



Universidade de Brasília  
Faculdade de Tecnologia  
Mecatrônica Júnior de Brasília

# Arduino Básico

Conteúdo básico para iniciação na plataforma Arduino

**Julho de 2013**



**Apostila:**

Rodrigo de Lima Carvalho – Diretor de Pesquisa e Desenvolvimento da Mecajun  
[rodrigocarvalho@mecajun.com.br](mailto:rodrigocarvalho@mecajun.com.br)

Cristiana Miranda de Farias – Consultora de Pesquisa e Desenvolvimento da Mecajun  
[cristianamiranda@mecajun.com.br](mailto:cristianamiranda@mecajun.com.br)

Marcos da Silva Pereira – Consultor de Pesquisa e Desenvolvimento da Mecajun  
[marcospereira@mecajun.com.br](mailto:marcospereira@mecajun.com.br)

**Apoio:**

Toda a equipe da **Mecajun – Mecatrônica Júnior de Brasília**

## Arduino Básico

*Conteúdo básico para iniciação na plataforma Arduino*

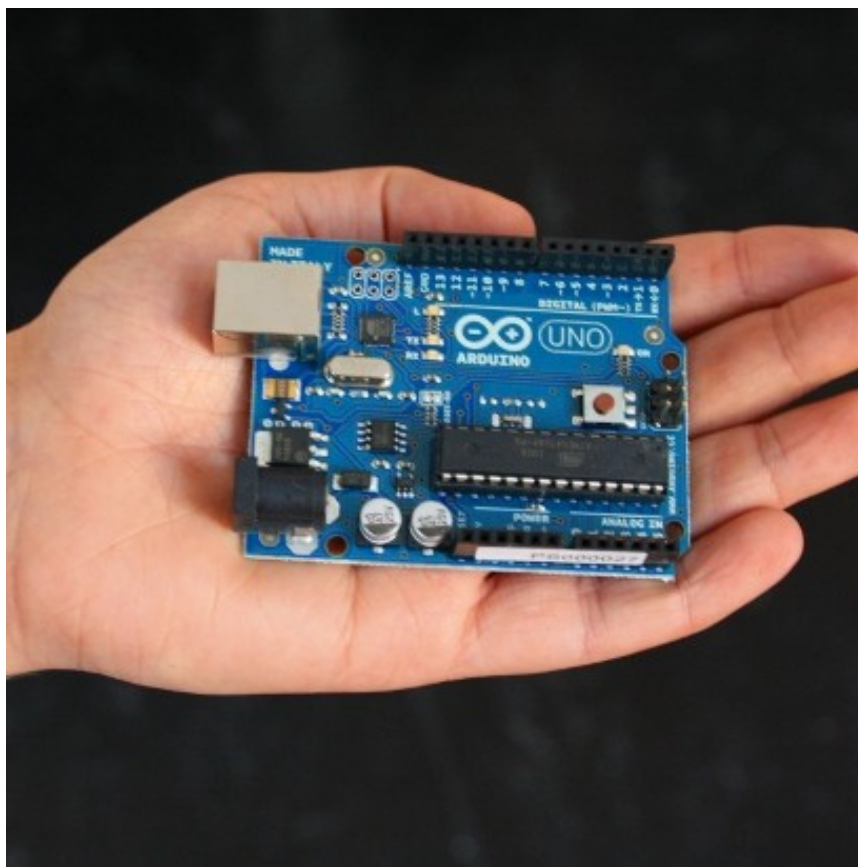
Mecajun – Mecatrônica Júnior de Brasília  
Contato: (61) 3107-5705  
[www.mecajun.com.br](http://www.mecajun.com.br)

<b>1. Introdução</b>	<b>3</b>	
1.1 Arduino – uma plataforma de hardware livre		3
1.1.1. – Hardware	5	
1.1.2. – Software	6	
1.2 Shields – os acessórios para Arduino		8
1.3 Tipos de Arduino		11
<b>2. Fundamentos de Eletrônica</b>		<b>13</b>
2.1 - Resistores e Lei de Ohm		13
2.2 - Divisor de tensão	15	
2.3 – Dispositivos de chaveamento		17
2.4 – Relé	18	
2.5 – Diodos	19	
2.6 – Transistores	22	
2.7 – Protoboard	24	
<b>3. Linguagem de programação</b>		<b>26</b>
3.1 - Funções base	26	
3.2 – Tempo	28	
3.3- Bibliotecas e #define	29	
3.4 – Variáveis		30
3.5 – Funções		32
3.6 - Operadores booleanos, comparação, incremento e decremento.		34
3.7 - Estruturas de controle de fluxo		35
3.8- Comunicação serial	38	
<b>4. Portas analógicas e digitais</b>		<b>40</b>
4.1 - Sensores digitais		40
4.1.1. Switch	41	
4.1.2. Detecção de movimento	43	
4.2- Sensores analógicos	45	
4.2.1. Sensor de luminosidade	46	
4.2.2. Sensor de distância	48	
4.2.3. Sensor de Temperatura	50	
4.3 – Atuadores	51	
4.4 - PWM – pulse-width modulation	52	
<b>5. Controle de motores</b>		<b>54</b>
5.1- Tipos de motores	54	
5.2- Ponte H – Controle de motor DC	57	
5.3- Controle de servo motores	60	
<b>6. Controle de tomadas</b>		
6.1 – Acionando 110V/220V com sinais de 5V	61	
6.2 – Implementação	62	

## 1. Introdução:

### 1.1 – Arduino – Uma plataforma de hardware livre

A plataforma Arduino foi desenvolvida para iniciantes que não têm experiência com programação ou eletrônica. Com um Arduino você é capaz de construir sistemas que controlam luz, motores, sons, que respondem de alguma forma às variações de temperatura, ao movimento, à intensidade de luz, ao toque e etc. Esses sistemas ainda podem se comunicar com um usuário através de displays LCD, pelo computador e até com celulares. Pela variedade e infinidade de possibilidades, hoje o Arduino tem sido bastante utilizado em diversas aplicações, inclusive instrumentos musicais, robótica, jogos, roupas interativas e casas inteligentes.



*Figura 1.1-1. Placa básica de um Arduino Uno*

Essa plataforma ainda é de livre acesso para qualquer pessoa. É possível acessar projetos, programas de exemplo, manuais de instrução para as mais diversas utilizações, tudo acessando o site oficial:

[www.arduino.cc](http://www.arduino.cc)

A facilidade com que se obtêm resultados interessantes com o Arduino atrai todos os tipos de usuários, desde engenheiros que já trabalham com micro controladores até aqueles que fazem por *hobby*. Essa placa é utilizada geralmente por essas pessoas para fazer protótipos de produtos definitivos.

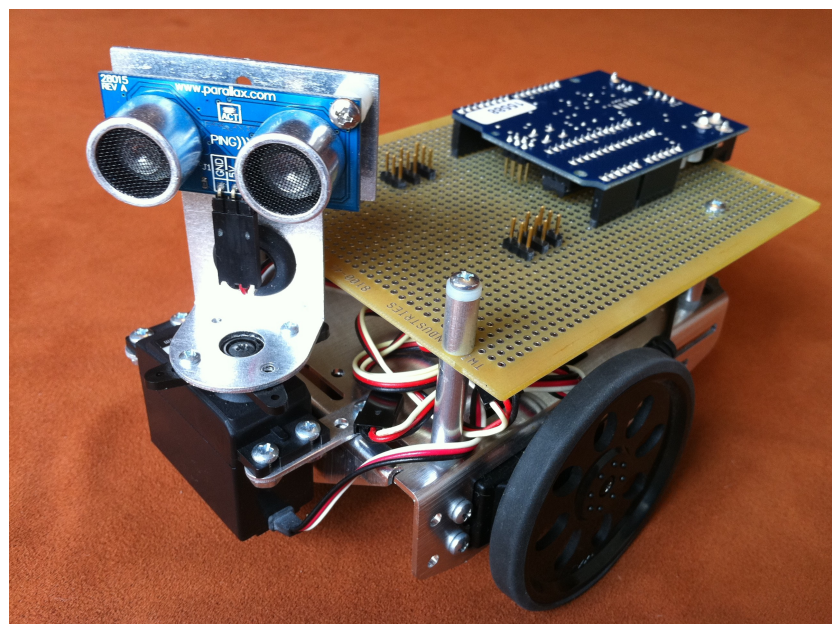
A seguir temos alguns exemplos de aplicações de Arduino para as mais diversas utilidades:



*Figura 1.1-2. Arduino LilyPad usado para confecção de roupas*



*Figura 1.1-3. Comunicação Arduino-IPhone – Automação residencial.*

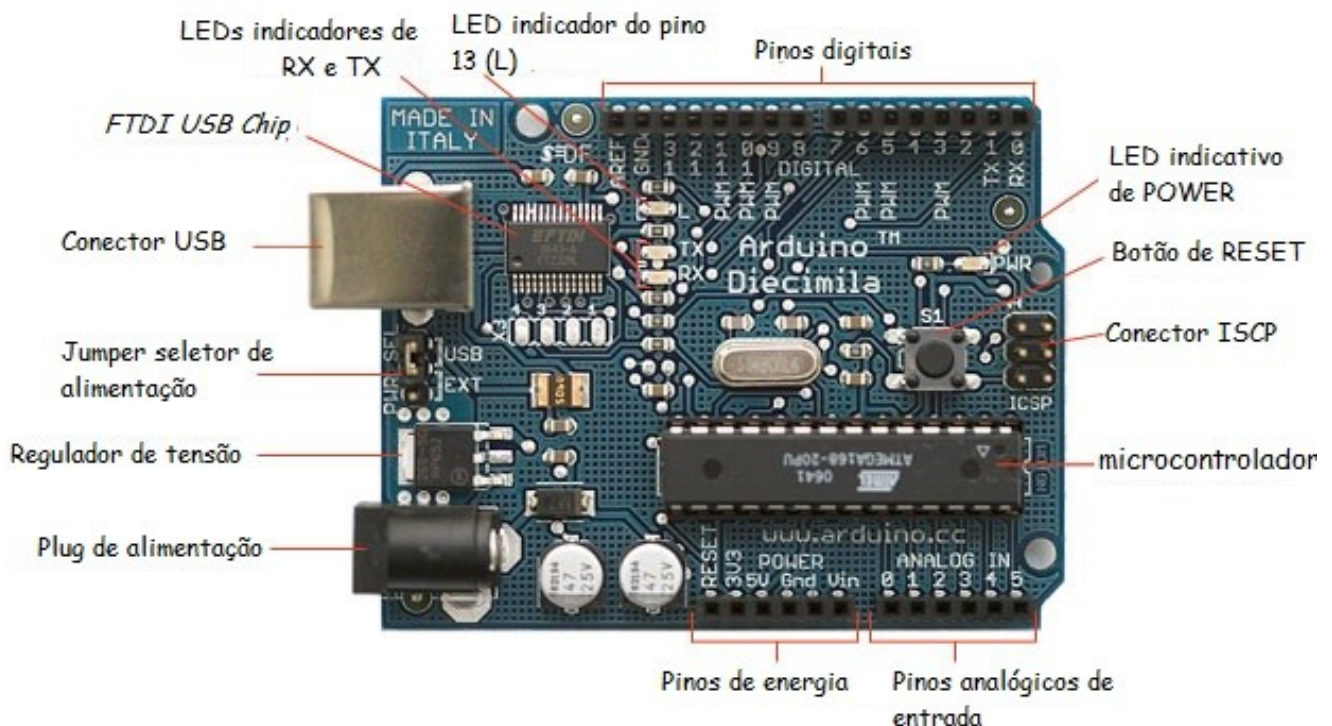


*Figura 1.1-4. Arduino na robótica.*



Nesse curso iremos dar o foco à aplicação na robótica. Mas para chegar lá, precisamos saber como funcionam o hardware e software do Arduino.

### 1.1.1 – Hardware



A seguir é apresentada uma placa básica de Arduino com seus componentes indicados.

*Figura 1.1.1-1 – Componentes de uma placa básica de Arduino Diecimila*

O Hardware do Arduino é baseado no **micro controlador** AVR. Um micro controlador é um computador em um chip, um circuito integrado (CI), que contém um processador, memória e periféricos de entrada/saída. Esse é o componente embarcado responsável pelo controle das saídas da placa, pois ele pode ser programado para funções específicas no dispositivo. Funções mais específicas e detalhes do micro controlador serão dados mais tarde quando serão úteis.

A conexão Arduino-computador é feita na maioria das vezes por USB. Para isso, as placas possuem um conector USB e um *FTDI USB chip*, um chip responsável pela adaptação Serial-USB, permitindo assim a comunicação entre os dois.

Algumas placas possuem um jumper para seleção da forma de alimentação, se é feita pela USB ou por uma fonte externa. Alimentação via USB não permite aplicações que exigem mais corrente, devido à limitação dessa fonte pelo computador.

Graças ao regulador de tensão, quando presentes nessa plataforma, é possível uma alimentação externa fornecida por uma fonte de tensão de 5V até 16V, sendo recomendado de 6V a 12V. Nos

pinos de energia, é possível obter tensões variadas, também direta da sua fonte de energia.

Os LEDs indicadores de RX e TX poderão auxiliar a identificar a comunicação do Arduino com outro dispositivo.

Arduino é *open-source*, ou seja, qualquer pessoa pode acessar a página oficial e obter todas as informações de como produzir a sua própria placa compatível. A página <http://arduino.cc/en/Main/Hardware> pode trazer o esquemático e arquivos para fabricação dos vários tipos de Arduino. Traremos com mais detalhes os principais tipos de Arduino que estão no mercado na seção 1.3.

Um importante aspecto é a padronização da maneira como os conectores são expostos, fazendo-se assim a possibilidade de adicionar módulos expansivos para aumentar as possibilidades de aplicações do Arduino, os quais chamamos de *Shields*. Veremos alguns exemplos e mais detalhes na seção 1.2.

### 1.1.2 – **Software**

O Arduino é famoso pelo seu hardware, ou seja, a sua placa. Mas também é composto por um software que facilita a vida dos usuários dessa plataforma.

O **Arduino IDE** é um aplicativo multiplataforma, ou seja, que roda em várias plataformas. Desenvolvido em Java e derivado dos projetos Processing e Wiring, foi especialmente projetado para pessoas que não são familiarizadas com o desenvolvimento de softwares.

Para instalar o software em seu computador é bastante simples. Acesse o link fornecido pelo site oficial, e faça o download da última versão de acordo com o seu sistema operacional (Windows, MAC OS X ou Linux):

<http://www.arduino.cc/en/Main/Software>

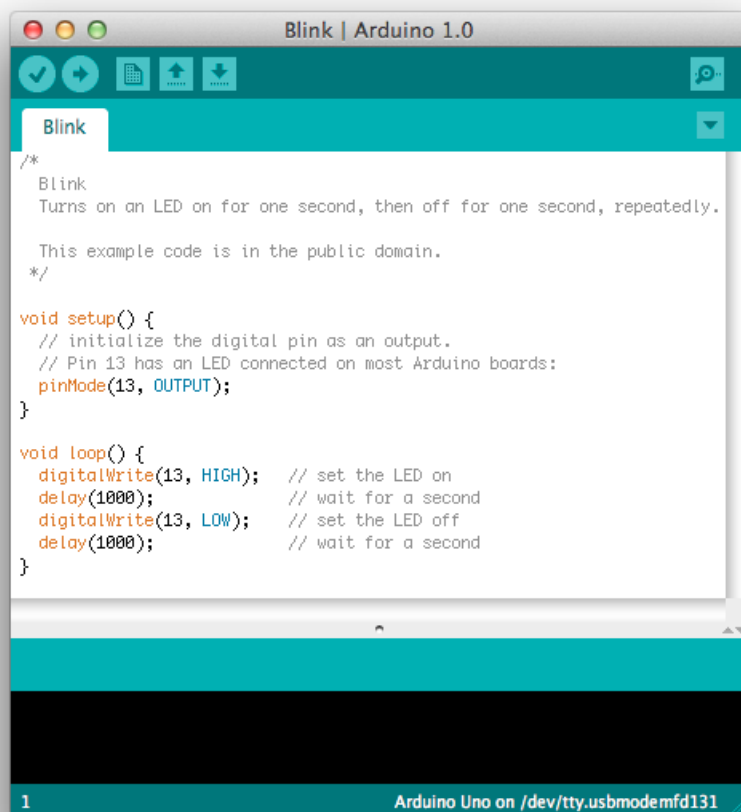


Figura 1.1.2-1 – Arduino IDE 1.0 com programa exemplo Blink

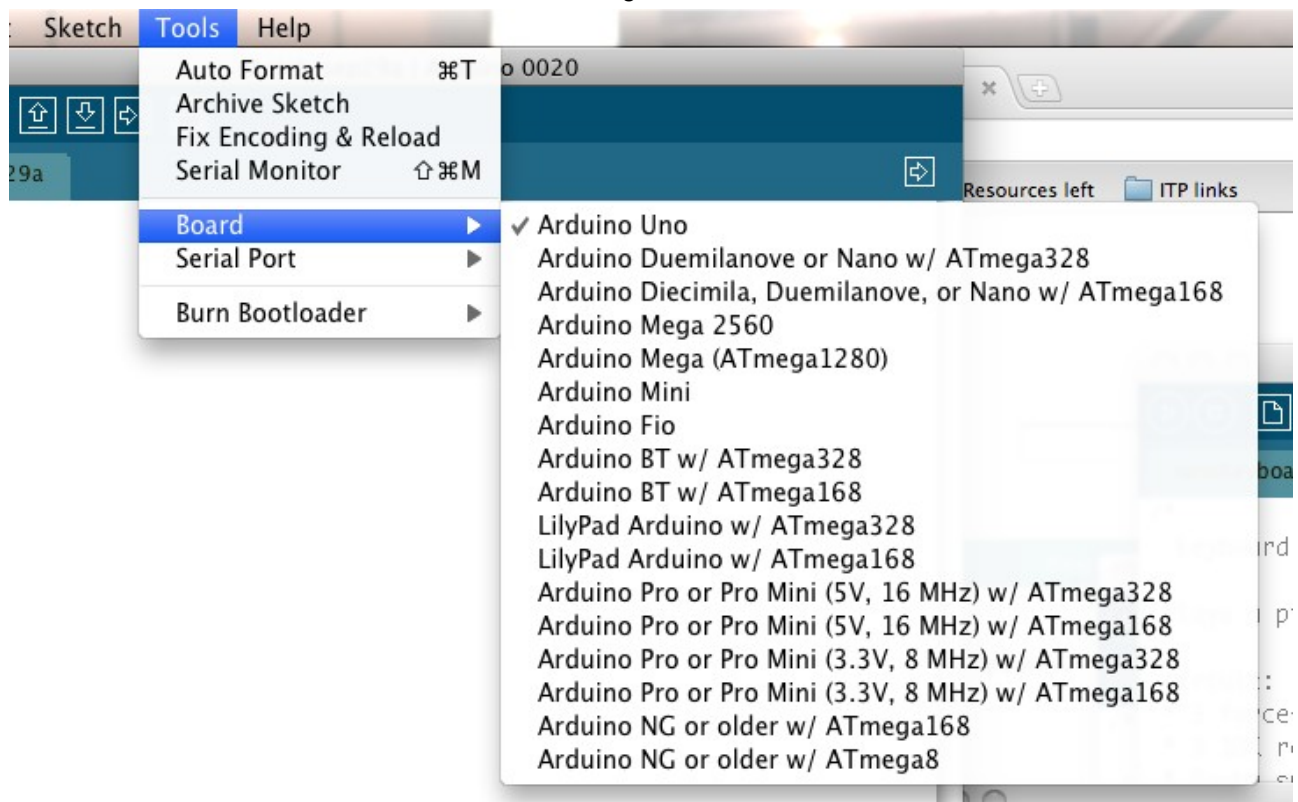
Caso haja dúvidas de instalação ou problemas com driver, por favor, acesse o link <http://arduino.cc/em/Guide/HomePage>, e siga os passos indicados para o seu caso.

O Arduino IDE permite o desenvolvimento da programação, compilação e upload desse programa para o seu Arduino. Também possui uma janela de comunicação serial (Serial Monitor). A utilização deste último recurso será vista com detalhes mais tarde.



Ao abrir o aplicativo, selecione o tipo de placa Arduino que está conectada e em seguida a porta serial que ela está usando no menu *Tools*, como indicado na figura 1.1.2-2.

Figura 1.1.2-2 – Menu Tools



Algumas vezes não sabemos em qual das portas seriais mostradas pelo software o Arduino está conectado. Para tirar essa dúvida, veja as portas que estão sendo mostradas, retire o Arduino da USB, feche e abra novamente o menu e veja qual porta não aparece mais. Conecte novamente e selecione essa porta.

Para passar o seu programa do Arduino IDE para a sua placa, basta clicar no botão *Upload*. Verifique os LEDs indicadores de RX e TX indicando a comunicação. Desta forma a sua programação ficará salva na memória não volátil de seu microcontrolador, isso significa que se o Arduino for retirado da fonte de alimentação e religado novamente, essa memória ainda conterá a programação gravada.

Outras funções do software serão vistas à medida que formos precisando usá-las. Vamos ver agora alguns exemplos de Shields para entendermos como usar o Arduino em algumas funções específicas. Para testar se a sua placa está funcionando, transfira o programa exemplo *Blink* que está na barra *File>Examples>Basics* e verifique se o LED indicativo do pino 13 (L) pisca em intervalos de um segundo.

## 1.2 - Shields – os acessórios para Arduino

Shields são placas extensivas ao Arduino que podem ser acopladas graças à disposição padronizada dos pinos de entrada/saída dessa plataforma. Com eles podemos estender as capacidades do Arduino de forma simples.

Segue abaixo uma lista dos Shields Oficiais para Arduino

### **Shields de Arduino oficiais**



**Arduino WiFi Shield**



**Arduino Wireless Proto Shield**



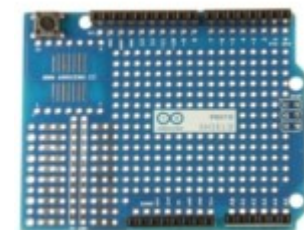
**Arduino Ethernet Shield**



**Arduino Motor Shield**



**Arduino Wireless SD Shield**



**Arduino Proto Shield**

Também encontramos alguns inventados por usuários da plataforma. Todos eles são open-source e podemos ter acesso às informações técnicas e geralmente temos bibliotecas prontas que facilitam a sua utilização. Iremos tratar de adição de bibliotecas mais tarde. Agora mostraremos alguns exemplos de Shields e as suas utilidades. Alguns deles exigem um nível maior de programação, mas todos eles têm suas documentações acessíveis e abertas aos usuários.

### **Ethernet Shield:**

Shield com uma entrada para o cabo RJ45 (cabo de rede), possibilitando o Arduino interagir em uma rede pessoal ou até mesmo com a internet, possibilitando assim uma vasta gama de atividades, como enviar ou receber informações do seu Arduino remotamente.



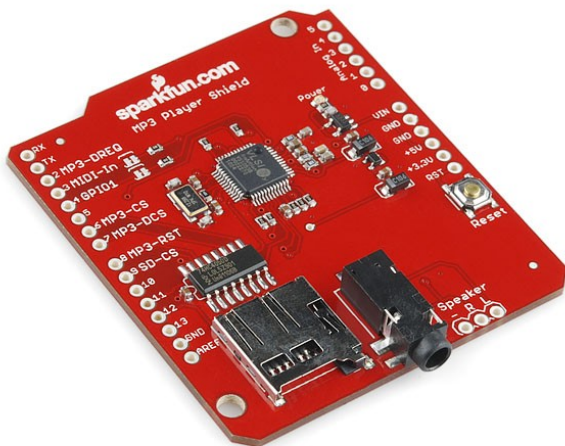
*Figura 1.2-1 – Shield Ethernet*

Todas as informações, inclusive esquemático da placa e bibliotecas para programação com este shield se encontram no link:

<http://arduino.cc/en/Main/ArduinoEthernetShield>

### **MP3 Player shield:**

Shield usado para tocar arquivos mp3 inseridos em um cartão microSD e utiliza uma saída de áudio para autofalantes. É possível acessar todas as informações, guias de inicialização rápida e códigos de exemplo no próprio site do fornecedor oficial do produto:



<https://www.sparkfun.com/products/10628>

Figura 1.2-2 – Shield MP3 Player sparkfun

**Shield de teclado e LCD:**

Shield usado para interface com usuário. Como ele fica fácil mostrar informações pelo display LCD e ainda obter controle por um teclado.



Figura 1.2-3 – LCD + Teclado Shield

O shield da figura 1.2-3 utiliza uma tela de LCD compatível com HD44780, e pode ser acessado pela biblioteca Arduino LCD4Bit, encontrada no site oficial do Arduino. Para saber o esquemático, link para biblioteca utilizada para programação, dados técnicos, basta acessar o site do fornecedor:

<http://huinfinito.com.br/shields-placas-extensveis/562-shield-teclado-com-lcd-16x02-az-br.html>



### 1.3 – Tipos de Arduino

Existem vários tipos de Arduinos, com diferentes utilidades, tamanhos e tipos de conexão. Algumas placas oficiais de Arduino com suas especificações são encontradas no site, no link <http://arduino.cc/en/Main/Hardware>. Abaixo, temos uma lista dos Arduinos oficiais:

:

#### Placas de Arduino oficiais



Arduino Uno



Arduino Pro



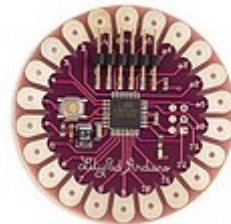
Arduino Leonardo



Arduino Mega 2560



Arduino Nano



Arduino LilyPad



Arduino Mega ADK



Arduino Mini



Arduino Fio



Arduino Ethernet



Arduino Pro



Arduino BT

Vemos que alguns Arduinos já vêm com funções específicas, como o BT (bluetooth) e Ethernet. Alguns são muito pequenos para aplicações especiais como o Mini. O Lilypad, com formato especial, serve para confecções de roupas com linhas especiais condutoras de energia. Outros têm um número maior de portas para utilizar um número maior de atuadores e sensores, como o Mega. As diferenças também podem vir no protocolo de comunicação.

Alguns usuários também fabricam outros tipos de Arduino com outras funções. No Brasil temos os exemplos do Severino (figura 1.3-1) e a programme (figura 1.3-2).

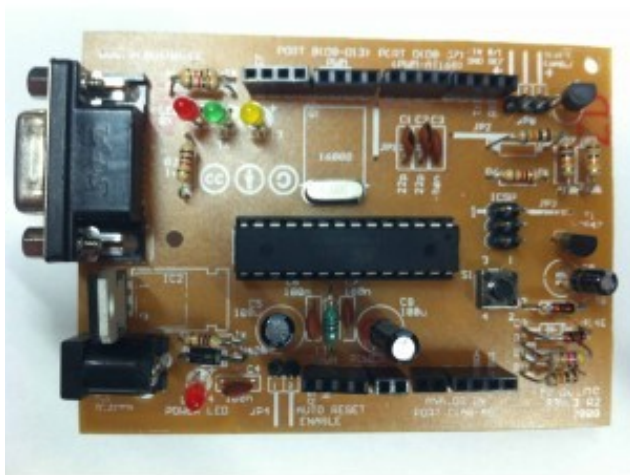


Figura 1.3-1 – Severino. Comunicação serial

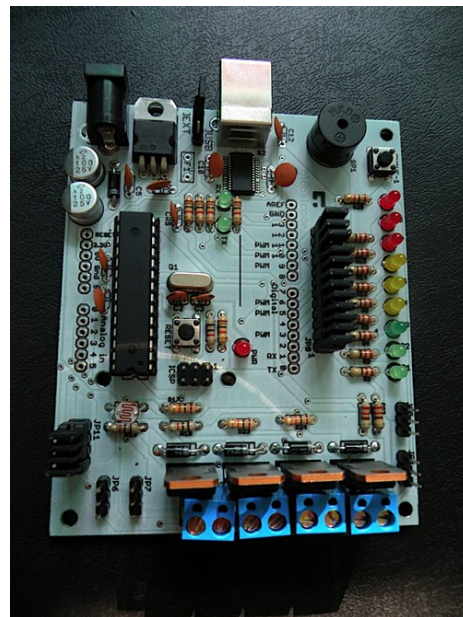


Figura 1.3-2 – Program-me, da Global Code. Placa usada interativa com leds e sensores embarcados.

O Arduino básico e mais comum é o Arduino Uno. Versões anteriores, como Duemilanove, são muito parecidas. O importante é que todas essas placas utilizam a mesma linguagem de programação e são open-source.

## 2 Fundamentos de Eletrônica

### 2.1 Lei de Ohm, Resistores e Leis de Kirchhoff

A Lei de Ohm estabelece que a tensão (**V**) entre os terminais de uma resistência é diretamente proporcional à corrente (**i**) que flui através dela. A



resistência (**R**) é a constante de proporcionalidade que relaciona a tensão e a corrente. O seu valor é medido em **Ohms**. Os componentes de circuito que possuem característica elétrica puramente resistiva são chamados de resistor. A relação matemática da lei é descrita pela equação 2.1

$$V = R \times i \text{ (2.1)}$$

Os resistores são componentes elétricos que limitam a passagem de corrente elétrica em um circuito. A tensão aplicada em um resistor limita a corrente proporcionalmente ao valor da sua resistência. A característica elétrica do resistor o torna um elemento de grande importância dentro de um circuito elétrico-eletrônico pois, ele previne a passagem de correntes altas que podem danificar outros componentes eletrônicos como, por exemplo, diodos emissores de luz (LEDs) que serão abordados na seção 2.3.

As cores presentes no corpo do resistor indicam o valor da sua resistência. A figura 2.1-1 apresenta um exemplo típico de um resistor e o seu símbolo em um circuito elétrico.



**Figura 2.1-1** – Resistor e sua simbologia em circuitos

O valor de um resistor é obtido a partir do código de cores apresentados na tabela 2.1-1. A leitura do valor da resistência é feita de acordo com a seguinte convenção

$$AB \times 10^C \pm D \text{ (2.2)}$$

onde **A** e **B** são os dois primeiros dígitos, correspondentes as duas primeiras faixas de cores. O valor de **C** é obtido a partir da terceira faixa e corresponde à potência de 10 que multiplica os dois primeiros dígitos. A letra **D** é a última faixa e corresponde à tolerância do elemento. Em alguns casos, a terceira

faixa é das cores Ouro ou Prata indicando multiplicadores 0,1 e 0,01, respectivamente.

Cor	Valor do Dígito
Preto	0
Marrom	1
Vermelho	2
Laranja	3
Amarelo	4
Verde	5
Azul	6
Violeta	7
Cinza	8
Branco	9
Ouro	Multiplicador 0,1 ou +- 5% de tolerância
Prata	Multiplicador 0,01 ou +- 10% de tolerância

**Tabela 2.1-1** – Código de cores do resistor

O cálculo do valor da resistência de um resistor com base no código de cores pode ser exemplificado observando as cores do resistor apresentado na figura 2.1-1. A primeira faixa de cor é marrom, logo, de acordo com a tabela 2.1-1 temos que o valor do primeiro dígito **A** na fórmula 2.2 é **1**. O segundo dígito é equivalente ao valor da faixa de cor verde, portanto **B** é **5**. A terceira faixa de cor laranja indica que o valor **C** da potência de 10 é **3**. Por fim, a quarta faixa dourada mostra que o valor da tolerância é de 5%. Consequentemente, o valor do resistor é **15000 Ohms** ou mais comumente escrito como **15 kOhm**.

A presente apostila tem como objetivo focar em aspectos práticos de Arduino, apesar disso, a seção 2.2 exige alguns conceitos teóricos extras de elétrica para ser apresentada. Eles serão abordados superficialmente de modo que seja possível desenvolver o raciocínio de outras seções. Caso você tenha interesse de aplica-los em outras situações, recomenda-se ao leitor procurar um texto com foco exclusivo na teoria de circuitos elétricos e eletrônicos.

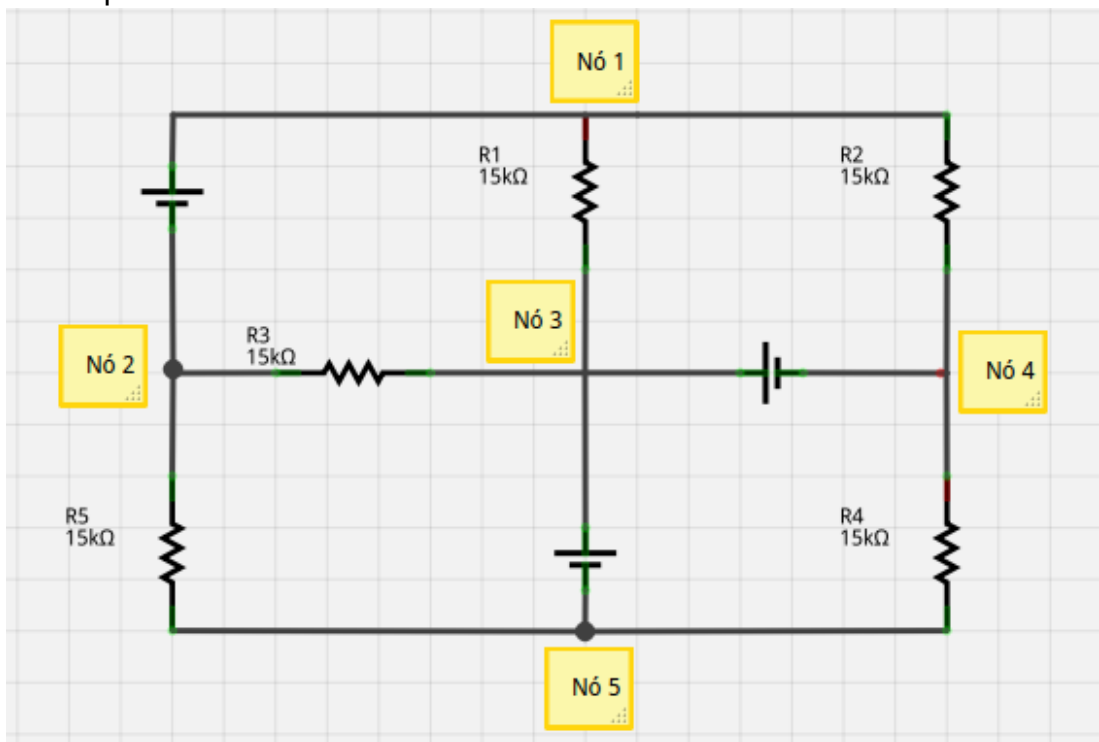
A análise de um circuito elétrico pode ser facilitada com a utilização de alguns termos. Eles são: (i) Nó, (ii) Laço e (iii) Ramo. Os termos são apresentados considerando que em um circuito elétrico os componentes são interconectados por fios e que a resistência dos fios é pequena em relação aos componentes. Portanto, a resistência do fio pode ser desconsiderada caso o circuito não exija alta precisão.

(i) Nó: Ele é um ponto de conexão de dois ou mais componentes do circuito. É importante perceber que ele pode estar posicionado em qualquer

posição do fio que conecta dois componentes. Os pontos determinados por nós 1, 2, 3 e 4 na figura 2.1-2 são exemplos de nós.

(ii) **Laço:** Ele é qualquer caminho fechado do circuito no qual nenhum nó seja encontrado mais de uma vez. Ou seja, ao percorrer o laço, só se passa por cada nó do circuito uma única vez. Na figura 2.1-2 temos um exemplo de laço ao percorrer o circuito do nó 4 passando por uma fonte a esquerda do nó, pelo nó 3, subindo pelo R1, passando pelo nó 1, voltando pelo R2 novamente ao Nó 4.

(iii) **Ramo:** São trechos do circuito que contém apenas um único componente e são delimitados por nós. Na figura 2.1-2 temos vários exemplos de ramos, entre eles estão o ramo entre o nó 1 e o nó 3 que contém R1 e o ramo entre o nó 2 e o nó 3 que contém uma fonte, por exemplo.



**Figura 2.1.2** – Apresenta exemplo de circuito para indicar exemplo de Nós, Laços e Ramos.

A partir dos conceitos apresentados anteriormente podemos definir as Lei de Kirchhoff das Correntes (LKC) e a Lei de Kirchhoff das Tensões (LTK). A primeira afirma que a soma das correntes entram em um nó é igual a soma das correntes que saem, ou de uma outra forma, a soma das correntes em um nó é nula (zero). A segunda estabelece que a soma das tensões ao longo de qualquer laço é nula. Para facilitar a aplicação de ambas as leis, costuma-se adotar a seguinte convenção (Convenção Passiva dos Sinais):

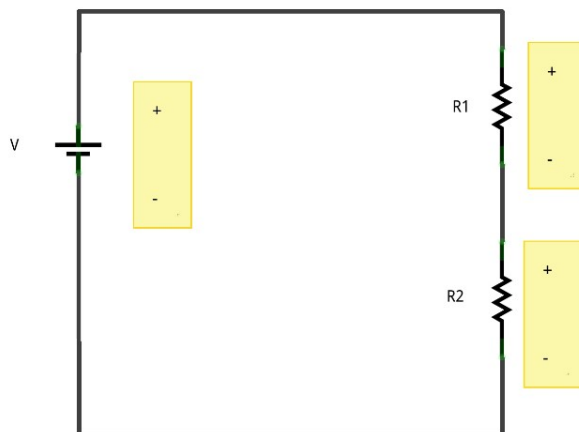
(i) Se o valor da corrente é positivo, assume-se que ela está percorrendo o circuito no sentido em que foi indicado. Caso seja negativa, ela está percorrendo o circuito no sentido contrário.

(ii) A tensão em um determinado componente do circuito é considerada positiva se a corrente estiver entrando no terminal positivo do componente.

Caso contrário ela é negativa.

## 2.2 Divisor de Tensão

As leis apresentadas anteriormente vão nos permitir analisar um circuito simples e obter uma expressão de extrema importância na construção de circuitos elétricos e eletrônicos. O circuito estudado será um único caminho ou laço de componentes. Ele contém apenas uma fonte de tensão **V** e dois resistores **R1** e **R2**.



**Figura**

**2.2-1** – Circuito analisado para obter expressão do divisor de tensão

A aplicação da LKC a cada nó(A, B e C) do circuito mostra que a mesma corrente passa através de todos os componentes. Logo, diz-se que tais componentes estão associados em série. Eles estão sujeitos a uma mesma corrente.

O circuito será analisado considerando que a corrente **i** flui no sentido horário e aplicando a polaridade (sinais) indicada na figura 2.2-1. Caso essa hipótese esteja correta, a solução das equações fornecerá um valor positivo para a corrente. Se o valor da corrente for negativo, ela está percorrendo o circuito no sentido oposto. Essa convenção respeita a que foi apresentada no final da seção 2.1.

O primeiro passo da análise do circuito é aplicar a LTK obtendo a seguinte expressão:

$$-V + V_{r1} + V_{r2} = 0 \quad (2.1)$$

ou

$$V = V_{r1} + V_{r2} \quad (2.2)$$

Pela Lei de Ohm definida no início da seção 2.1, temos

$$V_{r1} = R1 \times i \quad (2.3)$$

$$V_{r2} = R_2 \times i \quad (2.4)$$

Portanto,

$$V = (R_1 \times i) + (R_2 \times i) \quad (2.5)$$

Isolando-se a variável  $i$  temos

$$i = V / (R_1 + R_2) \quad (2.6)$$

A equação obtida para a corrente  $i$  nos permite aplicar a Lei de Ohm para determinar a tensão referente a cada resistor, ou seja,

$$V_{r1} = R_1 \times i$$

$$V_{r1} = R_1 \times [V / (R_1 + R_2)]$$

$$V_{r1} = [R_1 / (R_1 + R_2)] \times V \quad (2.7)$$

O mesmo raciocínio pode ser aplicado para obter a tensão em  $R_2$ ,

$$V_{r2} = [R_2 / (R_1 + R_2)] \times V \quad (2.8)$$

As equações 3.7 e 3.8 mostram a forma como a tensão se divide entre dois resistores em série. Elas são de grande relevância, pois elas descrevem a operação conhecida como *divisor de tensão*. De uma forma mais simples, ela mostra como a fonte de tensão  $V$  é dividida entre os resistores  $R_1$  e  $R_2$ . Portanto, a partir do interesse na tensão entre os terminais do resistor  $R_1$  ou  $R_2$ , pode-se obter a tensão daquele trecho facilmente.

## 2.3 Dispositivos de Chaveamento

Os dispositivos de chaveamento são componentes que permitem a manipulação e realização de processos em um circuito elétrico. As operações lógicas se utilizam diretamente de tais elementos pois elas exigem que determinada ação seja feita apenas se rotinas anteriores tiverem sido executadas. Os sistemas de hardware que controlam os computadores atualmente são baseados na lógica implementada por dispositivos de chaveamento.

As chaves mecânicas, como, por exemplo, as da figura 2.2-1, correspondem às representações mais simples de dispositivos de chaveamento. Elas são as chaves liga-desliga que vemos em aparelhos eletro-eletrônicos como, por exemplo, computadores, brinquedos e eletrodomésticos. Além disso, uma aplicação de chaves relativamente simples que vemos diariamente são os interruptores de luz.



ia

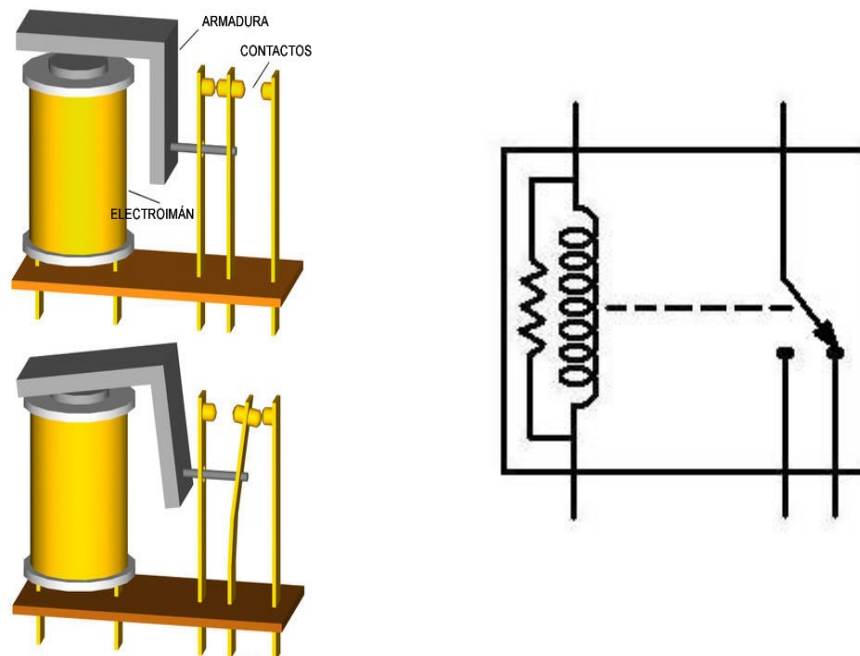


**Figura 2.2-1 – Chave mecânica e simbologia em circuito**

Os interruptores de luz são chaves mecânicas que suportam altas tensões e correntes. Um problema típico, que faz o uso deles em conjunto com lógica, é quando desejamos acender a luz ao entrar no quarto, mas desejamos apaga-la estando deitado na cama sem ter que levantar para desligar o interruptor. A resolução desse problema envolve a implementação de uma lógica a partir de dispositivos de chaveamento.

## 2.4 Relés

A operação de chaveamento executada pelos relés é eletromagnética. Ele consiste de uma bobina - enrolamento de fio de cobre - e uma chave mecânica composta de material que sofra ação de força eletromagnética, como mostra a figura 2.4-1. Esta pode ser configurada como Normalmente Aberta (N.A.) ou Normalmente Fechada (N.F.). A primeira significa que sempre que o relé não for ativado, a chave está sempre “desligada” e interrompe a passagem de corrente, desligando o circuito, enquanto a segunda indica que a chave está sempre “ligada” e permite a passagem de corrente, ligando dois pontos do circuito.





**Figura 2.4-1** – Esquema didático de relé e possível simbologia e circuito.

Os relés são ativados com a passagem de corrente elétrica (fluxo de carga) pela bobina. O fluxo de cargas na bobina induz um campo eletromagnético. Dessa forma ela passa a agir como um eletroímã e atrai ou repele a chave mecânica fechando ou abrindo o circuito, de acordo com a configuração da chave. É importante observar também que o circuito utilizado para ativar o relé e o circuito fechado por ele são mantidos independentes. Não há interação de corrente entre ambos.

As chaves mecânicas e eletromecânicas podem ser utilizadas na implementação de circuitos lógicos. Apesar disso, devido ao acionamento mecânico, elas produzem circuitos lentos e com pouca precisão. Portanto não costumam ser utilizadas em sistemas digitais, como por exemplo, os encontrados dentro do computador.

Os relés são aplicados, atualmente, para proteção de circuitos eletro e microeletrônicos. Eles fazem a ligação entre circuitos elétricos de baixa tensão com circuitos de alta tensão. Os sistemas eletrônicos costumam realizar tarefas de recebimento, processamento de dados (temperatura, luminosidade, umidade, pressão, etc) e envio de tarefas para atuadores (motores, aquecedores, etc).

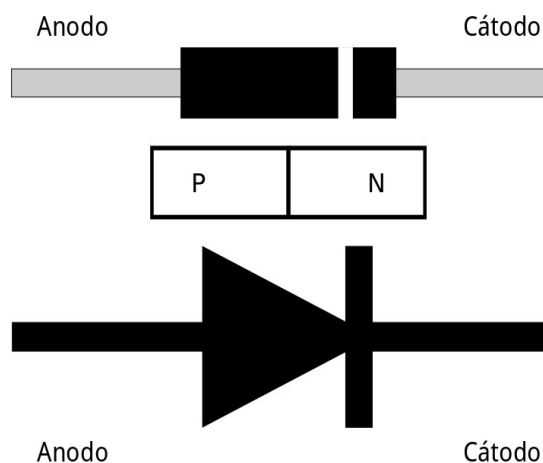
Os atuadores de tarefas em grande escala costumam trabalhar sobre alta tensão e conduzir valores elevados de corrente. Tais condições de operação podem danificar o funcionamento de circuitos eletrônicos de baixa tensão. Portanto, são utilizados relés, que podem ser ativados por circuitos de baixa tensão, para ativar circuitos de alta tensão.

## **2.5 Diodos**

O diodo é um componente utilizado para na construção de dispositivos de chaveamento mais precisos. Ele utiliza semicondutores como Germânio e Silício em sua construção. O seu princípio de funcionamento é baseado nos fenômenos físicos da junção P-N.

O material semicondutor do tipo N possui elétrons extras. Logo, é carregado negativamente e elétrons se movem da área carregada negativamente para uma área carregada positivamente. Os semicondutores com ausência de elétrons ou presença de buracos estão carregados positivamente. Portanto são chamados como tipo P e neles os elétrons podem se mover entre os buracos. Consequentemente, os buracos parecem se mover das áreas positivas para as áreas negativas. Os diodos são

compostos pela junção entre N e P e essa configuração permite passagem de corrente apenas em um sentido.

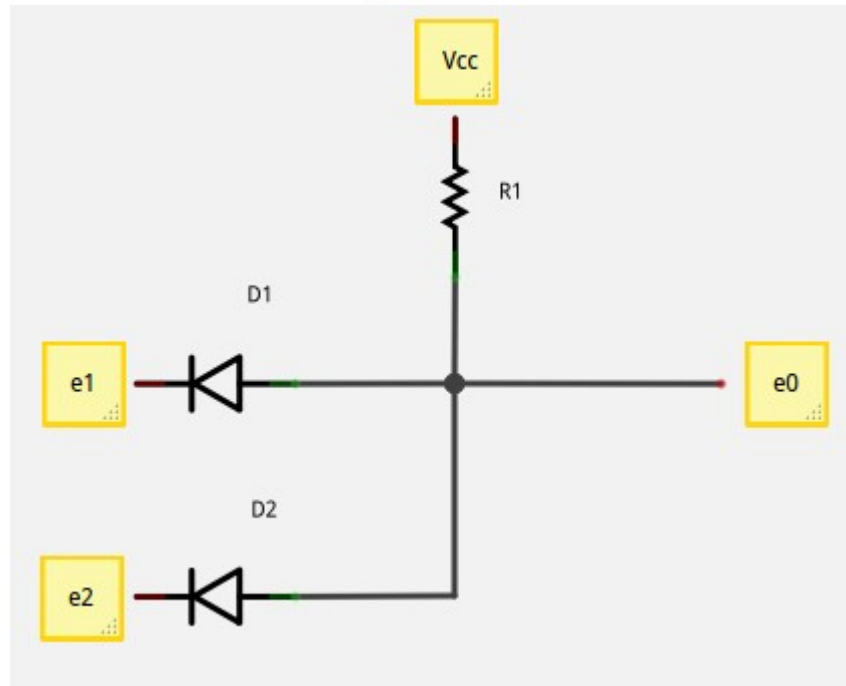


**Figura**

**2.5-1** – Esquemático de diodo real (parte superior), orientação da junção p-n e símbolo de diodo em um circuito (parte inferior)

A estrutura do componente é composta pelo anodo e pelo catodo. A orientação de acordo com a simbologia e componente real pode ser vista na figura 2.5-1. O diodo oferece resistência baixa para as correntes fluindo do anodo para o catodo e resistência alta no sentido oposto. Essa característica permite a criação de circuitos lógicos e, também, a proteção contra correntes de reversa. Além disso, em comparação com os relés, as respostas são rápidas e precisas.

Os tipos de operações lógicas que ele permite realizar são inúmeras e como exemplo vamos analisar o circuito da figura 2.5-2. A simbologia e0, e1 e e2 representam, respectivamente, a saída do circuito e0, a entrada 1 e a entrada 2 do circuito. O termo Vcc é uma tensão proveniente de uma fonte ou de outro trecho do circuito. As aplicações dadas a esse circuito se resumem a tabela 2.5-1.



**Figura 2.5-2** – Circuito lógico (AND) implementado com diodos

**Tabela 2.5-1** – Tabela verdade do circuito da figura 2.5-1

e1	e2	e0
0	0	0
0	Vcc	0
Vcc	0	0
Vcc	Vcc	Vcc

O funcionamento do circuito com diodos da figura 2.5-2 pode ser entendido com o auxílio da tabela 2.5-1, o conceito de corrente elétrica como fluxo ordenado de partículas portadoras de carga (nas aplicações do dia a dia, elétrons) e o princípio básico de que cargas elétricas fluem de pontos de maior potencial elétrico para menor potencial elétrico. Primeiramente, vamos considerar a primeira linha da tabela onde as duas entradas do circuito são mantidas em **0 V**. Nesse caso, de acordo com o princípio do fluxo de cargas, os elétrons vão caminhar em direção aos diodos **D1** e **D2** já que o potencial aplicado em ambas entradas é **0 V**, inferior a **Vcc**. Além disso não fluíram de volta devido a propriedade dos diodos. Logo, a saída **e0** do circuito vai possuir valor de saída de **0 V**.

As linha 2 e a linha 3 da tabela 2.5-1 exemplificam os casos em que for aplicada uma tensão **Vcc** em apenas uma das entradas **e1** ou **e2**. Em ambos os casos, os elétrons vão continuar fluindo do **Vcc** para a entrada que estiver em **0V**. Portanto, a saída **e0** continuará em **0 V**.

O último caso da tabela 2.5-1 mostra o caso em que são aplicadas a tensão de **Vcc** em **e1** e **e2**. De acordo com o princípio do fluxo de cargas,

os elétrons não caminham em direção aos diodos pois ambos estarão conduzindo com uma tensão de **Vcc**. Consequentemente, os elétrons fluirão para a saída **e0**, gerando uma tensão **Vcc** na saída do circuito.

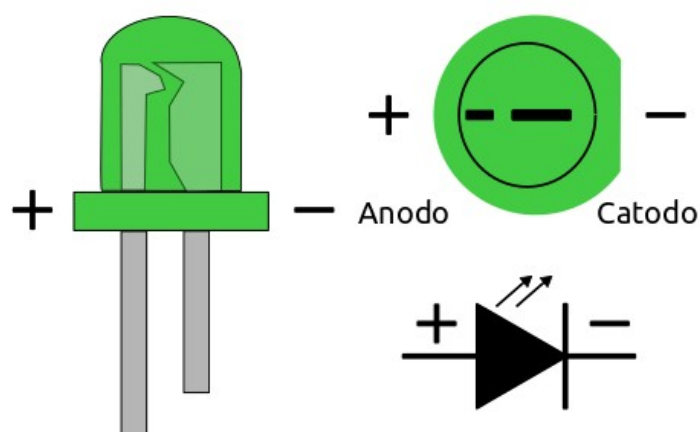
A figura 2.5-1 apresenta a construção da porta lógica E (AND) com diodos e a tabela 2.5-1 mostra seu funcionamento. O desenvolvimento apresentado nos últimos 3 parágrafos permite entender a aplicação do circuito e conhecer uma das aplicações dos diodos para realizar decisões lógicas. A operação lógica AND é utilizada em situações que exigem duas condições para que algo aconteça, como por exemplo, na compra de uma lata de refrigerante em uma máquina de latinhas.

A máquina é elaborada de tal forma que a lata só é entregue ao cliente caso ele aperte o botão da opção que deseja e insira o dinheiro. Caso ele apenas aperte o botão ou insira o dinheiro, a máquina não entrega o produto. Nesse exemplo, cada uma das condições pode ser indicada como verdadeira com uma tensão **Vcc** e fornecer a saída **Vcc** para outra parte do circuito que libere as latinhas quando essa tensão for aplicada.

### Diodos Emissores de Luz (LED)

A luz é composta por vários pacotes de partículas chamados fótons. Eles são liberados como resultado do movimento de elétrons. No átomo, elétrons se movem em orbitas ao redor do núcleo e cada uma delas exige que o elétron possua uma energia diferente. Quando o elétron se desloca de uma orbita de maior energia para uma de menor energia, ele libera energia na forma de um fóton. Este oscila numa determinada frequência que influencia na nossa capacidade de visualização.

Como vimos, os elétrons que se movem através do diodo se deslocam através dos buracos das seções P. Cada um dos buracos possui nível de energia diferente. Portanto, a medida que eles se deslocam são liberados fótons com determinadas frequências de oscilação. Os LEDs são diodos feitos de materiais que permitem a emissão de fótons em frequências visíveis ao olho humano.



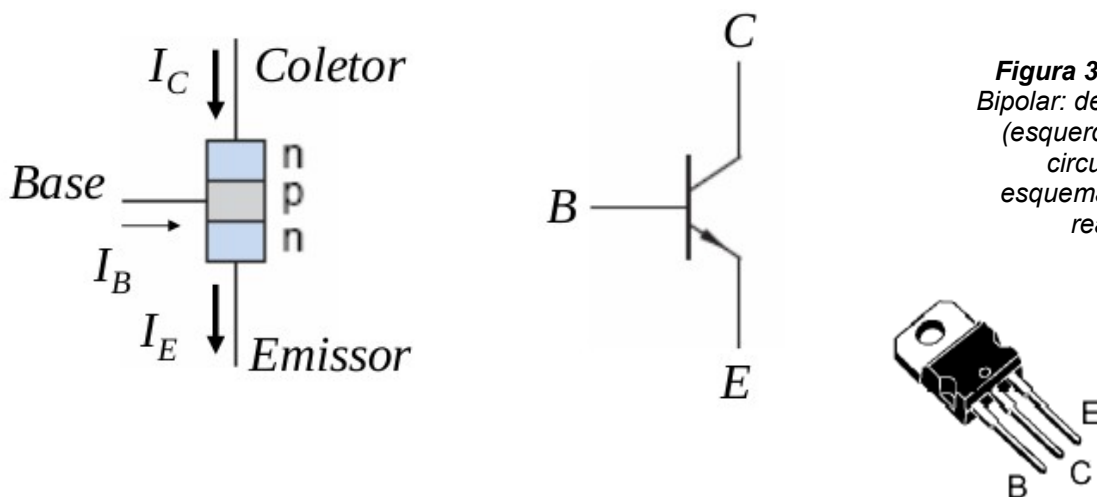
**Figura 2.5-3 – LED**

Os LEDs têm várias vantagens sobre as lâmpadas incandescentes convencionais. Ele não possui o filamento, geralmente de tungstênio, que se queima então seu período de funcionamento é longo. O revestimento de plástico que protege a junção PN os torna muito mais duráveis e cabem com maior facilidade nos circuitos eletrônicos atuais. Apesar disso, a principal vantagem é a eficiência pois, ao contrário das lâmpadas incandescentes, o seu processo de produção de luz não envolve produção de muito calor. Logo, a maior parte da energia é convertida em iluminação.

## 2.6 Transistores

As chaves eletromecânicas e as válvulas eletrônicas eram os componentes utilizados antigamente para controlar a passagem de corrente e aplicação de tensão entre pontos de um circuito. Apesar disso, as chaves eram devagar e a válvula necessitava ser aquecida para ser utilizada. A empresa norte americana Bell Telephone System precisava de algo melhor que válvulas eletrônicas para manter seus sistemas de comunicação funcionando. Ela criou um time de pesquisa para achar uma solução e em por volta de dois anos anunciou a criação dos Transistores.

Os transistores mais utilizados atualmente são compostos por junções PN em uma única pastilha de semicondutor. Ele é composto por uma pastilha do tipo P entre duas do tipo N. A estrutura possui a base ligada na junção do tipo P, o coletor e o emissor ligados nas junções do tipo N, como visível na figura 2.6-.1. É importante notar que dois diodos com anodos em comum não se comportam como uma única pastilha de semicondutor com duas junções PN. Isso é facilmente visualizável, pois o diodo impede que uma corrente passe do catodo para o anodo.

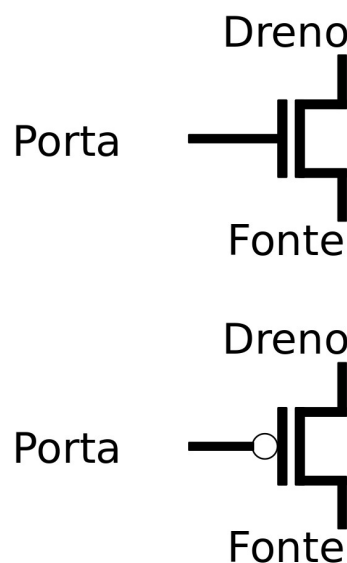


**Figura 3.6.1 – Transistor Bipolar:** desenho explicativo (esquerda), símbolo em circuito (centro), esquemático do modelo real (direita)

A corrente de base  **$I_B$**  controla o fluxo de corrente entre o coletor e o emissor. À medida que o valor de corrente aplicado à base aumenta até um determinado valor de saturação, a diferença de tensão entre a base e o coletor aumenta. Logo, a passagem de corrente entre coletor e emissor cresce. Essa propriedade permite assim como as chaves, as válvulas, os relés e os diodos controlar a passagem de corrente entre pontos de um

circuito.

Os diferentes tipos de situações de aplicação utilizam, principalmente, dois tipos de transistores: os transistores bipolares, apresentado na figura 3.6.1 e os transistores MOSFET (*Metal-Oxide-Semiconductor Field Effect Transistor*) que podem ser visualizados na figura 2.6-2. Os primeiros são utilizados na família lógica TTL. O segundo nas famílias lógicas PMOS, NMOS e o seu modo de utilização é semelhante ao bipolar. No NMOS, a aplicação de uma corrente na porta permite a passagem de corrente entre o dreno e a fonte enquanto no PMOS a passagem de corrente é bloqueada. As características elétricas de cada tipo de transistor influenciam diretamente nas suas condições de operação. Apesar disso, na presente apostila trataremos superficialmente as que podem causar mais problemas. Para detalhes mais aprofundados consulte um livro sobre sistemas digitais.



**Figura 2.6-2**

– Transistor NMOS(superior) e PMOS(inferior)

As construções de circuitos envolvendo transistores bipolares e circuitos TTL pode ter ao final do desenvolvimento entradas desconectadas devido a componentes que possuem mais portas do que o desejado. Tais entradas atuam como se uma tensão estivesse sendo aplicada a ela. Além disso, ela atua como uma antena e pode captar sinais irradiados capazes de causar o funcionamento inadequado do circuito. Portanto, caso as entradas de um circuito possuam entradas não utilizadas recomenda-se aplicar a tensão (para **0 V** ligue no terra – **GND**) que produza o resultado desejado.

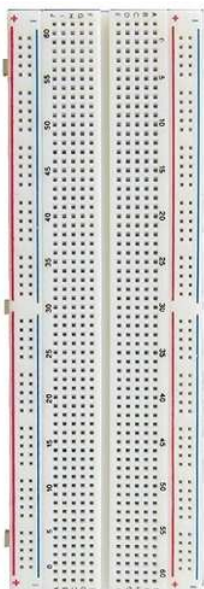
Os componentes das famílias PMOS, NMOS e CMOS que utilizam transistores MOSFET são relativamente simples se comparados aos TTL. Eles possuem baixo custo de fabricação e são menores. Como os elementos MOS, em geral, não utilizam resistores, eles ocupam menos espaço nos chips eletrônicos se comparados aos transistores bipolares. A principal desvantagem é um risco muito maior se comparado ao TTL de ser danificado por eletricidade estática devido a sua estrutura interna. Portanto,



você verá, na maioria das vezes, componentes TTL sendo usados em laboratórios para o aprendizado. Assim como os TTL, as entradas CMOS nunca devem ficar desconectadas, todas devem ser conectadas a um nível de tensão fixo ou a alguma outra entrada.

## 2.7 Protoboard

O processo de criação de um circuito envolve várias etapas e uma delas é testar se seu projeto funciona na prática criando um protótipo. Geralmente, as primeiras vezes em que você vai verificar o funcionamento do seu circuito exigem que você realize testes com diferentes componentes, configurações e adapte valores teóricos de acordo com a disponibilidade dos elementos no mercado. A rotina pode ser facilitada com o uso de uma Protoboard apresentada na figura 2.7-1.



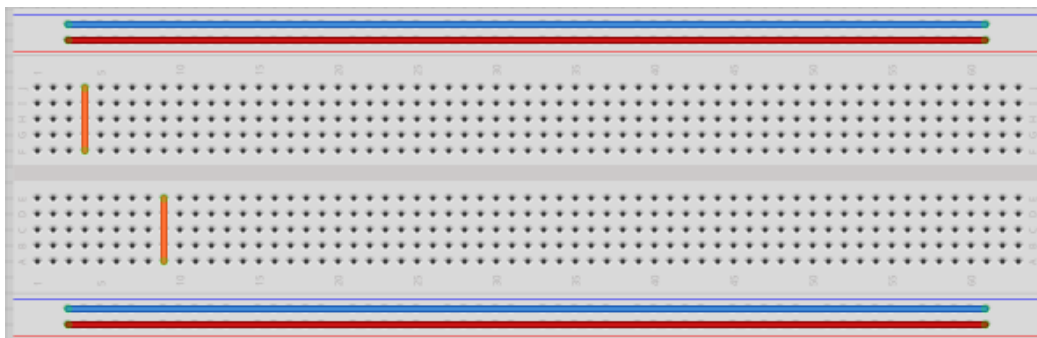
**Figura 2.7-1 - Protoboard**

A Protoboard é uma placa que reúne conjuntos de furos para dar acesso a pontos ligados em paralelo. A figura 2.7-2 mostra quais fileiras de furos estão ligadas em paralelo. Cada fio cobre a fileira de pontos que estão associados. O fio laranja indica que os cinco furos de cada coluna estão ligados em paralelo. O mesmo raciocínio vale para os fios vermelhos e azuis. É muito **importante** observar que os furos da metade superior da Protoboard (azuis, vermelhos e laranjas) não estão ligados em paralelo com os fios de mesma cor da metade inferior da Protoboard.

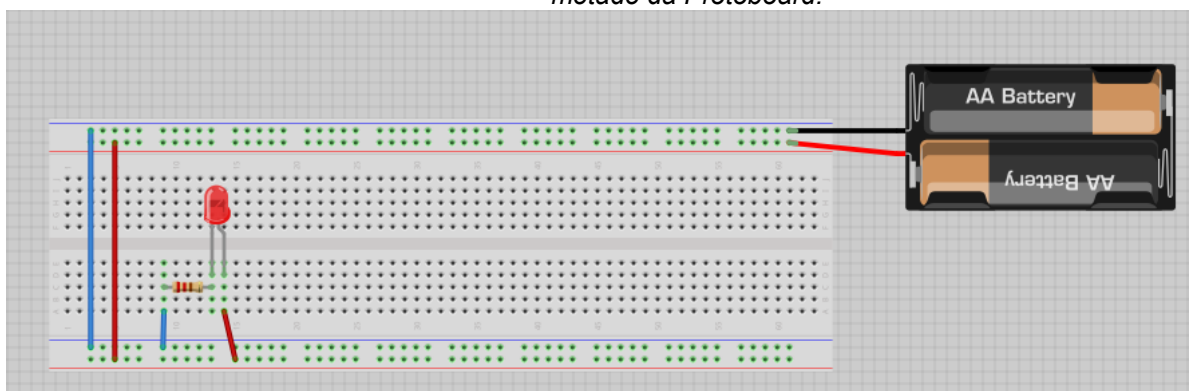
O principal ponto de vantagem da Protoboard é a facilidade com que os fios e componentes podem ser ligados na placa. As extremidades do componente podem ser facilmente inseridas ou desconectadas dos furos, eliminando a necessidade de realizar soldas e fitas isolantes para realizar o

contato de componentes. O processo de testes com ela permite corrigir todos os erros no circuito antes de soldar em um Shield definitivamente.

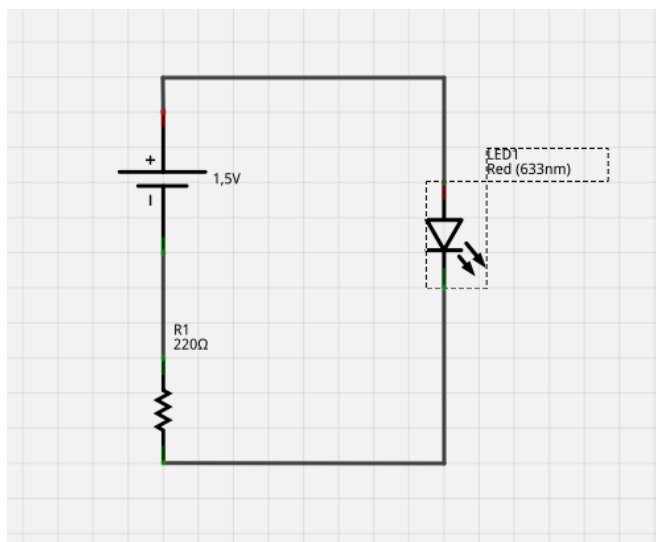
**Figura 2.7-2** – Ligação de fios na Protoboard indicando furos em paralelo. Observe que a



Protoboard é simétrica horizontalmente e que a linha de simetria isola os fios de cada metade da Protoboard.



**Figura 2.7-3** – Acionamento de LED utilizando Protoboard



**Figura 2.7-4** – Esquemático do circuito montado na Protoboard na figura 2.7-2

A figura 2.7-3 apresenta um exemplo de utilização da Protoboard para o acionamento de um LED. O circuito esquemático apresentado na figura 2.7-4 apresenta o equivalente ao circuito de acionamento de LED.

### 3. Linguagem de programação

#### 3.1 – Funções Base

Vamos chamar de programa uma sequência de ações programadas passadas previamente à placa Arduino. Toda vez que ela for ligada ela seguirá aquela programação de forma sequencial.

Um programa possui um corpo, uma **função** principal, onde ele será rodado. Em Arduino, temos como funções base as funções:

```
void setup() {  
  
}  
void loop() {  
  
}
```

A função **void setup()** é a função que inicializa. Ela é rodada apenas uma vez ao alimentar a placa Arduino e nela contém as informações necessárias para a programação feita na função **void loop()**, que por sua vez roda até que a alimentação seja cortada. Dentro da **void loop()** contém as ações principais do programa.

Uma das coisas que devemos informar na função é a **configuração dos pinos** que vamos usar como **pinos de entrada** (INPUT) ou **pinos de saída** (OUTPUT). Fazemos isso da seguinte maneira:

```
pinMode(pino, MODO);
```

Onde *pino* é o número indicativo da porta que está sendo configurada e

#### Observação:

Repare que cada placa possui a indicação dos pinos. Os pinos de entrada analógica são indicados por A0, A1, A2, A3, A4 e A5, no caso da placa Arduino UNO, Duemilanove e semelhantes. Essas portas não precisam ser identificadas na função de setup se estiverem sendo usadas para **leitura de dados analógicos**. Porém, essas portas também podem ser usadas como pinos digitais, e nesse caso eles devem ser identificados e a numeração deles é dada de **14** (no caso do pino A0) até **19** (no caso do pino A5).

**MODO** é a configuração que ela está recebendo (entrada ou saída).

Para tratar os dados de forma adequada devemos usar alguns comandos específicos. A tabela abaixo mostra comandos de portas digitais e suas funções:

Comando	Função
<b>digitalRead</b> (pino*);	Esse comando é responsável pela leitura de dados digitais. Ela tem o valor de HIGH ou LOW dependendo do estado do dado de entrada.
<b>digitalWrite</b> (pino*, VALOR);	Esse comando define o estado daquela porta previamente configurada na função <b>void setup()</b> em alta (HIGH) ou baixa (LOW). Em que HIGH significa ligada e LOW significa desligada.

As estruturas de controle de fluxo serão tratadas mais adiante. Mas, para obtermos um exemplo interessante e entendermos o funcionamento das funções base, a seguir é apresentada a estrutura **IF**:

```
if(condicao){  
  //comandos...  
}
```

IF testa se a condição foi alcançada, como por exemplo, se uma entrada digital encontra-se em HIGH ou LOW. Se o resultado da condição for 0 (falsa/false), os comandos não serão executados. Se o resultado da condição for 1 (verdadeira/true), os comandos serão executados.

O exemplo mostrado a seguir configura os pinos A0 e A3 como pinos de saída, onde serão ligados atuadores, os pinos 4 e 5 como pinos de entrada. Em seguida, liga a porta A0 caso a entrada em 4 esteja em HIGH e a porta A3 caso a entrada em 5 esteja em HIGH:

A screenshot of the Arduino IDE interface. The title bar reads 'exemplo1 | Arduino 1.0.1'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for checking, running, uploading, and downloading. The main text area shows the following code:

```
exemplo1 $  
  
void setup() {  
  pinMode(14, OUTPUT); //configura porta A0 como saída  
  pinMode(17, OUTPUT); //configura A3 como saída  
  pinMode(4, INPUT); //configura 4 como entrada  
  pinMode(5, INPUT); //configura 5 como entrada  
}  
  
void loop() {  
  if(digitalRead(4)==HIGH) { //Se entrada 4 estiver em HIGH  
    digitalWrite(14); //Liga a saída 14  
  }  
  
  if(digitalRead(5)==HIGH) { //Se a entrada 5 estiver em HIGH  
    digitalWrite(17); //Liga a saída 17  
  }  
}
```

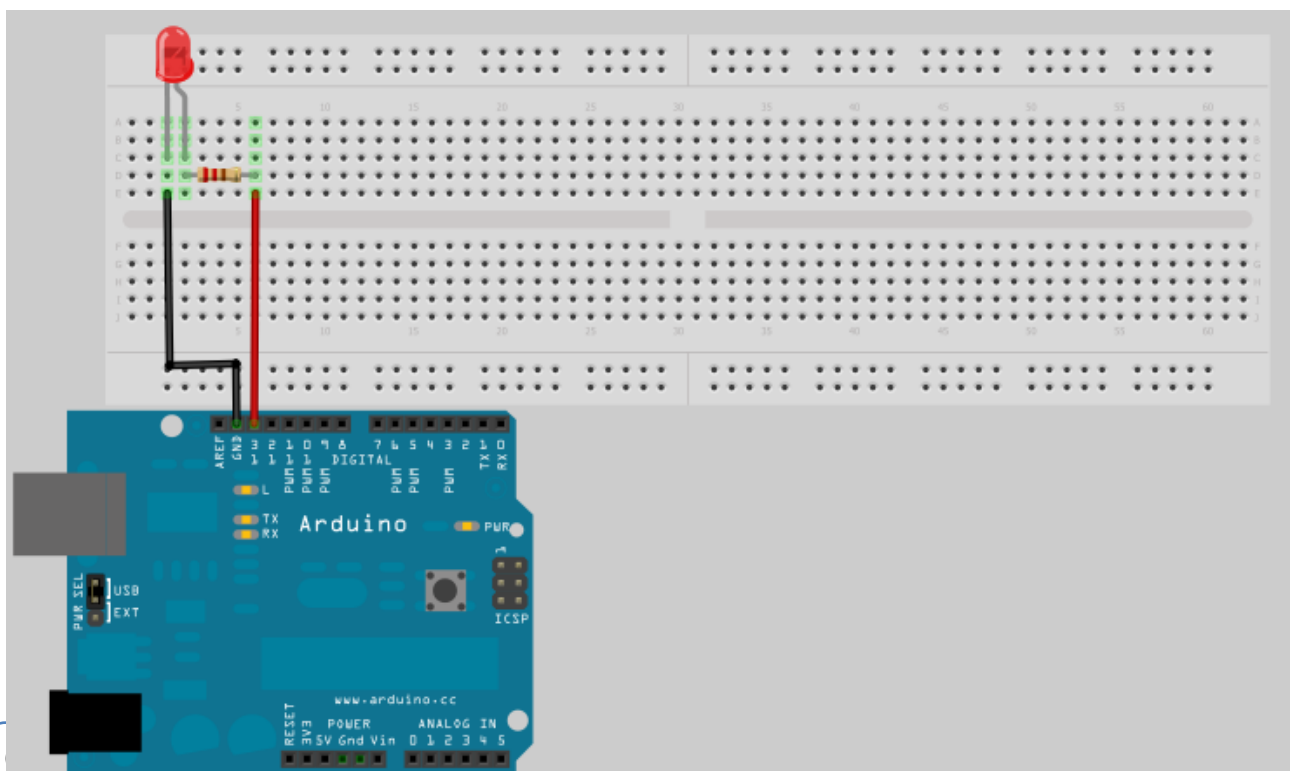
Below the code editor, a status bar shows 'Done Saving.' and '7'. At the bottom right, it says 'Arduino NG or older w/ ATmega8 on COM3'.

### 3.2 – Tempo

A placa Arduino é capaz de calcular intervalos de tempo com grande precisão. Existem alguns comandos que permitem o controle da placa de acordo com o tempo decorrido. Na próxima tabela são apresentados alguns desses comandos:

Comando	Função
<code>delay</code> (tempo em ms)	Este comando possibilita um atraso (pausa) no programa em execução em uma quantidade específica de milissegundos que é nomeada no campo <b><i>tempo em ms</i></b>
<code>delayMicroseconds</code> (tempo em µs)	Este comando é parecido com o <code>delay</code> , mas nesse caso o tempo é colocado em microssegundos.
<code>millis</code> ()	Este comando retorna a quantidade de tempo que se passou desde que o programa atual começou a ser executado, em milissegundos. Para usar este comando é necessário o uso da variável <b><i>unsigned long</i></b> .

O exemplo 2 apresentado a seguir tem como função piscar um LED conectado a uma saída configurada no pino 13. Para vê-la em funcionamento, monte o circuito como apresentado na figura 2.2-1.



A comparação de `digitalRead()` com `HIGH` é feita com o operador `"=="`. Esse assunto vai ser tratado com detalhes em 2.6.

O `"//"` dentro da programação serve para adicionar comentários. O que estiver escrito na mesma linha e depois de um `"//"` não será considerado pelo Arduino.

Figura 2.2-1 – Circuito para exemplo 2. Arduino + Resistor 220Ω + LED



Exemplo 2 – Pisca LED na porta 13.

### 3.3 – Bibliotecas e #define

As funções que falamos até agora estão todas em uma biblioteca que é adicionada automaticamente nos programas do Arduino. Entretanto, as vezes será necessário incluir uma ou mais bibliotecas extras em seu programa. Mas o que é uma biblioteca?

Os programadores devem sempre simplificar ao máximo os códigos visando economizar espaço na memória do programa e facilitar mudanças e revisões. Para isso são fornecidas bibliotecas com funções diferentes das básicas apresentadas pelo Arduino, e uma das vantagens dessa plataforma é justamente a facilidade de se encontrar bibliotecas prontas.

A comprar um sensor muitas vezes o fabricante disponibiliza uma biblioteca para ser usada, assim, ao programa-lo, você evita ter que escrever extensas linhas de código e basta usar funções já presentes na biblioteca.

Para incluir uma biblioteca você deve ir ao menu “**Sketch > Import**





**Library...**” . Irá aparecer no seu código algo como mostrado a seguir:

```
#include <NomeDaBiblioteca.h>
```

Cada biblioteca tem as suas funções, provavelmente são mostradas em alguma instrução passada pelo fabricante. Ao baixar, a pasta dela deve ser extraída para **libraries**, presente no mesmo diretório do software Arduino.

Outra maneira de simplificar o seu código é usando a ferramenta `#define` no cabeçalho do programa. Ela é colocada da seguinte forma:

```
#define nomeVariavel dados (OBS: não é usado “;” no final)
```

Dessa forma, ao longo de todo o programa, toda vez que a palavra `nomeVariavel` for usada, ela será substituída por `dados`. Por exemplo:

```
#define PINO_LED 13
```

Toda vez que colocarmos `PINO_LED` no código, essa palavra será substituída por `13`. Mas em quê isso pode simplificar meu código? Vamos supor que em várias partes do seu programa tenha o número `13` para indicar o pino de um LED conectado e, por algum motivo, você mudou a porta desse LED para o pino `10`. Agora você não precisa substituir todos os números `13` por `10`, basta você mudar no cabeçalho o seu `#define PINO_LED 13` para `#define PINO_LED 10`.

### 3.4 – Variáveis

As variáveis, do ponto de vista da área de programação, são regiões de memória previamente selecionadas que tem por finalidade armazenar os dados ou informações de um programa por um determinado espaço de tempo. Uma variável limita-se a armazenar apenas um valor por vez.

As variáveis podem ter duas classes: variáveis locais ou globais.

#### **Variáveis locais:**

Quando uma variável é declarada dentro de uma função específica (entre { e } ), ela é denominada variável local. Estas variáveis apenas existem enquanto o bloco onde está armazenada estiver sendo executado. A partir do momento em que o programa voltar à função principal esta variável deixará de existir.

#### **Variáveis globais:**

Uma variável global é aquela que é conhecida por todo o programa, ou seja, independente da função que estiver sendo executada ela será reconhecida e rodará normalmente. Uma variável global é declarada antes mesmo da função `void setup()`.

Para usar uma variável deve-se declará-la da seguinte maneira:

***Tipo\_de\_variavel lista\_de\_variaveis***

Ex: `int x, y;`

Há vários tipos de variáveis, dependendo do tipo de informação que elas podem conter. Cada tipo de variável ocupa um espaço diferente na memória de programa do computador, e este espaço também é influenciado pela plataforma onde o programa está sendo rodado. Os cinco tipos de dados primitivos são:

Tipo de variável	Palavra-chave	Espaço em disco ocupado
Sem informação (vazia)	void	-
Caractere	char	Geralmente 1 byte*
Número inteiro	int	Geralmente 4 bytes*
Número real	float	Geralmente 4 bytes*
Número real com dupla precisão	double	Geralmente 8 bytes*

\*Considerando o tamanho de uma “palavra” para um computador de 32-bits.

### Caractere

O tipo char em C é utilizado para representação de caracteres na tabela ASCII, mas também pode ser utilizado para armazenar valores numéricos inteiros. Não é aconselhável utilizar operadores numéricos em variáveis do tipo caractere. No geral uma variável do tipo char consegue armazenar um valor numérico entre -128 a 127.

### Número Inteiro

O tipo int em C é utilizado para armazenar uma variável numérica inteira, isto é, não é armazenado componentes decimais ou fracionários. O tipo inteiro em C costuma ter o tamanho de uma “palavra” de máquina, por exemplo, para um computador com um processador de 32 bits uma palavra de máquina possui 32 bits, ou 4 bytes, de tamanho. Sabendo que cada bit consegue armazenar apenas 2 valores, o número total de combinações é de  $2^{31}$ , pois o primeiro bit armazena a informação do sinal. Nesse padrão o maior número que uma variável do tipo int pode armazenar é de  $(2^{31} - 1) = +2147483647$  e o menor número é de  $-(2^{31}) = -2147483648$ .

### Número real

No C existem duas formas de se representar os números reais com tipos de dados básicos: o float e o double.

O padrão ANSI C determina que a faixa mínima de um valor de ponto flutuante é de  $10^{-37}$  a  $10^{+37}$ . O número mínimo de dígitos de precisão para o tipo float é de seis dígitos e o do double é de dez dígitos.

### Void

O tipo void é utilizado apenas para dois fins muito específicos: Indicar que uma função não retorna nenhum valor ou para a criação de ponteiros genéricos.

O exemplo 3, mostrado a seguir, mostra um código que utiliza variáveis para armazenar o número de vezes que o LED da porta 13 piscou

e quando esse número chegar em 10, pausa por 10 segundos.

Observe que a atribuição de um valor para a variável é feita usando o operador "=", e pode ser feita a qualquer momento. A operação contador = contador + 1; é feita para adicionar 1 ao valor antigo da variável, e assim que é feita a contagem.



```
#define LIMITE 10

int contador=0; //declara a variavel global para contar e atribui 0 a ela

void setup() {
  pinMode(13, OUTPUT); //configura porta 13 como saída
}

void loop() {
  digitalWrite(13, HIGH); //liga o LED ligado à porta 13
  delay(200); //pausa de 500 milissegundos
  digitalWrite(13, LOW); //desliga o LED ligado à porta 13
  delay(200); //pausa de 500 milissegundos

  contador = contador + 1; //soma mais 1 ao contador

  if(contador == LIMITE) {
    delay(10000); //pausa por 10 segundos
  }
}
```

*Exemplo 3 – Aplicação de variável inteira para contador.*

### 3.5 – Funções

Como já vimos, existem algumas funções básicas já presentes na linguagem Arduino de programação, que são as `main()` e `loop()`, indispensáveis para a compilação do código. Além dessas funções, podemos criar outras. Uma função é uma chamada a uma série de instruções passadas ao Arduino e servem para facilitar a programação quando essas instruções são passadas várias vezes pelo

código.

Uma função nova que é criada pode receber um valor de entrada, um parâmetro, que serve como uma variável dentro do escopo daquela estrutura. Para isso, deve-se informar o nome e o tipo de variável desses parâmetros de entrada. Também pode retornar um valor, depois que executada, para a função que a chamou.

Para criar uma função, deve seguir da seguinte maneira:

```
Tipo_da_função nome_da_função (tipo var1, tipo var2,...,tipo varN)
{
    corpo da função
}
```

Um exemplo pode ser dado numa situação em que, em várias partes do programa, se utiliza LEDs piscando 3 vezes, não necessariamente na mesma frequência. Para isso, cria-se uma função que recebe como parâmetro de entrada, o tempo em que o LED permanece ligado e o tempo em que permanece desligado. Essa função não retorna nenhum valor, por isso o tipo da função é **void**. Faz-se isso da seguinte maneira:

```
void pisca_LED (int tempo_ligado, int tempo_desligado) {
    digitalWrite (13, HIGH);
    delay (tempo_ligado);
    digitalWrite (13, LOW);
    delay (tempo_desligado);
    digitalWrite (13, HIGH);
    delay (tempo_ligado);
    digitalWrite (13, LOW);
    delay (tempo_desligado);
    digitalWrite (13, HIGH);
    delay (tempo_ligado);
    digitalWrite (13, LOW);
    delay (tempo_desligado);
}
```

Agora, toda vez que quiser piscar LED, basta chamar a função com os valores desejados para tempo ligado e tempo desligado, como por exemplo, da seguinte forma: `pisca_LED (500, 300);`

Para que uma função seja validada, ela deve ser posicionada antes da função que a chama, caso contrário, deve ser feito uma declaração da função antes, isto é, informar ao programa o nome, tipos de entradas e saída da função, mas sem colocar o corpo inteiro, da seguinte forma:

```
Tipo_da_função nome_da_função (tipo var1, tipo var2,...,tipo varN);
```

O comando **return** é utilizado para retornar valores dentro de uma função. Após o comando `return`, a função sempre termina sua execução, independente do

que esteja escrito posteriormente.

### 3.6 – Operadores booleanos, de comparação, incremento e decremento

Operadores de comparação

X é igual a Y:	$X == Y$
X é diferente de Y:	$X != Y$
X é menor que Y:	$X < Y$
X é maior que Y:	$X > Y$
X é menor ou igual a Y:	$X <= Y$
X é maior ou igual a Y:	$X >= Y$

Operadores booleanos são aqueles que verificam se a função é verdadeira ou não (1 ou true para verdadeiro e 0 ou false para falso)

#### Operador E (&&)

A condição é verdadeira quando os dois itens forem verdadeiros. No exemplo abaixo, o corpo da condição if só vai acontecer se a leitura digital dos dois pinos forem 1/HIGH/true:

```
if(digitalRead(2) == 1 && digitalRead(3) == 1) {  
... }
```

#### Operador OU (||)

A condição é verdadeira quando pelo menos um dos itens for verdadeiro:

```
if(digitalRead(2) == 1 || digitalRead(3) == 1) {  
... }
```

#### Operador NÃO (!)

A condição é verdadeira se o item for falso:

```
if(!digitalRead(2)) {  
... }
```

#### Operadores de incremento e decremento

Na linguagem usada para o Arduino, usamos operadores unários e binários. Operadores unários agem em apenas uma variável e modifica ou não, enquanto os operadores binários utilizam duas variáveis, pegam seus valores sem alterá-los e retorna um terceiro valor. A soma é um exemplo de operador binário. Os operadores de incremento e decremento são operadores unários, pois alteram a variável em que estão aplicados. Dizemos:

$x++$ ; que equivale a  $x=x+1$   
 $x--$ ; que equivale a  $x=x-1$ ;

Os operadores de incremento e decremento podem ser pré-fixados ou pós-fixados. Os pré-fixados incrementam e retornam o valor da variável já



incrementada, enquanto os pós fixados retornam o valor da variável sem o incremento e depois incrementam a variável. Por exemplo:

#### **Pré-fixado**

Se:  $x=48$   
 $y=++x$   
Então:  
 $x=49$   
 $y=49$

#### **Pós fixado**

Se:  $x=48$   
 $y=x++$   
Então:  
 $x=49$   
 $y=48$

Outras usos dos operadores de incremento e decremento são:

$X+=y$	equivalente a	$x=x+y$
$x-=y$	equivalente a	$x=x-y$
$x*=y$	equivalente a	$x=x*y$
$x/=y$	equivalente a	$x=x/y$

### **3.7 Estruturas de controle de fluxo**

São as estruturas mais importantes do seu código. Com elas é possível tomar decisões ou repetir ações.

#### **Estruturas de Decisão**

O principal comando de decisão em C é o comando if (se). A estrutura do comando if é a seguinte:

```
if(condicao)
comando;
```

O funcionamento é simples. Caso a condição entre parênteses for verdadeira, o comando abaixo é executado. Apenas o comando imediatamente posterior será executado. Caso mais de um comando devam ser executados, estes devem estar entre chaves:

```
if(condicao)
{
comando1;
comando2;
comando3;
}
```

Desta forma, todos os comandos serão executados caso a condição seja verdadeira.

Juntamente com o comando if, também pode ser utilizado o comando else (senão), da seguinte maneira:

```
if(condicao) {
comando1; }
else {
```

```
comando2; }
```

Com isso, se a condição entre parênteses for verdadeira, o comando 1 é executado, se for falsa, o comando 2 é executado. Mais de um comando pode ser colocado após o else, utilizando chaves.

É possível colocar um if dentro de outro, para realizar diferentes checagens antes de realizar uma ação.

### **Estruturas de Repetição**

Estruturas de repetição são utilizadas para realizar um comando (ou um grupo de comandos) repetidamente, desde que estes sigam um padrão. São muito úteis pois podem poupar inúmeras horas de trabalho e também várias linhas de código.

Por exemplo, se quisermos fazer um código que subtraia 2 de um número até que este seja menor que 10. Como fazer isso sem uma estrutura de repetição?

Os principais comandos de repetição são for, do...while e while.

#### **For**

O comando for tem a seguinte estrutura:

```
for(inicializacao;condicao_de_repeticao;comando_final) {  
comando;  
}
```

Sendo:

Inicialização: um comando qualquer, geralmente utilizado para criar e inicializar um contador.

Condição de repetição: ao final dos comandos, esta condição é checada. Caso seja verdadeira, haverá mais uma repetição (loop).

Comando final: Este comando é executado sempre que o loop termina, logo após a condição de parada ser checada, geralmente incrementa um contador.

Daquela maneira, o comando será realizado até que a condição de repetição seja falsa.

#### **While**

O comando while (enquanto) também é utilizado para realizar repetições de comandos dentro do código. Sua estrutura padrão é a seguinte:

```
while(condicao_de_repeticao) {  
comando;  
}
```

Desta maneira o comando será executado repetidamente enquanto a condição de parada for verdadeira. A condição de parada é checada no início do loop, antes da execução do comando. Assim, se a condição for falsa na primeira checagem, o comando não será executado nenhuma vez.



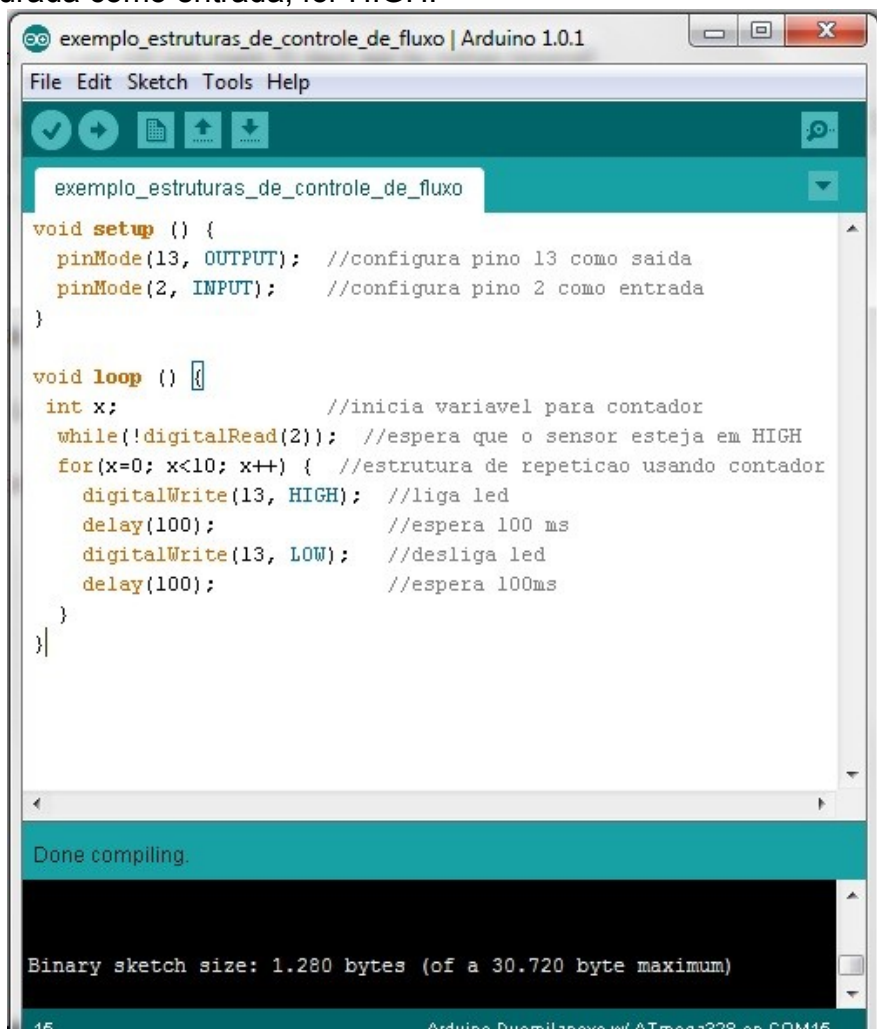
### Do...While

Este comando é uma variação do comando while, e tem a seguinte estrutura:

```
do {  
  comando;  
}  
while(condicao_de_repeticao)
```

A diferença entre esta estrutura e a do while é que, se a condição de repetição for falsa desde o início, o comando é executado uma vez, pois a checagem da condição é feita apenas no final do loop.

Segue um exemplo de programa utilizando as estruturas de controle de fluxo. O código passado para o Arduino fará com que ele pisque um LED conectado ao pino 13 dez vezes quando um sensor digital conectado à porta 2, configurada como entrada, for HIGH.



Exemplo 4 – Estruturas de controle de fluxo

### 3.8 – Comunicação serial



Uma importante ferramenta para comunicação com outros dispositivos é a comunicação serial. Um Arduino, estando conectado a outro dispositivo, é capaz de enviar e receber dados. Para isso, usa-se funções Serial.

Primeiro passo para usar de forma adequada a comunicação serial é configurar a taxa com que vão ser transmitidos os dados, em bits por segundo. O Arduino consegue emitir a taxa nos valores de 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200 bits por segundo. Há equipamentos que exigem essa taxa específica. Um computador, por exemplo, é configurada por padrão por receber dados da USB numa taxa de 9600.

Para tal configuração, na função void setup(), devemos iniciar a comunicação e informar a taxa da seguinte maneira:

**Serial.begin(taxa);**

Alguns comandos relacionados à comunicação serial estão citados a seguir:

**Serial.available();**

Obtém o número de bytes disponíveis para a porta serial. Este comando retorna o número de bytes disponível para leitura.

**Serial.read();**

Recebe dados da porta serial para leitura.

**Serial.print();**

Envia dados para a porta serial como texto legível ASCII. Este comando pode possuir diferentes formas, por exemplo:

- **Serial.print(78)** resulta em "78"
- **Serial.print(1.23456)** resulta em "1.23"
- **Serial.print(byte(78))** resulta em "N" (em ASCII seu valor é 78)
- **Serial.print('N')** resulta em "N"
- **Serial.print("Ola Mundo. ")** resulta em "Ola Mundo."

**Serial.println();**

Envia dados para a porta serial, porém, com um caractere de retorno. Este

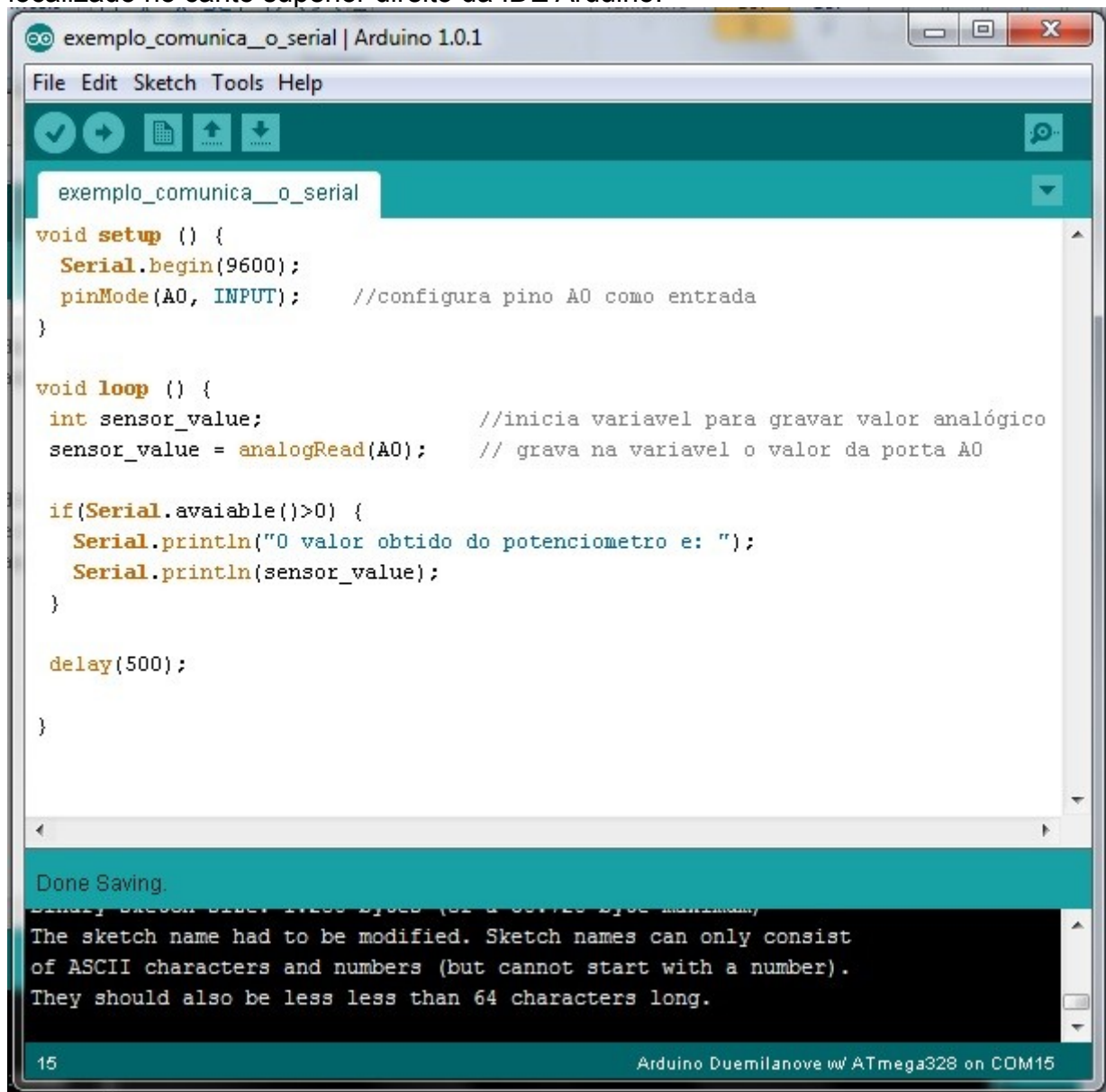
#### **Observação:**

Para facilitar a tarefa de programar, existem algumas constantes ou caracteres que possuem significados diferenciados. A lista abaixo mostram esses significados:

**CaractereSignificado**  
\\nNova linha\\tTabulação Horizontal ("tab")  
\\aTabulação verticalBINBinário ou Base 2OCTBase 8HEXHexadecimal ou base 16DECDecimal ou base 10BYTEResulta em Byte

comando funciona da mesma forma que **Serial.print()**.

Vamos mostrar agora um exemplo usando a comunicação serial com o computador para exibir na tela os valores lidos por um sensor analógico localizado na porta analógica A0. Para ver o resultado, basta clicar no botão “Serial Monitor”, localizado no canto superior direito da IDE Arduino.



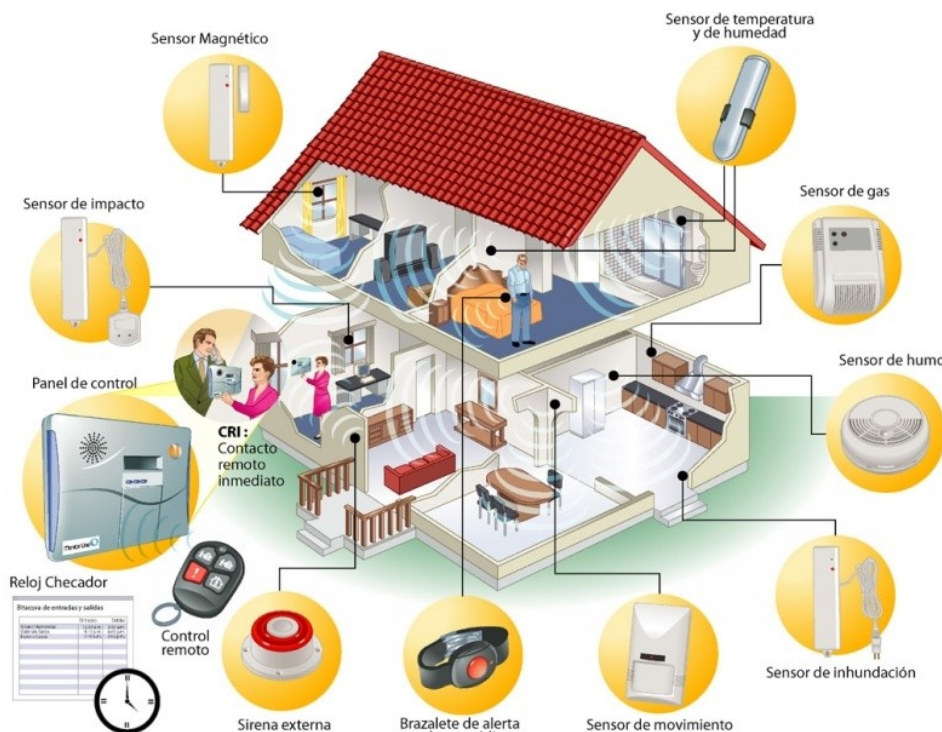
Exemplo 5 – Comunicação Serial.

## 4. Portas analógicas e digitais

### 4.1 Sensores Digitais

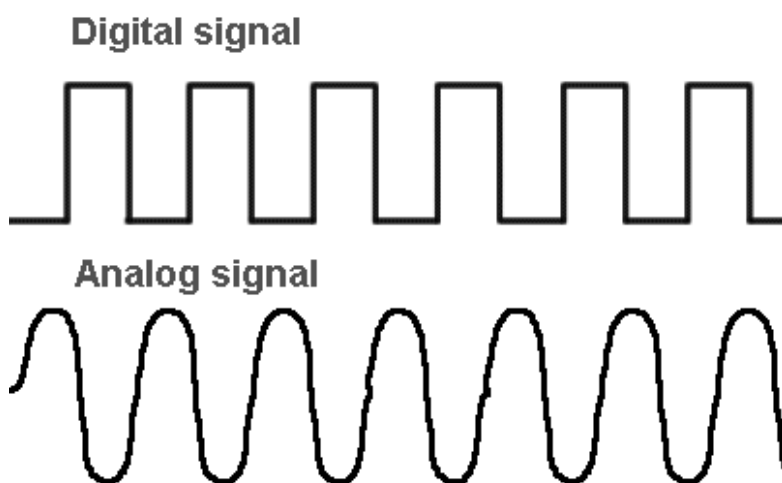
Sensores são dispositivos usados em circuitos eletrônicos que tem como finalidade a obtenção de variáveis físicas de um meio externo. O termo sensor é

usado para designar qualquer elemento sensível a alguma forma de energia, podendo essa energia ser luminosa, térmica, cinética, etc. Sensores também tem a função de relacionar a energia captada a medida de uma grandeza como distância, temperatura pressão, etc . A figura 4.1-1 mostra alguns exemplos de sensores.



*Figura 4.1-1 –Exemplos de sensores em uma residencia*

Os sensores podem ser divididos em analógicos ou digitais. Sensores analógicos (abordados com mais detalhes na seção 4.2 dessa apostila) são aqueles que medem variáveis que podem assumir qualquer valor dentro de uma faixa de operação. Sensores digitais, por outro lado, podem assumir somente dois valores binários: 1 e 0 (HIGH/LOW). Como se vê na figura 4.1-2, os gráfico de uma medida analógica é contínuo e o de uma medida digital não.



*Figura 4.1-2 – Sinal digital Vs. Sinal analógico*

Os sensores digitais, em um arduino, vão poder detectar presença ou ausência de voltagem em um pino. Para fazer a leitura

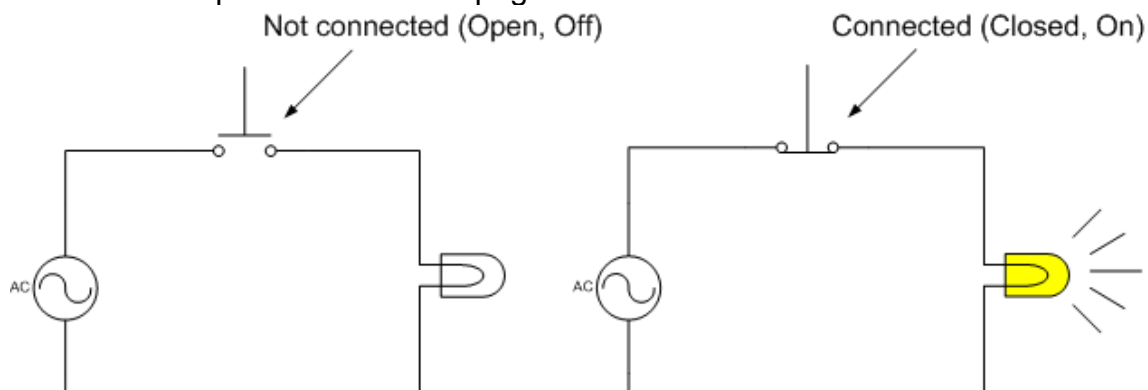
de um sensor se deve utilizar a função “digitalRead()” que para uma voltagem de 5V terá como o valor lido HIGH e para uma voltagem de 0V terá seu valor sendo LOW. A conexão do sensor no arduino será feita através dos pinos digitais, destacados na figura 4.2-3. É importante notar que nem sempre a voltagem na entrada é de 0V ou 5V, para resolver esse problema é necessária a utilização de resistores para “puxar” a tensão para HIGH ou LOW.



*Figura 4.1-3 – Pinos digitais de um arduino*

#### 4.1.1 Usando um Switch

Um switch nada mais é do que um botão ou chave que, como um interruptor de luz, ao ser acionado interrompe ou refaz o contato de um circuito elétrico. Switchs são os mais simples exemplos de sensores digitais: Se a chave está fechada o sinal elétrico passará e o Arduino poderá fazer a leitura de um sinal HIGH, se a chave estiver aberta o sinal não passará e o Arduino fará a leitura do sinal como LOW (a lógica inversa também pode ser implementada). Para ilustrar melhor pode-se pensar em uma lâmpada de uma casa, quando o interruptor está ligado o contato será feito e o sinal passará acendendo a luz(HIGH), quando o interruptor estiver desligado o sinal não passará e a luz será apagada(LOW). A figura 4.1.1-1 mostra um circuito para acender ou apagar a luz.

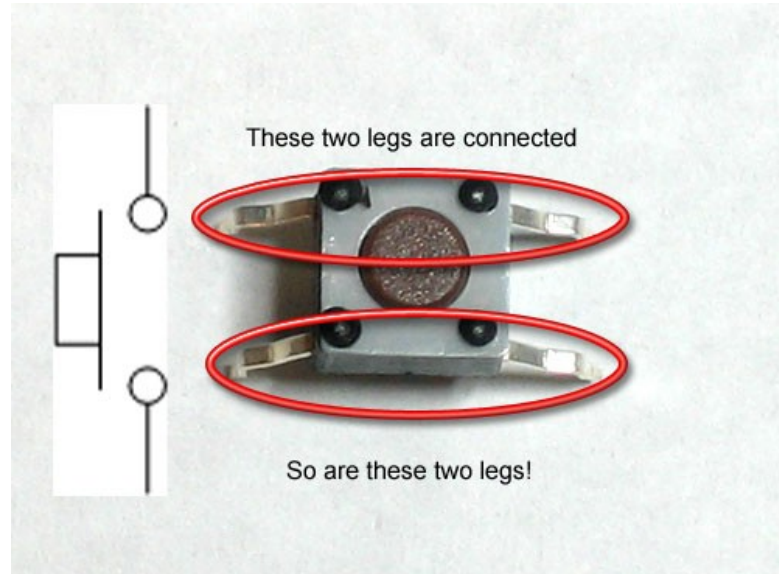


*Figura 4.1.1-1 – Chave acendendo e apagando a luz*

Vamos agora para um exemplo prático, feito em no arduino. A

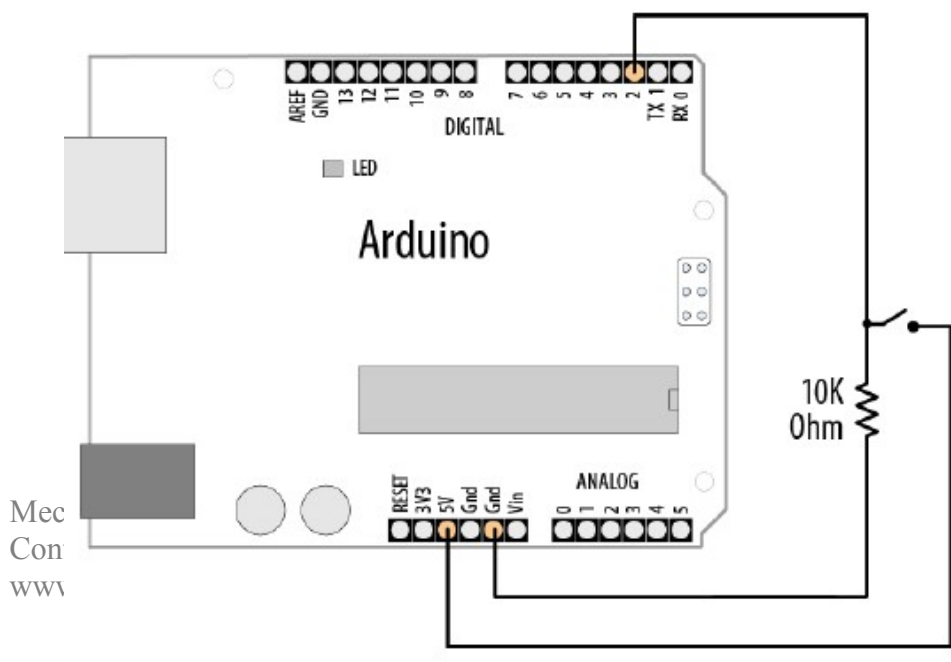
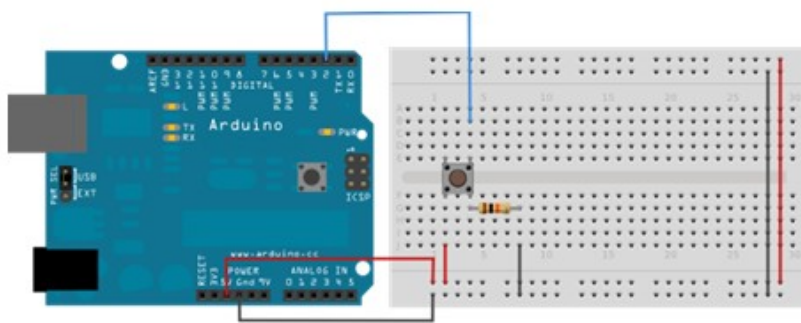


montagem envolve o uso de um botão, como o da figura 4.1.1-2, para apagar ou acender um LED em um arduino. Quando o botão está pressionado o contato deve se fechar e a LED acender.




*Figura 4.1.1-2 – Exemplo de um Switch (botão)*

A montagem a ser feita está esquematizada na figura 4.1.1-3 e o código pode ser visto na figura 4.1.1-4



*Figura 4.1.1-3 – Modelo esquemático para a montagem de um switch*



A screenshot of the Arduino IDE interface. The title bar reads 'Exemplo\_4\_1\_1\_1 | Arduino 1.0.1'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for opening, saving, and running a sketch. The main text area shows the following code:

```
Exemplo_4_1_1_1

const int ledPin = 13; // Escolhe o pino para o LED
const int inputPin = 2; // Escolhe o pino para o botão

void setup() {
  pinMode(ledPin, OUTPUT); // declara o LED como output
  pinMode(inputPin, INPUT); // declara o botão como input
}

void loop(){
  int val = digitalRead(inputPin); // Lê o input (Valor do botão)
  if (val == HIGH) // checa se o input está em nível alto (HIGH)
  {
    digitalWrite(ledPin, HIGH); // Liga o LED caso o botão esteja pressionado
  }
  else
  {
    digitalWrite(ledPin, LOW); // Desliga o LED caso o botão não esteja pressionado
  }
}
```

*Figura 4.1.1-4 – Código para o switch acender o LED ao pressionado*

Note que essa função está acendendo o LED que já se encontra embutido no Arduino.

Observando a montagem feita na figura 4.1.1-3, pode-se notar que é colocado um resistor em série com o Ground. Esse resistor está puxando a tensão para baixo (pull down) quando a chave está aberta. O que ocorre é que o pino de leitura vai ler qualquer valor acima de 2,5V como HIGH, então quando a chave está aberta o valor de tensão que chega nele é indeterminado e pode acabar sendo HIGH mesmo sem o botão estar pressionado. Dizemos que esse valor pode sofrer flutuações e podemos resolver esse problema colocando o resistor, no entanto, nos dá a garantia de que o valor lido será LOW, pois a tensão existente acabará sendo levada para o resistor. (reescrever menos confuso)

#### **4.1.2 Detecção de movimento**

Um sensor de movimento pode ser comparado ao switch. Quando há movimento a chave de fecha (ou abre) emitindo um pulso e mudando o estado da variável lendo a presença do movimento.

O esquema da figura 4.1.2-1 e o código da figura 4.1.2-2 ilustram o funcionamento de um sensor infravermelho.

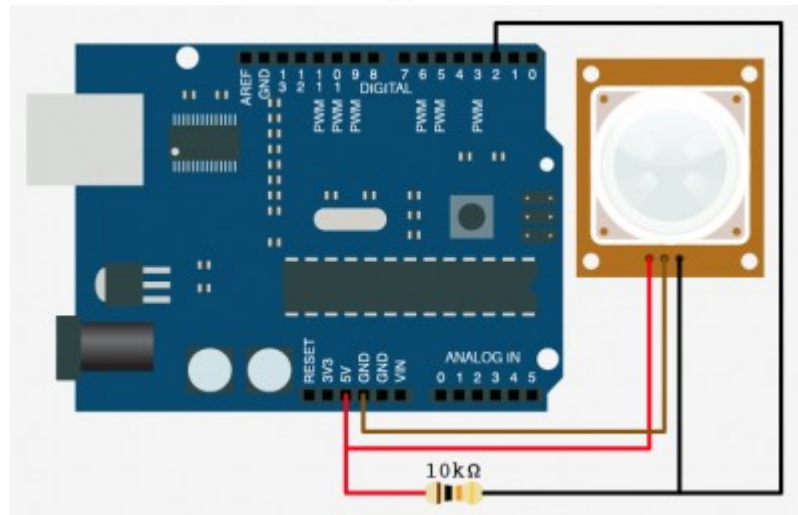


Figura 4.1.2-1 – Montagem esquemática da ligação de um sensor infravermelho

```
Exemplo_4_1_1_2 | Arduino 1.0.1
File Edit Sketch Tools Help
Exemplo_4_1_1_2 $
int sensorPin = 2; //pino que o sensor está conectado
int LedPin = 13; //pino que o led está conectado (embutido no arduino)
void setup(){
    pinMode(sensorPin, INPUT); //declara o sensor como input
    pinMode(LedPin,OUTPUT); // Declara o Led como output
}
void loop(){
    int sensorVal = digitalRead(snesorPin); // Coloca o valor detectado pela sensor
                                         // na variável sesnorVal
    if(sensorVal == LOW)
    {
        digitalWrite(LedPin,HIGH); //Detecta se houve movimento
        delay(2000);
    }
    else
    {
        digitalWrite(LedPin,LOW);
    }
}
```

Figura 4.1.2-2 – Código para a detecção de movimento

O sensor do exemplo, sendo infravermelho, emite um feixe de luz. Enquanto esse feixe não é interrompido não há movimento e, portanto, a entrada será HIGH (representando que o feixe não foi interrompido). Caso haja uma interrupção no feixe haverá também uma interrupção na corrente e a tensão na entrada deverá cair para zero, mudando o estado da variável

lida para LOW. Quando a mudança ocorrer o LED deverá ser aceso.

## 4.2 Sensores analógicos

Os dados e informações utilizados, atualmente, em quase todas as áreas, como por exemplo, computadores, automação, robôs, tecnologia e ciência médica, transportes, telecomunicações, entretenimento, exploração espacial, são *analisados* no formato *digital*. Apesar disso, todas as características em cada um desses exemplos são *transmitidos* por meio do ambiente no formato *analógico*. No item 4.1 vimos que os sinais digitais são discretos e podem assumir apenas um valor entre um número finito de valores, ou seja, possuem uma resolução limitada. Em contraste, a eletrônica analógica trabalha com sinais contínuos que, teoricamente, possuem uma resolução infinita. As portas analógicas recebem, portanto, dados fornecidos por sensores analógicos, que detectam, entre outras coisas, mudanças físicas no ambiente como, por exemplo, sonoridade, luminosidade, temperatura, posição e pressão.

Os pontos positivos da utilização de sinais analógicos são relacionados, principalmente, a sua alta resolução e a simplicidade de processamento. Em comparação com os sinais digitais, eles são mais densos, ou seja, possuem maior quantidade de informações, e podem ser processado diretamente por componentes analógicos. A principal desvantagem é sua susceptibilidade a ruído, que está presente em qualquer sistema, e a dificuldade de recuperar os dados danificados pela interferência. Os próximos tópicos da seção 4.2 apresentarão exemplos de circuitos para obtenção e interpretação de valores obtidos por sensores analógicos.

A alta susceptibilidade dos sinais analógicos à ruídos elétricos e distorções produzidos, principalmente, por equipamentos de telecomunicações e rotinas de processamento de sinal fazem com que a quantidade final de ruído seja superior a de dados de interesse. Portanto, os sistemas de processamento analógico de sinal se tornam muito complexos e em algumas situações a degradação do sinal final torna o resultado obtido com um sistema digital superior. Isso ocorre pois, ao realizar uma conversão analógica-digital (A/D), o único ruído que está presente é o ruído de quantização, definido pela resolução finita do seu sistema digital. Logo, basta utilizar um sistema que forneça a resolução desejada na aplicação. Nas etapas seguintes do processamento, não há adição de mais ruídos.

O presente tópico da apostila tem como objetivo apresentar apenas como realizar a leitura de sinais analógicos em portas analógicas. Esses valores são utilizados em um sistema digital. Portanto, os sinais entram no microcontrolador na forma analógica, como uma tensão que varia de acordo com o sinal lido, e passam diretamente por um conversor A/D de modo que o valor lido e fornecido ao seu programa já é apresentado na forma digital. Sua precisão é determinada pela resolução do controlador.

A Arduino possui uma resolução é de **10 bits**, ou seja, o valor fornecido pelo sensor vai variar de **0** a **1023**. A tensão deve variar,

obrigatoriamente, de **0 V** a **5 V** pois, a Arduino não suporta tensões superiores a **5 V**. Para utilizar uma tensão superior a **5 V** é necessário utilizar um regulador de tensão, que reescale(**encontrar palavra melhor**) a tensão de **0 V** a **5 V**. A interpretação do valor obtido pelo sensor após passar pelo conversor A/D passa pelo seguinte processo:

- 1) (automático) O valor de tensão, que deve variar de **0 V** a **5 V**, lido na porta analógica passa por um conversor A/D.
- 2) O resultado digital obtido após a conversão A/D pode ser lido por uma variável como, por exemplo, **analog\_value**, utilizando a função `analogRead()`. Esse valor vai variar de **0** a **1023**.
- 3) O valor de **0** a **1023** é equivalente a tensão de **0 V** a **5 V**.
- 4) Para converter o valor digital para a grandeza que deseja medir utilize a seguinte fórmula:

$$\text{valor\_desejado} = (\text{Vref} \cdot \text{analog\_value} \cdot \text{val\_datasheet}) / \text{resolução}$$

- **valor\_desejado**: Valor lido pelo sensor apresentado já na unidade que você deseja, como por exemplo, Celsius para um sensor de temperatura.
- **Vref**: Tensão máxima de entrada no microcontrolador em volts (**5 V** no caso da Arduino)
- **analog\_value**: Valor analógico fornecido pela função de leitura do valor da porta analógica após passar pelo conversor A/D (estará entre **0** e a resolução máxima do microcontrolador.)
- **val\_datasheet**: Valor de referência para uma medida conhecida. Geralmente, encontra-se em *datasheet*. No caso do sensor de temperatura LM35, sabe-se que **val\_datasheet = 100** que relaciona **1 V** com **100** graus Celsius. Quando não houver um valor exato coloque como **1** e consulte um gráfico de referência no *datasheet*.
- **Resolução**: Resolução do microcontrolador. No caso da arduino, são **10 bits**, ou seja **1024**.

A rotina até o passo 3 ocorrerá sempre idêntica, mas a fórmula do item 4 pode ser alterada de acordo com os componentes utilizados. Os fundamentos teóricos por trás dessa rotina podem ser encontrados em mais detalhes em livros de sistemas digitais e analógicos. Os valores para cada sensor e microcontrolador são encontrados, normalmente, em seus *datasheets*.

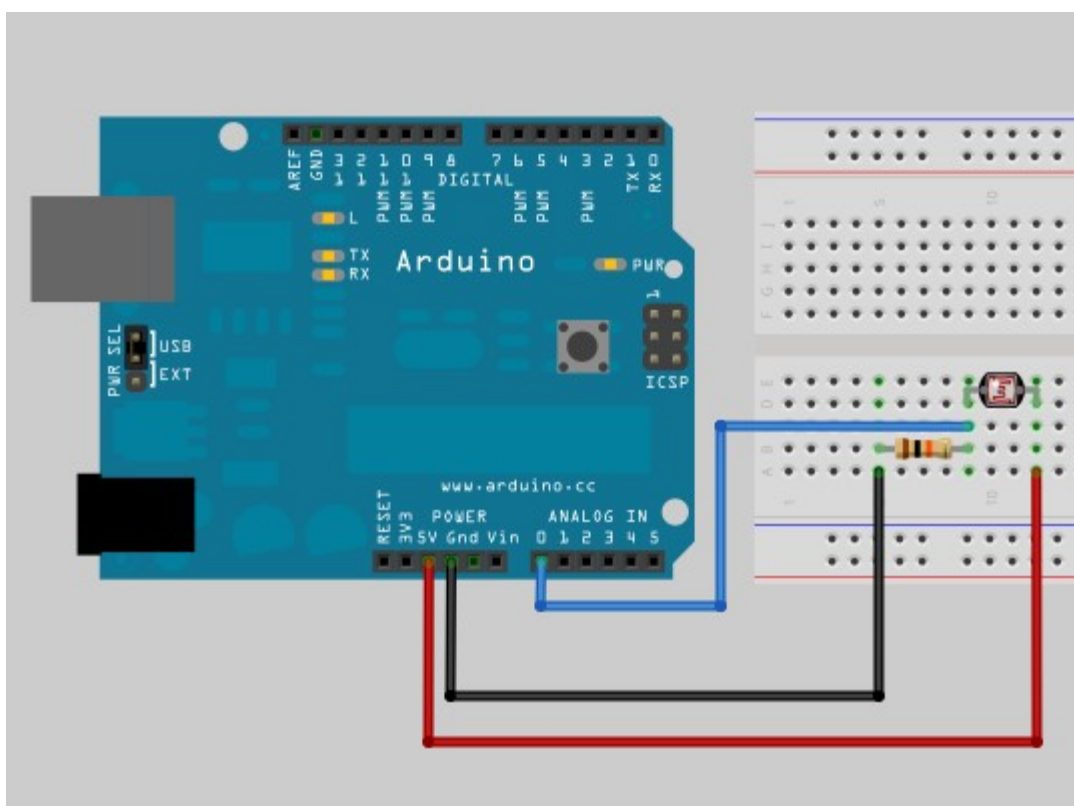
#### 4.2.1 Sensor de luminosidade

A intensidade da luminosidade em um ambiente é uma característica medida por meio de um sensor analógico. Os dois tipos clássicos de aplicação são a verificação da luminosidade em um ambiente determinando se ele está claro ou escuro e a detecção de objetos que passam na frente do sensor. A segunda aplicação se aproveita do fato de que quando um objeto passa na frente de um sensor, ele interfere na trajetória dos raios de luz que chegam ao sensor.

O processo de medição pode ser realizado de maneira simples com a utilização de um *light dependent resistor*(LDR). A resistência do LDR varia de acordo com a luminosidade. Os terminais do sensor, quando ligados a um circuito, como por exemplo o da figura 4.2-1, produzem uma variação na

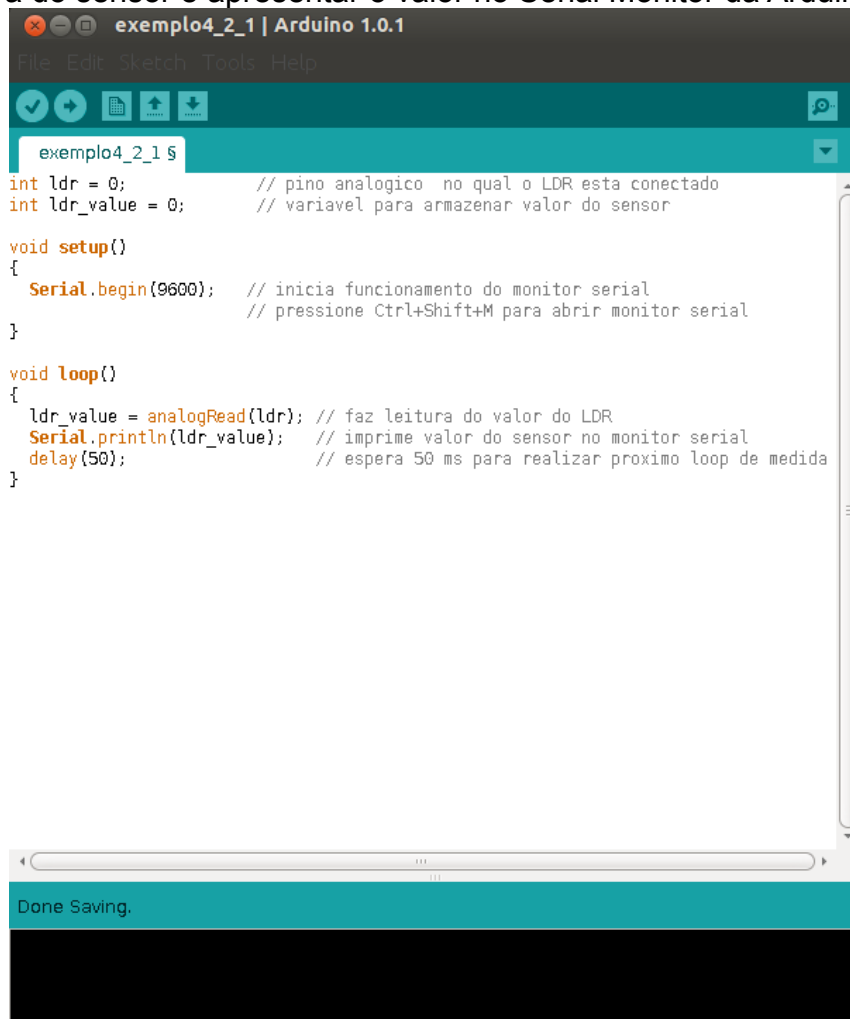
tensão na entrada de uma porta analógica. Essa tensão é convertida para um valor digital e interpretada pelo programa gravado na Arduino.

O circuito que demonstra como utilizar o LDR pode ser utilizado para qualquer sensor que sofre alteração em sua resistência de acordo com um fenômeno físico. A montagem do circuito apresentada na figura 4.2-1 fornecerá apenas parte da gama de valores entre 0 e 1023 (para a Arduino), obtidos a partir da conversão A/D, pois, a tensão não variará de **0 V** a **5 V**. Isso ocorre devido as quedas de tensão através de cada resistor. Logo, é necessário verificar os valores que o circuito está lendo em cada situação. Em seguida, você pode determinar como o valor digital será convertido para a escala que você precisa de acordo com a fórmula apresentada na introdução da seção 4.2.



*Figura 4.2-1 – Circuito para utilização de LDR com resistor de 10kOhm*

O código apresentado na figura 4.2-2 pode ser utilizado para fazer a leitura do sensor e apresentar o valor no Serial Monitor da Arduino.



```
exemplo4_2_1 | Arduino 1.0.1
File Edit Sketch Tools Help

exemplo4_2_1 $
int ldr = 0;           // pino analogico no qual o LDR esta conectado
int ldr_value = 0;     // variavel para armazenar valor do sensor

void setup()
{
  Serial.begin(9600);  // inicia funcionamento do monitor serial
                      // pressione Ctrl+Shift+M para abrir monitor serial
}

void loop()
{
  ldr_value = analogRead(ldr); // faz leitura do valor do LDR
  Serial.println(ldr_value);   // imprime valor do sensor no monitor serial
  delay(50);                  // espera 50 ms para realizar proximo loop de medida
}
```

Done Saving.

*Figura 4.2-2 – Código para leitura de valor do sensor LDR*

## 4.2.2 Sensor de Distância

A medição da distância até um determinado objeto ou a aproximação de algo em direção a seu sensor é realizada por meio de um sensor analógico. O processo pode ser feito com a utilização de um sensor



ultrasônico.

### Sensor Ultrasônico

O sensor ultrasônico adotado como exemplo será o Parallax PING. Ele mede a distância de um objeto que se encontra de 2 cm a 3 m de distância. A emissão do pulso sonoro ocorre quando o pino SIG do sensor é colocado no nível *HIGH* após dois microsegundos no nível *LOW*. O sensor gera um pulso na saída SIG até que o pulso sonoro emitido retorne ao sensor. A largura do pulso gerado é proporcional a distância percorrida pelo pulso sonoro e a partir disso é possível descobrir a distância do objeto.

O circuito apresentado na figura 4.2-2 mostra como utilizar o sensor com a Arduino. O código imediatamente após a figura pode ser utilizado para testar o sensor e visualizar o resultado no *Serial Monitor* da Arduino. Os comentários no código permitem entender como gerar os pulsos e converter a largura de pulso para distância em centímetros.

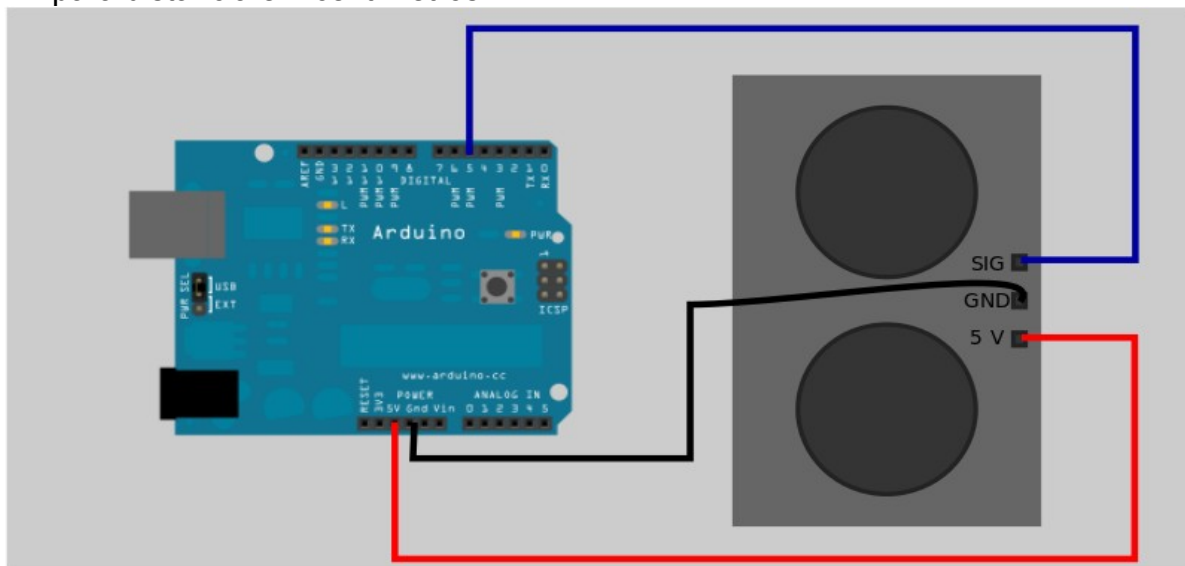


Figura 4.2-2 – Circuito para utilização de sensor de distância Parallax PING)))

```
// código para utilização do sensor Parallax PING)))

const int pingPin = 5; // pino no qual está ligado a saída de sinal do
                        // sensor
const int ledPin = 13; // pino para ligar LED que piscara mais
                        // rapidamente a medida que um objeto se aproxima

void setup()
{
  Serial.begin(9600); // inicia serial
  pinMode(ledPin, OUTPUT); // define pino do LED como OUTPUT
}

void loop()
{
  int cm = ping(pingPin); // declara variavel para emitir pulso
                           // e armazenar distancia
                           // utiliza funcao ping() para emitir
                           // pulsos e obter distancia
}
```

```

Serial.println(cm);           // imprime valor da distancia
digitalWrite(ledPin, HIGH);   // acende LED
delay(cm * 10);               // cada centimetro adiciona um delay
                                // de 10 ms para o LED ficar aceso

digitalWrite(ledPin, LOW);     // apaga LED
delay( cm * 10);               // delay para LED se manter apagado
}

// funcao pin() retorna a distancia do objeto em cm
int ping(int pingPin)
{
    // Estabelece variaveis para a duracao do ping
    // para a distancia em polegadas e centimetros
    long duration, cm;

    // O PING))) e ativado por um pulso HIGH de 2 micro segundos
    // Emite-se um pulso LOW para garantir um pulso HIGH limpo
    pinMode(pingPin, OUTPUT);
    digitalWrite(pingPin, LOW);
    delayMicroseconds(2);
    digitalWrite(pingPin, HIGH);
    delayMicroseconds(5);
    digitalWrite(pingPin, LOW);
    pinMode(pingPin, INPUT);
    duration = pulseIn(pingPin, HIGH);
    // converte o tempo em distancia
    cm = microsecondsToCentimeters(duration);
    return cm ;
}

long microsecondsToCentimeters(long microseconds)
{
    // A velocidade do som eh 340 m/s ou 29 microsegundos por centimetro
    // O ping percorre o caminho de ida e volta,
    // logo para determinar a distancia do objeto,
    // utilizamos metade da distancia
    return microseconds / 29 / 2;
}

```

#### 4.2.3 Sensor de Temperatura

A monitoração da temperatura de um ambiente ou objeto é de grande importância quando se deseja manter determinada temperatura. Isso pode ser feito através do clássico sensor analógico LM35. Ele produz uma tensão analógica diretamente proporcional a temperatura com uma resolução de 1 mV para 0,1 grau celsius.

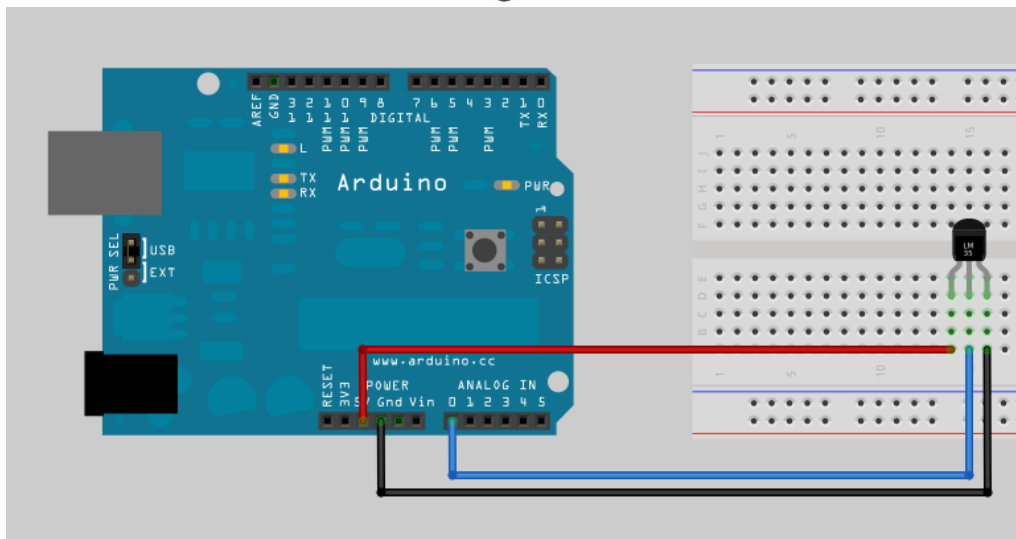


Figura 4.2-4 – Circuito para leitura de temperatura com LM35

A montagem visualizada na figura 4.2-4 apresenta como realizar a ligação do LM35 na Arduino. O código imediatamente abaixo permite entender como aplicar a rotina apresentada no início da seção 4.2. Ele apresenta o valor da temperatura em Celsius no *Serial Monitor* da Arduino.

```
// código para obtenção de temperatura com LM35
const int inPin = 0; // pino analógico utilizado para
// fazer leitura do LM35

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

void loop()
{
  int value = analogRead(inPin); // realiza leitura do valor do LM35
  Serial.print(value); Serial.print(" > "); // imprime valor obtido da
// porta analógica

  float millivolts = (value / 1024.0) * 5000; // imprime tensão lida na
// porta analógica

  float celsius = millivolts / 10; // a saída do sensor
// eh 10mV por grau Celsius

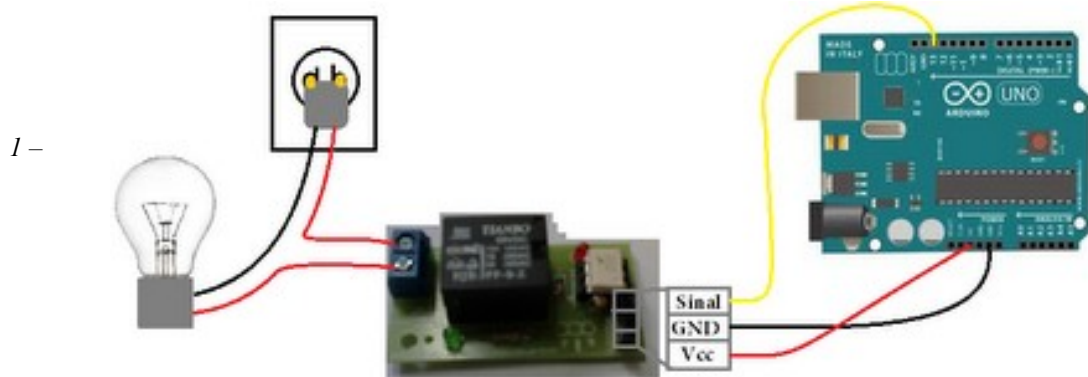
  Serial.print(celsius); // imprime temperatura em Celsius
  Serial.print(" degrees Celsius, ");
  Serial.print( (celsius * 9) / 5 + 32 ); // imprime temperatura em
// Fahrenheit
  Serial.println(" degrees Fahrenheit");
  delay(1000); // wait for one second
}
```

### 4.3 Atuadores

Atuadores, assim como sensores, transformam uma forma de energia em outra. No entanto, ao contrário dos sensores, os atuadores transformam um sinal elétrico em uma grandeza física como o movimento de um motor.

Um exemplo importante de atuador é o relé, que ao receber a corrente gera um campo magnético através de uma bobina. Esse campo irá provocar a atração entre um contato e permitir a passagem de energia no circuito.

Um exemplo de aplicação com o relé é acender uma lâmpada. Fazendo a montagem da figura 4.3-1 e usando o código de exemplo da figura 4.3-2 podemos



fazer a lâmpada piscar.

Figura 4.3- Montagem

esquemática da ligação de um módulo relé

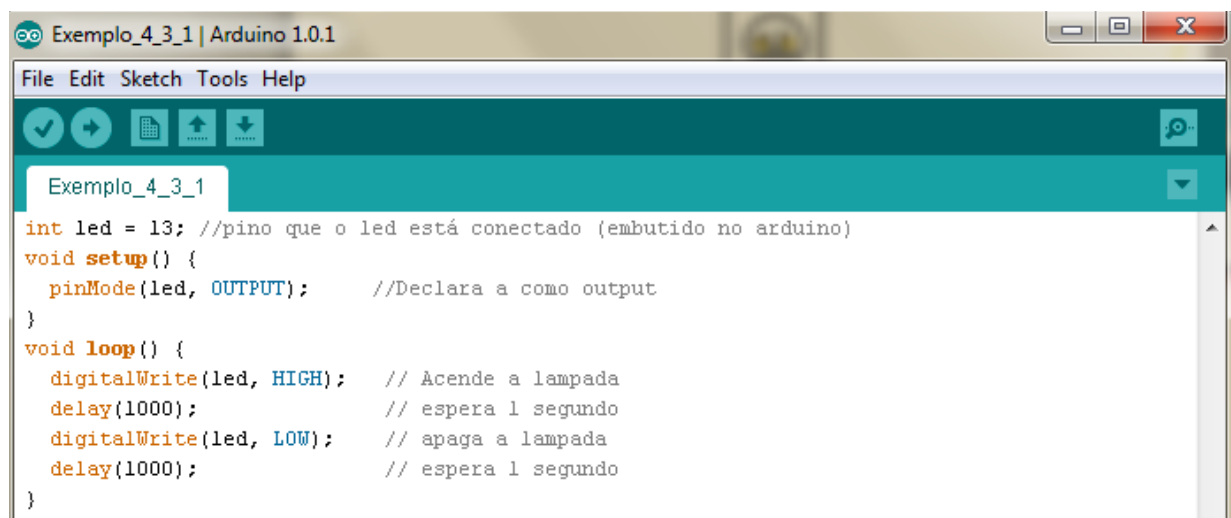


Figura 4.3-2 – Código de exemplo para a lâmpada piscar

Outro exemplo de atuadores são os motores, esses serão tratados mais a frente.

#### 4.4 PWM – Pulse Width Modulation

O arduino, como é um aparelho eletrônico, funciona a partir de pulsos elétricos representando sinais digitais. Ele portanto não tem a capacidade de emitir sinais analógicos reais, o que esse aparelho faz é simular um sinal digital a partir de uma técnica chamada de PWM ou Pulse Width Modulation (Modulação de largura do pulso).

Para entender o PWM vamos pensar em uma série de pulsos que fiquem metade do tempo em HIGH e metade em LOW, como mostrado na figura 4.4-1.

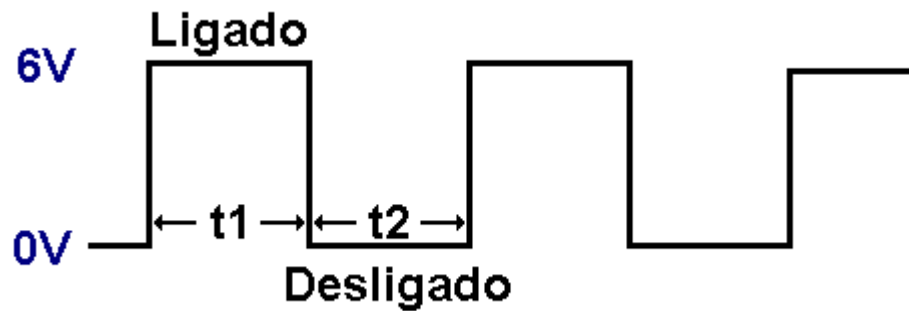


Figura 4.4-1 – Pulsos exemplificando o funcionamento do PWM

Como metade na metade do tempo temos o pulso em HIGH (6V no caso) e na outra metade temos o pulso em LOW (0V) podemos calcular a tensão média como 3V. Se diminuirmos a frequência dos pulsos, no entanto, teremos uma diminuição da tensão média. Como mostra a figura 4.4-2 temos o pulso em HIGH 30% do tempo, então a tensão será 30% de 6V, ou seja, ela cairá para 1,8V.

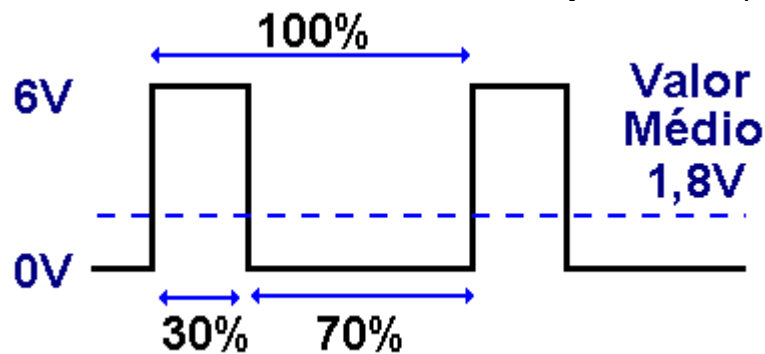
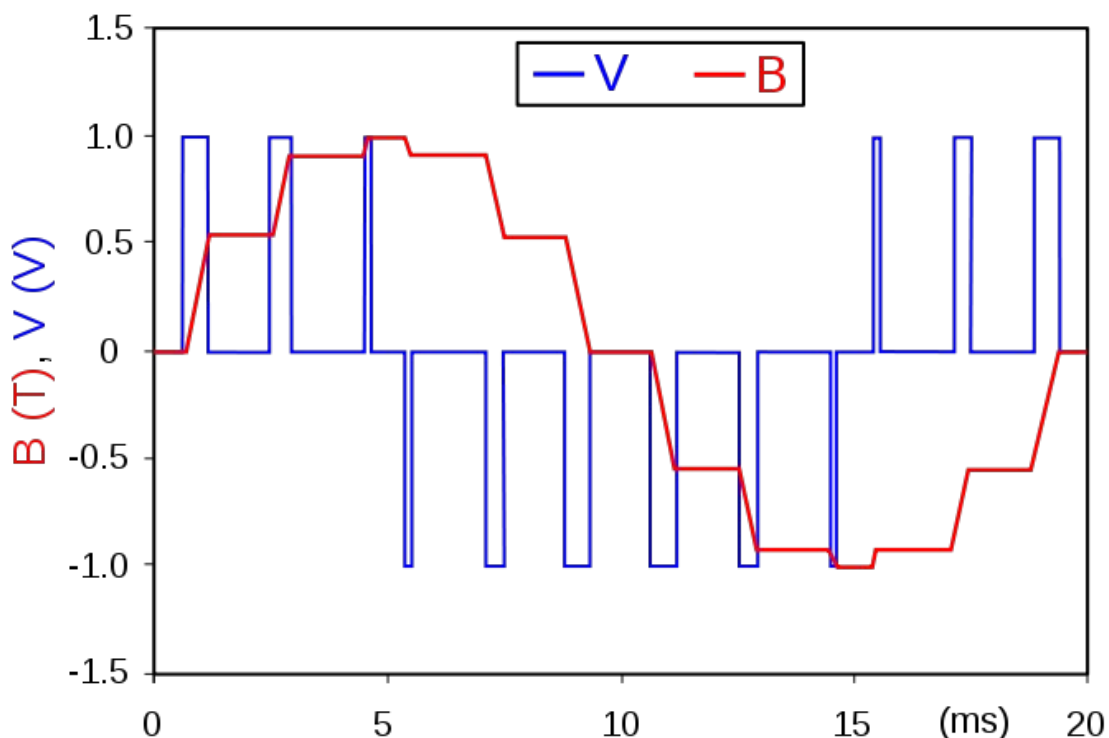


Figura 4.4-2 – Pulsos exemplificando uma diminuição do valor simulado.

Se, por outro lado, aumentarmos a frequência, a tensão irá subir do mesmo modo que ela caiu no exemplo anterior. Se colocarmos um valor HIGH constante teremos a tensão máxima.

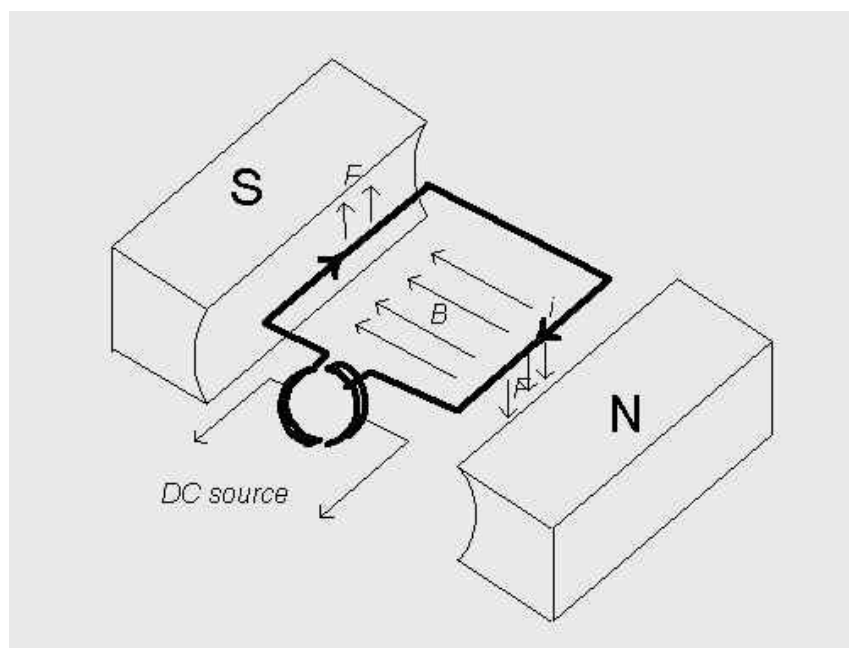
A partir desse método é possível modular um sinal analógico, fazendo a saída escrita pelo arduíno simular qualquer valor entre o máximo e o mínimo. A figura 4.4-3 mostra um sinal modulado pelo PWM.



*Figura 4.4-3 – Modulação de um sinal Analógico pelo PWM*

## 5. Controle de Motores

Os atuadores que realizam a maior parte dos trabalhos que exigem movimentação são os motores. A presente apostila focará, exclusivamente, nos motores elétricos devido a interação relativamente simples com microcontroladores. Os motores elétricos consistem basicamente em uma espira de fio de cobre percorrida por corrente imersa em um campo eletromagnético como pode ser visto na figura 5.1. De acordo com o tipo de aplicação são realizadas as devidas modificações e adaptações em cima da montagem.



*Figura*

*5.1 – Esquema de motor elétrico simples*

Os princípios físicos do funcionamento do motor podem ser encontrados em detalhes em um livro sobre eletricidade e magnetismo. As etapas envolvidas no processo de projetar, construir e operar os diferentes tipos podem ser encontrados em livros sobre conversão eletromecânica de energia. Nas próximas seções serão



apresentados os tipos mais conhecidos de motores elétricos e como realizar seu controle.

### 5.1 Tipos de Motores

Os objetos e máquinas podem ser movidas com a utilização de motores, que podem ser controlados por um microcontrolador. Os diferentes tipos de motores variam de acordo com a aplicação.

#### Servo Motor



O servo motor, que pode ser visto na figura 5.1-1 ao lado, é composto por um pequeno motor elétrico que transmite movimento de rotação através de engrenagens para um eixo de saída. Este eixo é conectado a um potenciômetro que fornece a informações sobre a posição para um circuito interno de controle. Portanto, os servos permitem que você controle o movimento de rotação de modo que ele se mantenha em uma posição específica ao invés de rodar continuamente.

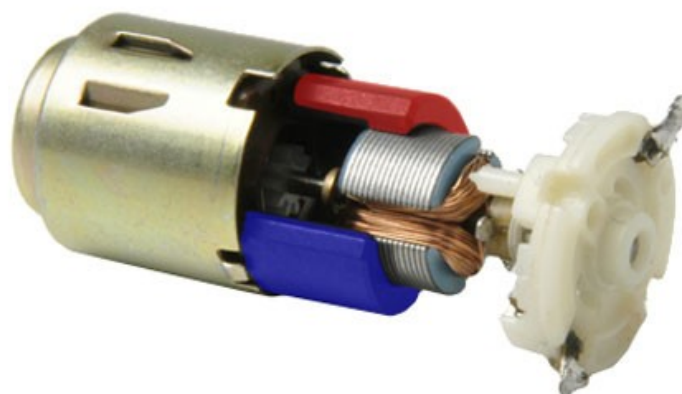
As aplicações dos servos são, principalmente, aquelas que exigem movimentos precisos de rotação. O controle e a conexão são simples de ser realizados, pois ele possui um driver interno de controle. Eles são ideais para fazer com que objetos rotacionem em um intervalo de 0 a 180 graus. Apesar disso, é possível obter rotação contínua ao desconectar o controle de posição que limita as posições.

O circuito de controle integrado no motor é feito de modo que ele responda a durações de pulsos de tensão ao configurar uma porta de saída em valor *HIGH*. A relação entre a duração dos pulsos e a posição e rotação do motor variam de acordo com o servo. Portanto, é necessário realizar testes com os comandos enviados pelo microcontrolador ao motor.

#### Motores Elétricos DC (Corrente Contínua)

A estrutura mais próxima do motor elétrico ideal apresentado na figura 5.1 é a de um motor elétrico DC visível na figura 5.1-2. Ele é um simples enrolamento de fio, normalmente, de cobre posicionado entre ímãs nas mais diversas variações. O motor possui dois contatos por onde a corrente passa para percorrer o fio e gerar um campo eletromagnético que interage com o dos ímãs provocando movimento. O sentido de rotação depende do sentido em que a corrente é aplicada nos contatos.

Os motores elétricos DC estão disponíveis em vários tamanhos. As suas dimensões e o material utilizado para o fio influenciam na corrente exigida pelo motor. Portanto, até os menores, como por exemplo, motores que realizam a vibração em aparelhos celulares, necessitam de transistores ou outro componente elétrico para fornecer a corrente adequada.



*Figura 5.1-2 – Motor DC e estrutura interna. É possível ver as espiras e os ímãs que realizam a rotação do motor, semelhantemente, a figura 5.1.*

A estrutura do motor pode conter um elemento a mais de acordo com a aplicação. Este componente é a escova, que reduz a complexidade do controle do motor. Apesar disso, quando ela não está presente, os motores são mais fortes e eficientes. Nessa breve descrição é importante apenas saber da existência desses dois tipos de montagem e que caso deseje-se uma alta performance pode-se utilizar um circuito eletrônico de controle de velocidade aplicado em radio controle. A Arduino pode fazer o uso de tal componente e obter um resultado próximo ao de um servo-motor.

### **Motor de Passo**

A configuração da estrutura do motor de passo, que pode ser vista na figura 5.1-3, seria semelhante a um eixo com um conjunto de ímãs ou eletroímãs rodeado por bobinas atuando como eletroímãs. O seu princípio de funcionamento faz o uso dessa montagem para girar uma quantidade específica de graus em resposta à ativação dos eletroímãs. Ele é chamado motor de passo pois a ativação de cada uma das bobinas em determinada sequência faz com que ele gire uma quantidade fixa de graus, ou seja, ele se move em passos iguais.

O número de graus que ele gira em cada passo varia de motor para motor. Os valores podem variar de 1 ou 2 graus até 30 ou mais por passo. A rotação contínua ocorre quando os eletroímãs são ativados em determinada sequência contínua.

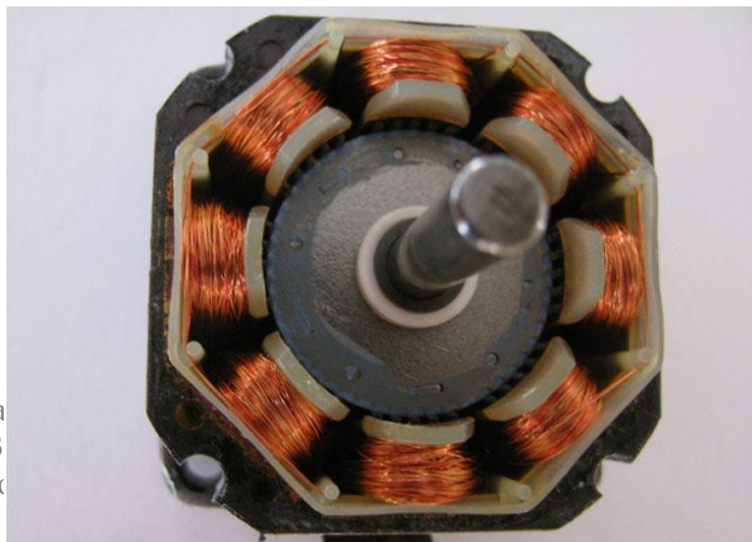


Figura 5.1-3 – Motor de Passo

Os tipos mais comuns de motores de passo utilizados com a Arduino são os bipolares e os unipolares. Os primeiros possuem, tipicamente, quatro fios para serem ligados ao circuito de controle. Os unipolares fazem o uso de seis fios. A maior complexidade de controle e seu preço de ambos os tipos os tornam menos comuns em textos básicos de microcontroladores.

## 5.2 Ponte H – Controle de motor DC

As aplicações de motores DC em um projeto, provavelmente, exigirão que ele rode nos dois sentidos. Segundo seu princípio de funcionamento, ele rodará nos sentidos horário e anti-horário de acordo com o sentido em que a corrente o percorre. Portanto, para realizar o controle do sentido de rotação é necessário um circuito que permita a passagem da corrente em ambos os sentidos. A solução para esse problema é a utilização de uma Ponte H que, além de realizar o controle da direção, permite o controle da velocidade do motor.

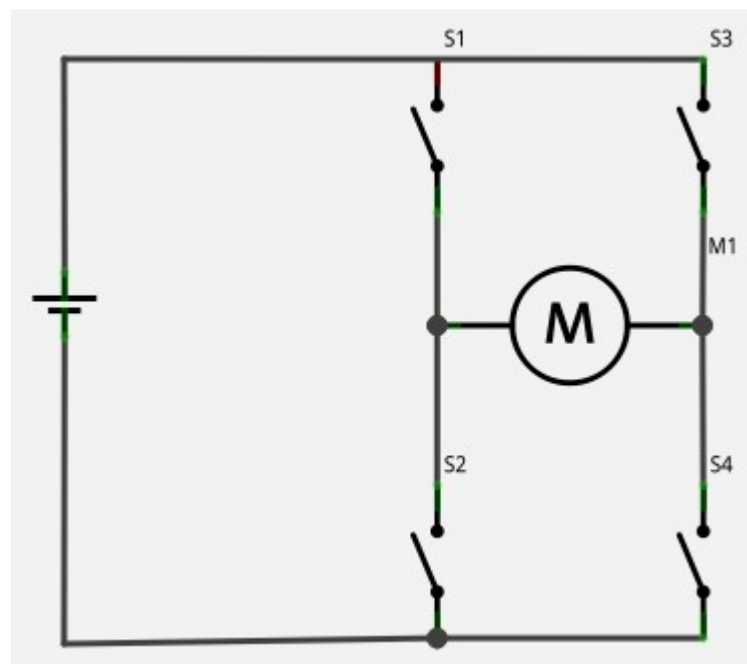


Figura 5.2-1 -

Ponte H controlando motor M

Tabela 5.2-1 - Tabela de operação da Ponte H. Os “0” indicam que a chave está aberta e os “1” indicam chave fechada.

S1	S2	S3	S4	Resultado
1	0	0	1	Gira Horário

0	1	1	0	Gira Anti-Horário
0	0	0	0	Gira Livremente
0	1	0	1	Freio
1	0	1	0	Freio
1	1	0	0	<b>Curto-Circuito</b>
0	0	1	1	<b>Curto-Circuito</b>
1	1	1	1	<b>Curto-Circuito</b>

O circuito apresentado na figura 5.2-1 mostra uma construção simples de uma Ponte H utilizando chaves liga-desliga, um motor e uma fonte de tensão. A tabela 5.2-1 mostra como utilizar a ponte para controlar o sentido de rotação do motor. Os modos de operação demonstrados na tabela 5.2-1 permitem perceber, intuitivamente, o sentido de percorrimto da corrente pelo motor.

Os modo de freio faz com que o motor pare bruscamente e o modo de Giro Livre desliga o motor, mas permite que ele continue girando até parar. É **importante** ressaltar os estados que contém **curto-circuito** como resultado. Nessa configuração os terminais da fonte que alimenta o motor estão fechando um circuito entre si, o que pode danificar a fonte e, eventualmente causar explosões.

A partir de agora sabemos o princípio de funcionamento da Ponte H e vamos abordar um exemplo que realiza o controle do sentido de rotação do motor com uma Ponte H e uma Arduino. Os estados de curto-circuito apresentados na tabela 5.2-1 podem causar diversos problemas em um circuito, portanto, as Pontes H costumam ser vendidas prontas implementadas em um circuito integrado(CI) com a correção desse problema. O CI mais comum para essa aplicação é o L293D.

A figura 5.2-2 apresenta a montagem do circuito de controle do sentido de rotação e velocidade de um motor DC. O código de controle do programa pode ser visualizado imediatamente abaixo.

```
const int enPin = 5;      // porta digital ligada ao ENA da L293D
const int in1Pin = 7;    // porta digital ligada ao IN1 da L293D
const int in2Pin = 4;    // porta digital ligada ao IN2 da L293D
void setup()
{
    Serial.begin(9600);
    pinMode(in1Pin, OUTPUT);
    pinMode(in2Pin, OUTPUT);
    Serial.println("Speed (0-9) or + - to set direction");
}
void loop()
{
    if ( Serial.available())
    {
        char ch = Serial.read();    // leitura de dígito inserido no
```

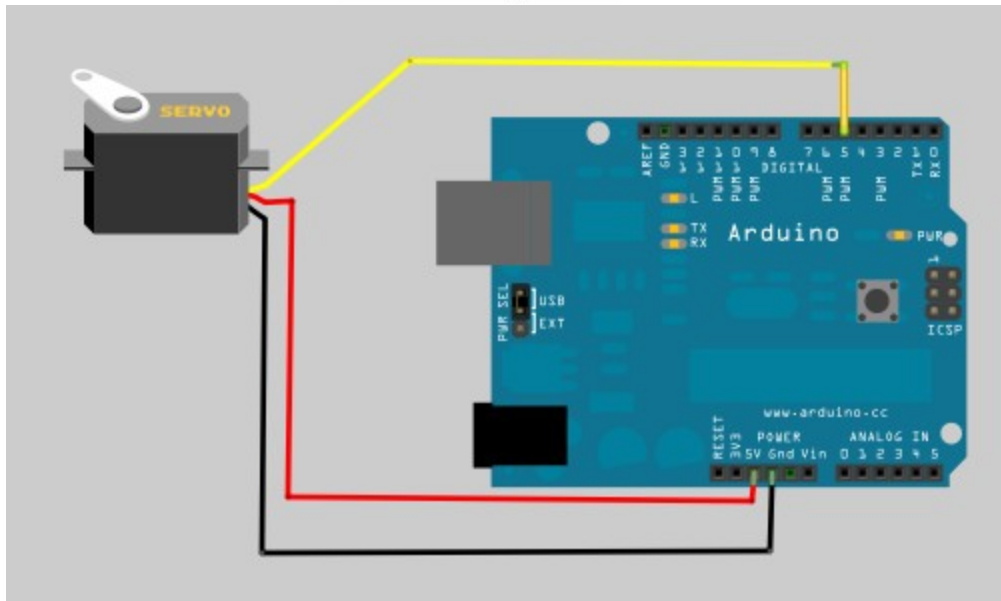
```
        // serial monitor
        if(ch >= '0' && ch <= '9')
        {
            // reescala velocidade para enviar a ponte H
            int speed = map(ch, '0', '9', 0, 255);

            analogWrite(enPin, speed);
            Serial.println(speed);
        }
        else if (ch == '+') // define sentido de rotacao horario
        {
            Serial.println("CW");
            digitalWrite(in1Pin, LOW);
            digitalWrite(in2Pin, HIGH);
        }
        else if (ch == '-') // define sentido de rotacao anti horario
        {
            Serial.println("CCW");
            digitalWrite(in1Pin, HIGH);
            digitalWrite(in2Pin, LOW);
        }
        else
        {
            Serial.print("Unexpected character ");
            Serial.println(ch);
        }
    }
}
```

### 5.3 Controle de Servo Motor

O controle da posição de um servo motor utilizando uma Arduino pode ser feito de maneira simples ao utilizar funções da biblioteca Servo.h . A montagem consiste em, simplesmente, conectar a alimentação e o terra do servo-motor em uma fonte de energia e o pino de controle do servo em uma porta digital da arduino. Os servos pequenos que são alimentados por 5V podem ser ligados no 5V do Arduino.





*Figura 5.3-1 Controle de Posição de Servo com Arduino*

A ligação do servo motor na arduino pode ser vista na figura 5.3-1. O código de controle pode ser visualizado na Figura 5.3-2. É importante sempre observar as cores dos fios de conexão do servo com suas funções, pois as cores podem variar de acordo com a marca.



```

exemplo5_3_1 | Arduino 1.0.1
File Edit Sketch Tools Help
exemplo5_3_1 $
#include <Servo.h> // inclui biblioteca para controle de servo motor

Servo myservo; // declara objeto do tipo servo para controlar servo motor
               // numero maximo de servos que podem ser utilizados: 8

int pos = 0;   // variavel inteira armazena posicao do servo motor

void setup()
{
  myservo.attach(5); // associa o servo motor ligado no pino digital 5 ao objeto myservo
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1) // varia posicao do servo de 0 a 180 graus em passos de 1 grau
  {
    myservo.write(pos);              // faz com que o servo se posicione em "pos" graus
    delay(15);                       // espera 15ms para o servo se posicionar
  }
  for(pos = 180; pos>=1; pos-=1)    // varia posicao do servo de 180 a 0 graus em passos de 1 grau
  {
    myservo.write(pos);              // faz com que o servo se posicione em "pos" graus
    delay(15);                       // espera 15ms para o servo se posicionar
  }
}
Done Saving.

```

Figura 5.3-2 – Código do programa para controle de servo motor

## 6. Controle de tomadas

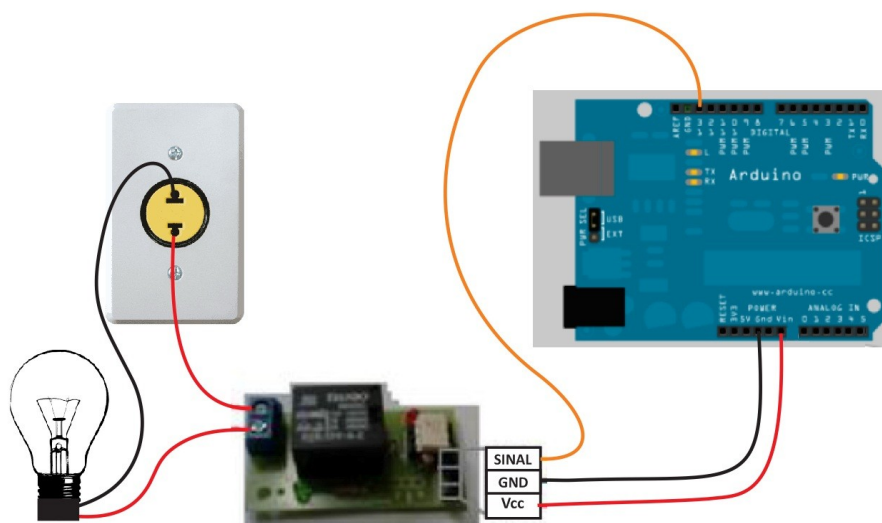
Em muitos projetos eletrônicos precisamos realizar chaveamento de circuitos de altas tensões. Neste curso, portanto, nossa intenção seria chavear um circuito que necessita de altas tensões utilizando o Arduino. O problema é que temos então tensões de saída de até 5V DC. Portanto, como poderíamos controlar uma tomada utilizando o Arduino, por exemplo? O controle de tomadas é um tópico essencial na automação residencial e envolve a aplicação de relés ou módulos relés.

### 6.1 Acionando 110V/220V com sinais de 5V

No controle de tomada, a questão principal é utilizar um sinal de 5V para acionar 110V/220V. O componente eletromecânico, acionado pelos 5V, que permite o chaveamento do circuito, e o consequente acionamento da tomada, é o relé (ver t

ópico 2.4). Dependendo da tensão de acionamento do relé, 'necessitamos de um transistor que permite a passagem de uma corrente mais alta do que a corrente de saída do Arduino (os relés são acionados por correntes de no mínimo 40 mA).

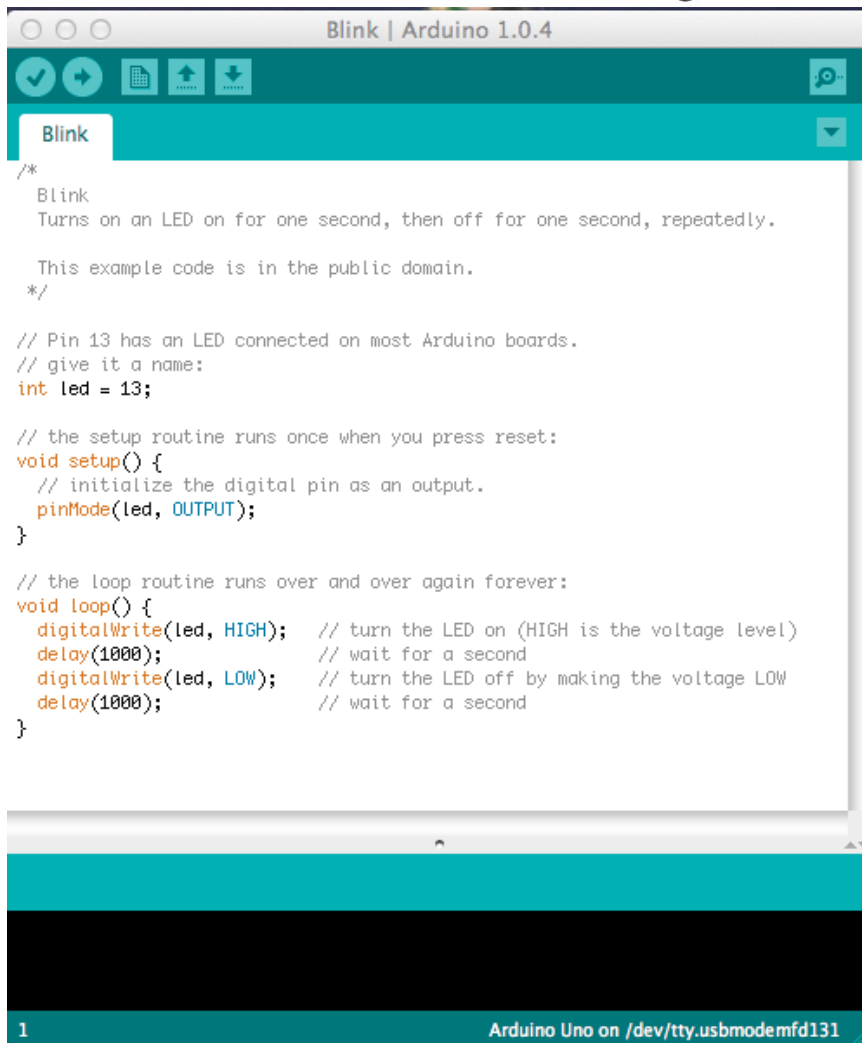
Nos kits disponíveis, temos um módulo relé que é composto por três terminais de entrada e dois terminais de saída. Na entrada temos um sinal terra (GND, fio preto), um sinal de alimentação Vcc (Vin, fio vermelho) e um sinal de controle (fio laranja). Os sinais de saída são conectados na tomada e na lâmpada em série, como ilustrado na Figura 6.1-1. O acionamento se dá pelo fio laranja (sinal de controle) de modo que quando em HIGH o relé será chaveado e a lâmpada irá acender, e quando em LOW a lâmpada ficará apagada. Veja como se dá a montagem abaixo.



**Figura 6.1 - Circuito ilustrativo do controle de tomada**

## 6.2 Implementação

Agora que já conhecemos o circuito, utilizamos o Blink (File > Examples > Basics > Blink ) para ligar e desligar a lâmpada. Agora você pode mudar o delay, acionar de outra maneira, etc.



**Figura 5.2 - Blink**

## Bibliografia:

Aprendendo a Programar em Arduino – Micael Bronzatti Gaier

Arduino CookBook – Michael Margolis

[www.arduino.cc](http://www.arduino.cc)