



APOSTILA
KIT INICIANTE V7.1
PARA ARDUINO

Parabéns por adquirir o Kit Iniciante para Arduino da RoboCore!

Este material é composto por 30 experimentos, que são intitulados módulos, projetos e exercícios. O intuito principal é que o usuário que está começando a entender o fascinante mundo da eletrônica, ou mesmo o usuário que já tem boas noções, possa começar a construir protótipos utilizando sua placa Arduino. Para efeitos de explicação, chamamos de **módulos** os experimentos que, por si só, não apresentam grande efeito e, quando juntados 2 ou mais módulos, podemos fazer um **projeto**, que consiste em algo útil ou ao menos agradável de ser apreciado.

Abaixo segue a lista de módulos e projetos que podem ser construídos com o auxílio deste material:

- **Módulo 1** pág. 19
Componentes: 1 Botão + 1 Led
Descrição: Conforme você pressiona um pushbutton, um led é aceso.
Dificuldade: 
- **Módulo 2** pág. 21
Componentes: 3 Botões + 3 Leds
Descrição: Conforme você pressiona qualquer um dos botões, leds de diferentes cores são acesos.
Dificuldade: 
- **Projeto Piano** pág. 23
Componentes: 3 Botões + 3 Leds + Buzzer
Descrição: Cada botão toca uma nota musical diferente e acende um led. É expansível – por conta do usuário – para mais uma nota musical com o botão (e o led) reserva
Dificuldade: 
- **Módulo 3** pág. 25
Componentes: 1 Sensor de Temperatura LM35
Descrição: Com o auxílio da porta serial e do monitor serial, o usuário irá fazer a leitura do sensor de temperatura em °C para fazer o projeto seguinte.
Dificuldade: 
- **Projeto Alarme** pág. 30
Componentes: 1 Sensor de Temperatura LM35 + 1 buzzer
Descrição: A partir dos valores lidos no módulo 3, o usuário poderá montar um alarme que, se a temperatura de onde o sensor estiver localizado for maior, ou menor, ele soará.
Dificuldade: 
- **Projeto Termômetro** pág. 32
Componentes: 1 Barra gráfica de LEDs + 1 Buzzer + 1 Sensor de Temperatura LM35
Descrição: Conforme a temperatura do ambiente onde o sensor LM35 está localizado aumenta, os leds da barra gráfica acendem, como um termômetro. Se existir uma temperatura muito alta, um alarme deverá soar.
Dificuldade: 
- **Módulo 4** pág. 36
Componentes: 1 Potenciômetro + 1 Led
Descrição: Conforme o valor do potenciômetro é alterado, o led pisca de forma mais rápida ou mais lenta
Dificuldade: 

▪ Projeto Dimmer

pág. 38

Componentes: 1 Potenciômetro + 1 Led Alto Brilho

Descrição: Conforme o valor do potenciômetro é alterado, o led fica mais claro ou mais escuro graças ao PWM

Dificuldade: **▪ Projeto Iluminação Automatizada**

pág. 42

Componentes: 1 Led Alto Brilho + 1 Sensor de Luminosidade LDR

Descrição: Se a iluminação ambiente, por qualquer motivo, diminuir ou apagar completamente, um led de alto brilho acende gradativamente

Dificuldade: **▪ Projeto Alarme Multipropósito**

pág. 46

Componentes: 2 Leds Verdes + 2 Leds Amarelos + 2 Leds Vermelhos + 1 Sensor de Luminosidade LDR + 1 Sensor de Temperatura LM35 + 1 Led Alto Brilho + 1 buzzer

Descrição: Temos dois indicadores: um de luminosidade e outro de temperatura, através das cores dos leds. Se a temperatura estiver alta deverá acender os 3 Leds que a corresponde. Se os 3 Leds correspondentes à luminosidade estiverem apagados – indicando uma falta total de luminosidade no ambiente - um alarme deverá soar e um led de alto brilho irá acender.

Dificuldade: **▪ Módulo 5**

pág. 49

Componentes: 01 Display de LCD + 01 Potenciômetro

Descrição: Aprenda como usar este dispositivo, muito útil para mostrar os dados internos da placa Arduino em um modo inteligível, ou seja, possível de seres humanos entenderem.

Dificuldade: **▪ Módulo 6**

pág. 52

Componentes: 01 Display de LCD + 01 Potenciômetro

Descrição: Aprenda a enviar caracteres do computador para o Arduino e vê-los no display.

Dificuldade: **▪ Projeto Freqüencímetro**

pág. 55

Componentes: 01 Display de LCD + 01 Potenciômetro + 01 Buzzer + 02 Botões


Descrição: Vamos fazer barulho novamente! Mas agora vamos variar a frequência e verificar a todo instante a quantos hertz estamos ouvindo o Buzzer tocar.

Dificuldade: **▪ Display de 7 Segmentos**

pág. 59

Componentes: 01 Display de 7 Segmentos + 01 CI 4511

Descrição: Aprenda a utilizar o display de 7 segmentos juntamente a um conversor BCD para display. Neste experimento será explicado os fundamentos básicos de um circuito integrado e contagem binária.


Dificuldade: **▪ Projeto RGB**


pág. 67

Componentes: 01 LED RGB

Descrição: Aprenda como utilizar o led RGB, que possui em um mesmo encapsulamento três leds com as cores primárias. Através de PWM você poderá ver diversas combinações de cores.

Dificuldade: 

- **Projeto Timer** *pág. 70*
Componentes: 01 Buzzer + 03 PushButtons + 01 Potenciometro + 01 LCD
Descrição: Aprenda como fazer um timer com seu Arduino, onde você seleciona o tempo e ouve um aviso sonoro quando o tempo se esgota.
Dificuldade: 

- **Projeto Theremin Agudo** *pág. 79*
Componentes: 01 Buzzer + 01 Sensor de Luminosidade LDR
Descrição: Aprenda como fazer outro instrumento musical, usando apenas dois componentes do kit.
Dificuldade: 

- **Exercícios** *pág. 80*
- **Resolução do Exercício 1** *pág. 81*
- **Resolução do Exercício 2** *pág. 81*
- **Resolução do Exercício 3** *pág. 82*
- **Resolução do Exercício 4** *pág. 82*
- **Resolução do Exercício 5** *pág. 83*
- **Resolução do Exercício 6** *pág. 86*
- **Resolução do Exercício Desafio** *pág. 87*
- **Mini Glossário** *pág. 90*

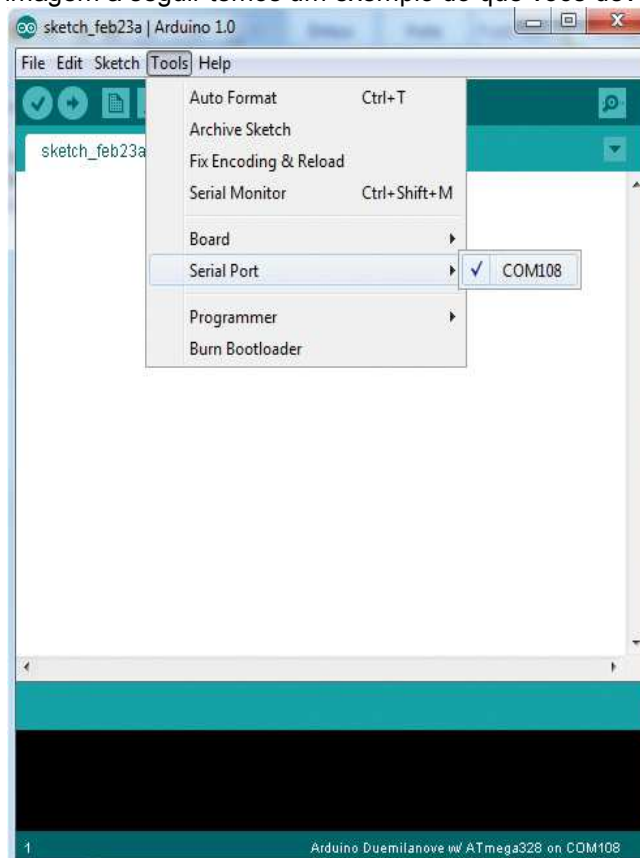
INSTALAÇÃO DO SOFTWARE ARDUINO

Existem procedimentos diferentes para instalação do ambiente de desenvolvimento (IDE) do Arduino em cada sistema operacional. Se você está usando Windows, basta acessar a pasta do CD chamada "Ambiente de Desenvolvimento (IDE)" e depois acessar a pasta Windows. Lá você encontra um arquivo chamado "*arduino-1.0.5-r2-windows.exe*". Para ter o programa funcionando corretamente em seu PC, basta copiar este arquivo para uma pasta no seu disco rígido (como C:\Arduino, criada por você) e clicar duas vezes no ícone copiado. Siga o procedimento de instalação e você poderá usar o programa sem problemas. No MacOSX o procedimento é um pouco diferente. Basta acessar a pasta do CD "Ambiente de Desenvolvimento (IDE) > MacOSX" e copiar o aplicativo "Arduino.app" para seu computador (sugerimos para a pasta de aplicativos pra ficar padronizado). No Linux talvez a maneira mais fácil é instalar via terminal, porém dentro do CD também pode ser encontrado os arquivos para instalação. Sugerimos sempre utilizar a IDE mais recente, a qual poderá ser encontrada em www.arduino.cc, site oficial do Arduino. Para abrir a IDE e começar a programar, basta clicar no ícone "**arduino**" no sistema operacional que estiver usando.

INSTALAÇÃO DO DRIVER DA PLACA NOS SISTEMAS OPERACIONAIS

O dispositivo Arduino é totalmente Plug & Play. Uma vez rodando o ambiente de desenvolvimento, insira o cabo USB AB no Arduino e depois no computador. Seu computador deverá reconhecer automaticamente o Arduino e uma nova porta de comunicação será criada (no caso de sistema operacional Windows será criada uma porta COM, no caso do Linux será criada uma porta do tipo /dev/ttyUSBx e no caso do MacOSX será criado algo como tty.usbmodem). Caso o sistema operacional não reconheça a placa automaticamente, veja nas páginas a seguir como proceder em cada caso.

Para selecionar esta nova porta de comunicação onde o Arduino está localizado, no ambiente de desenvolvimento clique em **TOOLS > SERIAL PORT > COM X** no Windows ou **TOOLS > SERIAL PORT > /dev/tty.usbmodem X** no Mac (onde X é o número da porta que o Arduino foi instalado automaticamente). Na imagem a seguir temos um exemplo do que você deverá ver:



Note que o número da porta COM não é necessariamente 108 como na imagem acima. Cada computador poderá mostrar um número de porta diferente.

Seu Arduino não está sendo reconhecido pelo **Windows 7**? Veja abaixo a solução:

Por causa de fatores ligados a permissões do sistema, o Windows 7 algumas vezes impede que o driver seja instalado de uma determinada pasta, onde estão os drivers e ambiente de desenvolvimento do Arduino. Desta forma, temos que fazer com que o Windows “force” a instalação destes drivers de alguma forma.

Siga os seguintes passos:

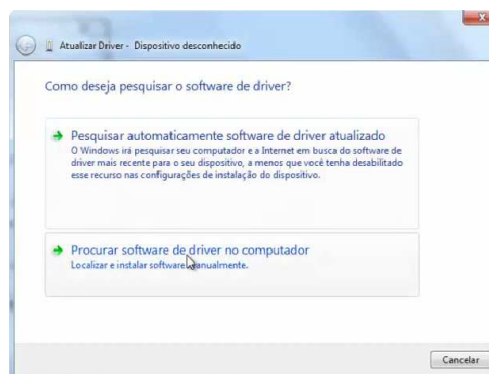
- 1) Conecte seu Arduino à porta USB de seu computador. Aguarde até aparecer a mensagem de erro de instalação de driver. A mensagem deve se parecer com a seguinte:



- 2) Feche esta mensagem. Clique em “Iniciar”  depois em “Dispositivo e Impressoras”. Você verá um dispositivo como “Não Especificado”, como mostra a figura abaixo:



- 3) Clique com o botão direito do Mouse neste “Dispositivo Desconhecido” e depois em Propriedades;
- 4) Clique na aba “Hardware” e depois em “Propriedades”;
- 5) Na nova janela, clique no botão “Alterar Configurações”;
- 6) Clique agora em “Atualizar Driver...”;
- 7) Na janela que abrir, clique em “Procurar Software de Driver no Computador”



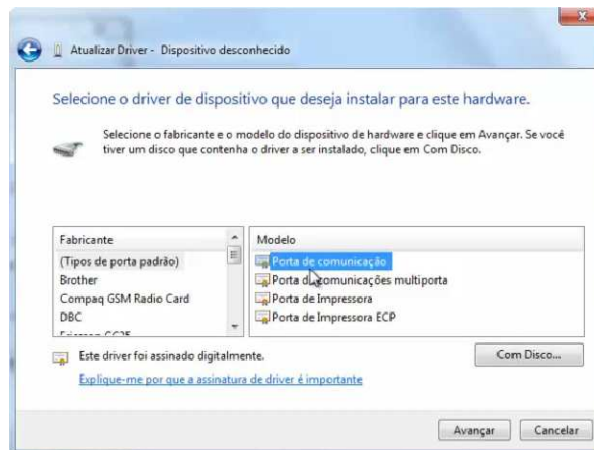
- 8) Neste ponto, não use a opção de seleção de diretório para escolher o driver do Arduino. Se você fizer isto, o Windows não irá permitir que o driver seja instalado, pois ele não tem

permissão para carregar o driver da pasta em questão. Clique então em “PERMITIR QUE EU ESCOLHA EM UMA LISTA DE DRIVERS E DISPOSITIVOS NO COMPUTADOR”, como na figura a seguir:

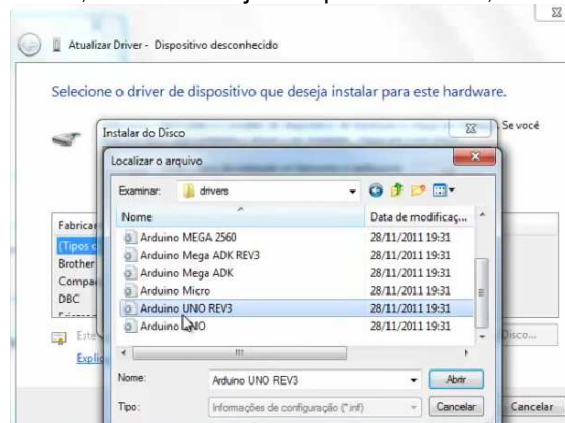


9) Na janela que abrir, role a lista para baixo até encontrar “Portas (COM e LPT)” e clique em Avançar;

10) Na próxima janela, selecione “Porta de comunicação”, como na figura abaixo e clique em “Com Disco...”:



11) Na janela que abrir, faça a busca do driver pelo botão “Procurar”. Direcione esta busca para a pasta DRIVERS, do ambiente de desenvolvimento Arduino, e dentro dela clique em “ARDUINO UNO REV3”, caso esta seja sua placa Arduino, conforme a figura abaixo:



12) Clique em “Abrir”, então em “Ok” e depois em “Avançar”;

13) Clique em “Instalar este software mesmo assim”;

14) Pronto! Seu Arduino está instalado e pronto para ser usado! Agora, basta selecionar a porta serial do mesmo no ambiente de desenvolvimento Arduino e usá-lo.

INSTALAÇÃO NO WINDOWS 8

No Windows 8, a instalação de drivers não assinados é um pouco diferente. Para instalar o driver corretamente, siga o seguinte procedimento:

ATENÇÃO: Antes de prosseguir, certifique-se que o você salvou todos os documentos que estão abertos em sua máquina. Pois sua máquina será reiniciada.

- 1 - Pressione as teclas "windows" e "R" simultaneamente.
- 2 - Copie e cole o seguinte comando: shutdown.exe /r /o /f /t 00
- 3 - Selecione "Troubleshoot"
- 4 - Selecione "Startup Settings"
- 5 - Selecione "Disable Driver Signature Enforcement"
- 6 - Instale novamente o driver do Arduino.

INSTALAÇÃO NO LINUX - Distribuição Ubuntu

A instalação dos drivers no Linux é muito fácil, basta entrar no terminal de comandos com a placa conectada ao computador, e digitar o seguinte:

```
$ sudo apt-get install arduino
```

INSTALAÇÃO NO LINUX - Distribuição Fedora 17 ou posterior

Digite o seguinte comando no terminal:

```
$ sudo yum install arduino
```

- **Nota para uso deste manual com a placa BlackBoard:**

O driver da placa BlackBoard pode ser encontrado na pasta:

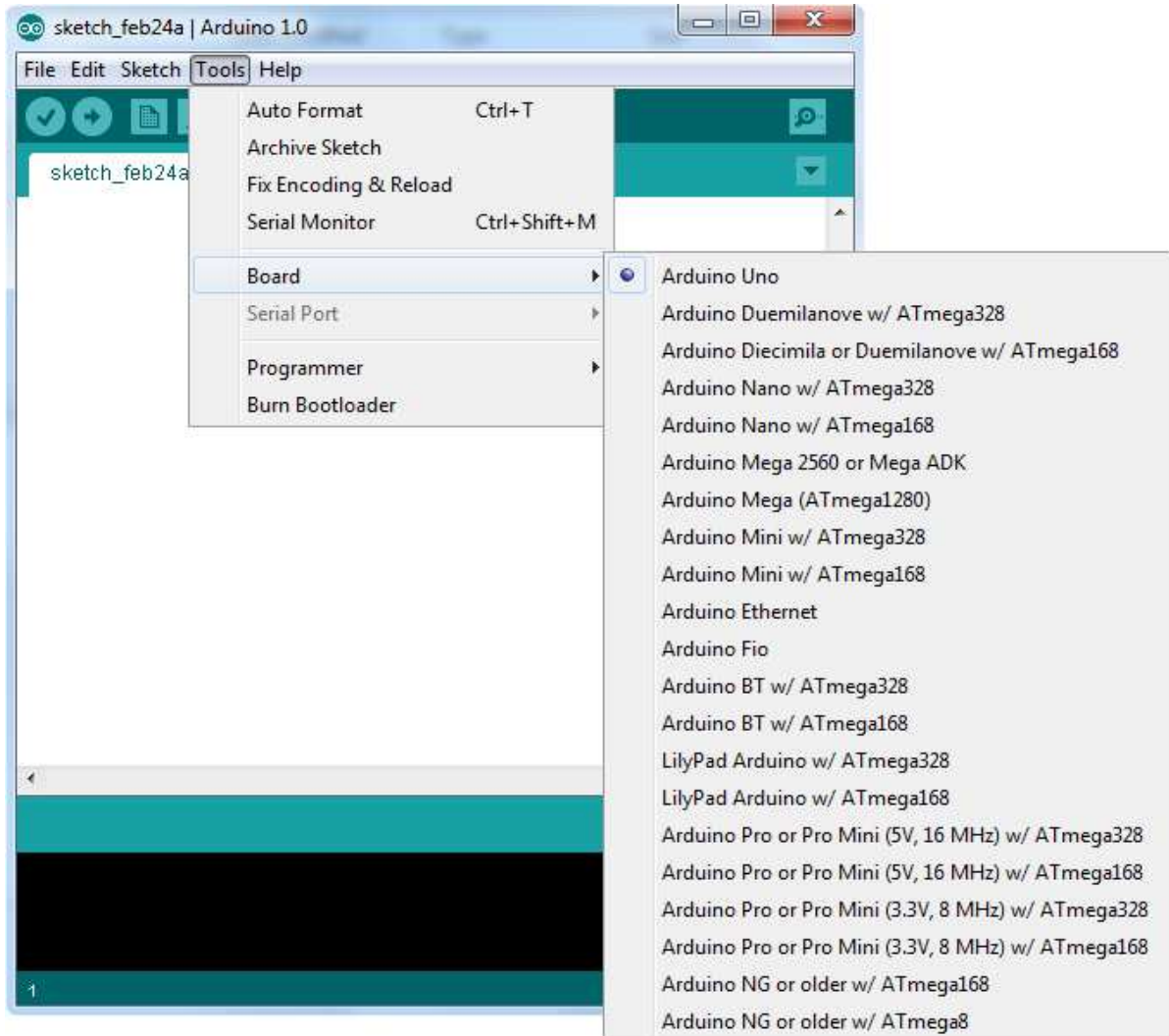
[\arduino-1.0.x\drivers\FTDI USB Drivers](#)

Como a BlackBoard possui um chip da FTDI para conversão de sinais USB para Serial (ou seja, conversão das informações que saem do computador para a informação que o microcontrolador propriamente dito entende), o driver a ser instalado é o da FTDI.

Na parte de seleção de placas na IDE, você deverá selecionar "Arduino UNO R3" quando for utilizar a BlackBoard, já que ela utiliza o mesmo bootloader do Arduino UNO R3.

SELEÇÃO DE PLACA ARDUINO

Para salvar códigos em sua placa Arduino, você precisa selecionar qual placa está usando no ambiente de desenvolvimento Arduino. Para isto, basta ir ao menu **TOOLS** e depois **BOARD**, conforme a figura abaixo:



Atenção: Caso você não esteja utilizando a placa Arduino UNO (ou BlackBoard), selecione a placa correta.

Atenção: Nos experimentos abaixo procure sempre utilizar o ambiente de desenvolvimento (IDE) Arduino MAXIMIZADO.

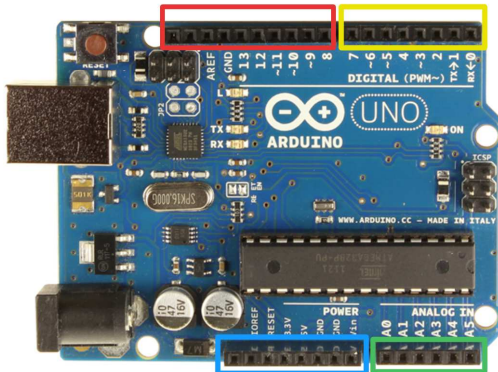
Dica: Para tornar o envio de códigos do computador para a placa Arduino mais rápido, desconecte os periféricos de seu computador, como dispositivos de Bluetooth, etc.

IMPORTANTE:

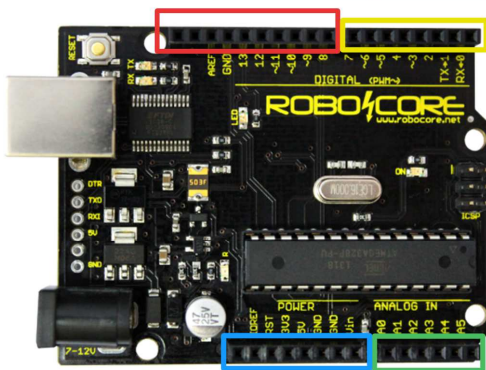
Qual placa você vai usar com o kit?

Arduino UNO, BlackBoard, Arduino Leonardo, Arduino Mega...tantas opções!
O melhor é saber que, independente da placa que você escolher, você conseguirá usar este material. Isto porque os pinos das placas usados nos experimentos deste material são comuns a todas as placas. Veja na próxima página os pinos comuns entre todas as placas.

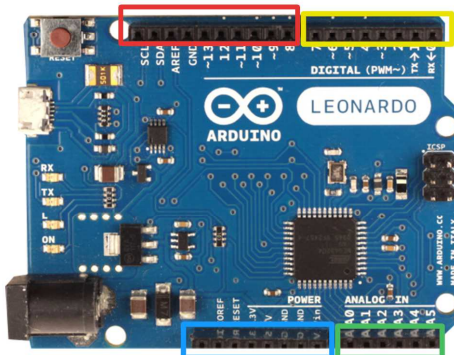
A compatibilidade entre as placas



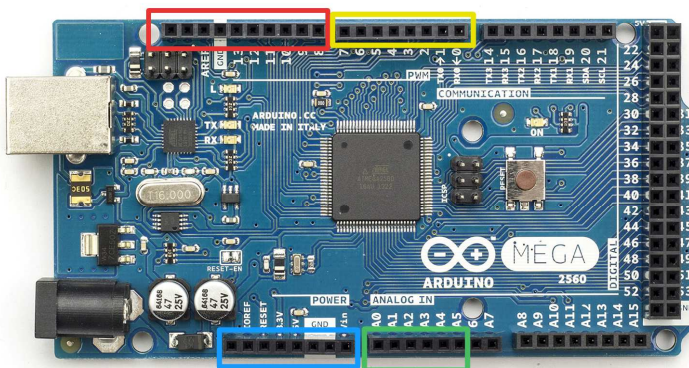
Arduino UNO



BlackBoard



Arduino Leonardo



Arduino Mega

o que é comum a todas as placas:

pinos digitais de 8 a 13

pinos digitais de 0 a 7

pinos analógicos

pinos de alimentação

Assim, mesmo os esquemas abaixo sendo ilustrados com a placa BlackBoard, nada impede que você use seu Arduino UNO, por exemplo, ou qualquer outra placa!

UMA BREVE DESCRIÇÃO DOS COMPONENTES

Para montar os experimentos deste kit não é necessário nenhum tipo de curso anterior de eletrônica. Mas, para você identificar cada um dos componentes e deixar a compreensão do circuito um pouco mais fácil, aqui detalharemos um pouco cada um deles.

- **RESISTOR**

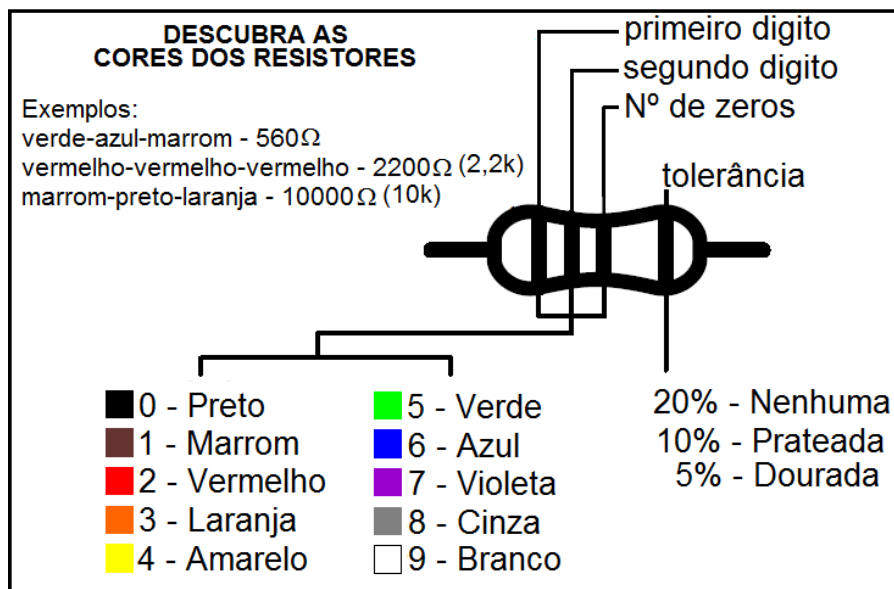


O que isto faz: Limita a corrente elétrica que passa pelo circuito. Para limitar mais ou menos corrente, o valor deste componente pode variar.

Número de pinos: 2 pinos de mesmo comprimento

+ **Detalhes:** <http://pt.wikipedia.org/wiki/Resistor>

Para saber o valor de cada resistor, basta seguir o esquema abaixo:



Em nossos experimentos neste kit, usaremos apenas três valores de resistores, o de 300Ω , o de $10k\Omega$ e o de $1M\Omega$. Você encontrará o símbolo deles nos esquemas elétricos. Para começar a se familiarizar, os símbolos são estes ao lado. Você consegue distinguir qual é o de 300Ω e qual é o de $10k\Omega$? Veja que eles são muito parecidos, porém você deve começar a leitura de cores pelo lado oposto ao dourado.



- **BUZZER**



O que isto faz: Quando uma corrente elétrica passa por ele, ele emite um som.

Número de pinos: 2 pinos (este componente tem polaridade, portanto fique atento na hora de ligá-lo)

+ **Detalhes:** http://pt.wikipedia.org/wiki/Sensor_piezoel%C3%A9trico

- **DISPLAY DE LCD**



O que isto faz: Mostra dados lidos pelo Arduino em letras e números, muito utilizado em diversos equipamentos eletrônicos. Este dispositivo mostra os dados que estão dentro do Arduino para os seres humanos de uma forma inteligível

Número de pinos: 16 pinos.

+ **Detalhes:** <http://pt.wikipedia.org/wiki/LCD>

- **LED RGB**

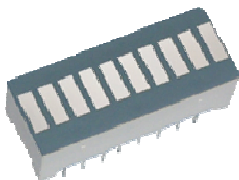


O que isto faz: Pense em três LEDs de alto brilho: um vermelho, um verde e um azul. Agora, junte todos eles em um só. Pronto, isso é um LED RGB.

Número de pinos: 4 pinos, onde o maior deles é comum a todas as cores.

+ **Detalhes:** <http://pt.wikipedia.org/wiki/RGB>

- **BARRA GRÁFICA DE LEDS**

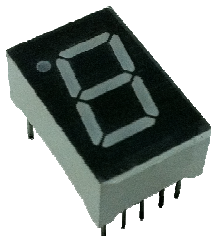


O que isto faz: Cada retângulo pequeno na barra gráfica é um LED. Serve para mostrar visualmente a intensidade de alguma grandeza

Número de pinos: 20 pinos, usaremos todos - a cada 2 acendemos um LED

+ **Detalhes:** <http://goo.gl/GJXqER>

- **DISPLAY DE 7 SEGMENTOS CATODO COMUM**



O que isto faz: Possui 7 LEDs em formato de número "8", com os quais é possível acender os números de 0 a 9. Ainda possui um LED indicador de ponto

Número de pinos: 10 pinos

+ **Detalhes:** http://pt.wikipedia.org/wiki/Display_de_sete_segmentos

- **CIRCUITO INTEGRADO 4511**

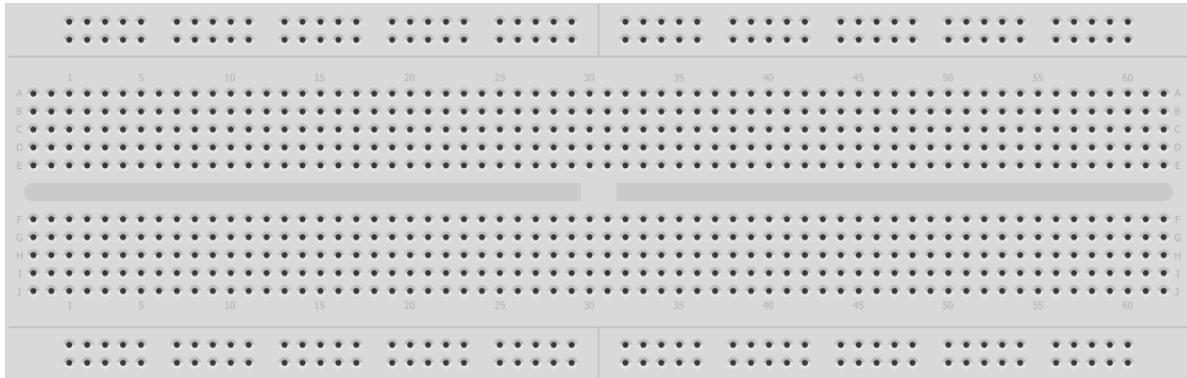


O que isto faz: Traduz um número em binário para o display de 7 segmentos, facilitando o uso do display e economizando portas do microcontrolador

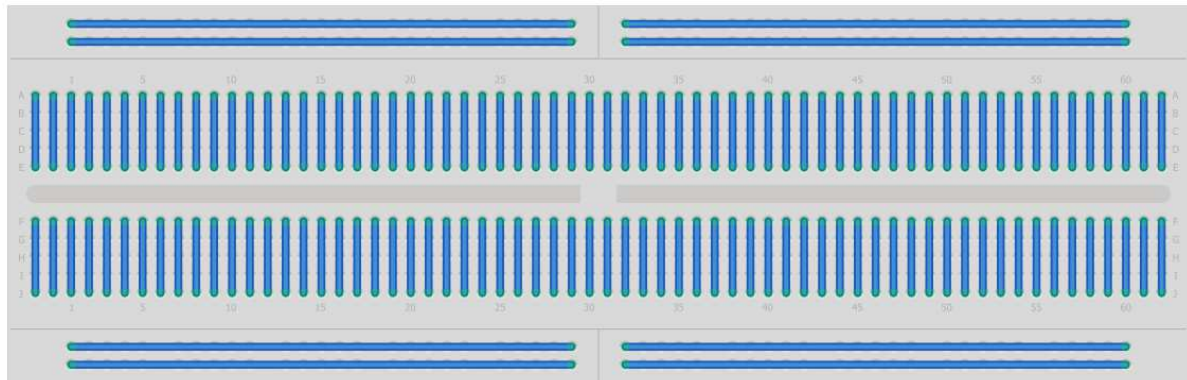
Número de pinos: 16 pinos

Detalhes: <http://goo.gl/h72GoC>

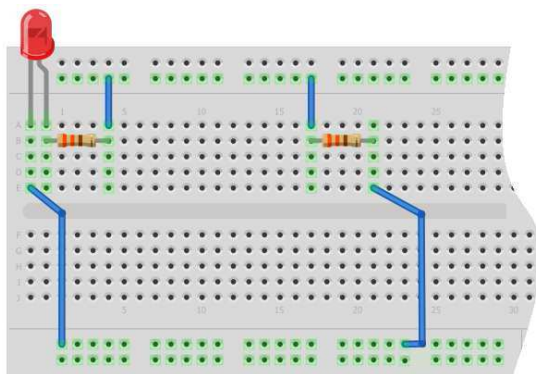
• **PROTOBOARD**



O que isto faz: trata-se de uma placa de plástico, cheia de pequenos furos com ligações internas, onde você irá fazer as ligações elétricas. Os furos nas extremidades superior e inferior são ligados entre si na horizontal, enquanto que as barras do meio são ligadas na vertical. Para ilustrar isto, veja abaixo como são as ligações internas da protoboard:



Cada fio azul acima representa uma ligação interna. Para deixar este componente totalmente entendido, veja o exemplo abaixo:



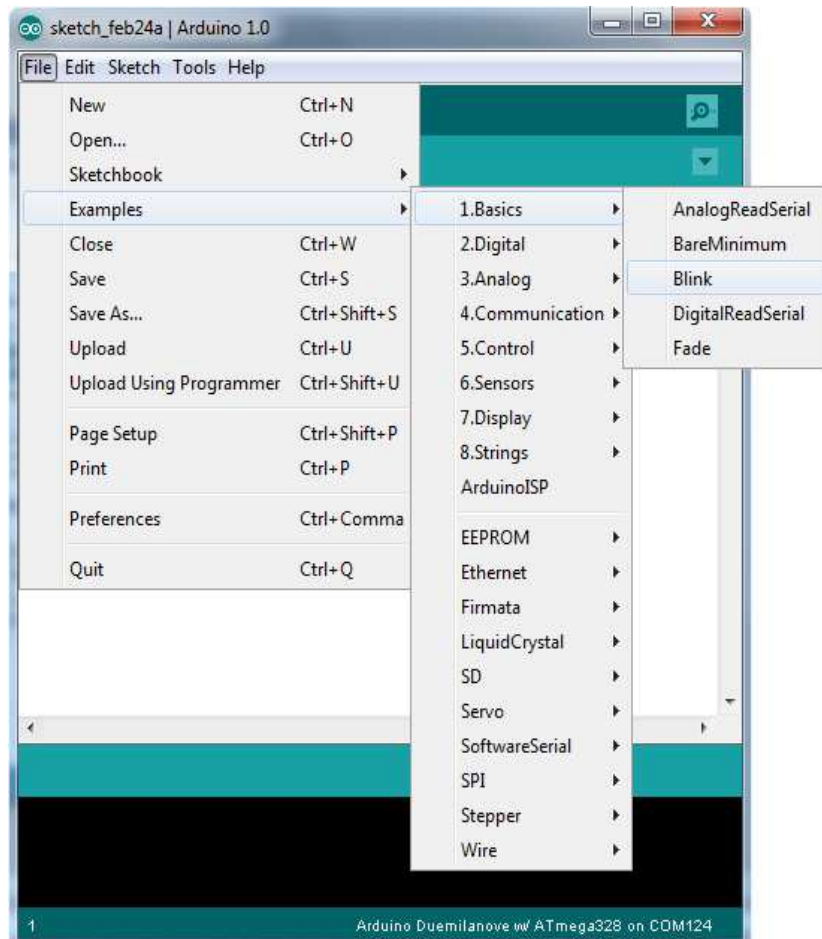
O led vermelho tem a extremidade direita ligada a um resistor. Este resistor está ligado a outro resistor por meio de uma das ligações internas superiores da protoboard. Este último resistor, por sua vez, está ligado à extremidade esquerda do led, utilizando uma das ligações internas inferiores da protoboard.

Número de pinos: na protoboard que acompanha o kit existem 840 furos, porém existem protoboards com menos e com mais furos.

+ **Detalhes:** <http://pt.wikipedia.org/wiki/Protoboard>

INTRODUÇÃO

Para entender como funciona o Arduino, vamos começar com o experimento mais simples e básico, o exemplo BLINK que está pronto no software de compilação do Arduino. Para acessá-lo clique em **FILE > EXAMPLES > 1.BASICS > BLINK** como mostrado na figura abaixo:



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

Código:

```
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
The circuit:
* LED connected from digital pin 13 to ground.
* Note: On most Arduino boards, there is already an LED on the board
connected to pin 13, so you don't need any extra components for this example.
Created 1 June 2005
By David Cuartielles
http://arduino.cc/en/Tutorial/Blink
based on an original by H. Barragan for the wiring i/o board
*/

int ledPin = 13; // LED connected to digital pin 13
// The setup() method runs once, when the sketch starts
void setup() {
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}
// the loop() method runs over and over again,
// as long as the Arduino has power
void loop()
{
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(1000); // wait for a second
  digitalWrite(ledPin, LOW); // set the LED off
  delay(1000); // wait for a second
}
```

Para iniciar o entendimento do código, devemos observar o que são e como são feitos os comentários em um código de linguagem C.

Para fazer um comentário quer irá se desenvolver por mais de 1 linha, devemos usar os caracteres:

`/*` para começar um comentário de mais de 1 linha

`*/` para finalizar os comentários que foram feitos anteriormente

Para fazer um comentário em 1 linha apenas, podemos utilizar:

`//` para fazer um comentário de apenas 1 linha

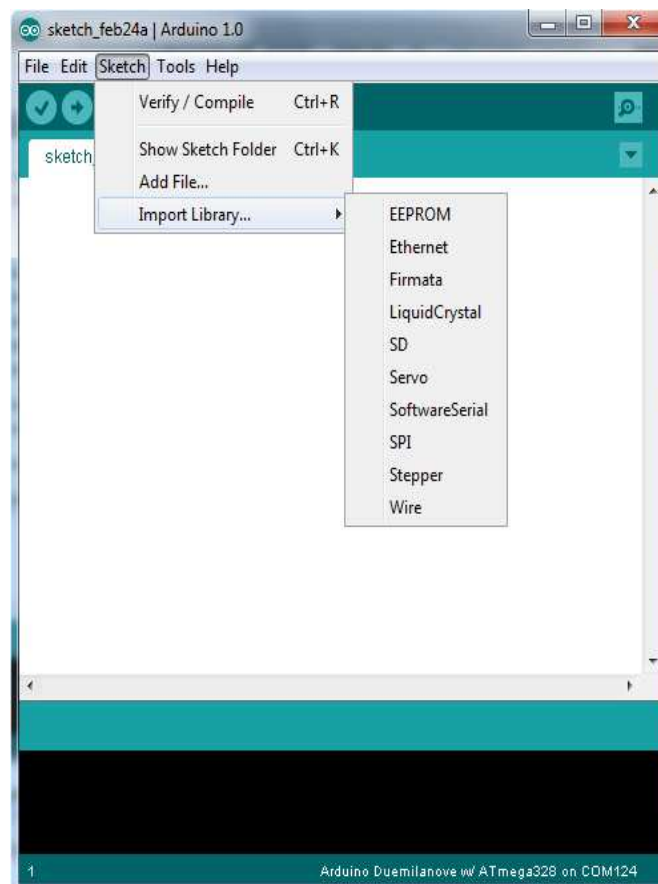
Entendido o que é um comentário, e se olharmos o código do BLINK mais a fundo, veremos que o código está escrito em apenas 11 linhas. Veja se você consegue identificar quais são estas 11 linhas.

Vamos agora entender a estrutura dos programas. No início de todos os programas, uma ordem deve ser respeitada:

1. Estrutura de Inclusão de Bibliotecas
2. Estrutura de Declaração de Variáveis
3. Estrutura Setup
4. Estrutura Loop
5. Demais estruturas de funções

O que são estas 5 estruturas citadas acima?

O diferencial de uma placa como o Arduino está profundamente ligada à estrutura de número 1 citada acima. Quando você estiver pensando em fazer algum projeto mirabolante, você pode ter certeza de que há 90% de chances de alguém já o ter feito. Desta forma, quando alguém já o fez, é bem provável que este alguém, em qualquer parte do mundo, já tenha escrito toda uma biblioteca para fazer o tal projeto. Por exemplo, digamos que em um sábado a noite dê uma vontade louca de fazer um carrinho de controle remoto controlado por um controle de PlayStation®. Você não faz a menor ideia de como começar a pensar em como programar este carrinho controlado por este controle tão comum no seu dia-a-dia. Então, a primeira coisa que você deve fazer é ir ao Google e perguntá-lo (pesquisar) se alguém já desenvolveu uma biblioteca para você utilizar um controle de PlayStation® com seu Arduino. Digite no Google: **ARDUINO PLAYSTATION CONTROLLER LIBRARY**. Na data que este documento está sendo redigido, você encontra "Aproximadamente 9.160 resultados". E sim, um deles, pelo menos, é a biblioteca que você está precisando para desenvolver seu carrinho de controle remoto com controle de PlayStation®. Portanto, o que são Bibliotecas? São conjuntos de funções desenvolvidas para uma aplicação particular. Seu ambiente de desenvolvimento Arduino já vem com algumas bibliotecas instaladas. Para vê-las, simule que você quer importar uma biblioteca (apenas simule, não precisa clicar em nenhuma por enquanto). Para tanto, clique em **SKETCH > IMPORT LIBRARY...** e veja quantas bibliotecas prontas para seu uso já existem:



Neste momento não iremos utilizar nenhuma das bibliotecas mostradas acima, mesmo porque nosso programa BLINK não necessita de uma biblioteca para funcionar, pois é um programa muito básico e utiliza apenas escritas digitais e delays, funções que já estão inserida em todos os programas feitos no ambiente de desenvolvimento Arduino. Por este motivo você pode notar que o programa BLINK, após os comentários iniciais, começa com a declaração de variáveis:

```
int ledPin = 13; // LED connected to digital pin 13
```

A linha anterior quer dizer o seguinte:

int : variável do tipo inteira

ledPin = 13; ; nome da variável. Neste caso, como o próprio nome diz, temos que a variável PINO DO LED vale 13.

// LED connected to digital pin 13 : comentário dizendo que existe um LED conectado ao pino digital de número 13.

Agora nós te convidamos a olhar seu Arduino mais de perto. Se você notar, verá que logo abaixo do pino 13 digital existe um LED SMD, ou seja, um microled, já colocado na placa, como mostra a figura abaixo:



Vamos agora olhar a estrutura de Setup do programa:

```
void setup() {
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}
```

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

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

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

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

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

```
void loop()
{
digitalWrite(ledPin, HIGH); // set the LED on
delay(1000);                // wait for a second
digitalWrite(ledPin, LOW);  // set the LED off
delay(1000);                // wait for a second
}
```

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

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


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

digitalWrite(ledPin, LOW); // set the LED off
delay(1000); // wait for a second : Estes dois comandos são análogos aos dois vistos anteriormente, com a única diferença que a escrita digital escreverá um 0 no pino do led, ou seja, um nível lógico baixo: o led apagará e o Arduino espera 1 segundo para fazer a próxima instrução que, no caso, volta a ser o **digitalWrite(ledPin, HIGH);** .

Se este programa está 100% entendido, já podemos compilar o mesmo e fazer o upload para nossa placa Arduino. Para compilar o programa devemos clicar no botão Verify do ambiente de

desenvolvimento, para ver se não existe nenhum erro de código. O botão é o seguinte: 

Se na barra inferior aparecer a mensagem: *Done Compiling*, o programa está pronto para ser

enviado ao Arduino. Para tanto, basta clicar no botão Upload que é o seguinte:  . Espere então o upload ser completado e pronto. Você deverá ver o led da placa piscando com intervalos de 1 segundo.

Vista toda esta explicação, agora sim podemos começar a estudar o primeiro módulo deste material.

▪ **Módulo 1**

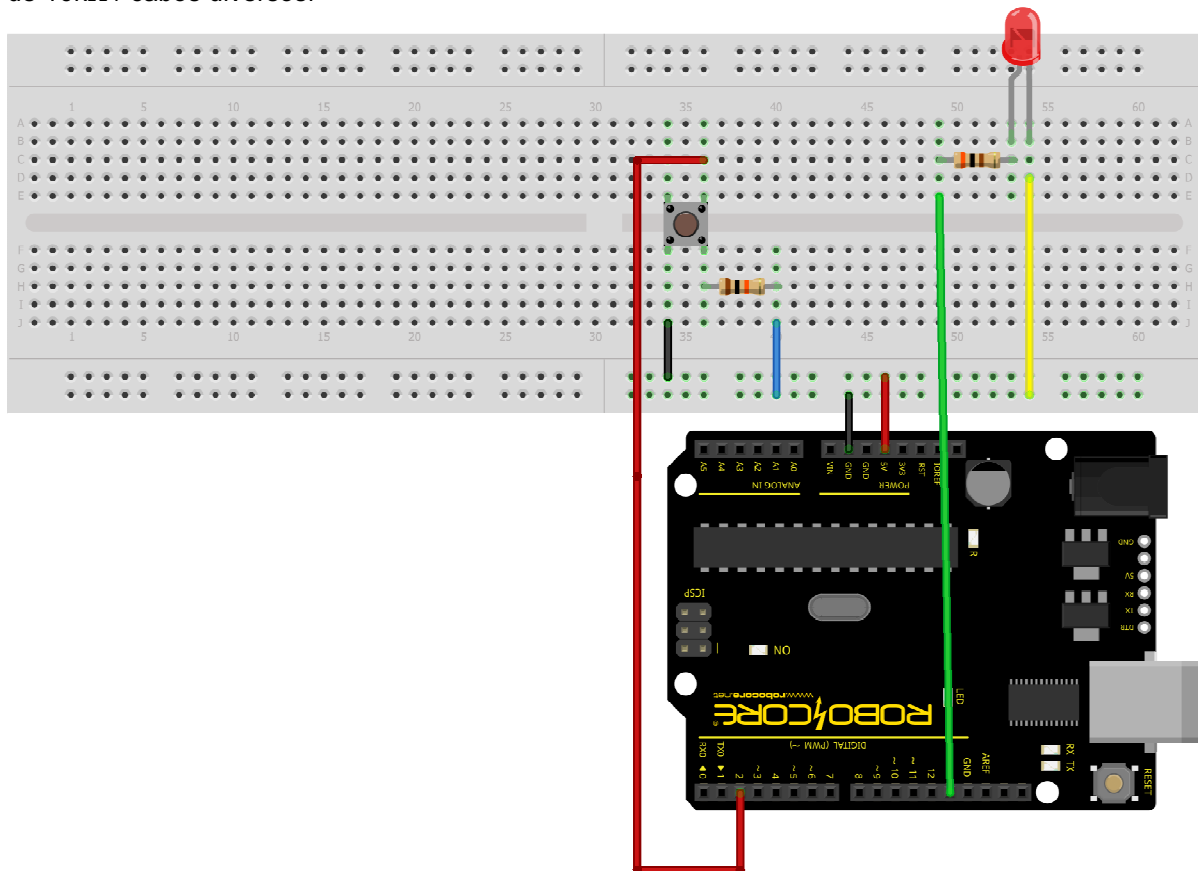
Componentes: 1 Botão + 1 Led

Descrição: Conforme você pressiona um pushbutton, um led é aceso

Dificuldade: 

Trata-se de fazer um botão acender um led quando pressionado e, quando solto, o led deverá apagar. Coloque os componentes como está sendo mostrado na imagem abaixo, bem como suas ligações:

Componentes utilizados: 01x Led Vermelho / 01x Resistor de 300Ω / 01x PushButton / 01x Resistor de 10kΩ / cabos diversos.



fritzing

Dica 1: Caso tenha dificuldades em montar o circuito, a imagem a cima (e todas as outras imagens de circuitos) estão disponíveis em alta definição na pasta "**Imagens dos Experimentos**" no CD que acompanha este kit.

Dica 2: Não importam as cores dos fios na hora de ligação, e sim fazer a ligação correta. Preste bastante atenção ao fazer as ligações pois apenas 1 fio no lugar errado pode comprometer todo o experimento.

Dica 3: Se divirta! Mais que estudar e aprender, você vai se divertir com esta placa :)

Código:

```
/******\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**              Módulo 1                  **
\*****/

const int ledPin = 13; //led no pino 13
const int Botao = 2; //botao no pino 2
int EstadoBotao = 0; //Variavel para ler o status do pushbutton
void setup(){
  pinMode(ledPin, OUTPUT); //Pino do led será saída
  pinMode(Botao, INPUT); //Pino com botão será entrada
}
void loop(){
  EstadoBotao = digitalRead(Botao); /*novo estado do botão vai ser igual ao que
                                     Arduino ler no pino onde está o botão.
                                     Poderá ser ALTO (HIGH)se o botão estiver
                                     Pressionado, ou BAIXO (LOW),se o botão
                                     estiver solto */

  if (EstadoBotao == HIGH){ //Se botão estiver pressionado (HIGH)
    digitalWrite(ledPin, HIGH); // acende o led do pino 13.
  }
  else{ //se não estiver pressionado
    digitalWrite(ledPin, LOW); //deixa o led do pino 13 apagado
  }
}
```

Recomendamos que você tente entender passo a passo o programa anterior, para não ter problemas quando os códigos começarem a ficar mais complexos.

Após compilar o código e fazer o upload na sua placa, você já deve poder apertar o botão e o led da protoboard acender e, quando soltar, o led deve apagar.

Se houve algum problema, procure seu erro e tente consertá-lo. Se não, parabéns! Você concluiu o primeiro módulo RoboCore Arduino Kit Iniciante. Agora você está pronto para começar o módulo de número 2.

▪ **Módulo 2**

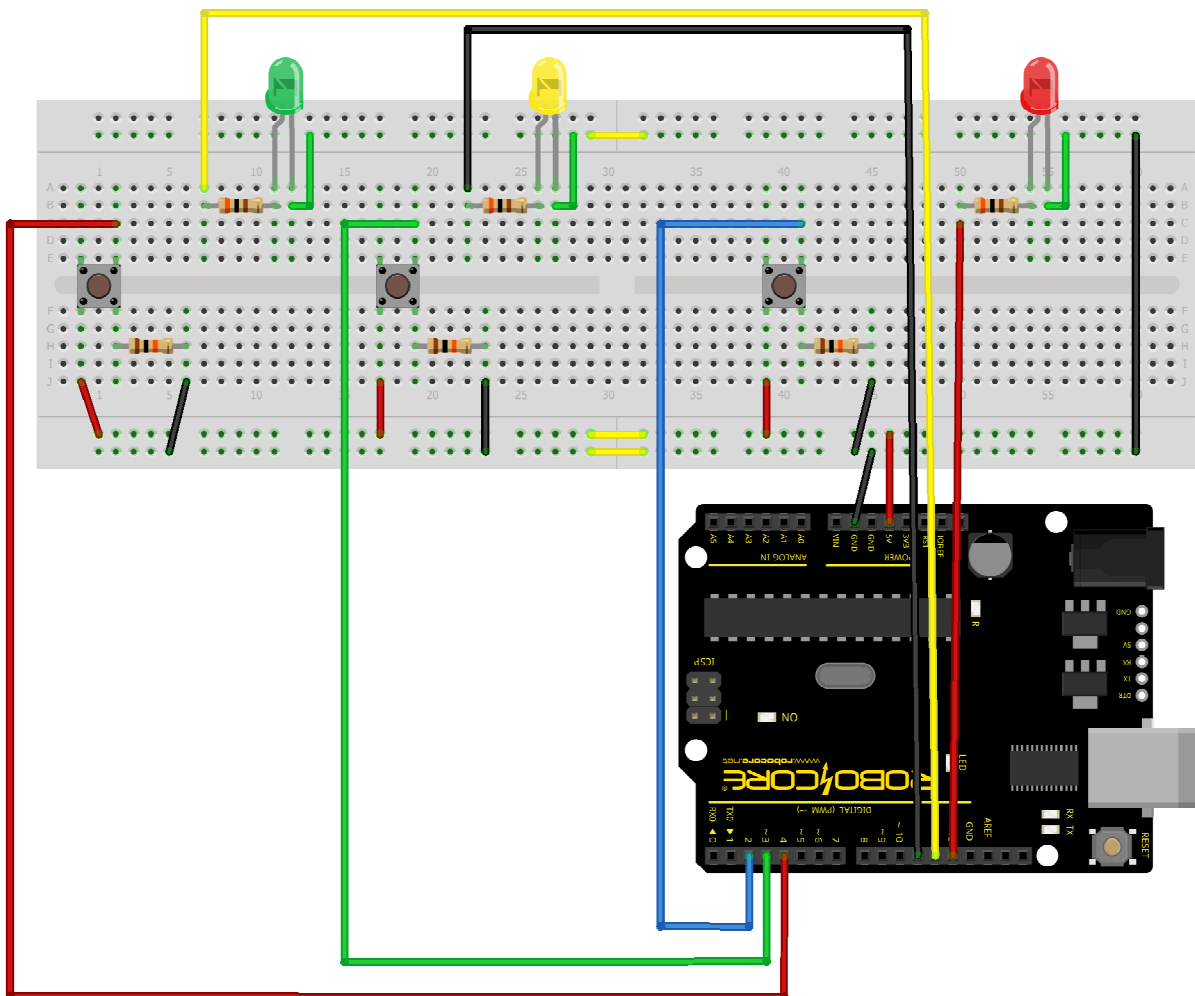
Componentes: 3 Botões + 3 Leds

Descrição: Conforme você pressiona qualquer um dos botões, leds de diferentes cores são acesos

Dificuldade: 

Este módulo é uma expansão do módulo anterior. A diferença deste com o módulo 1, é que neste teremos mais 2 botões e mais 2 leds de cores diferentes. Você pode tentar montar sozinho o novo circuito, ou utilizar o modelo de ligações abaixo:

Componentes utilizados: 01x Led Verde / 01x Led Amarelo / 01x Led Vermelho / 03x Resistores de 300 Ω / 03x Resistores de 10k Ω / 03x Pushbutton / cabos diversos.



fritzing

Neste ponto você já tem autonomia para desenvolver o resto do programa, mas se preferir, um código para utilizar os 3 botões pode ser o seguinte:

Código:

```
/*  
**      ROBOCORE ARDUINO KIT INICIANTE      **  
**  
**          Módulo 2          **  
**  
**      *****  
*/  
  
const int ledPin1 = 13;  
const int ledPin2 = 12;  
const int ledPin3 = 11;  
const int Botao1 = 2;  
const int Botao2 = 3;  
const int Botao3 = 4;  
int EstadoBotao1 = 0;  
int EstadoBotao2 = 0;  
int EstadoBotao3 = 0;  
void setup(){  
  pinMode(ledPin1, OUTPUT);  
  pinMode(Botao1, INPUT);  
  pinMode(ledPin2, OUTPUT);  
  pinMode(Botao2, INPUT);  
  pinMode(ledPin3, OUTPUT);  
  pinMode(Botao3, INPUT);  
}  
void loop(){  
  EstadoBotao1 = digitalRead(Botao1);  
  EstadoBotao2 = digitalRead(Botao2);  
  EstadoBotao3 = digitalRead(Botao3);  
  if (EstadoBotao1 == HIGH){  
    digitalWrite(ledPin1, HIGH);  
  }  
  else{  
    digitalWrite(ledPin1, LOW);  
  }  
  if (EstadoBotao2 == HIGH){  
    digitalWrite(ledPin2, HIGH);  
  }  
  else{  
    digitalWrite(ledPin2, LOW);  
  }  
  if (EstadoBotao3 == HIGH){  
    digitalWrite(ledPin3, HIGH);  
  }  
  else{  
    digitalWrite(ledPin3, LOW);  
  }  
}
```

Lembrete: Nunca se esqueça dos ponto e vírgula (;) no final dos comandos em seu programa em C.

▪ **Projeto Piano**

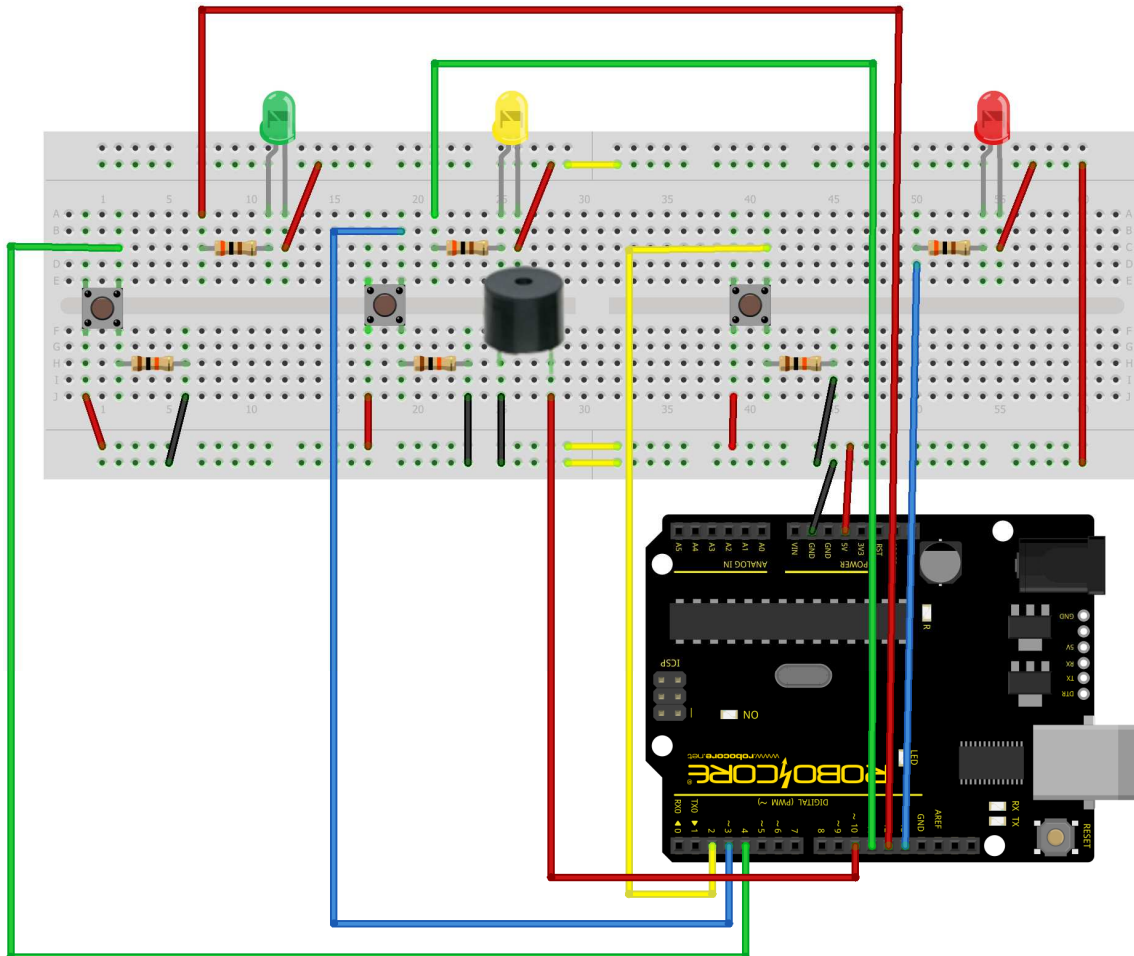
Componentes: 3 Botões + 3 Leds + Buzzer

Descrição: Cada botão toca uma nota musical diferente e acende um led. É expansível – por conta do usuário – para mais uma nota musical com o botão (e o led) reserva.

Dificuldade: 

Utilizando os conceitos aprendidos nos módulos 1 e 2, podemos agora montar o primeiro projeto: o Projeto Piano. Neste projeto cada um dos 3 botões tocará uma nota musical diferente. Para montar o projeto usaremos um novo componente: o Buzzer. Um Buzzer nada mais é do que um pequeno alto-falante. Obviamente que ele não consegue tocar músicas, mas consegue fazer apitos soarem, como sirenes ou alarmes. A maioria dos alarmes de pequenos equipamentos eletrônicos é feito através de um buzzer. Ele funciona da seguinte maneira: quando alimentado por uma fonte, componentes metálicos internos vibram da frequência da fonte, produzindo assim um som. Para este experimento, você também pode utilizar um pequeno alto-falante (o som sai mais puro e a diferença entre as notas musicais é mais nítida). Último detalhe sobre o Buzzer: ele tem polaridade. Se você retirar o adesivo superior do buzzer poderá ver um sinal de positivo (+). Este sinal mostra onde está o pino positivo do componente. Sempre ligue este a uma saída digital do Arduino e o outro em GND. Para fazer a montagem, o modelo a seguir pode ser seguido:

Componentes utilizados: 01x Led Verde / 01x Led Amarelo / 01x Led Vermelho / 03x Resistores de 300 Ω / 03x Resistores de 10k Ω / 03x Pushbutton / cabos diversos / 01x Buzzer 5V.



fritzing

Note que são usados dois tipos de resistores, por mais que pareçam ter o mesmo valor eles são diferentes!

Veja que a única diferença entre este projeto e o módulo 2 é a inserção de um Buzzer.

Código:

```

/*****\
**   ROBOCORE ARDUINO KIT INICIANTE   **
*                                     *
**           Projeto Piano           **
\*****/

const int ledPin1 = 13;
const int ledPin2 = 12;
const int ledPin3 = 11;
const int Botao1 = 2;
const int Botao2 = 3;
const int Botao3 = 4;
const int Buzzer = 10; //o buzzer está colocado no pino 10
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int EstadoBotao3 = 0;
int Tom = 0; //Variavel para armazenar a nota musical

void setup() {
  pinMode(Buzzer, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(Botao1, INPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(Botao2, INPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(Botao3, INPUT);
}

void loop(){
  EstadoBotao1 = digitalRead(Botao1);
  EstadoBotao2 = digitalRead(Botao2);
  EstadoBotao3 = digitalRead(Botao3);
  if(EstadoBotao1 && !EstadoBotao2 && !EstadoBotao3) {
    Tom = 100;
    digitalWrite(ledPin1, HIGH);
  }
  if(EstadoBotao2 && !EstadoBotao1 && !EstadoBotao3) {
    Tom = 200;
    digitalWrite(ledPin2, HIGH);
  }
  if(EstadoBotao3 && !EstadoBotao2 && !EstadoBotao1) {
    Tom = 500;
    digitalWrite(ledPin3, HIGH);
  }
  if(Tom > 0) { //enquanto Tom for maior que zero faça o que esta descrit o baixo:
    digitalWrite(Buzzer, HIGH); // Liga buzzer
    delayMicroseconds(Tom); // Espera o tempo proporcional ao comprimento de onda da nota musical em milisegundos
    digitalWrite(Buzzer, LOW); // Desliga buzzer
    delayMicroseconds(Tom); // Espera o tempo proporcional ao comprimento de onda da nota musical em milisegundos
    Tom = 0; // Reseta o Tom para zero, para sair do loop while e nao tocar o som constantemente
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, LOW);
    digitalWrite(ledPin3, LOW);
  }
}

```

▪ **Módulo 3**

Componentes: 1 Sensor de Temperatura LM35

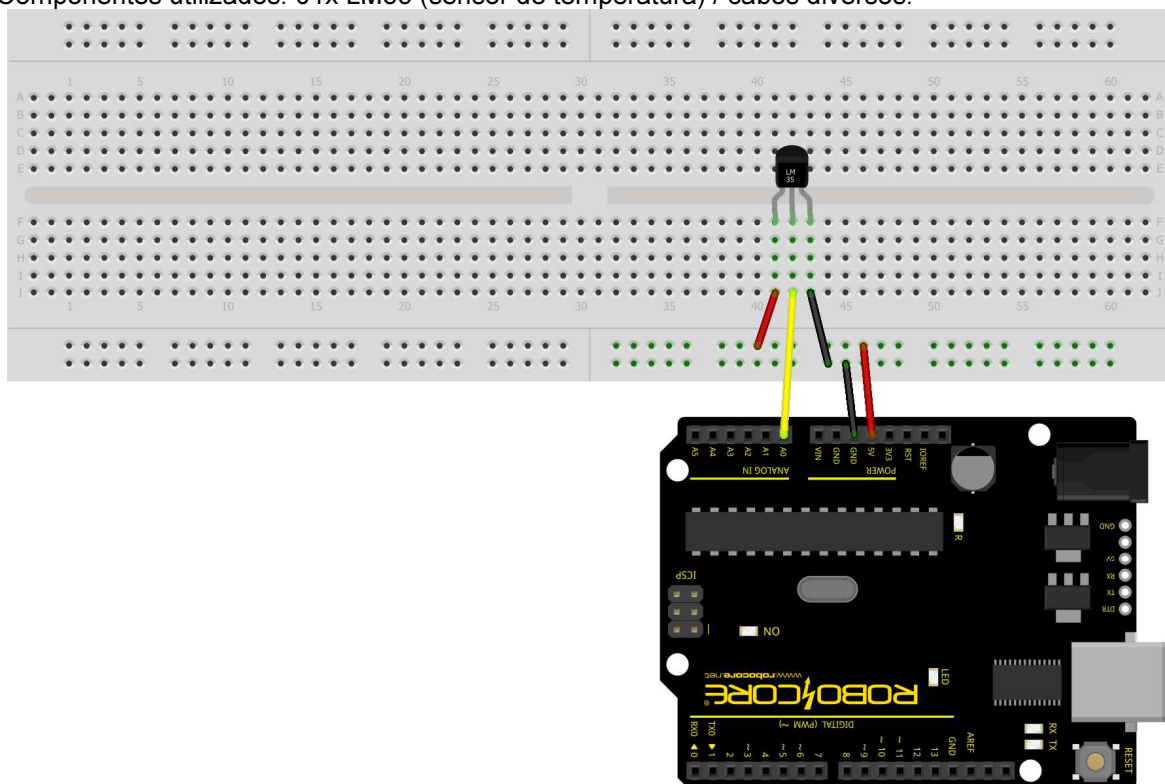
Descrição: Com o auxílio da porta serial e do monitor serial, o usuário irá fazer a leitura do sensor de temperatura em °C para fazer o projeto seguinte.

Dificuldade: 

Este experimento é muito simples, mas tem um valor agregado muito grande. Iremos aqui fazer a primeira aquisição de dados do mundo externo pra dentro do Arduino. Usaremos para tanto um sensor de temperatura LM35 ligado a uma das entradas analógicas da placa. Iremos utilizar este sensor pois ele tem a capacidade de fornecer temperaturas na escala de °C para o Arduino, ou seja, o valor que lermos será a temperatura ambiente!

O circuito a ser montado é o seguinte:

Componentes utilizados: 01x LM35 (sensor de temperatura) / cabos diversos.



fritzing

Atenção: cuidado para não ligar o sensor invertido! Ele deve ficar com a face reta virada para frente, conforme a imagem acima, e a face arredondada virada para trás.

Como já sabemos, o LM35 lê a temperatura do ambiente em °C. A informação que ele nos dá na saída (pino do meio do sensor, o qual está ligado na porta analógica 0 - cabo amarelo) é uma tensão que varia linearmente com a temperatura. Sua faixa de medição, segundo o datasheet (folha de dados com todas as informações do componente), vai até +150°C e ele possui um incremento de 10mV/°C. Isto quer dizer que, cada vez que a temperatura ambiente aumenta 1°C, o sensor aumenta em 10mV a tensão no pino de saída. Por isso diz-se que a tensão de saída varia linearmente com a temperatura medida. Supondo, por exemplo, uma temperatura de 25°C, teremos na saída uma tensão de 250mV.

Nossa placa Arduino opera sua lógica em uma tensão de 5V (5000mV), isso quer dizer que nas portas analógicas podemos inserir uma tensão variável de 0V a 5V para fazer leituras. Para transformar estas tensões variáveis em dados que o Arduino entende, ele utiliza internamente um

conversor analógico-digital, ou simplesmente, **ADC**. Estes conversores são muito utilizados na eletrônica, desde os primórdios da invenção dos circuitos integrados para transformar um sinal analógico em digital. Caso você queira conhecer mais sobre estes conversores, sugerimos fazer uma boa pesquisa no fórum da RoboCore ou mesmo no Google, mas o que precisamos saber neste momento é que o conversor ADC do Arduino UNO, Mega, Leonardo e alguns outros modelos, tem uma resolução de **10 bits**. Este valor informa a qualidade da tradução do sinal que era analógica e vai se tornar digital. Estes 10 bits de resolução nos diz que o valor máximo que teremos na leitura de uma porta analógica é de $2^{10} = 1024$ valores. Isso quer dizer que, se colocarmos 0V na entrada analógica da placa, leremos no monitor serial (você já vai saber o que é isso) o valor "0". Se colocarmos 5V na entrada analógica, leremos o valor "1023" no monitor serial, pois o "0" também conta. Se colocarmos 2,5V na entrada analógica, leremos o valor 512. E assim por diante.

Agora, se pensarmos que a temperatura máxima que o sensor lê é 150°C, e sabendo que a cada 1°C a tensão de saída aumenta 10mV, quer dizer que a 150°C teremos 1500mV, ou seja, uma tensão máxima entrando na porta analógica de 1,5V. Isto é muito pouco comparado com o máximo de 5V que a entrada analógica do Arduino suporta, e vamos combinar que vai ser um pouco difícil testar o sensor a 150°C, certo? Não, não pense em colocar o sensor dentro do forno!

Se usarmos o sensor com a referência padrão da porta analógica de 5V, não teremos uma boa precisão do mesmo pois não estaremos usando o conversor ADC do Arduino da melhor forma possível. Para contornar isto, podemos usar uma artimanha para diminuir a tensão de referência das portas analógicas. Utilizando no setup do programa no Arduino um comando chamado **analogReference(INTERNAL) - no Arduino Mega utilize analogReference(INTERNAL1V1)**, alteramos a tensão máxima que pode entrar nas portas analógicas do Arduino de 5V para 1,1V. Isso é realmente muito útil quando estamos usando um sensor como o LM35. Se nossa tensão máxima de leitura é de 1,1V, podemos calibrar nosso sensor para ler no máximo 110°C (que daria 1100mV na leitura). Portanto saberemos que, quando lermos 1023 no monitor serial do Arduino, teremos a temperatura de 110°C. Esta teoria toda parece ser muito difícil, mas na prática você verá que é bastante simples.

Código:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
**                                          **
**              Módulo 3                  **
**                                          **
\*****/

const int LM35 = 0;
float temperatura = 0;
int ADClido = 0;

void setup(){
  Serial.begin(9600);
  analogReference(INTERNAL); //Se estiver usando Arduino Mega, use INTERNAL1V1
}

void loop(){
  ADClido = analogRead(LM35);
  temperatura = ADClido * 0.1075268817204301;
  Serial.print("Temperatura = ");
  Serial.print(temperatura);
  Serial.println(" *C");
  delay(1000);
}

```

Vamos entender este código. Como já foi dito, no começo do programa colocamos as bibliotecas usadas para fazer o projeto. Novamente, não temos nenhuma biblioteca por enquanto. O próximo conjunto de instruções são as variáveis e a declaração das mesmas:

```
const int LM35 = 0;
float temperatura = 0;
int ADClido = 0;
```

Significa que LM35 é uma **CONSTANTE INTEIRA** – por isso o “CONST INT”. É uma constante porque, a posição do sensor ligado no Arduino não mudará: ficará sempre na entrada analógica 0.
- **float temperatura = 0;** // temperatura é uma variável do tipo **FLOAT** e começa valendo zero. Este tipo de variável aceita casas decimais, e nós teremos uma precisão de temperatura de duas casas depois da vírgula (acredite, será uma ótima precisão).
- **int ADClido = 0;** // ADClido é a variável que vai armazenar o valor lido diretamente pelo pino analógico depois da conversão feita pelo ADC, sem tratamento nenhum. É do tipo inteiro pois este valor vai de 0 a 1023.

Seguindo com o programa vamos à parte do **setup**:

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

O comando **Serial.begin** serve para dizer ao Arduino que você irá coletar ou escrever dados no Arduino utilizando a porta serial, ou seja, através do cabo USB AB você vai ler ou escrever valores no mundo externo. O número entre os parênteses trata-se da taxa de dados que a placa vai se comunicar com o computador, neste caso usaremos 9600kbps. Logo abaixo, damos o comando da referência das portas analógicas que vimos anteriormente para passar a referência de 5V para 1,1V.

Quanto ao loop principal:

```
void loop(){
  ADClido = analogRead(LM35);
  temperatura = ADClido * 0.1075268817204301;
  Serial.print("Temperatura = ");
  Serial.print(temperatura);
  Serial.println(" *C");
  delay(1000);
}
```

O loop é muito simples. Na primeira linha, o Arduino irá assimilar o valor lido na entrada analógica 0 (que é nossa constante LM35) à variável ADClido. Após isto, multiplicamos o valor por 0,107421875 e salvamos na variável temperatura. Mas por que fazemos esta multiplicação?

Quando lemos o que existe no pino que está o LM35, estamos lendo a conversão do sinal analógico feito pelo ADC do Arduino. Ou seja, estamos lendo um valor que vai de 0 a 1023. Como mudamos a referência da porta analógica, se tivermos 1023 na leitura, teremos 1,1V entrando e teremos 110°C na temperatura (que calor!). Então, temos que fazer uma conta bastante simples para transformar o valor lido para graus Celsius. Vamos pensar: se 110°C representa um valor de leitura de 1023, logo o valor da temperatura que o sensor estiver lendo vai ser o valor traduzido pelo ADC. Colocando nos padrões de uma regra de três:

$$\begin{array}{ccc}
 110^{\circ}\text{C} & \times & 1023 \\
 \text{temperatura} & & \text{ADCLIDO} \\
 \downarrow & & \\
 \text{temperatura} = \frac{110}{1023} \times \text{ADCLIDO} & & \\
 \downarrow & & \\
 \text{temperatura} = 0,1075268817204301 \times \text{ADCLIDO} & &
 \end{array}$$

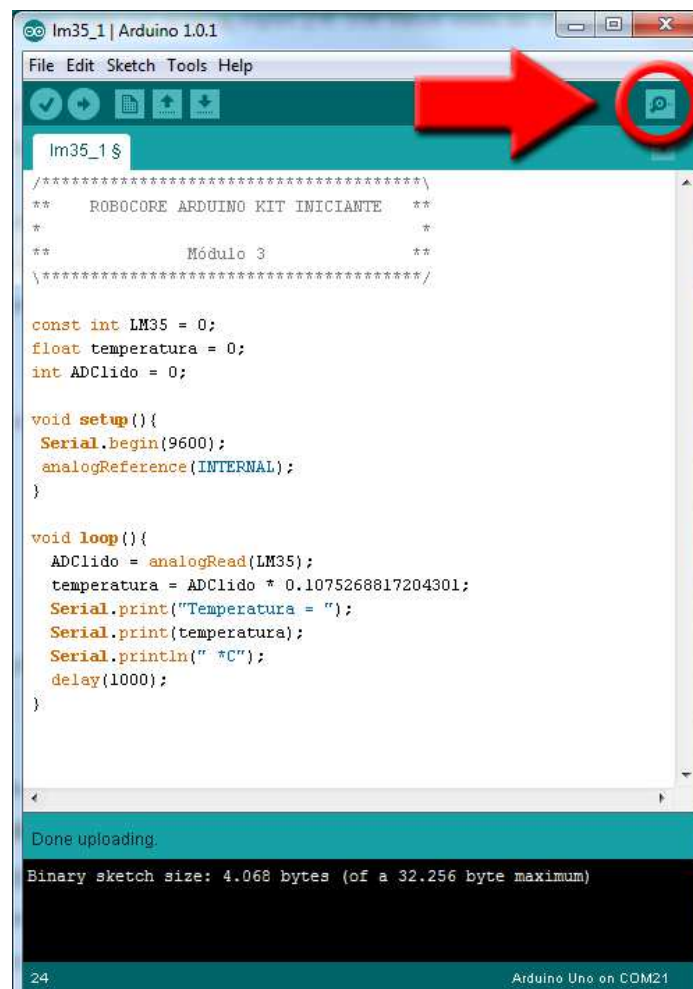
Dessa forma, para sabermos a temperatura em graus Celsius, basta multiplicar o valor lido pelo conversor analógico digital por aquele número encontrado!

Depois da multiplicação, colocamos os seguintes comandos:

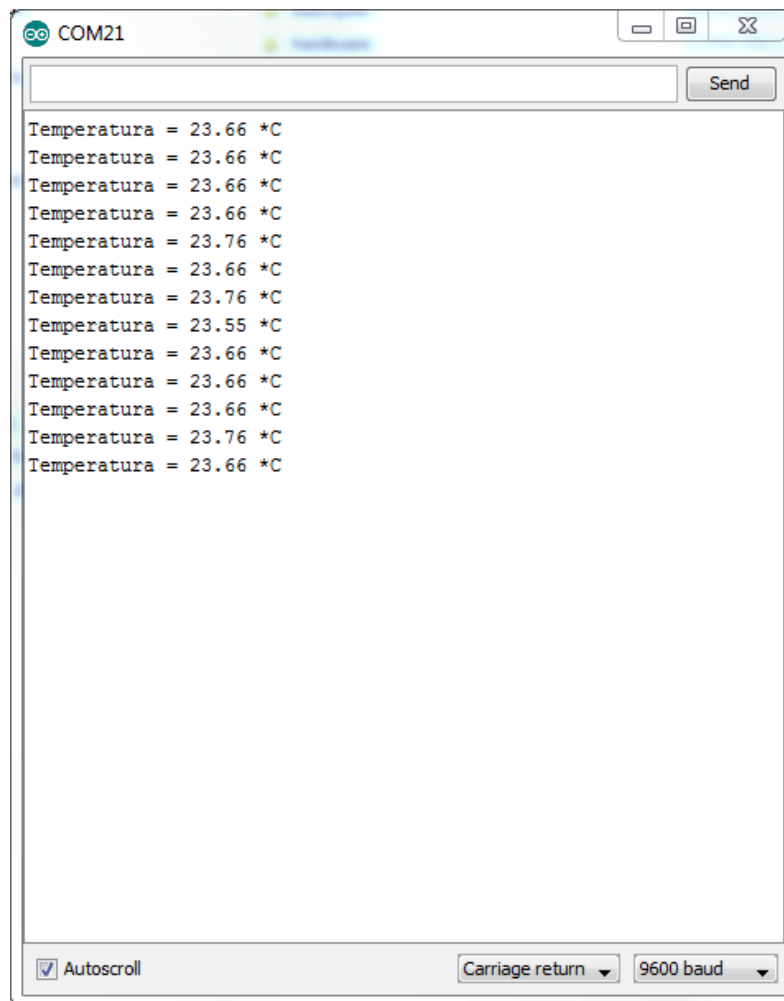
```
Serial.print("Temperatura = ");
Serial.print(temperatura);
Serial.println(" *C");
delay(1000);
```

O primeiro comando significa que iremos escrever na porta serial a frase "Temperatura = ". Logo depois, mostramos na tela o valor da variável temperatura que já está em graus Celsius, e para finalizar, colocamos um " *C" para indicar a unidade (não usamos o símbolo tradicional de graus ° pois o caracter fica um pouco estranho no monitor serial - você pode testar depois). Veja que nos dois primeiros comandos seriais, escrevemos `Serial.print` e no terceiro escrevemos `Serial.println`. A diferença entre ".print" e ".println" é que este último pula a linha depois do que escrever. Então depois de colocar o " *C", pulamos uma linha pra ficar mais fácil de ler. Depois damos um delay, ou seja, uma espera de 1 segundo para o leitor fazer uma nova temperatura.

Bastante teoria, não? Mas agora vamos ver tudo funcionando. Faça o upload do código para sua placa e abra o bendito **MONITOR SERIAL**, que falamos tanto mas mal sabemos o que ele é e o que ele faz. O monitor serial é a tela que o Arduino envia os comandos para o computador e também a tela que podemos enviar comandos em tempo real para o Arduino. Para ver os dados no monitor serial, basta clicar no botão indicado com a seta vermelha na figura a seguir, no ambiente de desenvolvimento do Arduino:



Depois de feito o compilamento e o upload do programa para sua placa Arduino, e após abrir o monitor serial pelo botão indicado anteriormente, você deverá ver algo parecido com:



(Sim, aqui está uma temperatura bem agradável!! O que você está lendo? Compartilhe conosco pelo fórum da RoboCore :)

Vale lembrar que a porta COM não é necessariamente 21, como está no topo da imagem anterior. Cada computador tem sua numeração de portas.

Veja que no canto inferior direito temos selecionado 9600 baud. Isto tem de ser selecionado conforme a configuração do parâmetro `Serial.begin` do **setup** de seu programa. Também é bom ressaltar que, como os componentes eletrônicos não são totalmente iguais e que a temperatura ambiente em cada ponto do mundo é diferente, você não necessariamente vai ler valores como os acima. Esta é a temperatura ambiente lida pelo sensor no local onde este material foi desenvolvido. Para fazer um teste com o sensor de temperatura, podemos utilizar um ferro de solda, ou um ferro de passar roupas, ou um secador de cabelo (qualquer coisa que esquente rapidamente) bem como seus dedos, visto que a temperatura deles é diferente do que a ambiente, mas não coloque a fonte de calor muito perto do sensor pois isto pode eventualmente danificá-lo.

ATENÇÃO: Somente utilize o comando `analogReference(INTERNAL)` se for utilizar sensores que enviem no máximo 1,1V para a porta analógica. Se você utilizar este comando e introduzir uma tensão maior de 1,1V, poderá causar danos permanentes a sua placa Arduino.

▪ **Projeto Alarme**

Componentes: 1 Sensor de Temperatura LM35 + 1 buzzer

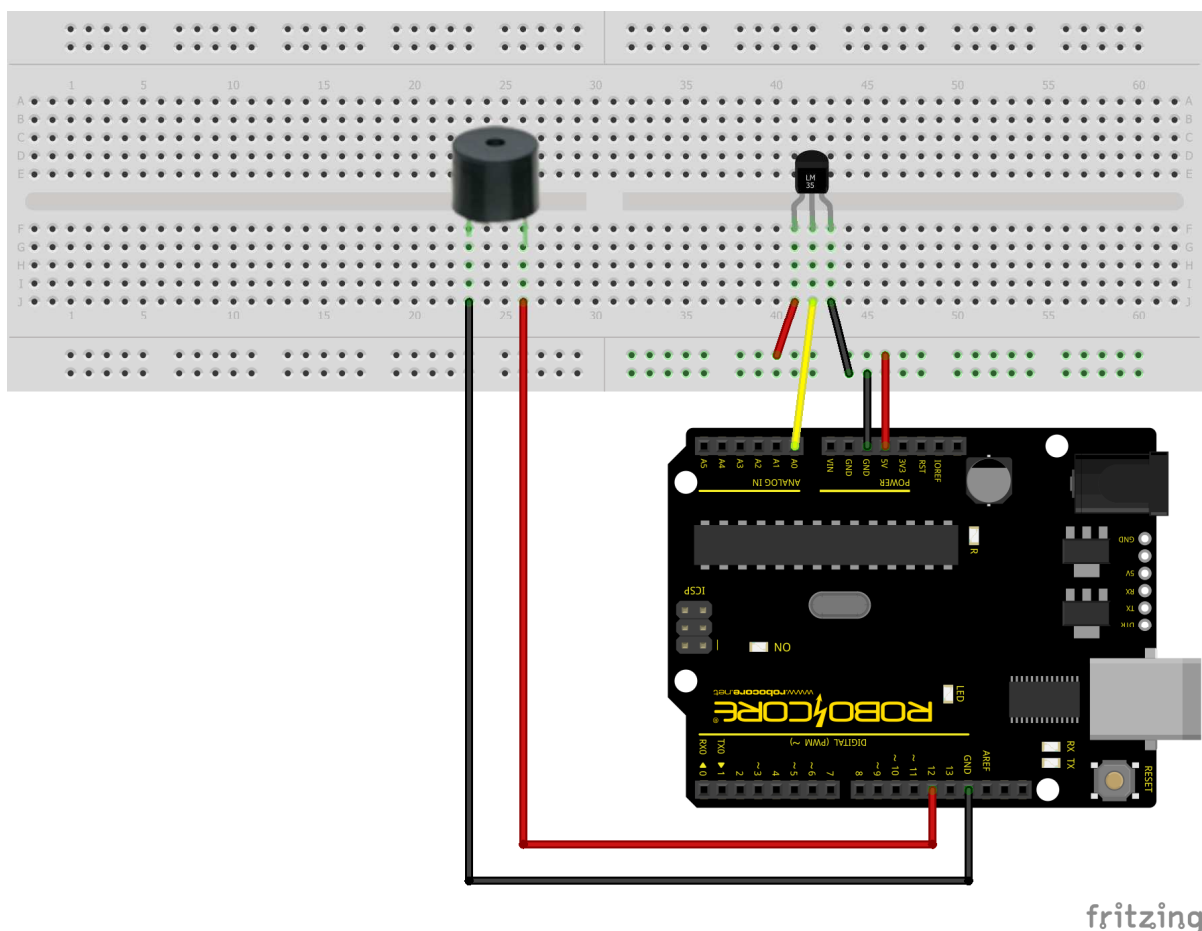
Descrição: A partir dos valores lidos no módulo 3, o usuário poderá montar um alarme que, se a temperatura de onde o sensor estiver localizado for maior, ou menor, ele soará.

Dificuldade: 

O intuito é muito simples: quando a temperatura for maior que um valor, escolhido por você, o buzzer começará a soar até que a temperatura volte ao estado perfeito. Este procedimento é muito usado em indústrias com processos e também em sensores de calor.

O circuito é o seguinte:

Componentes utilizados: 01x LM35 (sensor de temperatura) / cabos diversos / 01x Buzzer 5V.



Veja que a única diferença desta montagem para a anterior é a inserção de um buzzer, com seu pino positivo (com um sinal de + no topo do componente) ligado no pino digital 12 e o outro pino ligado em GND.

Código:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**              Projeto Alarme              **
\*****/

const int LM35 = 0;
float temperatura = 0;
int ADClido = 0;
const int Buzzer = 12;

void setup(){
  analogReference(INTERNAL);
  pinMode(Buzzer, OUTPUT);
}

void loop(){
  ADClido = analogRead(LM35);
  temperatura = ADClido * 0.1075268817204301;
  if(temperatura > 25){ // setei como 25°C
    digitalWrite(Buzzer, HIGH);
  }
  else{
    digitalWrite(Buzzer, LOW);
  }
}

```

Deste código para o do módulo anterior tivemos poucas mudanças. A começar pela inserção de uma nova variável chamada **Buzzer**, que de fato não é uma variável por isso escrevemos o **const**. Esta constante irá informar ao Arduino onde nosso pino positivo do buzzer está ligado, neste caso no pino digital 12. A próxima mudança no código foi dizer, no bloco **setup** do código, que o pino onde está o buzzer é uma saída, ou seja, iremos ligar o pino quando a temperatura for maior que alguma escolhida. Você conseguiu identificar no código onde estão estas duas mudanças?

A mudança no **loop** é um pouco mais significativa. Vamos dar uma olhada:

```

void loop(){
  ADClido = analogRead(LM35);
  temperatura = ADClido * 0.1075268817204301;
  if(temperatura > 25){ // setei como 25°C
    digitalWrite(Buzzer, HIGH);
  }
  else{
    digitalWrite(Buzzer, LOW);
  }
}

```

A começar que tiramos a escrita no monitor serial e também o delay de 1 segundo, isto porque queremos esta que esta automação seja o mais rápido possível, ou seja, assim que atingirmos a temperatura desejada queremos que nossa buzina ligue para nos informar. Se deixássemos o delay de 1 segundo, o buzzer não iria ligar na hora exata que o sensor leu a temperatura desejada e isto poderia nos causar problemas em projetos maiores.

Depois inserimos uma rotina condicional **IF**, a qual diz o seguinte:


SE a temperatura for **MAIOR** que **25** faça: `digitalWrite(Buzzer, HIGH);`
SE NÃO faça: `digitalWrite(Buzzer, LOW);`

Enviar um comando `digitalWrite(Buzzer, HIGH)`, quer dizer que estamos fazendo uma escrita digital no pino do Buzzer, e colocando ele em nível lógico ALTO (HIGH), assim o Arduino entende que é para ligar o pino do Buzzer, ou seja, colocar 5V nele. Paralelamente, se a temperatura for menor que 25°, desligamos o Buzzer enviando o comando `digitalWrite(Buzzer, LOW)`. Veja que só coloquei 25°C no teste pois onde o manual foi escrito a temperatura ambiente é de 23°C. Colocando o dedo no sensor, a temperatura sobe acima dos 25°C e assim consigo testar. Se você está em um lugar mais quente, altere este valor até encontrar um que dê resultados.

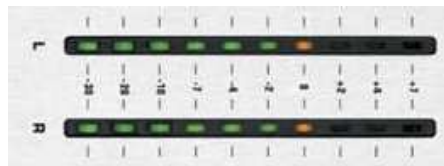
▪ **Projeto Termômetro**

Componentes: 1 Barra gráfica de LEDs + 1 Buzzer + 1 Sensor de Temperatura LM35

Descrição: Conforme a temperatura do ambiente onde o sensor LM35 está localizado aumenta, os leds da barra gráfica acendem, como um termômetro. Se existir uma temperatura muito alta, um alarme deverá soar.

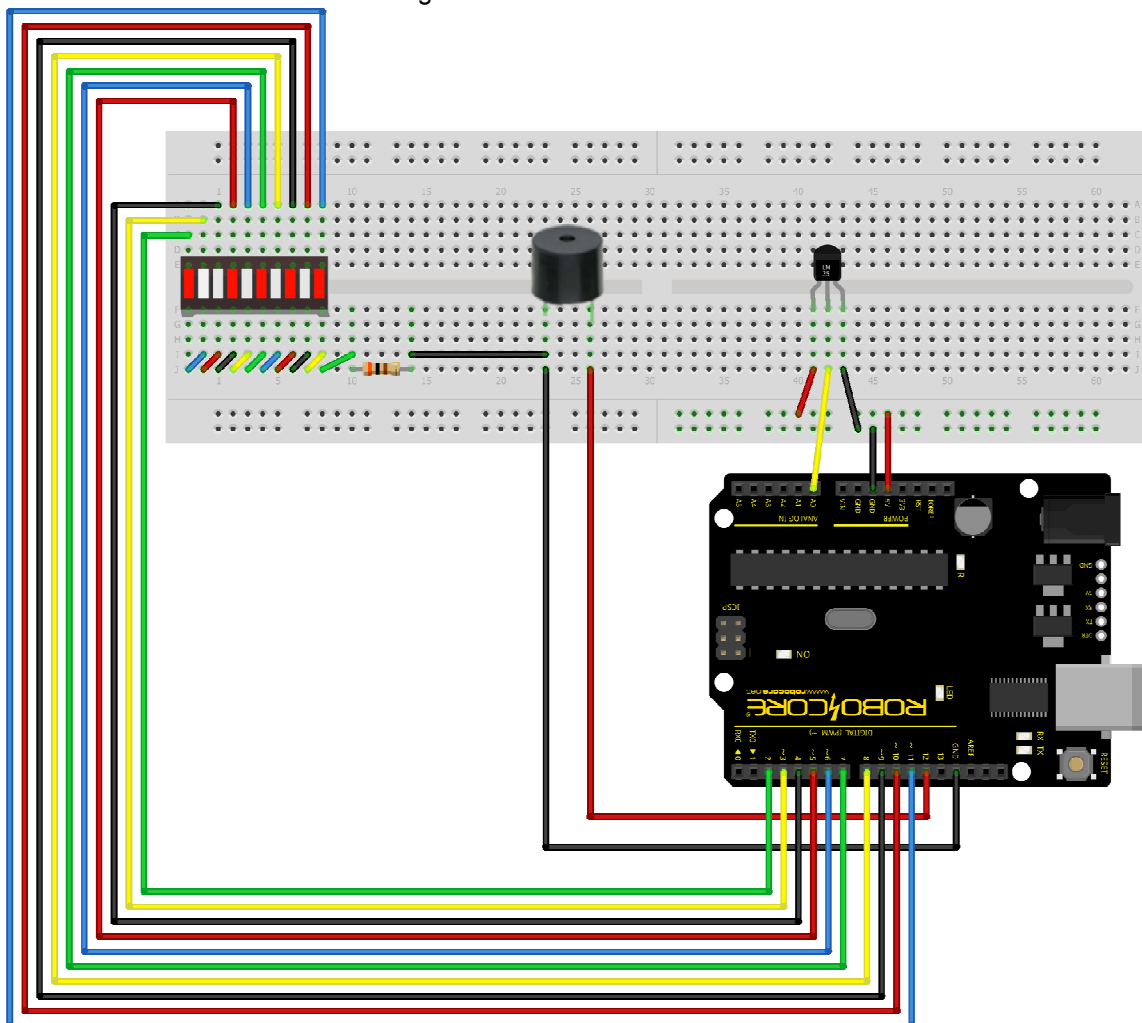
Dificuldade: 

Este projeto é, sem dúvida, muito bonito para os olhos tanto dos aficionados em eletrônica quanto das pessoas comuns. Neste projeto teremos o conceito de um *bargraph*, que nada mais é do que uma barra de leds que acendem conforme algum parâmetro. *Bargraphs* muito conhecidos são os de equipamentos de som. Quando o som está alto, ou com os graves altos, as luzes, como na figura a seguir:



Exemplo de um típico *bargraph* na horizontal

Componentes utilizados: 01x LM35 (sensor de temperatura) / 10 x Resistor de 300 Ω / 28x cabos diversos / 01x Buzzer 5V / 01x Barra gráfica



fritzing

Tanto o esquema de ligações quanto o código parecem ser mais complexos, portanto tenha muita calma e atenção para montar o esquema. Revise o circuito algumas vezes antes de ligá-lo.

Vamos começar desta vez passando o código para a placa Arduino:

```
/*
*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
**                                          **
**      Projeto Termômetro                  **
**                                          **
*****/\
const int LM35 = 0;
float temperatura = 0;
int ADClido = 0;
const int Buzzer = 12;
const int LED[] = {
  2,3,4,5,6,7,8,9,10,11};

void setup(){
  analogReference(INTERNAL);
  pinMode(Buzzer, OUTPUT);
  for(int x = 0; x < 10; x++){
    pinMode(LED[x], OUTPUT);
  }
}

void loop(){
  ADClido = analogRead(LM35);
  temperatura = ADClido * 0.1075268817204301;
  if(temperatura > 23.50){
    digitalWrite(LED[0], HIGH);
  }
  else{
    digitalWrite(LED[0], LOW);
  }

  if(temperatura > 24.00){
    digitalWrite(LED[1], HIGH);
  }
  else{
    digitalWrite(LED[1], LOW);
  }

  if(temperatura > 24.50){
    digitalWrite(LED[2], HIGH);
  }
  else{
    digitalWrite(LED[2], LOW);
  }

  if(temperatura > 25.00){
    digitalWrite(LED[3], HIGH);
  }
  else{
    digitalWrite(LED[3], LOW);
  }

  if(temperatura > 25.50){
    digitalWrite(LED[4], HIGH);
  }
  else{
    digitalWrite(LED[4], LOW);
  }

  if(temperatura > 26.00){
    digitalWrite(LED[5], HIGH);
  }
  else{
    digitalWrite(LED[5], LOW);
  }

  //continua na proxima pagina
}
```

```
//continuação:

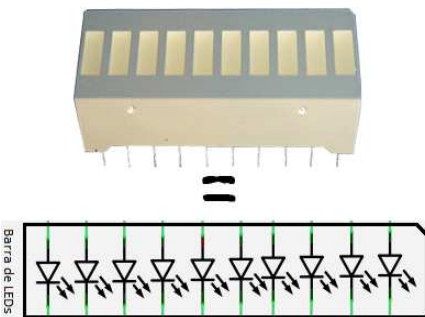
if(temperatura > 26.50){
  digitalWrite(LED[6], HIGH);
}
else{
  digitalWrite(LED[6], LOW);
}

if(temperatura > 27.00){
  digitalWrite(LED[7], HIGH);
}
else{
  digitalWrite(LED[7], LOW);
}

if(temperatura > 27.50){
  digitalWrite(LED[8], HIGH);
}
else{
  digitalWrite(LED[8], LOW);
}

if(temperatura > 28.00){
  digitalWrite(LED[9], HIGH);
  digitalWrite(Buzzer,HIGH);
}
else{
  digitalWrite(LED[9], LOW);
  digitalWrite(Buzzer,LOW);
}
}
}
```

NÃO SE ASSUTE! O código é grande, mas é completamente entendível. Vamos estudá-lo daqui a pouco. Agora vamos entender como funciona a barra de LEDs.



A barra de LEDs é um componente muito usado em bargraphs em todo o mundo, em diversos equipamentos tanto comerciais quanto industriais. Ela serve justamente para, de uma forma fácil e inteligível, verificar como estão níveis de motores, de combustíveis, de audio, etc. Este componente possui 20 pinos, 10 de cada lado, ou seja, a cada 2 pinos temos 1 LED, logo temos 10 LEDs ao todo. Para facilitar a ligação, ligamos os pinos de baixo da barra juntos para podermos usar apenas 1 resistor e limitar a corrente de todos os LEDs. Preste bastante atenção pois a barra tem polaridade (verifique a posição do chanfro no canto superior direito da barra).

O que mudou deste código para o do Projeto Alarme foi que adicionamos a barra gráfica de LEDs, que possui 10 LEDs. Desta forma, no começo do código tivemos que declarar onde estes leds estavam colocados no Arduino. Poderíamos muito bem declarar os pinos como fizemos com tudo até aqui, ou seja, um por um como é colocado a seguir:

```
const int LED1 = 2;
const int LED2 = 3;
const int LED3 = 4;
const int LED4 = 5;
const int LED5 = 6;
const int LED6 = 7;
```

```
const int LED7 = 8;
const int LED8 = 9;
const int LED9 = 10;
const int LED10 = 11;
```

Porém, para deixar o código mais enxuto, escrevemos da seguinte forma:

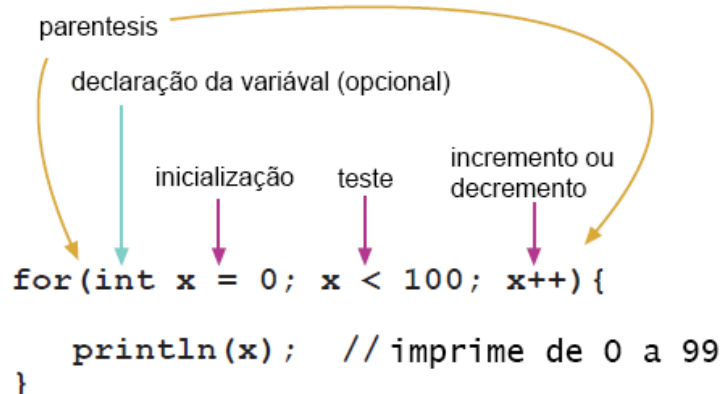
```
const int LED[] = {
  2,3,4,5,6,7,8,9,10,11};
```

Desta forma, criamos a variável LED em forma de **ARRAY**. Usamos este artifício pois fica mais fácil declarar e mexer nos LEDs, quando queremos ligar mais de um por vez. A única diferença na hora de chamar a variável é que ao invés de escrevermos **LED1** deveremos escrever **LED[0]** (zero é o primeiro número).

A próxima mudança foi dentro do **setup** do programa. Colocamos a seguinte instrução:

```
for(int x = 0; x < 10; x++){
  pinMode(LED[x], OUTPUT);
}
```

Esta instrução serve para economizarmos código na hora de dizer que todos os pinos onde estão os LEDs da barra são saída. Esta estrutura chama-se **FOR**. Sua estrutura é assim:



fonte: www.arduino.cc

O **for** é usado para repetir, por um determinado número de vezes, uma instrução. No caso do nosso código, a instrução: `pinMode(LED[x], OUTPUT);` é repetida 10 vezes, ou seja, de 0 a 10. Cada vez que é executada, o valor de X dentro dos colchetes muda. Portanto, este código faz exatamente o seguinte:

```
pinMode(LED[0], OUTPUT);
pinMode(LED[1], OUTPUT);
pinMode(LED[2], OUTPUT);
pinMode(LED[3], OUTPUT);
pinMode(LED[4], OUTPUT);
pinMode(LED[5], OUTPUT);
pinMode(LED[6], OUTPUT);
pinMode(LED[7], OUTPUT);
pinMode(LED[8], OUTPUT);
pinMode(LED[9], OUTPUT);
```

Ou seja, diz que os 10 LEDs estão em pinos que devem ser tratados como saída. Esta é mais uma estrutura para economizar código.

Agora entramos no **loop** do programa. O que mudou foi muito simples, colocamos mais algumas estruturas condicionais **if**, que, conforme determinada temperatura é atingida acendemos os LEDs na sequencia e quando o último LED está aceso ligamos o buzzer. Novamente, aumente a temperatura do sensor encostando os dedos nele.

Vamos agora mudar um pouco o foco dos projetos. Vamos aprender a fazer outro tipo de leitura analógica, utilizando um potenciômetro.

▪ **Módulo 4**

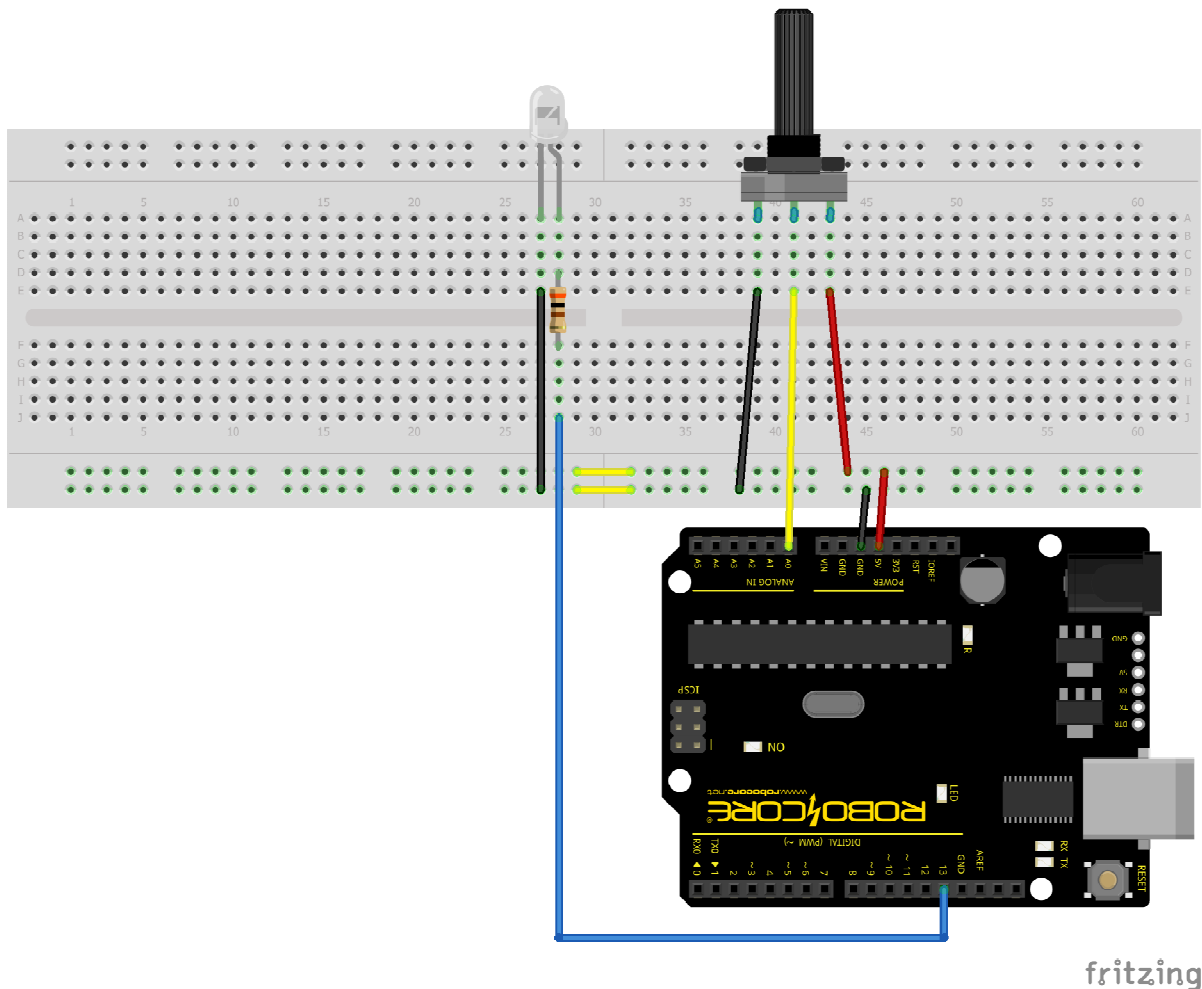
Componentes: 1 Potenciômetro + 1 Led

Descrição: Conforme o valor do potenciômetro é alterado, o led pisca de forma mais rápida ou mais lenta.

Dificuldade: 

Vamos voltar aos circuitos simples. Neste módulo faremos com que um led pisque mais rápido ou mais devagar conforme os parâmetros de um potenciômetro.

Componentes utilizados: 01x Potenciômetro de 10k / 01x Led de Alto Brilho / 01x Resistor de 300 / cabos diversos.



fritzing

Dica: Você consegue usar os próprios pinos do potenciômetro para ligar ele à sua protoboard.

Código:

```
/*  
*****\n**   ROBOCORE ARDUINO KIT INICIANTE   **  
*                                     *  
**           Módulo 4                 **  
\\*****\n  
const int PinoPotenciometro = A0;  
const int Led = 13;  
int ValorPot = 0;  
  
void setup() {  
  pinMode(Led, OUTPUT);  
}  
  
void loop() {  
  ValorPot = analogRead(PinoPotenciometro);  
  digitalWrite(Led, HIGH);  
  delay(ValorPot);  
  digitalWrite(Led, LOW);  
  delay(ValorPot);  
}
```

Este código deve ser de fácil entendimento. Primeiro declaramos que o pino do potenciômetro será o Analógico 0 e será constante:

```
const int PinoPotenciometro = A0;
```

Depois dizemos que teremos um led no pino13 e também será constante:

```
const int Led = 13;
```

Então declaramos uma variável do tipo inteira para armazenar os valores do potenciômetro. Veja que esta variável irá de 0 a 1023, pois estes são os valores que a entrada analógica pode variar por termos uma resolução de 10 bits. Neste caso estamos usando a referência interna do Arduino para as portas digitais, ou seja, os valores aceitos são até 5V.

Declaramos o pino do Led como saída no **setup**, como já feito na maioria dos nossos programas e então vem o **loop** principal:

```
void loop() {  
  ValorPot = analogRead(PinoPotenciometro);  
  digitalWrite(Led, HIGH);  
  delay(ValorPot);  
  digitalWrite(Led, LOW);  
  delay(ValorPot);  
}
```

Primeiramente, como já não tem mais segredo para nós, assimilamos o valor lido no Pino do Potenciômetro à variável ValorPot, ou seja, ao valor do potenciômetro. Então ligamos o Led. Esperamos um tempo, que varia de 0 a 1023 milissegundos porque são esses valores que nosso potenciômetro pode ter, e desligamos o led. Novamente esperamos o tempo e voltamos para a primeira instrução do **loop** para fazer a nova leitura de uma nova posição do potenciômetro.

Agora já estamos aptos a fazer um projeto muito comum nas iluminações residenciais hoje, o Dimmer.

▪ **Projeto Dimmer**

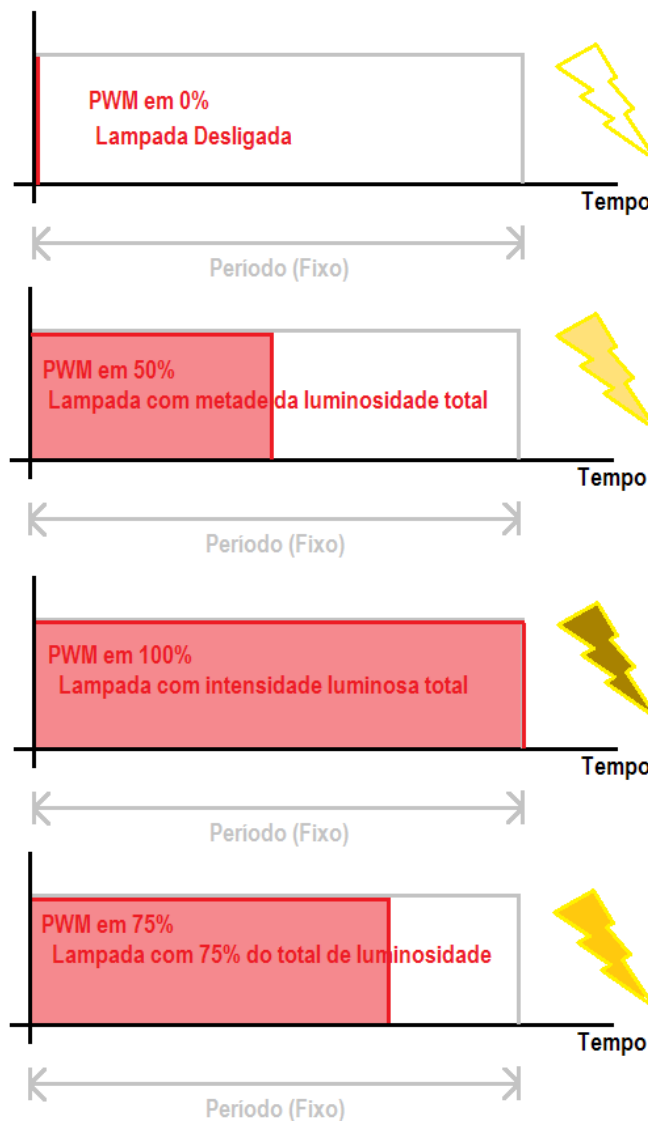
Componentes: 1 Potenciômetro + 1 Led Alto Brilho

Descrição: Conforme o valor do potenciômetro é alterado, o led fica mais claro ou mais escuro graças ao PWM.

Dificuldade: 

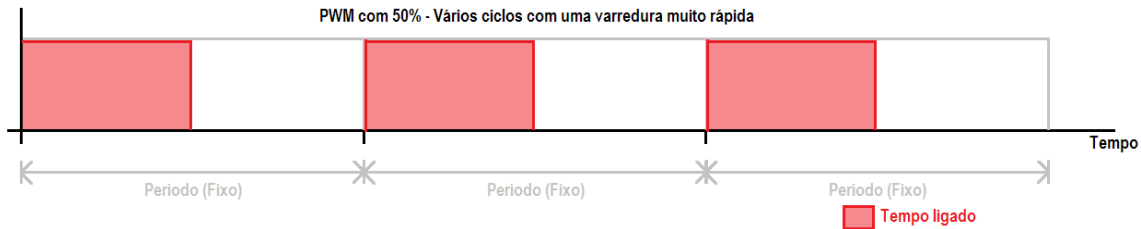
Este projeto é muito simples, mas é outro que dirá respeito a um conceito importantíssimo na eletrônica: o PWM. Esta sigla significa *Pulse Width Modulation*, ou seja, modulação por largura de pulso. De uma maneira bem simples, esta técnica pode ser explicada como: utilizando bases de tempo, conseguimos ligar e desligar uma porta tão rapidamente que para nossos olhos parece estar sempre ligado, e o que muda é a intensidade com a qual a porta está ligada.

A figura a seguir ilustra esta ideia:



O período é fixo. Por exemplo, se nosso período for 10 milissegundos e ligarmos o PWM em 50%, ou seja, 5 milissegundos ligado e 5 milissegundos desligado, veremos a lâmpada (ilustrada pelo raio

amarelo) acesa com metade da intensidade luminosa que teríamos se deixássemos a lâmpada ligada os 10 milissegundos. Acontece que existe um tempo de varredura, que quando um período chega ao fim, outro começa instantaneamente e a lâmpada fica ligada, como podemos ver na figura a seguir:

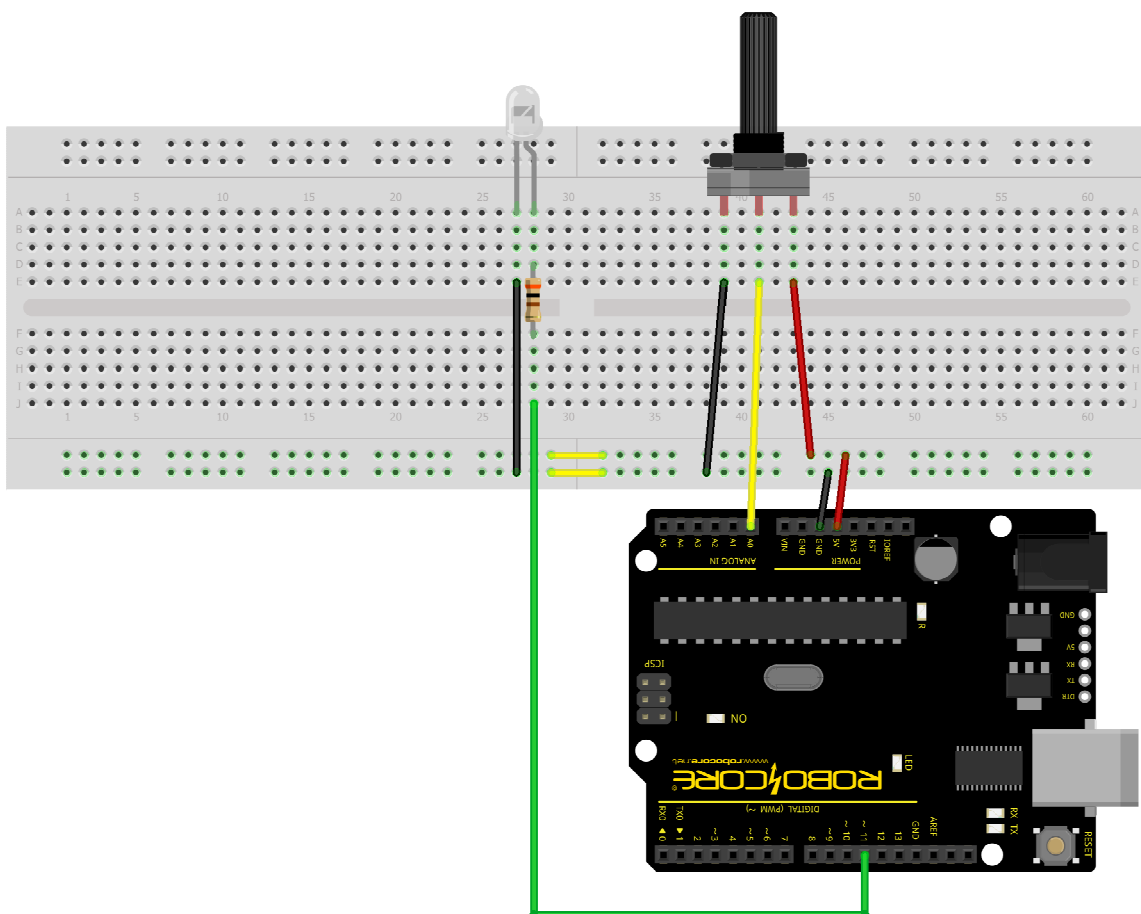


Como o tempo em que isso ocorre é muito rápido, não enxergamos a lâmpada ligar e desligar. Mas, se tivéssemos um período de 10 segundos, e deixássemos a lâmpada acesa 5 segundos e apagada outros 5 segundos veríamos o tempo aceso e apagado, pois nossos olhos conseguem distinguir o que são 5 segundos, mas não o que são 5 milissegundos.

No Arduino UNO, temos 6 saídas digitais que podem ser utilizadas como PWM. Neste projeto, vamos utilizar uma delas, no caso o pino 11.

O esquema a ser montado é muito semelhante ao anterior, apenas agora teremos que mudar o pino onde colocamos o led anteriormente.

Componentes utilizados: 01x Potenciômetro de 10k / 01x Led de Alto Brilho / 01x Resistor de 300 / cabos diversos.



fritzing

Código:

```
/*
*****\
**   ROBOCORE ARDUINO KIT INICIANTE   **
*                                       *
**           Projeto Dimmer           **
\*****/

const int PinoPotenciometro = A0;
const int Led = 11;
int ValorPot = 0;
int pwm = 0;

void setup() {
  pinMode(Led, OUTPUT);
}

void loop() {
  ValorPot = analogRead(PinoPotenciometro);
  pwm = map(ValorPot, 0, 1023, 0, 255);
  analogWrite(Led, pwm);
}
```

Este código possui duas estruturas ainda não vistas neste material. Vamos estudá-las passo a passo:

```
const int PinoPotenciometro = A0;
const int Led = 11;
int ValorPot = 0;
int pwm = 0;
```

Declaração de constantes e variáveis: os pinos onde estão o Potenciômetro e o Led (agora no pino 11) são constantes, ValorPot e pwm são variáveis.

```
void setup() {
  pinMode(Led, OUTPUT);
}
```

No **setup** setamos o pino do Led como saída.

```
void loop() {
  ValorPot = analogRead(PinoPotenciometro);
  pwm = map(ValorPot, 0, 1023, 0, 255);
  analogWrite(Led, pwm);
}
```

Aqui estão 2 estruturas ainda não vistas. Na primeira linha assimilamos o valor lido no Pino do Potenciômetro à variável `ValorPot`. Logo depois fazemos um mapeamento de uma variável. Isto significa que vamos redefinir os limites de valores de uma variável. Antes de continuar a mostrar como funciona este mapeamento, vamos estudar a estrutura **analogWrite**.

Como o próprio nome diz, esta estrutura faz uma escrita analógica, ou seja, faz aquela escrita variável para fazermos nosso PWM. No Arduino está predefinido que para ter 0% de PWM, basta você escrever: **analogWrite(pino a ser escrito, 0)**; do mesmo modo que, para escrever 100% de PWM, basta você escrever: **analogWrite(pino a ser escrito, 255)**, ou seja, na estrutura que o Arduino entende como PWM, os valores vão de 0 (mínimo, ou seja, 0%) até 255 (máximo, ou seja, 100%). Voltamos então para o mapeamento. Vamos entender esta estrutura: **pwm = map(ValorPot, 0, 1023, 0, 255)**;

Com isto, queremos dizer que a variável "pwm" irá receber valores mapeados da seguinte forma:

Variável Recebedora = map(Valor Lido, Mínimo do Potenciômetro, Máximo do Potenciômetro, Novo Mínimo definido por você, Novo Máximo definido por você)

Portanto,

Valor Lido vale ValorPot: é o valor lido anteriormente pela função **analogRead**;

Mínimo do Potenciômetro vale 0;

Máximo do Potenciômetro vale 1023;

Novo Mínimo definido por você vale 0;

Novo Máximo definido por você vale 255.

Procure ver se você entendeu que os valores da variável **pwm** irão variar de 0 a 255, conforme o potenciômetro varia de 0 a 1023.

Após compilar e fazer o upload deste projeto, você terá um Dimmer de leds. Use sua imaginação e conhecimentos para utilizar isto da maneira mais adequada a suas necessidades ou projetos.

▪ **Projeto Iluminação Automatizada**

Componentes: 1 Led Alto Brilho + 1 Sensor de Luminosidade LDR

Descrição: Se a iluminação ambiente, por qualquer motivo, diminuir ou apagar completamente, um led de alto brilho acende gradativamente.

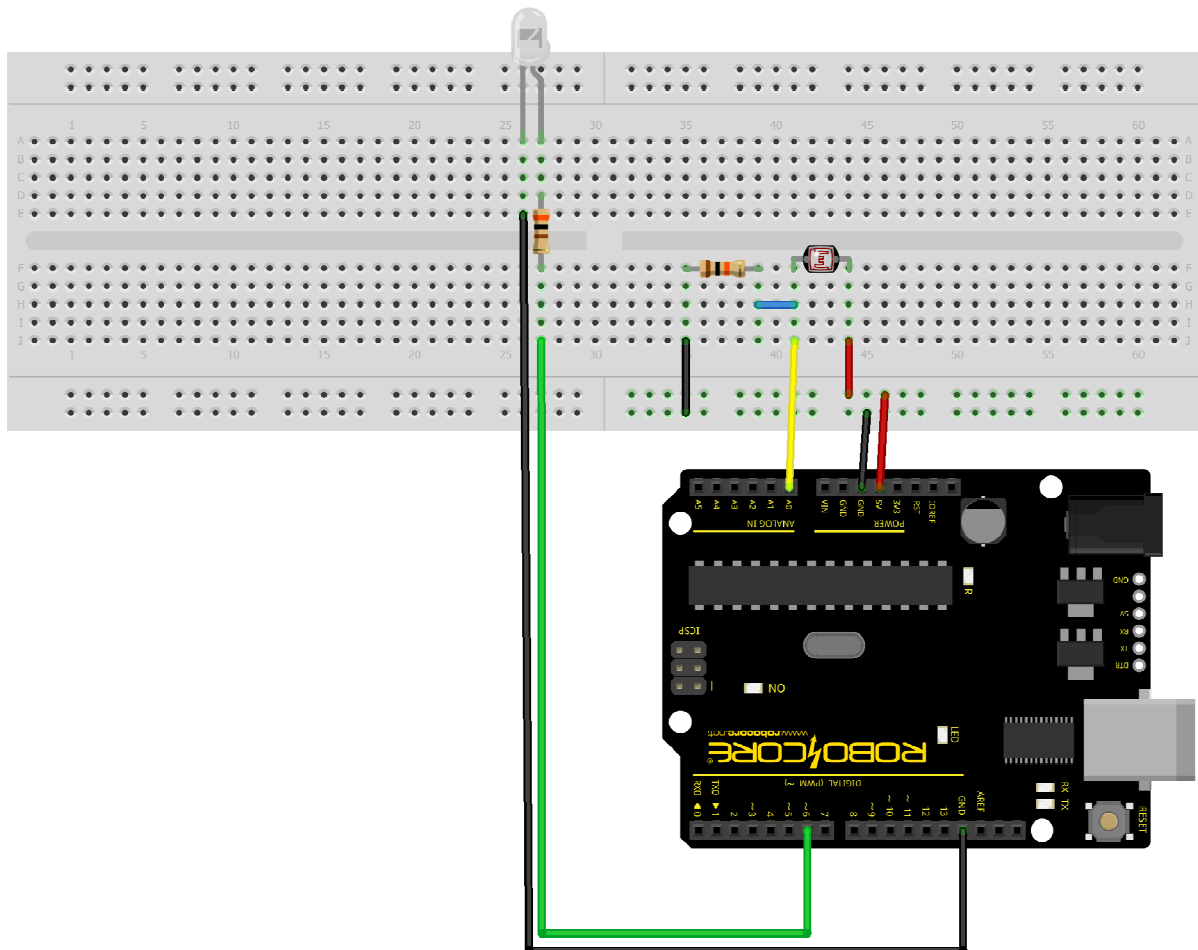
Dificuldade: 

Com conceitos de entradas analógicas, sensores e pwm, já podemos pensar em um projeto de automação. Projeto semelhante é utilizado em postes de luz, onde as lâmpadas acendem sozinhas, conforme a luminosidade do dia - ou você acha que todo dia uma pessoa responsável liga e desliga todas as luzes de todos os postes de todas as ruas?

Para este projeto, usaremos um LDR. LDR nada mais é do que uma resistência que varia conforme a luminosidade: é um sensor de luminosidade.

Monte o seguinte circuito:

Componentes utilizados: 01x LDR (sensor de luminosidade) / 01x Reistor de 10k / 01x Led de Alto Brilho / 01x Resistor de 300 / cabos diversos.



fritzing

Veja que o led está colocado no pino 6 digital e o sensor de luminosidade no pino 0 analógico. Antes de qualquer coisa, temos que calibrar o sensor.

Código:

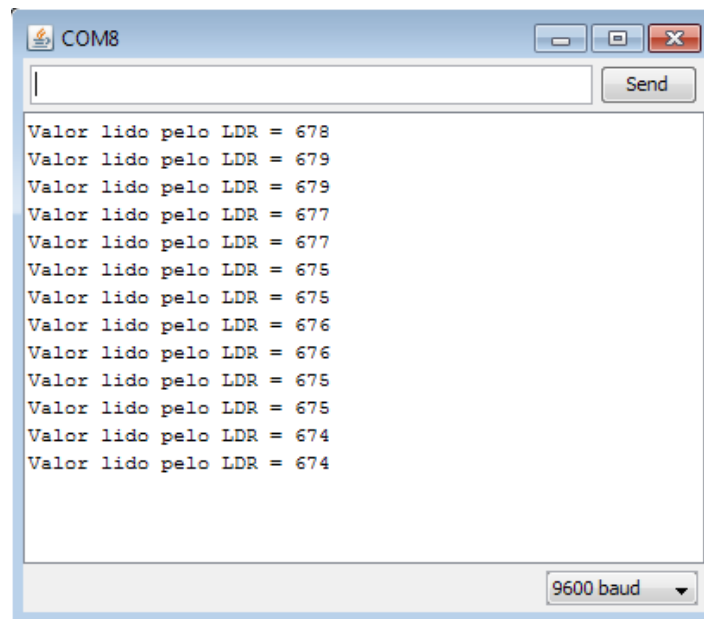
```
/*
*****
**   ROBOCORE ARDUINO KIT INICIANTE   **
*                                       *
**           Calibrar LDR              **
*****
*/

const int LDR = 0;
int ValorLido = 0;

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

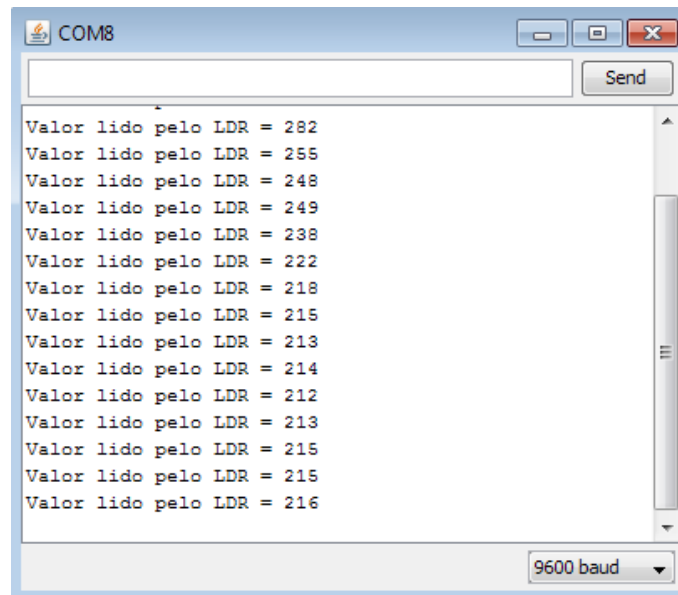
void loop() {
  ValorLido = analogRead(LDR);
  Serial.print("Valor lido pelo LDR = ");
  Serial.println(ValorLido);
  delay(500);
}
```

O código acima mostra no monitor serial os valores que o LDR está lendo. No caso da iluminação no local onde este material foi escrito, os valores lidos pelo LDR são os seguintes:



Você deve estar vendo em seu monitor algo parecido com esta figura acima. Se você está lendo um valor fixo de 1023 ou 0, certifique-se que os componentes estão bem colocados e na posição correta. Este é um erro muito comum neste tipo de experimento.

Coloque agora a palma da sua mão, ou qualquer outro material que tampe a luz ambiente, sobre o sensor tampando a luz e fazendo o sensor ficar na sombra. Você deve ler valores como os seguintes:



Como deve ter ficado subentendido:
 Quanto mais luz o LDR receber, mais alto será o valor lido.
 Quanto menos luz o LDR receber, menor será o valor lido.

Agora já temos os valores para calibrar nosso sensor. Vamos supor que você queira fazer com que um led acenda quando o valor lido é de 500 (uma sombra moderada sobre o LDR). Podemos então utilizar o seguinte código para fazer este projeto:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE  **
*                                     *
**  Projeto Iluminação Automatizada  **
\*****/

const int LDR = 0;
const int Led = 6;
int valorLido = 0;
int pwm = 0;

void setup() {
  pinMode(Led, OUTPUT);
}

void loop() {
  valorLido = analogRead(LDR);

  if (valorLido < 500){
    analogWrite(Led, pwm);
    pwm++;
    delay(100);
  }
  else{
    digitalWrite(Led, LOW);
    pwm = 0;
  }

  if(pwm > 255){
    pwm=255;
  }
}
  
```

A maior parte dos elementos deste código já foi estudada. Vamos para a parte que merece nossa atenção no **loop**:

```
void loop() {
  ValorLido = analogRead(LDR);

  if (ValorLido < 500){
    analogWrite(Led, pwm);
    pwm++;
    delay(100);
  }
  else{
    digitalWrite(Led, LOW);
    pwm = 0;
  }

  if(pwm > 255){
    pwm=255;
  }
}
```

Primeiramente assimilamos o valor lido pelo LDR com a variável ValorLido. Depois disso fazemos as seguintes condições:

SE a variável **ValorLido** for **MENOR** que 500 (uma leve sombra), **FAÇA:**

Escreva de uma maneira **ANALÓGICA**, ou seja, PWM no **Led** e

Some **1** na variável **pwm** (na linguagem C, colocar uma variável seguida de dois sinais de positivo significa somar 1 a esta variável), aguarde 100 milissegundos para podermos ver o efeito ocorrer;

SE NÃO (ou seja, Se ValorLido for MAIOR que 500), **FAÇA:**

Escreva de uma maneira **DIGITAL**, ou seja, alto ou baixo no **Led** e o apague (**LOW**), e também zere a variável **pwm** para que o efeito possa acontecer sempre que o led apagar;

A próxima condição serve apenas para garantir que a variável **pwm** não ultrapasse 255, pois, como já visto, para fazer escritas analógicas com **pwm** podemos usar valores indo de 0 a 255.

SE a variável **pwm** for **MAIOR** que 255, **FAÇA:**

pwm é **IGUAL** a 255 (desta forma garantimos que **pwm** nunca passará dos 255).

Pronto. Compile e faça o upload deste código juntamente com o circuito montado e veja que circuito útil você tem agora em mãos.

Agora podemos ir para o último projeto, e sem dúvida o mais complexo de todos.

Se você não teve dúvidas até agora está pronto para desenvolvê-lo. Se algo não saiu como os conformes, refaça quantas vezes for necessário o experimento.

Código:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
**      Projeto Alarme Multipropósito      **
**      *****/
const int LDR = 0;
const int LM35= 1;
const int Buzzer = 2;
const int led[] = {
  5,6,7,8,9,10,11};
int ValorLDR = 0;
int ADClido = 0;
float temperatura = 0;
int pwm = 0;

void setup(){
  for(int x = 0; x < 7 ; x++){
    pinMode(led[x], OUTPUT);
  }
  pinMode(Buzzer,OUTPUT);
}

void loop(){
  ValorLDR = analogRead(LDR);
  ADClido = analogRead(LM35);
  temperatura = ADClido * 0.48828125;
  if (temperatura > 20.00){
    digitalWrite(led[0], HIGH);
  }
  else{
    digitalWrite(led[0], LOW);
  }

  if (temperatura > 22.00){
    digitalWrite(led[1], HIGH);
  }
  else{
    digitalWrite(led[1], LOW);
  }

  if (temperatura > 24.00){
    digitalWrite(led[2], HIGH);
  }
  else{
    digitalWrite(led[2], LOW);
  }

  if (ValorLDR > 500){
    digitalWrite(led[5], HIGH);
  }
  else{
    digitalWrite(led[5], LOW);
  }
  if (ValorLDR > 400){
    digitalWrite(led[4], HIGH);
  }
  else{
    digitalWrite(led[4], LOW);
  }
  if (ValorLDR > 350){
    digitalWrite(led[3], HIGH);
    digitalWrite(led[6], LOW);
    digitalWrite(Buzzer, LOW);
  }
  else{
    digitalWrite(led[3], LOW);
    digitalWrite(led[6], HIGH);
    digitalWrite(Buzzer, HIGH);
  }
}

```


Neste ponto você já é capaz de entender perfeitamente o que se passa neste projeto, mesmo porque ele é apenas uma junção de alguns módulos vistos nesta apostila com alguns projetos.

Existe apenas uma pequena diferença do que fizemos até agora, você consegue encontrar qual é?

Nos primeiros experimentos com LM35, multiplicávamos o valor do ADC_{LIDO} por 0,1075268817204301 para encontrar a temperatura em graus Celsius após a leitura pelo conversor analógico digital, e agora estamos multiplicando por 0.4887585532746823. Você sabe dizer por quê?

Se você está atento, vai lembrar que nos primeiros experimentos com LM35 utilizamos um recurso chamado **analogReference(INTERNAL)** para mudar a referência das portas analógicas para um valor máximo de 1,1V. Neste experimento do alarme multipropósito não podemos usar este artifício e é claro que você sabe o porque disso. Nós temos que usar a referência das portas analógicas no modo padrão, ou seja, 5V, pois o sensor de luminosidade LDR envia valores para a entrada analógica de 0 a 5V, e se usássemos a referência de 1,1V provavelmente iríamos danificar nossa placa Arduino ao ler os dados do LDR. Chegamos no valor de 0.4887585532746823, pois fizemos aquela mesma regra de três anterior, mas com o valor máximo de 5V ao invés de 1,1V, ou seja, máximo de 500°C, portanto fica assim:

$$\begin{array}{ccc}
 500^{\circ}\text{C} & \times & 1023 \\
 \text{temperatura} & & \text{ADC}_{\text{LIDO}} \\
 \downarrow & & \\
 \text{temperatura} = \frac{500}{1023} \times \text{ADC}_{\text{LIDO}} & & \\
 \downarrow & & \\
 \text{temperatura} = 0.4887585532746823 \times \text{ADC}_{\text{LIDO}} & &
 \end{array}$$

A diferença de se usar o sensor desta forma é que ele não fica tão preciso quanto ficaria se usássemos a mudança da referência analógica.

Quanto aos valores das condições **IF**, se eles não estão satisfatórios para seu projeto, sinta-se a vontade para alterá-los como quiser. Uma dica é sempre fazer a calibragem do sensor que você for utilizar, com o código proposto no Projeto Iluminação Automatizada.

▪ **Módulo 5**

Componentes: 01 Display de LCD

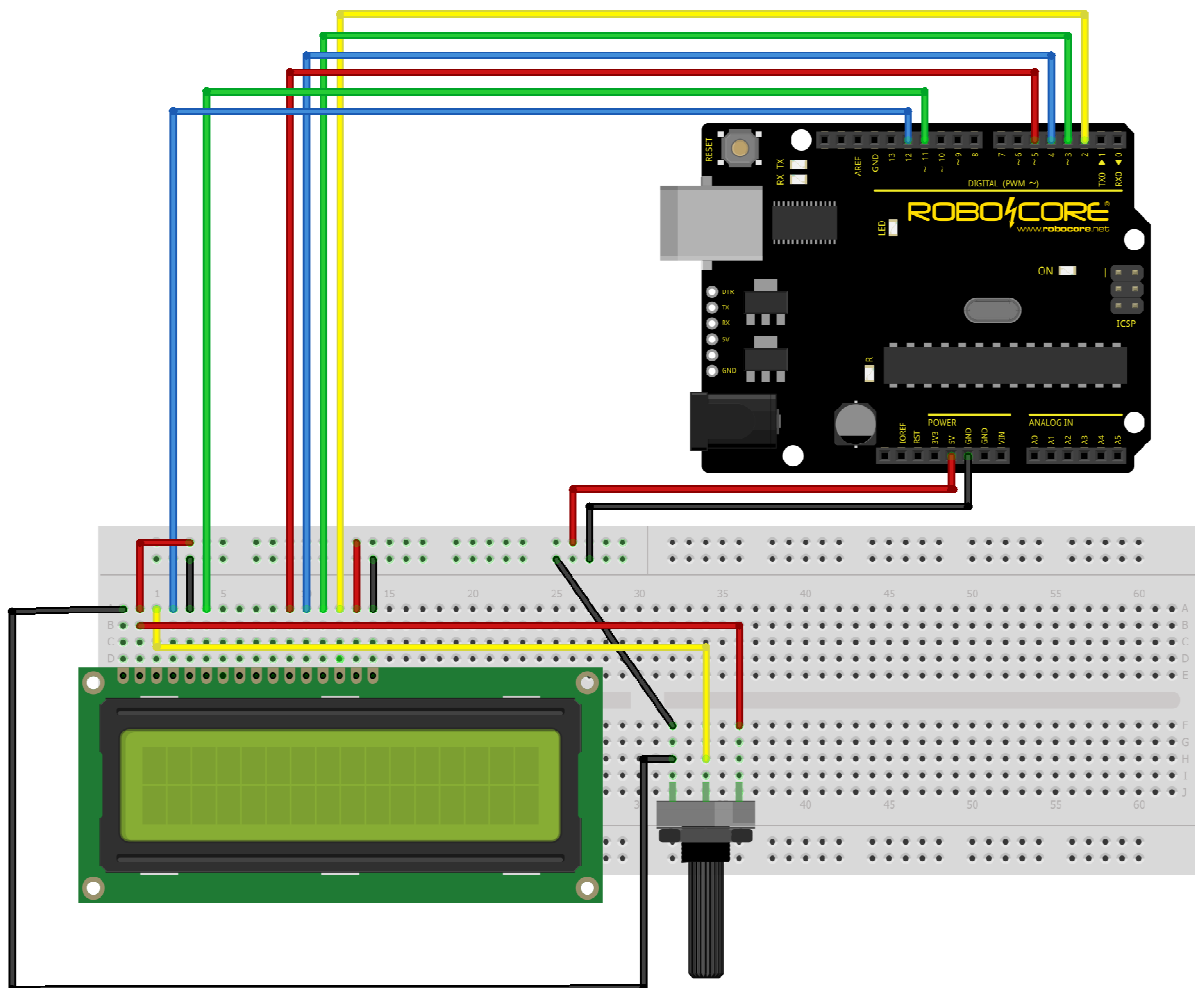
Descrição: Aprenda como usar este dispositivo, muito útil para mostrar os dados internos da placa Arduino em um modo inteligível, ou seja, possível de seres humanos entenderem.

Dificuldade: 

Podemos encontrar displays de LCD em milhares de equipamentos eletrônicos no mundo todo. Sejam os de 16x2 (como o que vem neste kit), como de outros tipos, com a letra preta ou de outras cores, estes displays tornam fácil o entendimento visual do que está acontecendo naquele determinado momento dentro do microcontrolador, sem a necessidade de um monitor e um computador. Pense no display de LCD como um "mini-monitor", nele você poderá ver dados de sensores em tempo real, poderá verificar o que aconteceu com o Arduino em um determinado período de tempo, como se fosse um LOG de eventos, ou mesmo conversar com seu Arduino (sim, conversar! Você pode fazer uma programação que, usando botões como os que vem neste kit você fala sim ou não e para o Arduino e pode inclusive fazer um jogo desta forma). Como você já deve ter percebido, agora que estamos na página 45 do manual, as possibilidades são infinitas!

Vamos começar com algo simples, o famoso *Hello World* do display. Vale lembrar que chamamos este display de LCD de 16x2 pois ele tem 16 colunas de dígitos e 2 linhas, ou seja, você tem duas linhas para escrever até 16 caracteres. Isto vai ficar mais óbvio quando virmos o dispositivo funcionando. Monte o seguinte circuito:

Componentes utilizados: 01x LCD 16x2 / 01x Potenciômetro 10k Ω / cabos diversos



fritzing

Preste muita atenção ao ligar o circuito anterior. Você irá precisar usar diversos cabos. Preste muita atenção também nos pinos do Display que irá utilizar (veja que o display deve ser colocado no canto esquerdo da protoboard para tornar mais fácil a contagem de furos). Veja que iremos usar os **6** primeiros pinos do LCD, depois pularemos os 4 próximos e então usaremos mais **4** pinos, deixando os **2** últimos sem usar. Para nossa aplicação, usaremos apenas estes **10** pinos.

Sem muita conversa, agora carregue o seguinte programa em seu Arduino:

```
/*
*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**              Módulo 5                  **
\*****/

#include <LiquidCrystal.h>

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

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("ola, meu nome e:");
  lcd.setCursor(0,1);
  lcd.print("xxx"); //Coloque seu nome no lugar dos xxx
}

void loop() {
}
```

Após gravar o código olhe para seu display. Caso não consiga enxergar nada, vá rodando o eixo do potenciômetro a fim de melhorar o contraste. Rode o eixo até encontrar um ponto em que as letras estejam legíveis.

Veja que em nossa primeira linha de código damos o seguinte comando:

```
#include <LiquidCrystal.h>
```

Até então nunca tínhamos visto isto. Com este comando nós incluímos (*#include*) uma biblioteca em nosso programa. O que seria isto? Uma biblioteca é um conjunto de comandos previamente feitos por alguém. No caso, a biblioteca **LiquidCrystal** foi desenvolvida pelo time que desenvolve a IDE do Arduino, portanto ela faz parte da IDE do Arduino. Nós podemos criar nossas próprias bibliotecas e/ou usar outras bibliotecas que não vieram junto no programa Arduino, para isto temos que efetuar um procedimento que é descrito no Arduino Kit Avançado. Neste momento, apenas devemos saber que o uso de bibliotecas facilita - e MUITO - o desenvolvimento de programas mais complexos com a plataforma Arduino.

Incluída a biblioteca para LCD (que seria esta **LiquidCrystal**), mostramos para o Arduino em que pinos de nosso Arduino está o display:

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

A biblioteca do LCD faz com que o programa do Arduino saiba que ao dar o comando **LiquidCrystal lcd** iremos utilizar um display, e faremos a comunicação utilizando os seguintes pinos: **12, 11, 5, 4, 3** e **2**. Caso você tenha curiosidade de saber o porque colocamos estes 6 pinos nesta linha de código, o convidamos a ler a fundo a biblioteca **LiquidCrystal**, a qual possui todas as explicações dos comandos.

No bloco de *SETUP* deste programa fazemos tudo o que precisamos para este exemplo, veja:

```
lcd.begin(16, 2);  
lcd.setCursor(0,0);  
lcd.print("Ola, meu nome e:");  
lcd.setCursor(0,1);  
lcd.print("xxx"); //Coloque seu nome no lugar dos xxx
```

Na primeira linha desta parte do código dizemos ao Arduino o tamanho de nosso display, ou seja, de quantas linhas e quantas colunas é nosso display de LCD. No caso, estamos usando um display com 16 colunas e 2 linhas. Existem muitos tipos de displays no mercado. Caso você fosse utilizar um display de 20 colunas e 4 linhas, esta linha de código ficaria assim:

```
lcd.begin(20, 4);
```

E então o programa já saberia o tamanho do LCD que você iria utilizar. Na linha seguinte do código temos o seguinte:

```
lcd.setCursor(0,0);
```

Nesta linha, dizemos para o Arduino qual a posição do próximo carácter que iremos escrever. A origem de tudo, ou seja, a coluna 0 e a linha 0 fica, por padrão, no canto superior esquerdo do display. Portanto, ao setar o cursor no ponto **0,0** iremos escrever no primeiro local disponível no display.

```
lcd.print("Ola, meu nome e:");
```

Nesta linha de código fazemos a escrita no display propriamente dita. Escrevemos a seguinte frase no display: Ola, meu nome e: (infelizmente o display não aceita caracteres especiais ou acentos no chipset que sai de fábrica, porém há como fazer a alteração mesmo sendo um processo um tanto quanto complexo). Veja como é simples escrever informações no display utilizando esta biblioteca. Simplesmente escrevemos no display como se estivéssemos escrevendo no Monitor Serial do Arduino, porém ao invés de utilizar **Serial.print("ola")** utilizamos **lcd.print("ola")**. Com esta frase nós já usamos os 16 caracteres do display (não se esqueça que espaço também conta como carácter), portanto temos que pular para a próxima linha para escrever mais alguma coisa:

```
lcd.setCursor(0,1);
```

Já vimos este comando nesta explicação e agora você consegue entender melhor como funciona. Quando colocamos o cursor no ponto **0,1** fazemos com que o próximo carácter a ser escrito vá para a coluna **0** na linha **1**, ou seja, desta forma pulamos para a segunda linha do display (lembrando que a primeira linha é discriminada como sendo **0** e a segunda linha como sendo **1**).

```
lcd.print("xxx"); //Coloque seu nome no lugar dos xxx
```

Se você não colocar seu nome onde está **XXX**, o display irá mostrar o seguinte:

```
Ola, meu nome e:  
XXX
```

Portanto coloque seu nome no lugar do **XXX** e, após passar o programa para o Arduino você verá seu nome escrito no display.

Este é um simples código introdutório, mas você já conseguiu perceber quão poderosa é esta nova ferramenta.

▪ **Módulo 6**

Componentes: 01 Display de LCD

Descrição: Aprenda a enviar caracteres do computador para o Arduino e vê-los no display.

Dificuldade: 

Que tal enviar comandos do computador para o Arduino e lê-los em seu display em tempo real? Vamos fazer isto utilizando o **mesmo circuito** do módulo anterior, porém com um código um pouco mais complexo. Grave o seguinte código em seu Arduino:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**              Módulo 6                  **
\*****/

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int incomingByte, x, y;

void setup() {
  lcd.begin(16, 2);
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    lcd.print(char(incomingByte));
    x = x + 1;
    y = y + 1;
    if(x > 15){
      lcd.setCursor(0,2);
      x = 0;
    }
    if(y > 31){
      lcd.setCursor(0,0);
      y = 0;
      x = 0;
    }
  }
}

```

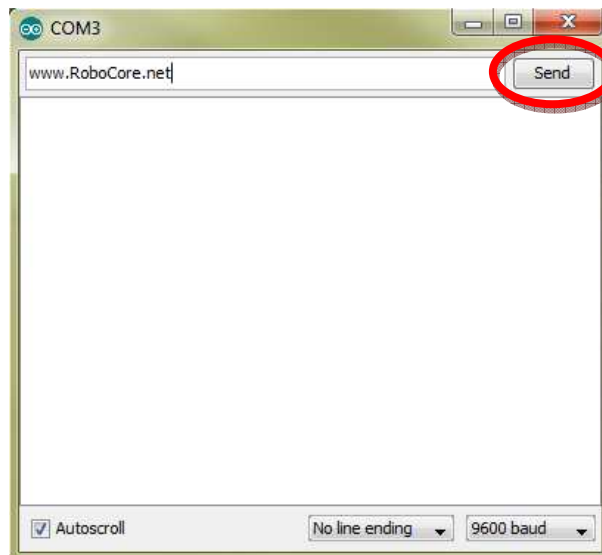
No começo do código incluímos novamente a biblioteca do LCD e dizemos para o Arduino em quais pinos o LCD está ligado. Após isto, declaramos três variáveis do tipo inteiro, são elas **incomingByte**, **x** e **y**. Explicaremos o porquê de declarar estas três variáveis no decorrer da explicação do código.

No bloco de *SETUP*, dizemos ao programa que iremos utilizar um display com 16 colunas e 2 linhas e logo depois começamos a comunicação serial, dando um comando **Serial.begin(9600)**. Lembrando que o intuito deste código é fazer o que escrevermos no teclado do computador aparecer no display de LCD, por isto temos que inicializar a comunicação serial do programa.

Neste programa, no bloco *LOOP* é onde acontece toda a mágica. Veja que no módulo 5 fazemos toda a programação no bloco *SETUP* do programa, isto porque não iremos alterar o que está escrito no display com o decorrer do programa. Poderíamos ter colocado aquele código no *LOOP*, porém é desnecessário uma vez que o que escrevemos não muda (iria consumir memória SRAM do Arduino à toa). Vamos ver o que temos no bloco *LOOP* deste programa:

```
void loop() {
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    lcd.print(char(incomingByte));
    x = x + 1;
    y = y + 1;
    if(x > 15){
      lcd.setCursor(0,2);
      x = 0;
    }
    if(y > 31){
      lcd.setCursor(0,0);
      y = 0;
    }
  }
}
```

Analisando linha por linha, na primeira linha deste bloco temos `if (Serial.available() > 0)` . Esta linha prepara o programa para possíveis caracteres que venham pela porta serial. Para usar este programa, teremos que abrir o Serial Monitor, como já fizemos em outros programas, e escrever naquela linha superior então clicar em **Send** conforme a seguinte imagem:



Ao clicar no **Send** (ou apertar a tecla ENTER no teclado), o programa do Arduino irá enviar pelo cabo USB estes caracteres para a placa Arduino e esta irá enviar para o display de LCD.

Voltando ao programa, naquela primeira linha podemos pensar que o programa faz o seguinte: caso venha algo pela serial, ou seja, caso algo venha pelo cabo USB, faça o seguinte:

```
incomingByte = Serial.read();
```

Nesta linha salvamos na variável **incomingByte** (byte que está chegando) o que estiver chegando pela porta serial, ou seja, quando você enviar algo pelo Monitor Serial, cada letra ficará salva momentaneamente na variável **incomingByte**. Na linha a seguir fazemos a mágica acontecer:

```
lcd.print(char(incomingByte));
```

Nós escrevemos no LCD com o comando **lcd.print** o dado que estiver salvo no **incomingByte**, porém não escrevemos simplesmente isto pois o que está salvo no **incomingByte** é o byte que representa a letra que enviamos pelo teclado. Se enviarmos para o LCD simplesmente o que está salvo nesta variável iremos ver no LCD o número que representa a letra do teclado. Não é isto que nós queremos, queremos ver a letra em si! Portanto colocamos o comando **char()** junto ao **incomingByte**. Este comando converte o número do código ASCII da letra do teclado para letras e

então mostramos no display a letra em si. Cada letra no teclado possui um código na tabela ASCII, caso você tenha curiosidade pesquise no Google por tabela ASCII.

O código que escreve os caracteres no display está pronto e entendido, o restante do código é apenas um pequeno "macete" para pular a linha na hora certa ou voltar para a linha anterior quando o espaço acabar no display.

Veja que cada vez que escrevemos uma letra no display, fazemos a seguinte soma:

```
x = x + 1;  
y = y + 1;
```

A variável **x** tomará conta de pular para a segunda linha e a variável **y** tomará conta de voltar para o começo da linha anterior, caso acabe o espaço para escrever na segunda linha. A cada caractere que escrevemos, somamos 1 unidade tanto na variável **x** quanto na **y**. Então fazemos as seguintes análises condicionais:

```
if(x > 15){  
  lcd.setCursor(0,2);  
  x = 0;  
}
```

e

```
if(y > 31){  
  lcd.setCursor(0,0);  
  y = 0;  
}
```

Na primeira condicional vemos se temos mais de 16 caracteres na linha. Se **x** chegar a 15, temos que pular para a próxima linha (lembre que a contagem começa em zero, portanto de zero a quinze temos dezesseis caracteres). Se a soma for maior que 15, setamos o cursor para a próxima linha com o comando `lcd.setCursor(0,2)`; e zeramos a variável que acumula o **x** com o seguinte comando `x = 0`; (se não zerássemos esta variável ela nunca saberia quando voltamos a ter mais que 16 caracteres). Em paralelo a isto temos a contagem de **y**. Esta contagem vai de 0 a 31, para dar 32 caracteres, ou seja, as duas linhas do LCD completamente cheias de letras. Caso a soma do **y** dê mais que 31, movemos o cursor do LCD para a origem do display e zeramos **y**.

Este tipo de "macete" acaba virando algo normal em sua programação quando você treina lógica de programação em seu dia-a-dia. Muitas vezes, algo que parece bobo pode resolver muitos problemas em programação.

▪ **Projeto Freqüencímetro**

Componentes: 01 Display de LCD + 02 Botões + 01 Buzzer

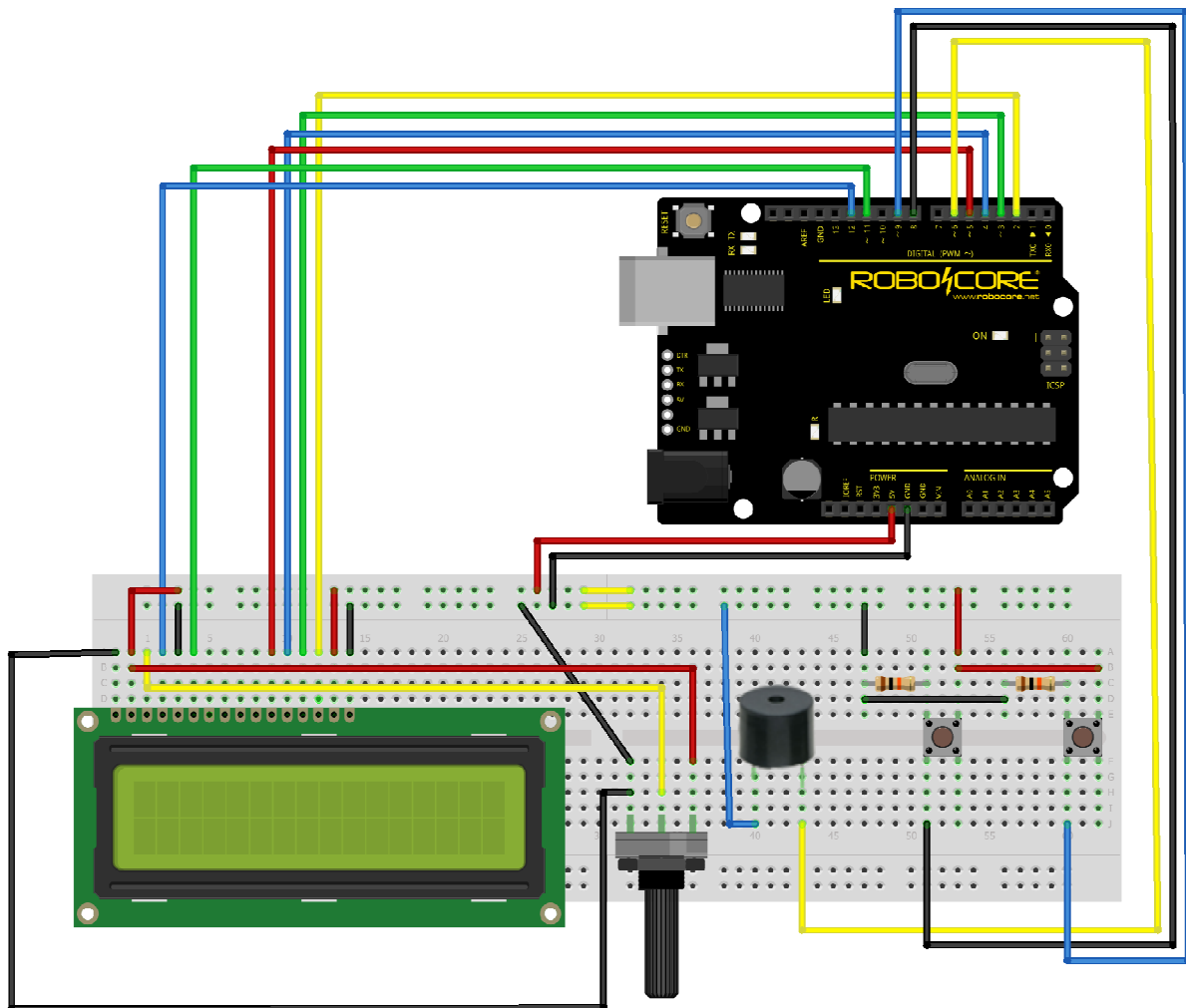
Descrição: Vamos fazer barulho novamente! Mas agora vamos variar a frequência e verificar a todo instante a quantos hertz estamos ouvindo o Buzzer tocar.

Dificuldade: 

Hora de fazer barulho novamente! Quem não gosta de fazer barulho com Arduino? Agora iremos ouvir alguns barulhos que os mais velhos irão lembrar dos antigos jogos de Atari®. Iremos fazer um freqüencímetro. Infelizmente não teremos uma frequência com exatidão pois estamos usando um processador com baixa velocidade e um buzzer, que não emite sons muito fiéis.

Monte o seguinte circuito, com muita calma e atenção (trata-se de um circuito complexo e cheio de fios para ligar, portanto se não houver atenção e paciência, o circuito pode ser ligado errado e pode não funcionar):

Componentes utilizados: 01x LCD 16x2 / 01x Buzzer 5V / 01x Potenciômetro 10k Ω / 02x PushButton / 02x Resistor 10k Ω.



fritzing

Devemos colocar o Buzzer com seu terminal positivo no pino 6 (um pino que pode ser usado como saída PWM).

O código deste projeto é um tanto quanto complexo, mas após ler todas as páginas deste manual você não terá muitos problemas para entender. Copie o seguinte código e grave em seu Arduino:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**      Projeto Freqüencímetro              **
\*****/

#include <LiquidCrystal.h>

int freq = 0;
int Botao1 = 8;
int Botao2 = 9;
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int Buzzer = 6;
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("Frequencia = ");
  lcd.setCursor(6,1);
  lcd.print("Hz");
  pinMode(Botao1, OUTPUT);
  pinMode(Botao2, OUTPUT);
  pinMode(Buzzer, OUTPUT);
}

void loop() {
  EstadoBotao1 = digitalRead(Botao1);
  EstadoBotao2 = digitalRead(Botao2);

  if (EstadoBotao1 == HIGH){
    freq = freq + 100;
  }

  if (EstadoBotao2 == HIGH){
    freq = freq - 100;
  }

  if(freq <= 0){
    freq = 0;
  }

  if(freq >= 20000){
    freq = 20000;
  }

  if(freq <= 99){
    lcd.setCursor(0,1);
    lcd.print(" ");
    lcd.setCursor(1,1);
    lcd.print(" ");
    lcd.setCursor(2,1);
    lcd.print(" ");
    lcd.setCursor(3,1);
  }

  if(freq >= 100){
    lcd.setCursor(0,1);
    lcd.print(" ");
    lcd.setCursor(1,1);
    lcd.print(" ");
    lcd.setCursor(2,1);
  }

  if(freq >= 1000){
    lcd.setCursor(0,1);
    lcd.print(" ");
    lcd.setCursor(1,1);
  }

  if(freq >= 10000){
    lcd.setCursor(0,1);
  }

  lcd.print(freq);
  tone(Buzzer, freq) ;
  delay(100);
}

```

E, novamente, na primeira linha do programa invocamos a biblioteca do LCD, sem esta biblioteca seria extremamente complicado escrever caracteres no display.

Depois disto, declaramos diversas variáveis e mostramos ao Arduino onde está ligado nosso display de LCD:

```
int freq = 20;
int Botao1 = 8;
int Botao2 = 9;
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int Buzzer = 6;
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Na primeira linha declaramos uma variável do tipo inteira, a variável `freq`, que irá ser a variável que guardaremos a frequência "apitada" pelo Buzzer. Depois, declaramos onde estão os dois botões (pinos 8 e 9) e logo em sequência declaramos duas variáveis que irão guardar o estado atual do botão (como visto anteriormente). Depois falamos que o buzzer está ligado ao pino 6 do Arduino. Então falamos para o Arduino quais pinos estamos usando para falar com o LCD.

Temos então os seguintes comandos no bloco *SETUP*:

```
lcd.begin(16, 2);
lcd.setCursor(0,0);
lcd.print("Frequencia = ");
lcd.setCursor(6,1);
lcd.print("Hz");
pinMode(Botao1, OUTPUT);
pinMode(Botao2, OUTPUT);
pinMode(Buzzer, OUTPUT);
```

Novamente, na primeira linha mostramos para o Arduino qual o tipo do display que vamos usar, com 16 colunas e 2 linhas. Então setamos o cursor para a origem do display e então escrevemos o seguinte: "Frequencia = ". Setamos então o cursor para a posição 6 da segunda linha para escrever a unidade de frequência, também conhecida como hertz, ou simplesmente Hz. Veja que escrevemos tudo isto no bloco *SETUP* pois são informações estáticas, ou seja, não irão se alterar conforme o programa for rodando. No final do bloco dizemos para o Arduino que os pinos onde estão os botões e o Buzzer são de saída.

Entrando no bloco de *LOOP* do programa temos o seguinte logo de início:

```
EstadoBotao1 = digitalRead(Botao1);
EstadoBotao2 = digitalRead(Botao2);
```

Estes dois comandos anteriores, bem como as rotinas IF que seguem já vimos anteriormente, e basta entendermos que, quando apertamos um botão (o de incremento) somaremos 100 à variável **freq** e quando apertamos o outro botão, subtraímos 100. Este 100 é o passo de nosso sistema, caso queira maior variedade de frequências, troque este valor para 10 ou mesmo 1. Caso queira menor variedade de frequências, troque para 1000, por exemplo.

Vamos ver as seguintes linhas:

```
if(freq <= 0){
    freq = 0;
}

if(freq >= 20000){
    freq = 20000;
}
```

Nas duas linhas anteriores limitamos a variável **freq**. Seu valor mínimo passa a ser 0 e seu valor máximo passa a ser 20000 (valor próximo ao limite escutado pelo ouvido humano).

A seguinte parte do código é apenas mais um "macete" para que apareçam no display apenas os dígitos da frequência atual, que você setar pelo potenciômetro, e não fique no display valores antigos (precisamos de dar um jeito de apagar os dígitos no display, para isto simplesmente escrevemos espaços!):

```
if(freq <= 99){  
  lcd.setCursor(0,1);  
  lcd.print(" ");  
  lcd.setCursor(1,1);  
  lcd.print(" ");  
  lcd.setCursor(2,1);  
  lcd.print(" ");  
  lcd.setCursor(3,1);  
}
```

```
if(freq >= 100){  
  lcd.setCursor(0,1);  
  lcd.print(" ");  
  lcd.setCursor(1,1);  
  lcd.print(" ");  
  lcd.setCursor(2,1);  
}
```

```
if(freq >= 1000){  
  lcd.setCursor(0,1);  
  lcd.print(" ");  
  lcd.setCursor(1,1);  
}
```

```
if(freq >= 10000){  
  lcd.setCursor(0,1);  
}
```

Tente estudar e ver o que acontece nas linhas de código acima. O "macete" é parecido com o que fizemos no último experimento.

Por fim, temos as seguintes linhas de código:

```
  lcd.print(freq);  
  tone(Buzzer, freq) ;  
  
  delay(100);
```

Na primeira linha, escrevemos no display a frequência. A rotina **tone()** serve para tocar frequências em um pino. Para usar este comando você deve passar dois parâmetros para o mesmo, são eles:

1º: Pino que irá tocar a frequência, o pino deve ser do tipo saída PWM. No nosso caso colocamos a palavra Buzzer, que já está atrelada ao pino 6.

2º: Frequência em hertz que o Buzzer deverá tocar. No nosso experimento, esta frequência vai de 0 a 20000Hz.

No final do programa temos um **delay()** de 100 milissegundos para que possamos ler no display de LCD a frequência que estamos escutando.

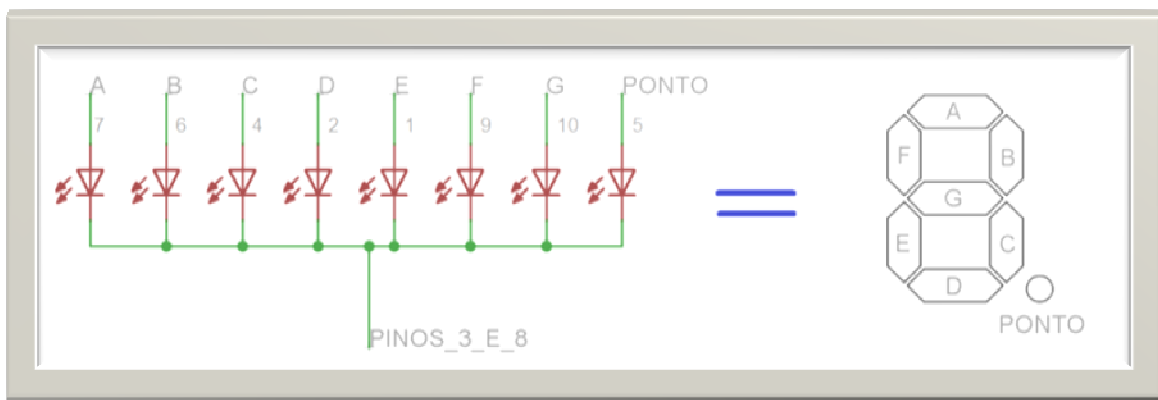
- **Display de 7 Segmentos**

Componentes: 01 Display de 7 Segmentos + 01 CI 4511

Descrição: Aprenda a utilizar o display de 7 segmentos juntamente a um conversor BCD para display. Neste experimento será explicado os fundamentos básicos de um circuito integrado e contagem binária.

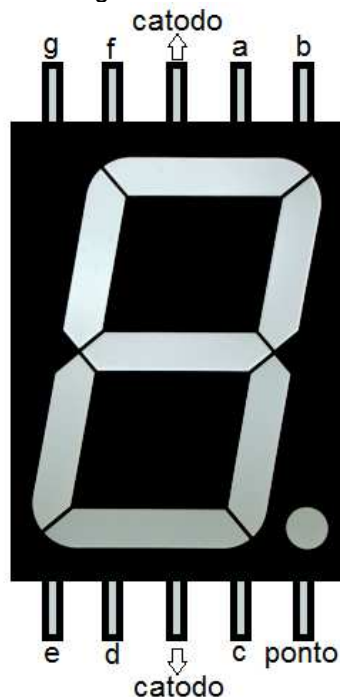
Dificuldade: 

Este experimento irá mostrar como utilizar um display de 7 segmentos. Este tipo de display é amplamente utilizado em relógios digitais, eletrodomésticos, equipamentos industriais, contadores, é muito visto em filmes fazendo contagem decrescente de tempo, e por aí vai. O circuito encapsulado no display de 7 segmentos é extremamente simples. Trata-se apenas de 8 leds ligados com um ponto em comum. É possível encontrar no mercado, displays de catodo comum e de ânodo comum. O que iremos utilizar neste experimento é do tipo catodo comum, ou seja, cada terminal catodo de cada led está ligado entre si. Para ilustrar, dê uma olhada na imagem abaixo. Ela é auto-explicativa.

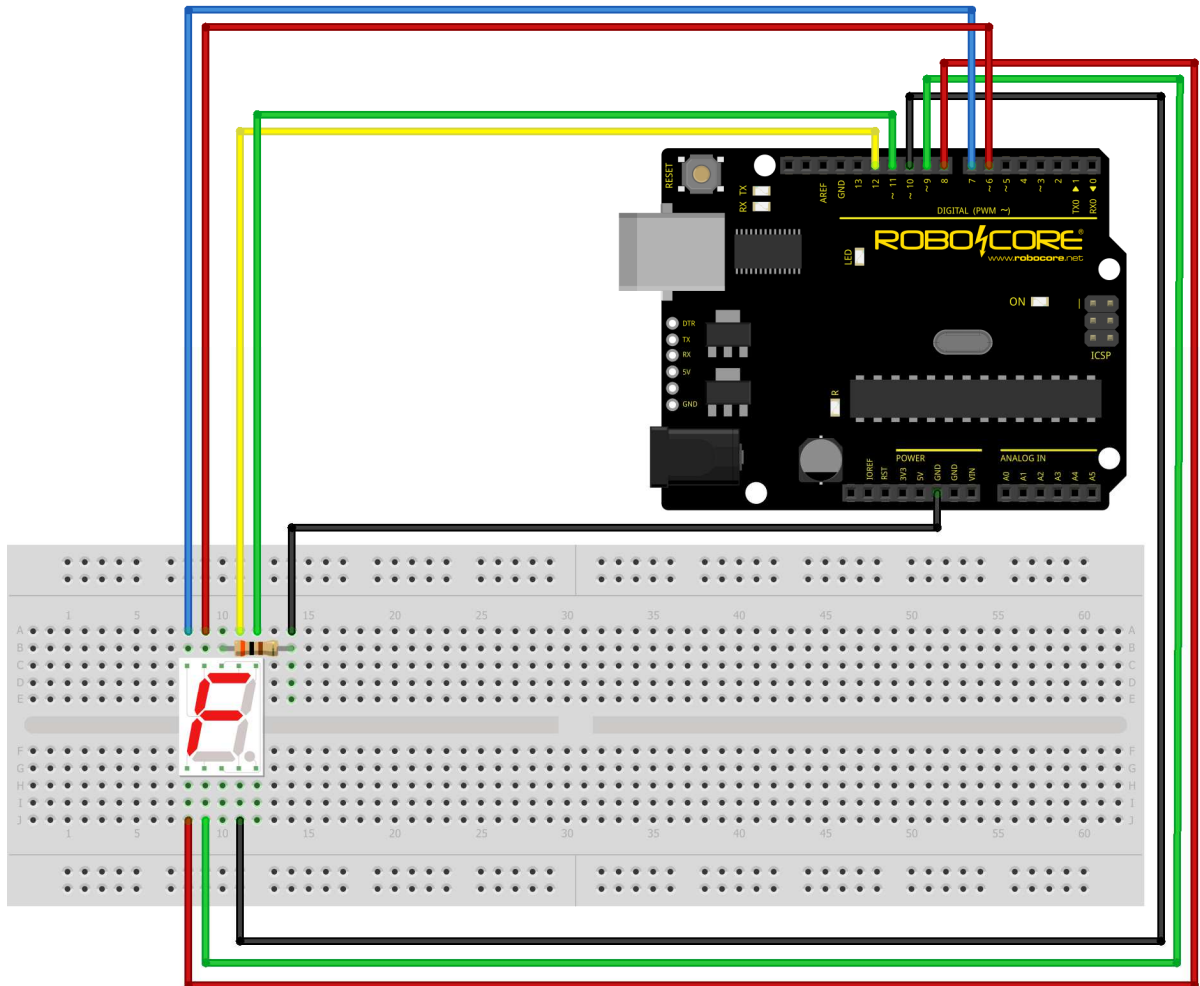


Como se pode ver, não existe um resistor que limita a corrente no encapsulamento, portanto deve ser usado um resistor afim de não queimar os leds devido a corrente. Utilize para isto, resistores de 300Ω.

A posição dos pinos é obtida conforme a figura abaixo:



Como o display de 7 segmentos é simplesmente um conjunto de leds, nada nos impede de tratá-los como simples leds! Portanto, poderíamos fazer um contador com a placa Arduino utilizando 7 saídas digitais. Obviamente, este não é o melhor método de se utilizar este display, porém, caso você não tenha um *conversor BCD para display de 7 segmentos*, esta se torna uma boa saída. Primeiramente, vamos fazer um circuito deste tipo com o Arduino. Monte o seguinte circuito com o Arduino, o display e um resistor:



fritzing

Não esqueça o resistor de 300Ω no pino do catodo do circuito! Não faz diferença colocar no pino de cima ou no pino de baixo, pois os dois pinos de catodo estão ligados juntos.

Grave no Arduino o seguinte código:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**      Display 7 Segmentos 1              **
\*****/

const int A = 12; // Primeiramente setamos os 7 pinos
const int B = 11;
const int C = 10;
const int D = 9;
const int E = 8;
const int F = 6;
const int G = 7;

void setup(){
  pinMode(A, OUTPUT); // seta todos as portas que estão os leds do display como saída
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
  pinMode(D, OUTPUT);
  pinMode(E, OUTPUT);
  pinMode(F, OUTPUT);
  pinMode(G, OUTPUT);
}

void loop(){
  digitalWrite(A, HIGH); //acende os leds que representam o número 0
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(G, LOW);
  delay(1000); //aguarda 1 segundo para mostrar próximo número

  digitalWrite(A, LOW); //acende os leds que representam o número 1
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, LOW);
  digitalWrite(E, LOW);
  digitalWrite(F, LOW);
  digitalWrite(G, LOW);
  delay(1000); //aguarda 1 segundo para mostrar próximo número

  digitalWrite(A, HIGH); //acende os leds que representam o número 2
  digitalWrite(B, HIGH);
  digitalWrite(C, LOW);
  digitalWrite(D, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(F, LOW);
  digitalWrite(G, HIGH);
  delay(1000); //aguarda 1 segundo para mostrar próximo número

  digitalWrite(A, HIGH); //acende os leds que representam o número 3
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(F, LOW);
  digitalWrite(G, HIGH);
  delay(1000); //aguarda 1 segundo para mostrar próximo número

  digitalWrite(A, LOW); //acende os leds que representam o número 4
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, LOW);
  digitalWrite(E, LOW);
  digitalWrite(F, HIGH);
  digitalWrite(G, HIGH);
  delay(1000); //aguarda 1 segundo para mostrar próximo número

//continua

```

```

//continuacao:

digitalwrite(A, HIGH); //acende os leds que representam o número 5
digitalwrite(B, LOW);
digitalwrite(C, HIGH);
digitalwrite(D, HIGH);
digitalwrite(E, LOW);
digitalwrite(F, HIGH);
digitalwrite(G, HIGH);
delay(1000); //aguarda 1 segundo para mostrar próximo número

digitalwrite(A, LOW); //acende os leds que representam o número 6
digitalwrite(B, LOW);
digitalwrite(C, HIGH);
digitalwrite(D, HIGH);
digitalwrite(E, HIGH);
digitalwrite(F, HIGH);
digitalwrite(G, HIGH);
delay(1000); //aguarda 1 segundo para mostrar próximo número

digitalwrite(A, HIGH); //acende os leds que representam o número 7
digitalwrite(B, HIGH);
digitalwrite(C, HIGH);
digitalwrite(D, LOW);
digitalwrite(E, LOW);
digitalwrite(F, LOW);
digitalwrite(G, LOW);
delay(1000); //aguarda 1 segundo para mostrar próximo número

digitalwrite(A, HIGH); //acende os leds que representam o número 8
digitalwrite(B, HIGH);
digitalwrite(C, HIGH);
digitalwrite(D, HIGH);
digitalwrite(E, HIGH);
digitalwrite(F, HIGH);
digitalwrite(G, HIGH);
delay(1000); //aguarda 1 segundo para mostrar próximo número

digitalwrite(A, HIGH); //acende os leds que representam o número 9
digitalwrite(B, HIGH);
digitalwrite(C, HIGH);
digitalwrite(D, LOW);
digitalwrite(E, LOW);
digitalwrite(F, HIGH);
digitalwrite(G, HIGH);
delay(1000); //aguarda 1 segundo para mostrar próximo número
}
    
```

Quase duas páginas inteiras de código para fazer um display de 7 segmentos fazer uma simples contagem de 0 a 9 com intervalos de 1 segundo? Deve ter algum modo mais fácil, não? Sim! Graças a um pequeno circuito integrado não é necessário perder espaço na memória de seu microcontrolador nem tantas portas para programar apenas o display. Juntamente ao Arduino Kit Iniciante você recebeu um CI 4511, ou seja, um circuito integrado do tipo 4511. Circuitos integrados são compostos de milhares de conjuntos de transistores (de um tamanho comparável a um grão de areia) que, através da lógica entre eles, desempenham funções específicas. A função específica do CI 4511 é a de *conversor BCD para display de 7 segmentos*. Conveniente não? Para saber mais sobre este, e qualquer outro CI, você deve procurar pelo **datasheet** do mesmo. Os *datasheets*, como o nome diz, são folhas de dados onde são apresentados todas as características do circuito integrado. Faça uma busca de “datasheet 4511” no Google e veja o arquivo .PDF que você irá encontrar. Mas... o que seria um conversor BCD para display de 7 segmentos?

Pra começar, o que é BCD?

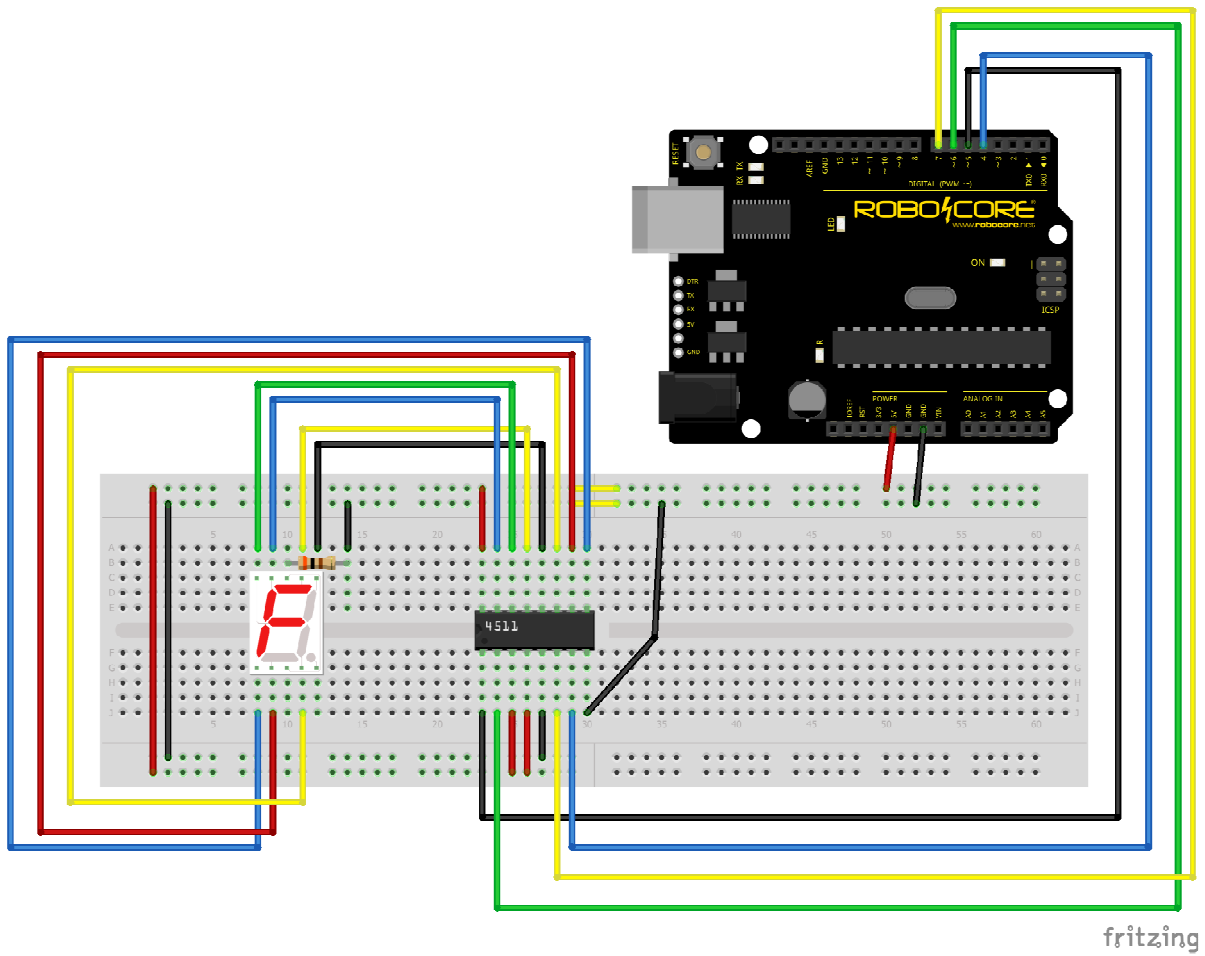
Entende-se pela sigla BCD o termo *binary-coded decimal*, ou seja, é um número decimal, porém em formato binário. Um número binário é um número que é representado apenas por números 0 e

números 1. Para ter uma explicação aprofundada deste assunto, é amplamente aconselhável procurar em livros de eletrônica e microeletrônica. Apenas para o conhecimento, e para dar continuidade a este manual, veja na tabela abaixo como este chip entenderá cada número colocado em suas entradas:

N.º em Binário	N.º em Decimal
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9

O conceito de número binário é algo muito extenso para ser tratado aqui. Por isto é sempre aconselhado procurar fontes de leitura externas para aprimorar seus conhecimentos. A saber para continuar os experimentos: números binários são tratados na eletrônica como níveis. Um número 0 é tratado como nível lógico baixo e um número 1 é tratado como nível lógico alto. O que seria nível lógico? Nada mais é do que DESLIGADO (nível lógico 0) e LIGADO (nível lógico 1) - já pensamos nesta forma quando colocamos a instrução **digitalWrite** nos códigos que fizemos. Em um circuito como o Arduino, nível lógico 0 seria 0V de tensão, ou seja, nenhuma tensão, a porta estaria desligada. Nível lógico 1 seria 5V nas portas digitais, ou seja, tensão total, a porta estaria ligada. Este zero e um é comumente visto em chaves liga/desliga (ou você achava que aquela bolinha e aquele traço vertical na chave significava alguma outra coisa se não um 0 e um 1, de desligado e ligado?). Toda esta explicação foi feita, pois na entrada do circuito integrado 4511 é necessário introduzir sinais binários para que o display acenda o número desejado. O usuário mais esperto já percebeu que, se introduzir, por exemplo, o número binário **0110** no circuito integrado, ele responderá ligando automaticamente um número **6** no display, pois ele é um *conversor BCD para display de 7 segmentos*! E como inserimos **0110** no CI? Usando 4 pinos digitais do Arduino, deixando um deles com 0V, outro com 5V, outro com 5V e outro com 0V. Ou seja, quatro comandos **digitalWrite**.

Vamos verificar como funciona o circuito, trabalhando com este CI. Monte o seguinte circuito:



ATENÇÃO: Fique muito atento à ligação do circuito integrado. Se ligar um fio errado, pode danificar o componente de tal forma que pode queimá-lo e deixá-lo inutilizável!

Grave o seguinte código em seu Arduino:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE      **
*                                       *
**          Display 7 Segmentos 2      **
\*****/

const int a = 4;
const int b = 5;
const int c = 6;
const int d = 7;

void setup(){
pinMode(a, OUTPUT);
pinMode(b, OUTPUT);
pinMode(c, OUTPUT);
pinMode(d, OUTPUT);
}
void loop(){
digitalwrite(a, LOW); //DIGITO 0
digitalwrite(b, LOW);
digitalwrite(c, LOW);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, HIGH); //DIGITO 1
digitalwrite(b, LOW);
digitalwrite(c, LOW);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, LOW); //DIGITO 2
digitalwrite(b, HIGH);
digitalwrite(c, LOW);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, HIGH); //DIGITO 3
digitalwrite(b, HIGH);
digitalwrite(c, LOW);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, LOW); //DIGITO 4
digitalwrite(b, LOW);
digitalwrite(c, HIGH);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, HIGH); //DIGITO 5
digitalwrite(b, LOW);
digitalwrite(c, HIGH);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, LOW); //DIGITO 6
digitalwrite(b, HIGH);
digitalwrite(c, HIGH);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, HIGH); //DIGITO 7
digitalwrite(b, HIGH);
digitalwrite(c, HIGH);
digitalwrite(d, LOW);
delay(1000);
digitalwrite(a, LOW); //DIGITO 8
digitalwrite(b, LOW);
digitalwrite(c, LOW);
digitalwrite(d, HIGH);
delay(1000);
digitalwrite(a, HIGH); //DIGITO 9
digitalwrite(b, LOW);
digitalwrite(c, LOW);
digitalwrite(d, HIGH);
delay(1000);
}

```

O código está um pouco mais enxuto, mas o que mais diferencia do código anterior é que agora estamos usando apenas 4 saídas digitais. Ou seja, utilizando a contagem binária e o circuito integrado 4511, nós ganhamos mais espaço em nossa memória do microcontrolador e temos mais entradas/saídas disponíveis em nossa placa Arduino para colocar mais componentes no projeto.

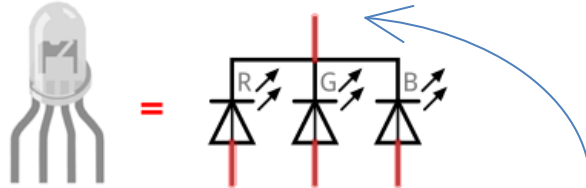
▪ **Projeto RGB**

Componentes: 01 LED RGB

Descrição: Aprenda como utilizar o led RGB, que possui em um mesmo encapsulamento três leds com as cores primárias. Através de PWM você poderá ver diversas combinações de cores.

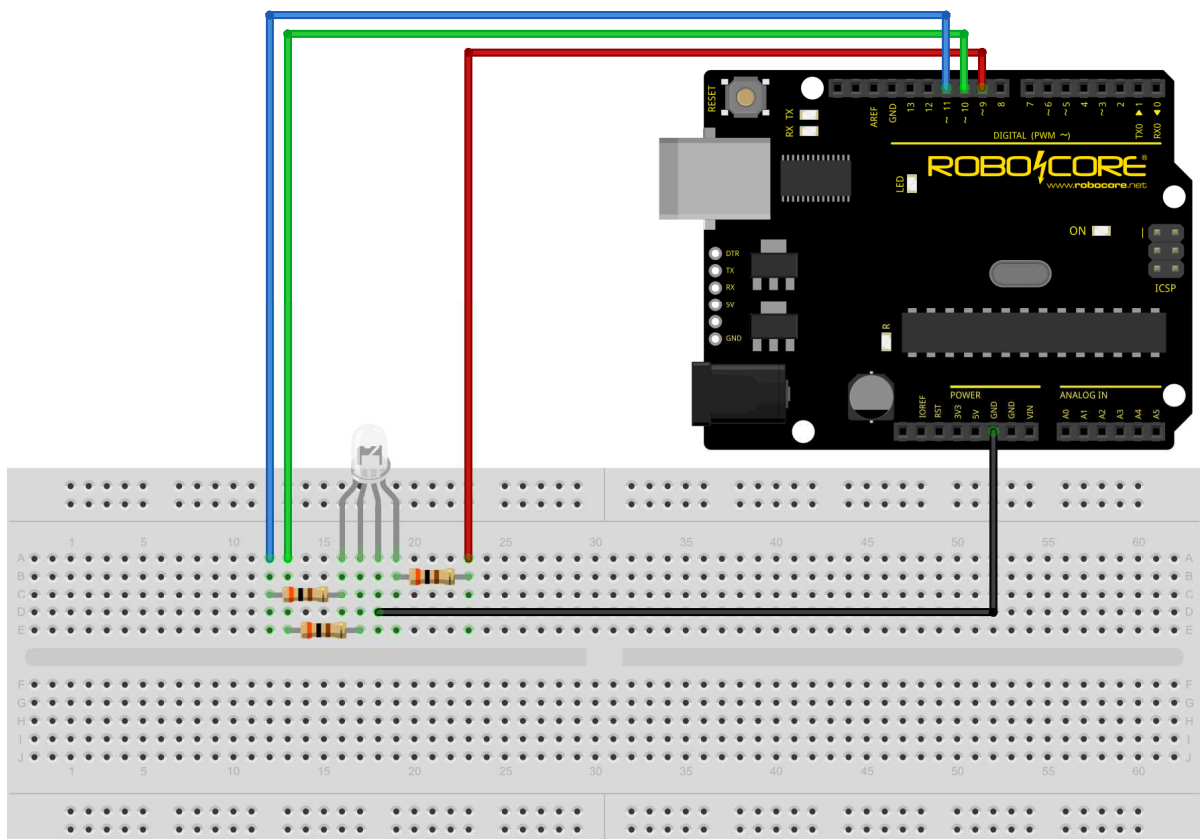
Dificuldade: 

Um led RGB nada mais é do que três leds juntos, um vermelho (daí a letra R de **RED**), um verde (daí a letra G de **GREEN**) e um azul (daí a letra B de **BLUE**). Combinando estas três cores em um só encapsulamento, podemos apreciar as mais diversas cores que as combinações de vermelho, verde e azul podem nos oferecer.



O maior pino do LED é comum a todos os LEDs internos, ou seja, o pino "que vai pra cima" no esquema acima. Como este LED é **catodo comum**, o pino comum deve ir diretamente para **GND**, enquanto que os outros vão para **5V** quando você quiser acendê-los. Por exemplo, se quiser apenas a cor **verde**, você precisará colocar o pino comum em GND e o pino referente ao **G** em 5V. Dessa forma, você vai ter uma diferença de tensão de 5V entre os pinos, fazendo a cor indicada acender. Alterando as tensões entre os três pinos, você terá diversas combinações de cores.

Monte o seguinte circuito, e nunca se esqueça de utilizar 3 resistores de **300Ω** que servem como limitadores de corrente:



fritzing

O código a seguir acende cada uma das cores separadamente, com intervalos de 500 milissegundos, ou 0,5 segundos (meio segundo). Nesta altura você consegue entender perfeitamente como funciona esse programa e poderia, até mesmo, criá-lo sozinho (mas faremos isso dessa vez):

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**              Led RGB 1                  **
\*****/

const int R = 9;
const int G = 10;
const int B = 11;

void setup(){
  pinMode(R, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(B, OUTPUT);
}

void loop(){
  digitalWrite(R, HIGH);
  digitalWrite(G, LOW);
  digitalWrite(B, LOW);
  delay(500);
  digitalWrite(R, LOW);
  digitalWrite(G, HIGH);
  digitalWrite(B, LOW);
  delay(500);
  digitalWrite(R, LOW);
  digitalWrite(G, LOW);
  digitalWrite(B, HIGH);
  delay(500);
}

```

Este teste foi extremamente fácil de fazer e compreender. Vamos passar para algo um pouco mais, no mínimo, bonito – e quanto mais bonito, mais complexo. Grave o seguinte código no Arduino:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**              Led RGB 2                  **
\*****/

const int R = 9;
const int G = 10;
const int B = 11;
int ValorRed = 255 ;
int ValorGreen = 0 ;
int ValorBlue = 0 ;

void setup(){
  pinMode(R, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(B, OUTPUT);
  analogWrite(R, ValorRed);
  analogWrite(G, ValorGreen);
  analogWrite(B, ValorBlue);
}

void loop(){
  for (ValorGreen = 0; ValorGreen <255; ValorGreen=ValorGreen+5){
    analogWrite(G, ValorGreen);
    delay(50);
  }
  for (ValorRed = 255; ValorRed > 0; ValorRed=ValorRed-5){
    analogWrite(R, ValorRed);
    delay(50);
  }
  for (ValorBlue = 0; ValorBlue < 255; ValorBlue=ValorBlue+5){
    analogWrite(B, ValorBlue);
    delay(50);
  }
}
//continua

```

```
//continuacao
for (valorGreen = 255; valorGreen > 0; valorGreen=valorGreen-5){
  analogwrite(G, valorGreen);
  delay(50);
}
for (valorRed = 0; valorRed < 255; valorRed=valorRed+5){
  analogwrite(R, valorRed);
  delay(50);
}
for (valorBlue = 255; valorBlue > 0; valorBlue=valorBlue-5){
  analogwrite(B, valorBlue);
  delay(50);
}
}
```

Veja se você compreende este programa. Ele simplesmente utiliza a técnica de PWM, que aprendemos no experimento do projeto Dimmer, através da palavra chave **analogWrite**, para "escrever" analogicamente diversos valores de tensão nos terminais do led RGB. Quando a função **analogWrite** "escreve" em um pino do led o valor de 255, quer dizer que terá uma tensão de 5V passando por este terminal. Analogamente, quando a função **analogWrite** escreve em um terminal o valor de 127, quer dizer que terá uma tensão de aproximadamente 2,5V passando por este terminal. Dessa forma, vamos rodando todas as cores do espectro RGB, e fazendo junções entre elas usando rotinas **for**, as quais vimos anteriormente.

Entrando mais a fundo no código, no começo declaramos os pinos onde os 3 LEDs internos estão ligados. Então declaramos três variáveis que guardarão o valor, que vai de 0 a 255, que "escreveremos" em cada terminal do LED RGB. Veja que a variável **ValorRed** começa com 255, enquanto que as demais começam com 0. Dessa forma, assim que ligamos o Arduino, o LED começa com a cor vermelha acesa. Logo depois a cor começa a mudar, pois começamos a injetar um pouco de verde no LED, até chegar ao seu máximo com o valor de 255 (primeira rotina **for** do **loop**). Veja que incrementamos a cor de 5 em 5 unidades e também colocamos um delay de 50 milissegundos entre cada incremento. Se você quiser deixar o procedimento mais lento, pode diminuir o incremento de 5 para 1, e também pode aumentar o delay (é um experimento interessante a se fazer). Então, ao final do primeiro **for** temos a cor vermelha no máximo e a cor verde também no máximo. A junção dessas duas cores primárias, dá a cor amarela. Então começamos o segundo **for**. Nele, diminuímos toda a cor vermelha, então ao final dele você enxerga apenas a cor verde. No terceiro **for**, temos a cor azul acendendo até chegar ao seu máximo. No final do terceiro **for** temos a junção de verde com azul, que dá um azul bem claro. Então chegamos no quarto **for**. Neste diminuímos totalmente a cor verde, e no final dele temos apenas a cor azul. Chegamos ao quinto **for**, onde aumentamos até o máximo a cor vermelha e no final deste teremos a junção de azul com vermelho, que dá a cor violeta. No sexto e último **for**, diminuímos ao máximo a cor azul, e no final dele temos o mesmo que tínhamos no início, ou seja apenas a cor vermelha. Então começamos tudo novamente.

Você consegue ver quantas cores diferentes vemos neste experimento? Sem dúvida, um dos experimentos mais bonitos deste material!

Dica: coloque algo branco sobre o LED RGB para conseguir enxergar melhor as cores formadas, como um copo descartável branco.

Projeto Timer

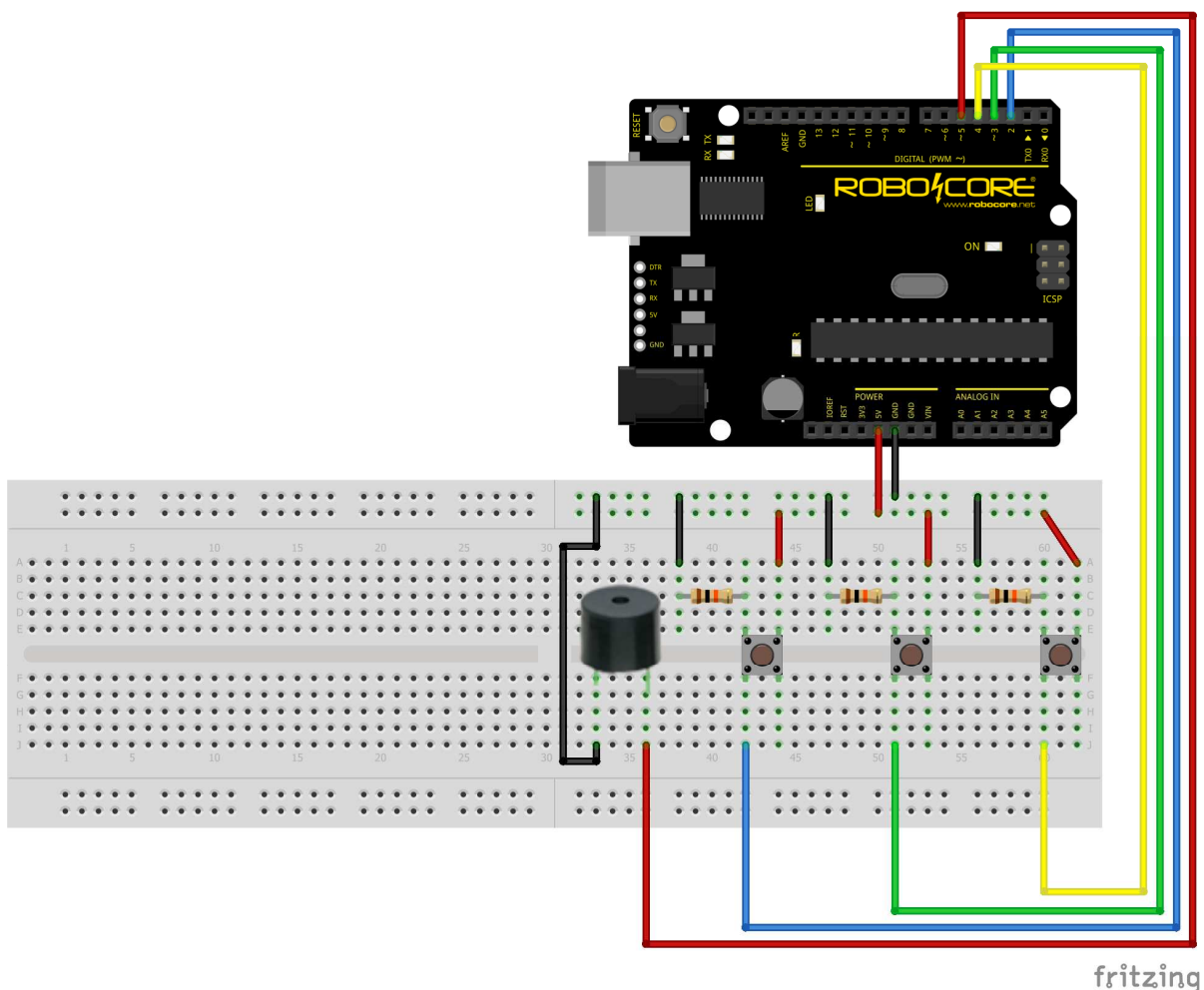
Componentes: 01 Buzzer + 03 PushButtons + 01 Potenciometro + 01 LCD

Descrição: Aprenda como fazer um timer com seu Arduino, onde você seleciona o tempo e ouve um aviso sonoro quando o tempo se esgota.

Dificuldade: 

Este projeto é muito legal. Nele você vai usar apenas conceitos já aprendidos no decorrer desta apostila e fará um timer, ou seja, um relógio decrescivo, no qual através de 2 botões você seleciona quantos segundos e quantos minutos quer que o mesmo tenha. Passado o tempo do timer, ele nos informará visivelmente que o tempo acabou e também através de um timer.

Para começar, iremos visualizar todas as informações no monitor serial. Portanto, a primeira montagem que devemos fazer na protoboard é a seguinte:



fritzing

Sugerimos fazer a montagem assim como está na imagem acima, pois ainda vamos colocar o LCD na protoboard.

Esta montagem parece bastante a montagem do projeto piano que fizemos no começo desta apostila. Como sempre, preste bastante atenção nas ligações e utilize a menor quantidade de fios possíveis, pois vamos usar praticamente todos neste experimento.

Quanto a programação desta primeira parte do projeto, vamos dar uma olhada no seguinte código:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**      Projeto Timer no Monitor Serial      **
\*****/

int segundo = 0;
int minuto = 0;
const int LED13 = 13;
const int Buzzer = A2;
const int Botao1 = A5;
const int Botao2 = A4;
const int Botao3 = A3;
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int EstadoBotao3 = 0;

void setup(){
  pinMode(Botao1, INPUT);
  pinMode(Botao2, INPUT);
  pinMode(Botao3, INPUT);
  pinMode(LED13,OUTPUT);
  pinMode(Buzzer,OUTPUT);
  Serial.begin(9600);
  Serial.println("Selecione o tempo do timer...");
  Serial.println("Minutos: 0");
  Serial.println("Segundos: 0");
}

void loop(){

  EstadoBotao1 = digitalRead(Botao1);
  if (EstadoBotao1 == HIGH){
    delay(250);
    if (EstadoBotao1 == HIGH){
      segundo++;
      if(segundo>=60){
        segundo=0;
      }
      Serial.print("Segundos: ");
      Serial.println(segundo);
    }
  }

  EstadoBotao2 = digitalRead(Botao2);
  if (EstadoBotao2 == HIGH){
    delay(250);
    if (EstadoBotao2 == HIGH){
      minuto++;
      if(minuto>=60){
        minuto=0;
      }
      Serial.print("Minutos: ");
      Serial.println(minuto);
    }
  }

  EstadoBotao3 = digitalRead(Botao3);
  if (EstadoBotao3 == HIGH){
    if((minuto==0) && (segundo==0)){
      //nao faz nada
    }
  }

  // continua

```

```
//continuacao
else{
  Serial.println("START");
  delay(1000);

  if(segundo==0){
    minuto--;
    segundo=59;
  }

  for(int y = 0; y<segundo; y--){
    if(minuto<10){
      Serial.print("0");
      Serial.print(minuto);
      Serial.print(":");
    }
    else{
      Serial.print(minuto);
      Serial.print(":");
    }
    if(segundo<10){
      Serial.print("0");
      Serial.println(segundo);
    }
    else{
      Serial.println(segundo);
    }

    segundo--;

    if(segundo<0){
      minuto--;
      segundo=59;
    }

    delay(1000); //base de tempo de 1 segundo
    //para deixar timer mais rápido, diminuir aqui

    if((segundo<=0) && (minuto<=0)) {

      Serial.println("ACABOU O TEMPO!");

      digitalWrite(Buzzer,HIGH);
      delay(250);
      digitalWrite(Buzzer,LOW);
      delay(100);
      digitalWrite(Buzzer,HIGH);
      delay(250);
      digitalWrite(Buzzer,LOW);
      delay(100);
      digitalWrite(Buzzer,HIGH);
      delay(250);
      digitalWrite(Buzzer,LOW);
      delay(100);
      digitalWrite(Buzzer,HIGH);
      delay(1000);
      digitalWrite(Buzzer, LOW);
      break; //sai do loop for
    }
  }

  Serial.println("selecione o tempo do timer...");
}
}
```

Por maior que este código possa ser, ele não é difícil de ser entendido. No começo, como em todos os códigos, declaramos as variáveis e incluímos a biblioteca do display de LCD. No bloco de **setup** dizemos quais pinos são entradas e quais são saídas, além de escrever no monitor serial 3 frases. No **loop**, onde acontece toda a magia, fazemos a leitura dos 3 botões. Cada um tem sua função.

O primeiro apenas aumenta a quantidade de segundos de seu timer. O segundo aumenta a quantidade de minutos no timer, e o terceiro faz tudo acontecer.

Vamos dar uma olhada no que cada botão faz, analisando seus códigos:

Botão 1:

```
EstadoBotao1 = digitalRead(Botao1);
if (EstadoBotao1 == HIGH){
  delay(250);
  if (EstadoBotao1 == HIGH){
    segundo++;
    if(segundo>=60){
      segundo=0;
    }
    Serial.print("Segundos: ");
    Serial.println(segundo);
  }
}
```

No começo, usamos um delay entre duas leituras do botão. Fazemos isso para garantir que estamos apertando o botão e não é um ruído que está provocando a entrada de 5V na porta onde este botão está ligado. Não fizemos isso no projeto do piano, por exemplo, pois não precisávamos de precisão ao pressionar o botão. Quando garantimos que somos nós que estamos apertando o botão e não um ruído, incrementamos a variável **segundo** com o seguinte comando:

```
segundo++;
```

Este comando significa o mesmo que:

```
segundo = segundo + 1;
```

Ou seja, ele simplesmente soma 1 à variável chamada **segundo**. Feito isso, vamos para uma estrutura condicional **IF**, na qual verificamos se o valor da variável está maior ou igual a 60. Se ela for maior ou igual a 60, devemos zerá-la (pois podemos ter no máximo 59 segundos em nosso timer). Feito esta checagem, mostramos no monitor serial o valor da variável segundos.

Se dermos uma olhada na rotina do botão 2, podemos ver que ela é bastante parecida com a do botão 1, veja:

```
EstadoBotao2 = digitalRead(Botao2);
if (EstadoBotao2 == HIGH){
  delay(250);
  if (EstadoBotao2 == HIGH){
    minuto++;
    if(minuto>=60){
      minuto=0;
    }
    Serial.print("Minutos: ");
    Serial.println(minuto);
  }
}
```

A única diferença é que com este botão alteramos a quantidade de minutos em nosso timer. O restante é idêntico. Tudo acontece quando o botão 3 é pressionado, vamos dar uma olhada:

```
EstadoBotao3 = digitalRead(Botao3);
if (EstadoBotao3 == HIGH){
  if((minuto==0) && (segundo==0)){
    //nao faz nada
  }
}

//continua
```

```
//continuacao
else{
  Serial.println("START");
  delay(1000);

  if(segundo==0){
    minuto--;
    segundo=59;
  }

  for(int y = 0; y<segundo; y--){
    if(minuto<10){
      Serial.print("0");
      Serial.print(minuto);
      Serial.print(":");
    }
    else{
      Serial.print(minuto);
      Serial.print(":");
    }
    if(segundo<10){
      Serial.print("0");
      Serial.println(segundo);
    }
    else{
      Serial.println(segundo);
    }
  }

  segundo--;

  if(segundo<0){
    minuto--;
    segundo=59;
  }

  delay(1000); //base de tempo de 1 segundo
  //para deixar timer mais rápido, diminuir aqui

  if((segundo<=0) && (minuto<=0)) {
    Serial.println("ACABOU O TEMPO!");

    digitalWrite(Buzzer,HIGH);
    delay(250);
    digitalWrite(Buzzer,LOW);
    delay(100);
    digitalWrite(Buzzer,HIGH);
    delay(250);
    digitalWrite(Buzzer,LOW);
    delay(100);
    digitalWrite(Buzzer,HIGH);
    delay(250);
    digitalWrite(Buzzer,LOW);
    delay(100);
    digitalWrite(Buzzer,HIGH);
    delay(1000);
    digitalWrite(Buzzer, LOW);
    break; //sai do loop for
  }
  }
  Serial.println("Selecione o tempo do timer...");
}
}
```

A primeira coisa que fazemos quando pressionamos o botão 3 é uma rotina **IF...ELSE**. No **IF** testamos se existe algum número nas nossas variáveis minuto e segundo, ou se são iguais a zero. Se forem iguais a zero, nosso programa não deve fazer nada, pois se fizesse nosso timer faria o seguinte:

```
00:00
0 -59
```

0 -58
0- 57

e assim por diante. Colocando esta rotina, garantimos que somente vamos começar a rodar o timer quando temos algum valor diferente de zero na variável segundo OU minuto. Se tivermos qualquer número diferente de zero nas variáveis, entramos no **ELSE** desta rotina. Dentro do **else**, a primeira coisa que fazemos é escrever START no monitor serial. Então aguardamos 1 segundo para começar o timer. Agora começa nossa série de rotinas condicionais. Primeiro temos um **IF** verificando se a variável segundo é igual a 0. Se for, logo de cara tiramos uma unidade da variável minuto e colocamos que a variável segundo é igual a 59. Depois disso entramos em uma grande rotina **FOR**. Nesta rotina, nós inicializamos uma variável **y**, informamos que sua condição de existência vai até ela ser menor que a variável segundo e damos a informação que cada vez que acabar a rotina contida dentro do **for**, decrementamos **y** de uma unidade (**y--** é o mesmo que **y=y-1**). A primeira instrução deste for é uma condicional **IF** que verifica se a variável minuto é menor que 10. Se for, escrevemos no monitor serial o número zero (0) seguido do valor da variável minuto e depois um símbolo de dois pontos (:). Se a variável minuto não for menor que 10, escrevemos diretamente a variável minuto no monitor serial, seguida do dois pontos. Por que fazemos isso? Simples, pois se simplesmente imprimissemos no monitor serial a variável minuto sem formatação, ficaria um tanto quanto feio, ia ficar algo assim:

12:0 -> 11:0 -> 10:0 -> 9:0 -> 8:0 -> 7:0 -> 6:0 -> etc

Com esta formatação, temos o tempo visualmente da seguinte forma:

12:0 -> 11:0 -> 10:0 -> 09:0 -> 08:0 -> 07:0 -> 06:0 -> etc

De maneira similar, fazemos no próximo **IF** do programa o tratamento da variável segundo. Fazendo este tratamento, passamos a ler o seguinte na contagem de tempo:

12:00 -> 11:00 -> 10:00 -> 09:00 -> 08:00 -> 07:00 -> 06:00 -> etc

Que fica com mais cara de tempo, não é mesmo?

Feito estes dois **IFs**, damos a instrução que realmente importa no código:

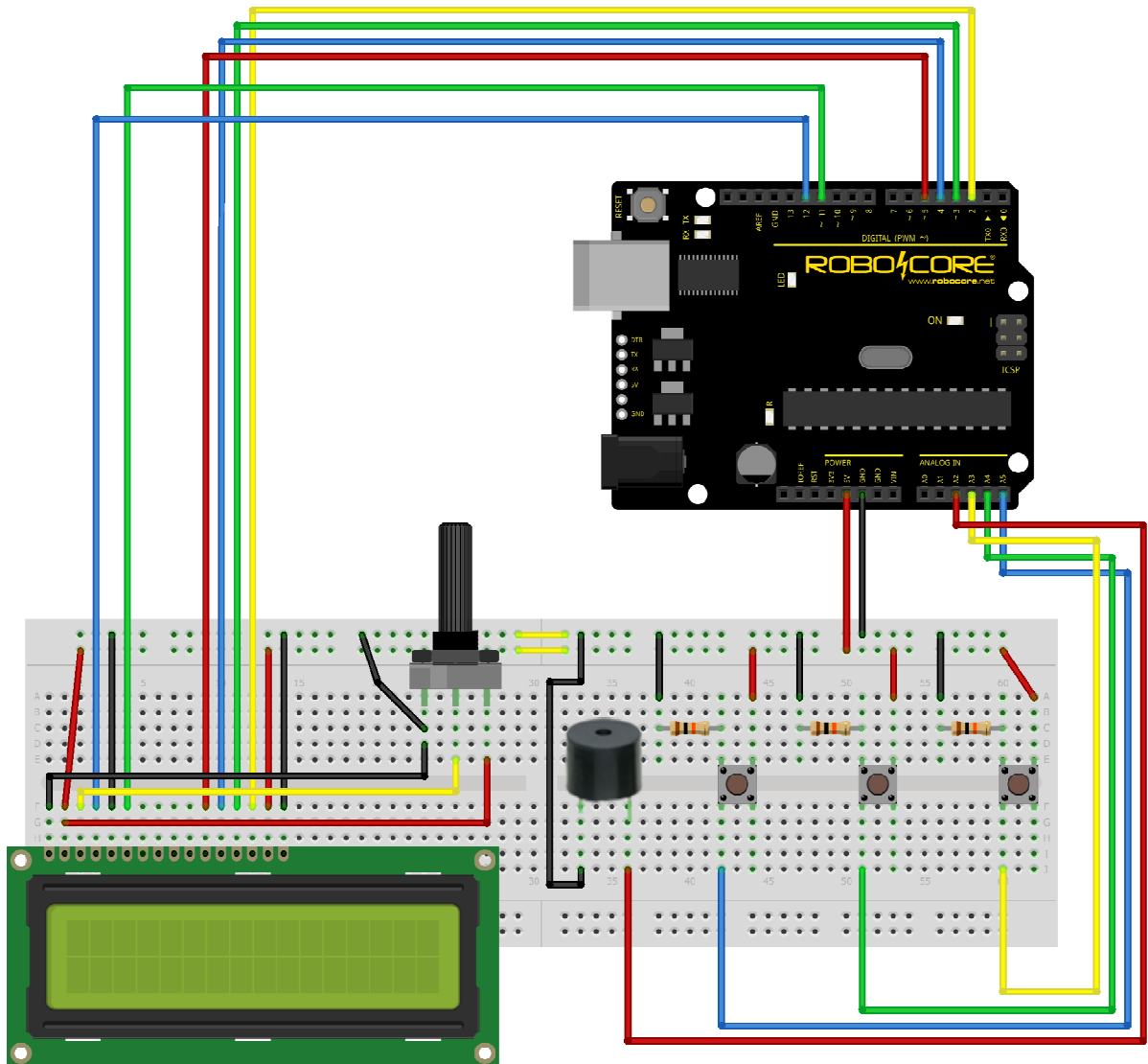
segundo-- ;

Ou seja, decrementamos o segundo em 1 unidade, e é isso que vai fazer nosso timer decrescer de 1 em 1 segundo (mas não se confunda, não é aqui que informamos quanto tempo irá demorar para dar este 1 segundo).

Após isso, verificamos se a variável segundo é menor que zero. Se for, devemos decrescer um da variável minuto e deixar a variável segundo igual a 59. Após isso fazemos um delay de 1 segundo. Esta instrução é bastante importante pois ela que dá a base de tempo para nosso timer. Se diminuirmos ou aumentarmos este valor, teremos um timer mais lento ou mais rápido, respectivamente.

A próxima instrução é verificarmos se o tempo do timer acabou. Se segundo e minuto forem iguais ou menores que zero, entramos na rotina **IF**, escrevemos no monitor serial que "ACABOU O TEMPO!", e emitimos uma série de apitos com nosso buzzer. A última instrução dentro deste **if** é o **break**. Esta instrução é interessante porque ela sai do loop **for** naquele exato momento. Então voltamos com a informação de selecionar o tempo do timer. Simples, não? Fica mais simples depois de ver tudo funcionando. Faça o upload do código para sua placa Arduino, e utilize o botão mais próximo do buzzer para adicionar segundos ao seu timer, o botão do meio para adicionar minutos e o último botão é o botão 3, que faz toda mágica acontecer. Pra brincar agora, apenas adicione cerca de 10 segundos e veja tudo funcionar no monitor serial. Se quiser colocar alguns minutos, fique a vontade, mas não demore a fazer a próxima parte deste experimento pois é ainda mais legal!

A segunda parte deste experimento é fazer o mesmo que fizemos até então, mas vendo tudo em um display de LCD. Como você já está craque em ligar o LCD no Arduino, nem irá precisar seguir o esquema, mas se precisar tome como base o circuito abaixo:



fritzing

Feita a montagem, grave o seguinte código em seu Arduino:

```

/*****\
**      ROBOCORE ARDUINO KIT INICIANTE      **
*                                           *
**      Projeto Timer no LCD                **
\*****/

#include <LiquidCrystal.h>

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

int segundo = 0;
int minuto = 0;
const int LED13 = 13;
const int Buzzer = A2;
const int Botao1 = A5;
const int Botao2 = A4;
const int Botao3 = A3;

//continua
    
```

```

//continuacao
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int EstadoBotao3 = 0;

void setup(){
  pinMode(Botao1, INPUT);
  pinMode(Botao2, INPUT);
  pinMode(Botao3, INPUT);
  pinMode(LED13,OUTPUT);
  pinMode(Buzzer,OUTPUT);
  lcd.begin(16, 2);
  inicializacao();
}

void loop(){

  EstadoBotao1 = digitalRead(Botao1);
  if (EstadoBotao1 == HIGH){
    delay(150);
    segundo++;
    if(segundo>=60){
      segundo=0;
      lcd.setCursor(11,0);
      lcd.print(" ");
    }

    lcd.setCursor(0,0);
    lcd.print("Segundos: ");
    lcd.print(segundo);
  }

  EstadoBotao2 = digitalRead(Botao2);
  if (EstadoBotao2 == HIGH){
    delay(150);
    if (EstadoBotao2 == HIGH){
      minuto++;
      if(minuto>=60){
        minuto=0;
        lcd.setCursor(10,1);
        lcd.print(" ");
      }
      lcd.setCursor(0,1);
      lcd.print("Minutos: ");
      lcd.print(minuto);
    }
  }

  EstadoBotao3 = digitalRead(Botao3);
  if (EstadoBotao3 == HIGH){
    if((minuto==0) && (segundo==0)){
      //nao faz nada
    }
    else{
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("      START      ");
      delay(1000);

      if(segundo==0){
        minuto--;
        segundo=59;
      }

      for(int y = 0; y<segundo; y--){
        if(minuto<10){
          lcd.setCursor(5,1);
          lcd.print("0");
          lcd.print(minuto);
          lcd.print(":");
        }
        else{
          lcd.setCursor(5,1);
          lcd.print(minuto);
          lcd.print(":");
        }
      }
    }
  }
  //continua

```

```

//continuacao
if(segundo<10){
    lcd.print("0");
    lcd.print(segundo);
}
else{
    lcd.print(segundo);
}

segundo--;

if(segundo<0){
    minuto--;
    segundo=59;
}

delay(1000); //base de tempo de 1 segundo
//para deixar timer mais rápido, diminuir aqui

if((segundo<=0) && (minuto<=0)) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("ACABOU O TEMPO!");

    digitalWrite(Buzzer,HIGH);
    delay(250);
    digitalWrite(Buzzer,LOW);
    delay(100);
    digitalWrite(Buzzer,HIGH);
    delay(250);
    digitalWrite(Buzzer,LOW);
    delay(100);
    digitalWrite(Buzzer,HIGH);
    delay(250);
    digitalWrite(Buzzer,LOW);
    delay(100);
    digitalWrite(Buzzer,HIGH);
    delay(1000);
    digitalWrite(Buzzer, LOW);
    break; //sai do loop for
}
}

inicializacao();
}
}

void inicializacao(){ //criacao do procedimento
    lcd.begin(16, 2);
    lcd.print("Selecione o ");
    lcd.setCursor(0,1);
    lcd.print("tempo do timer..");
    delay(1500);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Segundos: ");
    lcd.print(segundo);
    lcd.setCursor(0,1);
    lcd.print("Minutos: ");
    lcd.print(minuto);
}

```

A diferença deste código para o anterior, é que ao invés de imprimir no monitor serial, imprimimos as informações em um display de LCD. Também fazemos alguns pequenos ajustes como o setamento de posição do cursor e a limpeza do LCD para garantir que nosso código funciona da maneira esperada. Se você reparar, neste código criamos o chamado **procedimento**. Um procedimento é como se fosse uma função, mas não retorna uma variável para o local do código onde a chamamos. Criamos este procedimento para não ter que escrever duas vezes no código o mesmo set de informações. Isso é bastante útil para deixar o código enxuto.

▪ **Projeto Theremin Agudo**

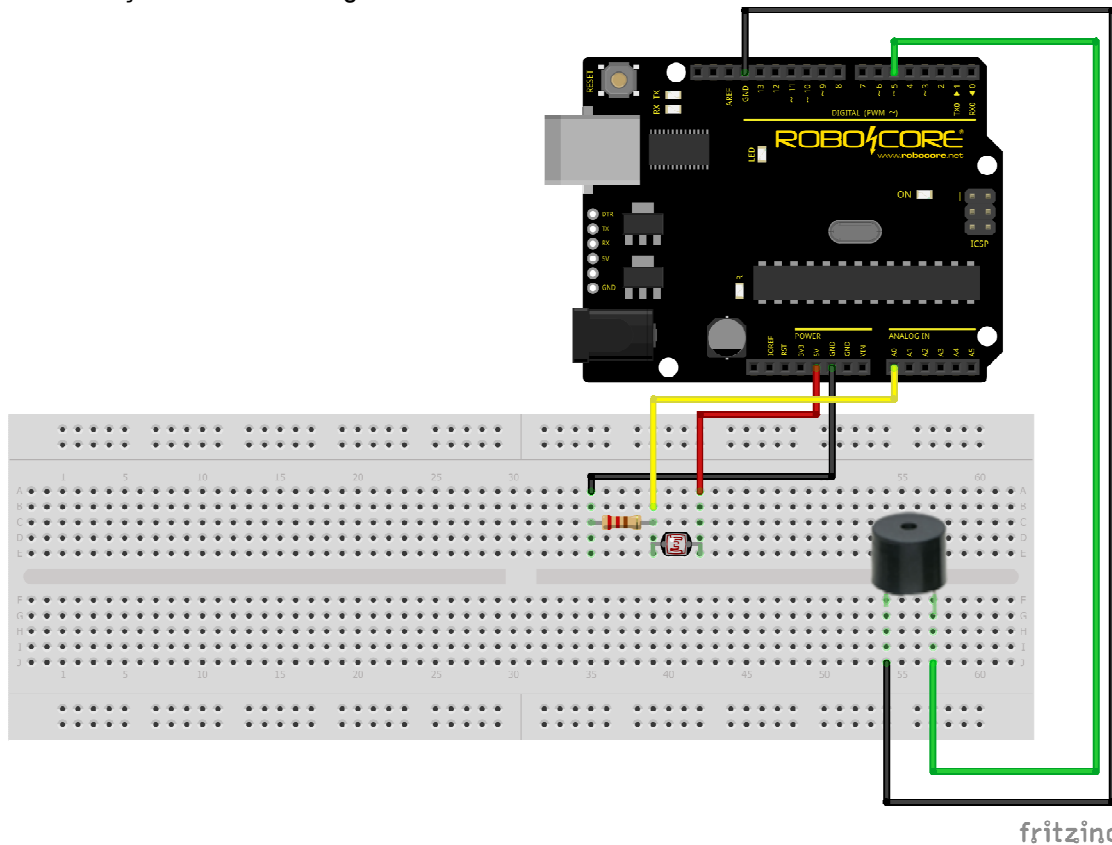
Componentes: 01 Buzzer + 01 Sensor de Luminosidade LDR + 01 Resistor 10k

Descrição: Aprenda como fazer outro instrumento musical, usando apenas três componentes.

Dificuldade: 

Muita gente não conhece este instrumento, trata-se de um dos primeiros instrumentos musicais completamente eletrônicos que não precisa do contato físico do músico para fazer soarem as notas. Claro que utilizando os componentes e a velocidade de processamento que temos em nossa placa, não teremos notas extremamente definidas, mas podemos fazer uma brincadeira.

Vamos começar montando o seguinte circuito:



Abaixo iremos propor um código bastante simples para este projeto:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE  **
*                                     *
**      Projeto Theremin Agudo      **
\*****/

int ValorSensor = 0;
int nota = 0;
const int Buzzer = 5;
void setup(){
}
void loop(){
  ValorSensor = analogRead(A0);
  nota = map(ValorSensor, 1023, 0, 1000, 6000);
  tone(Buzzer, nota, 20);
  delay(10);
}

```

Tape a luminosidade do LDR e destape com a mão vagarosamente para ouvir as diferentes frequências do buzzer. Este código é bastante básico, mas agora, com seus conhecimentos, você consegue calibrar o sensor e deixar o instrumento mais afinado. Este é um bom exercício treino.

CHEGOU A HORA DE TESTAR SEUS CONHECIMENTOS!

- **Exercício 1)** Utilizando a barra gráfica de LEDs, faça um código com poucas linhas, no qual cada um dos LEDs é aceso em uma ordem, e depois eles são apagados um a um, na ordem que foram acesos. (dica: use a rotina condicional **for**).
- **Exercício 2)** Utilizando o mesmo circuito do exercício anterior, acenda todos os LEDs da barra gráfica em uma ordem e depois apague do último que foi aceso para o primeiro (o contrário do exercício anterior).
- **Exercício 3)** Mostre a temperatura lida pelo LM35 no monitor serial na escala de Fahrenheit.
- **Exercício 4)** Mostre a temperatura lida pelo LM35 no display LCD 16x2, em uma das linhas mostre em graus Celsius e na debaixo mostre em Fahrenheit.
- **Exercício 5)** Utilizando o display de 7 segmentos, faça um dado de 6 números que mostra o valor quando você aperta um botão. (dica: pesquise no site do Arduino sobre a instrução *random*).
- **Exercício 6)** Utilizando a barra gráfica de LEDs, o potenciômetro e o LED de alto brilho, faça um programa que mostre na barra gráfica qual a intensidade do LED, cujo brilho é alterado pelo potenciômetro
- **Exercício Desafio:** Utilizando o display LCD e dois botões, faça um jogo no estilo Ping Pong (dica: coloque o display no centro da protoboard, um botão de um lado e o outro botão do outro lado, para ter espaço para os dois jogadores).

RESOLUÇÃO DOS EXERCÍCIOS

- **Exercício 1)** Utilizando a barra gráfica de LEDs, faça um código com poucas linhas, no qual cada um dos LEDs é aceso em uma ordem, e depois eles são apagados um a um, na ordem que foram acesos. (dica: use a rotina condicional **for**).

Ligue a barra gráfica de LEDs na protoboard e no Arduino, conforme o experimento do Termômetro. O código que propomos nesta solução é o seguinte:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE  **
*                                     *
**      Sugestão de solução ex 1      **
\*****/

const int LED[] = {
  2,3,4,5,6,7,8,9,10,11};

void setup(){
  for(int x = 0; x < 10; x++){
    pinMode(LED[x], OUTPUT);
  }
}

void loop(){
  for(int x = 0; x < 10; x++){
    digitalWrite(LED[x], HIGH);
    delay(50);
  }
  for(int x = 0; x <10 ; x++){
    digitalWrite(LED[x], LOW);
    delay(50);
  }
}

```

- **Exercício 2)** Utilizando o mesmo circuito do exercício anterior, acenda todos os LEDs da barra gráfica em uma ordem e depois apague do último que foi aceso para o primeiro (o contrário do exercício anterior).

Ligue a barra gráfica de LEDs na protoboard e no Arduino, conforme o experimento do Termômetro. O código que propomos nesta solução é o seguinte:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE  **
*                                     *
**      Sugestão de solução ex 2      **
\*****/

const int LED[] = {
  2,3,4,5,6,7,8,9,10,11};

void setup(){
  for(int x = 0; x < 10; x++){
    pinMode(LED[x], OUTPUT);
  }
}

void loop(){
  for(int x = 0; x < 10; x++){
    digitalWrite(LED[x], HIGH);
    delay(50);
  }
  for(int x = 9; x >= 0 ; x--){
    digitalWrite(LED[x], LOW);
    delay(50);
  }
}

```

Estes são bons exemplos de que usar rotinas **for** pode nos poupar muitas linhas de código!

- **Exercício 3)** Mostre a temperatura lida pelo LM35 no monitor serial na escala de Fahrenheit.

Monte na protoboard o circuito com LM35 e grave o seguinte código no Arduino e abra o monitor serial:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE  **
*                                     *
**      Sugestão de solução ex 3      **
\*****/

const int LM35 = 0;
int ADClido = 0;
float celsius = 0;
float fahrenheit = 0;

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

void loop(){
  ADClido = analogRead(LM35);
  celsius = ADClido * 0.1075268817204301;
  fahrenheit = (celsius * 1.8) + 32;
  Serial.print("Temperatura = ");
  Serial.print(celsius);
  Serial.print(" *C");
  Serial.print("\t"); //tab em linguagem C
  Serial.print("Temperatura = ");
  Serial.print(fahrenheit);
  Serial.println(" *F");
  delay(1000);
}

```

Aqui simplesmente fazemos a famosa conversão de graus Celsius para graus Fahrenheit (isso pode ser útil se você viajar para outros países, certo? Pena que este conversor ficaria um pouco grande para carregar...)

- **Exercício 4)** Mostre a temperatura lida pelo LM35 no display LCD 16x2, em uma das linhas mostre em graus Celsius e na debaixo mostre em Fahrenheit.

Ligue o LCD na protoboard conforme os diversos experimentos que já fizemos e grave o seguinte código no Arduino:

```

/*****\
**  ROBOCORE ARDUINO KIT INICIANTE  **
*                                     *
**      Sugestão de solução ex 4      **
\*****/

#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int LM35 = 0;
int ADClido = 0;
float celsius = 0;
float fahrenheit = 0;

void setup(){
  Serial.begin(9600);
  analogReference(INTERNAL);
  lcd.begin(16, 2);
}

void loop(){
  ADClido = analogRead(LM35);

//continua

```

```
//continuacao
celsius = ADClido * 0.1075268817204301;
fahrenheit = (celsius * 1.8) + 32;
lcd.setCursor(0,0);
lcd.print("Temp. = ");
lcd.print(celsius);
lcd.print(" *C");
lcd.setCursor(0,1);
lcd.print("Temp. = ");
lcd.print(fahrenheit);
lcd.print(" *F");
delay(1000);
}
```

Muito simples, certo? Agora que você já está craque em ler sensores e usar LCD com Arduino, nada mais é complicado para você.

- **Exercício 5)** Utilizando o display de 7 segmentos, faça um dado de 6 números que mostra o valor quando você aperta um botão. (dica: pesquise no site do Arduino sobre a instrução *random*).

A maneira mais fácil de começar a pensar neste código é aprendendo a usar a função *random()* do Arduino. Vamos começar escrevendo no monitor serial do Arduino números aleatórios de 1 a 6 (por enquanto não precisamos de componentes ligados no Arduino, apenas a placa):

```
int NumeroRandom;

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

void loop() {
  NumeroRandom = random(6);
  NumeroRandom = NumeroRandom + 1;
  Serial.println(NumeroRandom);
  delay(50);
}
```

No começo do programa declaramos uma variável chamada **NumeroRandom**. No **setup** só ativamos a comunicação serial e no **loop** usamos a função *random()*:

```
NumeroRandom = random(6);
NumeroRandom = NumeroRandom + 1;
Serial.println(NumeroRandom);
delay(50);
```

Primeiro dizemos que a variável **NumeroRandom** será um valor randômico de 0 a 5, ou seja, com 6 números. Na linha seguinte somamos 1. Fazemos isso pois um dado vai de 1 a 6 e não de 0 a 5, certo? Na próxima linha escrevemos o número no monitor serial e então usamos um *delay*. Agora temos que mostrar isso no display de 7 segmentos. Pra ficar mais rápido, vamos fazer sem usar o CI 4511, mas fique a vontade para fazer o experimento com o CI, é até bom pra você testar seus conhecimentos. Monte na protoboard o display de 7 segmentos conforme o primeiro experimento do display de 7 segmentos e coloque um na porta 2.

O código a ser usado pode ser o seguinte:

```
/*
*****\
**   ROBOCORE ARDUINO KIT INICIANTE   **
*                                     *
**           Sugestão de solução ex 5           **
\*****/

int NumeroRandom;
int Botao1 = 2;
int EstadoBotao1 = 0;
//continua
```

```
//continuacao

const int A = 12;
const int B = 11;
const int C = 10;
const int D = 9;
const int E = 8;
const int F = 6;
const int G = 7;

void setup(){
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
  pinMode(D, OUTPUT);
  pinMode(E, OUTPUT);
  pinMode(F, OUTPUT);
  pinMode(G, OUTPUT);
}

void loop() {
  EstadoBotao1 = digitalRead(Botao1);
  if (EstadoBotao1 == HIGH){
    delay(100);
    if (EstadoBotao1 == HIGH){
      Serial.println("entrei");
      NumeroRandom = random(6);
      NumeroRandom = NumeroRandom + 1;
      if(NumeroRandom == 1){
        acende1();
      }
      if(NumeroRandom == 2){
        acende2();
      }
      if(NumeroRandom == 3){
        acende3();
      }
      if(NumeroRandom == 4){
        acende4();
      }
      if(NumeroRandom == 5){
        acende5();
      }
      if(NumeroRandom == 6){
        acende6();
      }
    }
  }
}

void acende1(){
  digitalWrite(A, LOW); //acende os leds que representam o número 1
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, LOW);
  digitalWrite(E, LOW);
  digitalWrite(F, LOW);
  digitalWrite(G, LOW);
}

void acende2(){
  digitalWrite(A, HIGH); //acende os leds que representam o número 2
  digitalWrite(B, HIGH);
  digitalWrite(C, LOW);
  digitalWrite(D, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(F, LOW);
  digitalWrite(G, HIGH);
}

//continua
```

```
//continuacao

void acende3(){
  digitalWrite(A, HIGH); //acende os leds que representam o número 3
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(F, LOW);
  digitalWrite(G, HIGH);
}

void acende4(){
  digitalWrite(A, LOW); //acende os leds que representam o número 4
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(D, LOW);
  digitalWrite(E, LOW);
  digitalWrite(F, HIGH);
  digitalWrite(G, HIGH);
}

void acende5(){
  digitalWrite(A, HIGH); //acende os leds que representam o número 5
  digitalWrite(B, LOW);
  digitalWrite(C, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(F, HIGH);
  digitalWrite(G, HIGH);
}

void acende6(){
  digitalWrite(A, LOW); //acende os leds que representam o número 6
  digitalWrite(B, LOW);
  digitalWrite(C, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(G, HIGH);
}
```

- **Exercício 6)** Utilizando a barra gráfica de LEDs, o potenciômetro e o LED de alto brilho, faça um programa que mostre na barra gráfica qual a intensidade do LED, cujo brilho é alterado pelo potenciômetro

Comece montando o mesmo esquema do Projeto Dimmer. Depois monte na protoboard a barra de LEDs conforme o projeto termômetro, porém como o pino 11 já está sendo usado pelo LED de alto brilho, você deverá pular ele e usar os pinos seguintes. No caso, o LED de alto brilho irá usar o pino 11, e a barra gráfica irá usar os pinos 2, 3, 4, 5, 6, 7, 8, 9, 10 e 12.

Quanto ao código, podemos usar a seguinte sugestão:

```

/*****\
**   ROBOCORE ARDUINO KIT INICIANTE   **
*                                     *
**   sugestão de solução ex 6         **
\*****/

const int PinoPotenciometro = 0;
const int Led = 11;
int ValorPot = 0;
int pwm = 0;
const int LED[] = {
    2,3,4,5,6,7,8,9,10,12};

void setup() {
    pinMode(Led, OUTPUT);
    for(int x = 0; x < 10; x++){
        pinMode(LED[x], OUTPUT);
    }
}

void loop(){
    ValorPot = analogRead(PinoPotenciometro);
    pwm = map(ValorPot, 0, 1023, 0, 255);
    analogWrite(Led, pwm);

    if(pwm > 5){
        digitalWrite(LED[0], HIGH);
    }
    else{
        digitalWrite(LED[0], LOW);
    }

    if(pwm > 50){
        digitalWrite(LED[1], HIGH);
    }
    else{
        digitalWrite(LED[1], LOW);
    }

    if(pwm > 75){
        digitalWrite(LED[2], HIGH);
    }
    else{
        digitalWrite(LED[2], LOW);
    }

    if(pwm > 100){
        digitalWrite(LED[3], HIGH);
    }
    else{
        digitalWrite(LED[3], LOW);
    }

    if(pwm > 125){
        digitalWrite(LED[4], HIGH);
    }
    else{
        digitalWrite(LED[4], LOW);
    }
}
//continua
    
```

```
//continuacao
if(pwm > 150){
  digitalWrite(LED[5], HIGH);
}
else{
  digitalWrite(LED[5], LOW);
}
if(pwm > 175){
  digitalWrite(LED[6], HIGH);
}
else{
  digitalWrite(LED[6], LOW);
}

if(pwm > 200){
  digitalWrite(LED[7], HIGH);
}
else{
  digitalWrite(LED[7], LOW);
}

if(pwm > 225){
  digitalWrite(LED[8], HIGH);
}
else{
  digitalWrite(LED[8], LOW);
}
if(pwm > 250){
  digitalWrite(LED[9], HIGH);
}
else{
  digitalWrite(LED[9], LOW);
}
}
```

▪ **Exercício Desafio)** Utilizando o display LCD e dois botões, faça um jogo no estilo Ping Pong (dica: coloque o display no centro da protoboard, um botão de um lado e o outro botão do outro lado, para ter espaço para os dois jogadores).

Chame um amigo e vamos jogar! A ideia deste exercício é que você quebre a cabeça durante alguns dias (ou algumas horas) para fazer um jogo usando o que você aprendeu no manual e também conceitos de programação que podem ser aprendidos pelo site de referência www.arduino.cc/en/Reference. O código sugerido abaixo possui um botão no pino 2, outro no pino 3 e o LCD usando os pinos 4, 5, 6, 7, 11 e 12. Sugerimos colocar o LCD no centro da protoboard (com o potenciômetro de ajuste de brilho em cima dele), um botão em uma extremidade e outro botão na outra extremidade. A ideia do código proposto é: quando a bolinha de ping pong chegar no último pixel do canto direito do LCD, é preciso apertar o botão da direita, e quando ela chegar no último pixel do canto esquerdo, deve-se apertar o botão da esquerda.

O código proposto é:

```
/*
*****\
**   ROBOCORE ARDUINO KIT INICIANTE   **
*                                       *
**   Sugestão de solução ex Desafio   **
**                                       **
**                                       **
*****/

#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);

const int Botao1 = 6;
const int Botao2 = 7;
int EstadoBotao1 = 0;
int EstadoBotao2 = 0;
int velocidade = 1000; //velocidade inicial
int sentido = 0; //se for 0 a bola vai estar indo
//se for 1 a bola vai estar vindo
int i, x, k = 0;
//continua
```

```

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

  pinMode(Botao1, INPUT);
  pinMode(Botao2, INPUT);
  attachInterrupt(0, rebateuDireita, RISING); //interrupcao no pino 2
  attachInterrupt(1, rebateuEsquerda, RISING); //interrupcao no pino 3
  lcd.begin(16, 2);
  lcd.print(" Ping Pong 1.0 ");
  delay(2000);
  lcd.clear();
}

void loop() {
  jogo();
}

void jogo(){
  while(1){
    if(sentido == 0){ //da direita pra esquerda
      for(i=15; i>=0; i--){
        switch(k){
          case 0:
            lcd.setCursor(i,0);
            lcd.print("o");
            delay(velocidade);
            lcd.clear();
            k=1;
            break;

          case 1:
            lcd.setCursor(i,1);
            lcd.print("o");
            delay(velocidade);
            lcd.clear();
            k=0;
            break;
        }
      }
    }

    if(sentido == 1){ //da esquerda pra direita
      for(i=0; i<16; i++){
        switch(k){
          case 0:
            lcd.setCursor(i,0);
            lcd.print("o");
            delay(velocidade);
            lcd.clear();
            k=1;
            break;

          case 1:
            lcd.setCursor(i,1);
            lcd.print("o");
            delay(velocidade);
            lcd.clear();
            k=0;
            break;
        }
      }
    }
  }
}

//continua

```



```

void rebateuEsquerda(){
  if(i == 0){ //bola no canto esquerdo
    sentido = 1; //muda o sentido
    velocidade=velocidade-50; //aumenta a velocidade
    if(velocidade <= 50){ //garante a velocidade minima de 50
      velocidade = 50;
    }
  }
  else{
    fimDeJogo(0); //vai pra função fim de jogo levando 0
    //isso mostra que ESQUERDA perdeu
  }
}

void rebateuDireita(){
  if(i == 15){ //bola no canto direito
    sentido = 0; //muda o sentido
    velocidade=velocidade-50; //aumenta a velocidade
    if(velocidade <= 50){ //garante a velocidade minima de 50
      velocidade = 50;
    }
  }
  else{
    fimDeJogo(1); //vai pra função fim de jogo levando 1
    //isso mostra que DIREITA perdeu
  }
}

int fimDeJogo(int x){
  if(x == 0){
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Esquerda Perdeu!");
    lcd.setCursor(0,1);
    lcd.print("RESETE o Arduino");
    while(1);
  }

  if(x == 1){
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Direita Perdeu!");
    lcd.setCursor(0,1);
    lcd.print("RESETE o Arduino");
    while(1);
  }
}
    
```

Este código é um pouco complexo de início, ele usa conceitos de interrupção, estrutura SWITCH CASE, criação de funções que passam parâmetros, procedimentos, etc, conceitos abordados no Arduino Kit Avançado da RoboCore. De toda forma, entrando no site de referência informado acima, você tem uma ótima noção de como usar estes conceitos e muitos outros. Vale muito a pena dar uma olhada. O código proposto é o básico para um jogo de Ping Pong digital, porém ele poderia ter diversas melhorias, entre elas:

- Não precisar apertar reset cada vez que quiser recomeçar a jogar, após alguém perder;
- Ao rebater a bola, ela continuar da mesma posição ao invés de "subir" um espaço;
- Ao iniciar o jogo, a bola não voltar pro começo do LCD se ninguém apertar o botão na hora certa;
- e por aí vai..

Você está convidado a pensar em como melhorar este e todos os outros códigos. Este é também um ótimo exercício pra aprender lógica de programação. Para mais informações sobre tecnologia, eletrônica e robótica acesse www.RoboCore.net e fique por dentro de tudo que acontece no mundo tecnológico. Use também o fórum do site para discutir novos projetos e também falar sobre as experiências propostas aqui.

MINI GLOSSÁRIO

Abaixo você encontrará algumas estruturas usadas na programação do Arduino:

<p>Estrutura básica de código:</p> <pre>void setup(){ //setup do código } void loop(){ //loop do código }</pre>	<p>Estrutura condicional if:</p> <pre>if(variável == valor){ //faça alguma coisa se //valor da variável for //igual ao valor testado } else{ // senão, faça outra //coisa }</pre>	<p>Estrutura for:</p> <pre>for(int x = 0; x < 10; x++){ //contagem de 0 a 9 //seu código vai se repetir //10 vezes dentro desse loop //nessas condições }</pre>
<p>Estrutura switch case:</p> <pre>switch (variável){ case 1: //faça algo se //variável =1 break; case 2: //faça algo se //variável =2 break; default: //se variável não for //nem 1 nem 2, faça o //que estiver no default }</pre>	<p>Estrutura while:</p> <pre>while(variável < 10){ //faça algo durante 10 //vezes //pois incrementamos de 1 //em // 1 na linha abaixo variavel = variavel + 1; }</pre>	<p>Estrutura do-while:</p> <pre>do{ //faça algo enquanto.... } while(variavel < valor);</pre>
<p>Operadores de comparação:</p> <pre>== (igual a) != (diferente de) < (menor que) > (maior que) <= (menor ou igual a) >= (maior ou igual a) && (operador lógico AND) (operador lógico OR) ! (operador lógico NOT)</pre>	<p>Configuração de pinagem:</p> <pre>pinMode(pino, INPUT); //seta o pino como ENTRADA pinMode(pino, OUTPUT); //seta o pino como SAÍDA pinMode(pino, INPUT_PULLUP); //seta como pull-up o pino de entrada</pre>	<p>Pino Digital - Leitura e Escrita:</p> <pre>digitalWrite(pino, HIGH); //seta o pino como nível logico alto digitalWrite(pino, LOW); //seta o pino como nível logico baixo digitalRead(pino); //retorna se o pino está HIGH ou LOW</pre>
<p>Pino Analógico - Leitura e Escrita:</p> <pre>analogRead(pino); //faz a leitura de um pino de entrada analógica analogWrite(pino); //envia uma tensão analógica para um pino de saída usando PWM</pre>	<p>Comunicação Serial:</p> <pre>Serial.begin(9600); //ajusta o baudrate da comunicação Serial.print("olá"); //imprime na porta serial a palavra olá</pre>	<p>Conheça muitas outras funções e estruturas de código acessando:</p> <p>www.Arduino.cc</p>

Esperamos que esta apostila tenha-lhe sido válida e que você seja mais um *Arduinizador do Mundo*.