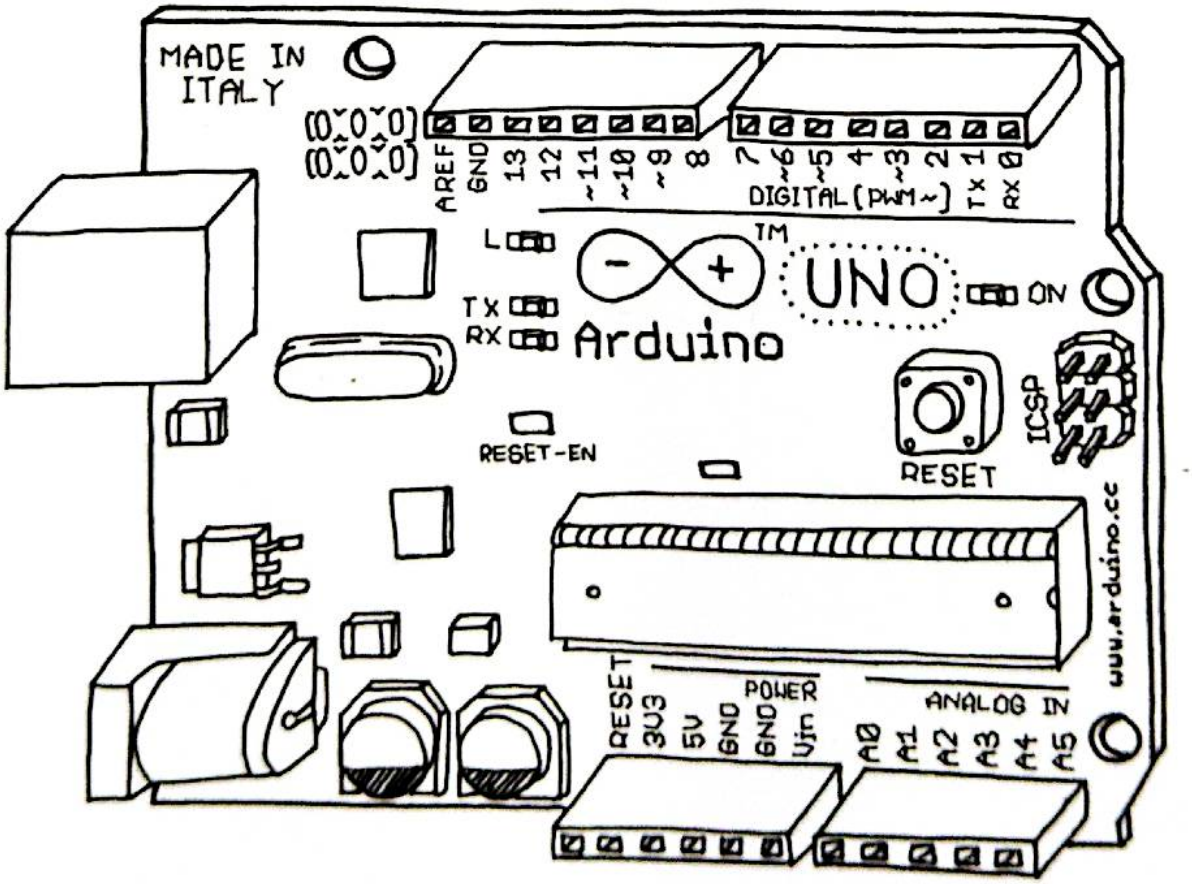
Primeiros Passos com o Arduino

Massimo Banzi cofundador do Projeto Arduino



O'REILLY
novatec

Digitalizado com CamScanner

Primeiros Passos com o Arduino

Digitalizado com CamScanner

Authorized Portuguese translation of the English edition of *Getting Started with Arduino*, Second Edition ISBN 9781449309879 © 2011 Massimo Banzi. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Tradução em português autorizada da edição em inglês da obra *Getting Started with Arduino*, Second Edition ISBN 9781449309879 © 2011, Massimo Banzi.
Esta tradução é publicada e vendida com a permissão da O'Reilly Media, Inc., detentora de todos os direitos para publicação e venda desta obra.

© Novatec Editora Ltda. 2012.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998.
É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito, do autor e da Editora.

Editor: Rubens Prates
Tradução: Rafael Zanolli
Revisão gramatical: Débora Facin
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-290-4

Histórico de impressões:

Dezembro/2011 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

Digitalizado com CamScanner

Sumário

Mensagem importante para nossos leitores 9

Prefácio 11

Capítulo 1 ▪ Introdução 17

Público-alvo 18
O que é Computação Física? 19

Capítulo 2 ▪ O jeito Arduino 21

Experimentação 22
Patching 23

Capítulo 3 ▪ A plataforma Arduino 33

Hardware do Arduino 34
Software (IDE) 36
Instalação do Arduino em seu computador 37
Instalação de drivers: Macintosh 37
Instalação de drivers: Windows 38
Identificação de porta: Macintosh 39
Identificação de porta: Windows 39

Capítulo 4 ▪ Dando seus primeiros passos com o Arduino 41

Anatomia de um dispositivo interativo 41
Sensores e atuadores 42
LED pisca-pisca 42
Passe-me o parmesão 47
O Arduino não permite o encerramento de programas 48
Verdadeiros experimentadores escrevem comentários 48

Digitalizado com CamScanner

O código-fonte, passo a passo 48
O que construiremos 52
O que é eletricidade? 52
Uso de um botão de pressão para controlar o LED 56
Como isso funciona? 60
Um circuito, mil comportamentos 60

Capítulo 5 ▪ Implementações avançadas de entradas e saídas 67

Experiências com sensores de ligado/desligado 67
Controle de luz com PWM 70
Uso de um sensor de luz, em vez de um botão de pressão 77
Entrada analógica 78
Experimente outros sensores analógicos 81
Comunicação serial 82
Projetos mais complexos (motores, lâmpadas e coisas do tipo) 83
Sensores complexos 85

Capítulo 6 ▪ Comunicação em nuvem 87

Planejamento 89
Código-fonte 90
Montagem do circuito 99
Como montar 100

Capítulo 7 ▪ Resolução de problemas 103

Teste da placa 104
Teste do circuito da protoboard 105
Isolamento de problemas 107
Problemas com o IDE 107
Como obter ajuda online 108

Apêndice A ▪ Protoboard 111

Apêndice B ▪ Leitura de resistores e capacitores 113

Apêndice C ▪ Referência rápida do Arduino 117

Estrutura 117
Símbolos especiais 117

Digitalizado com CamScanner

Constantes 119
Variáveis 119
Estruturas de controle 121
Aritmética e fórmulas 125
Operadores de comparação 126
Funções de entrada e saída 127
Funções de tempo 129
Funções matemáticas 129
Funções de números aleatórios 132
Comunicação serial 132

Apêndice D ▪ Leitura de diagramas esquemáticos 135

Índice remissivo 139

Digitalizado com CamScanner

Mensagem importante para nossos leitores

Garantir sua segurança é sua própria responsabilidade, inclusive no que se refere ao uso de equipamentos de segurança e na verificação da competência e habilidade necessárias para a realização dos projetos. A eletricidade e outros recursos utilizados nesses projetos são perigosos, a menos que utilizados corretamente e com as devidas precauções, incluindo o uso dos equipamentos de segurança corretos. Algumas ilustrações deixam de retratar as precauções e os equipamentos recomendados, deixando mais espaço para as imagens do projeto. Esses projetos não devem ser realizados por crianças. O uso das instruções e sugestões deste livro é de sua própria responsabilidade. A Novatec Editora e o autor renunciam a toda responsabilidade por qualquer dano, ferimento ou custo adicional resultante. É sua responsabilidade garantir que seus projetos obedeçam às leis aplicáveis, inclusive as de direitos autorais.

Digitalizado com CamScanner

Prefácio

Há alguns anos recebi um desafio muito interessante: ensinar a designers um mínimo de eletrônica para que pudessem construir protótipos interativos dos objetos que projetavam.

De início, segui um instinto subconsciente e ensinei eletrônica a tais profissionais da mesma forma que esse tópico me havia sido ensinado na escola. Posteriormente percebi que minha escolha não estava produzindo os resultados pretendidos e comecei a me lembrar da sensação de estar sentado na classe, totalmente entediado, tendo de ouvir toda aquela teoria despejada sobre mim, sem qualquer aplicação prática para o que era apresentado.

Na realidade, quando estava na escola já sabia eletrônica de forma muito empírica: conhecimento que adquirira como resultado de pouca teoria, mas bastante experiência prática.

Comecei a pensar em como havia sido o processo por meio do qual havia de fato aprendido eletrônica:

» Primeiro, aprendi gradualmente o que eram os mais variados componentes.

» Depois, comecei a experimentar com tais elementos, alterando algumas de suas conexões internas e vendo o que acontecia ao dispositivo: geralmente o resultado era algo entre uma explosão ou uma nuvem de fumaça.

» Passei então a construir alguns kits vendidos por revistas de eletrônica.

Digitalizado com CamScanner

» Combinava dispositivos que havia alterado, reformulando kits e outros circuitos encontrados em revistas, para produzir novos resultados.

Quando criança, sempre me interessara em descobrir como as coisas funcionavam; portanto, costumava desmontar muitos aparelhos. Essa paixão cresceu à medida que tomava objetos não utilizados da casa e os desmontava até encontrar seus menores componentes. Eventualmente, as pessoas passaram a trazer todo tipo de objetos para que eu os "dissecasse". Meus maiores projetos nessa época eram uma máquina lava-louças e um computador primitivo que viera de uma seguradora, acompanhado de uma enorme impressora, cartões eletrônicos, leitores de cartões magnéticos e vários outros componentes muito interessantes e difíceis de serem desmontados.

Depois de muitas experiências como essas, sabia o que os componentes eletrônicos eram e tinha uma noção aproximada de suas funções. Além disso, minha casa era repleta de revistas antigas de eletrônica que meu pai devia ter comprado no início dos anos 1970. Passava horas lendo os artigos e analisando os diagramas de circuito, mesmo sem compreendê-los por inteiro.

Esse processo de leitura repetida dos artigos, acompanhado do conhecimento que adquirira desmontando os equipamentos, criou um gradual círculo virtuoso.

Um grande avanço nesse sentido ocorreu certa vez, em um Natal, quando meu pai me presenteou com um kit cujo objetivo era ensinar eletrônica a adolescentes. Todos os componentes vinham embalados em cubos plásticos que se prendiam uns aos outros magneticamente, estabelecendo uma conexão; o símbolo eletrônico vinha marcado em sua parte superior. Mal sabia que o brinquedo era também um marco do design alemão, uma vez que Dieter Rams[1] o havia projetado na década de 1960.

1 N.T.: Dieter Rams é um dos mais influentes designers do século XX. Muitos de seus produtos tornaram-se peças fixas de museus, incluindo o MoMa, em Nova York (fonte: Wikipédia).

Digitalizado com CamScanner

Munido dessa nova ferramenta, pude rapidamente montar os circuitos e experimentá-los para ver o que acontecia. O ciclo de prototipagem tornava-se cada vez mais curto.

Depois disso, construí rádios, amplificadores, circuitos capazes de produzir barulhos horríveis e sons agradáveis, sensores de chuva e pequeninos robôs.

Passei muito tempo em busca de uma palavra do idioma inglês capaz de resumir esse tipo de atividade, sem plano específico; partimos de uma ideia e acabamos por atingir um resultado completamente inesperado. Finalmente, a palavra *tinkering*[2] me veio à mente. Reconheço que essa palavra tem sido utilizada em muitos outros campos para descrever uma forma de operação e retratar pessoas que se dedicam ao caminho da exploração. Por exemplo, os diretores franceses que deram início à "Nouvelle Vague" eram chamados de *tinkerers*[3]. A melhor definição de *tinkering* que já encontrei vem de uma exposição apresentada no Exploratorium de São Francisco (*www.exploratorium.edu/tinkering*):

> *Tinkering é o que ocorre quando você tenta realizar algo que ainda não domina, guiado por sua criatividade, imaginação e curiosidade. Nessa prática, não há instruções – nem fracassos; não há forma certa ou errada de proceder. Trata-se de descobrir como tudo funciona e reformular esses elementos.*
>
> *Instrumentos, máquinas e objetos aparentemente incompatíveis funcionando em harmonia – esse é o espírito do tinkering.*
>
> *Essa atividade é, em sua essência, um processo que combina diversão e investigação.*

---

2 N.T.: O verbo "to tinker" pode ser traduzido como brincar, mexer, experimentar, remendar, improvisar. Neste livro, o autor o utiliza para representar uma atividade que combina vários desses sentidos, indicando um comportamento exploratório e inventivo em que a aprendizagem é obtida por meio de experimentos com componentes, de modo empírico e envolvendo elementos essenciais de diversão e experimentação.

3 N.T.: Nesse sentido, o tinkerer é a pessoa que pauta sua atividade pela experimentação, pela inventividade, pela exploração de componentes e elementos preexistentes, aprendendo empiricamente enquanto desenvolve seu conhecimento.

Digitalizado com CamScanner

A partir de meus primeiros experimentos pude deduzir quanta prática seria necessária para que alguém inexperiente pudesse criar um circuito capaz de fazer o pretendido, partindo dos componentes mais básicos.

Outro avanço ocorreu no verão de 1982, quando viajei para Londres com meus pais, e passei muitas horas visitando o Museu de Ciências. Eles haviam acabado de inaugurar uma nova ala dedicada aos computadores; seguindo uma série de experimentos, aprendi o básico sobre matemática binária e programação.

Lá, percebi que em muitas aplicações os engenheiros não estavam mais construindo circuitos a partir de componentes básicos, mas antes implementando muito da inteligência em seus produtos por meio de microprocessadores. Softwares estavam substituindo muitas horas gastas com projetos eletrônicos e permitiriam um ciclo de experimentação mais curto.

Quando retornei, passei a economizar dinheiro, uma vez que queria comprar um computador e aprender programação.

Depois disso, meu primeiro e mais importante projeto veio da utilização de um computador ZX81 novinho para controlar uma máquina de solda. Sei que não parece um projeto muito empolgante, mas necessitava desse recurso e alcançar esse objetivo era um grande desafio para mim, uma vez que acabara de aprender a programar. Nessa época, tornou-se evidente que escrever linhas de código tomaria menos tempo do que modificar circuitos complexos.

Cerca de vinte anos depois, acredito que essa experiência me permite ensinar pessoas que não se recordam de suas aulas de matemática e também estimulá-las com a mesma energia e o mesmo interesse que tive em minha juventude e que mantenho desde então.

– Massimo

Digitalizado com CamScanner

Agradecimentos

Este livro é dedicado à Luisa e à Alexandra.

Primeiramente, gostaria de agradecer a meus parceiros na equipe do Arduino: David Cuartielles, David Mellis, Gianluca Martino e Tom Igoe. É incrível trabalhar com vocês!

Agradeço à Barbara Ghella. Ela não sabe, mas, sem seus preciosos conselhos, o Arduino e este livro talvez nunca tivessem se tornado realidade.

A Bill Verplank, por ter-me ensinado mais do que Computação Física.

À Gillian Crampton-Smith, por oferecer-me uma chance e por tudo que aprendi com ela.

A Hernando Barragan, por seu trabalho no Wiring.

A Brian Jepson, por ser um ótimo editor e nos apoiar em todo o projeto.

À Nancy Kotary, Brian Scott, Terry Bronson e Patti Schiendelman, por transformarem aquilo que escrevi em um livro concluído.

Também quero agradecer a muitas outras pessoas, mas Brian disse-me que estou ficando sem espaço, por isso apenas listarei um pequeno grupo de pessoas às quais tenho muito a agradecer pelos mais variados motivos:

Adam Somlai-Fisher, Ailadi Cortelletti, Alberto Pezzotti, Alessandro Germinasi, Alessandro Masserdotti, Andrea Piccolo, Anna Capellini, Casey Reas, Chris Anderson, Claudio Moderini, Clementina Coppini, Concetta Capecchi, Csaba Waldhauser, Dario Buzzini, Dario Molinari, Dario Parravicini, Donata Piccolo, Edoardo Brambilla, Elisa Canducci, Fabio Violante, Fabio Zanola, Fabrizio Pignoloni, Flavio Mauri, Francesca Mocellin, Francesco Monico, Giorgio Olivero, Giovanna Gardi, Giovanni Battistini, Heather Martin, Jennifer Bove, Laura Dellamotta, Lorenzo Parravicini, Luca Rocco, Marco Baioni, Marco Eynard, Maria

Digitalizado com CamScanner

Teresa Longoni, Massimiliano Bolondi, Matteo Rivolta, Matthias Richter, Maurizio Pirola, Michael Thorpe, Natalia Jordan, Ombretta Banzi, Oreste Banzi, Oscar Zoggia, Pietro Dore, Prof Salvioni, Raffaella Ferrara, Renzo Giusti, Sandi Athanas, Sara Carpentieri, Sigrid Wiederhecker, Stefano Mirti, Ubi De Feo, Veronika Bucko.

Como entrar em contato conosco

Verificamos ao máximo as informações deste livro, mas talvez você encontre elementos que passaram por mudanças (ou mesmo erros que tenhamos cometido!). Como leitor deste livro, você pode nos ajudar a melhorar edições futuras enviando-nos seu feedback. Por favor, informe-nos da presença de erros, imprecisões, afirmações confusas ou equivocadas e erros de digitação que possam vir a ser encontrados.

Por favor, mostre-nos o que podemos fazer para tornar este livro mais útil a você. Levaremos muito a sério seus comentários e tentaremos incorporar sugestões razoáveis em edições futuras.

Dirija seus comentários e questões sobre este livro à editora, escrevendo para: *novatec@novatec.com.br*.

Temos uma página na web para este livro, onde incluímos a lista de erratas, exemplos e qualquer outra informação adicional.

- Página da edição em português
  - *http://www.novatec.com.br/catalogo/7522290_arduino*
- Página da edição original em inglês
  - *www.makezine.com/getstartedarduino*

Para mais informações sobre o Arduino, incluindo fóruns de discussão e mais documentação, visite *www.arduino.cc*.

Para obter mais informações sobre livros da Novatec, acesse nosso site: *http://www.novatec.com.br*.

Digitalizado com CamScanner

CAPÍTULO 1

Introdução

O Arduino é uma plataforma de computação física de fonte aberta, com base em uma placa simples de entrada/saída (input/output, ou I/O), assim como em um ambiente de desenvolvimento que implementa a linguagem Processing (*www.processing.org*). O Arduino pode ser utilizado para desenvolver objetos interativos independentes, ou conectado a softwares de seu computador (como Flash, Processing, VVVV, ou Max/MSP). As placas podem ser montadas manualmente, ou compradas pré-montadas; você pode fazer o download gratuito do Integrated Development Environment (IDE) de código aberto em *www.arduino.cc*.

O Arduino é diferente de outras plataformas presentes no mercado em razão dos seguintes fatores:

» Trata-se de um ambiente multiplataforma; ele pode ser executado no Windows, Macintosh e Linux.

» Tem por base o IDE de programação Processing, ambiente de desenvolvimento fácil de ser utilizado e que costuma ser empregado por artistas e designers.

» Pode ser programado utilizando-se um cabo USB, sem necessidade de uma porta serial. Esse recurso é útil, uma vez que muitos computadores modernos não têm portas seriais.

» É um hardware e software de fonte aberta – se você quiser, pode fazer o download do diagrama de circuito, comprar

Digitalizado com CamScanner

todos os componentes e criar seu próprio Arduino, sem ter de pagar nada aos criadores originais.

» O hardware é barato. A placa USB custa cerca de €20 (atualmente, algo em torno de U$35) e substituir um chip queimado é muito fácil, além de não custar mais do que €5 ou US$4. Justamente por isso, seus eventuais erros não acarretarão grandes prejuízos.

» Há uma comunidade ativa de usuários, com muitas pessoas que podem ajudá-lo.

» O Arduino Project foi desenvolvido em um ambiente educacional; portanto, é ideal para iniciantes que desejam resultados rápidos.

Este livro foi projetado para auxiliar iniciantes a compreender os benefícios que podem obter ao aprender a utilizar a plataforma Arduino e ao adotar sua filosofia.

Público-alvo

Este livro foi escrito tendo em mente os usuários "originais" do Arduino: designers e artistas. Assim, o modo como o texto procura explicar tópicos poderá enlouquecer certos engenheiros. Na verdade, um deles chamou os capítulos introdutórios do meu primeiro rascunho de "tolice". É exatamente isso que queremos dizer. Vamos ser honestos: a maioria dos engenheiros não é capaz de explicar o que faz nem mesmo para outro engenheiro; imagine se tiverem de fazê-lo para um ser humano normal. Agora, vamos mergulhar fundo nessa tolice.

NOTA: O Arduino complementa o trabalho de tese feito por Hernando Barragan na plataforma Wiring, enquanto ele estudava comigo e com Casey Reas no IDII Ivrea.

Depois que o Arduino começou a se popularizar, notei quantos *experimentadores*, "hobistas" e hackers de todo tipo passavam a *utilizá-lo na criação* de projetos maravilhosos e malucos. Percebi

Digitalizado com CamScanner

que todas essas pessoas também são artistas e designers em suas atividades, por isso este livro também é para elas.

O Arduino foi criado para ensinar Design de Interação, uma disciplina de design que coloca a prototipagem no centro de sua metodologia. Há muitas definições para Design de Interação, mas a que prefiro é esta:

> *Design de Interação é o projeto de qualquer experiência interativa.*

No mundo atual, o Design de Interação preocupa-se com a criação de experiências significativas entre nós (humanos) e objetos. É uma ótima forma de explorarmos a criação de experiências belas – e até mesmo controversas – entre nós e a tecnologia. O Design de Interação encoraja projetos que utilizem um processo iterativo, com base em protótipos de fidelidade crescente. Essa abordagem – parte também de alguns tipos de projetos "convencionais" – pode ser estendida para incluir a prototipagem aplicada à tecnologia; em especial, a prototipagem de eletrônicos.

O campo específico do Design de Interação envolvido no Arduino é a Computação Física (ou Design de Interação Física).

O que é Computação Física?

A Computação Física utiliza elementos de eletrônica na prototipagem de novos materiais para designers e artistas.

Ela envolve o projeto de objetos interativos que podem se comunicar com humanos utilizando sensores e atuadores controlados por um comportamento implementado como software, executado dentro de um microcontrolador (um pequeno computador ou chip individual).

No passado, o uso de elementos de eletrônica significava que teríamos de lidar com engenheiros o tempo todo e criar circuitos desenvolvendo um componente de cada vez; esses problemas faziam com que pessoas criativas deixassem de se envolver

Digitalizado com CamScanner

diretamente com o meio. A maioria das ferramentas era destinada a engenheiros e exigia vasto conhecimento técnico. Nos anos recentes, microcontroladores tornaram-se mais baratos e fáceis de serem utilizados, permitindo a criação de ferramentas melhores.

O progresso que fizemos com o Arduino significa que aproximamos essas ferramentas do iniciante, permitindo que essas pessoas construam seus projetos depois de apenas dois ou três dias de trabalho.

Com o Arduino, um designer ou artista pode rapidamente aprender o básico de eletrônica e sensores e começar a criar seus protótipos com pouco investimento.

Digitalizado com CamScanner

CAPÍTULO 2

O jeito Arduino

A filosofia do Arduino concentra-se em desenvolver projetos, e não em falar sobre eles. Ela representa uma busca constante por meios mais rápidos e poderosos de criarmos protótipos. Exploramos muitas técnicas de prototipagem e desenvolvemos formas de pensar cada vez mais práticas.

A engenharia clássica depende de um processo estrito pelo qual avançamos de A para B; por outro lado "o jeito Arduino" (*Arduino Way*), sua filosofia e prática, oferece a possibilidade de encontrarmos alternativas no caminho e quem sabe terminarmos em C.

Este é o processo de experimentação criativa (*tinkering*) de que tanto gostamos – a possibilidade de brincarmos com os elementos de modo ilimitado e de descobrirmos o inesperado. Nessa busca por formas pelas quais podemos criar melhores protótipos, selecionamos alguns pacotes de software que permitem um processo de manipulação constante dos elementos de software e hardware envolvidos.

As próximas seções apresentarão algumas filosofias, eventos e pioneiros que serviram de inspiração ao jeito do Arduino.

Digitalizado com CamScanner

Prototipagem

A prototipagem ocupa papel central quando falamos do jeito do Arduino: criamos e construímos objetos que interagem com outros objetos, pessoas e redes. Buscamos sempre encontrar uma forma de produzir protótipos do modo mais simples e mais rápido, sempre com o menor custo possível.

Muitos iniciantes, ao ter seu contato inicial com eletrônica, pensam equivocadamente que terão de construir tudo do zero. Isso representa um desperdício de energia: o que você realmente deseja é a capacidade de confirmar muito rapidamente a funcionalidade de algo, para que se sinta motivado a dar o próximo passo, ou mesmo para motivar outra pessoa a financiar seu projeto.

É por isso que desenvolvemos a "prototipagem oportunista": por que desperdiçar tempo e energia construindo algo do zero, em um processo que requer tempo e conhecimentos técnicos avançados, quando podemos utilizar dispositivos já prontos e adaptá-los de modo a explorar todo o trabalho prévio realizado por grandes empresas e engenheiros qualificados?

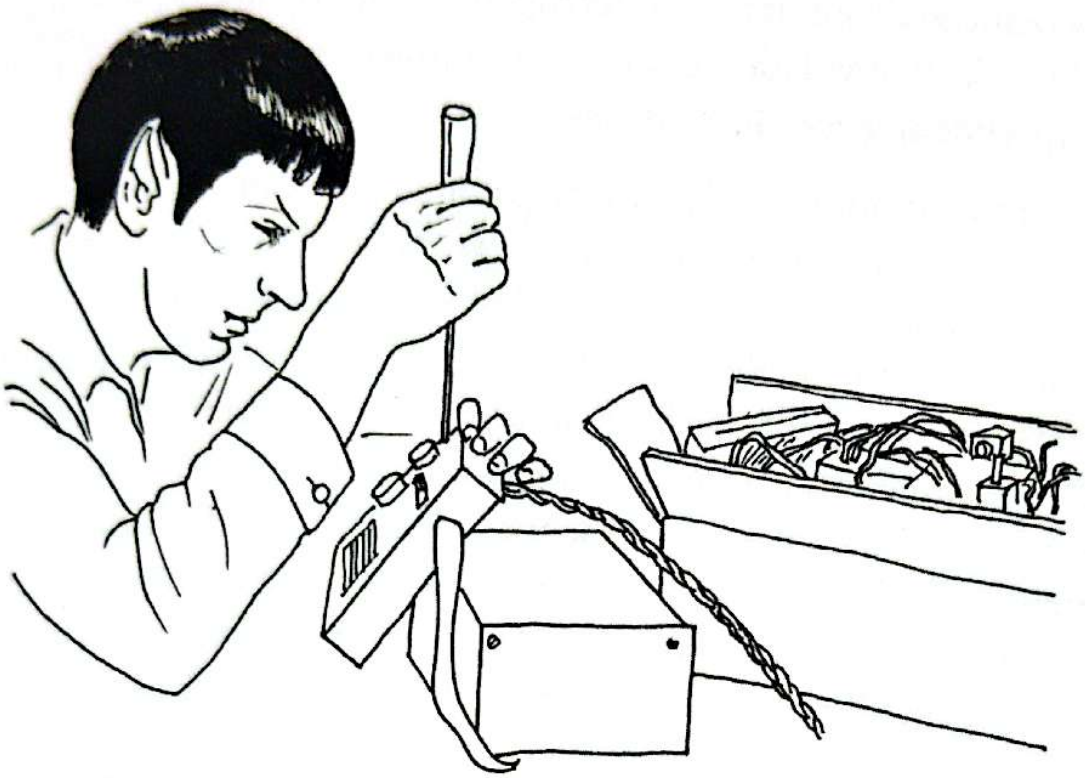
Nosso herói é James Dyson, que criou 5.127 protótipos para seu aspirador de pó antes de estar satisfeito com o resultado (*www.international. dyson.com/jd/1947.asp*).

Experimentação

Acreditamos ser essencial a oportunidade de experimentar com a tecnologia, explorando diferentes possibilidades diretamente em componentes de hardware e software – às vezes, sem um objetivo muito definido.

A reutilização de tecnologias existentes é uma das melhores formas de praticarmos essa experimentação. Alterar brinquedos baratos e equipamentos descartados, modificando-os para que realizem novas tarefas, é uma das melhores formas de obtermos ótimos resultados.

Digitalizado com CamScanner



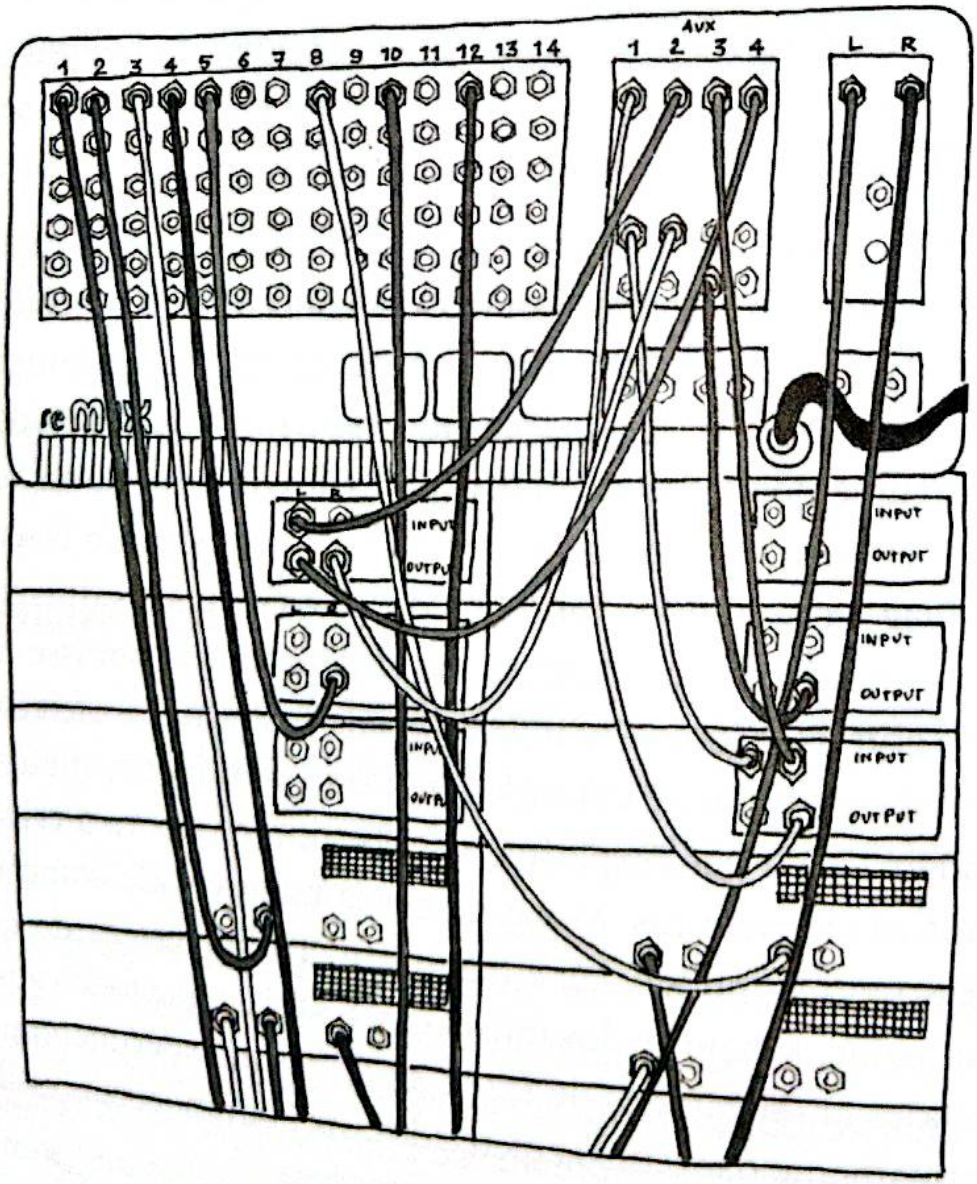
Patching

Sempre fui fascinado pela ideia de modularidade e pela capacidade de construir sistemas complexos conectando dispositivos simples. Esse processo foi muito bem representado por músicos como Robert Moog e seus sintetizadores analógicos. Munidos de tais instrumentos, músicos eram capazes de construir novos sons, experimentando combinações infinitas, simplesmente fazendo o *patching* (conectando), módulos distintos com seus diferentes cabos. O resultado fazia com que o sintetizador lembrasse uma antiga central telefônica, mas, combinada aos recursos oferecidos por inúmeros botões e controladores, essa abordagem era a plataforma perfeita para experimentações sonoras e para a criação de músicas inovadoras. Moog descrevia tal atividade como um processo entre "presenciar e descobrir". Tenho certeza de que, inicialmente, a maioria dos músicos não sabia o que todos aqueles botões eram capazes de fazer; ainda assim experimentaram continuamente os recursos oferecidos, aprimorando seus estilos individuais sem interrupção nesse processo.

Digitalizado com CamScanner

A redução do número de interrupções é muito importante para o fluxo da criatividade – quanto mais ininterrupto o processo, mais experimentações são possíveis.

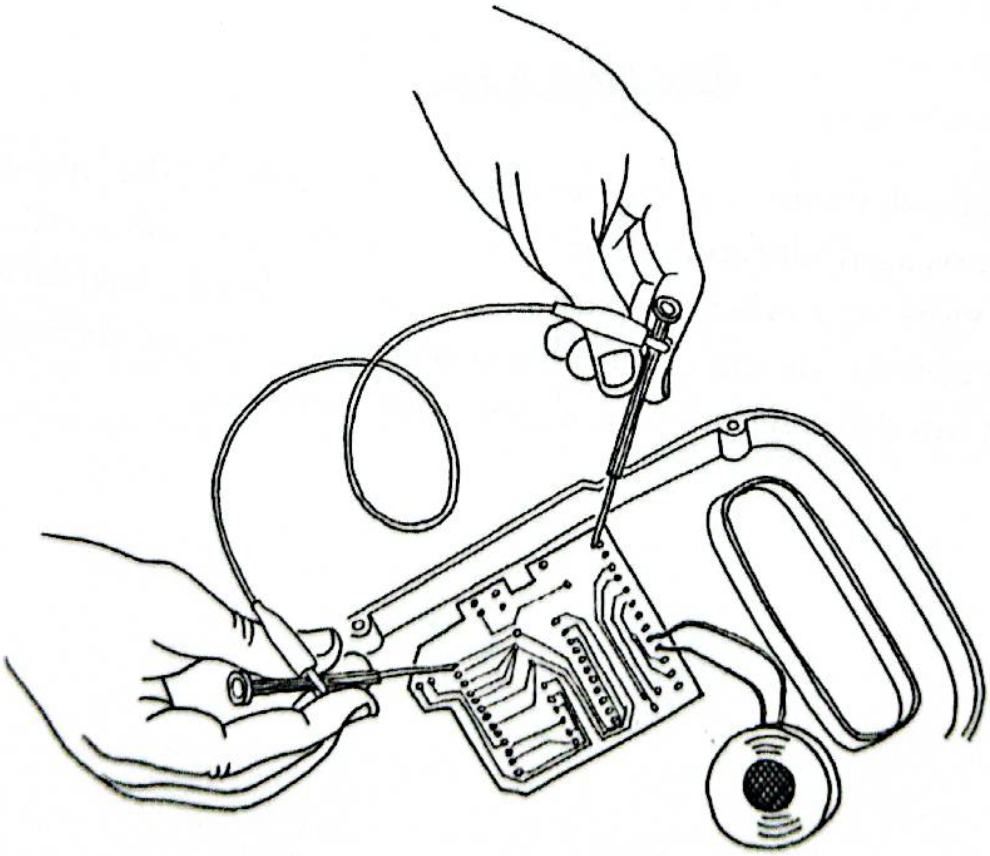
Essa técnica foi apresentada ao mundo do software por ambientes de "programação visual", como Max, Pure Data, ou VVVV. Tais ferramentas podem ser visualizadas como "caixas" que contêm as diferentes funcionalidades oferecidas, permitindo que o usuário construa "conexões" que ligam uma caixa a outra. Ambientes como esse permitem que o usuário experimente programar sem estar sujeito às constantes interrupções típicas do ciclo de desenvolvimento usual: "digite o programa, compile, droga – há um erro, corrija o erro, compile, execute". Se você é mais propenso a uma abordagem visual, recomendo que experimente esses recursos.



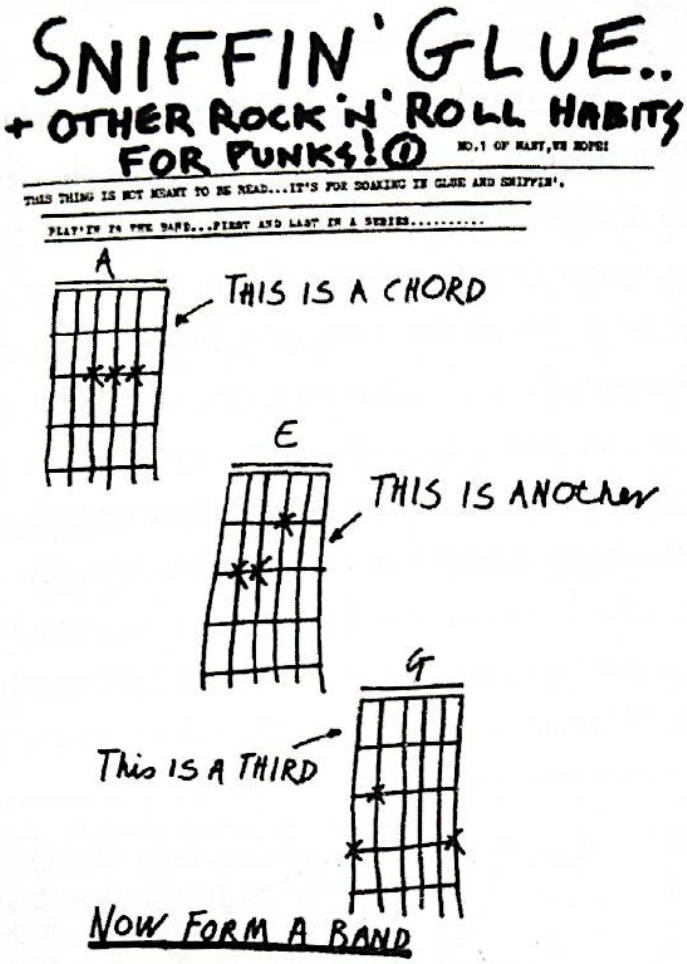
Digitalizado com CamScanner

Circuit bending

O *circuit bending* é uma das formas mais interessantes de experimentação. Refere-se à reformulação criativa de dispositivos de áudio de baixa tensão, como pedais de guitarras, brinquedos para crianças e pequenos sintetizadores, de modo a criar novos instrumentos musicais e geradores sonoros. Essencial a esse projeto é a "arte da oportunidade". Tudo começou em 1966, quando Reed Ghazala, ao acaso, provocou um curto-circuito ao conectar um amplificador de brinquedo a um objeto de metal da gaveta de sua mesa, produzindo uma sequência de efeitos sonoros inusitados. O que mais me agrada em *circuit benders* é sua capacidade de criar os mais incríveis dispositivos simplesmente experimentando com tecnologia, sem, necessariamente, entender o que está ocorrendo do ponto de vista teórico.



Digitalizado com CamScanner



É mais ou menos o que defendia o fanzine *Sniffin' Glue*[1] que mostramos aqui: durante a época do punk, conhecer três acordes de um violão era suficiente para se montar uma banda. Não deixe os especialistas de um campo lhe convencerem de que você nunca será um deles. Ignore-os e, depois, surpreenda-os.

1 N.T.: Sniffin' Glue é o nome de um famoso fanzine mensal criado por Mark Perry em julho de 1976. O nome surgiu da música "Now I Wanna Sniff Some Glue", dos Ramones, e a publicação chegou a ser chamada por alguns de "Bíblia do movimento punk" (fonte: Wikipédia).

Digitalizado com CamScanner

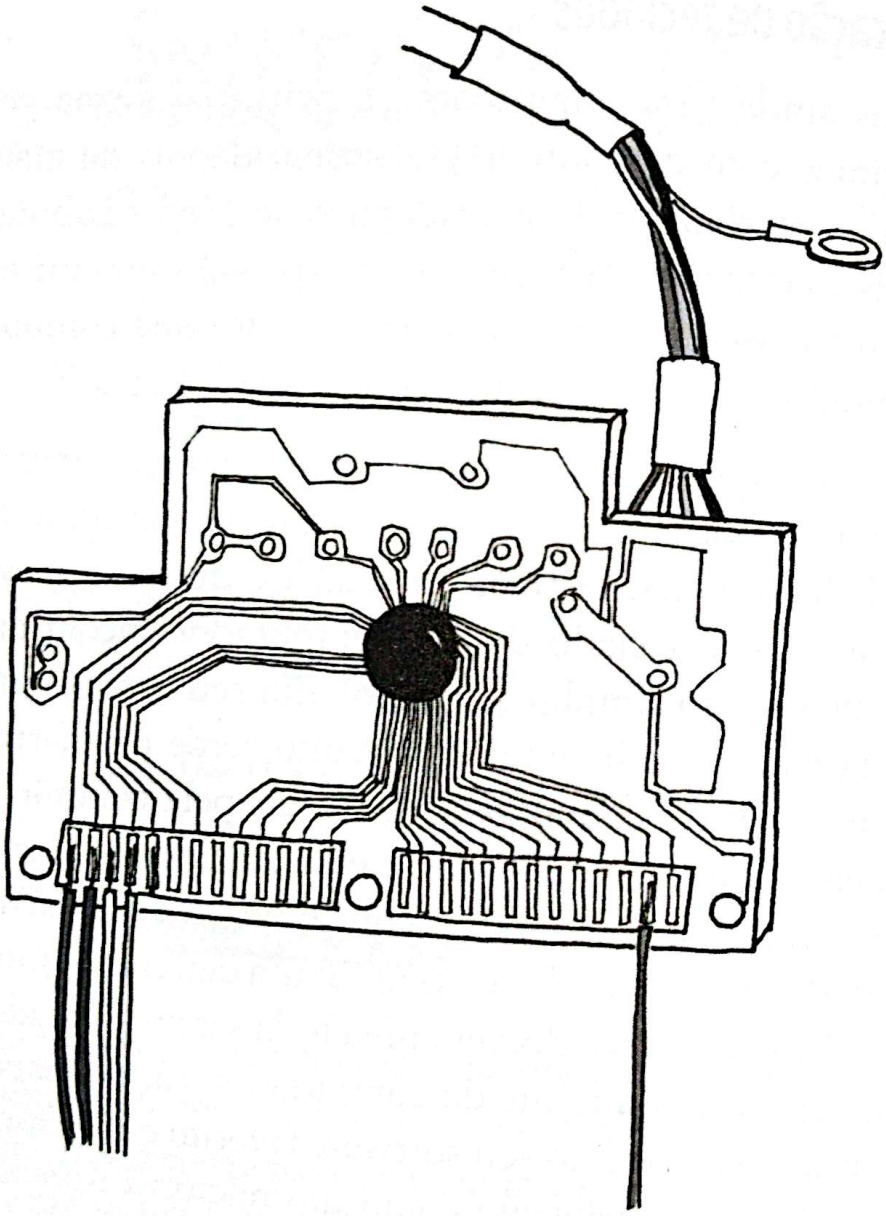
Modificação de teclados

Teclados ainda funcionam como a principal forma pela qual interagimos com computadores, mesmo depois de mais de 60 anos. Alex Pentland, diretor acadêmico do Media Laboratory do MIT, disse certa vez: "Perdoem-me a expressão, mas mictórios de banheiros masculinos são mais inteligentes que computadores. Computadores estão isolados daquilo que os cerca."[2]

Em nossos experimentos, podemos implementar novas formas de interação por software simplesmente substituindo as teclas do teclado por dispositivos capazes de detectar movimento. Ao desmontar o teclado de um computador encontramos um dispositivo muito simples (e básico). Em seu centro, temos uma pequena placa, geralmente um circuito verde ou marrom, com dois conjuntos de contatos responsáveis pela conexão entre as diferentes teclas. Se você remover esse circuito e utilizar um fio para conectar dois contatos, verá que uma letra surgirá na tela do computador. Se, depois disso, comprar um detector de movimento e conectá-lo a esse teclado, verá uma tecla ser pressionada sempre que alguém passar à frente do computador. Faça o mapeamento dessa funcionalidade ao seu software favorito e você tornará seu computador tão inteligente como um mictório. Aprender mais sobre as modificações (*hacks*) que podemos fazer com teclados é essencial quando falamos em prototipagem e Computação Física.

2 Citado em Sara Reese Hedberg, "MIT Media Lab's quest for perceptive computers." Intelligent Systems and Their Applications, IEEE, jul./ago. de 1998.

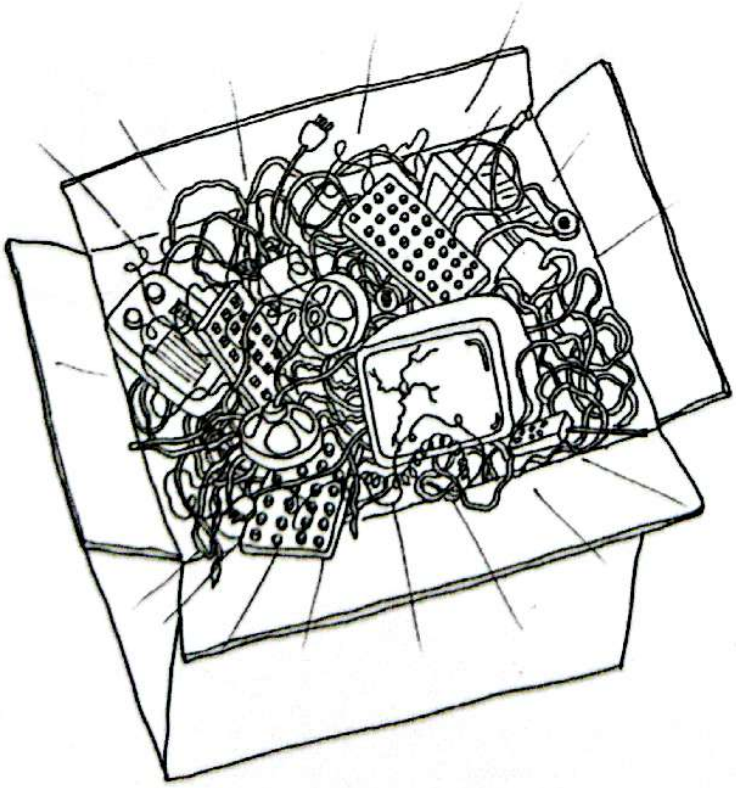
Digitalizado com CamScanner



Digitalizado com CamScanner

Adoramos tranqueiras!

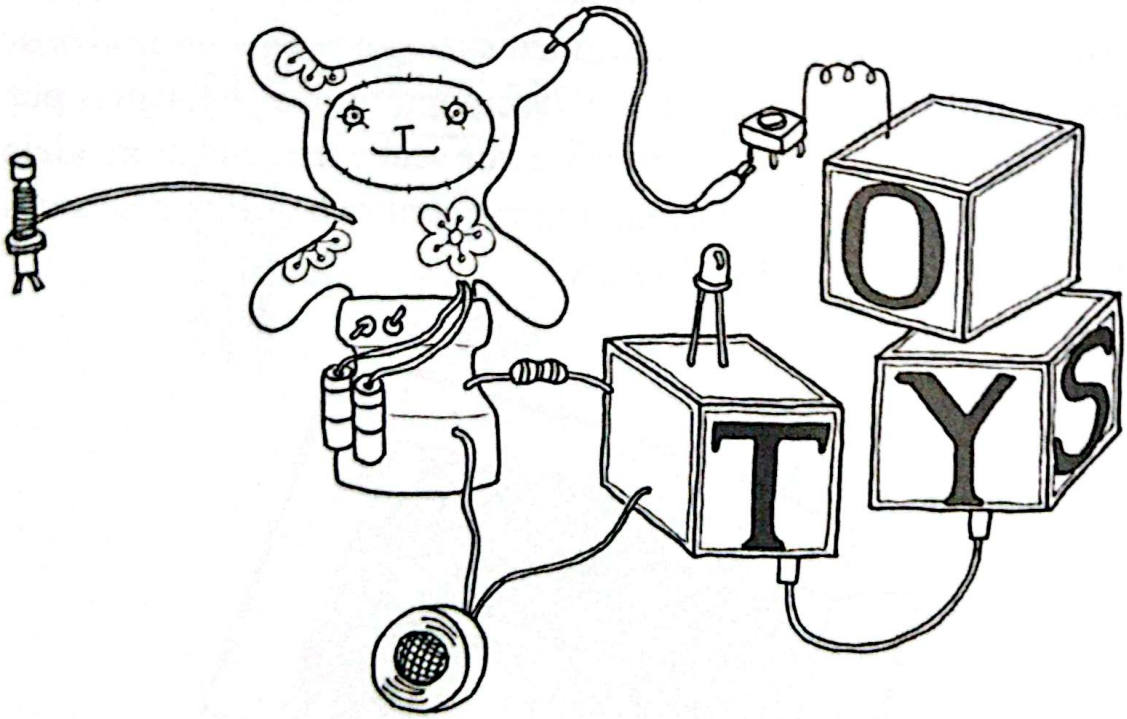
Nos dias atuais as pessoas jogam fora muita tecnologia: impressoras velhas, computadores, aparelhos de escritório, equipamentos técnicos e mesmo artigos militares. Sempre houve um grande mercado para tal excedente de tecnologia, especialmente entre hackers mais jovens e com poucos recursos, e aqueles que estão dando seus primeiros passos. Tal mercado se tornou evidente na cidade de Ivrea, onde desenvolvemos o Arduino. A cidade costumava ser a sede da empresa Olivetti, a qual desenvolvia computadores desde os anos 1960; em meados da década de 1990, a empresa jogava seu "lixo" em ferros-velhos da região, deixando-os repletos de peças de computadores, componentes eletrônicos e estranhos dispositivos de todos os tipos. Costumávamos passar horas e horas nesses locais, construindo os mais variados aparelhos e dispositivos, sem gastar praticamente nada e desenvolvendo nossos protótipos. Quando você pode comprar mil alto-falantes por um valor irrisório, é bem provável que acabe tendo alguma ideia criativa. Acumule seu lixo eletrônico e confira o que há nele antes de começar a criar algo do zero.



Digitalizado com CamScanner

Modificação de brinquedos

Brinquedos são uma fonte fantástica de tecnologia acessível para modificações e reutilização, como mostra a prática do *circuit bending* de que já falamos. Com a atual chegada de milhares de brinquedos baratos e de alta tecnologia vindos da China, você pode rapidamente construir suas ideias, com apenas alguns gatinhos barulhentos e algumas espadas de luz. Faço isso há alguns anos para que meus alunos compreendam que a tecnologia não é algo que deve assustá-los, ou um assunto difícil de ser abordado. Uma de minhas fontes favoritas é o livreto "Low Tech Sensors and Actuators", de Usman Haque e Adam Somlai-Fischer (*lowtech.propositions.org.uk*). Acredito que, nesse texto, os autores descreveram perfeitamente essa técnica; desde então, utilizo esse material.



Digitalizado com CamScanner

Colaboração

A colaboração entre usuários é um dos princípios essenciais do mundo do Arduino – por meio do fórum, em *www.arduino.cc*, pessoas de diferentes partes do mundo podem ajudar umas às outras a aprender mais sobre a plataforma. A equipe do Arduino encoraja usuários a colaborar localmente, assim como os auxilia na criação de grupos de usuários em todas as cidades que visitam. Também criamos um Wiki, o "Playground" (*www.arduino.cc/playground*), no qual os usuários podem registrar suas descobertas. É realmente incrível ver quanto conhecimento essas pessoas disponibilizam na Web, permitindo o acesso de todos. Essa cultura de compartilhamento e auxílio mútuo é um dos elementos que mais me orgulham no que se refere ao Arduino.

Digitalizado com CamScanner

CAPÍTULO 3

A plataforma Arduino

O Arduino é formado por dois componentes principais: a placa Arduino, elemento de hardware com o qual você trabalha ao construir seus objetos; o IDE do Arduino, o software que você executa em seu computador. O IDE pode ser utilizado para criar um sketch (esboço, um pequeno programa de computador), do qual você fará o upload para a placa Arduino. O sketch dirá à sua placa o que deve ser feito.

Há não muito tempo, trabalhar com hardware significava criar circuitos do zero, utilizando centenas de componentes diferentes, com nomes estranhos, como resistor, capacitor, indutor, transistor e assim por diante.

Todo circuito era projetado para uma aplicação específica e realizar alterações demandava que você cortasse fios, soldasse conexões e mais.

Com o surgimento de tecnologias e microprocessadores digitais, essas funções, antes implementadas por fios, foram substituídas por programas de software.

Software é mais fácil de ser modificado do que hardware. Com apenas alguns comandos, você pode alterar radicalmente a lógica de um dispositivo, experimentando duas ou três versões no mesmo tempo que seria necessário para soldar alguns resistores.

Digitalizado com CamScanner

Hardware do Arduino

A placa Arduino é uma pequena placa microcontroladora, ou seja, um pequeno circuito (a placa) que contém um computador inteiro dentro de um pequenino chip (o microcontrolador). Esse computador é ao menos mil vezes mais poderoso do que o MacBook que estou utilizando para escrever este texto, mas é muito mais barato e útil para a criação de dispositivos interessantes. Dê uma olhada na placa Arduino: você verá um chip de cor preta, com 28 "pernas" – esse chip é o ATmega328, o elemento essencial de sua placa.

Nós (a equipe do Arduino) colocamos nessa placa todos os componentes necessários para que esse microcontrolador funcione corretamente e possa se comunicar com seu computador. Há muitas versões para essa placa; utilizaremos neste livro o Arduino Uno, versão de uso mais simples e a melhor para aqueles que ainda estão aprendendo. Entretanto, as instruções que forneceremos também podem ser aplicadas a outras versões da placa, incluindo o Arduino Duemilanove, de 2009. A figura 3.1 mostra o Arduino Uno, enquanto a figura 3.2 mostra o Arduino Duemilanove.

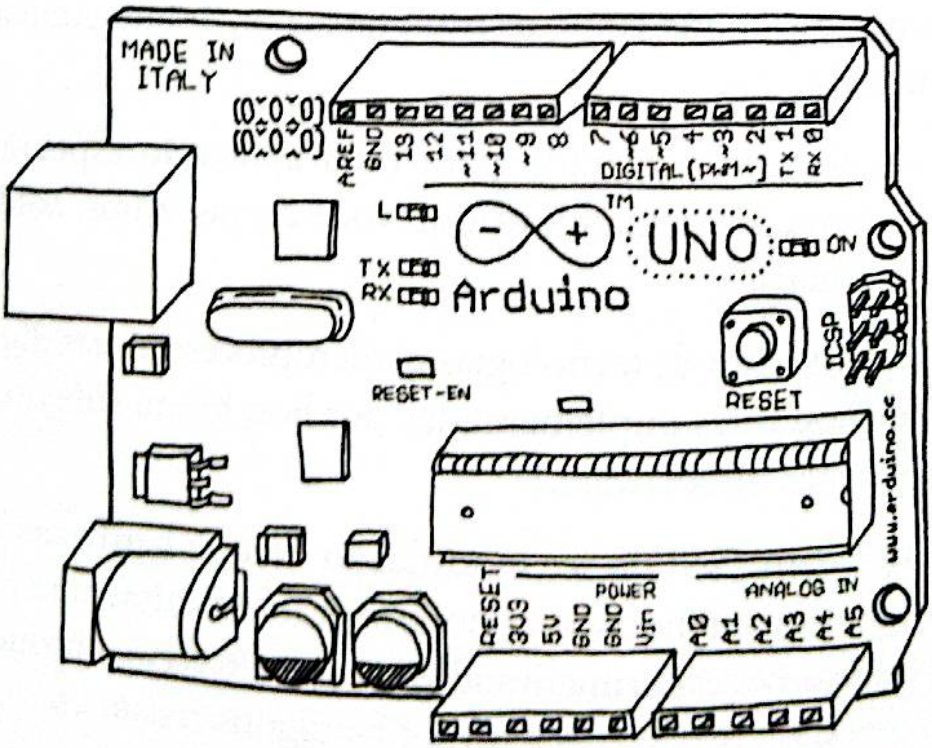


Figura 3.1 – Arduino Uno.

Digitalizado com CamScanner

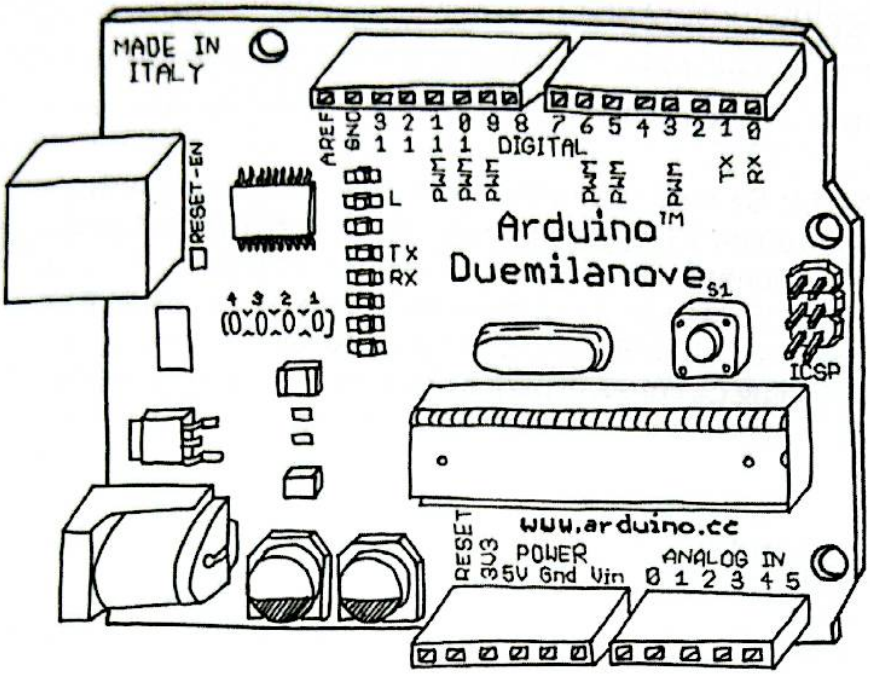


Figura 3.2 – Arduino Duemilanove.

Nessas ilustrações, você pode ver a placa Arduino. De início, todos esses conectores podem lhe parecer um tanto confusos. Veja uma explicação do que cada elemento da placa é capaz de fazer:

14 pinos digitais de entrada/saída (pinos 0-13)

Esses pinos podem ser utilizados como entradas ou saídas, de acordo com o especificado no sketch que você criou no IDE.

6 pinos de entrada analógica (pinos 0-5)

Esses pinos de entrada analógica dedicada recebem valores analógicos (ou seja, leituras de tensão de um sensor) e os convertem em um número entre 0 e 1023.

6 pinos de saída analógica (pinos 3, 5, 6, 9, 10 e 11)

Seis dos pinos digitais que podem ser reprogramados para saída analógica utilizando o sketch criado no IDE.

A placa pode receber sua alimentação a partir da porta USB de seu computador, da maioria dos carregadores USB, ou mesmo de um adaptador AC (recomendamos um de 9 volts, com pino redondo de 2,1 mm e centro positivo). Se não houver nenhuma

Digitalizado com CamScanner

fonte de alimentação ligada ao conector de energia, a alimentação será fornecida pela placa USB; entretanto, assim que você conectar uma fonte, a placa passará a utilizá-la automaticamente.

NOTA: Se você está utilizando as versões mais antigas do Arduino-NG ou Arduino Diecimila, será necessário definir o jumper de seleção de alimentação (marcado como PWR_SEL na placa), para especificar a alimentação EXT (externa), ou USB. Esse jumper pode ser encontrado entre o plug de conexão para o adaptador AC e a porta USB.

Software (IDE)

O Integrated Development Environment (IDE) é um programa especial executado em seu computador que permite a criação de sketches para a placa Arduino em uma linguagem simples, modelada a partir da linguagem Processing (*www.processing.org*). O truque de verdade acontece quando você pressiona o botão que faz o upload do sketch para a placa: o código que você escreveu é traduzido para a linguagem C (geralmente difícil de ser utilizada por iniciantes) e é transmitido para o compilador avr-gcc, importante software de código aberto que realiza a tradução final de seus comandos, agora para uma linguagem que pode ser compreendida pelo microcontrolador. Esse último passo é muito importante, é aqui que o Arduino simplifica sua vida, ocultando ao máximo as complexidades inerentes à programação de microcontroladores.

O ciclo de programação do Arduino pode ser dividido, basicamente, da seguinte maneira:

- Conecte sua placa a uma porta USB de seu computador.
- Escreva um sketch para dar vida à sua placa.
- Faça o upload desse sketch para a placa, utilizando a conexão USB, e aguarde alguns segundos para que ela reinicialize.
- A placa executará o sketch que você criou.

Digitalizado com CamScanner

NOTA: Quando da redação deste livro, a instalação em Linux é um pouco complicada. Visite *www.arduino.cc/playground/Learning/Linux* **para instruções.**

Instalação do Arduino em seu computador

Para programar a placa Arduino, você deve primeiro fazer o download do ambiente de desenvolvimento (o IDE) neste site: *www.arduino.cc/en/Main/Software*. Escolha a versão correta para seu sistema operacional.

Faça o download do arquivo e clique duas vezes nele para abri-lo; no Windows ou no Linux, isso criará uma pasta, com o nome *arduino-[versão]*, como *arduino-1.0*. Arraste a pasta para o local onde deseja deixá-la: seu desktop, sua pasta Arquivos de Programas (no Windows) etc. No Mac, clicar duas vezes no arquivo abrirá uma imagem de disco com uma aplicação do Arduino (arraste-a para sua pasta Aplicações). Agora, sempre que quiser executar o IDE do Arduino, você deverá abrir sua pasta *arduino* (no Windows ou Linux), ou *Aplicações* (no Mac), e clicar duas vezes no ícone do Arduino. Não faça isso imediatamente; ainda falta falarmos sobre um passo.

NOTA: Caso você tenha problemas ao executar o IDE do Arduino, consulte o capítulo 7, que trata da resolução de problemas.

Agora você deve instalar os drivers que permitem ao computador se comunicar com sua placa utilizando a porta USB.

Instalação de drivers: Macintosh

O Arduino Uno em um Mac utiliza os drivers fornecidos pelo sistema operacional, por isso o procedimento é muito simples. Conecte sua placa ao computador.

Na placa, a luz PWR deverá acender e o LED amarelo marcado como "L" deverá começar a piscar.

Digitalizado com CamScanner

Talvez você veja uma janela de pop-up avisando-lhe que uma nova interface de rede foi detectada.

Se isso ocorrer, clique em Network Preferences... (*Preferências de Rede...*) e, quando essa janela abrir, clique em Apply (*Aplicar*). O Uno será mostrado como Not Configured (*Não configurado*), mas estará funcionando corretamente. Feche a janela de Preferências do Sistema.

Caso você esteja utilizando uma placa Arduino mais antiga, consulte as instruções no seguinte endereço: *www.arduino.cc/en/Guide/MacOSX*.

Se o Arduino não funcionar, consulte o capítulo 7, que trata da resolução de problemas.

Instalação de drivers: Windows

Conecte a placa Arduino ao computador; assim que a janela do Assistente de Instalação de Novo Hardware surgir na tela, o Windows tentará localizar o driver no site Windows Update.

O Windows XP perguntará se você deseja verificar o site do Windows Update; caso negativo, selecione a opção No, not at this time (*Não, não desta vez*) e clique em Next (*Avançar*).

Na próxima tela, selecione Install from a list or specific location (*Instalar de uma lista ou local específico*) e clique em Next (*Avançar*).

Navegue e selecione o arquivo de driver do Uno, *ArduinoUNO.inf*, localizado na pasta *Drivers* no download feito do software do Arduino (não no subdiretório *FTDI USB Drivers*). A partir daí, o Windows concluirá sozinho a instalação do driver.

Caso você esteja utilizando uma placa mais antiga, consulte as instruções fornecidas por este site: *www.arduino.cc/en/Guide/Windows*.

Assim que os drivers estiverem instalados, você poderá iniciar o IDE do Arduino e começar a utilizá-lo.

Digitalizado com CamScanner

Agora, você deve descobrir qual porta serial foi atribuída à sua placa Arduino – essa informação será necessária para programá-la futuramente. As instruções para se obter esses dados estão nas seções que veremos a seguir.

Identificação de porta: Macintosh

A partir do menu Tools no IDE do Arduino, selecione Serial Port e escolha a porta que inicia com o nome */dev/cu.usbmodem*; esse é o nome que seu computador utilizará para se referir à placa Arduino. A figura 3.3 mostra a lista de portas.

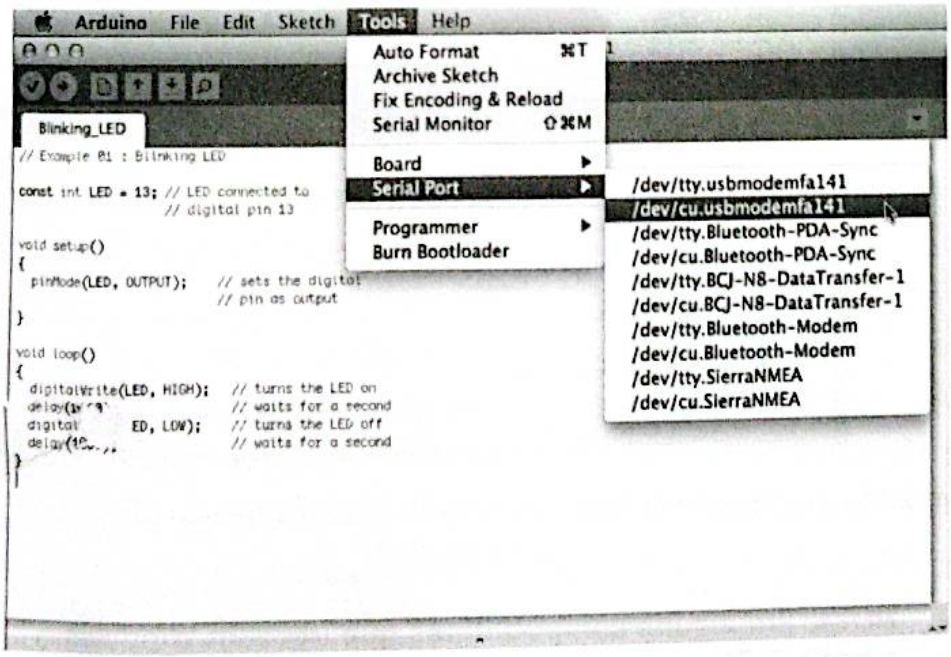


Figura 3.3 – Lista de portas seriais mostradas pelo Arduino IDE.

Identificação de porta: Windows

No Windows, o processo é um pouco mais complicado – ao menos de início. Para abrir o Gerenciador de Dispositivos, clique primeiro no menu Iniciar; depois, com o botão direito, clique em Computador (Vista) ou em Meu Computador (XP) e escolha Propriedades. No Window XP, clique em Hardware e escolha Gerenciador de Dispositivos. No Vista, clique em Gerenciador de Dispositivos (essa opção estará presente na lista de tarefas à esquerda da janela).

Digitalizado com CamScanner

Procure o dispositivo Arduino na lista, sob o texto "Portas (COM & LPT)". O Arduino estará marcado como "Arduino UNO" e terá um nome como COM3 (Figura 3.4).

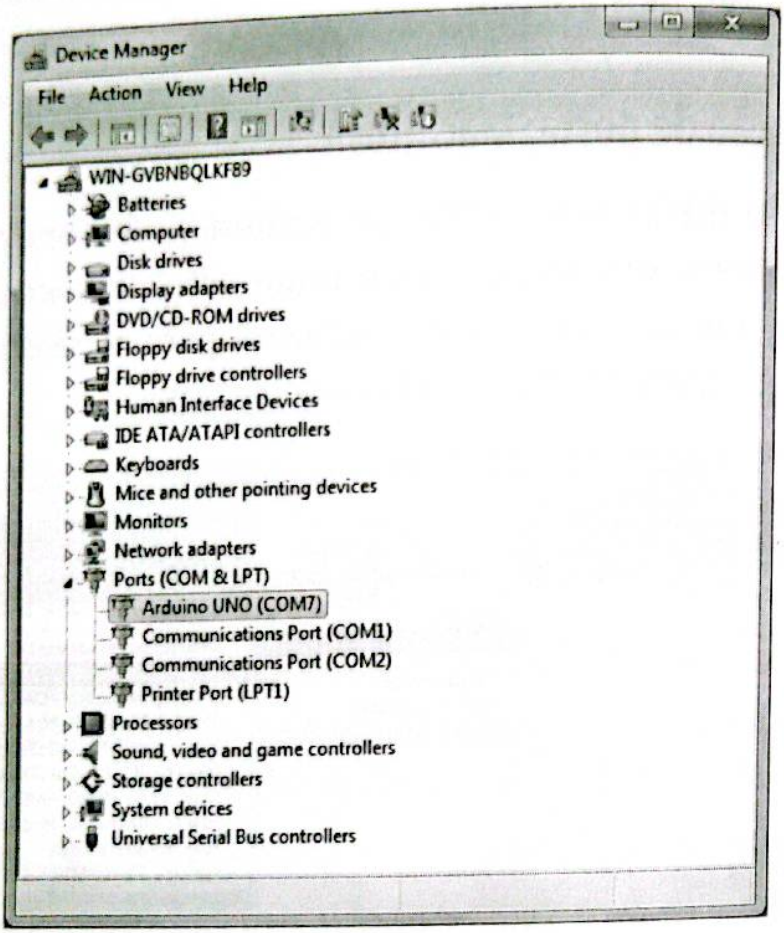


Figura 3.4 – Gerenciador de Dispositivos do Windows mostrando todas as portas seriais disponíveis.

NOTA: Em alguns computadores que utilizam Windows, a porta COM tem um número maior que 9; essa numeração cria alguns problemas quando o Arduino tenta se comunicar com ela. Consulte o capítulo 7, que trata da resolução de problemas, para ajuda nesse caso.

Assim que você encontrar a porta atribuída, selecione-a no menu **Tools > Serial Port** do IDE do Arduino.

Com isso, o ambiente de desenvolvimento do Arduino poderá se comunicar com a placa Arduino e programá-la.

Digitalizado com CamScanner

CAPÍTULO 4

Dando seus primeiros passos com o Arduino

Nesta seção você aprenderá a montar e programar um dispositivo interativo.

Anatomia de um dispositivo interativo

Todos os objetos que podemos criar utilizando o Arduino seguem um padrão bem simples, que chamamos de "dispositivo interativo" (Figura 4.1). O dispositivo interativo é um circuito eletrônico capaz de perceber o ambiente ao seu redor utilizando sensores (componentes eletrônicos que convertem medidas do mundo real em sinais elétricos). O dispositivo processa a informação que recebe dos sensores, utilizando um comportamento implementado como software, e é capaz de interagir com o mundo por meio de atuadores, componentes eletrônicos que podem converter um sinal elétrico em uma ação física.

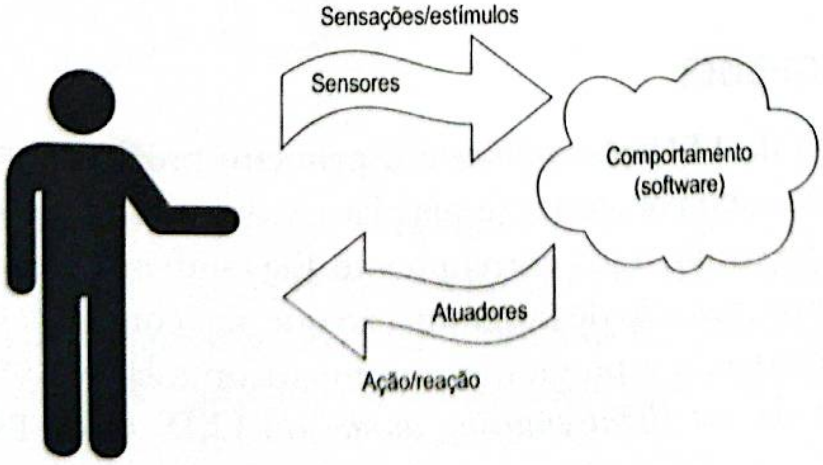


Figura 4.1 – Dispositivo interativo.

Digitalizado com CamScanner

Sensores e atuadores

Sensores e atuadores são componentes eletrônicos que permitem a um equipamento eletrônico interagir com o mundo.

Como o microcontrolador é um computador bem simples, ele pode processar apenas sinais elétricos (algo como os impulsos elétricos que são transmitidos entre os neurônios de nosso cérebro). Para que capte luz, temperatura, ou outros dados físicos, ele necessita de algo capaz de converter esses sinais em eletricidade. Em nosso corpo, por exemplo, o olho converte a luz em sinais que são enviados ao cérebro por meio de nervos. Nesse caso, em eletrônica podemos utilizar um dispositivo simples, que chamamos de resistor dependente de luz (*light-dependent resistor*, ou LDR), capaz de medir a quantidade de luz que o atinge e transmitir esses dados como um sinal que pode ser compreendido pelo microcontrolador.

Assim que os sensores são lidos, o dispositivo terá a informação necessária para que possa decidir como reagir. O processo de tomada de decisão é realizado pelo microcontrolador, e a reação é realizada por atuadores. Em nossos corpos, por exemplo, músculos recebem sinais elétricos de nosso cérebro e os convertem em movimento. No mundo eletrônico, essas funções poderiam ser implementadas por uma fonte de luz ou um motor elétrico.

Nas seções que veremos a seguir, você aprenderá a ler sensores de vários tipos e a controlar diferentes atuadores.

LED pisca-pisca

O sketch do LED pisca-pisca é o primeiro programa que você deverá executar para testar se sua placa Arduino está funcionando e se ela foi configurada corretamente. Ele também é, geralmente, o primeiro exercício de programação que alguém realiza quando está aprendendo a programar um microcontrolador. Um diodo emissor de luz (*light-emitting diode*, ou LED) é um pequeno

Digitalizado com CamScanner

componente eletrônico semelhante a uma lâmpada, porém mais eficiente e que requer tensões mais baixas para operar.

Sua placa Arduino vem com um LED pré-instalado. Ele está marcado como "L". Você também pode adicionar seu próprio LED – basta conectá-lo, como mostra a figura 4.2.

Se você pretende manter o LED aceso por um longo intervalo de tempo, deve utilizar um resistor, como descrito na página 72.

Na figura, K indica o cátodo (negativo), ou o terminal mais curto, enquanto A indica o ânodo (positivo), ou o terminal mais longo.

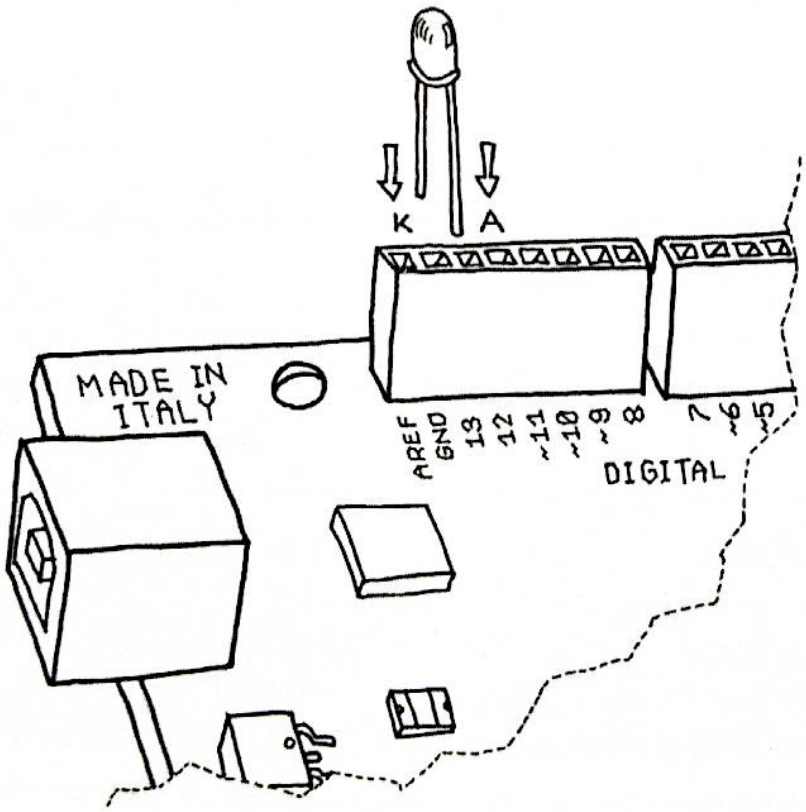


Figura 4.2 – Conexão de um LED ao Arduino.

Assim que o LED tiver sido conectado, você deverá dizer ao Arduino o que deve ser feito. Isso é feito por meio de código-fonte, ou seja, utilizando uma lista de instruções que fornecemos ao microcontrolador para que ele realize o que pretendemos.

Digitalizado com CamScanner

Em seu computador, abra a página para a qual você copiou o IDE do Arduino. Clique duas vezes no ícone do Arduino para iniciá-lo. Selecione File > New e escolha um nome para a pasta do sketch: é nela que seu sketch do Arduino será armazenado. Dê a ela o nome *Blinking_LED* e clique em OK. Depois, digite o texto que mostramos a seguir (Exemplo 01) no editor de sketch do Arduino (a janela principal do IDE do Arduino). Você também pode efetuar o download desse exemplo no site *makezine.com/getstartedarduino*. O resultado deve ser igual ao que mostra a figura 4.3.

```
// Exemplo 01 : LED pisca-pisca

const int LED = 13;    // LED conectado ao pino digital 13

void setup()
{
    pinMode(LED, OUTPUT);  // define o pino digital como saída
}

void loop()
{
    digitalWrite(LED, HIGH);        // acende o LED
    delay(1000);                    // aguarda um segundo
    digitalWrite(LED, LOW);         // apaga o LED
    delay(1000);                    // aguarda um segundo
}
```

Digitalizado com CamScanner

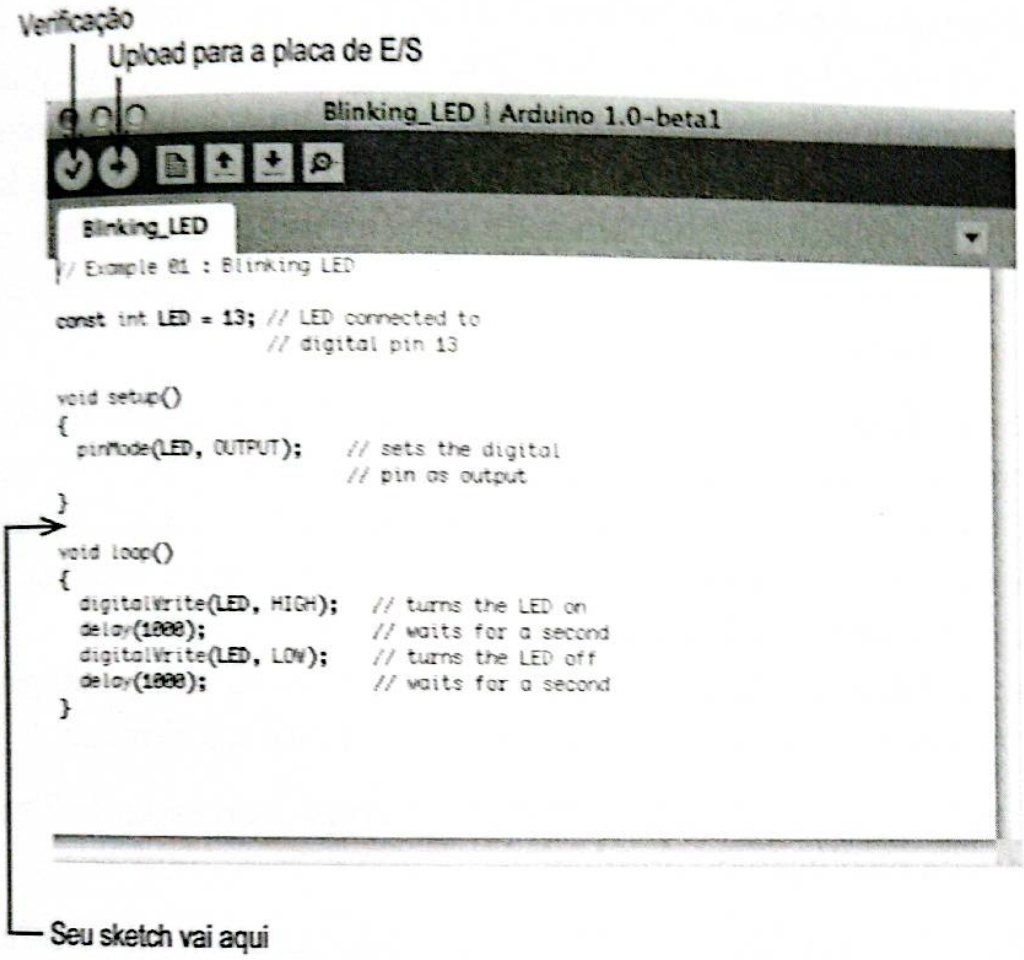


Figura 4.3 – IDE do Arduino com seu primeiro sketch carregado.

Agora que o código-fonte está em seu IDE, você deve verificar se ele está correto. Pressione o botão de verificação (a figura 4.3 mostra sua localização); se tudo estiver correto, você verá surgir a mensagem "Done compiling" (Compilação encerrada) na base do IDE do Arduino. O surgimento dessa mensagem significa que o IDE do Arduino traduziu seu sketch em um programa executável que pode ser executado pela placa, algo como um arquivo *.exe* no Windows, ou *.app* no Mac.

Agora você pode fazer o upload do programa para a placa pressionando o botão de upload (Figura 4.3). Isso reiniciará a placa, fazendo com que ela interrompa o que está fazendo e aguarde instruções vindas da porta USB. O IDE do Arduino enviará o sketch atual para a placa, que o armazenará em sua memória, executando-o quando apropriado.

Digitalizado com CamScanner

Você verá algumas mensagens surgirem na área em cor preta na base janela; logo acima dessa área, você verá surgir a mensagem "Done uploading" (Upload encerrado) para indicar que o processo foi concluído corretamente. Há dois LEDs, marcados como RX e TX, na placa; eles piscam toda vez que um byte é enviado ou recebido pela placa. Durante o processo de upload, eles piscarão ininterruptamente.

Se os LEDs não piscarem, ou se você receber uma mensagem de erro em vez de "Done uploading", então deve haver um problema de comunicação entre seu computador e o Arduino. Certifique-se de que você selecionou a porta serial correta para a comunicação de seu computador com o Arduino (consulte o capítulo 3) no menu Tools > Serial Port. Verifique também o menu Tools > Board para confirmar se o modelo correto do Arduino foi selecionado.

Se você ainda estiver tendo problemas, consulte o capítulo 7, que trata da resolução de problemas.

Assim que o código tiver sido transmitido para sua placa Arduino, ele ficará nela até que outro sketch o substitua. Um sketch sobreviverá mesmo se a placa for reinicializada ou desligada, da mesma forma que os dados do disco rígido de seu computador.

Supondo que o upload do sketch ocorra sem problemas, você verá o LED "L" acender por um segundo e, em seguida, apagar por um segundo. Se você instalou um LED separado, como na figura 4.2, ele também piscará. Isso que você acabou de escrever e executar é um "programa de computador", ou *sketch*, como os programas do Arduino são chamados. O Arduino, como mencionei, é um pequeno computador e pode ser programado para fazer aquilo que você deseja. Isso é feito utilizando uma linguagem de programação para digitar uma série de instruções no IDE do Arduino, que as transformará em um executável para sua placa Arduino.

Na próxima seção lhe ensinarei a compreender como funciona o sketch. Em primeiro lugar, o Arduino executa o código de cima

Digitalizado com CamScanner

para baixo, por isso a primeira linha no topo é a que será lida primeiro; depois, ele avança, descendo pelas linhas da mesma forma que o *playhead* (botão da barra de controle) de um player de vídeos, como o QuickTime Player ou o Windows Media Player, movimenta-se da esquerda para a direita, indicando que parte do vídeo você está assistindo.

Passe-me o parmesão

Note que, no código-fonte, utilizamos chaves para agrupar linhas de código. Eles são especialmente úteis quando você deseja dar nome a um grupo de instruções. Se você estiver em um jantar e pedir a alguém: "Por favor, passe-me o queijo parmesão", isso dará início a uma séria de ações, que podem ser resumidas pela pequena frase que você acabou de dizer. Como somos humanos, isso ocorre naturalmente; no caso do Arduino, todas as pequenas ações necessárias para que a ação tenha efeito têm de ser explicitadas, uma vez que ele não é tão poderoso como nosso cérebro. Assim, para agrupar algumas instruções, você deve colocar um { antes de seu código e um } depois dele.

Você pode ver que há dois blocos de código definidos dessa forma. Antes de cada um temos um comando que talvez você não conheça:

```
void setup()
```

Essa linha dá nome a um bloco de código. Se você tivesse de escrever uma lista de instruções para ensinar o Arduino a entregar-lhe o parmesão, você escreveria `void passeoparmesao()` no início de um bloco, e esse bloco se tornaria uma instrução que poderia ser chamada em qualquer ponto do código do Arduino. Esses blocos são chamados *funções*. Se, depois disso, você escrever `passeoparmesao()` em qualquer ponto de seu código, o Arduino executará essas instruções e continuará de onde parou.

Digitalizado com CamScanner

O Arduino não permite o encerramento de programas

O Arduino espera que duas funções existam – uma chamada `setup()` e outra, `loop()`.

Em `setup()` você coloca todo o código que deseja executar apenas uma vez no início de seu programa, enquanto que `loop()` contém o núcleo de seu programa, executado seguidas vezes. Esse padrão deve ser seguido, pois o Arduino não é como um computador normal – ele não pode executar diversos programas ao mesmo tempo e os programas não podem ser encerrados. Quando você ligar a placa, o Arduino executará essas instruções, iniciando do ponto em que parou.

Verdadeiros experimentadores escrevem comentários

Qualquer texto iniciado por // será ignorado pelo Arduino. Essas linhas são comentários, notas que você deixa no programa para que possa se lembrar do que fez quando escreveu o código, ou para outras pessoas, facilitando a compreensão de seu programa.

É muito comum (sei disso porque faço isso o tempo todo) escrevermos algum código, efetuarmos seu upload para a placa e dizermos "Perfeito – nunca mais terei de mexer nesse código!", apenas para percebermos, seis meses depois, que teremos de atualizar o código ou corrigir um bug. Nessas situações, se você não tiver incluído seus comentários no programa original, pensará "Caramba – que bagunça! Por onde começo?" Conforme avançamos, você verá alguns truques que mostram como tornar seus programas mais legíveis e como facilitar sua manutenção.

O código-fonte, passo a passo

De início, você pode considerar esse tipo de explicação desnecessária, assim como eu, quando estava na escola e tive de estudar a *Divina Comédia* de Dante (na Itália, todo aluno tem de estudar esse livro, assim como outro, *I promessi sposi*, ou *Os Noivos* – quantos

Digitalizado com CamScanner

pesadelos!). Para cada linha dos poemas, tínhamos centenas de linhas de comentários! Ainda assim, trabalhar com comentários será muito mais útil em seu caso, conforme você passa a escrever seus próprios programas.

```
// Exemplo 01 : LED pisca-pisca
```

Um comentário é uma ótima forma de escrevermos pequenas notas. Esse pequenino exemplo serve para lembrar-nos de que este programa, o Exemplo 01, pisca um LED.

```
const int LED = 13;     // LED conectado ao pino digital 13
```

`const int` significa que esse LED é um número inteiro que não pode ser alterado (ou seja, uma constante) cujo valor está definido como 13. É como uma busca e substituição automática em seu código; nesse caso, isso serve para dizer ao Arduino que o número 13 deve ser escrito toda vez que a palavra LED aparecer. Utilizamos esse comando para especificar que o LED que estamos piscando está conectado ao pino 13 do Arduino.

```
void setup()
```

Essa linha diz ao Arduino que o próximo bloco de código será chamado `setup()`.

```
{
```

Com essa chave de abertura, iniciamos um bloco de código.

```
pinMode(LED, OUTPUT);       // define o pino digital como saída
```

Na sequência, uma instrução interessante. `pinMode` diz ao Arduino como configurar determinado pino. Pinos digitais podem ser utilizados como INPUT (entrada) ou OUTPUT (saída). Nesse caso, necessitamos de um pino de saída para controlar nosso LED, por isso colocamos o número do pino e seu modo dentro dos parênteses.

Digitalizado com CamScanner

`pinMode` é uma função e as palavras (ou números) especificadas dentro dos parênteses são *argumentos*. `INPUT` e `OUTPUT` são constantes na linguagem do Arduino. (Assim como variáveis, constantes recebem valores, mas estes são predefinidos e nunca se alteram).

```
}
```

Essa chave de encerramento significa o fim da função `setup()`.

```
void loop()
{
```

`loop()` é o local em que você especifica o comportamento principal de seu dispositivo interativo. `loop()` se repetirá até que você desligue a placa.

```
digitalWrite(LED, HIGH);        // acende o LED
```

Como diz o comentário, `digitalWrite()` é capaz de ligar (ou desligar) qualquer pino que foi configurado como OUTPUT. O primeiro argumento (nesse caso, `LED`) especifica qual pino deve ser ligado ou desligado (lembre-se de que `LED` é um valor constante que se refere ao pino 13, por isso esse é o pino que será ligado ou desligado). O segundo argumento pode ligar (`HIGH`) ou desligar (`LOW`) o pino.

Suponha que todo pino de saída é uma pequena tomada, como as que você tem nas paredes de seu apartamento ou casa. As tomadas europeias são de 230V. As americanas, de 110V e o Arduino funciona com modestos 5V. O truque aqui ocorre quando o software se torna hardware. Quando você escreve `digitalWrite(LED, HIGH)`, liga o pino digital com 5V, fazendo com que, se um LED estiver conectado, ele se acenda. Assim, nesse ponto de seu código, uma instrução em software faz com que algo ocorra no mundo físico controlando o fluxo de eletricidade que é direcionado ao pino. Ao ligar e desligar o pino, podemos transformar esses comandos em algo mais visível a um ser humano; o `LED` é nosso atuador.

Digitalizado com CamScanner

```
delay(1000);    // aguarda um segundo
```

O Arduino tem uma estrutura muito simples. Assim, se você deseja que algo ocorra com certa regularidade, deve aguardar pacientemente até que chegue o momento do próximo passo. Basicamente, `delay()` faz com que o processador não faça nada e aguarde o tempo em milissegundos transmitido como argumento. Milissegundos são milésimos de segundos; portanto, 1.000 milissegundos são iguais a 1 segundo. Com esse comando, o LED ficará aceso por um segundo.

```
digitalWrite(LED, LOW);    // apaga o LED
```

Essa instrução apaga o LED que havíamos acendido. Por que utilizamos `HIGH` e `LOW`? Bem, essa é uma antiga convenção de eletrônica digital. `HIGH` significa que o pino está ligado e, no caso do Arduino, definido com 5V. `LOW` significa 0V. Para facilitar, você pode mentalmente substituir esses argumentos por `ON` (ligado) e `OFF` (desligado).

```
delay(1000);    // aguarda um segundo
```

Aqui, aguardamos mais um segundo. O LED ficará apagado por um segundo.

```
}
```

O colchete de encerramento marca o fim da função de loop.

Em resumo, esse programa faz o seguinte:

- Transforma o pino 13 em uma saída (apenas uma vez, no início).
- Entra em um loop.
- Acende o LED conectado ao pino 13.
- Aguarda um segundo.
- Apaga o LED conectado ao pino 13.
- Aguarda um segundo.
- Retorna ao início do loop.

Digitalizado com CamScanner

Espero que isso não tenha sido tão doloroso. Você aprenderá mais sobre programação à medida que executar os outros exemplos do livro.

Antes de avançarmos à próxima seção, quero que você brinque um pouco com o código. Por exemplo, reduza o tempo de espera, utilizando números diferentes para os pulsos de ligado e desligado, de modo a produzir padrões distintos com o LED. Mais especificamente, descubra o que ocorre quando você torna seus intervalos muito curtos, mas utiliza intervalos distintos para os estados de aceso e apagado... perceba que há um momento em que algo estranho acontece; esse "algo" será muito útil quando falarmos sobre modulação por largura de pulso.

O que construiremos

Sempre fui fascinado pela luz e pela capacidade de controlar diferentes fontes de luz utilizando recursos de tecnologia. Tive sorte de trabalhar em alguns projetos muito interessantes que envolviam o controle de fontes de luz e sua interação com pessoas. O Arduino é perfeito para isso. Neste livro, você trabalhará no projeto de "lâmpadas interativas", utilizando o Arduino para aprender o básico sobre a construção de dispositivos interativos.

Na próxima seção, tentarei explicar o básico sobre eletricidade, de uma forma que provavelmente aborrecerá um engenheiro, mas que não assustará um iniciante dando seus primeiros passos na programação com o Arduino.

O que é eletricidade?

Se você já fez algum trabalho de encanamento em sua casa, não terá problemas com eletrônica. Para compreender como a eletricidade e circuitos elétricos funcionam, a melhor forma é utilizar algo que chamamos de "analogia da água". Vamos analisar um dispositivo simples, como o ventilador portátil à pilha que temos na figura 4.4.

Digitalizado com CamScanner

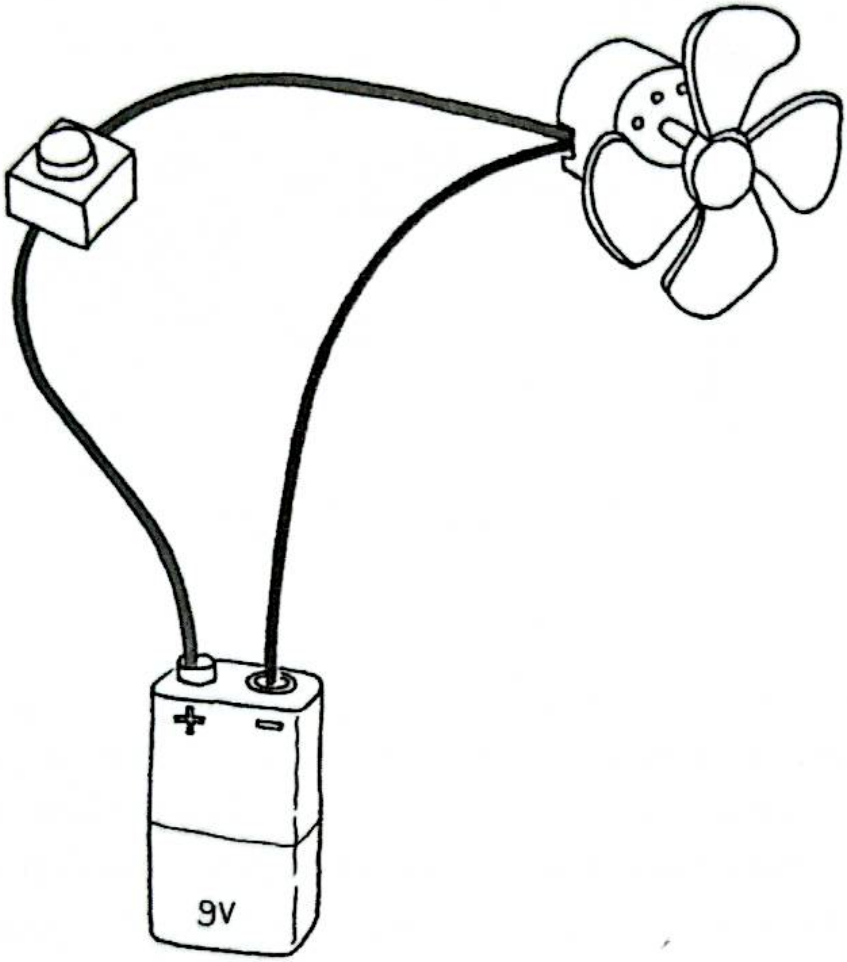


Figura 4.4 – Ventilador portátil.

Se você desmontar um ventilador, verá que ele contém uma pequena bateria, alguns fios e um motor elétrico, e que um dos fios que parte na direção do motor é interrompido por uma chave. Se o ventilador utiliza uma pilha nova e a chave estiver ligada, o motor começará a girar, produzindo o efeito desejado. Como isso funciona? Bem, imagine que a bateria corresponde a um reservatório de água e a uma bomba de água, que a chave é uma torneira e que o motor é uma daquelas rodas que vemos em moinhos d'água. Quando você abre a torneira, a água flui da bomba e faz com que a roda se movimente.

Nesse sistema hidráulico simples (Figura 4.5), dois fatores são importantes: a pressão da água (determinada pelo poder da bomba) e a quantidade de água que flui nos canos (isso depende do tamanho dos canos e da resistência que a roda fornece ao fluxo de água).

Digitalizado com CamScanner

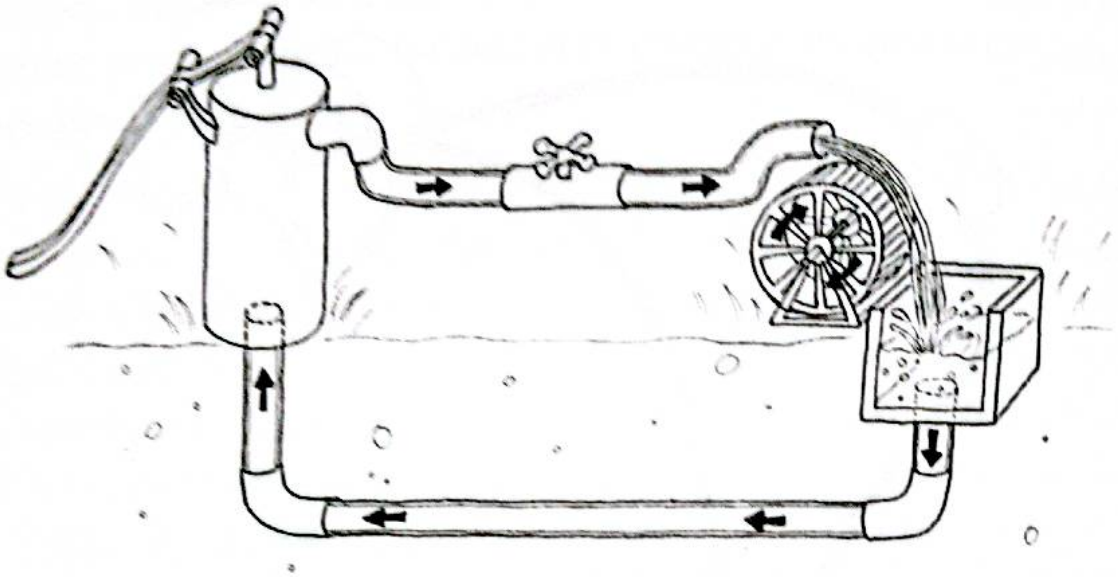


Figura 4.5 – Sistema hidráulico.

Você rapidamente perceberá que se quiser que a roda gire mais rápido terá de aumentar o tamanho de seus canos (o que funcionará até certo ponto) e aumentar a pressão que a bomba pode atingir. Ao aumentar o tamanho dos canos, permitimos a passagem de um fluxo maior de água e acabamos por reduzir a resistência que eles oferecem ao processo. Essa abordagem produz resultados apenas até certo ponto: eventualmente a roda deixará de girar mais rápido, pois a pressão da água não será suficientemente forte. Quando atingirmos esse ponto, será necessário tornar a bomba mais forte. Esse segundo método de aceleração do moinho pode ser utilizado até que a roda se quebre em razão da força do fluxo de água. Outro ponto de importância é que, à medida que a roda gira, o eixo esquenta um pouco, uma vez que, independentemente de nossa capacidade na montagem da roda, a fricção entre o eixo e os orifícios nos quais ele foi montado gera calor. É importante compreender que, em um sistema como esse, nem toda a energia bombeada para o sistema será convertida em movimento; parte dela será dissipada em razão de certas ineficiências, o que geralmente acabará resultando na emissão de calor por algumas partes do sistema.

Digitalizado com CamScanner

Quais são as partes mais importantes desse sistema? A pressão produzida pela bomba é uma delas; a resistência que os canos e a roda oferecem ao fluxo de água, assim como o fluxo em si (digamos que ele seja representado pelo número de litros de água que fluem em um segundo) são as outras. A eletricidade funciona mais ou menos como a água. Temos um tipo de bomba (qualquer fonte de eletricidade, como uma pilha ou uma tomada) que transmite cargas elétricas (imagine-as como "gotas" de eletricidade) pelos canos, nesse caso representado pelos fios – alguns dispositivos são capazes de utilizar essas cargas para produzir calor (o cobertor elétrico de sua avó), luz (o abajur de seu quarto), som (seu aparelho de som), movimento (seu ventilador) e muito mais.

Assim, ao ler que a tensão de uma bateria é 9V, pense nesse dado como a pressão de água que pode ser eventualmente produzida por essa pequena "bomba". A tensão é medida em *volts*, em homenagem a Alessandro Volta, inventor da primeira pilha.

Da mesma forma que a pressão fornecida pela água tem seu equivalente elétrico, o fluxo da água também o tem. Chamamos esse dado de corrente e o medimos em *ampères* (em homenagem a André-Marie Ampère, pioneiro do eletromagnetismo). O relacionamento entre tensão e corrente pode ser novamente ilustrado utilizando o moinho d'água: uma maior tensão (pressão) permite que você gire a roda mais rapidamente; um maior fluxo (corrente) permite que você gire uma roPor fim, a resistência oferecida ao fluxo da corrente em qualquer trajeto percorrido é chamada de – você adivinhou – resistência, medida em *ohms* (em homenagem ao físico alemão Georg Ohm). Herr Ohm também foi responsável por formular a mais importante lei da eletricidade – e a única fórmula da qual você não deve se esquecer. Ele foi capaz de demonstrar que, em um circuito, a tensão, a corrente e a resistência estão todas relacionadas; mais especificamente, a resistência de um circuito determina a quantidade de corrente que o percorre, dada certa tensão.

Digitalizado com CamScanner

Se pararmos para pensar, veremos que isso é bem intuitivo. Pegue uma bateria de 9V e conecte-a a um circuito simples. Medindo a corrente, você perceberá que, quanto mais resistência adicionamos ao circuito, menos corrente viaja por ele. Retornando à analogia da água: considerando certa bomba, se instalarmos uma válvula (que, em eletricidade, podemos comparar a um *resistor variável*), quanto mais eu fechar a válvula – aumentando a resistência ao fluxo da água – menos água percorrerá os canos. Ohm resumiu essa lei nestas fórmulas:

```
R (resistência) = V (tensão) / I (corrente)
V = R * I
I = V / R
```

Essa é a única regra que você deve memorizar e aprender a utilizar, uma vez que, na maioria dos casos, é a única que será realmente necessária.

Uso de um botão de pressão para controlar o LED

Fazer um LED piscar foi fácil, mas não acredito que você conseguirá manter sua sanidade mental se o abajur de sua mesa não parar de piscar enquanto você tenta ler um livro. Dessa forma, nesta seção aprenderemos a controlar esses efeitos. Em nosso exemplo anterior, o LED era nosso atuador e o Arduino o controlava. A peça que está faltando é um sensor.

Nesse caso, utilizaremos o tipo mais simples de sensor disponível: um botão de pressão (*push button*).

Se você desmontar um botão de pressão, verá que se trata de um dispositivo muito simples: duas peças de metal separadas por uma mola e uma cobertura plástica que, quando pressionada, provoca o contato das duas peças de metal. Quando essas peças estão afastadas, não há circulação de corrente no botão (assim como quando uma válvula de água é fechada); quando pressionamos o botão, fazemos uma conexão.

Digitalizado com CamScanner

Para monitorar o estado de uma chave, há uma nova instrução do Arduino que você deve conhecer: a função `digitalRead()`.

`digitalRead()` confere se há tensão aplicada ao pino especificado nos parênteses e retorna um valor `HIGH` ou `LOW`, dependendo do que foi verificado. As outras instruções que utilizamos até aqui não retornavam informações – apenas executavam o que lhes era solicitado. Mas o uso desse tipo de instrução é um pouco limitado, pois nos obriga a seguir sequências muito previsíveis de instruções, sem nenhuma influência externa. Com `digitalRead()`, podemos "fazer uma pergunta" ao Arduino e receber uma resposta, que poderá ser armazenada em algum ponto da memória, e utilizada, posteriormente, na tomada de decisões.

Monte o circuito da figura 4.6. Para fazê-lo, você deverá adquirir alguns componentes (que também serão úteis em projetos futuros):

» **Uma protoboard sem solda**: código MKKN3 no Maker Shed (*www.makershed.com*) e que também pode ser encontrada na Arduino Store (*bit.ly/ArduinoStoreBreadBoard*). O apêndice A apresenta uma introdução a protoboards sem solda.

» **Um kit de fios jumper pré-cortados**: MKNN4 no Maker Shed. Na Arduino Store eles acompanham a protoboard.

» **Um resistor de 10K ohms**: JM691104 no Maker Shed (pacote com 100) e que também pode ser encontrado na Arduino Store (*bit.ly/ArduinoStore10k*, pacote com 10).

» **Um botão de pressão táctil momentânea**: JM119011 no Maker Shed e que também pode ser encontrado na Arduino Store (*bit.ly/ArduinoStorePushButtons*).

Digitalizado com CamScanner

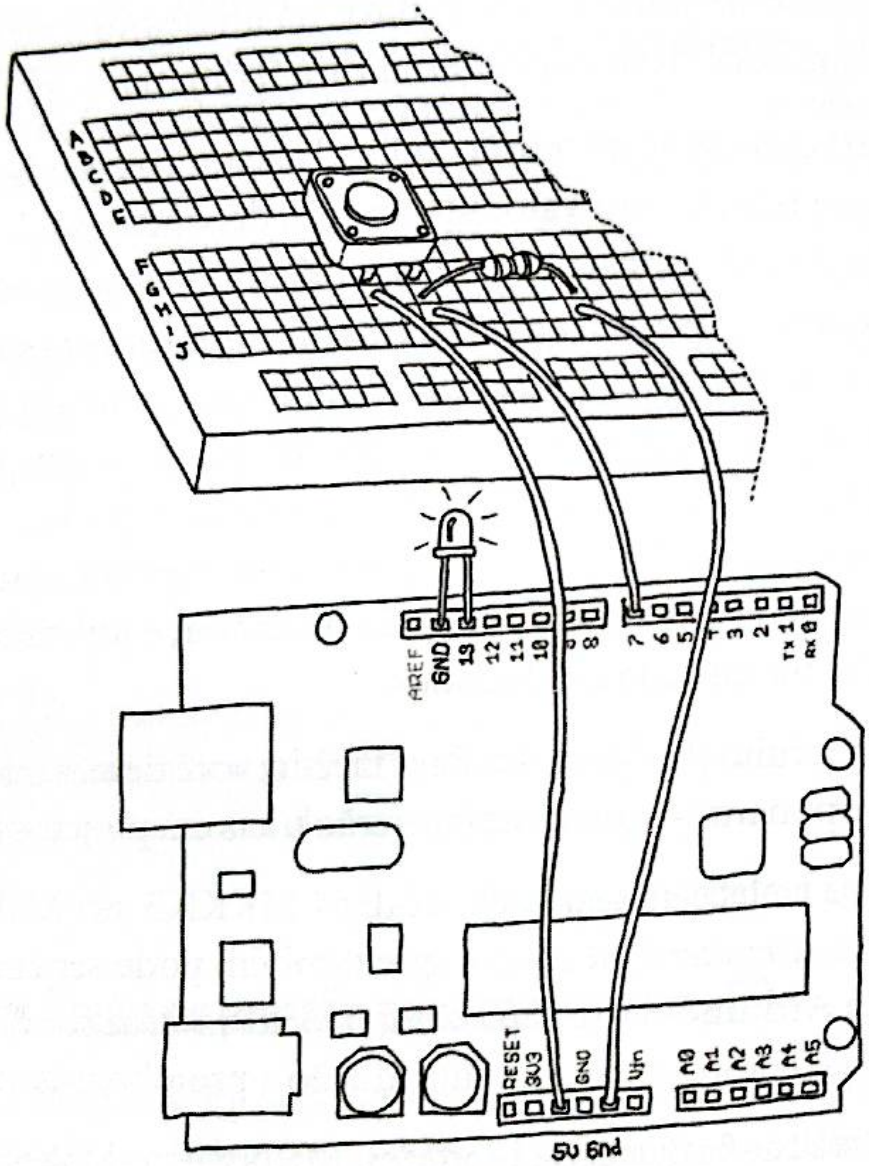


Figura 4.6 – Conectando um botão de pressão.

NOTA: Em vez de comprar fios jumper pré-cortados, você também pode adquirir fios de núcleo sólido 22 AWG em pequenos rolos e desencapá-los utilizando cortadores e desencapadores de fios.

Veja, a seguir, o código que utilizaremos para controlar o LED com nosso botão de pressão:

Digitalizado com CamScanner

```
// Exemplo 02: Acende o LED enquanto o botão estiver pressionado

const int LED = 13;     // pino para o LED
const int BUTTON = 7;   // pino da entrada na qual o
                        // botão de pressão está conectado
int val = 0;            // val será utilizada para armazenar o estado
                        // do pino de entrada

void setup() {
    pinMode(LED, OUTPUT);      // avisa o Arduino que LED é uma saída
    pinMode(BUTTON, INPUT);    // e que BUTTON é uma entrada
}

void loop() {
    val = digitalRead(BUTTON); // lê e armazena o valor da entrada

    // verifica se a saída está HIGH (botão pressionado)
    if (val == HIGH) {
        digitalWrite(LED, HIGH);   // acende o LED
    } else {
        digitalWrite(LED, LOW);
    }
}
```

No Arduino, selecione File > New (se você estiver com outro sketch aberto, é melhor salvá-lo primeiro). Quando o Arduino pedir que você nomeie a pasta de seu novo sketch, escreva *PushButton-Control*. Digite o código do exemplo 2 no Arduino (ou faça seu download no site *www.makezine.com/getstartedarduino* e cole o texto no IDE). Se tudo estiver certo, o LED acenderá quando você pressionar o botão.

Digitalizado com CamScanner

Como isso funciona?

Apresentei dois novos conceitos nesse programa de exemplo: funções que retornam o resultado de seu trabalho e a instrução `if`.

`if` talvez seja a instrução mais importante em uma linguagem de programação, pois permite que o computador (lembre-se: o Arduino é um pequeno computador) tome decisões. Depois da palavra-chave `if`, você deve escrever uma "questão" entre parênteses. Se a "resposta", ou resultado, for verdadeiro, o primeiro bloco de código será executado; do contrário, o bloco de código depois de `else` será executado. Perceba que utilizei o símbolo `==`, em vez de `=`. O primeiro é utilizado quando duas entidades estão sendo comparadas e retorna `TRUE` ou `FALSE`; o segundo atribui um valor a uma variável. Certifique-se de utilizar o sinal correto, pois é muito fácil esquecer essa distinção e utilizar apenas =, impedindo que seu programa funcione corretamente. Sei disso porque, mesmo depois de 25 anos de programação, às vezes ainda me engano.

Ter de segurar um botão pressionado enquanto você necessita de luz não é algo prático, ainda que isso possa fazer com que você perceba quanta energia desperdiça quando deixa uma lâmpada acesa desnecessariamente. Temos de descobrir como fazer para que a lâmpada permaneça acesa, mesmo depois de tirarmos nosso dedo do botão.

Um circuito, mil comportamentos

A grande vantagem da eletrônica digital e programável sobre a eletrônica clássica se tornará evidente no próximo exemplo: mostrarei como você pode implementar muitos "comportamentos" diferentes utilizando o mesmo circuito eletrônico da seção anterior e alterando apenas o software.

Como mencionei, não é muito prático ter de manter seu dedo sobre o botão para que a luz permaneça acesa. Assim, teremos de implementar algum tipo de "memória", na forma de um mecanismo de

Digitalizado com CamScanner

software que se recorde de quando pressionamos o botão e que mantenha a luz acesa mesmo depois de afastarmos nosso dedo.

Para fazê-lo, utilizaremos o que chamamos de *variável*. (Já utilizamos uma, mas ainda não as expliquei). Uma variável é um local na memória do Arduino no qual você pode armazenar dados. Pense nelas como aquelas pequenas anotações que utilizamos para nos lembrar de algo, como um número de telefone: você pega um papel, escreve nele "Luísa 02 555 1212" e cola esse lembrete no monitor de seu computador ou em sua geladeira. Na linguagem do Arduino o processo é igualmente simples: você deve decidir que tipo de dados deseja armazenar (um número ou algum texto, por exemplo), dar um nome a ele, e, quando quiser, armazenar os dados ou recuperá-los. Por exemplo:

```
int val = 0;
```

`int` significa que nossa variável armazenará um número inteiro, `val` é o nome da variável e `= 0` atribui a ela um valor inicial de zero.

Uma variável, como indica seu nome, pode ser modificada em qualquer ponto de seu código. Assim, futuramente em seu programa, você poderia escrever:

```
val = 112;
```

Com essa linhas, você atribuiria à variável um novo valor, 112.

NOTA: Percebeu como, no Arduino, toda instrução termina com um ponto e vírgula? Isso é feito para que o compilador (parte do Arduino que transforma seu sketch em um programa que o microcontrolador é capaz de executar) saiba que sua instrução está completa e que uma nova instrução virá logo em seguida. Lembre-se de utilizar sempre essa pontuação.

No programa que veremos a seguir, `val` será utilizada para armazenar o resultado de `digitalRead()`; qualquer dado que o Arduino receba da entrada acabará armazenado na variável e permanecerá lá até que outra linha de código altere esse dado. Note que variáveis

Digitalizado com CamScanner

utilizam um tipo de memória que chamamos de RAM. Ela é bem rápida, mas quando você desliga sua placa todos os dados armazenados nela acabam perdidos (o que significa que cada variável será redefinida com seu valor inicial quando a placa for religada). Seus programas em si estão armazenados em memória flash – do mesmo tipo utilizado por seu celular para armazenar números de telefone – a qual retém seu conteúdo mesmo quando desligada.

Agora, vamos utilizar outra variável para registrar se o LED deve ficar aceso ou apagado depois que pararmos de pressionar o botão. O exemplo 03A é uma primeira tentativa nesse sentido:

```
// Exemplo 03A: Acende o LED quando o botão é pressionado,
// mantendo-o aceso mesmo depois que deixamos de pressionar o botão.

const int LED = 13;     // pino para o LED
const int BUTTON = 7;   // pino de entrada ao qual o
                        // botão de pressão está conectado
int val = 0;            // val será utilizada para armazenar o estado
                        // do pino de entrada
int state = 0;          // 0 = LED apagado 1 = LED aceso

void setup() {
    pinMode(LED, OUTPUT);      // diz ao Arduino que LED é uma saída
    pinMode(BUTTON, INPUT);    // e que BUTTON é uma entrada
}

void loop() {
    val = digitalRead(BUTTON); // lê e armazena o valor da entrada

    // verifica se a entrada é HIGH (botão pressionado) e altera o estado
    if (val == HIGH) {
        state = 1 - state;
    }
```

Digitalizado com CamScanner

```
    if (state == 1) {
        digitalWrite(LED, HIGH);   // acende o LED
    } else {
        digitalWrite(LED, LOW);
    }
}
```

Agora, teste o código. Veja que ele funciona... mais ou menos; a luz altera seu estado tão rapidamente que é praticamente impossível utilizarmos o botão para definir com precisão se ela deve ficar acesa ou apagada.

Vamos analisar as partes mais interessantes do código: `state` é uma variável que armazena 0 ou 1 para recordar se o LED está aceso ou apagado. Depois que o botão é solto, nós a inicializamos com 0 (LED apagado).

Depois, lemos o atual estado do botão; se ele estiver pressionado (`val == HIGH`), alteramos o estado de 0 para 1 e vice-versa. Uma vez que o estado pode ser apenas 1 ou 0, isso é feito utilizando um pequeno truque que envolve uma simples expressão matemática, com base no fato de que 1 - 0 é 1 e 1 - 1 é 0.

```
state = 1 - state
```

Essa linha pode não fazer muito sentido em matemática, mas faz em programação. O símbolo = significa "atribua o resultado daquilo que vem depois de mim à variável nomeada antes de mim" – nesse caso, o novo valor de `state` recebe o valor correspondente a 1 menos o velho valor de `state`.

Mais à frente, no programa, você pode ver que utilizamos `state` para verificar se o LED deve estar ligado ou desligado. Como mencionei, essa implementação acaba por produzir resultados imprecisos.

Isso ocorre pela forma como lemos o botão. O Arduino é muito rápido e executa suas próprias instruções internas a uma taxa de 16 milhões de instruções por segundo – podendo executar alguns

Digitalizado com CamScanner

milhões de linhas de código nesse mesmo intervalo de tempo. Isso significa que, enquanto seu dedo pressiona o botão, o Arduino lê a posição do dispositivo milhares de vezes e altera state de acordo. Desse modo, os resultados acabam sendo imprevisíveis: talvez o LED fique aceso mesmo quando o quisermos apagado e vice-versa. Assim como podemos dizer que até um relógio quebrado marca a hora certa duas vezes ao dia, nosso programa também pode produzir o comportamento desejado de vez em quando, mas, na maioria das vezes, ele estará errado.

Como corrigimos esse problema? Bem, teremos de detectar o momento exato em que o botão é pressionado – o único momento em que temos de alterar o estado. Para tanto, acho interessante armazenar o valor atual de val antes de ler um novo valor; isso me permite comparar a posição atual do botão com a anterior e alterar seu estado apenas quando o botão se tornar HIGH depois de ter sido LOW.

O exemplo 03B apresenta o código para isso:

```
// Exemplo 03B: Acende o LED quando o botão é pressionado,
// mantendo-o aceso mesmo depois que deixamos de pressionar o botão.
// Agora com uma nova implementação!

const int LED = 13;     // pino para o LED
const int BUTTON = 7;   // pino de entrada ao qual o
                        // botão de pressão está conectado
int val = 0;            // val será utilizada para armazenar o estado
                        // do pino de entrada
int old_val = 0;    // esta variável armazena o valor anterior de
"val"
int state = 0;          // 0 = LED apagado e 1 = LED aceso

void setup() {
    pinMode(LED, OUTPUT);      // diz ao Arduino que LED é uma saída
    pinMode(BUTTON, INPUT);    // e que BUTTON é uma entrada
}
```

Digitalizado com CamScanner

```
void loop() {
    val = digitalRead(BUTTON); // lê e armazena o valor da entrada

    // verifica se houve uma transição
    if ((val == HIGH) && (old_val == LOW)) {
        state = 1 - state;
    }

    old_val = val;     // val agora está velha, vamos armazená-la

    if (state == 1) {
        digitalWrite(LED, HIGH); // acende o LED
    } else {
        digitalWrite(LED, LOW);
    }
}
```

Faça um teste. Estamos quase lá!

Talvez você tenha percebido que essa abordagem não é infalível, graças a outro problema inerente aos interruptores mecânicos. Botões de pressão são dispositivos muito simples: apenas duas peças de metal, separadas por uma mola. Quando você pressiona o botão, os dois contatos se encontram e a eletricidade pode fluir. Não parece haver problema nisso, mas, na vida real, essa conexão não é tão perfeita, especialmente quando o botão não é pressionado até o fim. Nesses casos, o dispositivo pode gerar alguns sinais imprecisos que chamamos de *repiques* (*bouncing*).

Quando o botão está nesse estado, o Arduino recebe uma sequência muito rápida de sinais que alternam entre ligado e desligado. Há muitas técnicas desenvolvidas para efetuar o *de-bouncing*, mas, em nosso caso, notei que geralmente basta adicionar um intervalo de 10 a 50 milissegundos quando o código detecta uma transição.

O exemplo 03C representa o código final:

Digitalizado com CamScanner

```
// Exemplo 03C: Acende o LED quando o botão é pressionado,
// mantendo-o aceso mesmo depois que deixamos de pressionar o botão.
// Inclui recurso simples de de-bouncing.
// Agora com mais uma nova implementação!!
const int LED = 13;     // pino para o LED
const int BUTTON = 7;   // pino de entrada ao qual o
                        // botão de pressão está conectado
int val = 0;            // val será utilizada para armazenar o estado
                        // do pino de entrada
int old_val = 0;        // esta variável armazena o valor
                        // anterior de "val"
int state = 0;          // 0 = LED apagado e 1 = LED aceso

void setup() {
    pinMode(LED, OUTPUT);      // diz ao Arduino que LED é uma saída
    pinMode(BUTTON, INPUT);    // e que BUTTON é uma entrada
}
void loop() {
    val = digitalRead(BUTTON); // lê e armazena o valor da entrada

    // verifica se houve uma transição
    if ((val == HIGH) && (old_val == LOW)) {
        state = 1 - state;
    delay(10);
    }
    old_val = val;     // val agora está velha, vamos armazená-la
    if (state == 1) {
        digitalWrite(LED, HIGH);   // acende o LED
    } else {
        digitalWrite(LED, LOW);
    }
}
```

Digitalizado com CamScanner

CAPÍTULO 5

Implementações avançadas de entradas e saídas

O que você acabou de aprender no capítulo 4 são as operações mais simples que podem ser feitas no Arduino: o controle de saídas e a leitura de entradas digitais. Se o Arduino fosse uma linguagem humana, essas seriam duas letras de seu alfabeto. Considerando que há apenas cinco letras nesse alfabeto, percebemos quanto trabalho ainda nos falta até que possamos escrever poesias com o Arduino.

Experiências com sensores de ligado/desligado

Agora que você aprendeu a utilizar um botão de pressão, saiba que há muitos outros sensores básicos que obedecem aos mesmos princípios:

Interruptores

Muito semelhantes a botões de pressão, mas que não alteram automaticamente de estado quando soltos.

Termostatos

Uma chave que se abre quando a temperatura atinge determinado valor.

Interruptores magnéticos (também conhecidos como "relés reed")

Dispositivos com dois contatos que se tocam quando próximos de um imã; utilizados em alarmes contra roubos para detectar a abertura de janelas, por exemplo.

Digitalizado com CamScanner

Interruptores de tapete

Pequenos dispositivos que você pode colocar sob um tapete para detectar a presença de um ser humano (ou de um gato pesado).

Interruptores de inclinação

Componente eletrônico simples que apresenta dois contatos e uma pequena bola de metal (ou uma gota de mercúrio, mas não me recordo de utilizar um desse tipo). Um exemplo de interruptor de inclinação é o que chamamos de sensor de inclinação. A figura 5.1 mostra o interior de um dispositivo desse tipo. Quando o sensor está em sua posição direita, a bola une os dois contatos, funcionando como se você tivesse pressionado um botão de pressão. Quando você inclina o sensor, a bola se move, abrindo o contato, como se você tivesse deixado de pressionar o botão. Utilizando esse simples componente você pode, por exemplo, implementar interfaces gestuais que reagem quando um objeto é movido ou chacoalhado (*bit.ly/ArduinoStoreTiltSensor*).

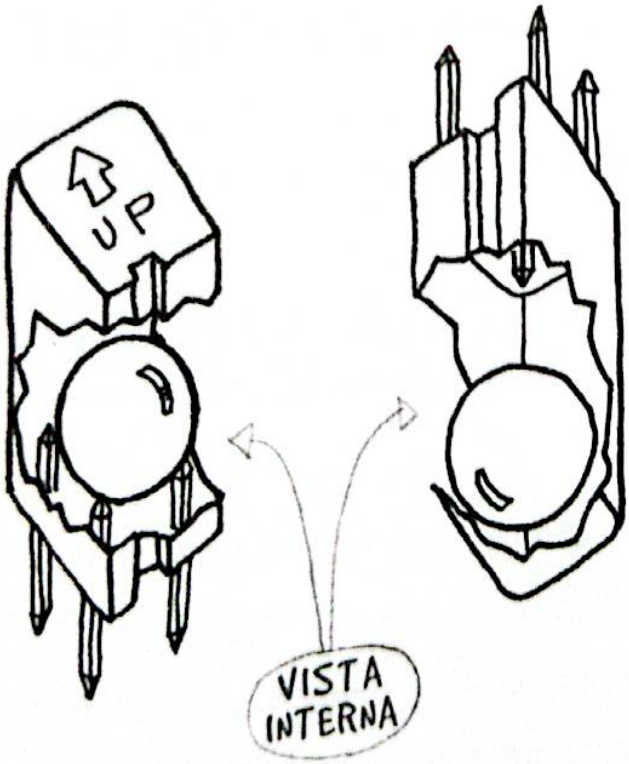


Figura 5.1 – Interior de um sensor de inclinação.

Digitalizado com CamScanner

Outro sensor que você pode experimentar é o sensor infravermelho utilizado em alarmes contra roubos (também conhecido como sensor infravermelho passivo, ou sensor PIR, e que pode ser visto na figura 5.2). Esse pequeno dispositivo dispara quando um ser humano (ou outra criatura) se move em sua proximidade. Trata-se de uma forma muito simples de detectar movimentos.

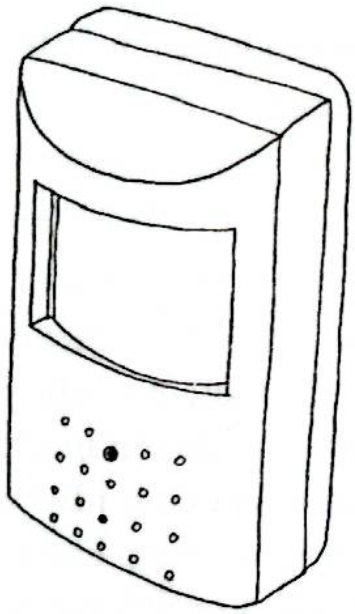


Figura 5.2 – Sensor PIR típico.

Seria interessante que você agora fizesse experiências com todo tipo de dispositivo que apresenta dois contatos próximos, como o termostato que define a temperatura de um cômodo (utilize um que não esteja mais conectado), ou simplesmente aproximando dois contatos e despejando água sobre eles.

Por exemplo, utilizando o exemplo final do capítulo 4 e um sensor PIR, você poderia fazer com que sua lâmpada respondesse à presença de seres humanos, ou ainda criar uma lâmpada, utilizando um sensor de inclinação, que se apaga quando inclinada.

Digitalizado com CamScanner

Controle de luz com PWM

Com o conhecimento que apresentamos até aqui, já somos capazes de montar e controlar uma lâmpada interativa – não apenas com um interruptor de ligado/desligado, mas talvez uma que seja um pouco mais elegante. Uma das limitações dos exemplos de LED pisca-pisca que utilizamos até agora é o fato de que nos foi permitido apenas acender ou apagar a luz. Uma lâmpada interativa mais elaborada deve apresentar também efeitos que regulem sua luminosidade gradualmente. Para solucionar esse problema, utilizaremos um pequeno truque que possibilita a existência de coisas como a TV e o cinema: a persistência da visão.

Como mencionei no primeiro exemplo do capítulo 4, se você alterar a duração dos intervalos entre os estados da lâmpada até que não perceba mais que ela está piscando, notará que o LED parece ter apenas 50% de seu brilho normal. Agora, altere esses números fazendo com que o LED esteja aceso um quarto do tempo e apagado pelo restante. Execute o sketch e você verá que o brilho corresponde a praticamente 25% do normal. Essa técnica é conhecida como modulação por largura de pulso (*pulse width modulation*, ou PWM), uma forma elaborada de dizer que se você piscar o LED rápido o bastante não mais o verá piscar, mas alterará seu brilho modificando a razão entre o tempo em que ele está aceso e o tempo em que ele está apagado. A figura 5.3 mostra como isso funciona.

Essa técnica também funciona em outros dispositivos além de LEDs. Você pode, por exemplo, alterar a velocidade de um motor utilizando essa mesma técnica.

Ao realizar seus experimentos, você perceberá que fazer o LED piscar implementando intervalos de tempo em seu código é um pouco inconveniente. Sempre que você quiser ler um sensor, ou enviar dados pela porta serial, fará o LED piscar enquanto aguarda o término da operação. Felizmente, o processador utilizado pela

Digitalizado com CamScanner

placa Arduino apresenta um componente de hardware que pode facilmente piscar três LEDs enquanto seu sketch faz algo diferente. Esse hardware é implementado nos pinos 9, 10 e 11, que podem ser controlados pela instrução `analogWrite()`.

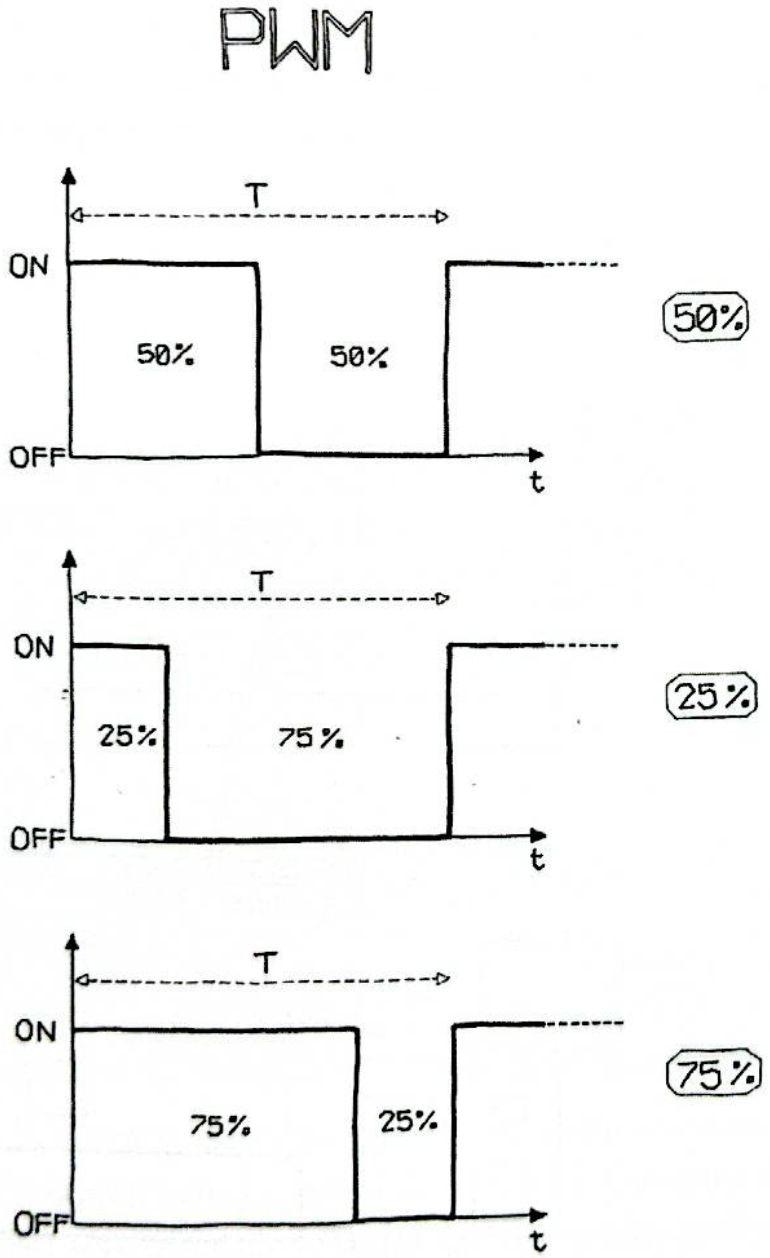


Figura 5.3 – PWM em ação.

Como exemplo, se você escrever `analogWrite(9,128)`, ajustará o brilho de um LED conectado ao pino 9 em 50%. Por que 128? `analogWrite()` espera um número entre 0 e 255 como argumento; 255 representa brilho máximo e 0, apagado.

Digitalizado com CamScanner

NOTA: É muito conveniente que tenhamos três canais à nossa disposição. Dessa forma, se você comprar três LEDs, um vermelho, um verde e outro azul, poderá misturar suas luzes e produzir as cores que quiser!

Vamos fazer um teste. Monte o circuito da figura 5.4. Note que os LEDs são polarizados: o pino longo (positivo) deve ir para a direita e o pino curto (negativo) para a esquerda. Da mesma forma, perceba que a maioria dos LEDs tem um lado negativo achatado, como mostra figura. Utilize um resistor de 270 ohms (vermelho, violeta e marrom).

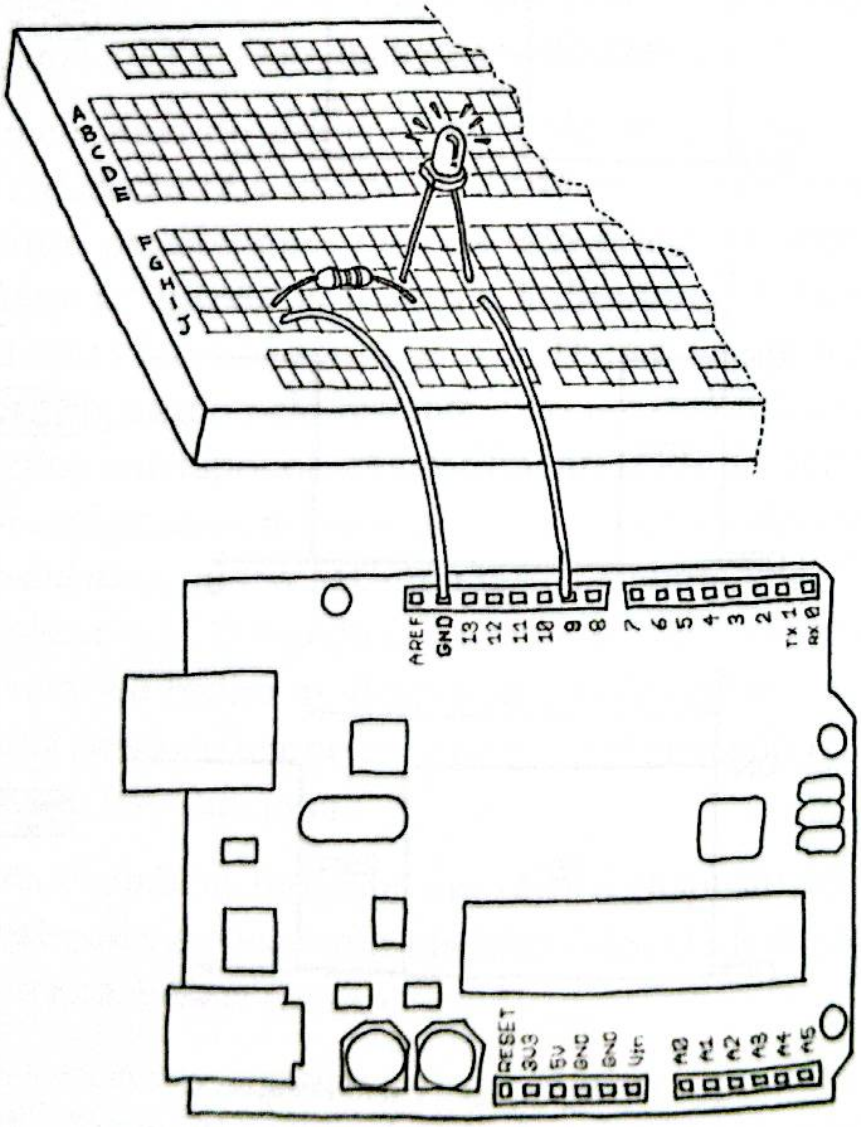


Figura 5.4 – LED conectado ao pino PWM.

Depois, crie um novo sketch no Arduino e utilize o exemplo 04 (você também pode fazer o download dos exemplos de código no site *www.makezine.com/getstartedarduino*):

Digitalizado com CamScanner

```
// Exemplo 04: Acenda e apague um LED gradualmente, produzindo um
// efeito como aquele que vemos em um computador Apple em suspensão

const int LED = 9;      // pino para o LED
int i = 0;              // Utilizaremos essa variável para contagem

void setup() {
    pinMode(LED, OUTPUT);  // diz ao Arduino que LED é uma saída
}
void loop() {

    for (i = 0; i < 255; i++) {    // faz um loop loop de 0 a 254 (acende
                                   // gradualmente)
        analogWrite(LED, i);       // define o brilho do LED
        delay(10);      // Espera 10ms, pois analogWrite  é
                        // instantâneo e não veríamos nenhuma alteração
    }

    for (i = 255; i > 0; i--) {    // faz um loop de 255 a 1 (apaga
                                   // gradualmente)

        analogWrite(LED, i);   // define o brilho do LED
        delay(10);             // Aguarda 10ms
    }
}
```

Com isso, acabamos de reproduzir uma característica de um computador laptop (ainda que utilizar o Arduino para algo tão simples talvez seja um desperdício). Agora, vamos utilizar o que aprendemos para aprimorar nossa lâmpada.

Primeiro, adicione à protoboard o circuito que utilizamos na leitura de um botão (Capítulo 4). Veja se consegue fazê-lo sem consultar a próxima página. Minha intenção é que você considere todos os circuitos elementares apresentados neste livro como elementos a serem utilizados na criação de projetos cada vez

Digitalizado com CamScanner

maiores. Ainda assim, mesmo se você tiver de consultar o texto, não se preocupe; o mais importante é que você ao menos pense em qual poderia ser o resultado.

Para criar esse projeto, você deverá combinar o circuito que acabou de montar (Figura 5.4) com o circuito do botão de pressão que vimos na figura 4.6. Se preferir, simplesmente monte os dois circuitos em partes diferentes da protoboard; há espaço para isso. Uma das vantagens da protoboard (Apêndice A) é que ela oferece um par de barramentos, distribuídos horizontalmente, tanto em sua base quanto em sua parte superior. Um é marcado em vermelho (para positivo) e outro, em azul ou preto (para o terra).

Esses barramentos são utilizados para distribuir alimentação e acesso ao terra, de acordo com suas necessidades. No caso de circuito que você tem de montar para esse exemplo, temos dois componentes (ambos resistores) que devem ser conectados ao pino GND (terra) do Arduino. Como o Arduino tem três pinos GND, você poderia simplesmente conectar esses dois circuitos exatamente como mostramos nas duas figuras; para isso, apenas conecte os dois circuitos ao Arduino simultaneamente. Como alternativa, você poderia conectar um fio do terra da protoboard a um dos pinos GND do Arduino e depois pegar os fios conectados ao GND nas figuras e ligá-los ao barramento do terra da protoboard.

Se você ainda não estiver pronto para tentar algo desse tipo, não se preocupe: simplesmente conecte ambos os circuitos ao Arduino, como nas figuras 4.6 e 5.4. No capítulo 6, veremos um exemplo que utiliza os barramentos do terra e da alimentação.

Voltando ao nosso exemplo, se você tem apenas um botão de pressão, como pode controlar o brilho da lâmpada? Nesse projeto, vamos aprender mais uma técnica de interação de design: o que fazer para detectar quanto tempo um botão permanece pressionado. Para isso, devemos aprimorar o exemplo 03C do capítulo 4, agora oferecendo uma forma de controlar o brilho da lâmpada. A

Digitalizado com CamScanner

ideia é montar uma "interface" na qual, quando pressionamos e soltamos o botão, a luz acende e apaga e, quando pressionamos e seguramos o botão, o brilho da lâmpada se altera.

Veja o sketch para esse projeto:

```
// Exemplo 05: Acende o LED quando o botão é pressionado,
// mantendo-o aceso mesmo depois que deixamos de pressionar o botão.
// Inclui recurso simples de de-bouncing.
// Se segurarmos o botão, alteraremos o brilho.


const int LED = 9;      // pino para o LED
const int BUTTON = 7;   // pino de entrada para o botão de pressão

int val = 0;            // armazena o estado do pino de entrada

int old_val = 0;        // armazena o valor anterior de "val"
int state = 0;          // 0 = LED apagado, enquanto 1 = LED aceso

int brightness = 128;           // Armazena o valor do brilho
unsigned long startTime = 0;    // quando começamos a pressionar o botão?

void setup() {
    pinMode(LED, OUTPUT);       // diz ao Arduino que LED é uma saída
    pinMode(BUTTON, INPUT);     // e que BUTTON é uma entrada
}

void loop() {
    val = digitalRead(BUTTON); // lê e armazena o valor da entrada

    // verifica se houve uma transição
    if ((val == HIGH) && (old_val == LOW)) {

        state = 1 - state; // altera o estado, de apagado para aceso
                           // ou vice-versa
```

Digitalizado com CamScanner

```
        startTime = millis();  // millis() é o relógio do Arduino
                               // ela retorna quantos milissegundos
                               // passaram desde que a placa foi
                               // reinicializada.

        // (esta linha recorda quando o botão foi
        // pressionado pela última vez)
        delay(10);
    }
    // verifica se estamos segurando o botão pressionado
    if ((val == HIGH) && (old_val == HIGH)) {
        // Se o botão for segurado por mais de 500ms
        if (state == 1 && (millis() - startTime) > 500) {

            brightness++;  // aumenta o brilho em 1
            delay(10);     // intervalo para evitar que o brilho
                           // aumente rápido demais

            if (brightness > 255) {    // 255 é o brilho máximo
                brightness = 0;    // se passarmos de 255,
                                   // devemos retornar a 0
            }
        }
    }

    old_val = val; // val agora está velha, vamos aramazená-la

    if (state == 1) {
        analogWrite(LED, brightness);  // acende o LED utilizando o
                                       // nível atual de brilho
    } else {
        analogWrite(LED, 0);   // apaga o LED
    }
}
```

Digitalizado com CamScanner

Faça um teste. Como pode ver, nosso modelo de interação está tomando forma. Se você pressionar o botão e retirar seu dedo imediatamente, estará acendendo e apagando a lâmpada. Por outro lado, se segurar o botão pressionado, você alterará o brilho da lâmpada: basta soltar o botão quando atingir o brilho desejado.

Agora, vamos aprender a utilizar alguns sensores mais interessantes.

Uso de um sensor de luz, em vez de um botão de pressão

Nesta seção, vamos realizar um experimento muito interessante. Pegue um sensor de luz como o da figura 5.5. Sensores como esse podem ser comprados no Maker Shed (peça número JM169578), ou no seguinte endereço: *bit.ly/ArduinoStoreLDR*.

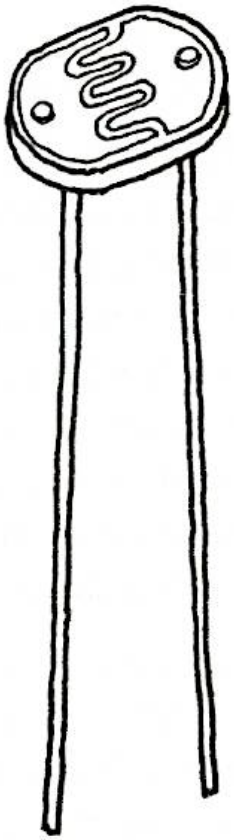


Figura 5.5 – Resistor dependente de luz (LDR).

Digitalizado com CamScanner

No escuro, a resistência de um resistor dependente de luz (LDR) é bem alta. Por outro lado, quando iluminado, sua resistência cai rapidamente, tornando-o um condutor razoável de eletricidade. Assim, podemos considerá-lo uma espécie de interruptor ativado por luz.

Monte o circuito do exemplo 02 (consulte a seção "Uso de um botão de pressão para controlar o LED", no capítulo 4) e faça o upload do código desse exemplo para seu Arduino.

Agora, em vez de ligar o LDR ao botão de pressão, conecte-o à protoboard. Note que, se você cobrir o LDR com suas mãos, o LED apagará. Descubra o LDR e a luz acenderá. Você acabou de montar seu primeiro LED verdadeiramente controlado por um sensor. Isso é importante, pois, pela primeira vez neste livro, estamos utilizando um componente eletrônico que não é apenas um simples dispositivo mecânico, mas um sensor rico em funcionalidades.

Entrada analógica

Como você aprendeu na seção anterior, o Arduino é capaz de detectar se há uma tensão aplicada a um de seus pinos e de informá-la por meio da função `digitalRead()`. Esse tipo de resposta é perfeito para muitas aplicações, mas o sensor de luz que acabamos de utilizar é capaz de nos informar não apenas se há luz, mas também quanta luz se faz presente. Essa é a diferença entre um sensor de ligado/desligado (que nos informa da presença de algo) e um sensor analógico, cujos valores se alteram continuamente. Para ler sensores como esse, necessitamos de um tipo diferente de pino.

Na parte inferior da placa Arduino, você verá seis pinos marcados como "Analog In"; são pinos especiais que podem nos informar não apenas se há uma tensão aplicada a eles, mas também seu valor. Utilizando a função `analogRead()`, podemos ler a tensão aplicada a um dos pinos. Essa função retorna um número entre 0 e 1023, representando voltagens entre 0 e 5 volts. Dessa forma, se tivermos uma voltagem de 2,5V aplicada ao pino 0, `analogRead(0)` retornará 512.

Digitalizado com CamScanner

Se você montar agora o circuito da figura 5.6 utilizando um resistor de 10k e executar o código listado no Exemplo 06A, verá que o LED da placa (você também poderia montar seu próprio LED nos pinos 13 e GND, como mostramos na seção "LED pisca-pisca", do capítulo 4) pisca a uma velocidade que depende da quantidade de luz que atinge o sensor.

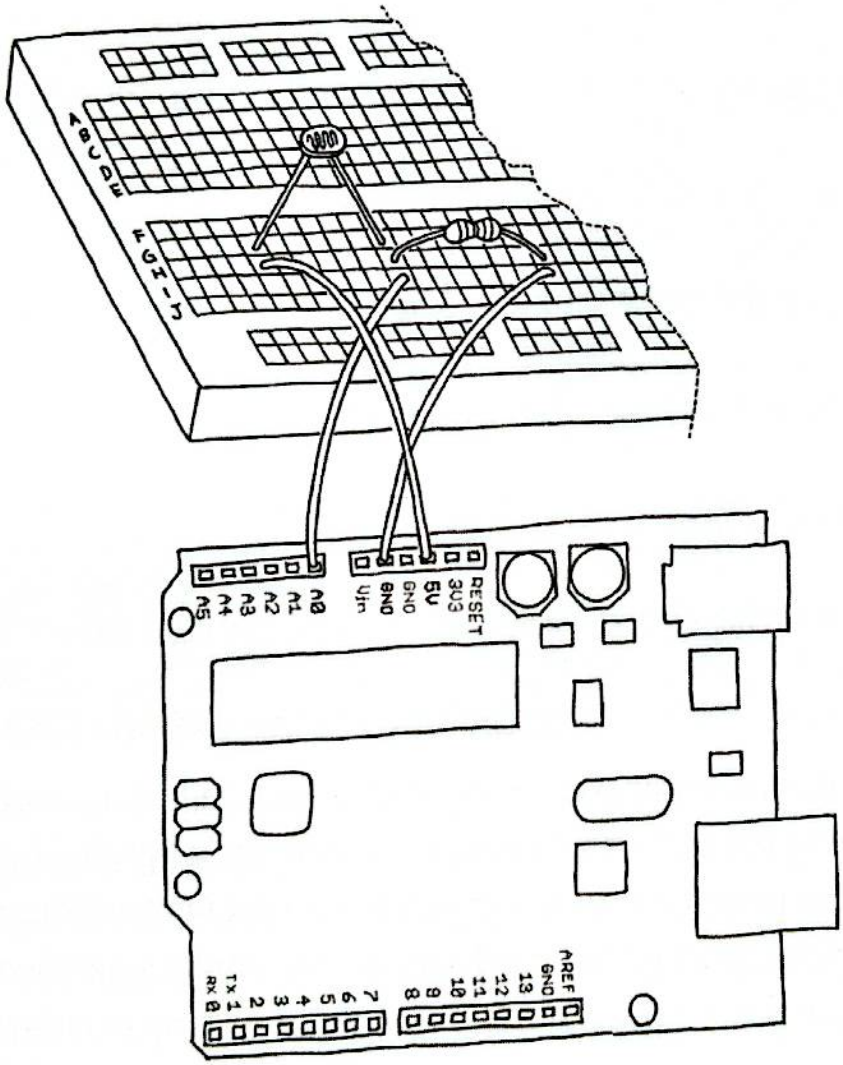


Figura 5.6 – Circuito de sensor analógico.

```
// Exemplo 06A: Pisca o LED a uma velocidade especificada
// pelo valor da entrada analógica

const int LED = 13;    // pino para o LED

int val = 0;           // variável utilizada para armazenar o valor
                       // que vem pelo sensor
```

Digitalizado com CamScanner

```
void setup() {
    pinMode(LED, OUTPUT);  // LED funciona como OUTPUT

    // Nota: pinos analógicos são automaticamente
    // definidos como entradas
}

void loop() {

    val = analogRead(0);   // lê o valor do sensor

    digitalWrite(LED, HIGH);   // acende o LED

    delay(val);    // interrompe o programa por algum tempo

    digitalWrite(LED, LOW);    // apaga o LED

    delay(val);    // interrompe o programa por algum tempo
}
```

Agora vamos testar o exemplo 06B. Antes de fazê-lo, você terá de modificar seu circuito. Consulte novamente a figura 5.4 e conecte o LED ao pino 9, como mostrado. Como você já tem componentes conectados à sua protoboard, terá de encontrar um espaço no qual o LED, os fios e o resistor não se sobreponham ao circuito do LDR.

```
// Exemplo 06B: Define o brilho do LED
// de acordo com o brilho especificado pelo
// valor da entrada analógica

const int LED = 9;     // pino para o LED

int val = 0;           // variável utilizada para armazenar o valor
                       // vindo do sensor
```

Digitalizado com CamScanner

```
void setup() {
    pinMode(LED, OUTPUT);  // LED funciona como OUTPUT

    // Nota: pinos analógicos são automaticamente
    // definidos como entradas
}

void loop() {
    val = analogRead(0);        // lê o valor do sensor
    analogWrite(LED, val/4);    // acende o LED com o brilho definido
                                // pelo sensor
    delay(10); // interrompe o programa por algum tempo
}
```

NOTA: Especificamos um brilho dividindo `val` por 4, uma vez que `analogRead()` retorna números até 1023 e `analogWrite()` aceita no máximo 255.

Experimente outros sensores analógicos

Utilizando o mesmo circuito que vimos na seção anterior, você pode conectar muitos outros sensores resistivos que funcionam praticamente da mesma forma. Por exemplo, você poderia conectar um termistor, dispositivo simples cuja resistência se altera com a temperatura. No circuito, mostrei como alterações de resistência se tornam alterações de voltagem que podem ser medidas pelo Arduino.

Se você estiver trabalhando com um termistor, saiba que não há uma conexão direta entre o valor lido e a temperatura medida em si. Se você necessita de uma leitura exata, é melhor utilizar os números enviados pelo pino analógico em medidas efetuadas por um termômetro real. Depois, você pode colocar esses números lado a lado em uma tabela e elaborar uma forma de calibrar os resultados analógicos de acordo com as temperaturas reais.

Digitalizado com CamScanner

Até aqui, utilizamos apenas um LED como dispositivo de saída, mas como podemos ler os valores em si que o Arduino está recebendo do sensor? Não podemos fazer a placa piscar os valores em código Morse (bem, até poderíamos, mas há uma forma mais fácil de permitirmos que humanos leiam esses dados). Para tanto, teremos de fazer o Arduino se comunicar com um computador utilizando uma porta serial. A próxima seção mostra como isso pode ser feito.

Comunicação serial

Você aprendeu no início deste livro que o Arduino tem uma conexão USB, utilizada pelo IDE ao fazer o upload do código para o processador. A boa notícia é que essa conexão também pode ser utilizada pelos sketches que escrevemos no Arduino para enviar dados de volta ao computador, ou para receber comandos emitidos por ele. Para tanto, vamos utilizar um objeto serial (um objeto é um agrupamento de capacidades, reunidas para facilitar o trabalho de quem está criando os sketches).

Esse objeto contém todo o código de que necessitamos para enviar e receber os dados. Vamos utilizar agora o último circuito que montamos com o fotoresistor e enviar os valores lidos de volta ao computador. Digite o código a seguir em um novo sketch (você também pode efetuar seu download em *www.makezine.com/getstartedarduino*):

```
// Exemplo 07: Envia ao computador os valores lidos na entrada
// analógica 0
// Não se esqueça de clicar em "Serial Monitor" depois do upload

const int SENSOR = 0;  // selecione o pino de entrada para
                       // o resistor do sensor

int val = 0;   // variável para armazenar o valor vindo do sensor
```

Digitalizado com CamScanner

```
void setup() {
    Serial.begin(9600); // abre a porta serial para enviar dados de
                        // volta ao computador a 9600 bits por seg.
}
void loop() {
    val = analogRead(SENSOR);  // lê o valor a partir do sensor
    Serial.println(val);   // envia o valor para a porta serial
    delay(100);    // espera 100ms entre cada envio
}
```

Depois de ter feito o upload do código para seu Arduino, pressione o botão "Serial Monitor" no IDE do Arduino (o último da barra de ferramentas); ao fazê-lo, você verá os números apresentados na base da janela. Agora, qualquer software capaz de efetuar a leitura da porta serial poderá se comunicar com o Arduino. Há muitas linguagens de programação que permitem a criação de programas capazes de se comunicar com a porta serial. A Processing (*www.processing.org*) é um ótimo complemento ao Arduino, em razão das semelhanças que apresenta com o IDE.

Projetos mais complexos (motores, lâmpadas e coisas do tipo)

Cada um dos pinos de uma placa Arduino pode ser utilizado para alimentar dispositivos que utilizam até 20 miliampères: essa é uma corrente muito pequena, suficiente apenas um LED. Se você tentar alimentar um motor, por exemplo, o pino deixará de funcionar imediatamente, podendo até queimar o processador inteiro. Para dispositivos que necessitam de correntes maiores, como motores e lâmpadas incandescentes, teremos de utilizar um componente externo, capaz de ligar e desligar tais equipamentos, e que seja controlado pelo Arduino. Um dispositivo desse tipo é um transistor MOSFET – ignore o nome curioso –, chave eletrônica que pode ser controlada aplicando-se uma tensão a um de seus três

Digitalizado com CamScanner

pinos, cada um chamado de *comporta* (*gate*). Seu funcionamento é semelhante ao dos interruptores de luz que temos em casa, no qual a ação do dedo acendendo e apagando a luz é substituída por um pino na placa Arduino que transmite a tensão à comporta do MOSFET.

NOTA: MOSFET significa "Metal Oxide Semiconductor Field Effect Transistor" (transistor de efeito de campo semicondutor de óxido de metal). Trata-se de um tipo de transistor que opera com base no princípio do efeito de campo. Isso significa que, assim que uma tensão for aplicada ao pino Gate, a eletricidade fluirá por um material semicondutor (entre os pinos Drain e Source). Como o Gate está isolado do restante por uma camada de óxido de metal, não há corrente do Arduino para o MOSFET, fazendo com que a interface seja muito simples. MOSFETs são ideais para ligar e desligar equipamentos que requerem mais energia a frequências elevadas.

Na figura 5.7, podemos ver como utilizaríamos um MOSFET, nesse caso o IRF520, para ligar e desligar um pequeno motor conectado a um ventilador. Note que o motor recebe sua alimentação a partir do conector de 9V da placa Arduino. Essa é outra vantagem do MOSFET: ele nos permite controlar dispositivos cuja alimentação difere daquela utilizada pelo Arduino. Como o MOSFET está conectado ao pino 9, podemos utilizar `analogWrite()` para controlar a velocidade do motor, utilizando PWM. Para montar o circuito, você necessitará de um MOSFET IRF520 (*bit.ly/ArduinoStoreIRF520*) e de um diodo 1N4007 (*bit.ly/ArduinoStore1N4007*). Se, durante o upload, o motor ligar aleatoriamente, coloque um resistor de 10K entre o pino 9 e o GND.

Digitalizado com CamScanner

Sensores complexos

Definimos sensores complexos como aqueles que produzem um tipo de informação cujo uso requer mais do que os recursos oferecidos pelas funções `digitalRead()` e `analogRead()`. Em geral, são pequenos circuitos, com microcontroladores que pré-processam informações. Alguns dos sensores complexos que temos à disposição incluem sensores de ultrassom, sensores infravermelhos e acelerômetros. Você pode encontrar mais exemplos do uso desses dispositivos em nosso site, na seção "Tutorials" (*www.arduino.cc/en/Tutorial/HomePage*).

O livro *Making Things Talk* (O'Reilly), de Tom Igoe, apresenta uma abordagem extensiva desses e de muitos outros sensores complexos.

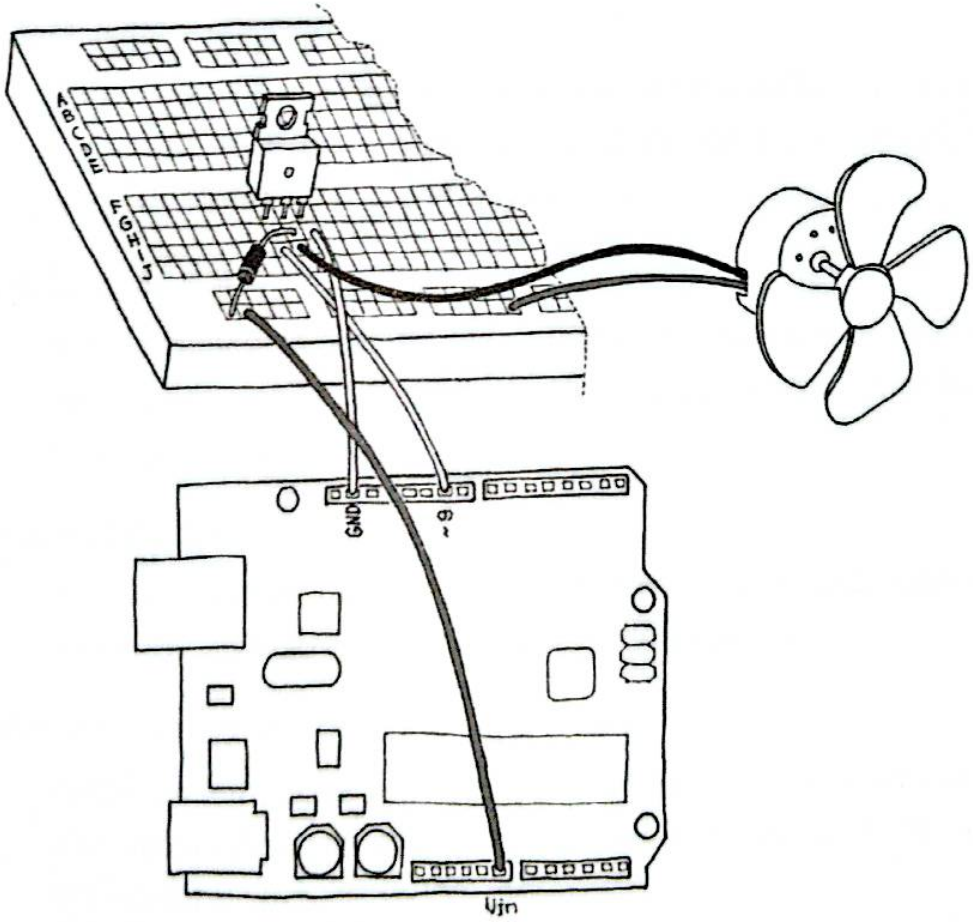


Figura 5.7 – Circuito de motor para o Arduino.

Digitalizado com CamScanner

CAPÍTULO 6

Comunicação em nuvem

Nos capítulos anteriores você aprendeu o básico sobre o Arduino e sobre os elementos fundamentais que estão à sua disposição. Vamos recordar os componentes do "alfabeto do Arduino":

Saída digital

Utilizamos a saída digital para controlar um LED, mas, com o circuito adequado, ela pode ser utilizada para controlar motores, emitir sons e muito mais.

Saída analógica

A saída analógica permite-nos controlar o brilho de um LED e não apenas acendê-lo e apagá-lo. Com ela, podemos controlar até a velocidade de um motor.

Entrada digital

Permite a leitura do estado de sensores mais simples, como botões de pressão ou interruptores de inclinação.

Entrada analógica

Permite a leitura de sensores que enviam um sinal contínuo, não apenas de ligado/desligado, como um potenciômetro ou um sensor de luz.

Digitalizado com CamScanner

Comunicação serial

Permite a comunicação com um computador e a transferência de dados, ou simplesmente o monitoramento daquilo que ocorre no sketch executado pelo Arduino.

Neste capítulo, veremos como podemos criar algo utilizando o que aprendemos nos capítulos anteriores. Nesse processo, você verá como cada exemplo que vimos pode ser utilizado no desenvolvimento de um projeto mais complexo.

É aqui que o aprendiz de designer que tenho dentro de mim vem à tona. Vamos criar uma versão século XXI de uma luminária criada pelo meu designer italiano favorito, Joe Colombo. O objeto que criaremos foi inspirado pela luminária "Aton", de 1964.

A luminária (Figura 6.1) é uma simples esfera apoiada sobre uma base que apresenta uma abertura, impedindo que ela role pela mesa. Esse design permite que você oriente a luminária em diferentes direções.

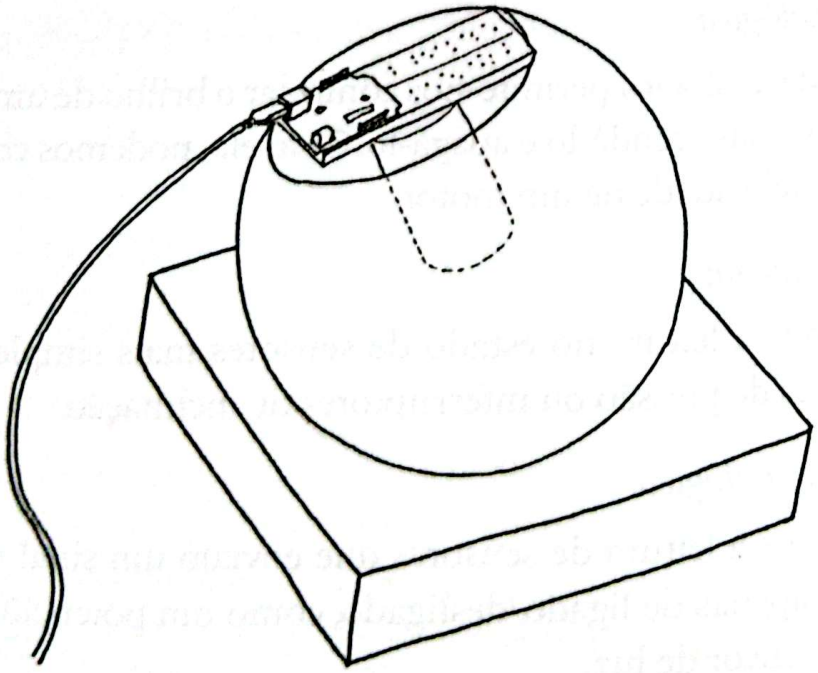


Figura 6.1 – A luminária, depois de concluída.

Digitalizado com CamScanner

Em termos de funcionalidade, queremos criar um dispositivo capaz de se conectar à Internet, buscar uma lista com os artigos mais recentes no blog da Make (*blog.makezine.com*) e contar quantas vezes as palavras "peace" (paz), "love" (amor) e "Arduino" são mencionadas. Partindo desses valores, geraremos uma cor a ser utilizada pela lâmpada. A luminária em si tem um botão que podemos utilizar para acendê-la e apagá-la e um sensor de luz para ativação automática.

Planejamento

Vamos primeiro analisar o resultado que pretendemos obter e as peças e equipamentos necessários para tanto. Antes de tudo, o Arduino deverá ser capaz de se conectar à Internet. Como a placa Arduino tem apenas uma porta USB, não podemos ligá-la diretamente a uma conexão da Internet, por isso teremos de descobrir uma forma alternativa de fazê-lo. Geralmente, as pessoas utilizam um aplicativo para se conectar à Internet, processar os dados e enviar ao Arduino informações mais simples, já processadas.

O Arduino é um computador simples com uma memória pequena. Justamente por isso, não pode processar arquivos muito grandes com facilidade. Se conectarmos o Arduino a um feed RSS, receberemos um arquivo XML muito extenso, que demandará muito mais RAM para ser processado. Assim, implementaremos um *proxy* para simplificar o XML utilizando a linguagem Processing.

Processing

A Processing é a base do Arduino. Adoramos essa linguagem e gostamos de utilizá-la para ensinar programação a iniciantes e criar programas eficientes. A Processing e o Arduino são uma combinação perfeita. Além disso, ela é desenvolvida como um projeto de código aberto, podendo ser executada em todas as principais plataformas (Mac, Linux e Windows). Também é capaz de gerar aplicativos independentes que poderão ser uti-

Digitalizado com CamScanner

lizados em todas essas plataformas e oferece uma comunidade dinâmica e prestativa de usuários, na qual você pode encontrar milhares de programas prontos para uso.

O proxy nos auxilia da seguinte maneira: ele efetua o download do feed RSS no site *makezine.com* e extrai todas as palavras do arquivo XML resultante. Depois, analisando todas elas, conta o número de vezes que "peace", "love" e "Arduino" surgem no texto. Com esses três números, podemos calcular um valor de cor e enviá-lo ao Arduino. A placa enviará de volta a quantidade de luz medida pelo sensor, mostrando-a na tela do computador.

No lado do hardware, combinaremos o exemplo do botão de pressão, do sensor de luz, do LED com PWM (multiplicado por 3!) e o recurso de comunicação serial.

Como o Arduino é um dispositivo simples, teremos de codificar a cor de modo simplificado. Utilizaremos a forma padrão de representação de cores em HTML: # seguido por seis dígitos hexadecimais.

Números hexadecimais são muito úteis, pois cada número de 8 bits é armazenado em exatamente dois caracteres; com números decimais, isso varia de um a três caracteres. A previsibilidade também torna o código mais simples: aguardamos até encontrar um #, depois lemos os seis caracteres seguintes em um *buffer* (variável utilizada como armazenamento temporário para dados). Finalmente, transformamos cada grupo de dois caracteres em um byte, representando o brilho de um dos três LEDs.

Código-fonte

Dois sketches serão executados neste projeto: um sketch da Processing e um sketch do Arduino. Aqui temos o código-fonte para a Processing; você pode efetuar seu download no site *www.makezine.com/getstartedarduino*.

Digitalizado com CamScanner

```
// Exemplo 08A: Arduino com luminária em rede
// Elementos deste código foram inspirados por
// uma postagem no blog de Tod E. Kurt (todbot.com)

import processing.serial.*;

String feed = "http://blog.makezine.com/index.xml";

int interval = 10; // recupera o feed a cada 60 segundos;
int lastTime;      // indica a última vez que buscamos o conteúdo

int love = 0;
int peace = 0;
int arduino = 0;

int light = 0;     // nível de luminosidade medido pela luminária

Serial port;
color c;
String cs;

String buffer = "";    // acumula caracteres vindos do Arduino

PFont font;

void setup() {
    size(640,480);
    frameRate(10);     // não necessitamos de atualizações rápidas

    font = loadFont("HelveticaNeue-Bold-32.vlw");
    fill(255);
    textFont(font, 32);

    // NOTA IMPORTANTE:
    // A primeira porta serial recuperada por Serial.list()
    // deve ser seu Arduino. Do contrário, remova o comentário da
```

Digitalizado com CamScanner

```
    // próxima linha, deletando as // antes dele, e execute novamente
    // o sketch para ver uma lista de portas seriais. Depois,
    // substitua o 0 entre [ e ] pelo número da porta à qual
    // seu Arduino está conectado.
    // println(Serial.list());
    String arduinoPort = Serial.list()[0];
    port = new Serial(this, arduinoPort, 9600);   // conecta-se ao
                                                  // Arduino

    lastTime = 0;
    fetchData();
}

void draw() {
    background( c );
    int n = (interval - ((millis()-lastTime)/1000));

    // Cria uma cor, com base nos 3 valores
    c = color(peace, love, arduino);
    cs = "#" + hex(c, 6);  // prepara uma string que será enviada
                           // ao Arduino

    text("Arduino Networked Lamp", 10, 40);
    text("Reading feed:", 10, 100); text(feed, 10, 140);
    text("Next update in "+ n + " seconds",10,450);
    text("peace", 10, 200);
    text(" " + peace, 130, 200);
    rect(200, 172, peace, 28);

    text("love ", 10, 240);
    text(" " + love, 130, 240);
    rect(200, 212, love, 28);

    text("arduino ", 10, 280);
    text(" " + arduino, 130, 280);
```

Digitalizado com CamScanner

```
rect(200, 252, arduino, 28);
// escreve a string da cor na tela
text("sending", 10, 340);
text(cs, 200, 340);
text("light level", 10, 380);
rect(200, 352, light/10.23, 28);  // transforma 1023 em 100

if (n <= 0) {
    fetchData();
    lastTime = millis();
}

port.write(cs); // envia os dados ao Arduino

if (port.available() > 0) {    // verifica se há dados em espera
    int inByte = port.read();  // lê um byte
    if (inByte != 10) {        // se byte não for uma nova linha
        buffer = buffer + char(inByte);    // apenas adiciona
                                           // esse byte ao buffer

    }
    else {

        // chegamos a uma nova linha, por isso vamos processar
        // os dados
        if (buffer.length() > 1) { // certifica-se de que há dados
                                   // suficientes

            // remove o último caractere, um retorno de carro
            // (caractere ao final de uma linha de texto)
            buffer = buffer.substring(0,buffer.length() -1);

            // transforma o buffer, de string em número inteiro
            light = int(buffer);
```

Digitalizado com CamScanner

```
                // limpa o buffer para o próximo ciclo de leitura
                buffer = "";

                // Provavelmente estamos atrasados na leitura de
                // dados do Arduino. Vamos limpar o backlog de
                // leituras vindas do sensor para que a próxima
                // leitura esteja atualizada.
                port.clear();
            }
        }
    }

void fetchData() {
    // utilizamos essas strings para efetuar o parsing do feed
    String data;
    String chunk;

    // vamos zerar os contadores
    love = 0;
    peace = 0;
    arduino = 0;
    try {
        URL url = new URL(feed);   // um objeto para representar o URL

        // vamos preparar a conexão
        URLConnection conn = url.openConnection();
        conn.connect();    // agora, conecta-se ao site

        // Nesta seção conectamos os dados a um leitor
        // em buffer, que lerá uma linha de cada vez
        BufferedReader in = new BufferedReader(new
            InputStreamReader(conn.getInputStream()));

        // lê cada linha do feed
        while ((data = in.readLine()) != null) {

            StringTokenizer st =
                new StringTokenizer(data,"\"<>,.()[] ");// faz as
                                                        //separações
```

Digitalizado com CamScanner

```
        while (st.hasMoreTokens()) {
                // cada fragmento de dados é transformado em caixa baixa
                chunk= st.nextToken().toLowerCase() ;

                if (chunk.indexOf("love") >= 0) // encontrou "love"?
                    love++;          // eleva love em 1 unidade
                if (chunk.indexOf("peace") >= 0)// encontrou "peace"?
                    peace++;         // eleva peace em 1 unidade
                if (chunk.indexOf("arduino") >= 0)    // encontrou
                                                      // "arduino"?
                    arduino++;       // eleva arduino em 1 unidade
            }
        }

        // Define 64 como número máx de referências
        if (peace > 64)
            peace = 64;
        if (love > 64)
            love = 64;
        if (arduino > 64)
            arduino = 64;

        peace = peace * 4; // multiplica por 4 para que o máximo
                           // seja 255,
        love = love * 4;   // isso facilita quando estamos criando uma
        arduino = arduino * 4;     // cor formada por 4 bytes (ARGB)
    }
    catch (Exception ex) { // se houver um erro, interrompe o sketch
        ex.printStackTrace();
        System.out.println("ERROR: " + ex.getMessage());
    }
}
```

Digitalizado com CamScanner

Há duas coisas que você deve fazer para que o sketch da Processing seja executado corretamente. Primeiro você deve informar à linguagem que ela deve processar a fonte a ser utilizada no sketch. Para tanto, crie e salve esse sketch. Depois, mantendo-o aberto, clique no menu Tools da Processing e selecione Create Font. Selecione a fonte HelveticaNeue-Bold, escolha 32 como tamanho e clique em OK.

Depois, você terá de confirmar que o sketch está utilizando a porta serial correta para se comunicar com o Arduino. Para isso, é preciso aguardar até que o circuito tenha sido montado e o upload do sketch tenha sido feito. Na maioria dos sistemas, esse sketch da Processing poderá ser executado sem problemas. Entretanto, se nada acontecer no Arduino e você não identificar nenhuma informação na tela vinda do sensor de luz, localize o comentário que marcamos como "NOTA IMPORTANTE" no sketch e siga as instruções que fornecemos.

Veja agora o sketch do Arduino (também disponível em *www.makezine.com/getstartedarduino*):

```
// Exemplo 08B: Arduino com luminária em rede
const int SENSOR = 0;
const int R_LED = 9;
const int G_LED = 10;
const int B_LED = 11;
const int BUTTON = 12;

int val = 0;   // variável para armazenar o valor vindo do sensor

int btn = LOW;
int old_btn = LOW;
int state = 0;
char buffer[7] ;
int pointer = 0;
byte inByte = 0;
```

Digitalizado com CamScanner

```
byte r = 0;
byte g = 0;
byte b = 0;

void setup() {
    Serial.begin(9600);    // abre a porta serial
    pinMode(BUTTON, INPUT);
}

void loop() {
    val = analogRead(SENSOR);  // lê o valor do sensor
    Serial.println(val);       // imprime o valor para a porta serial

    if (Serial.available() > 0) {

        // lê o byte de entrada:
        inByte = Serial.read();

        // Se o marcador foi encontrado, os próximos 6 caracteres
        // representam a cor
        if (inByte == '#') {

            while (pointer < 6) {  // acumula 6 caracteres
                buffer[pointer] = Serial.read();   // armazena no
                                                   // buffer
                pointer++;  // move o ponteiro 1 caractere à frente
            }

        // agora que temos os 3 números armazenados como hexadecimais
        // devemos decodificá-los em 3 bytes, r, g e b
        r = hex2dec(buffer[1]) + hex2dec(buffer[0]) * 16;
        g = hex2dec(buffer[3]) + hex2dec(buffer[2]) * 16;
        b = hex2dec(buffer[5]) + hex2dec(buffer[4]) * 16;

        pointer = 0; // zeramos o ponteiro para reutilizar o buffer
    }
}
```

Digitalizado com CamScanner

```
  btn = digitalRead(BUTTON);      // lê e armazena o valor de entrada

  // Verifica se houve uma transição
  if ((btn == HIGH) && (old_btn == LOW)) {
      state = 1 - state;
  }

  old_btn = btn; // val agora está velha, vamos armazená-la

  if (state == 1) {  // se a lâmpada estiver acesa

      analogWrite(R_LED, r);      // acende os LEDs
      analogWrite(G_LED, g);      // utilizando a cor
      analogWrite(B_LED, b); }    // enviada pelo computador
  } else {
      analogWrite(R_LED, 0);      // do contrário, apaga os LEDs
      analogWrite(G_LED, 0);
      analogWrite(B_LED, 0);
  }

  delay(100);             // aguarda 100ms entre cada envio
  }

  int hex2dec(byte c) {  // converte caractere hexadecimal em número
      if (c >= '0' && c <= '9') {
          return c - '0';
      } else if (c >= 'A' && c <= 'F') {
          return c - 'A' + 10;
      }
  }
```

Digitalizado com CamScanner

Montagem do circuito

A figura 6.2 mostra como montar esse circuito. Recomendamos que você utilize resistores de 10K para todos os resistores mostrados no diagrama, ainda que sejam permitidos valores menores para aqueles conectados aos LEDs.

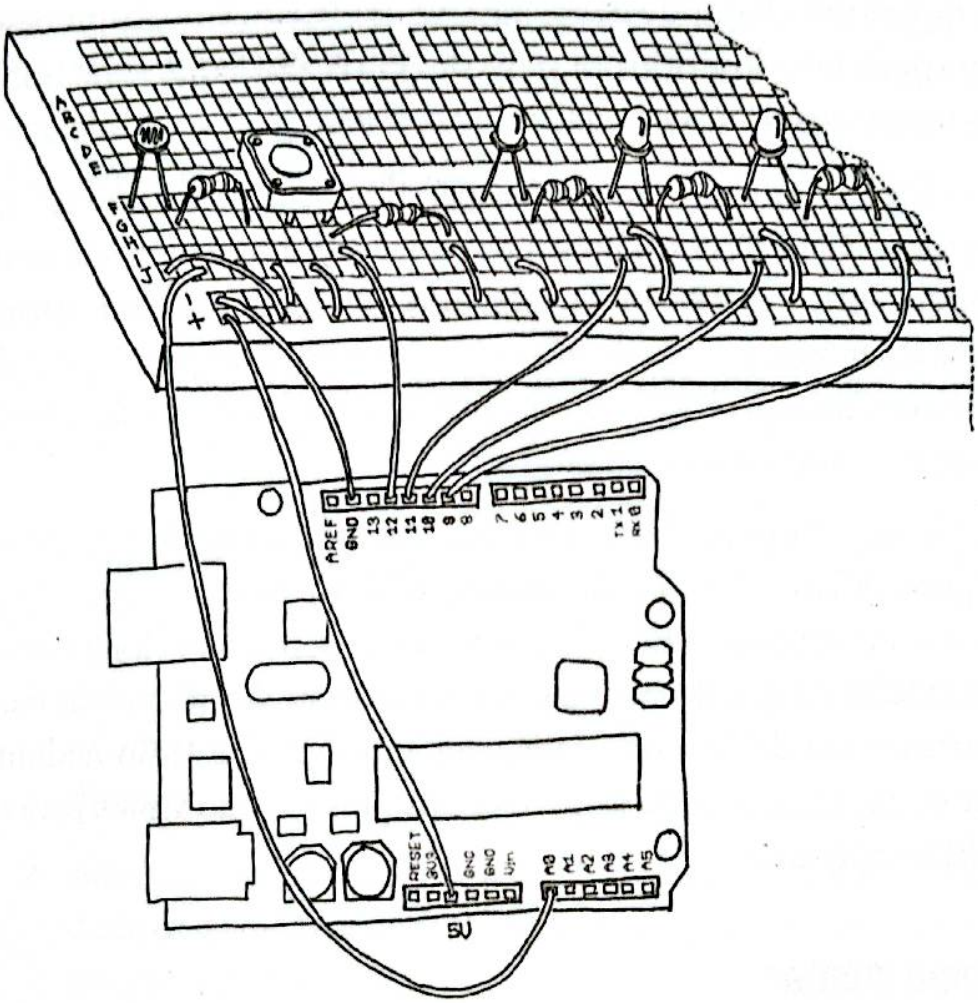


Figura 6.2 – Circuito do projeto "Arduino com luminária em rede".

Lembre-se de que, no exemplo da PWM visto no capítulo 5, mostramos que LEDs são polarizados: nesse circuito, o pino longo (positivo) deve ir para a direita e o pino curto (negativo), para a esquerda. (A maioria dos LEDs tem um lado negativo chanfrado, como mostra a figura).

Digitalizado com CamScanner

Monte o circuito utilizando um LED vermelho, um verde e outro azul. Depois, carregue os sketches no Arduino e no Processing e faça seus testes. Se você tiver qualquer problema, consulte o capítulo 7: "Resolução de problemas".

Agora, vamos completar nossa criação colocando a protoboard em uma esfera de vidro. A opção mais simples e econômica é comprar um abajur de mesa "FADO" da IKEA. Um abajur desse tipo pode ser adquirido por cerca de US$ 19,99/€ 14,99/£ 11,99 (ahh, os benefícios de se viver na Europa).

Em vez de utilizar três LEDs separados, você pode utilizar um único LED RGB, com quatro pernas. Você deverá conectá-lo da mesma forma que os LEDs que vimos na figura 6.2, com apenas uma diferença: em vez de três conexões separadas para o pino do terra do Arduino, você tem uma única perna (chamada de "cátodo comum") indo para o terra.

A Arduino Store vende um LED RGB de quatro pernas por apenas alguns dólares (*bit.ly/ArduinoStoreRGBLed*). Perceba apenas que, diferentemente de LEDs de cor única, a perna mais longa desse LED RGB é a que deve ser conectada ao terra. As três pernas mais curtas terão de ser conectadas aos pinos 9, 10 e 11 do Arduino (com um resistor entre as pernas e os pinos, como vimos para os LEDs separados).

Como montar

Desmonte a luminária e remova o cabo que passa para dentro dela a partir da base. Não há problema em fazê-lo, pois você não vai mais conectá-la à parede.

Conecte o Arduino a uma protoboard, colando esta à parte posterior da luminária.

Digitalizado com CamScanner

Solde fios mais compridos ao LED RGB e cole-os ao local onde antes tínhamos a lâmpada. Conecte os fios vindos do LED à protoboard (no local em que eles estavam conectados antes de serem removidos). Lembre-se de que você necessita de apenas uma conexão ao terra se estiver utilizando um LED RGB de 4 pernas.

Arrume um pedaço de madeira que possa servir de suporte para a luminária, ou simplesmente corte 5 cm da parte superior da caixa de papelão em que ela veio, criando uma abertura que possa abrigar a lâmpada. Reforce o interior da caixa utilizando cola quente em todas as arestas, para torná-la mais resistente.

Posicione a esfera no suporte e passe o cabo USB pela parte superior, conectando-o ao computador.

Ponha seu código Processing para funcionar, pressione o botão de acender/apagar e veja sua luminária ganhar vida.

Para praticar, tente implementar um código que faça a lâmpada acender quando o ambiente tiver pouca luz. Outros aprimoramentos possíveis são:

» Adicione sensores de inclinação que acendam e apaguem a lâmpada quando ela for movimentada em diferentes direções.

» Adicione um pequeno sensor PIR, que detecte a proximidade de alguém, e apague a lâmpada quando não houver ninguém por perto.

» Crie novas implementações para que você tenha controle manual da lâmpada, ou para que possa fazer com que ela apresente muitas cores diferentes.

Fique à vontade para pensar em outras aplicações. Experimente e divirta-se!

Digitalizado com CamScanner

CAPÍTULO 7

Resolução de problemas

Haverá um momento em seus experimentos em que nada dará certo, e você terá de descobrir como resolver seus problemas. Ainda que a resolução e depuração de problemas seja uma arte antiga e que conte com apenas algumas regras simples, a maioria dos resultados só será obtida com muito esforço e persistência.

Quanto mais você trabalha com componentes eletrônicos e com o Arduino, mais aprende e ganha experiência, o que acabará por tornar o processo de resolução de problemas menos doloroso. Não se assuste com os problemas que podem surgir – é mais fácil resolvê-los do que parece à primeira vista.

Uma vez que todo projeto com base no Arduino é formado por componentes de hardware e software, você terá de verificar esses ambientes se algo der errado. Ao procurar por um bug, siga estes três passos:

Entendimento

Tente compreender ao máximo como funcionam os componentes utilizados em seu projeto e como eles devem contribuir ao resultado final. Essa abordagem permitirá que você elabore uma forma de testar cada componente separadamente.

Simplificação e segmentação

Os antigos romanos diziam *divide et impera*: divida e conquiste. Tente separar (mentalmente) o projeto em seus menores

Digitalizado com CamScanner

componentes, utilizando aquilo que você já sabe para descobrir onde começa e termina a responsabilidade de cada componente.

Exclusão e certeza

Durante sua investigação, teste cada componente separadamente para que você possa estar absolutamente certo do funcionamento de cada um. Gradualmente, você deve perceber quais partes do projeto funcionam corretamente e quais apresentam um comportamento dúbio.

Depuração (*debugging*) é o termo que descreve esse processo quando falamos de software. Diz a lenda que essa expressão foi utilizada pela primeira vez por Grace Hopper, nos anos 1940, quando computadores ainda eram principalmente eletromagnéticos e um deles deixou de funcionar porque insetos (*bugs*, em inglês) entraram em seus mecanismos.

Muitos dos bugs de hoje não são mais físicos: mas antes virtuais e invisíveis, ao menos em parte. Problemas desse tipo requerem um processo por vezes lento e enfadonho para serem identificados.

Teste da placa

O que fazer se o primeiro exemplo, "LED pisca-pisca", não der certo? Isso seria um pouco decepcionante, não é mesmo? Vamos descobrir o que devemos fazer nesse caso.

Antes de culpar seu projeto, você deve garantir que certos elementos estão funcionando corretamente, da mesma forma que pilotos de avião conferem uma *checklist* antes da decolagem:

Conecte seu Arduino a um plug USB de seu computador.

» Certifique-se de que seu computador está ligado (mesmo que isso pareça óbvio, já aconteceu). Se a luz verde, marcada PWR, acender, isso significa que o computador está

Digitalizado com CamScanner

alimentando a placa. Se a luz do LED parecer fraca, algo deve estar errado com a energia: experimente um cabo USB diferente e inspecione a porta USB e o plug USB do Arduino para ver se há algum dano aparente nesses componentes. Se nada der resultado, tente outra porta USB, ou mesmo um computador diferente.

» Se o Arduino for novo, o LED amarelo, marcado L, começará a piscar de forma errática. Esse é o programa de teste carregado na fábrica para testar a placa.

» Se você vem utilizando uma fonte de alimentação externa e está utilizando um Arduino mais antigo (Extreme, NG, ou Diecimila), certifique-se de que a alimentação está sendo fornecida e que o jumper SV1 está conectando os dois pinos mais próximos do conector da fonte de alimentação externa.

NOTA: Se você tiver problemas com os outros sketches e quiser confirmar se a placa está funcionando corretamente, abra o primeiro exemplo, "LED pisca-pisca", no IDE do Arduino e faça seu upload para a placa. O LED da placa deverá piscar acompanhando um padrão regular.

Se você percorreu todos esses passos e obteve um resultando satisfatório, pode estar certo de que seu Arduino está funcionando corretamente.

Teste do circuito da protoboard

Agora, conecte sua placa à protoboard ligando um jumper das conexões 5V e GND aos barramentos positivo e negativo da protoboard. Se o LED verde, PWR, apagar, remova os fios imediatamente. Isso significa que há um sério problema em seu circuito, indicando a presença de um "curto-circuito" em algum ponto do projeto. Nesses casos, sua placa receberá muita corrente e a alimentação será cortada para proteger o computador.

Digitalizado com CamScanner

NOTA: Se você está preocupado em danificar seu computador, lembre-se de que em muitos computadores a proteção de corrente funciona satisfatoriamente e responde com rapidez. Assim também, a placa Arduino utiliza um "PolyFuse", dispositivo protetor de corrente que reinicia quando a falha é removida. Se você for realmente paranóico, pode sempre conectar sua placa Arduino utilizando um hub USB de alimentação independente. Nesse caso, se tudo der errado, apenas o hub USB será danificado, e não seu computador.

Caso haja um curto-circuito, você terá de iniciar o processo de "simplificação e segmentação", percorrendo cada sensor do projeto e testando um de cada vez.

O primeiro componente que deve ser testado é sempre a fonte de alimentação (as conexões do 5V e do GND). Analise o circuito e certifique-se de que todas as partes estão conectadas corretamente.

Trabalhar passo a passo e efetuar apenas uma modificação de cada vez é a melhor forma de solucionar seus problemas. Ainda jovem, aprendi a importância dessa regra com meu professor e primeiro chefe, Maurizio Pirola. Toda vez que estou realizando o *debugging* de algo e as coisas simplesmente não dão certo (acredite, isso acontece bastante), lembro-me de Maurizio dizendo "uma modificação de cada vez... uma modificação de cada vez", e é nessa hora que geralmente consigo corrigir os problemas. Seguir esse princípio é muito importante, pois, ao fazê-lo, você saberá exatamente o que solucionou seu problema (é muito fácil simplesmente perder de vista qual modificação realmente resolveu o problema, por isso é importante efetuar apenas uma de cada vez).

Cada experiência de depuração poderá criar em sua mente uma "base de conhecimento" com defeitos e suas possíveis soluções. Antes de perceber, você terá se tornado um especialista no assunto. Isso fará com que você fique bem descolado: assim que um novato disser "Mas isso não funciona!", você terá a resposta na ponta da língua.

Digitalizado com CamScanner

Isolamento de problemas

Outra regra importante determina que devemos encontrar uma forma confiável de reproduzir um problema. Caso seu circuito se comporte de modo inesperado aleatoriamente, tente descobrir exatamente em que momento o problema ocorre e qual sua causa. Esse processo permitirá que você pense em uma causa possível para seus problemas, além de ser de grande utilidade quando você tiver de explicar a outra pessoa o que está acontecendo.

Descrever um problema da forma mais precisa possível também é um bom modo de encontrar uma solução. Tente encontrar alguém para quem você possa explicar seu problema – em muitos casos, uma solução surgirá em sua mente à medida que você explica sua situação. Brian W. Kernighan e Rob Pike, no livro *The Practice of Programming* (Addision-Wesley, 1999), contam a história de uma universidade que "tinha um urso de pelúcia próximo do balcão de atendimento. Alunos com bugs misteriosos tinham de explicá-los primeiro ao ursinho antes que pudessem falar com um dos conselheiros."

Problemas com o IDE

Em alguns casos você pode ter um problema com o IDE do Arduino, especialmente no Windows.

Se receber um erro quando clicar duas vezes no ícone do Arduino, ou se nada acontecer, tente clicar duas vezes no arquivo *run.bat*, como alternativa.

Usuários do Windows também podem ter problemas caso o sistema operacional atribua ao Arduino um número de porta igual ou maior a COM10. Se isso acontecer, você geralmente poderá convencer o Windows a utilizar um número menor. Primeiro, abra seu Gerenciador de Dispositivos clicando no menu **Iniciar**, depois clique com o botão direito em **Computador** (Vista) ou **Meu Computador**

Digitalizado com CamScanner

(XP) e escolha Propriedades. No Windows XP, clique em Hardware e escolha Gerenciador de Dispositivos. No Vista, simplesmente clique em Gerenciador de Dispositivos (na lista de tarefas à esquerda da janela).

Procure os dispositivos seriais na lista sob "Portas (COM e LPT)". Encontre um dispositivo serial não utilizado e que tenha número COM9 ou menor. Clique nele com o botão direito e escolha Propriedades. Depois, escolha a guia Configurações de Porta e clique em Avançado. Defina o número de porta como COM10 ou maior e depois clique em OK e OK novamente para sair do diálogo de propriedades.

Repita o processo, com apenas uma diferença, para o dispositivo de porta serial USB que representa o Arduino: atribua a ele o número de porta COM que você acabou de liberar (COM9 ou menor).

Se essas sugestões não resolverem seu problema, ou se você está tendo um problema que não descrevemos aqui, consulte a página de resolução de problemas do Arduino, em *www.arduino.cc/en/Guide/Troubleshooting*.

Como obter ajuda online

Se você está tendo dificuldades, não gaste dias procurando soluções sozinho – peça ajuda. Uma das maiores vantagens do Arduino é sua comunidade. Desde que seja capaz de descrever adequadamente seu problema, você poderá sempre contar com a ajuda de outros usuários.

Habitue-se a recortar e colar trechos de seu código em mecanismos de busca para descobrir se mais alguém está trabalhando em algo parecido. Por exemplo, quando o IDE do Arduino mostrar uma terrível mensagem de erro, copie e cole seu conteúdo no Google Search para descobrir qual o resultado. Faça o mesmo com trechos do código em que você está trabalhando, ou com um nome específico de função. Pesquise sempre: tudo que já foi inventado está armazenado em alguma página da web.

Digitalizado com CamScanner

Para mais informações, consulte o site principal do Arduino, em *www.arduino.cc*, e também o FAQ (*www.arduino.cc/en/Main/FAQ*); não deixe de consultar ainda o playground (*www.arduino.cc/playground*), wiki que pode ser modificado por qualquer usuário interessado em contribuir à documentação. Essa é uma das melhores características da filosofia do código aberto: as pessoas podem contribuir com documentação e exemplos de tudo que pode ser feito com o Arduino. Antes de iniciar um projeto, consulte o playground e procure códigos ou diagramas de circuito capazes de ajudá-lo.

Se mesmo assim você ainda não conseguir encontrar uma resposta, pesquise no fórum (*www.arduino.cc/cgi-bin/yabb2/YaBB.pl*). Se sua pesquisa não der resultado, poste uma pergunta. Escolha a placa correta para seu caso: há áreas diferentes para problemas de hardware e software e até fóruns em cinco idiomas diferentes. Não se esqueça: poste o máximo possível de informações:

» Que placa Arduino você está utilizando?

» Que sistema operacional você está utilizando para executar o IDE do Arduino?

» » Forneça uma descrição geral do que você está tentando fazer. Lembre-se de postar links para as especificações dos componentes menos usuais utilizados.

O número de respostas que você vai receber depende de sua capacidade em formular sua questão.

Suas chances de encontrar uma solução aumentam se você evitar, a todo custo, os comportamentos que descrevemos a seguir (note que essas regras são úteis para qualquer fórum online, e não apenas para o do Arduino):

» Não digite suas mensagens em MAIÚSCULAS. Utilizar apenas maiúsculas irrita as pessoas; é como conversar com usuários que têm a palavra "novato" tatuada em suas testas (em comunidades online, digitar em maiúsculas é considerado gritar).

Digitalizado com CamScanner

» Não poste a mesma mensagem em vários locais diferentes do fórum.

» Não poste comentários logo depois de sua mensagem, dizendo "Ei, por que ninguém respondeu à minha pergunta?", para que sua dúvida volte à primeira página do fórum. Se você não recebeu uma resposta, verifique sua mensagem. Será que você foi claro o bastante? Será que forneceu uma descrição bem elaborada do problema que está encontrando? Você foi educado? Sempre seja educado.

» Não escreva mensagens como "Quero criar um ônibus espacial utilizando o Arduino. Alguém sabe como?". Isso significa apenas que você deseja que outras pessoas façam seu trabalho por você. Essa abordagem não é nem um pouco divertida para um verdadeiro explorador. É melhor explicar o que você deseja criar e, depois, fazer apenas perguntas específicas sobre o projeto.

» Uma variação do ponto anterior ocorre quando a questão postada claramente se refere a algo que o autor está sendo pago para fazer. Se você fizer perguntas específicas, as pessoas terão todo interesse em ajudá-lo, mas, se pedir que façam seu trabalho por você (e não dividir o pagamento), é natural que as respostas se tornem menos agradáveis.

» Não poste mensagens que parecem projetos escolares, pedindo que o fórum faça seu dever de casa por você. Professores, como eu, visitam vários fóruns e estão sempre atentos à participação de alunos que fazem algo do tipo.

Digitalizado com CamScanner

APÊNDICE A

Protoboard

O processo necessário para que um circuito funcione envolve muitas alterações até que ele se comporte como pretendido; esse é um processo muito rápido e iterativo e pode ser considerado o equivalente eletrônico da elaboração de rascunhos. Nessa etapa, o design evolui em suas mãos à medida que você experimenta combinações diferentes. Para obter os melhores resultados, recomendamos que você utilize um sistema que permite a alteração das conexões do modo mais rápido, prático e não destrutivo possível. Esses requisitos evidentemente excluem a soldagem, procedimento demorado e que pode danificar componentes sempre que você os esquenta e esfria.

A resposta para esse problema é um dispositivo muito prático, que chamamos de protoboard sem solda. Como você pode ver na figura A.1, trata-se de uma pequena placa de plástico, repleta de furos, cada um contendo um contato. Você pode inserir o terminal de um componente em um desses furos, fazendo com que ele estabeleça uma conexão elétrica com todos os outros furinhos na mesma coluna vertical. Cada furo está a 2,54 mm dos outros.

Como a maioria dos componentes tem suas pernas (conhecidas também como "pinos") espaçadas nessa mesma distância, chips com várias pernas se encaixam perfeitamente. Nem todos os componentes de uma protoboard são criados da mesma forma – há algumas diferenças. As linhas do topo e da base (pintadas

Digitalizado com CamScanner

de vermelho e azul e marcadas com + e -) estão conectadas horizontalmente e são utilizadas para transmitir a alimentação à placa para que, quanto você necessitar de energia, ou do terra, possa obter esse contato muito rapidamente com um *jumper* (pequeno fio utilizado para conectar dois pontos nos circuitos). A última coisa que você tem de saber sobre protoboards é que, no meio delas, há um espaço vazio, da largura de um pequeno chip. Cada linha vertical de furos é interrompida nesse ponto, de modo que, ao conectar um chip, você não produza um curto-circuito dos pinos que se encontram em seus dois lados. Brilhante, não?

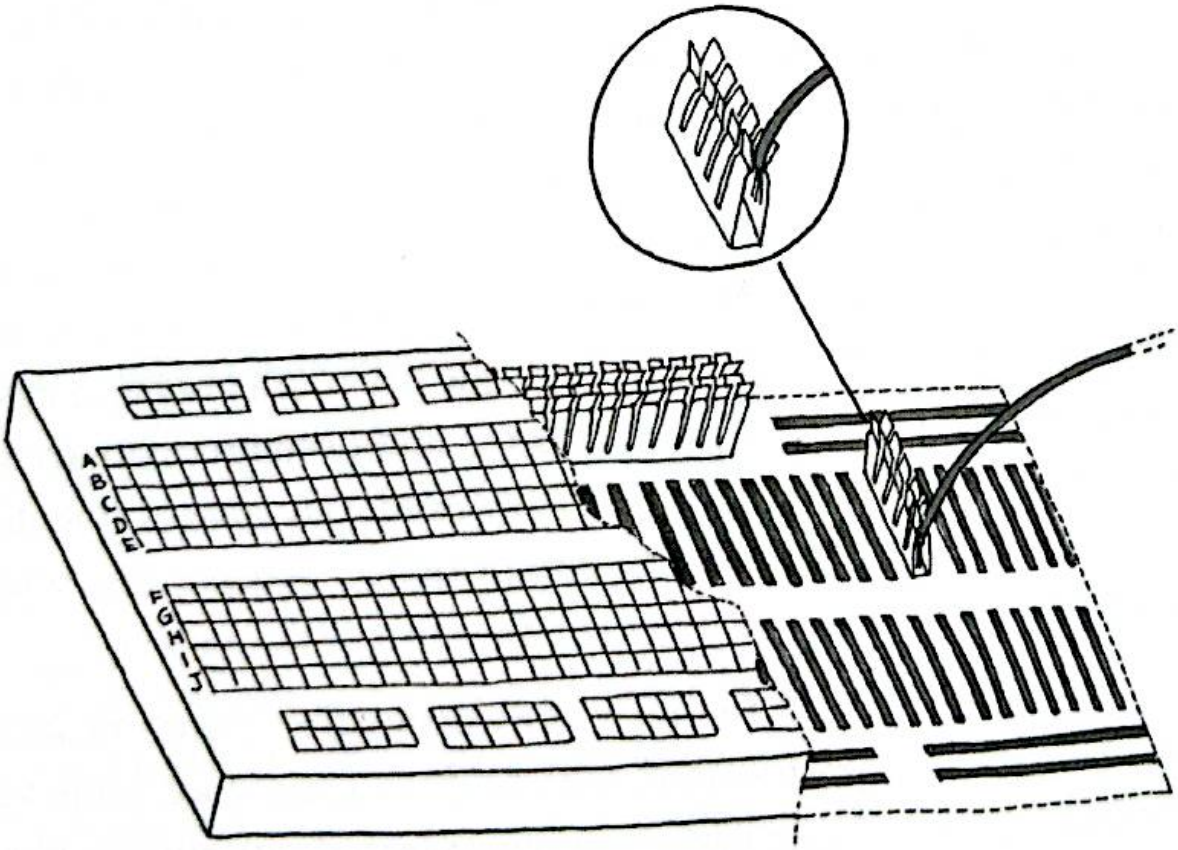


Figura A.1 – Protoboard sem solda.

Digitalizado com CamScanner

APÊNDICE B

Leitura de resistores e capacitores

Para que possa utilizar componentes eletrônicos, você deve ser capaz de identificá-los, tarefa que pode ser difícil para um iniciante. A maioria dos resistores encontrados em lojas tem um corpo cilíndrico, com duas pernas saindo uma de cada lado e marcações coloridas em seu corpo. Quando o primeiro resistor comercial foi criado, não havia como imprimir números pequenos o bastante que coubessem em seu corpo. Assim, engenheiros decidiram que representariam os valores dos resistores com listras de diferentes cores.

Usuários inexperientes em eletrônica devem descobrir uma forma de interpretar esses sinais. As indicações são bem simples: geralmente temos quatro listras, e cada cor representa um número. Um dos anéis geralmente é dourado e representa a precisão desse resistor. Para ler as listras na ordem correta, segure o resistor de modo que a listra dourada (ou prateada, em alguns casos) fique à direita. Depois, leia e traduza as cores de acordo com os números corretos. Na tabela que fornecemos a seguir, você encontrará uma tradução das cores e de seus valores numéricos correspondentes.

Cor	Valor
Preto	0
Marrom	1
Vermelho	2
Laranja	3
Amarelo	4
Verde	5

Digitalizado com CamScanner

Cor	Valor
Azul	6
Violeta	7
Cinza	8
Branco	9
Prata	10%
Dourado	5%

Dessa forma, um resistor com marcação marrom, preta e laranja significa 1 0 3 ±5%. Fácil, não? Nem tanto, pois há uma pegadinha: na realidade, o terceiro anel representa o número de zeros no valor. Assim, 1 0 3 represente, de fato, 1 0 seguido por 3 zeros. O resultado final é 10.000 ohms ±5%. Usuários mais experientes de eletrônica tendem a encurtar valores como esse, expressando-os em kilo ohms (para milhares de ohms) e mega ohms (para milhões de ohms). Assim, um resistor de 10.000 ohms é representado como 10k, enquanto um de 10.000.000, 10M. Fique atento, pois, uma vez que engenheiros gostam de otimizar tudo, em diagramas esquemáticos você poderá encontrar valores como 4k7, o que significa 4,7 kilo ohms, ou 4.700.

Capacitores são um pouco mais fáceis: um capacitor cilíndrico (eletrolítico) geralmente vem com seu valor impresso em seu próprio corpo. O valor de um capacitor é medidor em farads (F), mas a maioria dos capacitores tem seu valor medido em micro farads (µF). Desse modo, ao ver um capacitor marcado como 100 µF, saiba que se trata de um capacitor de 100 micro farads.

Muitos dos capacitores em forma de disco (capacitores cerâmicos) não têm suas unidades listadas dessa forma e utilizam um código numérico de três dígitos para indicar seu número de pico farads (pF). Há 1.000.000 pF em um µF. Nesses casos, assim como o que vimos no código dos resistores, você deve utilizar o terceiro número para determinar o número de zeros a ser colocado depois dos dois primeiros dígitos. Nesse caso, há apenas uma diferença: se você vir

Digitalizado com CamScanner

uma notação de 0 a 5, saiba que ela indica o número de zeros. 6 e 7 não são utilizados, e 8 e 9 não empregados de modo diferente. Se você vir 8, multiplique o número que os dois primeiros dígitos formam por 0,01, e se vir 9, multiplique-o por 0,1.

Assim, um capacitor marcado como 104 seria de 100.000 pF, ou 0,1 µF. Um capacitor marcado como 229, seria de 2,2 pF.

Digitalizado com CamScanner

APÊNDICE C

Referência rápida do Arduino

Nesta seção forneceremos uma rápida explicação de todas as instruções básicas aceitas pela linguagem do Arduino.

Para uma referência mais detalhada, consulte: *arduino.cc/en/Reference/HomePage*.

Estrutura

Um sketch do Arduino é executado em duas partes:

void setup()

Local em que você coloca seu código de inicialização – instruções que preparam a placa antes do início do loop principal do sketch.

void loop()

Seção que contém o código principal de seu sketch e que deve apresentar um conjunto de instruções a serem repetidas seguidas vezes até que a placa seja desligada.

Símbolos especiais

O Arduino inclui alguns símbolos que devem ser empregados em linhas de código, comentários e blocos de código:

Digitalizado com CamScanner

; (ponto e vírgula)

Toda instrução (linha de código) é encerrada por um ponto e vírgula. Essa sintaxe permite que você formate seu código livremente. Você pode até colocar duas instruções na mesma linha, desde que as separe por um ponto e vírgula. (Esteja atento, pois isso dificultará a leitura de seu código.)

Exemplo:

```
delay(100);
```

{} (chaves)

Essa notação é utilizada para marcar blocos de código. Por exemplo, quando você escreve código para a função `loop()`, tem de utilizar chaves antes e depois do código.

Exemplo:

```
void loop() {
    Serial.println("ciao");
}
```

Comentários

Porções do texto ignoradas pelo processador do Arduino, mas extremamente úteis para que você se recorde do que fez nesse trecho do código, ou para explicar essa funcionalidade a outras pessoas.

Há dois tipos de comentários no Arduino:

```
// de linha única: o texto até o fim da linha será ignorado
/* de várias linhas:
   você pode escrever
   um poema inteiro aqui
*/
```

Digitalizado com CamScanner

Constantes

O Arduino inclui um conjunto de palavras-chave predefinidas com valores especiais. `HIGH` e `LOW` são utilizadas, por exemplo, quando você deseja ligar ou desligar um pino do Arduino. `INPUT` e `OUTPUT` são utilizadas para definir um pino específico como entrada ou saída.

`true` e `false` indicam se uma condição ou expressão é verdadeira ou falsa, respectivamente.

Variáveis

Variáveis são áreas nomeadas da memória do Arduino nas quais você pode armazenar dados a serem utilizados e manipulados em seu sketch. Como sugere o nome, variáveis podem ser modificadas quantas vezes você quiser.

Como o Arduino é um processador muito simples, quando você declara uma variável, deve também especificar seu tipo, ou seja: dizer ao processador o tamanho do valor que você deseja armazenar.

Veja os *tipos de dados* disponíveis:

boolean

Pode ter um de dois valores: verdadeiro ou falso.

char

Armazena um único caractere, como A. Da mesma forma que qualquer computador, o Arduino armazena esse valor como um número, ainda que você veja um texto. Quando caracteres são utilizados para armazenar números, eles podem armazenar valores de -128 a 127.

NOTA: Há dois grandes grupos de caracteres disponíveis a sistemas de computadores: ASCII e Unicode. ASCII é um conjunto de 127 caracteres, utilizado, dentre outras coisas, para transmitir textos entre terminais seriais e sistemas de computadores de tempo compartilhado

Digitalizado com CamScanner

como mainframes e minicomputadores. Unicode, por sua vez, é um conjunto muito maior de valores utilizados por sistemas operacionais modernos para representar caracteres em muitos idiomas diferentes. ASCII ainda é útil, por exemplo, para transferência de pequenas quantidades de informações, em idiomas como italiano e inglês, que utilizam caracteres latinos, numerais arábicos e símbolos datilográficos comuns para pontuação.

byte

Armazena um valor entre 0 e 255. Assim como no caso de `char`, bytes utilizam apenas um byte de memória.

int

Utiliza dois bytes da memória para representar um número entre -32.768 e 32.767; trata-se do tipo de dado mais comum utilizado no Arduino.

unsigned int

Assim como `int`, utiliza 2 bytes, mas o prefixo `unsigned` significa que não pode armazenar números negativos. Seu alcance, portanto, é de 0 a 65.535.

long

Duas vezes o tamanho de um `int` e armazena números de -2.147.483.648 a 2.147.483.647.

unsigned long

Versão não assinalada de `long`; vai de 0 a 4.294.967.295.

float

Tipo de dado de tamanho considerável, capaz de armazenar valores de ponto flutuante, ou seja: números com ponto decimal. Um `float` consumirá 4 bytes de seus preciosos recursos de RAM. As

Digitalizado com CamScanner

funções que podem utilizá-lo também consomem muita memória, por isso utilize floats apenas quando estritamente necessário.

double

Número ponto flutuante de precisão dupla, com valor máximo de $1{,}7976931348623157 \times 10^{308}$. Nossa, que número enorme!

string

Conjunto de caracteres ASCII utilizados para armazenar informações textuais (você pode utilizar uma string quando quiser enviar uma mensagem por meio de uma porta serial, ou para mostrá-la em um monitor LCD). Quanto ao armazenamento, uma string utiliza um byte para cada caractere, mais um caractere nulo para avisar ao Arduino que o fim da linha foi atingido. As duas linhas seguintes são equivalentes:

```
char string1[] = "Arduino"; // 7 caracteres + 1 caractere nulo
char string2[8] = "Arduino";  // igual à linha anterior
```

array

Lista de variáveis que podem ser acessadas por meio de um índice. Um array é utilizado para criar tabelas de valores que podem ser acessados com facilidade. Por exemplo, caso você queira armazenar níveis diferentes de brilho a serem utilizados por um LED, pode criar seis variáveis, nomeadas `light01`, `light02` e assim por diante. Melhor ainda, você poderia utilizar um array bem simples, como:

```
int light[6] = {0, 20, 50, 75, 100};
```

A palavra "array" não chega a ser utilizada na declaração da variável: os símbolos `[]` e `{}` são suficientes.

Estruturas de controle

O Arduino inclui palavras-chave para controlar o fluxo lógico de seu sketch.

Digitalizado com CamScanner

if... else

Essa estrutura é responsável pela tomada de decisões em seu programa. `if` deve ser seguido por uma questão, especificada como uma expressão entre parênteses. Se a expressão for verdadeira, o que vier depois dela será executado. Se falsa, o bloco de código que segue `else` será executado. É possível utilizar apenas `if`, sem especificar uma cláusula `else`.

Exemplo:

```
if (val == 1) {
    digitalWrite(LED, HIGH);
}
```

for

Permite que você repita um bloco de código por um número especificado de vezes.

Exemplo:

```
for (int i = 0; i < 10; i++) {
    Serial.print("ciao");
}
```

switch case

Enquanto a instrução `if` funciona como uma bifurcação que oferece duas opções ao seu programa, `switch case` se parece mais com uma enorme rotatória. Ela permite que seu programa receba várias orientações, dependendo do valor de uma variável, sendo ótima para manter seu código organizado, uma vez que substitui listas de instruções `if`.

Exemplo:

```
switch (sensorValue) {
    case 23:
        digitalWrite(13, HIGH);
        break;
```

Digitalizado com CamScanner

```
        case 46:
            digitalWrite(12, HIGH);
            break;
        default:   // se nenhum case é executado, executa-se esta linha
            digitalWrite(12, LOW);
            digitalWrite(13, LOW);
    }
```

while

Semelhantemente a `if`, `while` executará um bloco de código enquanto determinada condição for verdadeira.

Exemplo:

```
// pisca o LED enquanto o valor do sensor estiver abaixo de 512
sensorValue = analogRead(1);
while (sensorValue < 512) {
    digitalWrite(13, HIGH);
    delay(100);
    digitalWrite(13, HIGH);
    delay(100);
    sensorValue = analogRead(1);
}
```

do... while

Idêntica à `while`. A única diferença é que, aqui, o código será executado antes de a condição ser avaliada. Essa estrutura é utilizada quando você deseja que seu bloco de código seja executado ao menos uma vez antes de conferir a condição.

Exemplo:

```
do {
    digitalWrite(13, HIGH);
    delay(100);
    digitalWrite(13, HIGH);
    delay(100);
```

Digitalizado com CamScanner

```
    sensorValue = analogRead(1);
} while (sensorValue < 512);
```

break

Esse termo permite que você saia de um loop e continue a execução do código que aparece depois dele. `break` também é utilizado para separar seções distintas de uma instrução `switch case`.

Exemplo:

```
// pisca o LED enquanto o valor do sensor estiver abaixo de 512
do {
    // sai do loop se um botão é pressionado
    if (digitalRead(7) == HIGH)
        break;
    digitalWrite(13, HIGH);
    delay(100);
    digitalWrite(13, LOW);
    delay(100);
    sensorValue = analogRead(1);
} while (sensorValue < 512);
```

continue

Quando utilizado dentro de um loop, `continue` permite que você pule o resto do código e faça com que a condição seja testada novamente.

Exemplo:

```
for (light = 0; light < 255; light++)
{
    // pula as intensidades entre 140 e 200
    if ((x > 140) && (x < 200))
        continue;
    analogWrite(PWMpin, light);
    delay(10);
}
```

Digitalizado com CamScanner

return

Interrompe a execução de uma função, retornando-a. Você também pode utilizar esse comando para retornar um valor de dentro da função.

Por exemplo, se você tiver uma função de nome `computeTemperature()` e quiser retornar seu resultado para a seção do código que a invocou, poderia escrever algo como:

```
int computeTemperature() {
    int temperature = 0;
    temperature = (analogRead(0) + 45) / 100;
    return temperature;
}
```

Aritmética e fórmulas

Você pode utilizar o Arduino para realizar cálculos complexos utilizando uma sintaxe especial. + e - funcionam como você aprendeu na escola. A multiplicação é representada com por * e a divisão, por /.

Há um operador adicional, chamado "módulo" (%), que retorna o resto da divisão de um inteiro. Você pode utilizar quantos níveis de parênteses forem necessários para agrupar as expressões. Ao contrário do que você aprendeu na escola, colchetes e chaves estão reservados a outros propósitos (índices e blocos de arrays, respectivamente).

Exemplos:

```
a =2 + 2;
light = ((12 * sensorValue) - 5) / 2;
remainder = 3 % 2;      // retorna 1
```

Digitalizado com CamScanner

Operadores de comparação

Quando você especifica condições ou testes para instruções `if`, `while` e `for`, estes são os operadores que podem ser utilizados:

`==`	igual a
`!=`	não igual a
`<`	menor que
`>`	maior que
`<=`	menor que ou igual a
`>=`	maior que ou igual a

Operadores booleanos

Operadores booleanos serão utilizados quando você quiser combinar várias condições. Por exemplo, caso você queira verificar se o valor vindo do sensor está entre 5 e 10, deve escrever:

```
if ((sensor => 5) && (sensor <=10))
```

Há três operadores: *e*, representado por `&&`; *ou*, representado por `||`; por fim, *não*, representado por `!`.

Operadores compostos

Operadores compostos são operadores especiais utilizados para tornar o código mais conciso quando temos de realizar operações comuns, como incrementar o valor de uma variável.

Por exemplo, para incrementar a variável `value` em uma unidade, você poderia escrever:

```
value = value +1;
```

Utilizando um operador composto, essa linha se tornará:

```
value++;
```

Incremento e decremento (-- e ++)

Incrementam e decrementam determinado valor em uma unidade.

Digitalizado com CamScanner

Tenha cuidado: ao escrever `i++`, você incrementa o valor de `i` em uma unidade e avalia um resultado equivalente a `i+1`; `++i`, por sua vez, avalia o valor de `i` e só então faz o incremento. O mesmo se aplica a `--`.

`+=`, `-=`, `*=` e `/=`

Esses operadores facilitam o uso de certas expressões. As duas expressões a seguir são equivalentes:

```
a = a + 5;
a += 5;
```

Funções de entrada e saída

O Arduino inclui funções para a manipulação de entradas e saídas. Você já viu algumas delas nos programas deste livro.

pinMode(pino, modo)

Reconfigura um pino digital para que ele se comporte como entrada ou saída.

Exemplo:

```
pinMode(7,INPUT);      // transforma o pino 7 em entrada
```

digitalWrite(pino, valor)

Liga ou desliga um pino digital. Pinos devem ser marcados explicitamente como saída, utilizando `pinMode`, antes que `digitalWrite` possa ter qualquer efeito.

Exemplo:

```
digitalWrite(8, HIGH); // liga o pino digital 8
```

int digitalRead(pino)

Lê o estado de um pino de entrada, retornando `HIGH` se o pino recebe voltagem, ou `LOW` se não há voltagem aplicada.

Digitalizado com CamScanner

Exemplo:

```
val = digitalRead(7);  // lê o valor do pino 7, atribuindo-o a val
```

int analogRead(pino)

Lê a tensão aplicada a um pino de entrada analógica e retorna um número entre 0 e 1023, representando voltagens entre 0 e 5V.

Exemplo:

```
val = analogRead(0);   // lê o valor da entrada analógica 0,
                       // atribuindo a val
```

analogWrite(pino, valor)

Altera a taxa PWM em um dos pinos marcados PWM. *pin* pode ser 11, 10, 9, 6, 5 e 3. *value* pode ser um número entre 0 e 255, representando a escala entre 0 e 5V da tensão de saída.

Exemplo:

```
analogWrite(9,128); // Diminui a luminosidade de um LED no pino 9 em 50%
```

shiftOut(dataPin, clockPin, bitOrder, valor)

Envia dados para um *registrador de deslocamento*, dispositivo utilizado para expandir o número de saídas digitais. Esse protocolo utiliza um pino para dados e um para o clock. *bitOrder* indica a ordem dos bytes (do menos significativo para o mais significativo) e *valor* é o byte em si que será enviado.

Exemplo:

```
shiftOut(dataPin, clockPin, LSBFIRST, 255);
```

unsigned long pulseIn(pino, valor)

Mede a duração de um pulso vindo de uma das entradas digitais. Isso é útil quando queremos, por exemplo, ler sensores infravermelhos ou acelerômetros que emitem seus valores como pulsos de duração diferente.

Digitalizado com CamScanner

Exemplo:

```
time = pulsein(7, HIGH);   // mede o tempo que o próximo
                           // pulso permanece high
```

Funções de tempo

O Arduino inclui funções capazes de medir o tempo transcorrido e também pausar o sketch.

unsigned long millis()

Retorna o número de milissegundos transcorridos desde que o sketch teve início.

Exemplo:

```
duration = millis()-lastTime; // computa tempo transcorrido desde
                              // "lastTime"
```

delay(ms)

Pausa o programa pelo tempo especificado em milissegundos.

Exemplo:

```
delay(500);     // interrompe o programa por meio segundo
```

delayMicroseconds(us)

Pausa o programa pelo tempo especificado em microssegundos.

Exemplo:

```
delayMicroseconds(1000);   // espera 1 milissegundo
```

Funções matemáticas

O Arduino inclui muitas funções matemáticas e trigonométricas comuns:

Digitalizado com CamScanner

min(x, y)

Retorna o menor *x* e *y*.

Exemplo:

```
val = min(10,20);  // agora, val é 10
```

max(x, y)

Retorna o maior *x* e *y*.

Exemplo:

```
val = max(10,20);  // agora, val é 20
```

abs(x)

Retorna o valor absoluto de *x*, o que transforma números negativos em positivos. Se *x* for 5, ela retornará 5; se *x* for -5, ela ainda retornará 5.

Exemplo:

```
val = abs(-5);     // agora, val é 5
```

constrain(x, a, b)

Retorna o valor de *x*, entre *a* e *b*. Se *x* for menor que *a*, ela retorna *a*. Se *x* for maior que *b*, ela retorna *b*.

Exemplo:

```
val = constrain(analogRead(0), 0, 255);   // rejeita valores maiores
                                          // que 255
```

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

Mapeia um valor no intervalo de *fromLow* a *maxLow*, a um intervalo entre *toLow* e *toHigh*. Esse recurso é muito útil para processar valores de sensores analógicos.

Digitalizado com CamScanner

Exemplo:

```
val = map(analogRead(0),0,1023,100, 200); // mapeia o valor do pino
                                          // analógico 0 a um valor
                                          // entre 100 e 200
```

double pow(base expoente)

Retorna o resultado da potenciação de um número (*base*) por um valor específico (*expoente*).

Exemplo:

```
double x = pow(y, 32);     // define x como y elevado à 32ª potência
```

double sqrt(x)

Retorna a raiz quadrada de um número.

Exemplo:

```
double a = sqrt(1138);     // aproximadamente 33,73425674438
```

double sin(rad)

Retorna o seno de um ângulo especificado em radianos.

Exemplo:

```
double sine = sin(2);  // aproximadamente 0,90929737091
```

double cos(rad)

Retorna o cosseno de um ângulo especificado em radianos.

Exemplo:

```
double cosine = cos(2);    // aproximadamente -0,41614685058
```

double tan(rad)

Retorna a tangente de um ângulo especificado em radianos.

Digitalizado com CamScanner

Exemplo:

```
double tangent = tan(2); // aproximadamente -2,18503975868
```

Funções de números aleatórios

Caso você tenha de gerar números aleatórios, pode utilizar o gerador de números pseudoaleatórios rduino.

randomSeed(semente)

Essa função redefine o gerador de números pseudoaleatórios do Arduino. Ainda que a distribuição dos números retornados por `random()` seja essencialmente aleatória, a sequência que ela oferece é previsível. Por isso, você deve redefinir o gerador utilizando um valor aleatório. Caso você tenha um pino analógico conectado, ele deve captar ruídos variados do ambiente (como ondas de rádio, raios cósmicos, interferência eletromagnética de celulares e luzes fluorescentes e assim por diante) que podem ser utilizados nesse caso.

Exemplo:

```
randomSeed(analogRead(5)); // randomiza utilizando o ruído do pino 5
```

long random(max)
long random(min, max)

Retorna um valor de inteiro *long* pseudoaleatório, entre *min* e *max -1*. Se o valor mínimo não for especificado, seu limite será 0.

Exemplo:

```
long randnum = random(0, 100);     // um número entre 0 e 99
long randnum = random(11);         // um número entre 0 e 10
```

Comunicação serial

Como vimos no capítulo 5, você pode efetuar a comunicação com dispositivos por meio da porta USB utilizando um protocolo de comunicação serial. Veja, a seguir, as funções seriais.

Digitalizado com CamScanner

Serial.begin(taxa)

Prepara o Arduino para iniciar a transferência de dados seriais. No caso do monitor serial do IDE do Arduino, você geralmente utilizará uma taxa de 9.600 bits por segundos (bps), mas há outras taxas de transferência disponíveis, normalmente inferiores a 115.200 bps.

Exemplo:

```
Serial.begin(9600);
```

Serial.print(dados)
Serial.print(dados, codificação)

Envia dados para a porta serial. A codificação é opcional; se não especificada, os dados serão tratados como texto simples.

Exemplos:

```
Serial.print(75);          // imprime "75"
Serial.print(75, DEC);     // igual ao comando anterior
Serial.print(75, HEX);     // "4B" (75 em notação hexadecimal)
Serial.print(75, OCT);     // "113" (75 em notação octal)
Serial.print(75, BIN);     // "1001011" (75 em notação binária)
Serial.print(75, BYTE);    // "K" (o byte que representa
                           // 75 no conjunto ASCII)
```

Serial.println(dados)
Serial.println(dados, codificação)

Funciona como *Serial.print()*, exceto adiciona um retorno de carro e um avanço de linha (*\r\n*), como se você tivesse digitado os dados e pressionado Return ou Enter.

Exemplos:

```
Serial.println(75);        // imprime "75\r\n"
Serial.println(75, DEC);   // igual ao comando anterior
Serial.println(75, HEX);   // "4B\r\n"
```

Digitalizado com CamScanner

```
Serial.println(75, OCT);   // "113\r\n"
Serial.println(75, BIN);   // "1001011\r\n"
Serial.println(75, BYTE);  // "K\r\n"
```

int Serial.available()

Retorna o número de dados não lidos, disponíveis na porta serial para leitura por meio da função `read()`. Depois que você tiver lido todos os dados disponíveis, `Serial.available` retornará 0 até que novos dados cheguem à porta serial.

Exemplo:

```
int count = Serial.available();
```

int Serial.read()

Busca um byte dos dados seriais entrantes.

Exemplo:

```
int data = Serial.read();
```

Serial.flush()

Como os dados podem chegar à porta serial mais rápido do que seu programa é capaz de processá-los, o Arduino armazenará todos os dados entrantes em um buffer. Caso você tenha de limpar o buffer para preenchê-lo com novos dados, utilize a função `flush()`.

Exemplo:

```
Serial.flush();
```

Digitalizado com CamScanner

APÊNDICE D

Leitura de diagramas esquemáticos

Até aqui, sempre utilizamos ilustrações detalhadas para descrever a montagem de nossos circuitos. Porém, como você pode imaginar, não será muito prático desenhar um diagrama desses quando você quiser simplesmente documentar um experimento.

Problemas semelhantes surgem, mais cedo ou mais tarde, em todas as áreas de atuação. Na música, por exemplo, depois de compor uma bela canção, você terá de anotá-la utilizando cifras musicais.

Engenheiros, pessoas práticas que são, desenvolveram uma forma rápida de capturarmos a essência de um circuito, para que possamos documentá-lo e reconstruí-lo posteriormente, ou mesmo passá-lo a outra pessoa.

Em eletrônica, *diagramas esquemáticos* permitem que você descreva seu circuito de uma forma que possa ser compreendida pelo restante da comunidade. Componentes individuais são representados por símbolos que representam abstrações de seu próprio formato, ou de sua essência. Por exemplo, o capacitor é formado por duas placas de metal separadas por ar ou plástico; assim, seu símbolo é:

Digitalizado com CamScanner

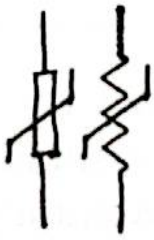
Outro exemplo evidente é o indutor, formado por um fio de cobre espiralado em formato cilíndrico; seu símbolo pode ser visto à esquerda.

Conexões entre componentes são geralmente feitas utilizando-se fios ou as próprias conexões da placa de circuito impresso e representadas no diagrama como linhas simples. Quando dois fios estão conectados, utilizamos um ponto para representar o local em que as duas linhas se cruzam, como na ilustração à esquerda.

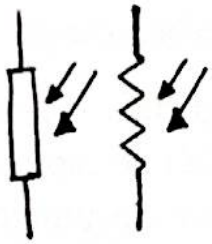
Isso é tudo que você deve saber para compreender o básico dos diagramas esquemáticos. Veja a seguir uma lista mais completa de símbolos e seus significados.

Resistor

Capacitor



Termistor

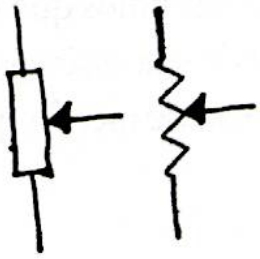


Sensor de luz LDR

Diodo

LED

Botão de pressão

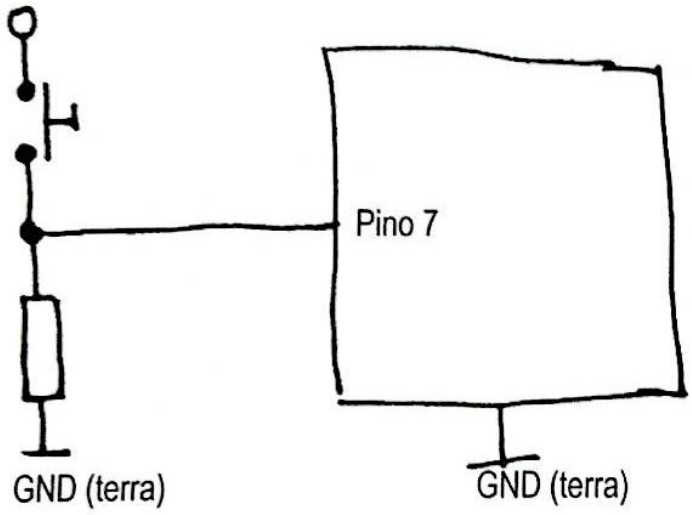


Potenciômetro

Digitalizado com CamScanner

Talvez você encontre variações desses símbolos (mostramos, por exemplo, os dois tipos de símbolos para resistores). Consulte *en.wikipedia.org/wiki/Electronic_symbol* para uma lista mais completa de símbolos eletrônicos. Por convenção, diagramas são desenhados da esquerda para a direita. O circuito de um rádio, por exemplo, seria desenhado com sua antena à esquerda, seguido pelo caminho que o sinal de rádio percorre até o alto-falante (desenhado à direita).

O diagrama esquemático a seguir descreve o circuito com botão de pressão que vimos neste livro.



Digitalizado com CamScanner

Símbolos

\ 24, 88, 100
+= (adição e atribuição), operador de 127
+ (adição), operador de 125
{} (chaves) 49, 50, 118
  em arrays 121
  em torno de blocos de código 125
[] (colchetes), em arrays 121, 125
-- (decremento), operador de 126
/**/, delimitador de comentário 118
//, delimitador de comentário 48, 118
/= (divisão e atribuição), operador de 127
/ (divisão), operador de 125
#, em códigos HTML coloridos 90
&& (e), operador 126
== (igual a), operador de 60, 126
++ (incremento), operador de 126
> (maior que), operador de 126
>= (maior que ou igual a), operador de 126
< (menor que), operador de 126
<= (menor que ou igual a), operador de 126
% (módulo), operador de 125
*= (multiplicação e atribuição), operador de 127
* (multiplicação), operador de 125
!= (não igual a), operador de 126
! (não), operador 126
|| (ou), operador 126
() (parênteses) 49
  agrupando expressões 125
  depois da palavra-chave if 60
; (ponto e vírgula)
  encerrando instruções 61
  encerrando linhas de código 118
= (sinal de igual), operador de atribuição 60, 63
-= (subtração e atribuição), operador de 127
- (subtração), operador de 125

A

A (ânodo) 43
Abajur de mesa 100
abs(), função 130
Acelerômetros 85
Adaptador AC 35
Ajuda online 108

Digitalizado com CamScanner

Alarme contra roubos, sensores infravermelhos 69
Alimentação 35
Ampères 55
Analógica
  entrada 78, 87
  pinos analógicos na placa do Arduino 35
  saída 87
analogRead(), função 78, 128
  valores de brilho 81
analogWrite(), função 71, 128
  valores de brilho 80
Ânodo 43
Arduino
  blocos fundamentais 87
  circuito de motor para 85
  colaboração entre usuários 31
  conexão à Internet 89
  diferenças de outras plataformas 17
  FAQs no site principal 109
  filosofia 21, 22, 25
  hardware 34
  instalação 37
  partes principais da placa 35
  placa Diecimila 36, 105
  placa Duemilanove 35
  placa NG 36, 105
  placa Uno 34, 37
  protoboard sem solda 111
  teste da placa 104
Arduino, linguagem 117
  caracteres especiais 117
  comunicação serial 132
  estruturas de controle 121
  funções de entrada e saída 127
  funções de números aleatórios 132
  funções matemáticas e trigonométricas 129
  operadores booleanos 126
  operadores compostos 126
  operadores de comparação 126
  tipos de dados 119
  variáveis 119
Arduino Store 57
ArduinoUNO.inf, arquivo 38
Argumentos 50, 51
  função delay() 51

Digitalizado com CamScanner

Aritmética e fórmulas 125
Arquivo XML para feed RSS 89
Arrays 121
ASCII 121
atuadores 19, 42
avr-gcc, compilador 36

B

blocos de código 47
  nomenclatura 47
boolean, tipo de dados 119
botão de pressão, diagrama esquemático 137
botão de pressão táctil momentânea 57
bouncing. Consulte repiques
brilho
  ajuste por meio da entrada analógica 80
  alteração em LEDs piscantes 70
brinquedos, modificação 22
buffers 90
byte, tipo de dados 120

C

cálculos 125
capacitores
  leitura 114
  símbolo para 135
caracteres especiais 117
  += (adição e atribuição), operador de 127
  + (adição), operador de 125
  {} (chaves) 47, 118
  [] (colchetes) 125
  -- (decrement), operador de 126
  /**/ (delimitador de comentário) 118
  // (delimitador de comentário) 118
  /= (divisão e atribuição), operador de 127
  / (divisão), operador de 125
  && (e), operador 126
  == (igual a), operador de 60, 126
  > (maior que), operador de 126
  < (menor que), operador de 126
  % (módulo), operador de 125
  *= (multiplicação e atribuição), operador de 127
  * (multiplicação), operador de 125
  != (não igual a), operador de 126
  ! (não), operador 126

Digitalizado com CamScanner

|| (ou), operador 126
() (parênteses) 49, 60, 125
; (ponto e vírgula) 61, 118
= (sinal de igual), operador de atribuição 60, 63
-= (subtração e atribuição), operador de 127
caracteres latinos 120
cátodo 43
cátodo comum 100
char, tipo de dados 119
circuitos
Circuit Bending 25
diagramas esquemáticos 135
fórmula da Lei de Ohm 56
lâmpada em rede 99
motor para o Arduino 85
relacionamento entre tensão, corrente e resistência 52
sensor analógico 79
um circuito, muitos comportamentos 60
código
acende o LED quando o botão é pressionado 62
acende o LED quando o botão é pressionado, com recurso simples de de-bouncing 66
acende o LED quando o botão é pressionado, mantendo-o aceso depois disso 64
ajuste do brilho do LED por meio da entrada analógica 80
Arduino com luminária em rede, na Processing 91
Arduino com luminária em rede, no Arduino 96
objetos seriais 82
piscando um LED a uma velocidade especifica pela entrada analógica 79
variáveis 60
colaboração entre usuários do Arduino 31
Colombo, Joe 88
comentários 48, 118
compilador 36
Computação Física 19
comunicação serial 82, 88, 132
objetos seriais 82
Serial Monitor, botão 83
conjuntos de caracteres 119
constantes 49, 119
constrain(), função 130
continue, instrução 124
cores, codificação em HTML 90
corrente 55
cos(), função 131

Digitalizado com CamScanner

D

debugging. Consulte depuração
delay(), função 51, 129
delayMicroseconds(), função 129
Depuração 104. Consulte resolução de problemas
Design de Interação 19
detector de movimentos, sensor infravermelho passivo (PIR) 69
Diagramas esquemáticos, leitura 135
Diecimila, placa 36, 105
Digital
  eletrônica programável, vantagens da 60
  entrada 87
  modo INPUT ou OUTPUT para pinos 49
  pinos 49
  pinos digitais 35
  saída 87
digitalRead(), função 57
  armazenando o resultado em uma variável 61
digitalWrite(), função 50, 127
Diodos
  1N4007 84
  símbolo para 136
Dispositivo Interativo 41
Divida e conquiste 103
double, tipo de dados 121
do... while, instrução 123
drivers, instalação 37
Duemilanove, placa 34
Dyson, James 22

E

Eletricidade 52
  fórmula da Lei de Ohm 56
  tensão 52
Entrada
  analógica 78, 87
  digital 87
  funções 127
Estruturas de controle 121
Extreme, placa 105

F

false 119
FALSE 60

Digitalizado com CamScanner

Fios jumper pré-cortados 57
float, tipo de dados 120
fonte de alimentação externa 36
  resolução de problemas com Arduinos mais antigos 105
for, instrução 122
Fórum 109
  regras para 109
Fotoresistor 82
Funções 47
  abs() 130
  analogRead() 78, 128
  analogWrite () 71
  analogWrite() 128
  comunicação serial 132
  constrain() 130
  cos() 131
  de entrada e saída 127
  delay() 51, 129
  delayMicroseconds() 129
  de números pseudo aleatórios 132
  de tempo 129
  digitalRead() 57, 127
  digitalWrite() 50, 127
  loop() 50, 117
  map() 130
  matemáticas e trigonométricas 129
  max() 130
  millis() 129
  min() 130
  pinMode() 49, 127
  pow() 131
  pulseIn() 128
  random() 132
  randomSeed() 132
  retornando resultados 60, 125
  setup() 117
  shiftOut() 128
  sin() 131
  sqrt() 131
  tan() 131

G

Gerenciador de Dispositivos (Windows) 39, 107
Ghazala, Reed 25
Grupos de usuários 31

Digitalizado com CamScanner

H

Haque, Usman 30
Hardware do Arduino 34
HIGH 50, 57
Hopper, Grace 104
HTML, cores representadas em 90

I

IDE (Integrated Development Environment) 36
  botão Serial Monitor 83
  download 17, 36
  menu Tools da Processing, Create Font 96
  resolução de problemas 107
  verificação do código 45
Identificação de porta 39
  número de porta no Windows 107
if... else, instrução 122
if, instruções 60
Indutor, símbolo para 136
INPUT 49
Interfaces gestuais 68
Interruptores 67
  botão de pressão táctil momentânea 57
  de inclinação 68
  de tapetes 68
  magnéticos 67
  transistores MOSFET 83
Intervalos
  ajustando para eliminar o repique do botão de pressão 65
  alterando para produzir padrões diferentes de iluminação 52
int, tipo de dados 120
int, variável 61

J

Jeito Arduino 21
Jumper de seleção de alimentação (PWR_SEL) 36

K

K (cátodo) 43
Kernighan, Brian W. 107

Digitalizado com CamScanner

L

Lâmpadas interativas 52
LDRs (light-dependent resistors) 42
LED pisca-pisca 42
  código-fonte, passo a passo 48
  código, piscando o LED a uma velocidade especificada pela entrada analógica 79
  controle com PWM 70
  sketch 42
LED RGB 100
LEDs
  conectando ao Arduino 43
  constante LED 49
  controlados por botões de pressão 56
  definindo o brilho com a entrada analógica 80
  polarização 99
  RGB 100
  sketch do LED pisca-pisca 42
  sketch do LED pisca-pisca, explicação do código 48
LEDs controlados por botões de pressão
  código 58
  código, acende o LED enquanto o botão estiver pressionado 59
  código, acende o LED quando o botão é pressionado, com recurso de debouncing 66
  código, acende o LED quando o botão é pressionado e o mantém aceso 62
  componentes 57
  repiques 65
Lei de Ohm, fórmula 56
Leitura de diagramas esquemáticos 135
Leitura de resistores e capacitores 113
light-dependent resistors. Consulte LDRs
light emitting diodes. Consulte LEDs
Linguagem C 36
Linux
  instalação do Arduino 37
L (LED) 43, 105
long, tipo de dados 120
loop(), função 48, 50, 117
LOW 50, 57
Luminária 88
Luminária esférica 88
Luz
  controle com PWM 70
  controle e interação 52

Digitalizado com CamScanner

M

Macintosh
  identificação de porta 39
  instalação de drivers 37
  instalação do Arduino 37
Make, blog 89
Maker Shed, componentes 57
map(), função 130
Memória
  Flash 62
  RAM e Flash 62
Microcontroladores 19
Milissegundos 51
millis(), função 129
min(), função 130
Moog, Robert 23
MOSFET, transistor 83
  IRF520 84

N

NG, placa 36, 105
Numerais arábicos 120
Números decimais 90
Números hexadecimais 90

O

Objeto, definição 82
ohms 55
Operadores
  booleanos 126
  compostos 126
  de comparação 126
OUTPUT 49

P

Patching 23
Pike, Rob 107
pinMode(), função 50, 127
Pinos, placa Arduino 35, 111
  20 miliampères, capacidade máxima 83
  analógicos 128
  configuração de pinos digitais 127
  controle com a função analogWrite() 71
  entrada analógica 78

Digitalizado com CamScanner

LEDs controlados por botões de pressão 59
problemas no upload 105
Sniffin' Glue 26
Somlai-Fischer, Adam 30
sqrt(), função 131
SV1, jumper, conexões 105
switch case, instrução 122

T

tan(), função 131
Teclados de computadores 27
Teclados, modificação 27
Tempo, funções de 129
Tensão
aplicada a um pino, verificação com a função digitalRead() 57
aplicada a um pino, verificando com analogRead() 78
definição 55
leitura 35
relacionamento com a corrente 55
Termostatos 67
Tipos de dados 119
Tranqueiras 29
TRUE 60
true e false 119
TX e RX (LEDs) 46

U

Unicode 119
Uno, placa 34
unsigned int, tipo de dados 120
USB
comunicação com o Arduino 82
fontes de alimentação 35
identificação de porta no Windows 108
programando o Arduino com um cabo USB 17
resolução de problemas da conexão com o Arduino 105

V

Variáveis 61, 119
state 63
tipos de dados 119
Variáveis state 63

Digitalizado com CamScanner

Vista (Windows)
identificação de porta 39
resolução de problemas na identificação de porta 108

W

while, instrução 123
Wiki 31
Windows
identificação de porta 39
instalação de drivers 38
instalação do Arduino 37
número de porta COM para o Arduino 107

X

XP (Windows)
identificação de porta 39
instalação de drivers 38
resolução de problemas na identificação de porta 108

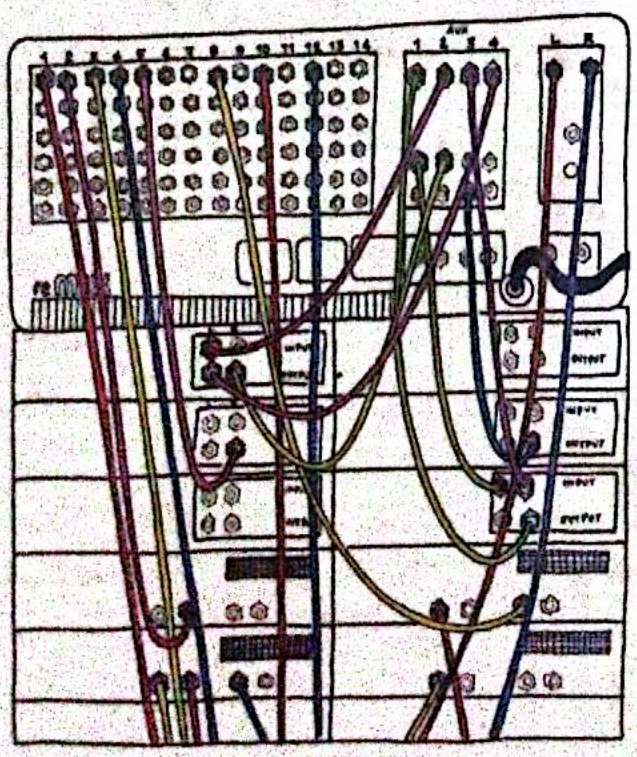
Digitalizado com CamScanner

Primeiros Passos com o Arduino

O Arduino é a plataforma de prototipagem eletrônica open source (fonte aberta) que está conquistando o mundo dos projetistas e inventores. A introdução completa fornecida neste livro, atualizada até o Arduino 1.0, oferece muitas ideias para seus projetos, além de ajudá-lo a começar imediatamente a trabalhar em suas criações. Desde dicas de organização até os toques finais de seu protótipo, todas as informações de que você precisa estão aqui!

Neste livro, você aprenderá sobre:

- Design de interação e computação física
- O hardware do Arduino e seu ambiente de desenvolvimento de software
- Noções básicas de eletricidade e eletrônica
- Prototipagem em uma protoboard sem solda
- Elaboração de diagramas esquemáticos



Este livro adota uma abordagem muito prática. Para utilizar os exemplos introdutórios deste guia, tudo que você precisa é de um Arduino Uno (ou modelo anterior), um cabo USB A-B e um LED. O download do ambiente de desenvolvimento do Arduino pode ser feito gratuitamente.

Junte-se às centenas de milhares de projetistas que descobriram essa incrível (e educativa) plataforma. Escrito pelo cofundador do Projeto Arduino, Primeiros Passos com o Arduino leva você para dentro dessa diversão.

Ilustrações de Elisa Canducci e Shawn Wallace.

ISBN 978-85-7522-290-4

9 788575 222904

O'REILLY
novatec

Digitalizado com CamScanner