você pois ele será aplicado em quase todos os projetos deste manual.

O circuito divisor de tensão (ou também conhecido como divisor de potencial) na maioria dos casos é montado com um ou mais resistores, mas pode também ser um potenciômetro. Utilizando dois resistores e passando a voltagem por apenas um deles, você pode reduzir a voltagem que entra no circuito.

Em nosso caso, vamos analisar um divisor de tensão padrão, utilizando um resistor e um LED para ver como isso funciona. Abaixo encontra-se uma ilustração de como funciona um divisor de tensão.



Como já mostrado anteriormente, para funcionar, um LED necessita de 2V e consome uma corrente de 20mA, então nossa alimentação é de 5V e 2V ficam sobre o LED, logo sobre nosso resistor devem ficar 3V. Outro ponto importante a ser lembrado é que o resistor está em série com o LED, logo a corrente que passa pelo resistor é a mesma corrente que passará pelo LED.

Agora para descobrirmos o valor do resistor que devemos utilizar para atender a nossa necessidade, basta aplicarmos a formula a seguir:

$$Resistor = \frac{\text{Tensão no Resistor (V)}}{\text{Corrente no resistor (I)}}$$

$$Resistor = \frac{3V (\text{tensão sobre o resistor})}{0,02 (\text{corrente do LED} = 20mA \text{ ou } 0,02A)}$$

$$Resistor = 150R$$

Em nossos cálculos encontramos que o resistor deve ser de 150R para que o LED opere com sua potência máxima. Mas evitar que tivéssemos muitos valores diferentes, um para cada projeto podendo ocasionar dúvidas e possíveis confusões, utilizamos apenas três valores, e para nosso projetinho vamos utilizar o valor mais próximo que no caso é o de 220R. Essa alteração no valor do resistor não mudará o funcionamento de nosso projeto, isso apenas fará com que o LED acenda um pouco mais fraco (quase imperceptível) já que a corrente do circuito será menor, pois como já foi dito, o resistor oferece resistência à passagem da corrente elétrica, e se aumentamos o valor do resistor, menos corrente circulará pelo circuito.

Vejamos então quanto de corrente circulará pelo circuito. Para isso usamos essa outra fórmula:

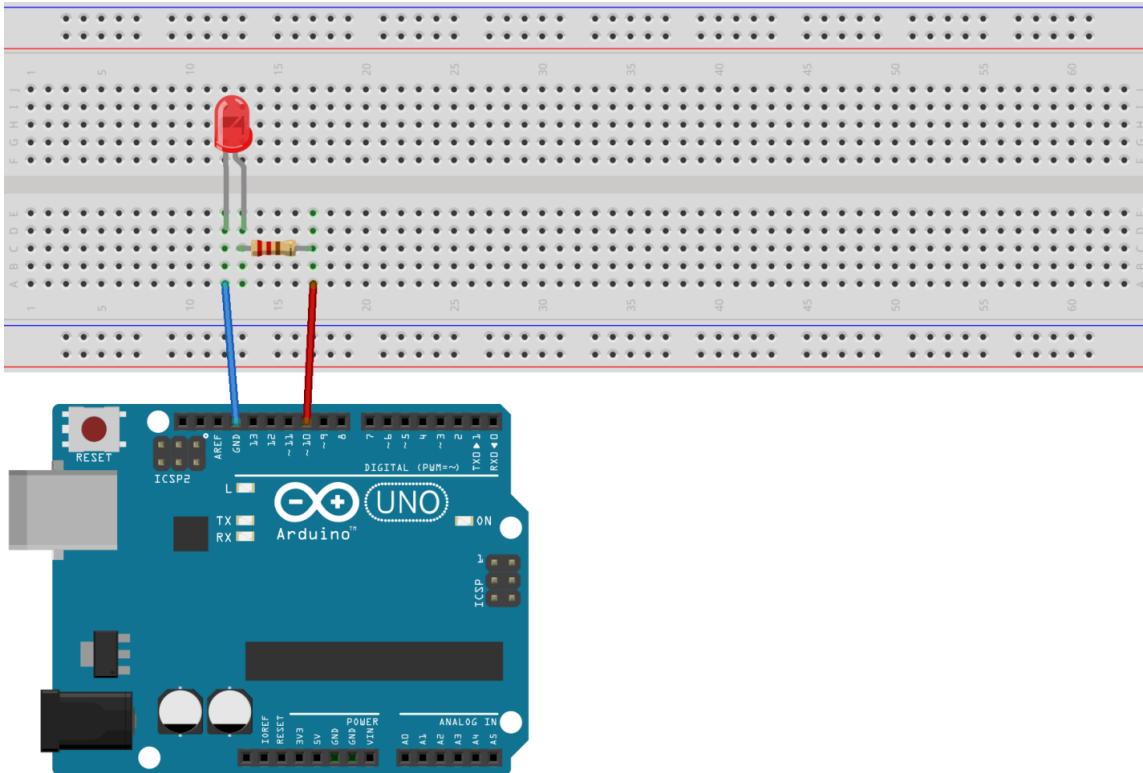
$$\text{Corrente no LED} = \frac{\text{Tensão no Resistor (V)}}{\text{Valor do Resistor (R)}}$$

$$\text{Corrente no LED} = \frac{3V (\text{tensão sobre o resistor})}{220 (\text{Valor do resistor})}$$

$$\text{Corrente no LED} = 0,01363 \text{ ou } 13,63mA$$

Essa corrente de 13,63 mA é suficiente para que nosso LED se acenda perfeitamente. Agora que você já sabe o porque da necessidade do resistor para ligar o LED, tente criar outros divisores de tensão com os resistores que você possui em seu kit para se familiarizar com esta ferramenta, pois ela será de extrema importância para que você crie seus próprios projetos.

Vamos agora à montagem do nosso circuito, primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue seu protoboard, o LED, o resistor e os fios, e conecte tudo como mostra a figura abaixo.



Importante:

Não importa se você utiliza fios de cores diferentes ou furos diferentes no protoboard, desde que os componentes e os fios estejam conectados na mesma ordem da figura. Tenha cuidado ao inserir os componentes na protoboard. Caso seu protoboard seja novo, a superfície dos furos ainda estará rígida. A não inserção cuidadosa dos componentes pode resultar em danos. Certifique-se de que seu LED esteja conectado corretamente, com o terminal (ou perna) mais longo conectado ao pino digital 10. Lembre-se sempre que o terminal longo é o anodo do LED, e deve sempre ir para a alimentação de +5 V (nesse caso, saindo do pino digital 10); o terminal curto é o catodo e deve ir para o terra (GND).

Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.1.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaeletronica.com.br
PROJETO 1 - LED Piscante
=====*/
int ledPin = 10;           //LED conectado ao pino 10

void setup()
{
    pinMode(ledPin, OUTPUT); //Pino 10 do arduino como saída
}
void loop()
{
    digitalWrite(ledPin, HIGH); //Liga o pino 10 e acende o LED
    delay(1000);              //Aguarda 1000ms (ou 1 segundo)
    digitalWrite(ledPin, LOW); //Desliga o pino 10 e acende o LED
    delay(1000);              //Aguarda 1000ms (ou 1 segundo)
}
```

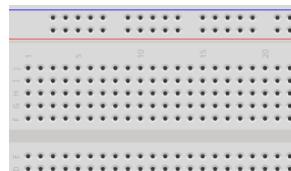
Pressione o botão Verify/Compile no topo do IDE para certificar-se de que não há erros em seu código. Se não houver erros, clique no botão Upload para fazer o upload do código na sua placa Arduino. Caso tudo tenha sido feito corretamente, agora você deverá ver o LED no protoboard, acendendo e apagando em intervalos de um segundo.

18.2 – Projeto 2 (Acendendo o LED com um botão)

Neste projeto iremos acender um LED através do comando por um botão, ao pressionar o botão o LED acende e se soltar o botão o LED se apaga.

18.2.1 – Componentes necessários

Protopboard



LED



Resistor 220R



Resistor 10K



Jumpers

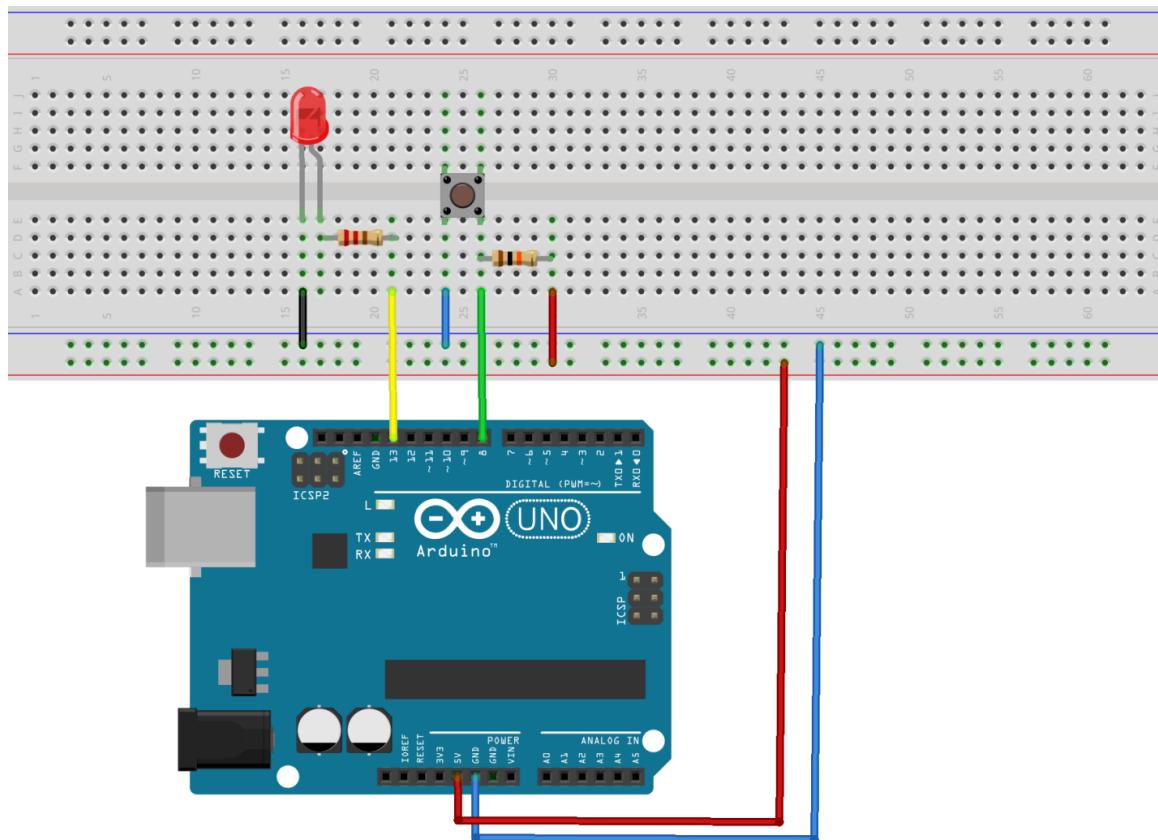


Chave Táctil



18.2.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.2.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*=====
 Baú da Eletrônica Componentes Eletrônicos
 www.baudaeletronica.com.br
 PROJETO 2 - Acendendo o LED com um botão
=====*/
 
int ledPin = 13;           //LED conectado ao pino 13
int botao = 8;             //Botão conectado ao pino 8
int estado_do_botao = 0;   //Variável para leitura do estado do Botão

void setup()
{
    pinMode(ledPin, OUTPUT);    //Pino 10 do arduino como saída
    pinMode(botao, INPUT);     //Pino com botão será entrada
}
void loop()
{
    estado_do_botao = digitalRead(botao); //Armazena o Estado do botão.
                                         //Se sim grava LOW (0) na variável
                                         //Se não grava HIGH (1) na variável
    if (estado_do_botao == LOW)    //Se botão estiver pressionado (LOW)
    {
        digitalWrite(ledPin, HIGH); //Acende o led conectado ao pino 13
    }
    else                         //se não estiver pressionado
    {
        digitalWrite(ledPin, LOW); //Apaga o led conectado ao pino 13
    }
}
```

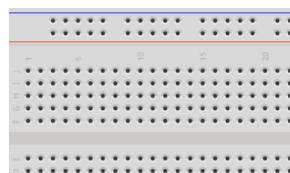
Agora faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora se você pressionar o botão o LED no protoboard deverá acender, e se soltar o botão o LED deverá se apagar.

18.3 – Projeto 3 (Semáforo)

Neste projeto faremos um semáforo que irá do verde ao vermelho, passando pelo amarelo, e que retornará depois de um intervalo de tempo igualzinho aos que vemos pelas ruas. Este projeto poderia ser utilizado para criar um conjunto funcional de semáforos para uma maquete ou para uma pequena cidade de brinquedo. Primeiro, entretanto, crie o projeto conforme mostrado a seguir, e faça as alterações apenas depois de saber como tudo funciona.

18.3.1 – Componentes necessários

Protopboard



LED Vermelho



LED Amarelo



LED Verde



3 Resistores 220R

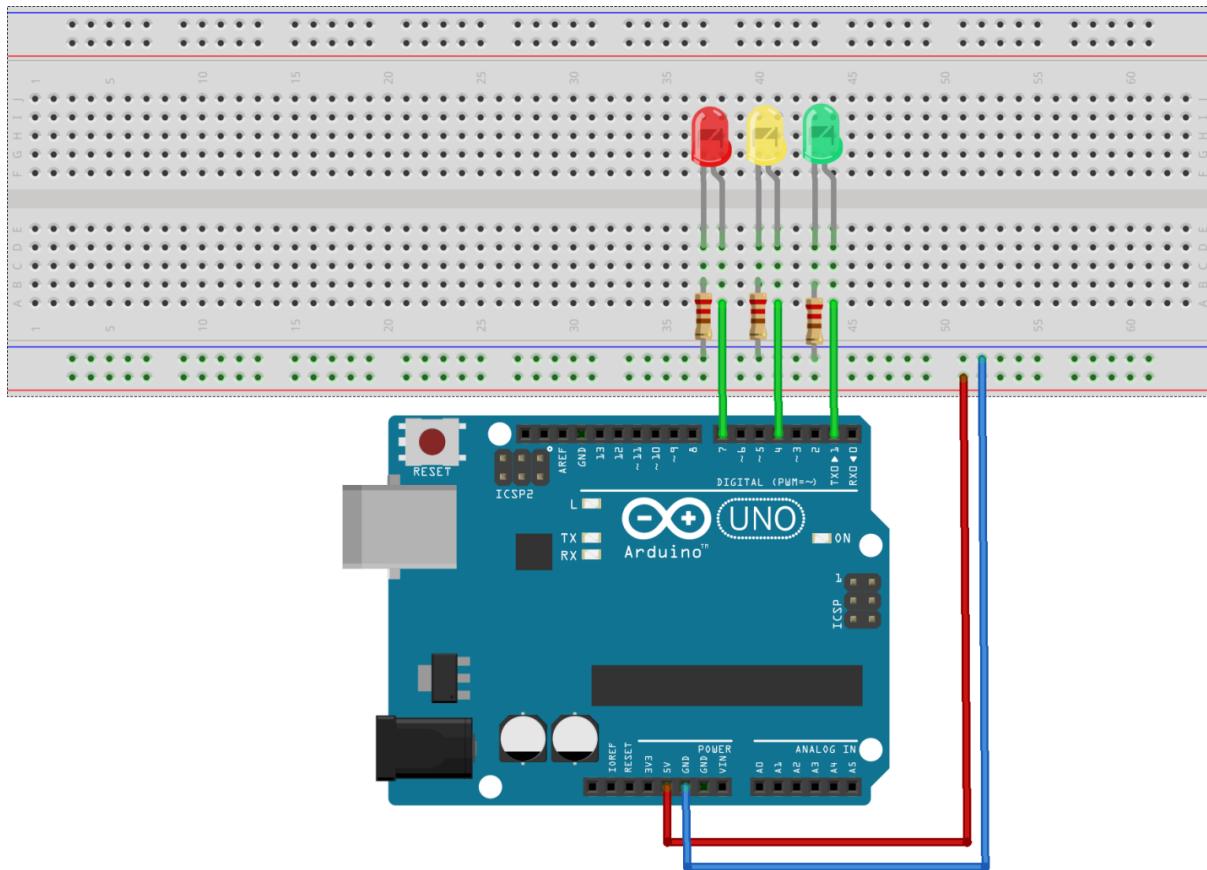


Jumpers



18.3.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.3.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
 Baú da Eletrônica Componentes Eletrônicos
 www.baudaelectronica.com.br
 PROJETO 3 - Semáforo
=====*/
int Led_Vermelho = 7;           //LED Vermelho conectado ao pino 7
int Led_Amarelo = 4;            //LED Amarelo conectado ao pino 4
int Led_Verde = 1;              //LED Verde conectado ao pino 1

void setup()
{
    pinMode(Led_Vermelho, OUTPUT); //Pino 7 do arduino como saída
    pinMode(Led_Amarelo, OUTPUT); //Pino 4 do arduino como saída
    pinMode(Led_Verde, OUTPUT);   //Pino 1 do arduino como saída
}
void loop()
{
    digitalWrite(Led_Vermelho, HIGH); //Acende o led vermelho
    delay(4000);                  //permanece 4s com o led vermelho aceso
    digitalWrite(Led_Vermelho, LOW); //Apaga o led vermelho

    digitalWrite(Led_Verde, HIGH);  //Acende imediatamente o led verde
    delay(4000);                  //permanece 4s com o led verde aceso
    digitalWrite(Led_Verde, LOW);  //Apaga o led verde

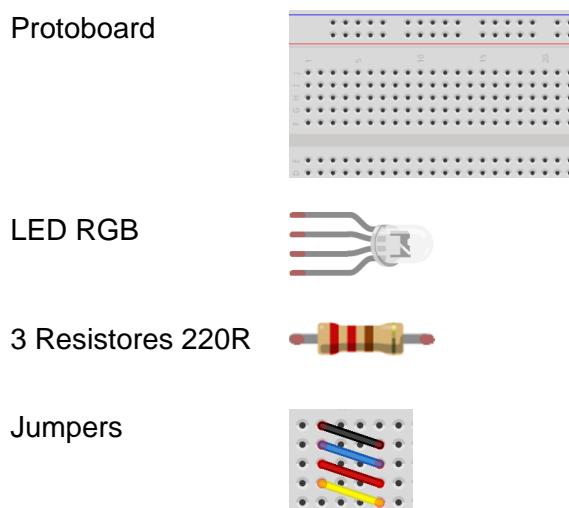
    digitalWrite(Led_Amarelo, HIGH); //Acende imediatamente o led amarelo
    delay(2000);                  //permanece 2s com o led amarelo aceso
    digitalWrite(Led_Amarelo, LOW); //Apaga o led amarelo
}
```

Faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora se você verá o seu semáforo em funcionamento, ficando 4 segundos fechado no vermelho, 4 segundos aberto no verde e na transição de aberto para fechado, passará pelo amarelo sinalizando atenção durante 2 segundos.

18.4 – Projeto 4 (Controlando o LED RGB)

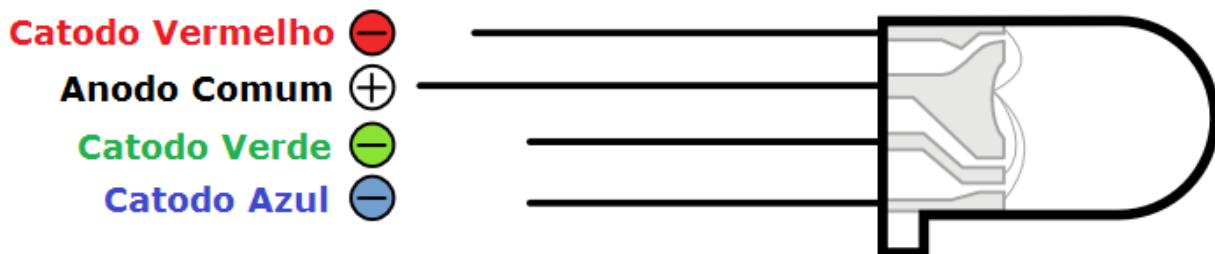
Neste projeto vamos controlar um led RGB através dos pinos PWM do Arduino. Como já mostrado no capítulo 1.4, o led RGB é um tipo de led, com quatro terminais, capaz de emitir diversos tipos de cores diferentes. Suas cores básicas são vermelho, verde e azul, e a mistura dessas cores pode formar diversas outras cores. Então vamos fazer com que o LED RGB emita cada uma de suas cores básicas e cores resultantes da mistura das cores básicas.

18.4.1 – Componentes necessários

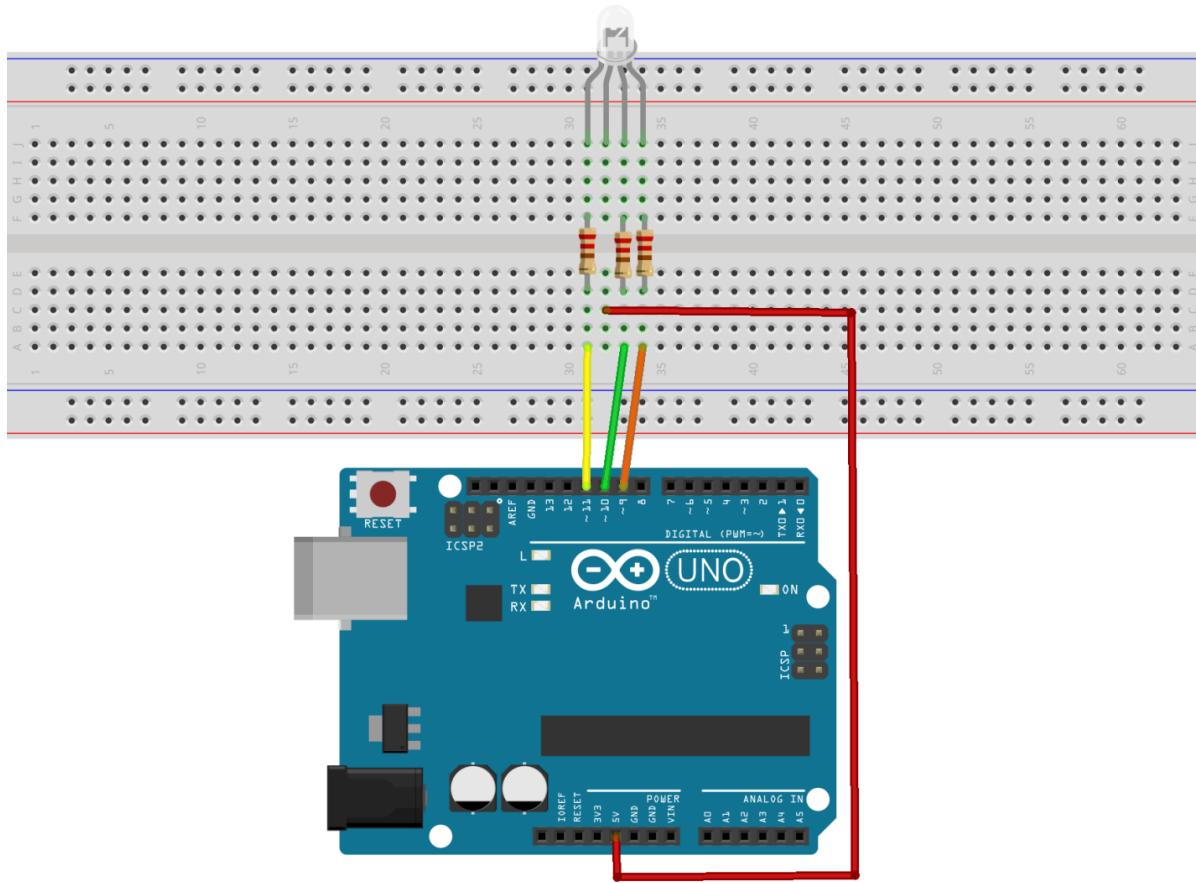


18.4.2 – Montando o circuito

Primeiro vamos entender como ligar o LED RGB. Note que ele possui quatro terminais e que cada terminal tem um tamanho diferente, isso para que possamos identificar o terminal referente a cada cor do LED conforme mostrado a seguir:



Agora, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB.
Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.4.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaeletronica.com.br
PROJETO 4 - Semáforo
=====*/
int Vermelho = 9;      //Pino Vermelho conectado ao pino 11 do Arduino
int Verde = 10;        //Pino Azul conectado ao pino 10 do Arduino
int Azul = 11;          //Pino Verde conectado ao pino 9 do Arduino
int tempo = 5;

void setup()
{
    pinMode(Vermelho, OUTPUT); //Pino 11 do arduino como saída
    pinMode(Azul, OUTPUT);     //Pino 10 do arduino como saída
    pinMode(Verde, OUTPUT);   //Pino 9 do arduino como saída
}
void loop()
{
    analogWrite(Vermelho, 255); //Apaga o led vermelho
    analogWrite(Azul, 255);     //Apaga o led azul
    analogWrite(Verde, 255);    //Apaga o led verde

    for(int i=0; i<255; i++)
    {
        analogWrite(Vermelho, i); //Acende os três LEDS e apaga o vermelho
        analogWrite(Azul, 0);     //lentamente mostrando a mistura do Azul
        analogWrite(Verde, 0);    //com o verde.
        delay(tempo);
    }
    for(int i=0; i<255; i++)
    {
        analogWrite(Vermelho, 255); //Agora com o vermelho ligado, apaga o azul
        analogWrite(Azul, i);       //lentamente mostrando apenas o verde
        analogWrite(Verde, 0);     //aceso.
        delay(tempo);
    }
    for(int i=255; i>0; i--)
    {
        analogWrite(Vermelho, i); //Com apenas o verde aceso, o vermelho começa a
        analogWrite(Azul, 255);   //acender lentamente ficando o verde e o
        analogWrite(Verde, 0);    //vermelho acesos
        delay(tempo);
    }
    for(int i=0; i<255; i++)
    {
        analogWrite(Vermelho, 0); //Com o verde e o vermelho acesos, o verde
        analogWrite(Azul, 255);   //começa a se apagar lentamente ficando então
        analogWrite(Verde, i);    //apenas o vermelho aceso
        delay(tempo);
    }
    for(int i=255; i>0; i--)
    {
        analogWrite(Vermelho, 0); //Com apenas o vermelho aceso, agora o azul
        analogWrite(Azul, i);    //começa a se acender lentamente, ficando
        analogWrite(Verde, 255); //o azul e o vermelho acesos
        delay(tempo);
    }
}

/*CONTINUA NA PRÓXIMA PÁGINA*/
```

```
for(int i=0; i<255; i++)
{
    analogWrite(Vermelho, i); //Neste ponto com azul e vermelho acesos, o
    analogWrite(Azul, 0);     //vermelho se apaga lentamente e fica apenas
    analogWrite(Verde, 255); //o azul aceso
    delay(tempo);
}
    for(int i=255; i>0; i--)
{
    analogWrite(Vermelho, 255); //Agora com apenas o azul aceso, o verde se
    analogWrite(Azul, 0);       //acende lentamente ficando novamente o verde
    analogWrite(Verde, i);      //e o azul acesos
    delay(tempo);
}
    for(int i=255; i>0; i--)
{
    analogWrite(Vermelho, i); //E por fim o vermelho se acende lentamente
    analogWrite(Azul, 0);     //juntamente com o azul e o verde voltando
    analogWrite(Verde, 0);    //acender branco novamente iniciando novamente
    delay(tempo);            //o ciclo
}
}
```

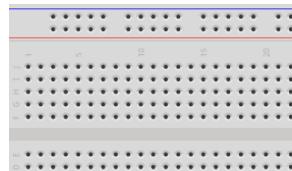
Faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora se você verá o led RGB se acendendo primeiramente branco e alternando sua cor lentamente entre o branco inicial, verde, vermelho, azul e combinações destas cores.

18.5 – Projeto 5 (Detector de objetos)

Agora faremos um detector de objetos ou sensor de presença que ao detectar a presença de algum objeto próximo ao leitor ótico informa ao sistema que um objeto foi detectado. Este projeto poderia ser utilizado para inúmeras aplicações como por exemplo em um alarme que dispararia se alguém se aproximasse, em uma esteira de linha de produção para contar produtos e em pequenos robôs que detectam faixas no chão. Primeiro, entretanto, crie o projeto conforme mostrado a seguir, e faça as alterações apenas depois de saber como tudo funciona.

18.5.1 – Componentes necessários

Protoboard



Sensor óptico TCRT5000



Resistor 10K



Resistor 300R

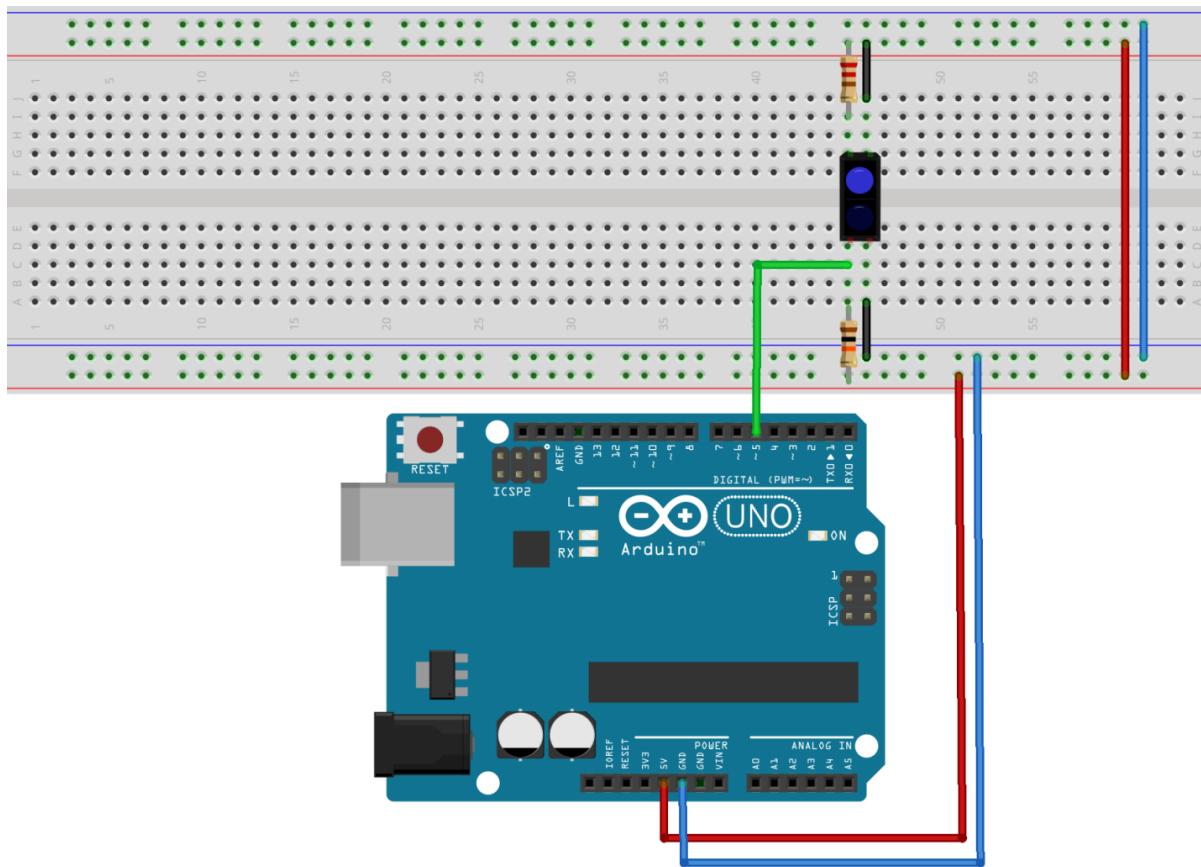


Jumpers



18.5.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.5.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaelectronica.com.br
PROJETO 5 - Detector de Objetos
=====*/
int Objeto = 0;           //Variável para armazenar dados do sensor
int Sensor = 5;          //Sensor conectado ao pino 5

void setup()
{
    pinMode(Sensor, INPUT); //Pino 7 do arduino como saída
    Serial.begin(9600); //Indica ao Arduino que vamos enviar e receber dados
                        //com o mundo externo através da porta USB
}
void loop()
{
    Objeto = digitalRead(Sensor); //Verifica sinal do sensor

    if (Objeto == 0)           //Se o valor do sensor for 0 = Objeto detectado
    {
        Serial.println("Nenhum objeto presente"); //Escreve mensagem no
monitor
    }

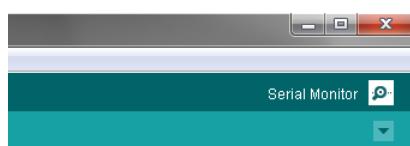
    else                      //Se o valor do sensor for 1 = Objeto nenhum objeto presente
    {
        Serial.println("Objeto detectado"); //Escreve mensagem
    }
}
```

Nesta etapa, para testar seu projeto, você deve utilizar uma nova ferramenta para verificar o funcionamento do seu projeto, o monitor serial.

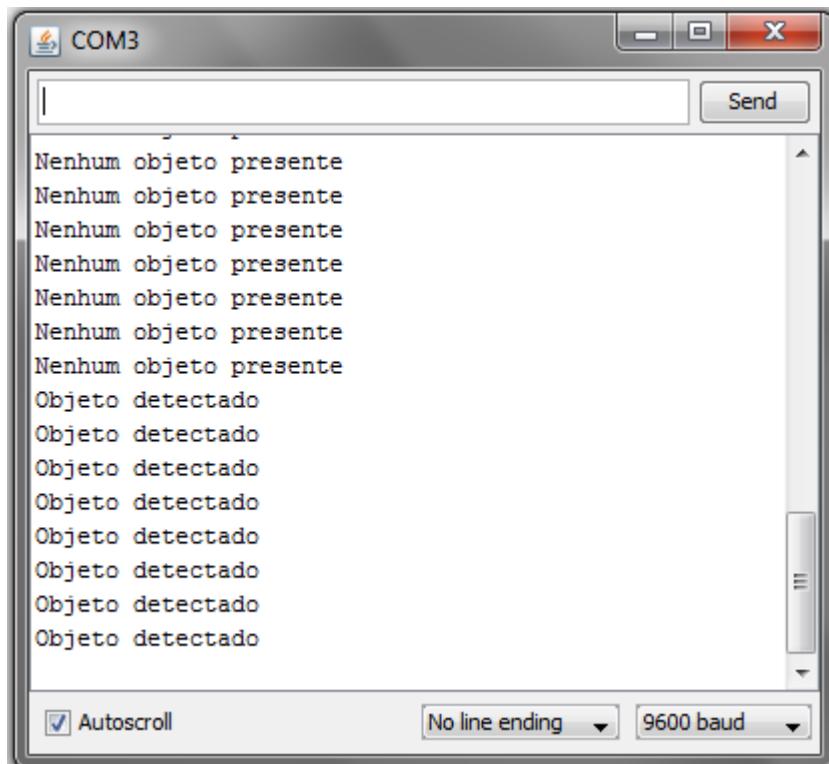
Vamos primeiramente entender o que é, e como funciona o monitor serial para então darmos sequência em nossos testes.

O monitor serial é uma ferramenta muito útil, com ele você pode exibir na tela do seu PC os dados seriais enviados de seu Arduino (USB ou placa serial). Com ele você pode ainda enviar dados de volta ao Arduino.

Clique no botão Serial Monitor no canto superior direito do software IDE do Arduino conforme imagem a seguir.



Ao clicar abrirá uma janela igual a janela mostrada na figura a seguir.



A porta COM não é necessariamente 3, como está no topo da imagem acima. Cada computador tem sua numeração de portas. Veja que no canto inferior direito temos selecionado 9600 baud. Isto tem de ser selecionado conforme a configuração do parâmetro `Serial.begin` do `setup` de seu programa. Esse valor nada mais é que a taxa de transmissão por segundo em que alterações de estado ou bits (dados) são enviados de/para a placa. A configuração padrão é 9.600 baud.

No topo, há uma caixa de texto em branco, para que você digite o texto a ser enviado de volta para o Arduino, e um botão `Send`, para enviar o texto. Note que o monitor serial não pode receber nenhum dado serial, a menos que você tenha preparado o código em seu sketch para que isso ocorra. Da mesma forma, o Arduino não receberá nenhum dado, a menos que você o tenha codificado para tanto. Por fim, a área central da janela é o local em que seus dados seriais serão exibidos.

Para iniciar o monitor serial, pressione o botão `Serial Monitor`. Para interrompê-lo, pressione o botão `Stop`. Em um Mac ou Linux, a placa do Arduino reiniciará sozinha (reexecutando o código desde o início), quando você clicar no botão `Serial Monitor`.

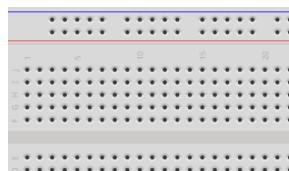
Agora que você já sabe como utilizar o monitor serial, faça o upload do código em sua placa Arduino, abra o serial monitor e aproxime objetos do sensor até que o objeto seja detectado. Para verificar se foi ou não detectado, veja no serial monitor se o status muda de "Nenhum objeto presente" para "Objeto detectado".

18.6 – Projeto 6 (Piano)

Neste projeto, vamos montar um pequeno piano de 4 teclas com indicação visual de qual teclas está sendo pressionada. Cada um dos 4 botões tocará uma nota musical diferente. Para emissão do sinal sonoro do nosso piano usaremos o Buzzer. Um Buzzer conforme já descrito anteriormente, é um pequeno alto-falante que obviamente não consegue tocar músicas, mas consegue emitir sinais sonoros similares aos de alarmes e sirenes. A maioria dos alarmes de pequenos equipamentos eletrônicos são feitos com buzzers. Ele funciona da seguinte maneira: quando alimentado por uma fonte, componentes metálicos internos vibram da frequência da fonte, produzindo assim um som.

18.6.1 – Componentes necessários

Protoboard



LED Vermelho



LED Amarelo



LED Verde



LED Azul



4 Resistores 10K



4 Resistores 220R



Buzzer



Jumpers

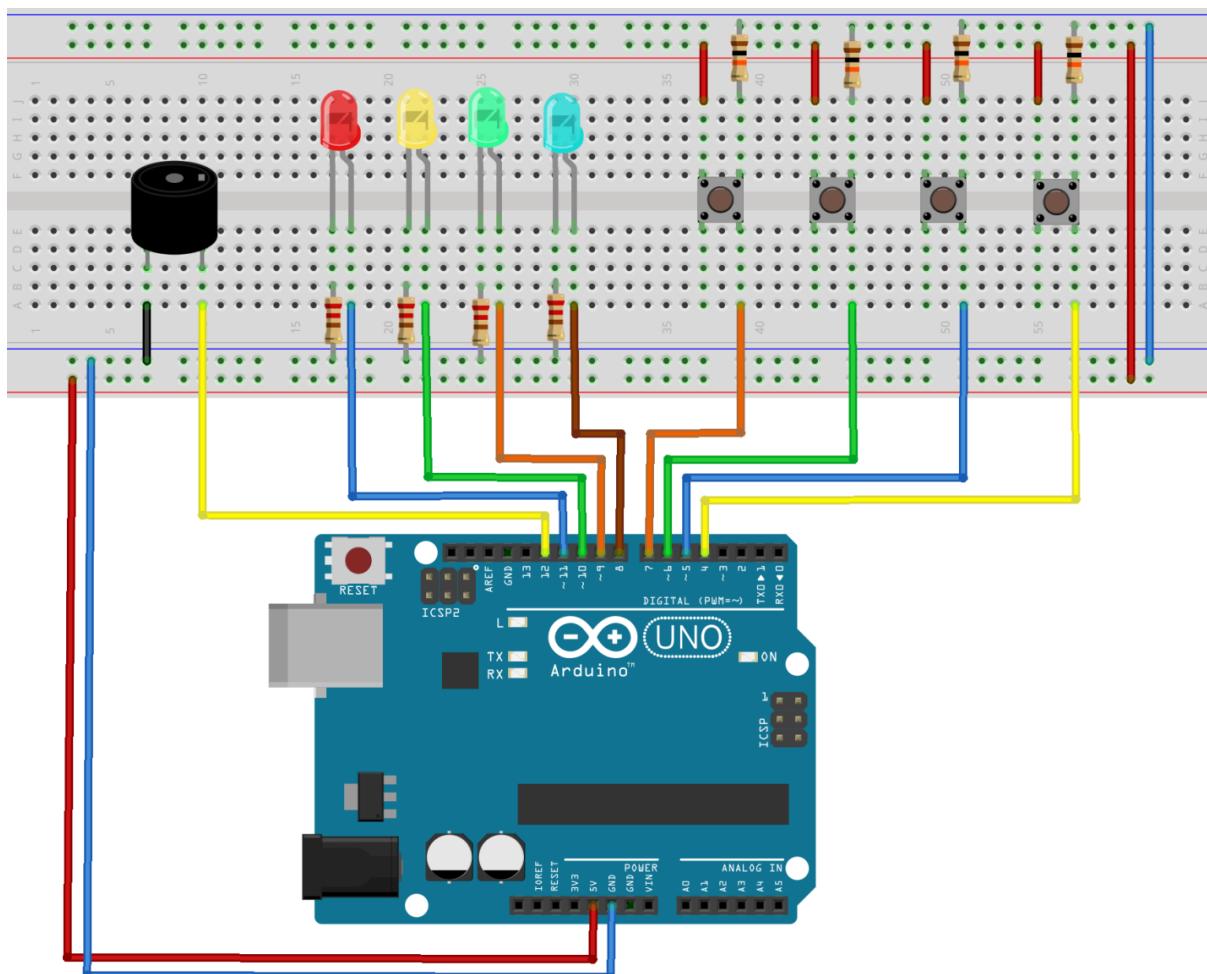


18.6.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.

**Importante:**

O Buzzer tem polaridade, o que significa que só pode ser ligado de uma maneira. Se você retirar o adesivo superior do buzzer poderá ver um sinal de positivo (+). Este sinal mostra onde está o pino positivo do componente. Sempre ligue este pino a uma saída digital do Arduino e o outro ao GND.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.6.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*=====
    Baú da Eletrônica Componentes Eletrônicos
    www.baudaelectronica.com.br
    PROJETO 6 - Piano
=====*/
 
int Led_Vermelho = 11;           //LED Vermelho conectado ao pino 11
int Led_Amarelo = 10;            //LED Amarelo conectado ao pino 10
int Led_Verde = 9;               //LED Verde conectado ao pino 9
int Led_Azul = 8;                //LED Azul conectado ao pino 8
int Botao_1 = 7;                 //Botão 1 conectado ao pino 7
int Botao_2 = 6;                 //Botão 1 conectado ao pino 6
int Botao_3 = 5;                 //Botão 1 conectado ao pino 5
int Botao_4 = 4;                 //Botão 1 conectado ao pino 4
int Estado_Botao_1 = 0;          //Variável para armazenar estado do botão 1
int Estado_Botao_2 = 0;          //Variável para armazenar estado do botão 2
int Estado_Botao_3 = 0;          //Variável para armazenar estado do botão 3
int Estado_Botao_4 = 0;          //Variável para armazenar estado do botão 4
int Nota = 0;                    //Variável para armazenar a nota musical
int Buzzer = 12;                //Buzzer conectado ao pino 12

void setup()
{
    pinMode(Led_Vermelho, OUTPUT); //Pino 7 do arduino como saída
    pinMode(Led_Amarelo, OUTPUT); //Pino 6 do arduino como saída
    pinMode(Led_Verde, OUTPUT); //Pino 5 do arduino como saída
    pinMode(Led_Azul, OUTPUT); //Pino 4 do arduino como saída
    pinMode(Botao_1, INPUT); //Pino 3 do arduino como entrada
    pinMode(Botao_2, INPUT); //Pino 2 do arduino como entrada
    pinMode(Botao_3, INPUT); //Pino 1 do arduino como entrada
    pinMode(Botao_4, INPUT); //Pino 0 do arduino como entrada
    pinMode(Buzzer, OUTPUT); //Pino 11 do arduino como saída
}
void loop()
{
    Estado_Botao_1 = digitalRead(Botao_1); //Lê o estado do botão 1
    Estado_Botao_2 = digitalRead(Botao_2); //Lê o estado do botão 2
    Estado_Botao_3 = digitalRead(Botao_3); //Lê o estado do botão 3
    Estado_Botao_4 = digitalRead(Botao_4); //Lê o estado do botão 4

    if(Estado_Botao_1 && !Estado_Botao_2 && !Estado_Botao_3 && !Estado_Botao_4)
    { Nota = 80;
        digitalWrite(Led_Vermelho, HIGH); } //Apaga o led vermelho

    if(!Estado_Botao_1 && Estado_Botao_2 && !Estado_Botao_3 && !Estado_Botao_4)
    { Nota = 160;
        digitalWrite(Led_Amarelo, HIGH); } //Apaga o led vermelho

    if(!Estado_Botao_1 && !Estado_Botao_2 && Estado_Botao_3 && !Estado_Botao_4)
    { Nota = 240;
        digitalWrite(Led_Verde, HIGH); } //Apaga o led vermelho

    if(!Estado_Botao_1 && !Estado_Botao_2 && !Estado_Botao_3 && Estado_Botao_4)
    { Nota = 320;
        digitalWrite(Led_Azul, HIGH); } //Apaga o led vermelho

/*CONTINUA NA PRÓXIMA PÁGINA*/
```

```
/*CONTINUAÇÃO*/  
  
if (Nota > 0)          //LED Vermelho conectado ao pino 7  
{  digitalWrite(Buzzer, HIGH);  
  delayMicroseconds(Nota);  
  digitalWrite(Buzzer, LOW);  
  delayMicroseconds(Nota);  
  Nota = 0;  
  digitalWrite(Led_Vermelho, LOW);  
  digitalWrite(Led_Amarelo, LOW);  
  digitalWrite(Led_Verde, LOW);  
  digitalWrite(Led_Azul, LOW);  
}  
}
```

Faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora se você verá o seu semáforo em funcionamento, ficando 4 segundos fechado no vermelho, 4 segundos aberto no verde e na transição de aberto para fechado, passará pelo amarelo sinalizando atenção durante 2 segundos.



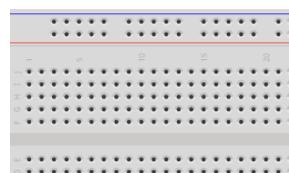
Veja que neste projeto do piano o código ficou bem maior que os códigos que já havíamos visto até aqui, mas não se assuste, analise com calma, tente entender o que fizemos e veja que nada muito complexo aconteceu. Apenas tivemos uma mesma estrutura repetida 4 vezes em função de termos 4 teclas e 4 leds.

18.7 – Projeto 7 (Medindo Temperatura com Sensor NTC)

Este projeto é bem simples porém será de grande valor para seu aprendizado, pois ele será o pontapé inicial para seu aprendizado sobre projetos de aquisição de dados do mundo externo e automação com o Arduino. Neste primeiro passo vamos apenas entender como realizar a leitura de temperatura através do sensor NTC.

18.7.1 – Componentes necessários

Protoboard



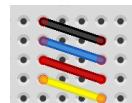
Sensor de Temperatura NTC



Resistor 10K

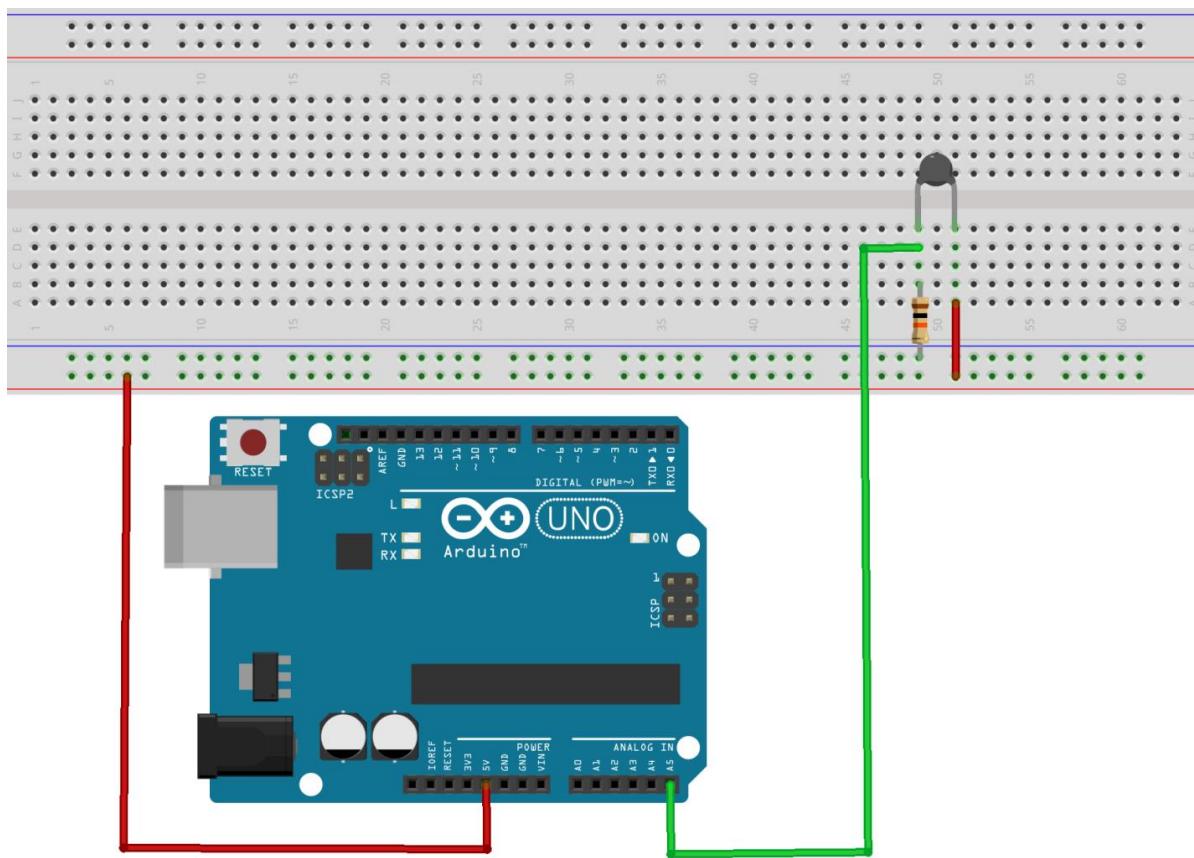


Jumpers



18.7.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.7.3 – Código fonte

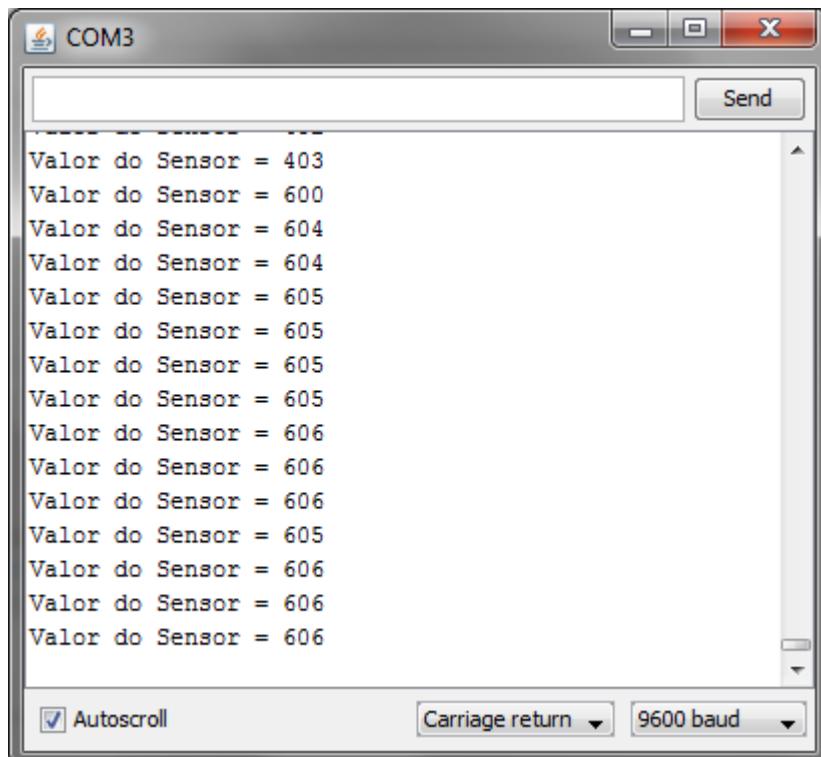
Abra seu IDE do Arduino e digite o código a seguir:

```
/*=====
 Baú da Eletrônica Componentes Eletrônicos
 www.baudaeletronica.com.br
 PROJETO 7 - Medindo Temperatura com Sensor NTC
=====*/
int Pino_NTC = 5;           //Sensor NTC conectado ao Pino 5
int Valor_Sensor = 0;       //Variável para armazenar valor lido do sensor

void setup()
{
    Serial.begin(9600); //Indica ao Arduino que vamos enviar dados
                         // através da porta USB
}
void loop()
{

    Valor_Sensor = analogRead(Pino_NTC); //Lê sinal do sensor
    Serial.print("Valor do Sensor = "); //Escreve mensagem no monitor
    Serial.println(Valor_Sensor);        //Escreve valor do sensor no monitor
    delay(1000);                      //Espera 1 segundo para mostrar novo valor
}
```

Para testar seu projeto você deve fazer o upload do código em sua placa Arduino e abrir o serial monitor. Ao fazer isso, você deverá ver algo similar ao mostrado na imagem abaixo.



É importante ressaltar que, como os componentes eletrônicos não são totalmente iguais e que a temperatura ambiente em cada ponto do mundo é diferente, você não necessariamente vai ler valores como 606. Esta é a temperatura ambiente lida pelo sensor no local onde este material foi desenvolvido. Para fazer um teste com o sensor de temperatura, podemos utilizar um ferro de solda, ou um ferro de passar roupas, ou um secador de cabelo (qualquer coisa que esquente rapidamente) ou também seus dedos, visto que a temperatura do seu corpo é maior do que a temperatura ambiente. Quando aproximamos um ferro de solda ao sensor de temperatura, fazemos leituras diferentes.

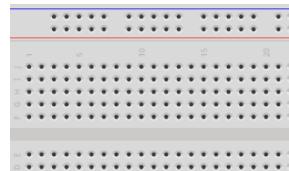
Pronto, agora que você já sabe como realizar a leitura dos valores do sensor, podemos integrar este projeto a mais componentes para realizar algo mais interessante como por exemplo um alarme de temperatura ou até mesmo um sistema para controlar a temperatura de um determinado ambiente.

18.8 – Projeto 8 (Alarme de temperatura)

Nesta etapa vamos integrar o projeto anterior ao Buzzer para criarmos um alarme de temperatura, que vamos programar para emitir um sinal sonoro de alerta quando a temperatura ultrapassar um determinado valor.

18.8.1 – Componentes necessários

Protoboard



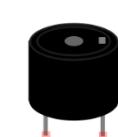
Sensor de Temperatura NTC



Resistor 10K



Buzzer

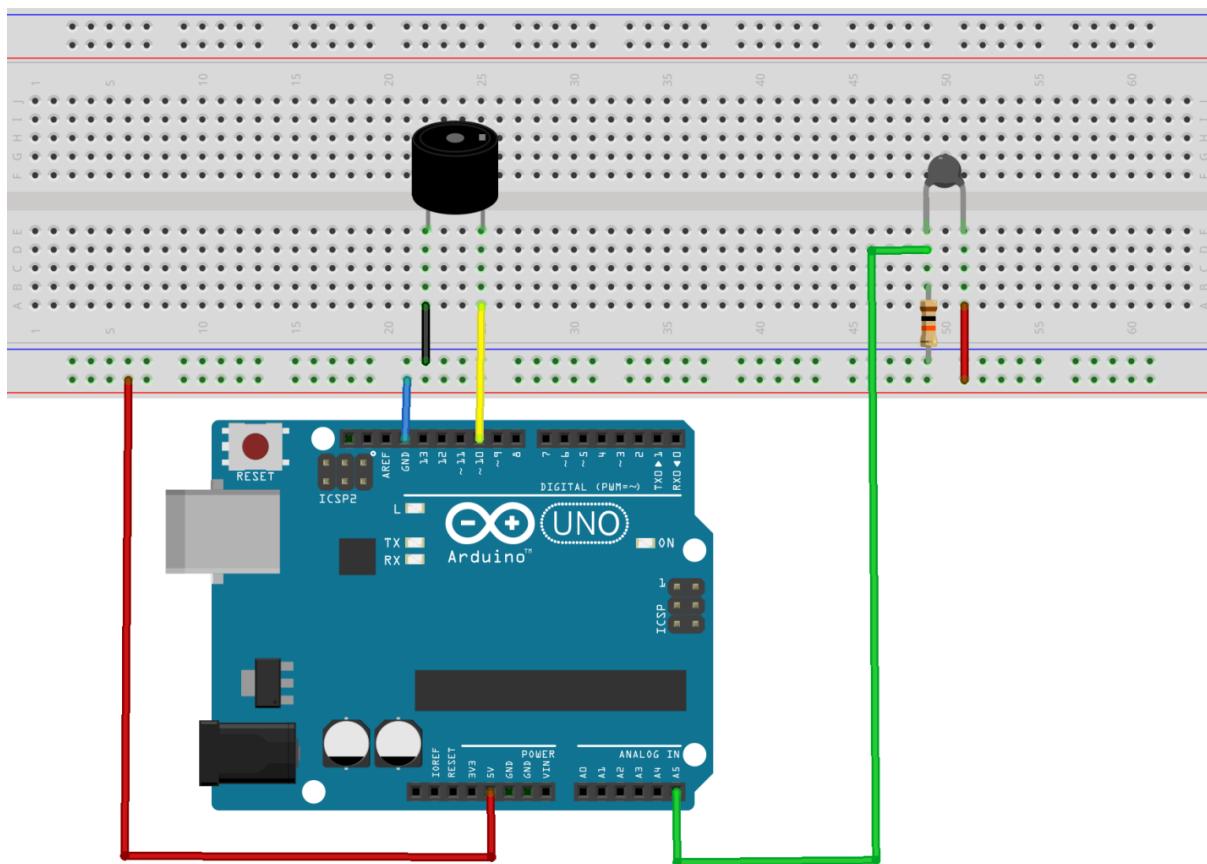


Jumpers



18.8.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.8.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaelectronica.com.br
PROJETO 8 - Alarme de Temperatura
=====*/
int Pino_NTC = 5;           //Sensor NTC conectado ao Pino 5
int Buzzer = 10;            //Sensor NTC conectado ao Pino 10
int Valor_Sensor = 0;       //Variável para armazenar valor lido do sensor

void setup()
{
    pinMode(Buzzer, OUTPUT);      //Pino 10 do arduino como saída
}

void loop()
{
    Valor_Sensor = analogRead(Pino_NTC); //Lê sinal do sensor

    if (Valor_Sensor > 900)        //Verifica se a temperatura ultrapassou o limite
    {
        digitalWrite(Buzzer, HIGH); // Se sim, toca alarme
    }

    else
    {
        digitalWrite(Buzzer, LOW); // Caso contrário desliga o alarme
    }
}
```

Para testar seu projeto você deve fazer o upload do código e simular alterações de temperatura de forma similar à experiência anterior, com um ferro de solda, ferro de passar, secador de cabos ou seus dedos.

Quando a temperatura ultrapassar o valor pré-definido por você, o alarme soará e quando ela baixar o alarme desligará. Este valor pré-definido, você deve encontrar de acordo com seu ambiente e as tolerâncias de seus componentes, para isso você deve aumentar ou diminuir este valor conforme mostrado abaixo.

if (Valor_Sensor > 900)

Quanto mais você aumentar esse valor mais alta terá de ser a temperatura para que o alarme toque, então faça alterações e descarregue o código até encontrar o melhor ajuste.

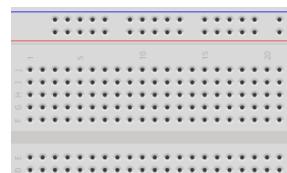
Pronto, agora que já sabe como realizar uma tarefa de acordo com uma variação do sensor de temperatura, você pode construir projetos para diversas outras aplicações, como por exemplo um controlador de temperatura. Para isso basta substituir o buzzer por um ventilador, aí quando a temperatura subir de mais, ao invés de tocar um alarme, um ventilador ligará para baixar a temperatura até que esta se estabilize e o ventilador se desligue e assim sucessivamente.

18.9 – Projeto 9 (Medindo Temperatura com LM35)

Neste projeto vamos aprender a utilizar o LM35 que como já mostrado é um sensor de temperatura como o NTC que já estudamos, mas que é mais preciso e já nos fornece um sinal de medição na escala Celsius o que facilita bastante na criação do código fonte .

18.9.1 – Componentes necessários

Protoboard



Sensor de Temperatura LM35

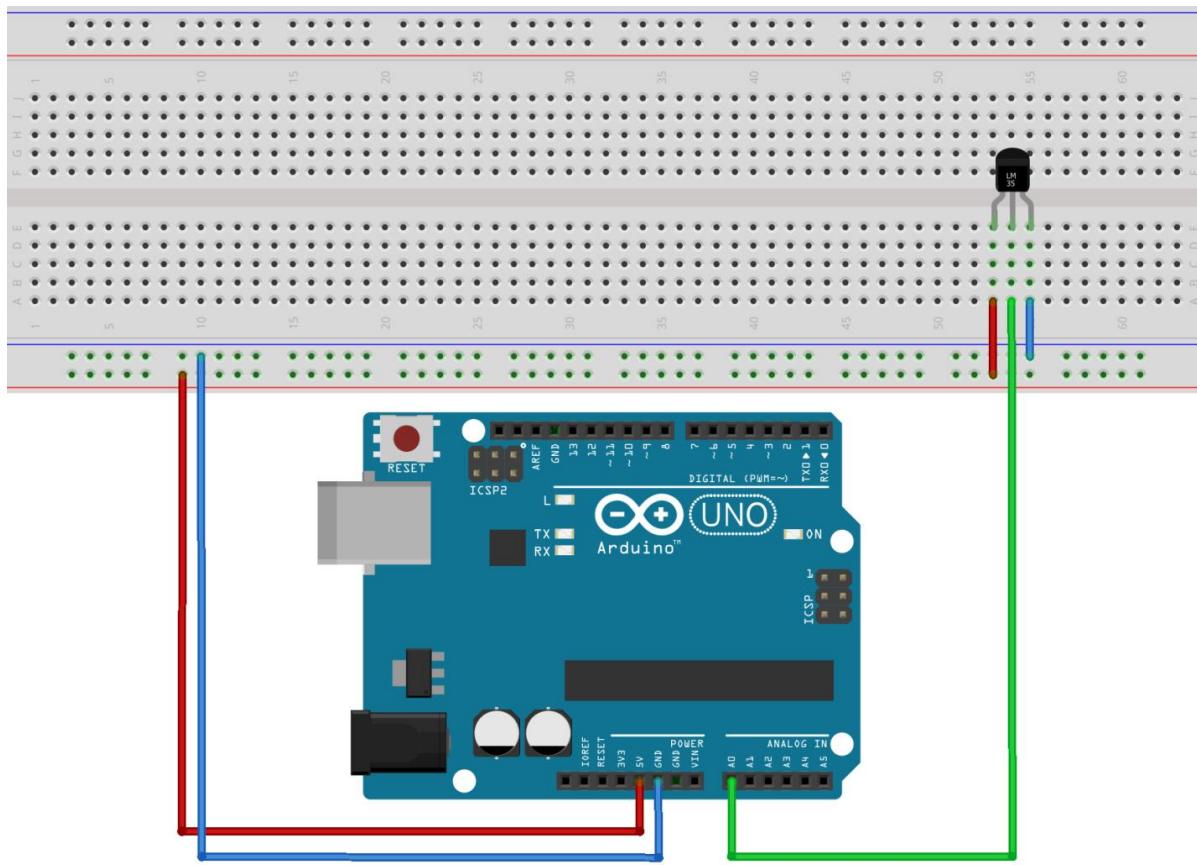


Jumpers



18.9.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Importante:

Tenha bastante atenção à posição correta do sensor LM35, a face achatada deve ficar voltada para frente e a face arredondada deve ficar para trás, a inversão pode ocasionar avarias ao componente.

Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.9.3 – Código fonte

Neste ponto de nossos estudos, veremos um pouco mais de teoria sobre o LM35 e sobre o conversor analógico digital do Arduino para então entendermos como o código fonte foi criado.

O conversor AD do Arduino possui resolução de 10 bits, isso significa que ele possui uma escala de 2^{10} valores, ou seja vai de 0 a 1023. Como nossa placa Arduino opera com 5V e nosso conversor AD possui escala de 0 a 1023, podemos concluir que quando temos 0V na entrada analógica teremos o valor 0 na variável de leitura e quando tivermos 5V na entrada analógica teremos o valor 1023 na variável de leitura. Então se dividirmos 5V por 1023 podemos encontrar quantos volts equivale cada incremento de 1bit, veja:

$$\text{Valor decada incremento} = \frac{5V}{1023} = 0.004887585533 \text{ Volts}$$

Logo teremos a variação de 1 na variável de leitura para cada variação de 0,004887585533 V na entrada analógica.

Conforme o datasheet (folha de dados técnicos ou manual de instruções do componente fornecido pelo fabricante) do LM35, ele varia 10mV para cada °C de temperatura. Então para obtermos o valor exato da temperatura em °C devemos realizar o seguinte cálculo em nosso código.

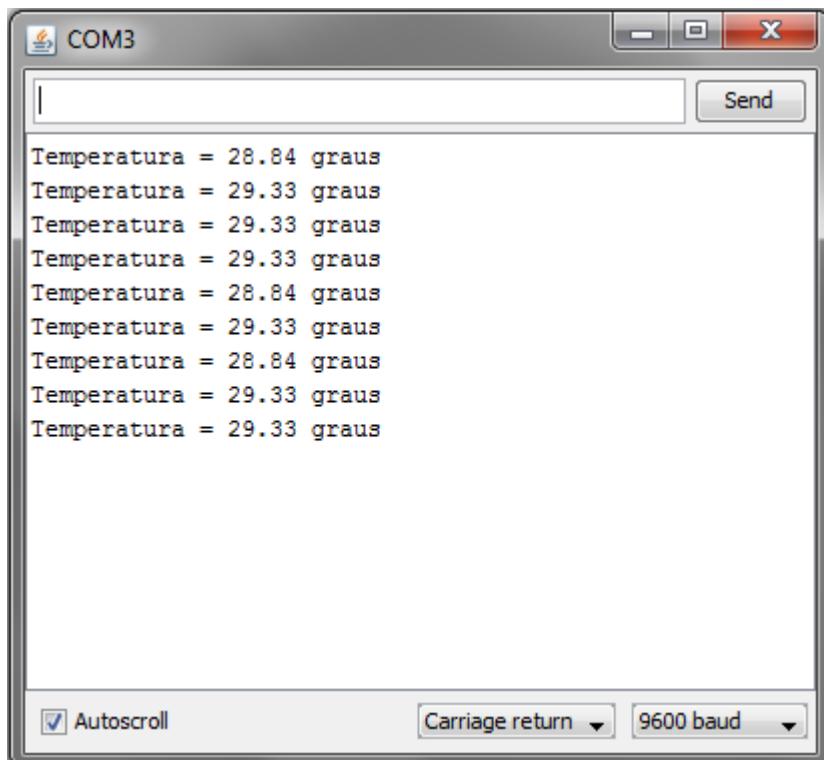
$$\text{Temperatura} = \frac{\text{Valor lido} \times 0.005887585533}{0.01}$$

Agora vamos ao nosso código fonte:

```
/*
=====
 Baú da Eletrônica Componentes Eletrônicos
 www.baudaelectronica.com.br
 PROJETO 9 - Medindo Temperatura com LM35
=====*/
int Pino_LM35 = 0;           //Sensor NTC conectado ao Pino 5
int Valor_Sensor = 0;         //Variável para armazenar valor lido do sensor
float Temperatura = 0;        //Variável para armazenar o valor em celsius

void setup()
{
    Serial.begin(9600); //Indica que vamos enviar dados pela porta USB
}
void loop()
{
    Valor_Sensor = analogRead(Pino_LM35); //Lê sinal do sensor
    Temperatura = (Valor_Sensor * 0.004887585533)/0.01; //Converte valor
    Serial.print ("Temperatura = "); //Escreve mensagem no monitor
    Serial.print (Temperatura);      //Escreve valor do sensor no monitor
    Serial.println (" graus");       //Escreve mensagem no monitor
    delay (1000);                  //Espera 1 segundo para mostrar novo valor
}
```

Para testar seu projeto você deve fazer o upload do código em sua placa Arduino e abrir o serial monitor. Ao fazer isso, você deverá ver algo similar ao mostrado na imagem abaixo.



Lembre-se que a porta COM não é necessariamente 3, como está no topo da imagem acima. cada computador tem sua numeração de portas.

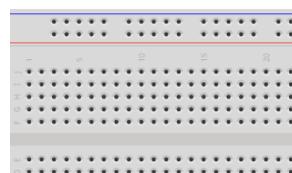
18.10 – Projeto 10 (Controle de iluminação automatizado)

Agora vamos aprender sobre um novo componente, o sensor de luminosidade LDR. Como já vimos, este sensor varia sua resistência de acordo com a incidência de luz no mesmo. Sendo assim, podemos verificar a luminosidade do ambiente e realizarmos diferentes tarefas de acordo com o valor lido, da mesma forma que já fizemos com o sensor de temperatura.

Neste projeto vamos medir a iluminação de um determinado ambiente, e caso o ambiente fique escuro, a lâmpada do sistema de iluminação se acende. Este sistema é similar ao que temos nos postes de iluminação nas ruas que se acendem ao escurecer.

18.10.1 – Componentes necessários

Protoboard



LDR



LED



Resistor 220R



Resistor 10K

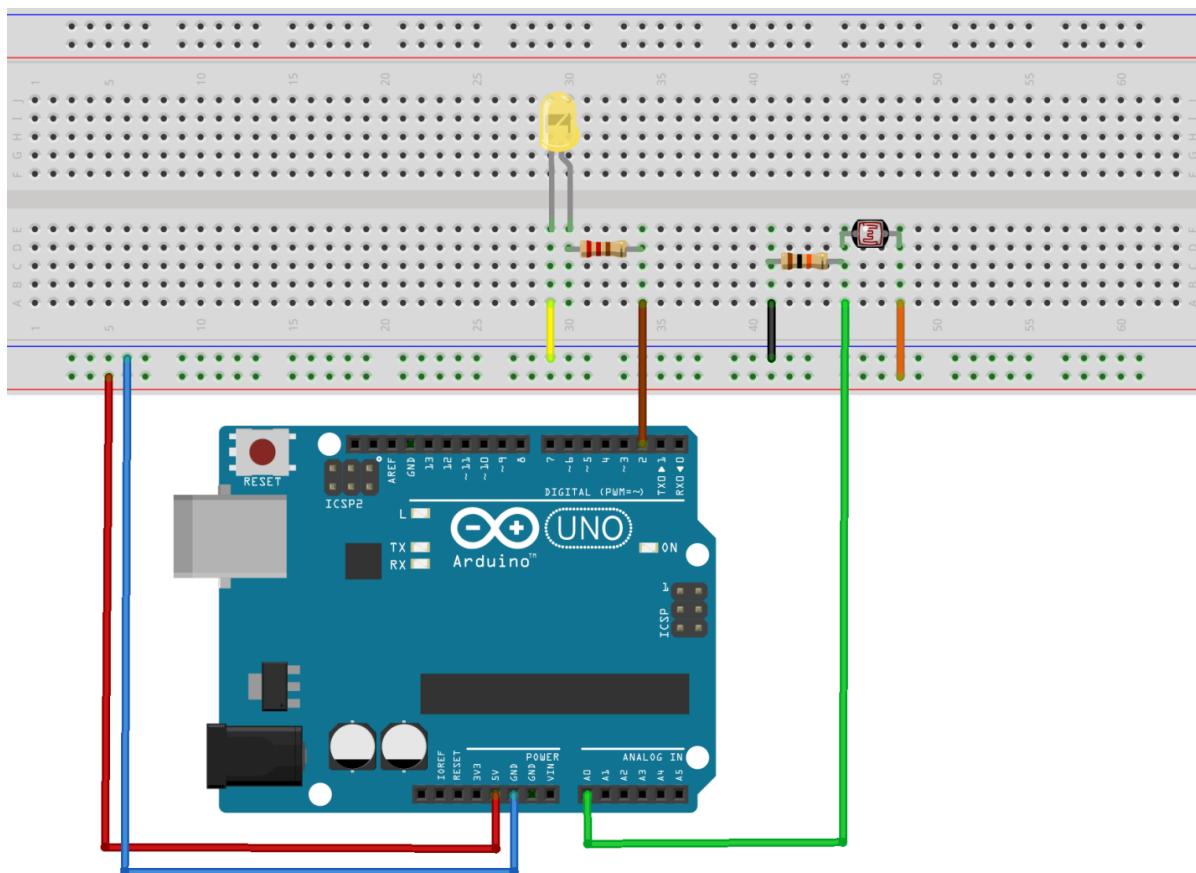


Jumpers



18.10.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.10.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
----- Baú da Eletrônica Componentes Eletrônicos
----- www.baudaelectronica.com.br
----- PROJETO 10 - Controle de iluminação automatizado
=====*/
int Pino_Led = 2;           //LED conectado ao pino 2
int LDR = 0;                //LDR conectado ao pino 0
int Luminosidade = 0;        //Variável para leitura do LDR

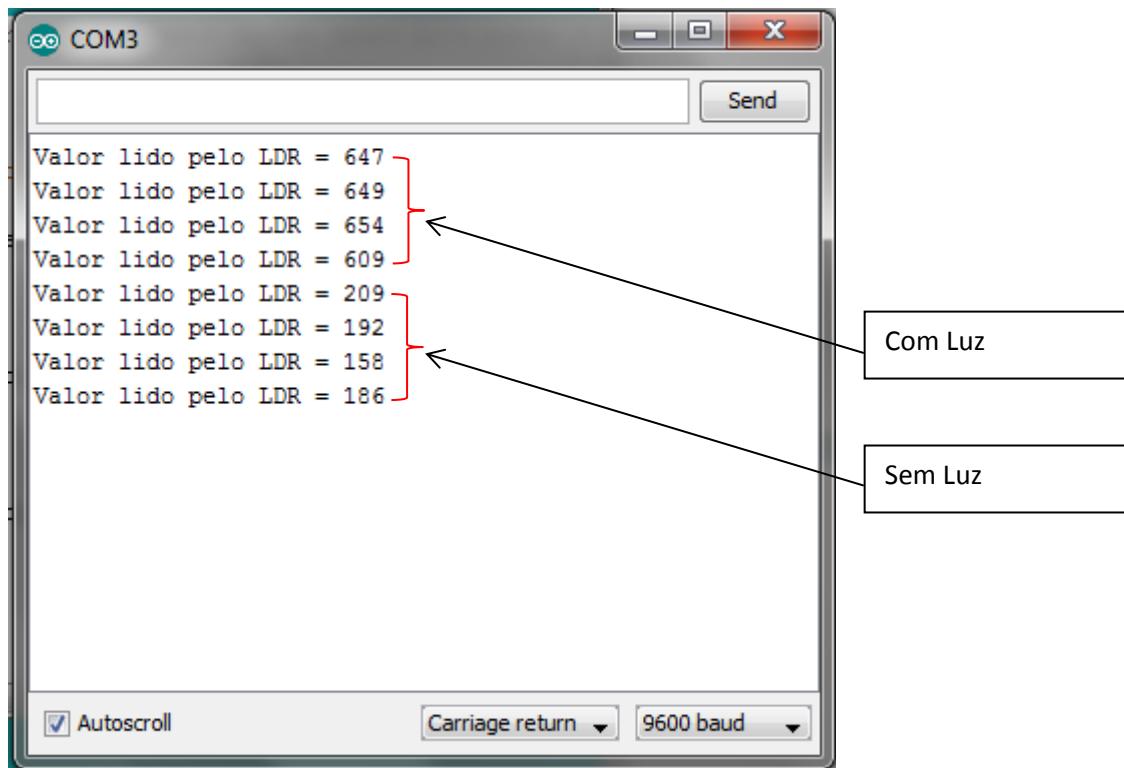
void setup()
{
    pinMode(Pino_Led, OUTPUT); //Pino 2 do arduino como saída
    Serial.begin(9600); //Indica que vamos enviar dados pela porta USB
}
void loop()
{
    Luminosidade = analogRead(LDR); //verifica valor do LDR
    Serial.print("Valor lido pelo LDR = "); //Escreve mensagem no monitor
    Serial.println(Luminosidade); // Escreve valor do LDR no monitor

    if (Luminosidade < 400) //Se luminosidade baixa
    {
        digitalWrite(Pino_Led, HIGH); //Acende a luz
    }
    else //se luminosidade alta
    {
        digitalWrite(Pino_Led, LOW); //Apaga a luz
    }
    delay (1000); //Aguarda 1 segundo para exibir nova leitura
}
```

Agora faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora se você tamar o LDR, o LED se acenderá, e se você deixar que a luz incida sobre o LDR o LED se apagará.

Como os componentes deste projeto podem sofrer variações de valores entre eles, foram inseridas no código fonte as instruções de escrita dos valores no monitor. Desta forma, você pode abrir o monitor e verificar o valor que está sendo lido com luz e o valor que está sendo lido com luz, e assim ajustar o ponto exato que você quer que o sistema atue.

No meu caso com luz o LDR apresenta valores entre 600 e 650 e sem luz o LDR apresenta valores entre 150 e 200 conforme mostrado na figura a seguir.



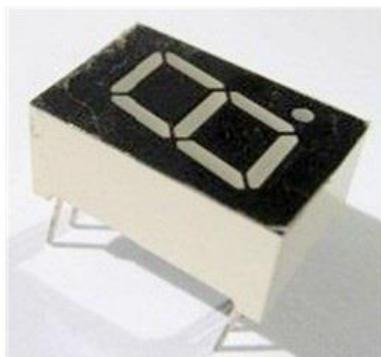
Então para o meu caso o valor 400 está funcional, caso você obtenha valores diferentes, ajuste o valor de atuação conforme mostrado a seguir para que seu projeto funcione adequadamente.

```
if (Luminosidade < 400) //Se luminosidade baixa
```

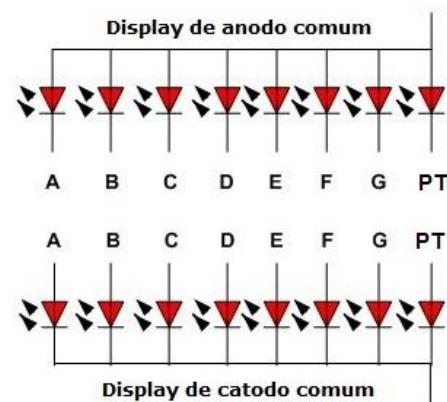
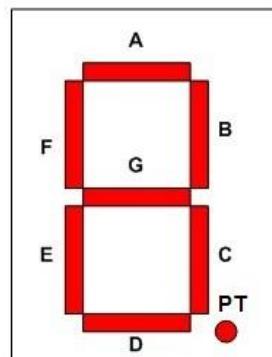
18.11 – Projeto 11 (Display BCD de 7 segmentos)

Neste experimento vamos aprender como funciona o display BCD de 7 segmentos. Este componente é bastante utilizado em relógios indicadores de temperatura entre outras aplicações.

Este tipo de display nada mais é que 8 leds dentro de um encapsulamento dispostos no formato de um número 8 e um ponto conforme ilustração a seguir.



Display BCD de 7 segmentos



Em seu kit você possui um display BCD do tipo catodo comum, isso quer dizer que os LEDs estão com os catodos conectados, dessa forma basta conectar o negativo ou GND ao pino catodo comum e conectar os segmentos que deseja acender ao positivo de maneira a formar um número ou letra.



Importante:

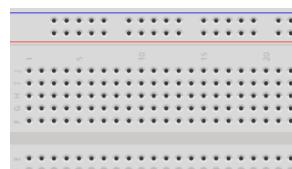
Tenha em mente que cada segmento do display é um LED, e já aprendemos que para ligar um LED devemos criar um divisor de tensão com um resistor, neste caso podemos utilizar apenas um resistor ligado ao catodo comum ao invés de colocar um resistor para cada LED.

Posição dos pinos do display BCD de 7 segmentos



18.11.1 – Componentes necessários

Protopboard



Display BCD



Resistor 220R

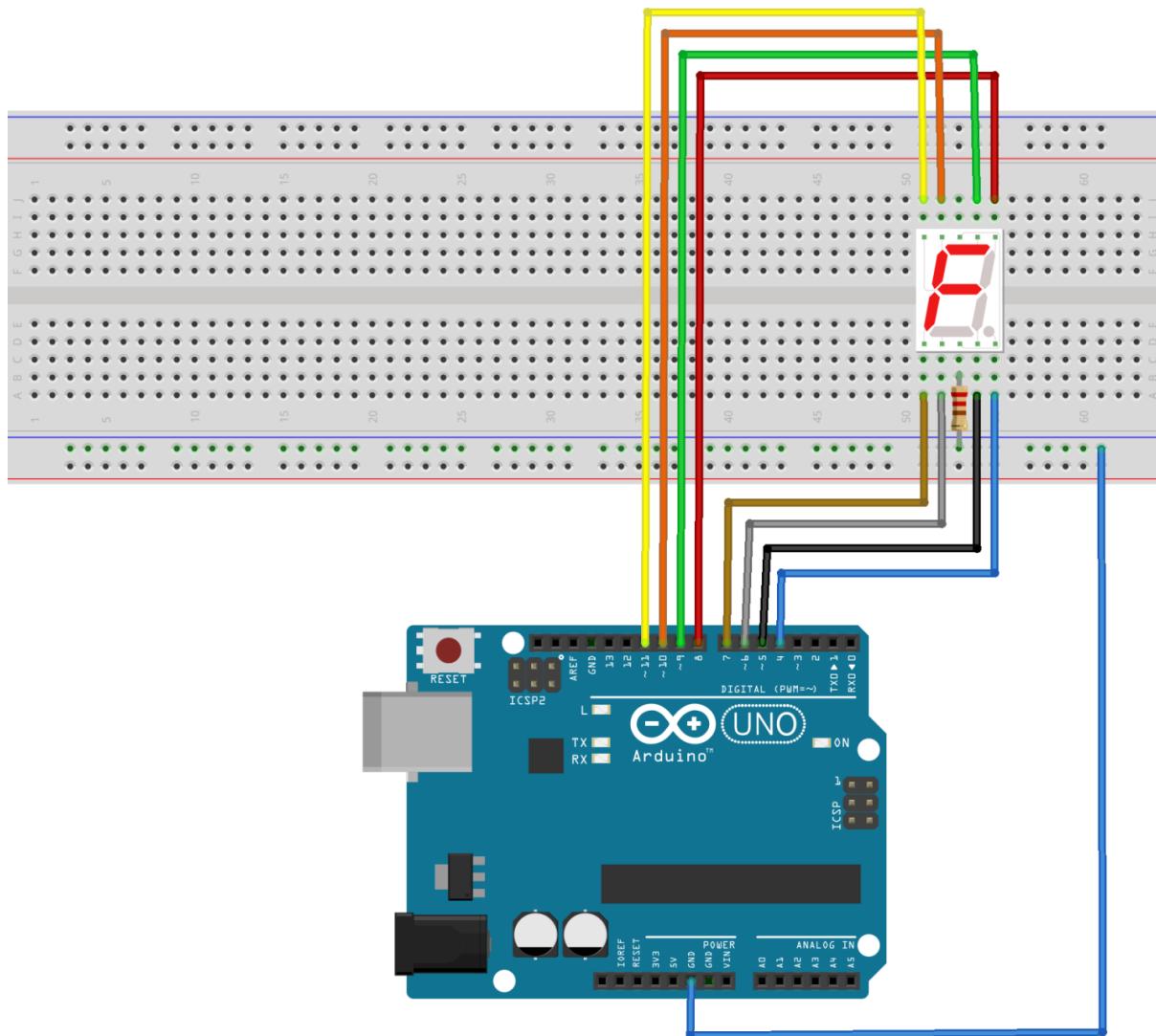


Jumpers



18.11.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.11.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
----- Baú da Eletrônica Componentes Eletrônicos
----- www.baudaeletronica.com.br
----- PROJETO 11 - Display BCD de 7 segmentos
----- */

int Pino_A = 9;           //Segmento A conectado ao pino 9 do Arduino
int Pino_B = 8;           //Segmento B conectado ao pino 8 do Arduino
int Pino_C = 5;           //Segmento C conectado ao pino 5 do Arduino
int Pino_D = 6;           //Segmento D conectado ao pino 6 do Arduino
int Pino_E = 7;           //Segmento E conectado ao pino 7 do Arduino
int Pino_F = 10;          //Segmento F conectado ao pino 10 do Arduino
int Pino_G = 11;          //Segmento G conectado ao pino 11 do Arduino
int Pino_PT = 4;          //Segmento PT conectado ao pino 4 do Arduino

void setup()
{
    pinMode(Pino_A, OUTPUT);    //Pino 9 do arduino como saída
    pinMode(Pino_B, OUTPUT);    //Pino 8 do arduino como saída
    pinMode(Pino_C, OUTPUT);    //Pino 5 do arduino como saída
    pinMode(Pino_D, OUTPUT);    //Pino 6 do arduino como saída
    pinMode(Pino_E, OUTPUT);    //Pino 7 do arduino como saída
    pinMode(Pino_F, OUTPUT);    //Pino 10 do arduino como saída
    pinMode(Pino_G, OUTPUT);    //Pino 11 do arduino como saída
    pinMode(Pino_PT, OUTPUT);   //Pino 4 do arduino como saída
}
void loop()
{
    digitalWrite(Pino_A, HIGH);  //Acende segmento A
    digitalWrite(Pino_B, HIGH);  //Acende segmento B
    digitalWrite(Pino_C, HIGH);  //Acende segmento C
    digitalWrite(Pino_D, HIGH);  //Acende segmento D
    digitalWrite(Pino_E, HIGH);  //Acende segmento E
    digitalWrite(Pino_F, HIGH);  //Acende segmento F
    digitalWrite(Pino_G, LOW);   //Apaga segmento G
    delay (1000);              //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_A, LOW);   //Apaga segmento A
    digitalWrite(Pino_B, HIGH);  //Acende segmento B
    digitalWrite(Pino_C, HIGH);  //Acende segmento C
    digitalWrite(Pino_D, LOW);   //Apaga segmento D
    digitalWrite(Pino_E, LOW);   //Apaga segmento E
    digitalWrite(Pino_F, LOW);   //Apaga segmento F
    digitalWrite(Pino_G, LOW);   //Apaga segmento G
    delay (1000);              //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_A, HIGH);  //Acende segmento A
    digitalWrite(Pino_B, HIGH);  //Acende segmento B
    digitalWrite(Pino_C, LOW);   //Apaga segmento C
    digitalWrite(Pino_D, HIGH);  //Acende segmento D
    digitalWrite(Pino_E, HIGH);  //Acende segmento E
    digitalWrite(Pino_F, LOW);   //Apaga segmento F
    digitalWrite(Pino_G, HIGH);  //Acende segmento G
    delay (1000);              //Aguarda 1 segundo para exibir novo número

    /*CONTINUA NA PRÓXIMA PÁGINA*/
```

```

/* CONTINUAÇÃO */

digitalWrite(Pino_A, HIGH);      //Acende segmento A
digitalWrite(Pino_B, HIGH);      //Acende segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, HIGH);      //Acende segmento D
digitalWrite(Pino_E, LOW);       //Apaga segmento E
digitalWrite(Pino_F, LOW);       //Apaga segmento F
digitalWrite(Pino_G, HIGH);      //Acende segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_A, LOW);       //Apaga segmento A
digitalWrite(Pino_B, HIGH);      //Acende segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, LOW);       //Apaga segmento D
digitalWrite(Pino_E, LOW);       //Apaga segmento E
digitalWrite(Pino_F, HIGH);      //Acende segmento F
digitalWrite(Pino_G, HIGH);      //Acende segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_A, HIGH);      //Acende segmento A
digitalWrite(Pino_B, LOW);       //Apaga segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, HIGH);      //Acende segmento D
digitalWrite(Pino_E, LOW);       //Apaga segmento E
digitalWrite(Pino_F, HIGH);      //Acende segmento F
digitalWrite(Pino_G, HIGH);      //Acende segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_A, HIGH);      //Acende segmento A
digitalWrite(Pino_B, LOW);       //Apaga segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, HIGH);      //Acende segmento D
digitalWrite(Pino_E, HIGH);      //Acende segmento E
digitalWrite(Pino_F, HIGH);      //Acende segmento F
digitalWrite(Pino_G, HIGH);      //Acende segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_A, HIGH);      //Acende segmento A
digitalWrite(Pino_B, HIGH);      //Acende segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, LOW);       //Apaga segmento D
digitalWrite(Pino_E, LOW);       //Apaga segmento E
digitalWrite(Pino_F, LOW);       //Apaga segmento F
digitalWrite(Pino_G, LOW);       //Apaga segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_A, HIGH);      //Acende segmento A
digitalWrite(Pino_B, HIGH);      //Acende segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, HIGH);      //Acende segmento D
digitalWrite(Pino_E, HIGH);      //Acende segmento E
digitalWrite(Pino_F, HIGH);      //Acende segmento F
digitalWrite(Pino_G, HIGH);      //Acende segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_A, HIGH);      //Acende segmento A
digitalWrite(Pino_B, HIGH);      //Acende segmento B
digitalWrite(Pino_C, HIGH);      //Acende segmento C
digitalWrite(Pino_D, HIGH);      //Acende segmento D
digitalWrite(Pino_E, LOW);       //Apaga segmento E
digitalWrite(Pino_F, HIGH);      //Acende segmento F
digitalWrite(Pino_G, HIGH);      //Acende segmento G
delay (1000);                  //Aguarda 1 segundo para exibir novo número
}

```

Agora faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora você verá o display indicar do número 0 ao 9 alterando em intervalo de 1 segundo.

Note que o código fonte deste projeto ficou relativamente grande, mas note que mais uma vez fizemos 10 cópias de uma mesma estrutura de código mudando apenas os segmentos do display que queríamos acender afim de formar os algarismos em sequência.

O código funcionou perfeitamente, mas para realizar este projeto utilizamos oito pinos da nossa placa Arduino. Isso não é problema quando temos oito pinos disponíveis para esta operação, mas imagine que você está realizando um projeto maior controlando, leds, buzzer, lendo sinais de sensores entre outras coisas, talvez não sobrem o pinos disponíveis para controlar o display. Neste caso podemos utilizar um outro componente bastante interessante para diminuir a quantidade de pinos necessários para o controle do display de 8 para 4 pinos e ainda facilitar a escrita do código fonte. Este componente é o circuito integrado 4511 (Conversor BCD para display de 7 segmentos).

Este circuito integrado converte um código binário para o display de 7 segmentos. A tabela a seguir mostra os números binários correspondentes a cada dígito do display.

Valor em Binário				Valor Decimal
Pino 1	Pino 2	Pino 3	Pino 4	Display
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

A partir da tabela acima podemos então criar nosso código fonte considerando que quando a tabela indicar 0, o pino do Arduino está desligado, ou em nível lógico 0 (utilizaremos para isso o comando `digitalWrite (Pino, LOW)`) e quando a tabela indicar 1, o pino do Arduino está ligado, ou em nível lógico 1 (utilizaremos para isso o comando `digitalWrite (Pino, HIGH)`).

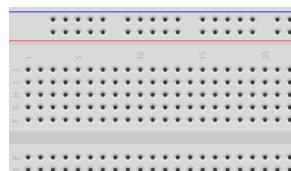
Desta forma com apenas 4 pinos do nosso Arduino podemos controlar nosso display. Vamos colocar isso em prática no projeto a seguir.

18.12 – Projeto 12 (Display BCD de 7 segmentos com 4511)

Dando sequência ao experimento anterior agora vamos montar o mesmo projeto montado anteriormente, mas agora utilizando apenas 4 pinos digitais de nossa placa Arduino e enviando para o display um código binário ao invés de controlar cada segmento de forma independente.

18.12.1 – Componentes necessários

Protopboard



Display BCD



Resistor 220R



CI 4511

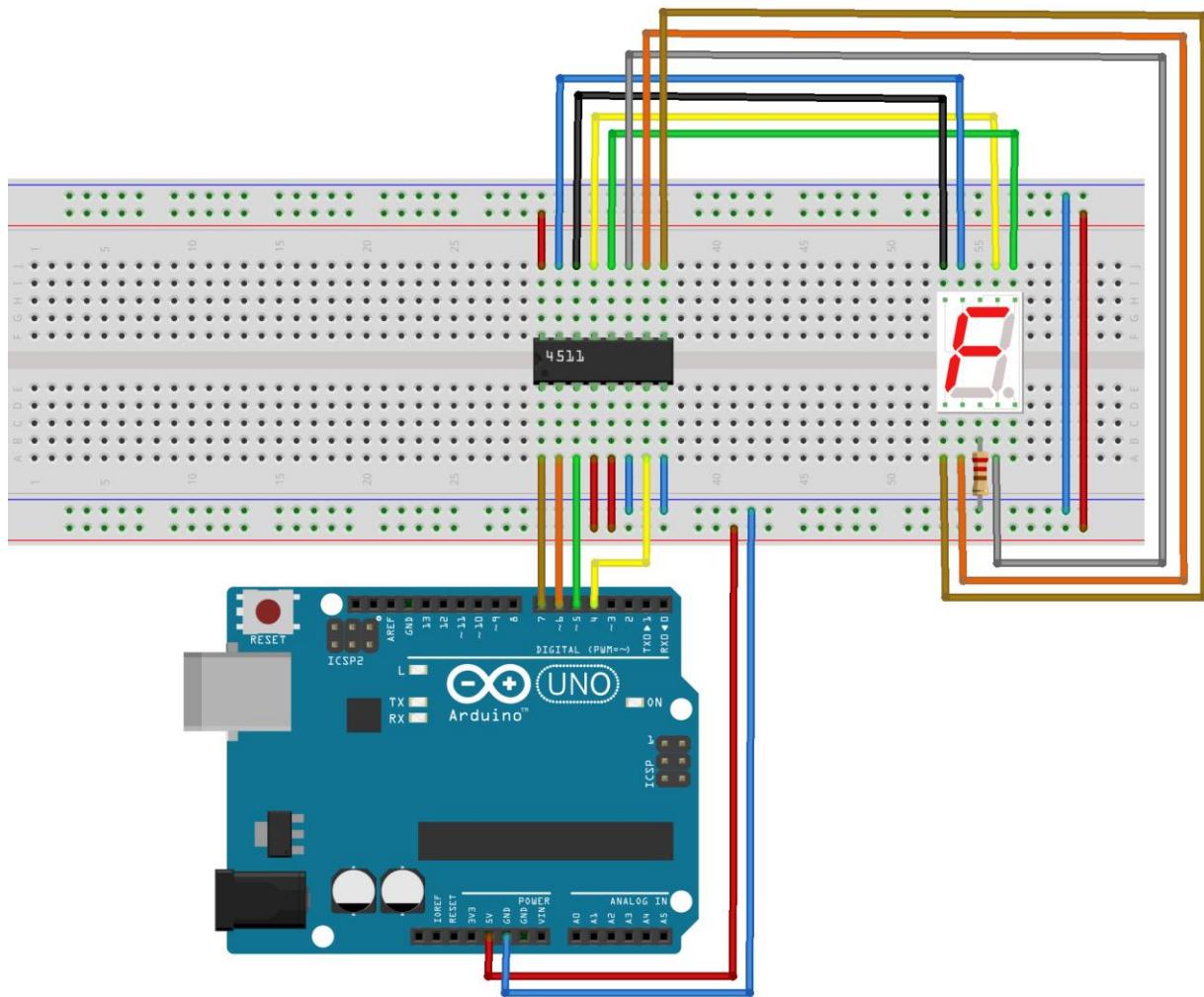


Jumpers



18.12.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.12.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
    Baú da Eletrônica Componentes Eletrônicos
    www.baudaelectronica.com.br
    PROJETO 12 - Display BCD de 7 segmentos com 4511
=====

int Pino_D0 = 4;           //Pino D0 do 4511 conectado ao pino 4 do Arduino
int Pino_D1 = 7;           //Pino D1 do 4511 conectado ao pino 7 do Arduino
int Pino_D2 = 6;           //Pino D2 do 4511 conectado ao pino 6 do Arduino
int Pino_D3 = 5;           //Pino D3 do 4511 conectado ao pino 5 do Arduino

void setup()
{
    pinMode(Pino_D0, OUTPUT);    //Pino 4 do arduino como saída
    pinMode(Pino_D1, OUTPUT);    //Pino 7 do arduino como saída
    pinMode(Pino_D2, OUTPUT);    //Pino 6 do arduino como saída
    pinMode(Pino_D3, OUTPUT);    //Pino 5 do arduino como saída
}
void loop()
{

    digitalWrite(Pino_D0, LOW);   //Envia a combinação binária equivalente
    digitalWrite(Pino_D1, LOW);   //ao dígito 0
    digitalWrite(Pino_D2, LOW);
    digitalWrite(Pino_D3, LOW);
    delay (1000);               //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_D0, HIGH);  //Envia a combinação binária equivalente
    digitalWrite(Pino_D1, LOW);   //ao dígito 1
    digitalWrite(Pino_D2, LOW);
    digitalWrite(Pino_D3, LOW);
    delay (1000);               //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_D0, LOW);   //Envia a combinação binária equivalente
    digitalWrite(Pino_D1, HIGH);  //ao dígito 2
    digitalWrite(Pino_D2, LOW);
    digitalWrite(Pino_D3, LOW);
    delay (1000);               //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_D0, HIGH);  //Envia a combinação binária equivalente
    digitalWrite(Pino_D1, HIGH);  //ao dígito 3
    digitalWrite(Pino_D2, LOW);
    digitalWrite(Pino_D3, LOW);
    delay (1000);               //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_D0, LOW);   //Envia a combinação binária equivalente
    digitalWrite(Pino_D1, LOW);   //ao dígito 4
    digitalWrite(Pino_D2, HIGH);
    digitalWrite(Pino_D3, LOW);
    delay (1000);               //Aguarda 1 segundo para exibir novo número

    digitalWrite(Pino_D0, HIGH);  //Envia a combinação binária equivalente
    digitalWrite(Pino_D1, LOW);   //ao dígito 5
    digitalWrite(Pino_D2, HIGH);
    digitalWrite(Pino_D3, LOW);
    delay (1000);               //Aguarda 1 segundo para exibir novo número

/*CONTINUA NA PRÓXIMA PÁGINA*/
}
```

```
/* CONTINUAÇÃO */

digitalWrite(Pino_D0, LOW);      //Envia a combinação binária equivalente
digitalWrite(Pino_D1, HIGH);     //ao dígito 6
digitalWrite(Pino_D2, HIGH);
digitalWrite(Pino_D3, LOW);
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_D0, HIGH);     //Envia a combinação binária equivalente
digitalWrite(Pino_D1, HIGH);     //ao dígito 7
digitalWrite(Pino_D2, HIGH);
digitalWrite(Pino_D3, LOW);
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_D0, LOW);      //Envia a combinação binária equivalente
digitalWrite(Pino_D1, LOW);      //ao dígito 8
digitalWrite(Pino_D2, LOW);
digitalWrite(Pino_D3, HIGH);
delay (1000);                  //Aguarda 1 segundo para exibir novo número

digitalWrite(Pino_D0, HIGH);     //Envia a combinação binária equivalente
digitalWrite(Pino_D1, LOW);      //ao dígito 9
digitalWrite(Pino_D2, LOW);
digitalWrite(Pino_D3, HIGH);
delay (1000);                  //Aguarda 1 segundo para exibir novo número
}
```

Agora faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora você verá o display indicar do número 0 ao 9 alterando em intervalo de 1 segundo.

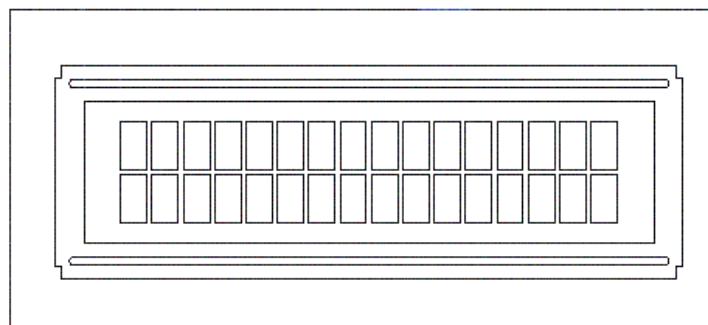
Note que agora o código fonte deste projeto ficou bem menor e ocupamos menos portas da nossa placa Arduino.

Nesta configuração com o CI 4511, o ponto permanece apagado, caso você queira acender o segmento do ponto, é necessário que você ligue-o a um pino da placa Arduino.

18.13 – Projeto 13 (Escrevendo no display LCD 16x2)

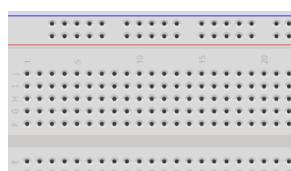
Nesta etapa do nosso aprendizado através deste manual, vamos aprender sobre o display de cristal líquido 16x2. Este display é bastante utilizado em equipamentos eletrônicos para possibilitar a interação do usuário com o equipamento.

Este display é conhecido com Display LCD 16x2 pois possui 16 colunas (posições para escrita de caracteres ASCII) e duas linhas conforme figura a seguir.

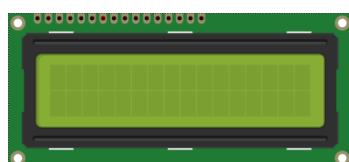


18.13.1 – Componentes necessários

Protopboard



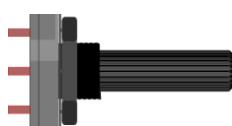
Display LCD 16x2



Resistor 300R



Potenciômetro 1K

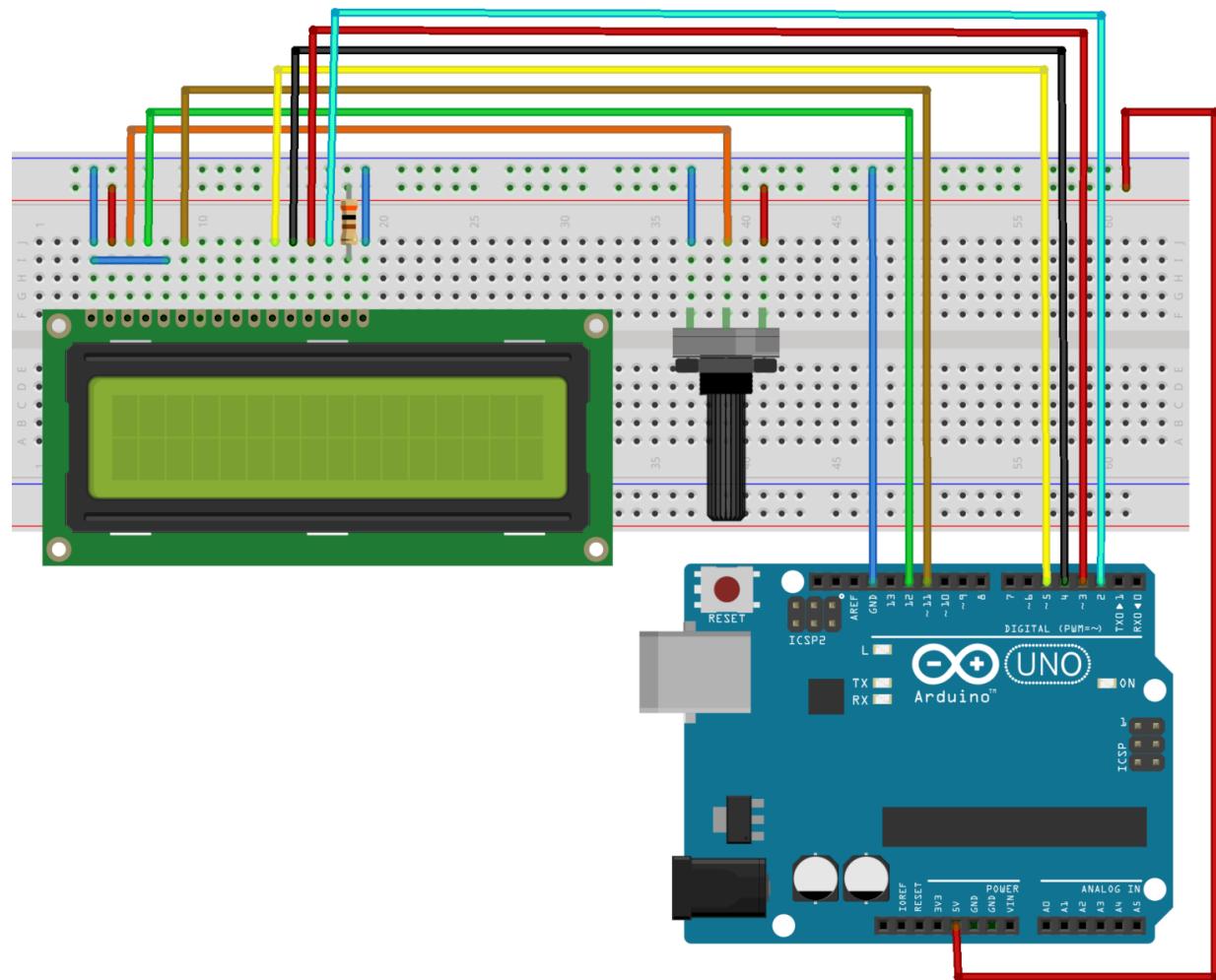


Jumpers



18.13.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.

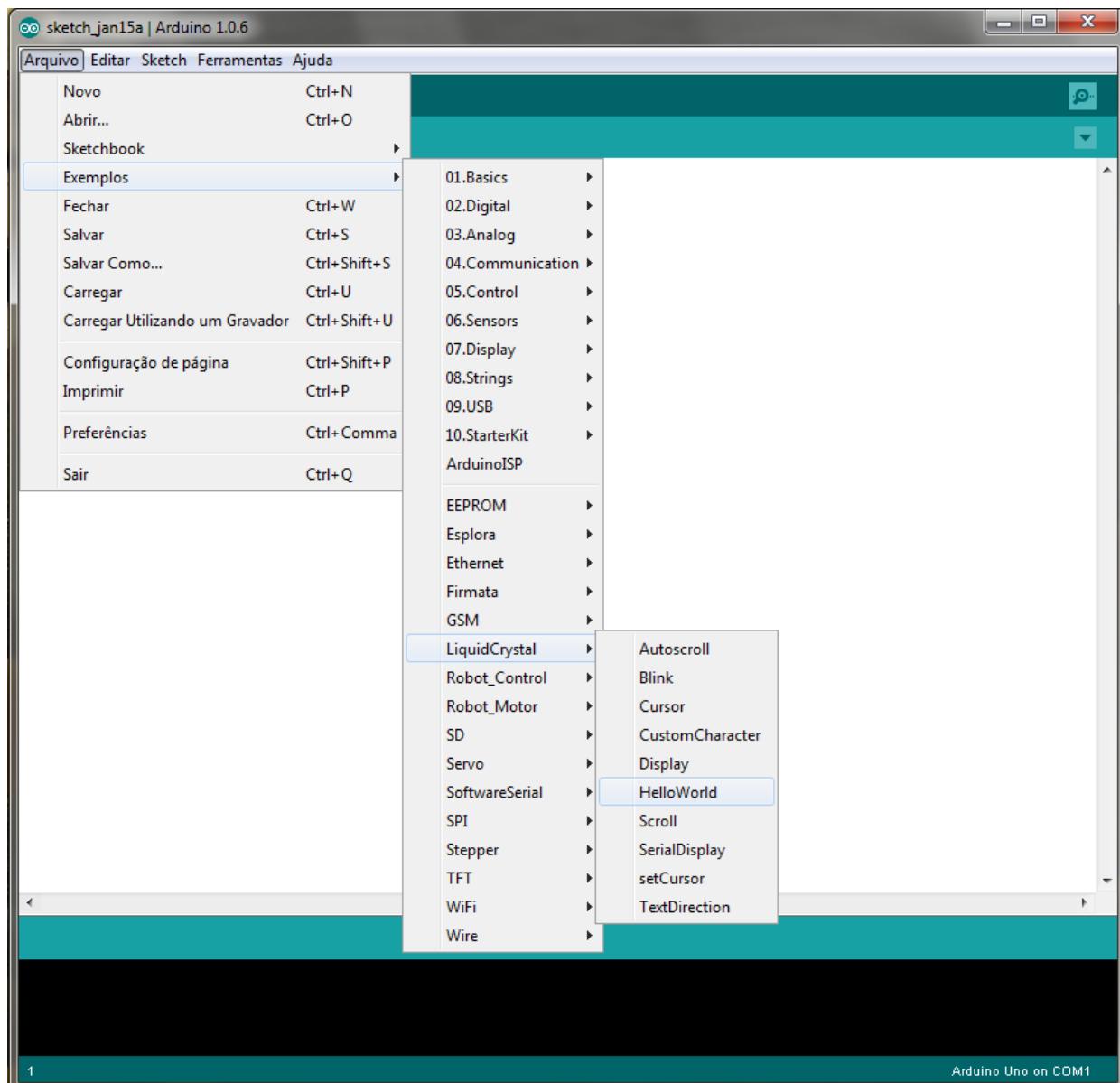


Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.13.3 – Código fonte

O software IDE do Arduino possui uma imensa biblioteca de códigos fonte de exemplo para diversas aplicações. Esta biblioteca é muito interessante pois com ela podemos aprender como utilizar um determinado componente. Para este projeto, utilizaremos um destes códigos fonte de exemplo, o “Hello World”. Desta forma veremos como utilizar este recurso.

Para abrir um código exemplo, clique no menu: “Arquivo > Exemplos > LiquidCrystal > HelloWorld” conforme imagem abaixo.



Feito isso, o seguinte código fonte se abrirá na área de trabalho do software IDE.

```
/*
LiquidCrystal Library - Hello World

Demonstrates the use a 16x2 LCD display. The LiquidCrystal
library works with all LCD displays that are compatible with the
Hitachi HD44780 driver. There are many of them out there, and you
can usually tell them by the 16-pin interface.

This sketch prints "Hello World!" to the LCD
and shows the time.

The circuit:
* LCD RS pin to digital pin 12
* LCD Enable pin to digital pin 11
* LCD D4 pin to digital pin 5
* LCD D5 pin to digital pin 4
* LCD D6 pin to digital pin 3
* LCD D7 pin to digital pin 2
* LCD R/W pin to ground
* LCD VSS pin to ground
* LCD VCC pin to 5V
* 10K resistor:
* ends to +5V and ground
* wiper to LCD VO pin (pin 3)

Library originally added 18 Apr 2008
by David A. Mellis
library modified 5 Jul 2009
by Limor Fried (http://www.ladyada.net)
example added 9 Jul 2009
by Tom Igoe
modified 22 Nov 2010
by Tom Igoe

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/LiquidCrystal
*/



// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
    lcd.print("hello, world!");
}

void loop() {
    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0):
    lcd.setCursor(0, 1);
    // print the number of seconds since reset:
    lcd.print(millis()/1000);
}
```

Agora faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora você verá o display indicar a mensagem “hello world! na primeira linha e uma contagem de segundos na segunda linha conforme imagem abaixo.



Utilizando este código de exemplo você já consegue escrever alguma coisa no seu display, caso queira alterar a mensagem a ser escrita, basta alterar dentro do código fonte, o que está entre aspas na instrução `lcd.print("hello, world!")`; e descarregar o código novamente em sua placa Arduino.

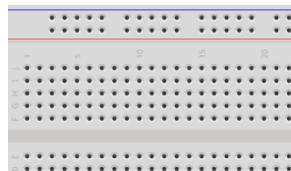
Desta forma sua mensagem será exibida no display de forma estática, ou seja para alterá-la você precisará alterar o código fonte e descarregar novamente. No projeto a seguir vamos deixar esta mensagem dinâmica, ou seja, ela poderá ser alterada em tempo real através do monitor serial.

18.14 – Projeto 14 (Enviando mensagens para o LCD 16x2)

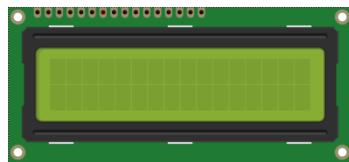
Neste projeto vamos aprender a enviar uma mensagem para nossa placa Arduino através do Serial Monitor e fazer com que esta mensagem seja exibida no display LCD 16x2.

18.14.1 – Componentes necessários

Protopboard



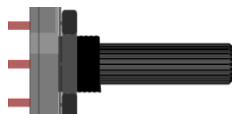
Display LCD 16x2



Resistor 300R



Potenciômetro 1K

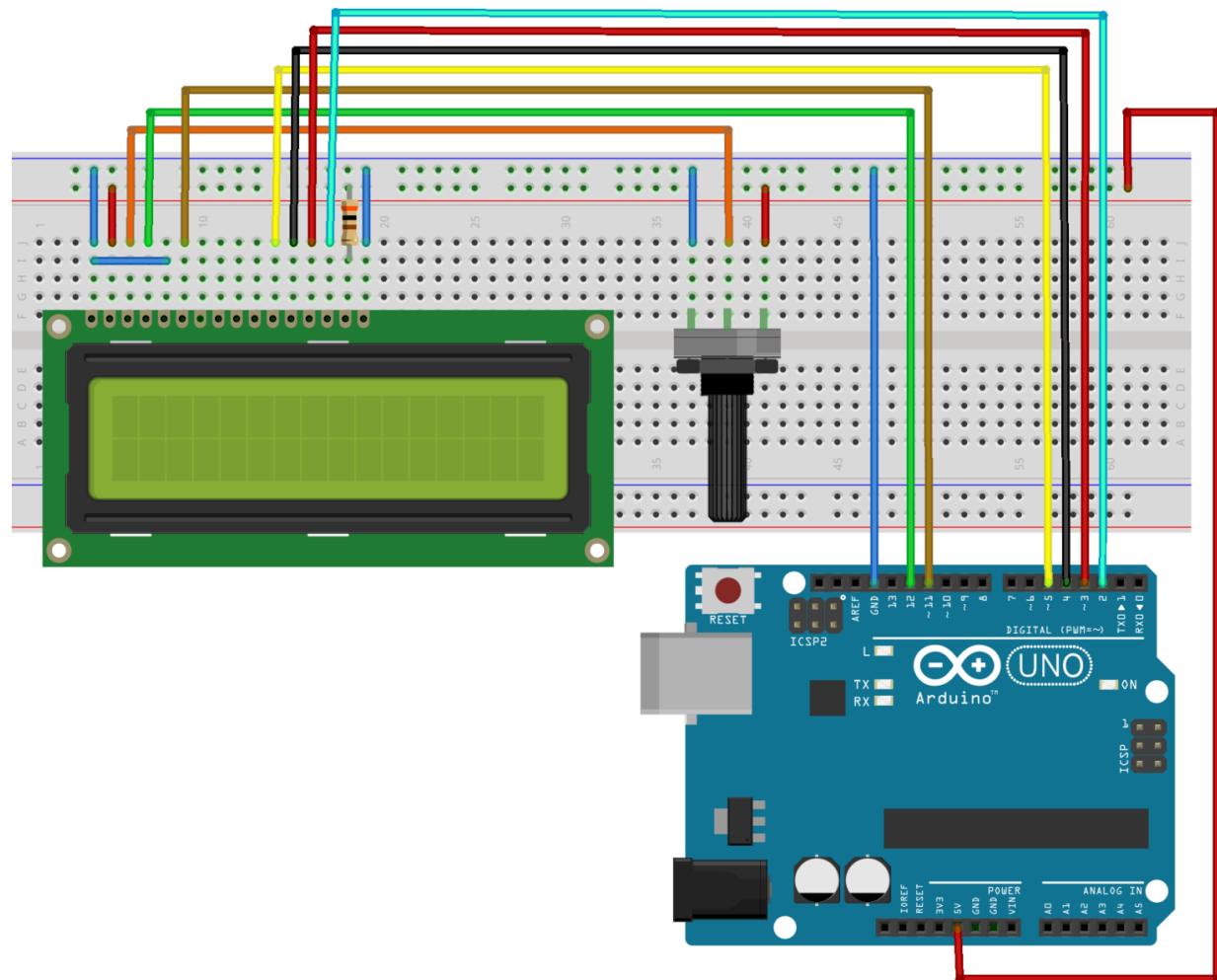


Jumpers



18.14.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.14.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

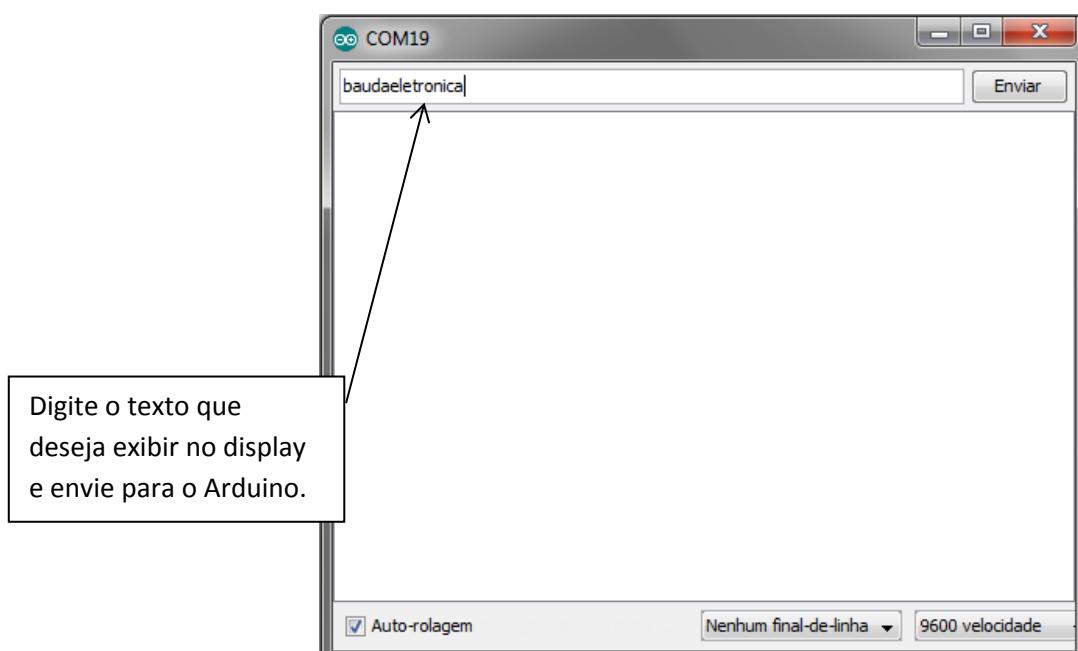
```
/*
    Baú da Eletrônica Componentes Eletrônicos
    www.baudaeletronica.com.br
    Projeto 14 - Enviando mensagens para o LCD 16x2
*/
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  lcd.begin(16, 2);
}

void loop()
{
  if (Serial.available())
  {
    delay(100);
    lcd.clear();
    while (Serial.available() > 0)
    {
      lcd.write(Serial.read());
    }
  }
}
```

Para testar seu projeto você deve fazer o upload do código em sua placa Arduino e abrir o serial monitor.



18.15 – Projeto 15 (Emitindo sons com o Piezo Buzzer)

Neste projeto vamos utilizar a função de emissor de ruídos do elemento piezoelétrico. Vamos utilizar o piezobuzzer para ouvir a frequência de 2Khz.

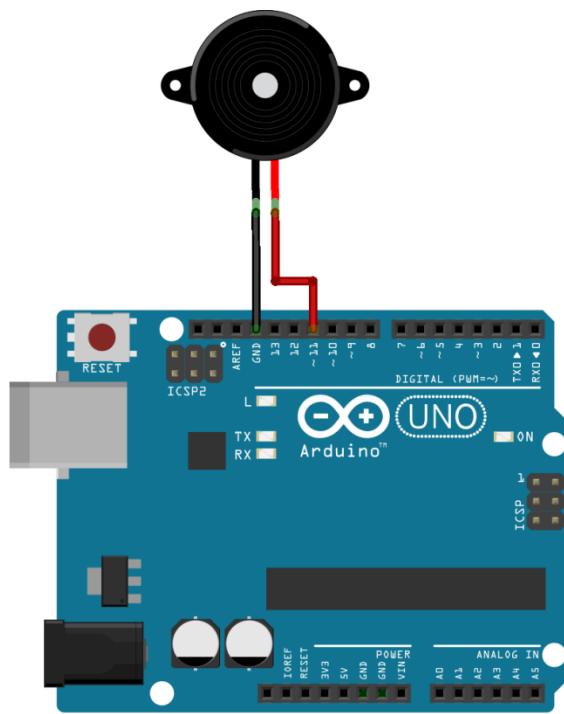
18.15.1 – Componentes necessários

Elemento Piezoelétrico



18.15.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Importante:

Tenha bastante atenção à posição correta de ligar o piezo, pois o mesmo possui polaridade, sendo o fio vermelho o positivo e o fio preto o negativo.

Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.15.3 – Código fonte

Neste código fonte vamos ver uma nova função, a função tone() do Arduino. Esta função é muito útil, pois permite gerar frequências de maneira muito simples e rápida, e deixando ainda o programa com boa performance de execução.

Agora vamos ao nosso código fonte:

```
/*
=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaeletronica.com.br
PROJETO 15 - Emitindo sons com o Piezo Buzzer
=====*/
int Piezo = 11;           // Piezo Buzzer conectado ao pino 11 do Arduino
void setup()
{
    pinMode(Piezo, OUTPUT); // Pino 11 configurado como saída
}
void loop()
{
    tone(Piezo, 2000) ;    //Frequência de 2kHz aplicada ao Buzzer
}
```

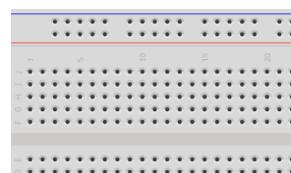
Para testar seu projeto você deve fazer o upload do código em sua placa Arduino e então você deverá ouvir um som agudo emitido pelo piezo buzzer.

18.16 – Projeto 16 (Sensor de toque com o Piezo Buzzer)

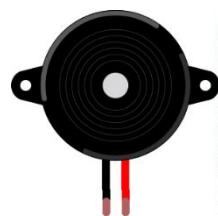
Agora vamos utilizar a função de sensor de toque do elemento piezoelétrico. Neste experimento ao tocar no elemento piezoelétrico, será mostrada uma informação no monitor serial de que houve o toque.

18.16.1 – Componentes necessários

Protoboard



Elemento Piezoelétrico



Resistor 10K

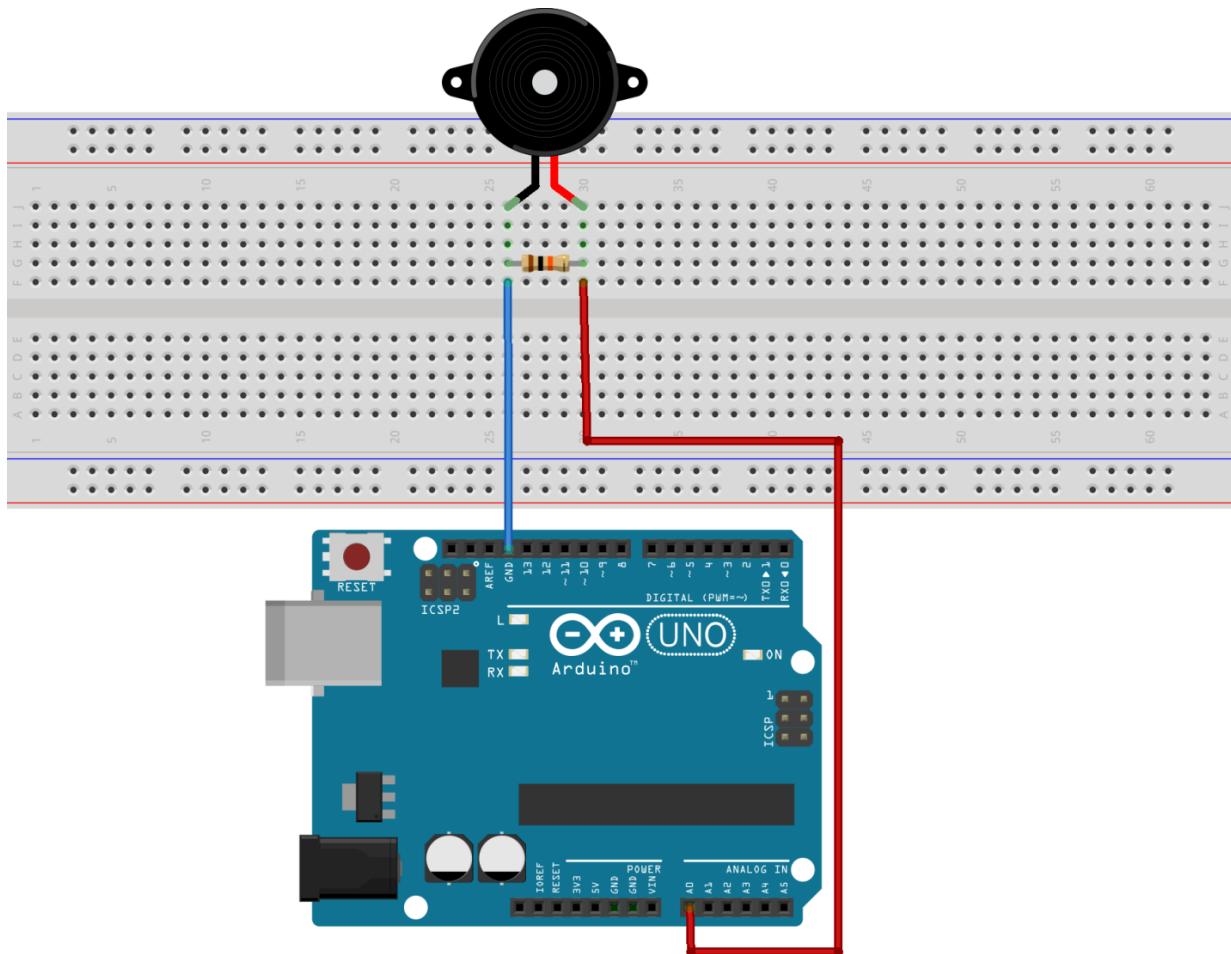


Jumpers



18.16.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Importante:

Tenha bastante atenção à posição correta de ligar o piezo, pois o mesmo possui polaridade, sendo o fio vermelho o positivo e o fio preto o negativo. Note também que existe um resistor de 10 K em paralelo com o piezo.

Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.16.3 – Código fonte

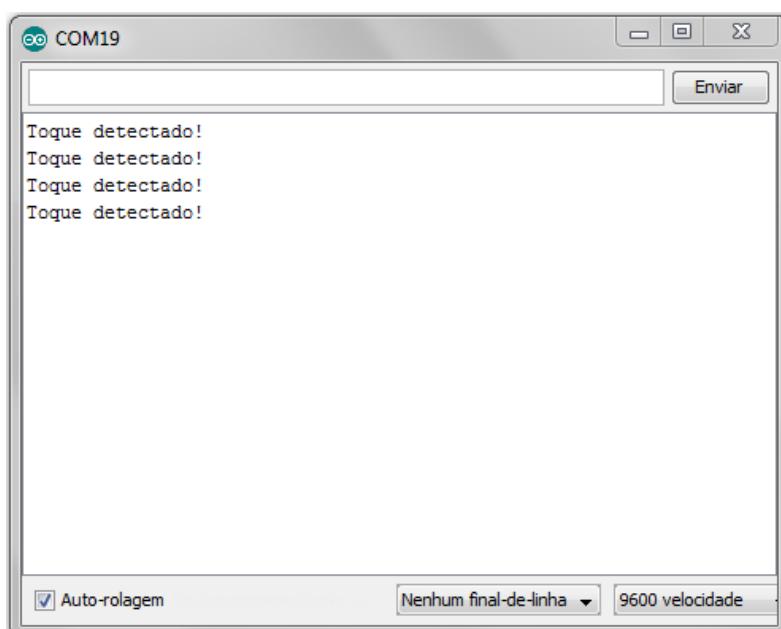
Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
 Baú da Eletrônica Componentes Eletrônicos
 www.baudaeletronica.com.br
 PROJETO 16 - Sensor de toque com o Piezo Buzzer
=====*/
int Piezo = A0; // Piezo conectado ao pino analógico 0
int sensibilidade = 30; // Valor de sensibilidade do toque
int pressao_toque = 0; // variável para armazenar o valor lido no sensor

void setup()
{
    Serial.begin(9600);
}
void loop()
{
    pressao_toque = analogRead(Piezo); //lê o valor analógico lido em (A0)
                                    // e armazena na variável pressao_toque

    if (pressao_toque >= sensibilidade)//Se o valor lido for mais que
                                    //o valor de sensibilidade, faça:
    {
        Serial.println("Toque detectado!"); //Imprima no Monitor Serial
                                            //a palavra Toque detectado! indicando
                                            //que houve toque.
    }
}
```

Para testar seu projeto você deve fazer o upload do código em sua placa Arduino e abrir o serial monitor. Então sempre que pressionar o piezo, você deverá ver a mensagem “Toque detectado” conforme mostrado abaixo.

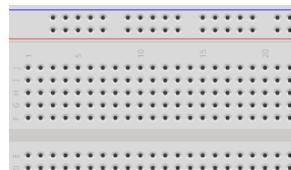


18.17 – Projeto 17 (Controlando o Motor DC)

Neste experimento vamos aprender como funcionam dois novos componentes do nosso kit, o motor DC e o Circuito integrado L293D. O experimento consiste em controlar o sentido e velocidade do motor DC, o sentido de rotação será controlado através de dois botões, onde ao apertar cada botão individualmente vamos selecionar um sentido de rotação diferente, ao pressionar os dois botões ao mesmo tempo o motor para, e a velocidade do motor será controlada através de um potenciômetro.

18.17.1 – Componentes necessários

Protoboard



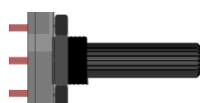
2 Chaves Tácteis



L293D



Potenciômetro



Motor DC



Jumpers

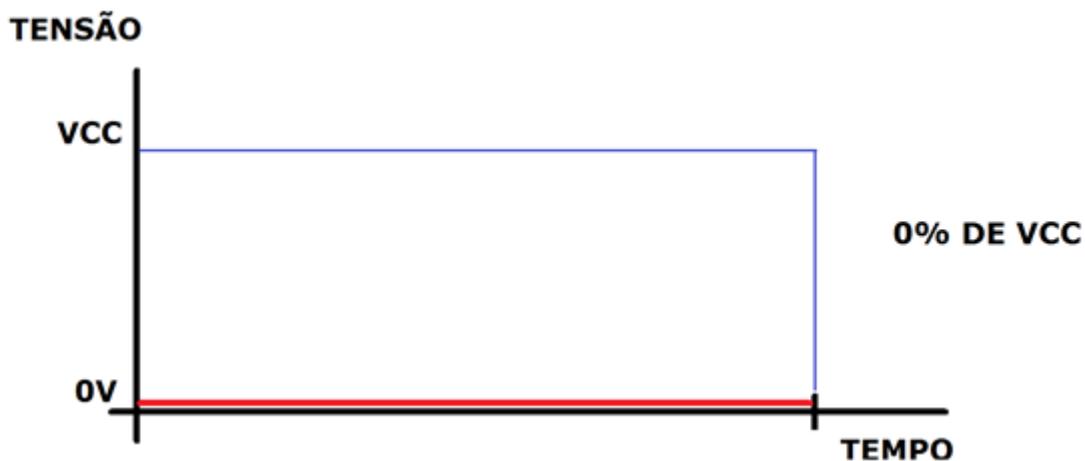


18.17.2 – Montando o circuito

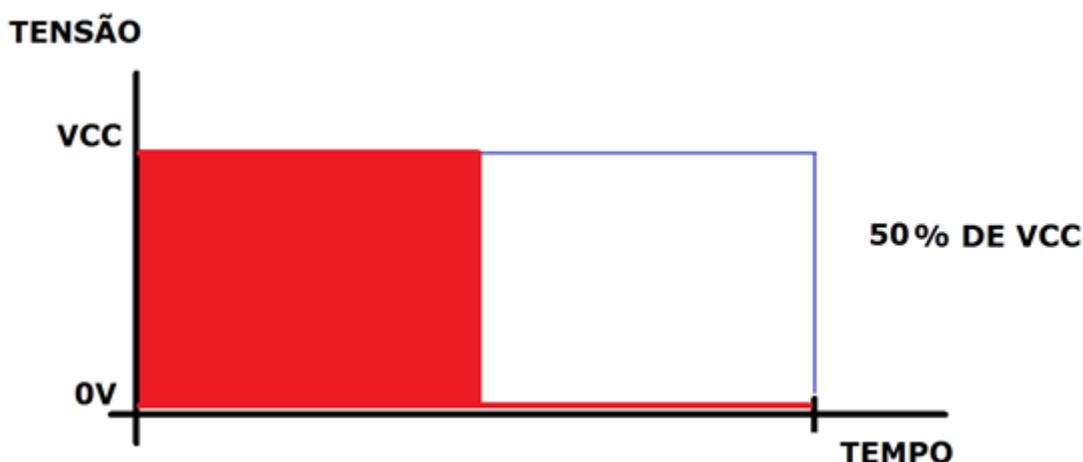
Antes de montar o circuito você precisa conhecer melhor os componentes do projeto. Primeiramente vamos ver o motor DC e saber de duas características importantes de seu funcionamento, uma é que para inverter seu sentido de rotação basta inverter o positivo com o negativo e a outra é que para variar sua velocidade basta variar a tensão aplicada em seus terminais.

Exemplo: no kit existe um motor de 5V, isso quer dizer que com 5V ele vai rodar em sua rotação máxima. Caso aplique 2,5V ele vai rodar com metade de sua rotação nominal.

Para variar a tensão no motor com o Arduino vamos utilizar o PWM. Esta sigla significa Pulse Width Modulation, ou seja, modulação por largura de pulso. De uma maneira bem simples, esta técnica pode ser explicada como a média da tensão em um determinado espaço de tempo. Veja nos gráficos a seguir:



Nos gráficos acima a tensão está representada pela cor vermelha, note que a tensão fica em zero durante todo o período, desta maneira a média será 0.

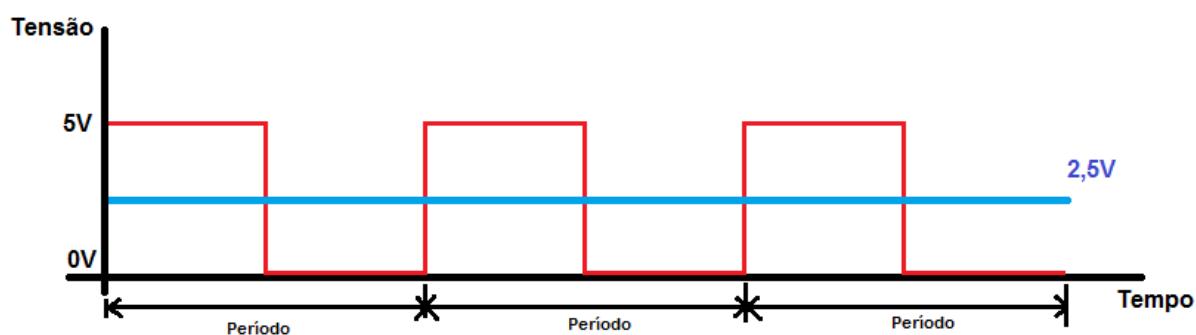


No segundo gráfico, note que a tensão é mantida em VCC por metade do período e depois é mantida em zero volts a outra metade, desta forma temos uma média de 50% do valor de tensão de VCC (no caso do nosso Arduino que VCC representa 5V teremos uma tensão resultante de 2,5V).



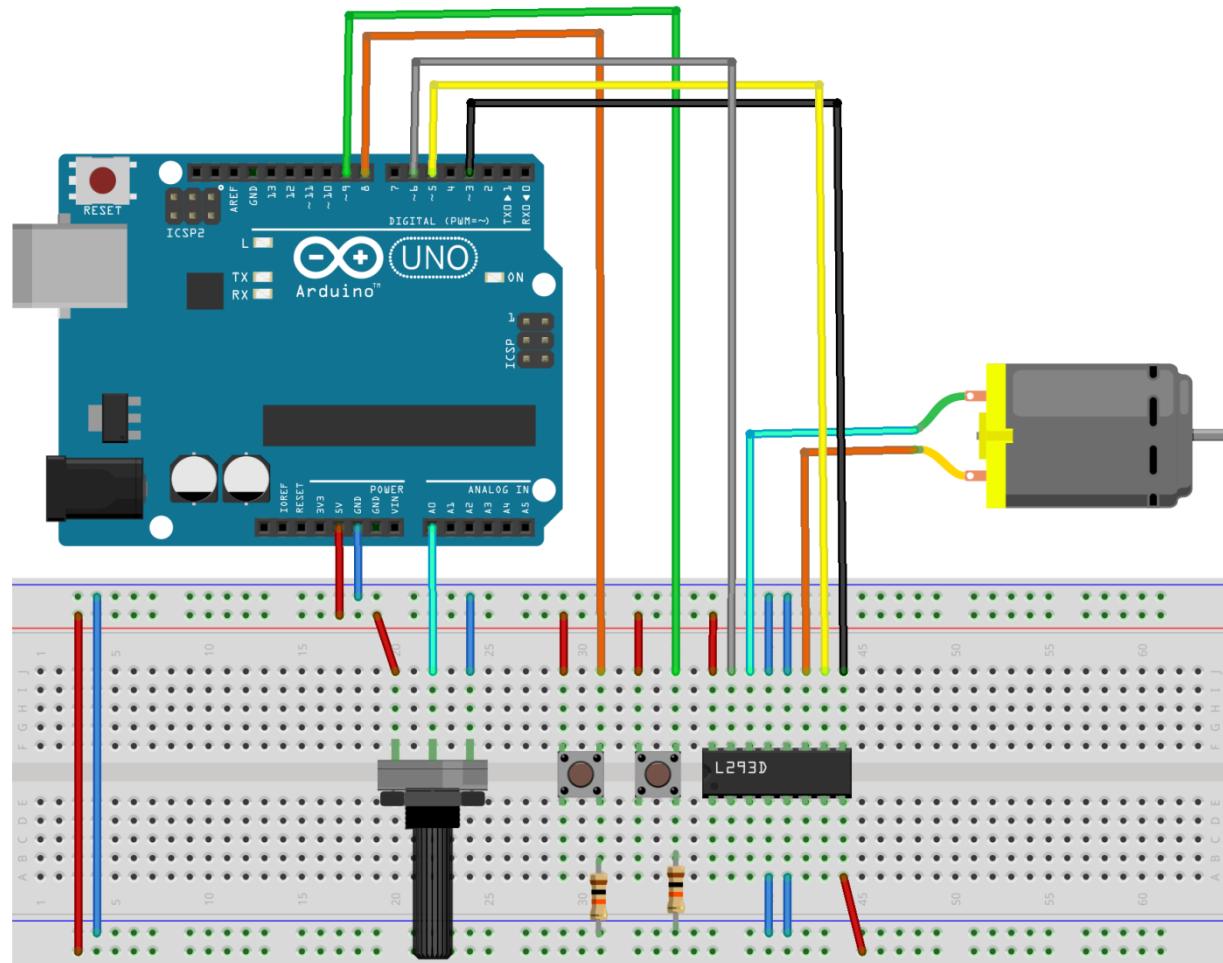
E neste outro gráfico a tensão é mantida no máximo por todo o período, desta forma temos uma média de 100% de VCC.

Vamos ver agora como isso ocorre em nossa placa Arduino. Imagine que nosso motor de 5 V está sendo alimentado por uma tensão PWM que fica metade do período ligado e metade desligado, isso vai resultar numa tensão de 2,5V, desta forma nosso motor irá funcionar na metade de sua velocidade, pois a média será 50% do valor de VCC. Veja isso exemplificado no gráfico abaixo.



Agora que vimos o que é o PWM, e para que serve, precisamos conhecer o circuito integrado L298D. Este circuito integrado, bastante conhecido como ponte H, é utilizado para facilitar o controle do motor DC. Ele possui uma saída para conectar o motor, e três entradas digitais para controle do motor, as quais ligamos aos pinos do Arduino para realizar o controle do motor.

Agora, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB.
Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.17.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaeletronica.com.br
PROJETO 17 - Controlando o Motor DC
=====*/
int botao1 = 8;
int botao2 = 9;
int horario = 5;
int anti_horario = 6;
int pwm = 3;
int potenciometro = 0;
int Velocidade = 0;
int valor_botao1 = 0;
int valor_botao2 = 0;

void setup()
{
    pinMode(horario, OUTPUT);
    pinMode(anti_horario, OUTPUT);
    pinMode(pwm, OUTPUT);
    pinMode(botao1, INPUT);
    pinMode(botao2, INPUT);
    Serial.begin(9600);
}
void loop()
{
    valor_botao1 = digitalRead (botao1);
    valor_botao2 = digitalRead (botao2);
    Velocidade = analogRead(potenciometro) / 4;

    while (valor_botao1==0 && valor_botao2 ==0)
    {
        valor_botao1 = digitalRead (botao1);
        valor_botao2 = digitalRead (botao2);
        Velocidade = analogRead(potenciometro) / 4;
        analogWrite(pwm, Velocidade);
    }

    if (valor_botao1==1 && valor_botao2 ==0)
    {
        digitalWrite(horario, LOW); // define a Entrada 1 do L293D como
        digitalWrite(anti_horario, HIGH); // define a Entrada 2 do L293D
    }

    else if (valor_botao1==0 && valor_botao2 ==1)
    {
        digitalWrite(horario, HIGH); // define a Entrada 1 do L293D como
        digitalWrite(anti_horario, LOW); // define a Entrada 2 do L293D
    }

    else if (valor_botao1==1 && valor_botao2 ==1)
    {
        digitalWrite(horario, LOW); // define a Entrada 1 do L293D como
        digitalWrite(anti_horario, LOW); // define a Entrada 2 do L293D
        delay (1000);
    }
}
```

Agora faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, ao pressionar um dos botões, o motor deve rodar em um sentido e se variar o potenciômetro, a velocidade do motor deve variar proporcionalmente.

Conforme descrito anteriormente na introdução do experimento, ao pressionar o outro botão o motor inverterá o sentido de rotação também com ajuste de velocidade através do potenciômetro e ao pressionar os dois botões ao mesmo tempo o moto desliga.

18.18 – Projeto 18 (Movimentos com Micro Servo 9g)

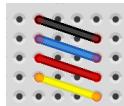
Para você iniciante em eletrônica que está tendo seu primeiro contato com este universo de possibilidades, este é sem dúvida um projeto bastante empolgante e que desperta muitas ideias, pois vamos controlar os movimentos do micro servo 9g com o Arduino. Junto ao micro servo 9g acompanham 3 diferentes acoplamentos plásticos para poder acoplá-lo facilmente ao seu projeto, escolha um deles e instale no eixo do micro servo para facilitar a visualização dos movimentos realizados. Este projeto poderia ser utilizado para criar um pequeno robô, um braço eletrônico para movimentação de objetos, entre outras muitas aplicações. Mas como já dito anteriormente, crie o projeto conforme mostrado a seguir, e faça as alterações apenas depois de saber exatamente como tudo funciona.

18.18.1 – Componentes necessários

Micro Servo 9g

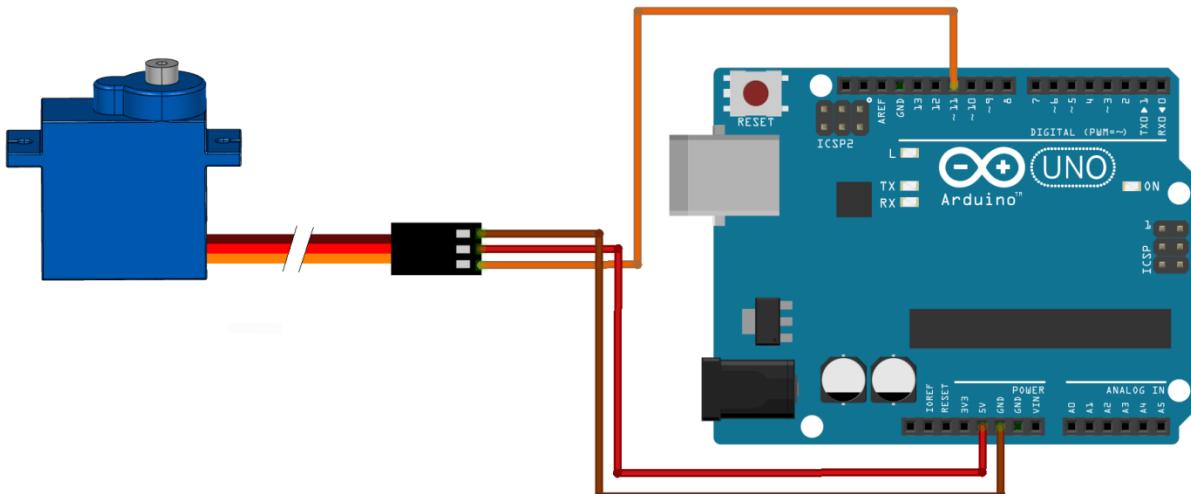


Jumpers



18.18.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.18.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
Baú da Eletrônica Componentes Eletrônicos
www.baudaelectronica.com.br
Projeto 18 - Movimentos com Micro Servo 9g
=====*/
#include <Servo.h>

Servo myservo;
int posicao = 0;
int dado_novo = 0;
int dado_anterior = 0;
int pino_controle = 11;

void setup()
{
    Serial.begin(9600);
    Serial.println("Envie a posicao em graus de 0 a 180!");
    myservo.attach(pino_controle);
    myservo.write(posicao);
    delay(500);
    myservo.detach();
}

void loop()
{
    if (Serial.available() > 0)
    {
        dado_novo = Serial.parseInt();
        if(posicao < dado_novo)
        {
            myservo.attach(pino_controle);
            for(posicao = dado_anterior; posicao <= dado_novo; posicao += 1)
            {
                myservo.write(posicao);
                delay(5);
            }
            myservo.detach();
            dado_anterior = dado_novo;
        }

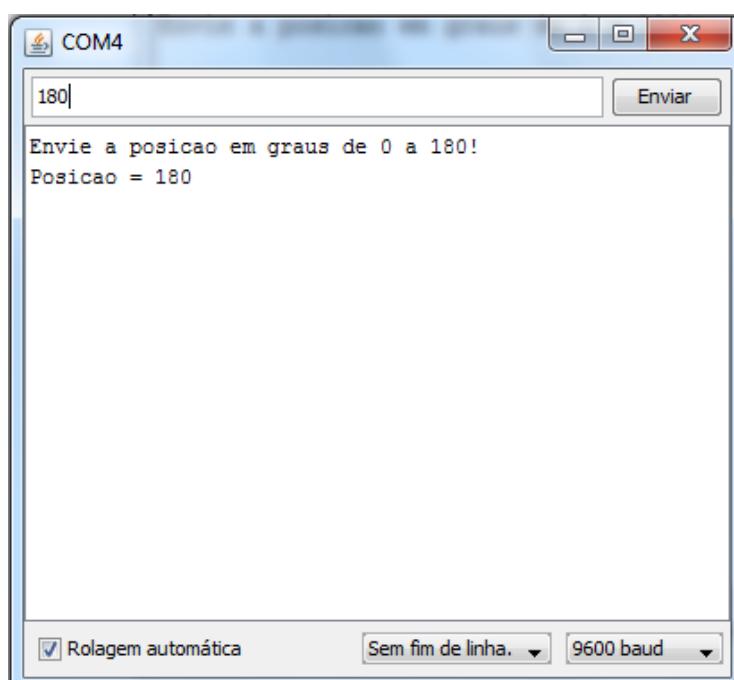
        if(posicao > dado_novo)
        {
            myservo.attach(pino_controle);
            for(posicao = dado_anterior; posicao >= dado_novo; posicao -= 1)
            {
                myservo.write(posicao);
                delay(5);
            }
            myservo.detach();
            dado_anterior = dado_novo;
        }

        Serial.print("Posicao = ");
        Serial.println(dado_novo, DEC);
    }
}
```

Faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, agora você verá o seu servo movimentar sozinho para a posição inicial (0). Após este movimento abra o serial monitor e você deve ter a seguinte mensagem:



Para controlar a posição absoluta do motor basta enviar a posição desejada (de 0 a 180) na linha de comandos na parte superior do serial monitor. Ao enviarmos 180, o microservo deve realizar uma meia volta (180°) e você deve receber a nova posição do motor como mostrado a seguir.



18.19 – Projeto 19 (Automação com módulo Relé)

Neste experimento vamos aprender a controlar (ligar e desligar) equipamentos que necessitem de uma corrente maior do que a que um pino de saída digital de nossa placa Arduino possa fornecer (40mA) através do módulo relé.

Um relê é um dispositivo eletromecânico que permite o acionamento de cargas elétricas a partir de um circuito de comando de baixa potência, como, por exemplo, um sinal digital. Um relê é constituído basicamente por uma bobina e contatos. Possuem contatos NA (Normalmente Aberto) ou NF (Normalmente Fechado), dependendo do modelo do relê escolhido. Esta nomenclatura se refere principalmente ao estado de repouso da bobina do relê. Quando o relê está desligado, o contato NA está aberto e o NF fechado, porém quando é acionada a bobina do relê os contatos mudam de estado, ou seja, o contato NA é fechado e o NF é aberto. Esse estado permanece enquanto a bobina estiver acionada, e, quando ela é desligada, os contatos retornam para o estado de repouso.

O módulo relé é bastante utilizado em projetos de automação residencial para ligar e desligar lâmpadas, ventiladores entre outros aparelhos que funcionem com tensão alternada por exemplo 110V ou 220V. Ele funciona como um intérprete entre o Arduino que opera em 5V e o dispositivo ou equipamento de 110V ou 220V.

Como trabalhar com estes valores de tensão alternada pode ser perigoso vamos simular o funcionamento da lâmpada 110V com o motor DC para exemplificar, onde o motor ligado significa a lâmpada acesa e o motor desligado significa lâmpada apagada.

18.19.1 – Componentes necessários

Módulo Relé



Resistor 10K



Chave Táctil



Motor DC

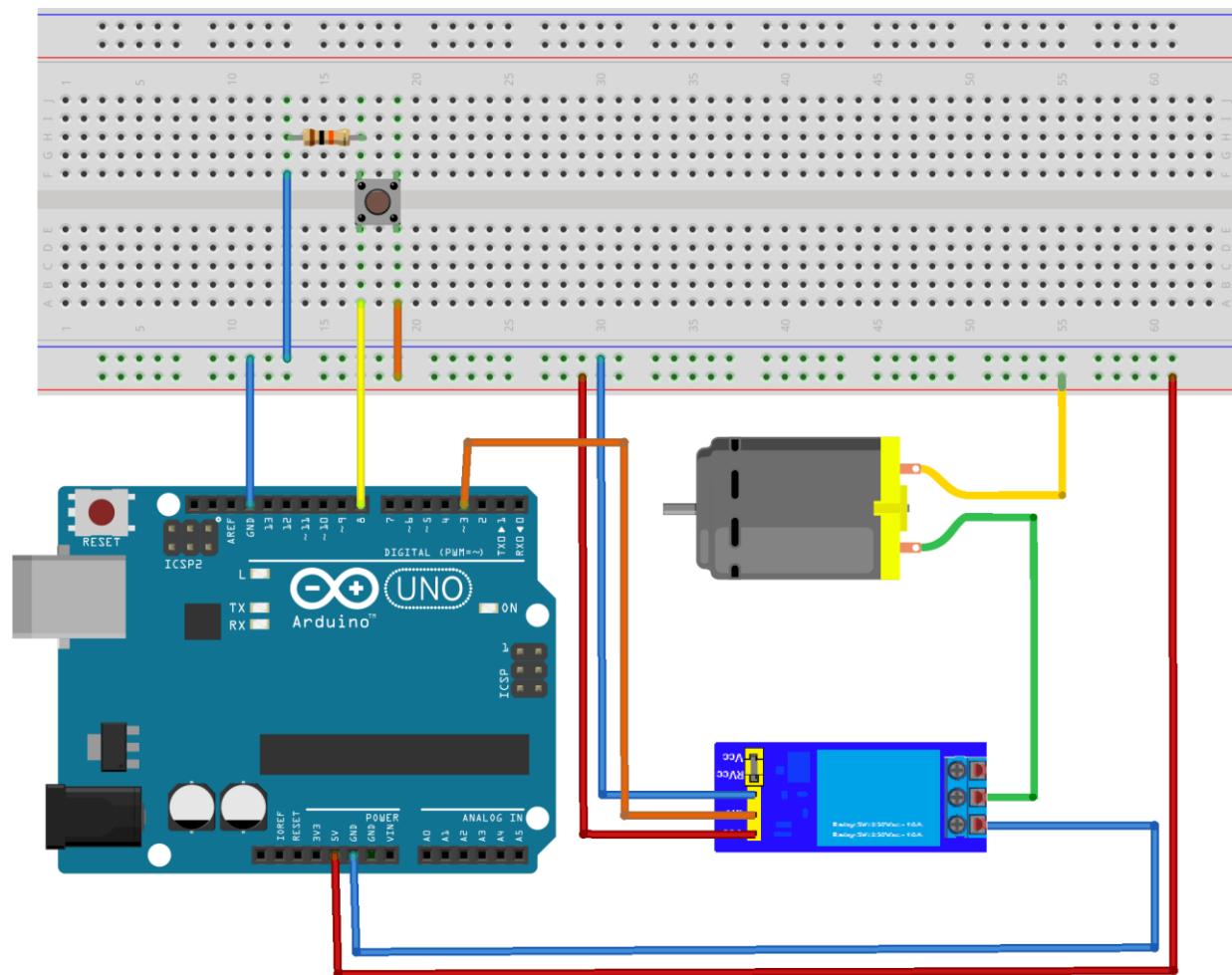


Jumpers



18.19.2 – Montando o circuito

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue os componentes e conecte tudo como mostra a figura abaixo.



Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

18.19.3 – Código fonte

Abra seu IDE do Arduino e digite o código a seguir:

```
/*
=====
    Baú da Eletrônica Componentes Eletrônicos
    www.baudaelectronica.com.br
    Projeto 19 - Automação com módulo Relé
=====*/
int botao = 8;                      //Botão conectado ao pino 8 do Arduino
int rele = 3;                        //Relé conectado ao pino 3 do Arduino
int valor_botao = 0;                 //Variável para leitura do botão
int liga_motor = 0;                  //Variável para armazenar o estado do motor
                                      //onde 1 = motor ligado e 0 = motor desligado

void setup()
{
    pinMode(botao, INPUT);           //Pino 8 do arduino como entrada
    pinMode(rele, OUTPUT);          //Pino 2 do arduino como saída
    digitalWrite (rele, HIGH);       // Desliga o Relé
}
void loop()
{
    valor_botao = digitalRead (botao); //Memoriza o estado do motor
    if (valor_botao == HIGH)          //Se o botão for pressionado
    {
        digitalWrite (rele, LOW);     //Liga o modulo Relé
        liga_motor = 1;              //Armazena o estado atual do motor
        while (valor_botao == HIGH)   //Caso o botão continue pressionado
        {valor_botao = digitalRead(botao);} //aguarda até que seja solto
        delay(200);
        while(liga_motor==1)          //Enquanto o motor estiver ligado
        {
            valor_botao = digitalRead (botao);
            if (valor_botao == HIGH)      //Se o Botão for Pressionado
            {
                digitalWrite (rele, HIGH); //Desliga o Relê
                liga_motor = 0;          //Armazena o estado do motor
                while ( valor_botao == HIGH) // Caso o botão continue
                //pressionado
                {valor_botao = digitalRead(botao);} //Se o Botão for Pressionado
                delay(200);
            }
        }
    }
}
```

Faça o upload do código na sua placa Arduino e caso tudo tenha sido feito corretamente, ao pressionar o botão, o motor liga e ao pressionar o botão novamente o motor desliga. Note que é possível ouvir o ruído do funcionamento do módulo relé ao fechar o contato e acionar o motor.

19 – Próximos passos

Chegamos ao fim de nossa jornada, agora convidamos você a pensar em como melhorar estes códigos e a criar seus próprios códigos e projetos.

Visite nosso blog <http://blog.baudaeletronica.com.br/> para mais tutoriais e informações tecnológicas sobre Arduinos, Shields, componentes e muito mais do mundo da eletrônica.

