

Erich Georg

Introdução à Programação de Arduino

Vol 1

**Aprenda a programar
na plataforma de desenvolvimento
mais popular da atualidade.**

**Explore o ambiente em nuvem
Arduino Create.**

**2ª Edição
Revista e Atualizada**



Sobre *Introdução à Programação de Arduino*

Programação já foi considerada uma atividade reservada a algumas pessoas de muito talento e inteligência acima da média, eleitas pelos deuses da matemática. Essa visão está mudando e a atividade de programar vem se tornando a cada dia mais presente, sobretudo depois da explosão da internet de maneira geral e em particular do internet das coisas - IoT, possível justamente graças a dispositivos programáveis como o Arduino, que nos brinda com uma plataforma de desenvolvimento que facilita de tal forma a programação que até aqueles que nunca pensaram serem capazes de tal atividade podem ter sucesso. E o propósito desse livro é justamente o de abrir uma porta larga, introduzindo de maneira didática essa poderosa ferramenta de programação que é, a uma só vez, útil, bonita, divertida, poderosa e, porque não dizer, necessária.

O livro é composto por três volumes e o estudo é feito de maneira gradual, em ordem crescente de complexidade. Esse primeiro volume se concentra em apresentar o Arduino como conceito e a plataforma de desenvolvimento, ensinando a instalar e testar o sistema. Mostra os componentes básicos usados para prototipagem, faz uma descrição detalhada das características do IDE - *Integrated Development Environment* e explica os conceitos necessários ao entendimento do processo de programar e transferir o programa desde o Ambiente de Programação Integrado para a memória do Arduino. Conclui com a montagem de

um experimento prático usando um *proto-board* e um *led*.

O texto está escrito em linguagem simples para que seja acessível, tendo sido feitos todos os esforços para esclarecer os conceitos indispensáveis ao perfeito entendimento do processo de programar um microcontrolador, tornando-o útil a um público o mais amplo possível e preparando, dessa forma, o alicerce que serve como base para estudos mais aprofundados e de ponto de partida para o que se seguirá nos outros dois volumes que dão continuidade à série.

OceanofPDF.com

Introdução à Programação de Arduino - volume 1

Erich Georg



OceanofPDF.com

Copyright© 2019 Erich Georg
e-book Edition

Escrito por Erich Georg
Capa, Projeto Gráfico e ilustrações pelo autor.

Todos os Direitos Reservados. Nenhuma parte dessa publicação pode ser reproduzida por quaisquer meios sem a expressa permissão do autor.

Aviso Legal: Embora todos os esforços sejam feitos nesse sentido, não há qualquer garantia de que o exposto nesse livro leve sempre aos resultados desejados. Use as informações aqui contidas por sua conta e risco. A manipulação de eletricidade oferece riscos e só deve ser levada a cabo por pessoa com conhecimentos e competência para tal. O autor e a editora não assumem qualquer responsabilidade por eventuais danos que possam ocorrer devido à utilização das informações aqui contidas. Use-as por sua conta e risco.

Visite nosso *website* e baixe o código fonte dos programas apresentados no texto. Lá você encontrará material adicional para consulta sobre o assunto exposto no livro.

Para comunicar alguma falha ou fazer algum comentário, crítica ou sugestão, por favor use o email: eeeprom@eeeprom.com.br

[EEPROM - eletro-eletrônica, programação e mecatrônica.](#)

[*OceanofPDF.com*](#)

Conteúdo

Sobre o livro

Folha de Rosto

Copyright

Dedicatória

Epígrafe

Capítulo 1 Admirável Mundo Novo

- ✧ Introdução
- ✧ Arduino, uma plataforma para programação de microcontroladores
- ✧ Arduino, o Conceito
- ✧ Arduino, a Plataforma
- ✧ Instalação do IDE do Arduino
- ✧ Menu Arquivo
- ✧ Menu Editar
- ✧ Menu Sketch
- ✧ Menu Ferramentas
- ✧ Conectando o Arduino

Capítulo 2 Democratizando a programação

- ✧ Programar é para Todos!
- ✧ Ampliando o Horizonte

Capítulo 3 Um Arsenal de Recursos

- ✦ Recursos do Arduino. E para ele!
- ✦ Analógico versus Digital
- ✦ Alimentação do Arduino
- ✦ Shields
- ✦ Material de apoio e prototipagem
- ✦ Leds
- ✦ Resistor
- ✦ Fios ou cabinhos

Capítulo 4 Mãos à Obra

- ✦ Montando um protótipo do Blink

Glossário

Livros do Autor

OceanofPDF.com

Dedicatória

À minha esposa, Márcia Gregório. Seu estímulo, confiança e paciência tem me motivado a continuar escrevendo.

OceanofPDF.com

Só há um lugar no universo no qual você pode ter a certeza de poder produzir progresso e esse lugar é dentro de você mesmo.

Aldous Huxley

OceanofPDF.com

Capítulo 1

Introdução

A imaginação dos criadores de ficção científica é algo digno de respeito e admiração. Assistindo a filmes como, por exemplo, **2001, Uma Odisseia no espaço** ou aos episódios de algumas séries antigas, como **Perdidos no Espaço**, percebemos o quão imaginativos podem ser esses autores. Mesmo assim, acredito que se um deles tivesse sido transportado por uma hipotética máquina do tempo, desde os anos 50 ou 60 do século passado para os dias atuais, ao chegar ficaria perplexo! Sim, nosso cotidiano é muito mais espetacular do que poderiam crer. Porque embora aqueles autores tenham tido ideias muito avançadas e até mesmo previsto a aparição de diversas tecnologias, nenhum deles imaginou a revolução de comunicação proporcionada pelo surgimento da internet e impulsionada pelas tecnologias de integração e automação de dispositivos. Essa combinação é explosiva, no melhor dos sentidos. É perfeitamente possível nos dias de hoje alguém em, digamos, Melbourne, Austrália, acessar um servidor de internet usando um celular comum, abrir uma página de um *website* usando seu celular, clicar num botão da

mesma e atuar num dispositivo qualquer localizado no outro lado do planeta, digamos, na cidade de Niterói, no Rio de Janeiro. Ou ainda, receber nesse mesmo celular um *email* de alerta enviado por um microcontrolador conectado à internet tendo assim a possibilidade de verificar, em tempo real, o motivo pelo qual seu cachorro está latindo tanto no quintal de casa. E, eventualmente, enviar uma mensagem de voz para acalmar o amigo. De fato, esse tipo de coisa já é mesmo comum e se lhe surpreende, não se sinta mal: acho que você não está sozinho porque algo me diz que espantaria até mesmo o nosso autor de ficção dos anos 50 ou 60 do século passado. E a verdade é que temos dispositivos e recursos tecnológicos dentro de nossas casas que nem **Os Jetsons** tinham!



Os Jetsons

O exemplo que eu dei linhas atrás, da pessoa que monitora o cão que dista milhares de quilômetros dela, é um entre muitos possíveis e envolve a automação de dispositivos e conectividade. Evidentemente é preciso uma boa dose de tecnologia e *know-how* de alto nível nos bastidores do desenvolvimento, para se implantar esse tipo de sistema. São necessários conhecimentos de programação para *web* e para microcontroladores, algumas boas noções de eletrônica e um perfeito entendimento sobre como se

pode integrar tudo isso. Em geral, essas tarefas podem ser distribuídas por profissionais de cada área ou podem ser todas levadas a cabo por um único profissional, que domina todas elas, coisa que não é tão rara como talvez possa parecer.

Do ponto de vista de nosso estudo, nos interessa a parte que diz respeito à programação e desenvolvimento do dispositivo que faz o controle efetivo. Ou seja, do microcontrolador, que no exemplo dado é responsável por detectar o latido insistente do cão e realizar uma ação em decorrência disso. A ação foi o envio de um *email*, mas poderia perfeitamente ter sido outro o evento, como por exemplo o acionamento de um sistema que abastece a vasilha de comida ou de água do amigo. Ou ainda de uma campainha tocando na casa de um vizinho, alertando para o desconforto do fiel amigo. Trocando em miúdos, nos interessa saber como programar o microcontrolador para realizar uma determinada tarefa em função de uma dada condição que se apresente e essa é a proposta desse livro e a razão de nosso estudo: aprender a programar o microcontrolador. Portanto, daqui até o final do livro trataremos de conhecer em pormenores aquela que vem sendo, nos últimos anos, a plataforma de desenvolvimento para microcontroladores que mais popularidade adquire: Arduino. Veremos ao longo do livro que essa popularidade não é fortuita, pelo contrário, ela se justifica plenamente.

Arduino, uma plataforma para programação de microcontroladores

Uma coisa que eu percebo com alguma frequência é que existe uma certa confusão sobre o Arduino. É um microcontrolador? Um ambiente de desenvolvimento, também conhecido como IDE?

Uma linguagem de programação de microcontroladores? Um conceito? O quê, afinal? É importante esclarecer isso antes de prosseguirmos.

Arduino é um conceito, uma ideia, aliás, uma ótima ideia. Mas primeiro, vejamos o que ele não é: não é um microcontrolador e não é uma linguagem de programação e, sendo assim, falar em microcontrolador Arduino ou em linguagem Arduino não faz sentido. Como eu disse, Arduino é, acima de tudo, um conceito, uma nova maneira de pensar e encarar a programação de microcontroladores. Quando falamos em Arduino, estamos falando ao mesmo tempo de, basicamente, três coisas:

- ✦ Em primeiro lugar, é uma placa, ou circuito, preparado para o desenvolvimento de programas para microcontroladores. Normalmente o microcontrolador, alojado na placa, é da Atmel© (leia-se Microchip©), um grande fabricante, podendo ser um entre alguns dos modelos disponibilizados pela Atmel©. Há várias versões de Arduino, como por exemplo o Uno, que utiliza um microcontrolador Atmega328P e que é o objeto de estudo desse livro.

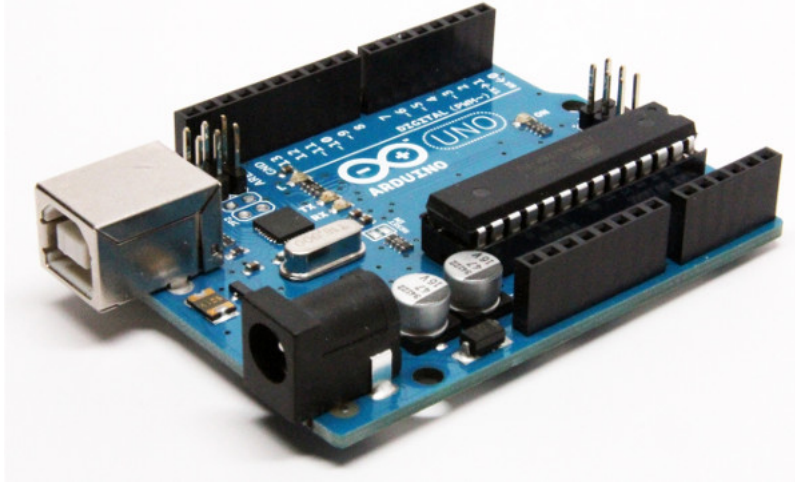
- ✦ É também um IDE - *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado, especialmente preparado para a programação e desenvolvimento com as placas de Arduino. O IDE pode ser usado com qualquer das versões de placas existentes, como o UNO, já citado, o Duemilanove, o LilyPad, entre outros. Ele é gratuito e *open source* e existe em duas versões, uma que roda localmente, instalada no seu computador e uma versão *web*, rodando em nuvem, como veremos mais tarde. Esse

IDE vem acompanhado de uma biblioteca extensa e de ferramentas que facilitam grandemente a programação, como veremos logo.

✦ Um compilador que possibilita converter o programa, escrito num dialeto do C++ especialmente preparado para essa tarefa, em linguagem de máquina, sequências de zeros e uns, do microcontrolador alvo.

O terceiro item é, na realidade, uma extensão do segundo mas eu o apresentei em separado para ressaltar sua importância. Na prática podemos pensar no Arduino como uma composição de *hardware* - a placa contendo o microcontrolador, periféricos e sistema de entrada e saída - e *software* - o IDE. Além dos itens citados, vale a pena destacar novamente a grande biblioteca de funções disponíveis para a execução das mais variadas tarefas comuns em programas dessa natureza. Essa biblioteca confere um enorme poder ao Arduino, permitindo que tarefas complexas sejam realizadas sem exigir esforço em demasia por parte do programador.

Agora, antes de detalharmos cada uma dessas partes, gostaria de falar um pouco sobre a concepção do Arduino e sobre as ideias que levaram à criação dessa plataforma.



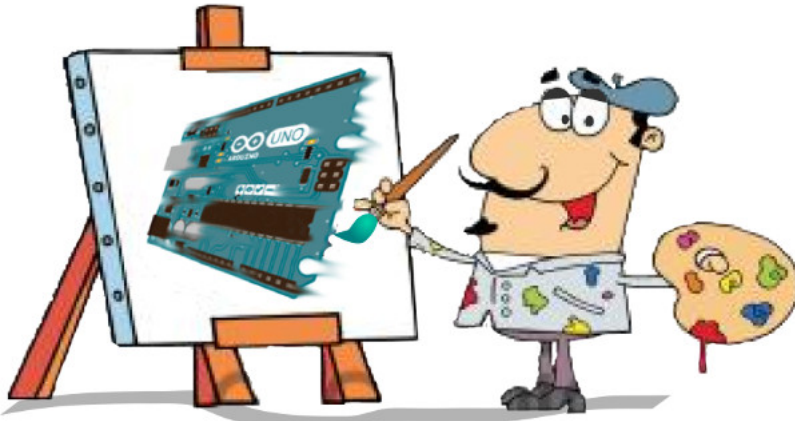
Um Arduino UNO

Arduino, o conceito

O Arduino foi desenvolvido na Itália por *Massimo Banzi, Barragan, David Cuartielles, Dave Mellis, Gianluca Marino e Nicholas Zambetti*. Segundo as palavras de *Massimo Banzi*, em seu livro - *Getting Started with Arduino* - editora **O'Reilly** - lhe foi dada a tarefa de ensinar um arcabouço mínimo de eletrônica para projetistas e artistas, de tal forma que pudessem expressar suas ideias e criações em termos de protótipos interativos. Cá entre nós, um desafio e tanto! Em seu livro, *Massimo Banzi* descreve a jornada empreendida para levar a cabo a tarefa, jornada essa que culminou no desenvolvimento da plataforma denominada Arduino. Portanto, do ponto de vista conceitual, Arduino é uma plataforma que prioriza a atitude criadora e experimental. Ela é perfeita para a criação de protótipos rápidos, possibilitando a verificação de resultados imediatamente porque permite criar, testar, modificar, testar novamente e voltar a modificar até que se obtenha o resultado desejado, sem grandes complicações e sem que se precise ter, obrigatoriamente, conhecimentos aprofundados de eletrônica ou programação. Essa é a ideia

central, de forma resumida e simplificada. Por conta de seu caráter, o Arduino é uma plataforma muito útil para ensino de eletrônica e programação, já que torna esse estudo muito mais instigante, desafiador e criativo. Além disso estimula no praticante a atitude experimental, base de toda a criatividade e espírito inovador. Mas é também adequada para qualquer tipo de profissional, mesmo artistas, que desejem explorar recursos tecnológicos em suas criações. Contudo, não se engane! O que acaba de ser dito de forma alguma significa que o Arduino seja uma plataforma de poucos recursos, super simplificada, útil apenas para pequenos projetos e estudantes. A plataforma pode ser - e de fato é - usada para aplicações profissionais de grande alcance e complexidade. Aliás, esse é o grande diferencial dessa plataforma: se você é estudante ou apenas tem paixão pelo tema, pode criar e obter resultados com extrema facilidade. Mas se é um técnico, profissional de engenharia, cientista, artista ou um aficionado com larga experiência, pode voar que o céu é o limite.

A título de exemplo, o hipotético caso do sistema que monitora o cão pode ser perfeitamente implementado usando um Arduino. E você, cara leitora ou leitor, há de concordar comigo que certamente não se trata de um sistema simplório.



O Arduino possibilita a livre expressão das ideias em termos de protótipos interativos, sem que se precise, obrigatoriamente, dominar profundamente conceitos técnicos. A plataforma convida à experimentação e torna o trabalho de desenvolvimento mais ágil, estimulante e, porque não, divertido!

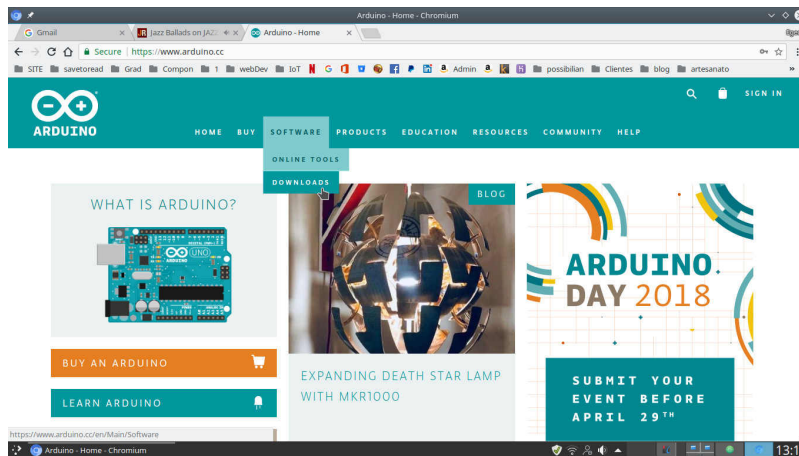
Arduino, a plataforma

Como eu já disse, o Arduino é composto de duas partes: uma peça de *hardware*, a placa contendo o circuito eletrônico do Arduino, e uma peça de *software*, o ambiente de desenvolvimento integrado, um programa de computador escrito numa linguagem chamada *Processing* - <https://www.processing.org> - que conta com todo o ferramental necessário ao trabalho de desenvolvimento dos programas, que na terminologia do Arduino são denominados *sketches*. Esse IDE, que pode ser baixado livremente no [site](#) do Arduino para ser instalado localmente ou pode ser acessado *on line*, conta com versões para Linux, Mac e Windows. Como eu já adiantei antes, existem algumas variantes de Arduino, como o UNO, o Mega, o Leonardo, mas nesse livro nos concentraremos no UNO. Se quiser conhecer mais modelos visite a [página de produtos](#). Portanto, é nesse ambiente que faremos nossos programas, ou *sketches* e

nossa primeira tarefa será a de fazer o *download* e instalar o ambiente. Faremos isso agora.

Instalação do IDE do Arduino

Acesse o site do Arduino, localizado em <https://www.arduino.cc> e clique no link *DOWNLOADS*.



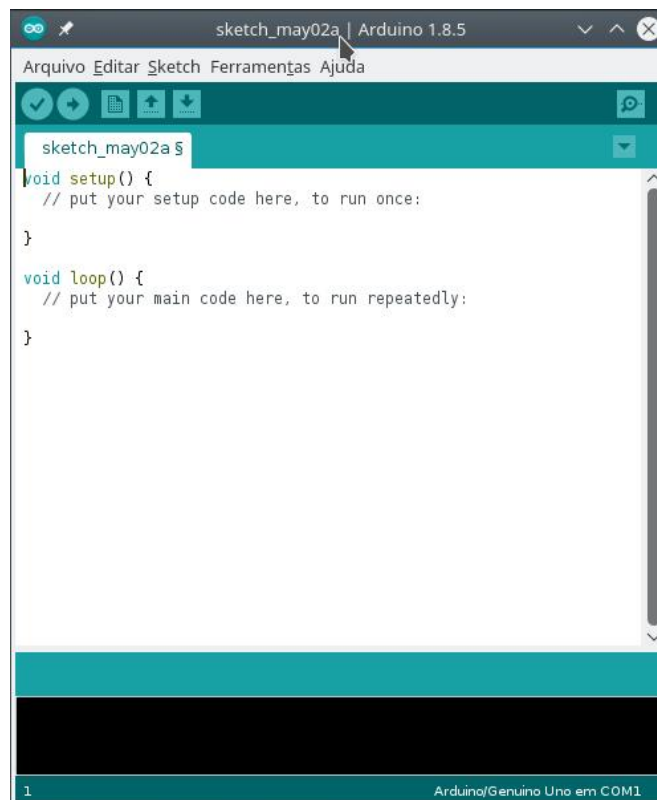
Na página que se abrirá você verá *links* para fazer o *download* de acordo com seu sistema operacional. Escolha o apropriado e baixe para sua máquina.

Download the Arduino IDE



Uma vez baixado o arquivo, descompacte se for o caso ou execute o instalador. Proceda normalmente, seguindo as instruções do instalador. A página <https://www.arduino.cc/en/Guide/HomePage> contém informações para ajudar em caso de necessidade.

Uma vez instalado o IDE podemos executar o programa e a maneira de se fazer isso depende do seu sistema operacional mas, normalmente, não constitui problema. Execute o programa da maneira usual, como está habituado a fazer com qualquer programa, e você terá a tela exibida a seguir.



A área branca abaixo dos menus é o editor de *sketches* e é ali que executamos toda a preparação de nossos programas. Como você pode notar, já existe um fragmento de código que é carregado automaticamente cada vez que executamos o ambiente. É uma ajudinha que o ambiente nos proporciona e trata-se de um esqueleto de código, apresentando os blocos básicos indispensáveis em qualquer programa que se escreva. Teremos muitas oportunidades de escrever programas utilizando esse arcabouço, mas por agora gostaria apenas de lhe pedir que olhasse

com atenção os dois blocos que aparecem lá e que estão apresentados a seguir:

```
void setup() {  
  // put your setup code here, to run once:  
}  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

São duas funções e se você não sabe o que é uma função, não se preocupe, teremos tempo para examinar isso. A primeira chama-se *setup()* e a outra *loop()*. Todo e qualquer *sketch* contém ao menos, e obrigatoriamente, essas duas funções. É por esse motivo que o ambiente já as coloca lá no editor. Veremos o que elas fazem depois porque agora vamos fazer um passeio pelos menus do ambiente, para que você possa se familiarizar com os principais recursos.

Menu Arquivo

Nesse menu temos alguns submenus que são comuns a qualquer programa e alguns que são específicos do Arduino. Passemos à descrição de cada um deles. Ao lado do nome de cada está indicado o atalho de teclado, quando existe.

Novo - Ctrl + N

Nesse submenu podemos iniciar um novo sketch. Você pode fazer isso clicando ou usando o atalho de teclado, Ctrl+N e, nesse caso, nem precisa clicar no menu Arquivo. Usar atalhos de

teclado aumenta enormemente a sua produtividade, não apenas aqui mas em qualquer tarefa no computador.

Abrir... - Ctrl+O

Aqui abrimos, ou carregamos, um *sketch* que foi salvo anteriormente e estava fechado. Você pode fazer isso clicando ou usando o atalho de teclado, Ctrl+O e, como disse antes, nesse caso nem precisa clicar no menu Arquivo.

Abrir Recente

Se você pousar o ponteiro do *mouse* sobre esse submenu verá aparecer, à direita dele, uma lista contendo os últimos *sketchs* nos quais trabalhou, em ordem cronológica, primeiro os mais recentes.

Sketchbook

Sketchbook é um alias para uma pasta que o Arduino cria automaticamente, denominada **Arduino**, em sua pasta de usuário. Como sempre isso depende do S.O. A pasta **Arduino** abriga os *sketches* que você cria, quando usa a opção Salvar, em sub pastas, como veremos logo. Caso não queira que seu *sketch* seja salvo no *Sketchbook* você pode usar a opção Salvar como...

Exemplos

Esse é um dos submenus mais quentes de todos, porque aqui temos uma enorme quantidade de programas prontos, agrupados por categorias: **Basics**, **Digital**, **Analog...** e muitos mais. Por exemplo, na categoria **Digital** temos nove exemplos, uma admirável fonte de estudo, sem mencionar que esses exemplos podem perfeitamente servir como ponto de partida

para seus programas. Ou podem ser usados, com adaptações se necessário, como parte de outros programas que você venha a criar. E por se tratar de código testado, seu uso aumenta a confiabilidade do *sketch* que o reutiliza.

Fechar - Ctrl + W

Fecha o IDE.

Salvar - Ctrl + S

Como acontece em qualquer programa de edição, aqui também temos a opção de salvar nosso trabalho, quer dizer, guardar o arquivo em uma pasta. Na primeira utilização desse submenu num novo *sketch* o sistema operacional lhe oferecerá uma caixa de diálogo, de tal forma que possa escolher a pasta onde deseja salvar o arquivo. A pasta sugerida pelo sistema é a apontada pelo alias, *Sketchbook*, por padrão, e caso você a aceite o sistema cria uma pasta identificada pelo nome que você escolher para o *sketch* e aninhada dentro da pasta **Arduino**. Cuidado aqui para não se confundir. A pasta no sistema de arquivos chama-se **Arduino** e não *Sketchbook*, que é apenas um alias para a pasta mesmo. Quando dermos uma olhada no submenu **Preferências** você verá como essa associação é configurada e entenderá perfeitamente como isso funciona.

Salvar como... - Shift + Ctrl + S

Se você não usa algum tipo de controle de versão, como **Git**, é comum sentir a necessidade de mudar o nome do arquivo ao realizar uma ação de salvar, sobretudo em programação. Isso ocorre amiúde, quando temos uma daquelas ideias “brilhantes”, tão brilhantes que o clarão não nos permite ter certeza do

resultado. Essa percepção, de que o desfecho da modificação pode não ser tão brilhante, nos faz desejar preservar o arquivo original, sem a modificação que está para ser introduzida. Nessa hora o Salvar como... vem em nossa ajuda, possibilitando criar um novo arquivo e manter o original, caso nos arrependamos. Procure guardar o atalho de teclado dessa opção, muito útil, assim como a do Salvar.

Configuração da página - Shift + Ctrl + P

Outra opção bem conhecida, permite configurar a página de maneira que sua impressora possa imprimir com a formatação adequada. Com certeza você já usou essa funcionalidade muitas vezes em seus programas prediletos de edição de textos e planilhas.

Imprimir - Ctrl + P

Abre a caixa de impressão do sistema operacional para que possa imprimir o código do *sketch*.

Preferências - Ctrl + ,

Nesse submenu podemos configurar uma série de aspectos do ambiente. Logo no alto da janela apresenta-se um campo indicando o **Local do Sketchbook**. Portanto, é aqui que acontece a associação da pasta real, por padrão denominada **Arduino**, com o alias, *Sketchbook*. Você é livre para alterar essa associação para uma pasta qualquer de sua conveniência. Em seguida temos campos para escolha de idioma, tamanho de fontes, exibição de mensagens de erro, avisos. É uma boa idéia marcar as caixas de verificação dessas opções relacionadas às mensagens de saída, assim como mudar a opção *Avisos do compilador* para *Todos*.

Marcar as caixas *Mostrar números de linhas* e *Habilitar Dobramento de Código* pode ser uma boa ideia quando se estiver lidando com programas longos. A opção *Habilitar Dobramento de Código* abre a possibilidade de recolher código dentro de funções, bastando clicar no - à esquerda de cada uma.

Menu Editar

Alguns submenus desse menu, como Desfazer, Refazer, Copiar, Colar, Localizar etc, com seus atalhos respectivos, não trazem qualquer novidade em relação aos comportamentos já conhecidos de outros aplicativos. Sigamos então para o primeiro que merece destaque, o submenu Copiar para Fórum.

Copiar para Fórum - Shift + Ctrl + C

A cópia de código usando esse recurso formata o mesmo da maneira adequada para a postagem, poupando tempo e trabalho. Selecione o código e use o atalho ou clique na opção. Depois basta colar no Fórum.

Copiar como HTML - Ctrl + Alt + C

HTML é a linguagem na qual as páginas de internet e os *e-books* são escritos e faz uso de *tags*. Essa opção formata o seu código de acordo com as regras do HTML, permitindo que seja usado para ser inserido no código de numa página de internet.

Comentar/Descomentar - Ctrl + /

Comentários ao programa aumentam a legibilidade, sendo prática corrente fazer comentários. Os comentários não são

processados pelo compilador, quer dizer, tecnicamente falando não fazem parte do programa, servindo apenas para nós, humanos, termos dicas sobre como funciona o código. Você deve comentar cuidadosamente seus programas, sob pena de ter dificuldades, você mesmo, de entendê-los no futuro! Acredite-me, isso acontece a todos os programadores. Comentários são feitos usando-se duas barras inclinadas para a direita `//`. Claro que para comentar uma só linha é mais prático usar diretamente a tecla de `/` no seu teclado. Mas caso precise comentar um trecho grande, selecione o trecho e use o recurso, comentando ou descomentando tudo de uma só vez.

Aumentar indentação Tab

Indentar é promover um recuo à direita no código. Isso ajuda a melhorar a legibilidade, por exemplo, dentro de funções. Se você der uma olhada no código que o Arduino cria em cada novo *sketch* perceberá que os comentários estão indentados.

Diminuir indentação - Shift + Tab

Experimente colocar o cursor no início de um comentário, prenda a tecla *Shift* e tecle *Tab*. Note como a indentação foi recuada à esquerda por um nível.

Menu Sketch

Esse menu será detalhado item por item, uma vez que seus submenus são exclusivos desse ambiente. Vamos lá:

Verificar/Compilar - Ctrl + R

O termo compilar já apareceu anteriormente sem que eu desse explicações sobre o que representa. Se você não sabe o que isso vem a ser, não se preocupe porque voltaremos a tocar nesse assunto mais adiante. Por hora gostaria apenas de colocar que compilar é converter seu código, escrito em linguagem de alto nível no *sketch*, para uma forma que o microcontrolador pode processar, denominada linguagem de máquina. A representação da linguagem de máquina é feita por sequências de zeros e uns, nada amigável para um humano. Portanto, usando esse submenu, ou teclando Ctrl+R, estamos solicitando ao ambiente que faça a compilação do programa, que então poderá ser carregado, da forma como veremos adiante, para a memória de programa do microcontrolador que está na placa do Arduino e que, por se tratar um Arduino UNO, é um Atmega328P.

Carregar - Ctrl + U

Compila e carrega o arquivo resultante, um binário correspondente ao código do programa traduzido para linguagem de máquina, na placa do Arduino conectada à porta usb. Quer dizer, quando você usa esse submenu não precisa do anterior, que apenas compila, sem carregar.

Carregar usando um programador - Ctrl + Shift + U

Essa opção deve ser usada com cautela, porque ela remove o *bootloader* do microcontrolador. Ainda é cedo para falarmos sobre *bootloader*, mas basicamente é um trecho de código pré instalado na memória de programa do microcontrolador e que permite que o ambiente carregue os programas no microcontrolador alojado na placa do Arduino usando o submenu Carregar, sem maiores dificuldades.

Exportar Binário compilado - Ctrl + Alt + S

O arquivo binário gerado pelo compilador, que traduz o nosso programa para linguagem de máquina, pode ser visto num formato mais compacto do que sequências de zeros e uns, codificado em notação hexadecimal. Aqui podemos exportar esse arquivo para a pasta do *sketch*, ficando disponível para ser usado, por exemplo, com outras ferramentas fora do ambiente.

Mostra a página do Sketch - Ctrl + K

Esse submenu abre o gerenciador de arquivos de seu sistema exibindo a pasta do *sketch* correspondente. É uma forma de abrir rapidamente o diretório do projeto.

Incluir Biblioteca

O Arduino conta com uma extensa Biblioteca de funções úteis que facilitam muito a elaboração de programas. A utilização dessa biblioteca evita que o programador tenha que ficar “reinventado a roda”, além de possibilitar a criação de código mais confiável, uma vez que as funções da biblioteca são testadas. Esse menu dá acesso a essas bibliotecas e também ao gerenciador delas.

Adicionar Arquivo...

Permite que se adicione um arquivo ao projeto. Por exemplo, código que você já tenha criado e esteja salvo em outro arquivo.

Menu Ferramentas

Esse menu também é específico do IDE, por isso veremos todos os itens.

Autoformatação - Ctrl + T

Essa opção realiza uma formatação automática no seu código, indentando o código de forma conveniente e melhorando a legibilidade do mesmo.

Arquivar Sketch

Cria um arquivo cópia do *sketch* ativo, compactando-o em formato ZIP e armazenando-o no mesmo diretório.

Corrigir codificação e recaregar

Corrige possíveis discrepâncias que possam haver entre caracteres de diferentes sistemas operacionais devido às diferenças nas codificações desses caracteres.

Monitor serial - Ctrl + Shift + M

Abre a janela do monitor serial, que permite visualizar dados trocados entre o Arduino e o computador. Dependendo das características da placa do Arduino, abrir o terminal reseta a mesma. A utilidade desse monitor ficará clara à medida em que avançarmos no estudo.

Plotter serial - Ctrl + Shift + L

Abre a janela da *plotter* serial, que permite visualizar os mesmos dados trocados entre o Arduino e o computador, mas dessa vez plotados num gráfico com eixos ortogonais.

WiFi101 Firmware Updater

A placa *WiFi101* permite que o Arduino se conecte à internet usando a biblioteca *WiFi101*, similar à biblioteca *Ethernet*. Esse menu possibilita a atualização do *firmware* dessa placa.

Placa

Como eu já citei, existe uma grande variedade de modelos de Arduino, variedade essa que cresce sem parar. Portanto, nesse sub menu temos a possibilidade de informar ao ambiente qual é o modelo que estamos usando.

Porta

Possibilita a escolha de uma porta específica para a conexão do Arduino.

Obter informações da Placa

Exibe informações sobre a sua placa, como por exemplo, o número de série.

Programador

Você pode usar um outro programador para programar o *chip* do microcontrolador em vez do Arduino. Nesse caso é preciso informar ao ambiente de que *hardware* se trata e isso é feito nessa opção.

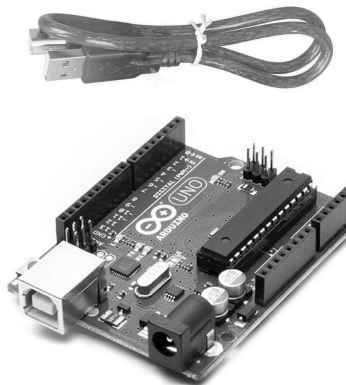
Gravar *Bootloader*

Quando você compra uma placa de Arduino, o microcontrolador alojado na placa já vem pré gravado com o *bootloader*. Contudo,

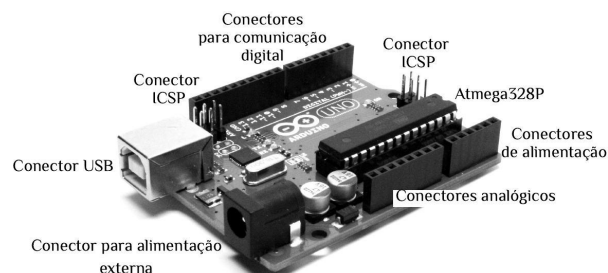
se a placa usa soquete nada lhe impede de trocar o microcontrolador por um novo, digamos, em caso de defeito no original. Mas normalmente, quando você compra um novo *chip* ele não vem gravado com um *bootloader* e, nesse caso, é preciso gravar um. Essa tarefa pode ser levada a cabo usando esse submenu.

Conectando o Arduino

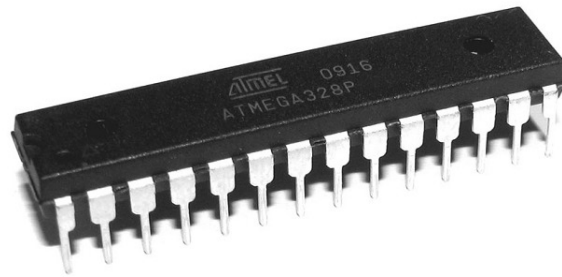
Como dito antes, nosso estudo será baseado no Arduino UNO. A figura mostra novamente esse modelo e o cabo usb que normalmente o acompanha.



A seguir vemos a imagem da placa, com a identificação dos principais elementos que a compõem.



Na figura a seguir vemos o microcontrolador usado pelo UNO, um Atmega328P.



Depois de tudo o que foi visto até o momento, acho que fica claro que o Arduino representa muito mais do que pode ser posto em poucas palavras. E também deve estar evidente para você que não se trata de um microcontrolador, mas de uma plataforma de desenvolvimento, composta por uma série de modelos de placas (tanto originais quanto clones de desenvolvedores independentes, já que é uma plataforma *open source*), um microcontrolador, um ambiente de desenvolvimento com linguagem própria e uma extensa biblioteca de apoio. Além disso, mas nem de longe menos importante, conta com toda uma comunidade ativamente engajada e participante, um site oficial repleto de informações e muitos outros dedicados ao assunto. Arduino é tudo isso e crescendo!

Passaremos agora à conexão da placa ao computador. Siga os passos indicados e tudo correrá a contento:

1. Conecte o cabo usb que acompanha seu UNO ao soquete usb da placa do Arduino.
2. Conecte o outro conector do cabo a uma porta usb livre em seu computador.

No momento em que você realiza a conexão o *led* ON , na placa, acende e o outro, marcado com L, começa a piscar. O *led* ON aceso significa apenas que a placa foi energizada. Mas o motivo pelo qual o outro *led* pisca é o fato de que a memória de programa do UNO vem pré carregada, de fábrica, com um pequeno programa de

demonstração, que instrui o microcontrolador a acender e apagar o *led* com frequência de 0,5 Hz. Perceba que o *led* fica um segundo aceso e um segundo apagado, repetindo esse ciclo indefinidamente enquanto a placa estiver energizada. Muito bem, sua placa está conectada e funcionando perfeitamente mas, contudo, seu sistema operacional ainda não está configurado. Se você abrir agora o ambiente de desenvolvimento provavelmente não conseguirá fazê-lo se conectar à placa ainda. A configuração depende do sistema operacional que você está usando e daremos a seguir as instruções para usuários do Windows®, uma vez que a maioria das pessoas em nosso país ainda usa esse sistema. Caso você seja, como eu, usuário do Linux, faça uma visita a essa página: <http://playground.arduino.cc/Learning/Linux>. Nela você encontra instruções detalhadas sobre como configurar seu Linux.

Bem, prosseguindo com a instalação no Windows®, há duas possíveis situações:

1. O Windows detecta e instala automaticamente os drives para o Arduino, atribuindo uma porta **COM** ao mesmo. É o caso do Windows Vista e também do Windows 7, pelo menos até onde pude testar. Você pode verificar qual porta **COM** foi atribuída ao Arduino indo ao **Painel de Controle** e acessando o **Gerenciador de Dispositivos**. Nesse caso, basta abrir agora o IDE do Arduino, acessar o menu **Ferramentas - Porta** e escolher a **COM** que foi atribuída à placa. Pronto, seu Arduino está configurado para ser usado.
2. O sistema detectou o novo *hardware* e deu início ao processo de instalação mas, lamentavelmente, a instalação falha em algum momento do processo. Faça então o seguinte:
 1. Abra o **Painel de Controle** e navegue até **Sistema e Segurança**.

2. Clique em **Sistema** e abra o **Gerenciador de Dispositivos**.
3. Localize **Portas (COM&LPT)**. Você deverá encontrar algo como **Arduino UNO (COMxx)**. Caso não encontre, verifique em **Outros Dispositivos** à procura de um dispositivo desconhecido.
4. Clique com o botão direito do *mouse* em **Porta "Arduino UNO (COMxx)"** e escolha **Atualizar Driver de Software**.
5. Agora escolha a opção **"Navegar nas pastas à procura de Driver de Software"**.
6. Por fim, navegue e escolha o **driver "arduino.inf"** localizado na pasta *Drivers* do Arduino. Agora o Windows® deverá finalizar a instalação do *Driver*, corretamente.

OceanofPDF.com

Capítulo 2

Programar é para todos!

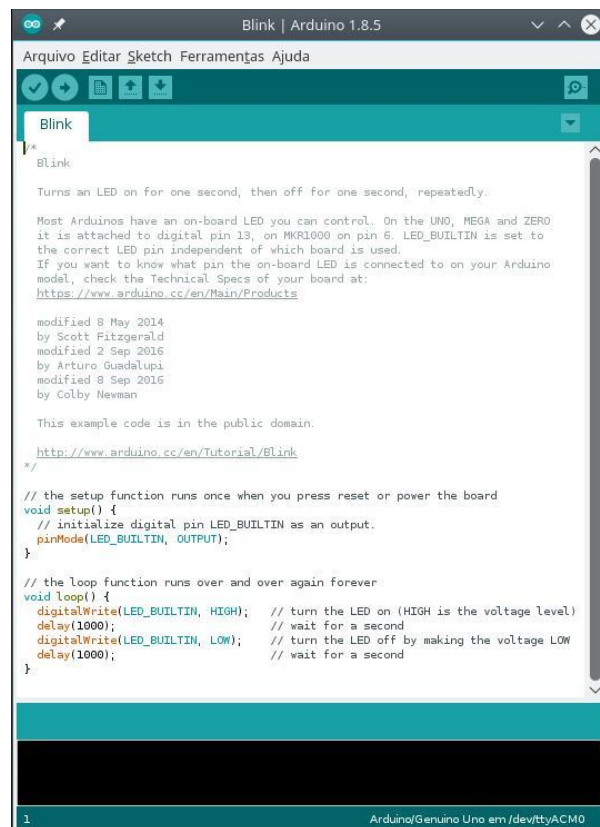
A atividade de programar já foi um ofício para bem poucos e nos primeiros tempos da programação muitos eram os obstáculos: pouca informação disponível, pouco intercâmbio entre os programadores, ambientes de programação - quando já havia algum - com poucos recursos. E poucas linguagens de programação, muitas nada amigáveis. O panorama é outro, bem diverso, hoje. Internet, *help on line*, ambientes de programação inteligentes, linguagens a cada dia mais amigáveis, inclusive gráficas, plataformas de desenvolvimento são alguns dos recursos disponíveis para quem se dedica à atividade. Mesmo assim, muitos conhecimentos ainda são exigidos por parte de quem se dispõe a criar programas e a gama e profundidade do preparo do programador ou da programadora estão diretamente relacionadas aos requisitos que o programa deve atender. Formalmente falando, antes de iniciar a programação de qualquer coisa, quem almeja tornar-se programador ou programadora deveria aprender uma ou mais linguagens, estudar algoritmos, estruturas de dados, fazer exercícios visando

desenvolver raciocínio lógico, aprender um bocado de matemática. Teoricamente é assim, teoricamente esse é o método ideal. Teoricamente...

Como mencionei no início, a proposta do Arduino é possibilitar que pessoas com pouco ou nenhum treino em programação possam expressar ideias as mais variadas, inclusive artísticas, por meios interativos e se valendo da eletrônica e da programação de dispositivos. Mas isso não significa que seja possível criar algo em termos de programas sem, ao menos, um embasamento primário, mínimo, sobre a atividade de programação. Quer dizer, alguns conceitos básicos precisam ser compreendidos antes que as linhas de um programa possam fazer sentido para alguém, por mais simples que seja o programa. Temos dois caminhos possíveis aqui: o tradicional, que importaria numa série de capítulos envolvendo os tópicos usuais e comuns nos livros de programação ou, então, um menos ortodoxo, mais “à moda Arduino”, mais interativo, no qual os conceitos são colocados à medida em que são necessários. Essa será a nossa vereda, não apenas por apresentar uma afinidade evidente com o espírito com que a plataforma foi concebida, como pelo fato de que esse livro está dirigido a um público o mais amplo possível.

Quando você conectou a placa do seu Arduino à porta usb do seu computador, o *led* da placa começou a piscar, com intervalos de um segundo entre cada comutação. Naquele momento eu disse que havia um programa de demonstração, instalado pela fábrica, na memória de programa do microcontrolador que está alojado na placa. Em outro momento, quando eu descrevi os menus, mencionei um muito útil sobretudo a quem está começando a aprender. O submenu **Exemplos**, alojado no menu **Arquivos**. Lá, em

01.Basics, existe um exemplo chamado **Blink**. Pois bem, esse é o *sketch* que está carregado na memória de programa do seu Arduino e é por ele que iniciaremos o estudo de programação. Abra o seu IDE, clique em **Arquivos, Exemplos, 01.Basics** e por fim *Blink*. Observe como uma nova janela do Arduino se abre, exibindo o *sketch* correspondente ao **Blink**. A figura a seguir mostra a janela do editor do Arduino carregada com o *sketch*.

The image is a screenshot of the Arduino IDE interface. The title bar at the top reads "Blink | Arduino 1.8.5". Below the title bar is a menu bar with "Arquivo", "Editar", "Sketch", "Ferramentas", and "Ajuda". Under the "Sketch" menu, a dropdown list shows "Blink" as the selected option. The main text area contains the following text: "Blink", "Turns an LED on for one second, then off for one second, repeatedly.", a paragraph explaining that most Arduinos have an on-board LED and providing a link to the technical specifications, a list of modification dates and authors, a statement that the code is in the public domain, another link, and the C++ code for the sketch. The code includes comments in Portuguese and uses color-coded syntax. At the bottom of the window, a status bar shows "1" on the left and "Arduino/Genuino Uno em /dev/ttyACM0" on the right.

```
Blink

Turns an LED on for one second, then off for one second, repeatedly.

Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
the correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your Arduino
model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products

modified 8 May 2014
by Scott Fitzgerald
modified 2 Sep 2016
by Arturo Guadalupi
modified 8 Sep 2016
by Colby Newman

This example code is in the public domain.

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

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

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

Antes de partirmos para a análise do código, vamos nos divertir e aprender ao mesmo tempo. Já sabemos que, de alguma forma, esse programa instrui o microcontrolador a mudar o estado do *led* da placa a intervalos de um segundo. Antes de entendermos como isso é feito, vamos mudar a frequência das piscadas. Que tal aumentar essa frequência, digamos para o dobro. Atualmente a frequência equivale a 0,5 Hertz e queremos aumentá-la para 1 Hz.

Isso fará com que o *led* pisque com o dobro da velocidade. Ora, raciocinemos um pouco: aumentar a velocidade das piscadas significa reduzir o intervalo entre as mudanças de estado do *led*, de acordo? Se desejamos dobrar a velocidade, é razoável admitir que reduzir pela metade o tempo entre comutações deve levar ao resultado almejado. Veja abaixo o código do *sketch*, ainda na forma original. Aqui, traduzi os comentários ao programa para nosso idioma, de forma a facilitar o entendimento. E por falar em comentários, note que as linhas acima do código próprio dito também são comentários usando o estilo `/*` e `*/` que permite comentar múltiplas linhas de código de uma só vez.

```
/*
```

```
Blink
```

```
Turns an LED on for one second, then off for one second, repeatedly.
```

```
Liga um LED por um segundo, e o desliga por um segundo, repetidamente.
```

```
Most Arduinos have an on-board LED you can control.
```

```
On the UNO, MEGA and ZERO it is attached
```

```
to digital pin 13, on MKR1000 on pin 6.
```

```
LED_BUILTIN is set to the correct LED pin
```

```
independent of which board is used.
```

```
If you want to know what pin the on-board LED
```

```
is connected to on your Arduino model,
```

```
check the Technical Specs of your board at:
```

```
https://www.arduino.cc/en/Main/Products
```

A maior parte dos Arduinos tem um LED on-board que você pode controlar. NO UNO, MEGA e ZERO ele está conectado ao pino digital 13, no MKR1000 ao pino 6. LED_BUILTIN está configurado para o led correto independentemente de qual placa você está usando. Se você quiser saber em que pino o LED

está conectado em seu Arduino, verifique
as Especificações Técnicas de sua placa em:
<https://www.arduino.cc/en/Main/Products>

modified 8 May 2014

by Scott Fitzgerald

modified 2 Sep 2016

by Arturo Guadalupi

modified 8 Sep 2016

by Colby Newman

This example code is in the public domain.
Esse exemplo encontra-se em domínio público.

<http://www.arduino.cc/en/Tutorial/Blink>

*/

```
// the setup function runs once when  
// you press reset or power the board  
// a função setup roda sempre que há  
// um reset ou energização da placa.
```

```
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  // inicializa o pino digital LED_BUILTIN como saída.  
  pinMode(LED_BUILTIN, OUTPUT);  
}
```

```
// the loop function runs over and over again forever  
// a função loop executa repetidamente sem parar.  
void loop() {  
  // liga LED (HIGH - nível alto de tensão)  
  digitalWrite(LED_BUILTIN, HIGH);  
  // aguarda um segundo.  
  delay(1000);
```

```
// desliga LED tornando a voltagem LOW, baixa.  
digitalWrite(LED_BUILTIN, LOW);  
// aguarda um segundo.  
delay(1000);  
}
```

A chave para alterarmos o tempo entre comutações no estado do *led* está na função *delay()*. Ela recebe um valor para seu argumento, aquilo que vai entre parênteses, igual a 1000. *Delay* é atraso, em inglês. Como você provavelmente sabe, as linguagens de programação usam o idioma inglês e, embora uma função não faça parte das palavras reservadas da linguagem e possa ter qualquer nome, em qualquer idioma, aqui as funções também estão em inglês. Essa função promove um atraso, ou dizendo de outra forma, perde tempo.

— Mas quanto tempo ela perde? Mil? Mas mil o quê? Dias, semanas?
— Mil milissegundos.

A função *delay()* está implementada para receber valores em milissegundos, guarde isso ou anote aí. Bem, 1000 milissegundos representam um segundo.

— Ahá! Então essa função faz com que o programa pare por um segundo, sem fazer nada?

Exato! Cada vez que o fluxo de programa atinge uma função *delay()*, o processamento entra num laço no qual nada acontece durante o tempo equivalente ao número de milissegundos que a função recebeu como argumento. Portanto, em nosso *sketch*, há duas paradas de um segundo cada. Numa delas o *led* está aceso e na outra está apagado. Daqui a pouco você vai entender porque ele muda de estado entre um *delay()* e outro, mas agora vamos alterar o código e ver o Arduino promover a mudança no comportamento do *hardware*. Edite o valor que está entre parêntes na função

`delay()` para tornar-se 500. Faça isso nas duas ocorrências da função. Depois, com seu Arduino conectado ao seu computador via porta USB, vá ao menu **Sketch** e clique em **Carregar**, ou simplesmente use o atalho de teclado, Ctrl+U. Observe na placa do Arduino que os dois *leds* marcados com TX e RX vão mudar de estado rapidamente, várias vezes. Esses *leds* sinalizam troca de dados entre a placa e seu programa, significando que o programa na memória do microcontrolador está sendo atualizado. Não demorará e você verá o *led* L começar a piscar novamente e mais rapidamente do que antes, exatamente com o dobro da frequência, como determina o programa agora.

Parabéns! Ao realizar esses passos você completou uma rotina que irá se repetir muito daqui para frente, qual seja: editar um programa, carregá-lo na memória do microcontrolador e vê-lo entrar em funcionamento novamente, exibindo o novo comportamento.

Antes de prosseguirmos, altere novamente os valores dos argumentos que são passados ao parâmetro da função `delay()`. Note que os valores não necessariamente tem que ser iguais nas duas ocorrências da função, podendo ser diferentes. Experimente fazer com que o valor na primeira ocorrência seja o dobro do valor na segunda, por exemplo, 2000 para a primeira ocorrência e 1000 para a segunda. Edite o código, recarregue o programa e observe o comportamento do sistema. Pelo resultado, sem tornar a ler os comentários ao programa, você é capaz de dizer qual parte do código liga o *led* e qual parte desliga? Procure responder a esse desafio antes de prosseguir na leitura. E lembre-se: em programação, a atividade de pesquisa e análise é fundamental para o bom desempenho da função, mas você não deveria se angustiar

caso não possa, ainda, chegar a uma conclusão. Peter Norwig, uma das maiores autoridades em inteligência artificial do nosso planeta, defende que aprender a programar com perícia não demanda menos do que 10 anos de estudo.

O fragmento do programa modificado está mostrado abaixo.

```
// liga LED (HIGH - nível alto de tensão)
digitalWrite(LED_BUILTIN, HIGH);
// aguarda dois segundos.
delay(2000);
// desliga LED tornando a voltagem LOW, baixa.
digitalWrite(LED_BUILTIN, LOW);
// aguarda um segundo.
delay(1000);
```



A função *delay()* recebe um argumento em milisegundos. Anote aí.

Se você fez a experiência percebeu que agora o *led* passa mais tempo aceso do que apagado. Na verdade, passa dois segundos aceso e um segundo apagado. Vamos analisar juntos o código, uma linha antes e uma depois da primeira ocorrência da função *delay()*.

Note que fizemos o argumento da primeira ocorrência da função valer 2000, equivalendo a dois segundos de atraso. A linha de código imediatamente anterior ao primeiro *delay()* é:

```
// liga LED (HIGH - nível alto de tensão).
digitalWrite(LED_BUILTIN, HIGH);
```

digitalWrite() também é uma função, assim como *delay()*. Ambas são funções da biblioteca do Arduino. Note que *digitalWrite()* recebe dois argumentos em seus parâmetros funcionais, LED_BUILTIN e HIGH, porque essa função está implementada para receber o número de um dos pinos de I/O da placa e um estado para esse pino. O que ela faz então é comutar o pino correspondente para o estado passado para a função. Trocando em miúdos, quando essa instrução é executada o pino 13 da placa - 19 no microcontrolador - vai para estado alto, fornecendo 5 Volts para o que quer que esteja conectado ao pino. Nesse caso, o *led*, que então acende. Executada essa instrução, o programa passa à instrução seguinte, porque é dessa maneira que todo programa funciona, executando uma instrução após a outra. E a seguinte é um *delay()* que obriga o processamento a aguardar dois segundos. Ora, então significa dizer que o *led*, que acaba de ser ligado, vai passar dois segundos aceso, veja.

```
// aguarda dois segundos.  
delay(2000);
```

Transcorrido esse tempo, o programa executa a instrução a seguinte:

```
// desliga LED (LOW - nível baixo de tensão).  
digitalWrite(LED_BUILTIN, LOW);
```

Por analogia e sabendo que LOW significa baixo, já dá para imaginar o que essa instrução faz. Se você disse apaga o *led*, acertou, é isso mesmo. Mas não perca de vista o curso dos acontecimentos, vamos recapitular: o *led* foi aceso, transcorreram dois segundos e ele foi

apagado. Vejamos o que acontece em seguida, analisando a próxima instrução:

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

Mais um atraso de um segundo, correto? Significa que o *led* fica apagado por um segundo. Mas e depois? O que você acredita que acontece? Você sabe, você viu o programa executando no seu Arduino. O processo recomeça, magicamente. Mas porque? Onde está a próxima instrução? Porque o processamento não parou, já que atingimos a última instrução do programa? Observando novamente a listagem completa do programa, vemos que todas as instruções que acabamos de analisar estão encerradas num bloco de programa. Esse bloco começa com:

```
void loop() {
```

e termina com:

```
}
```

Acontece que *loop()* também é uma função e desempenha um papel crucial em todo programa, em parceria com *setup()*. São tão importantes que o Arduino já as coloca em cada novo *sketch* automaticamente, poupando nosso trabalho. Tudo o que estiver dentro da função *loop()*, entre o primeiro abre chaves e o último, será executado repetidamente, sem parar, enquanto o sistema estiver energizado. E isso responde à nossa pergunta. Não parou porque, ao atingir a última instrução, a execução do programa é desviada para a primeira instrução dentro do bloco *loop()* e tudo

recomeça. Por isso o *led* continua piscando sem parar, respeitando os tempos entre as comutações. Todo programa para microcontrolador roda dentro de um laço, ou *loop()* e isso não acontece apenas no Arduino, mas em qualquer programa para qualquer microcontrolador.

Estamos quase terminando a dissecação de nosso exemplo, falta falar sobre a função *setup()*. Veja o fragmento de código que mostra a função:

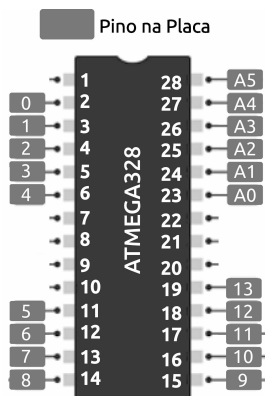
```
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  // inicializa o pino digital LED_BUILTIN como saída.  
  pinMode(LED_BUILTIN, OUTPUT);  
}
```

Um microcontrolador é um circuito complexo, dotado de recursos variados. Internamente, um microcontrolador típico contém áreas de memória, circuitos dedicados que executam tarefas eletrônicas que são comuns nos projetos, como temporizadores, geradores de PWM etc. Além disso, todo microcontrolador tem um sistema de entrada e saída, comumente referido como I/O. As características variam de modelo para modelo e de fabricante para fabricante, mas todo microcontrolador tem um núcleo básico de recursos que está presente em qualquer modelo. Sistema de I/O é um exemplo, afinal de nada serviria um microcontrolador que não pudesse se comunicar com o exterior. Pois bem, em qualquer programa a primeira tarefa do programador é a de configurar os recursos do microcontrolador em questão. Isso inclui definir que pinos serão usados como pinos de entrada de dados e quais serão usados como pinos de saída de dados. Além disso, há pinos que podem ser

usados com sinais digitais ou analógicos e isso também precisa ser definido. Essa é a tarefa da função *setup()*. Em seu bloco de código é que são feitas tais definições e é por essa razão que ela precisa estar presente em qualquer programa, sob pena de o sistema não funcionar, ou pior ainda, fazê-lo de forma errática. Note que, em alguns casos extremos, configurações incorretas podem até causar danos ao microcontrolador. Observe o código dentro do bloco da função *setup()*. O comentário já diz tudo:

```
//inicializa o pino digital LED_BUILTIN como saída.
```

Portanto, o autor do programa está, aqui, configurando o pino 13 como pino de saída - OUTPUT. O pino 13 só pode ser usado como pino digital, pela própria constituição do UNO. Note que usando a constante LED_BUILTIN você não precisa saber qual é o pino na placa que corresponde ao *led on board*. Quero aproveitar o ensejo para fazer uma observação: o pino 13 da placa do UNO não corresponde ao pino 13 do microcontrolador, mas ao 19. Isso é muito importante, não existe correspondência numérica entre os pinos da placa e os pinos do microcontrolador. A figura a seguir ilustra, apenas a título de informação complementar, a correspondência entre os pinos do microcontrolador e a pinagem na placa. Note que, em geral, você não precisa se preocupar com a pinagem no *chip*, bastando se ater à pinagem na placa, mesmo quando utiliza *leds* ou outros periféricos externos porque eles estarão conectados à pinagem da placa e não diretamente ao *chip*.



Ampliando o horizonte

Ao longo de nossa jornada até aqui mencionei, em um ou outro momento, termos e conceitos que agora devemos clarificar um pouco. Não nos aprofundaremos muito porque, a rigor e nessa fase inicial de seu estudo, não há necessidade e talvez nem seja conveniente. Falarei um pouco sobre eles para evitar que lhe fique uma sensação de estar perdendo algo e, também, lhe fornecer algum subsídio caso queira buscar aprofundamento em outras fontes de consulta, o que lhe estimulo a fazer mesmo. Além disso, ao final desse volume você encontra um [glossário](#) contendo uma relação de termos técnicos que aparecem ao longo do texto e que podem ocasionar alguma dúvida. Consulte se precisar.

Linguagem de programação

Bem lá no início da computação, a programação era feita ajustando-se várias chaves de duas posições em determinadas combinações. Depois surgiu a programação diretamente em linguagem da máquina, sequências de zeros e uns. O passo seguinte foi a criação de mnemônicos para as instruções em linguagem de máquina e esse tipo de linguagem, usada até hoje,

ficou conhecida como *Assembly*, classificada com linguagem de baixo nível porque se comunica diretamente como o *hardware*. Mas por ser uma forma de programar muito desgastante e sujeita a erros, prejudicando em muito a produtividade, o *Assembly* evoluiu para linguagens mais amigáveis, chamadas linguagens de alto nível. Para nosso conforto e produtividade, são muitas as linguagens de programação disponíveis atualmente.

Uma linguagem de programação é um conjunto de estruturas e palavras especiais, denominadas palavras reservadas, e que podem ser usadas para que se componha conjuntos de instruções que serão executadas pelo dispositivo para o qual se programa, que pode ser um computador, um celular, um microcontrolador, um CLP etc. Entre as linguagens mais proeminentes para a programação em geral estão C, C++, Java, Python, entre tantas outras. Para microcontroladores são comumente usadas C e suas derivadas e *Assembly*.

Compilador

Esse é um termo que pode ser intrigante para quem nunca programou. Compilar, no contexto da programação, é converter código que foi escrito em uma linguagem de programação qualquer para código que pode ser executado em uma máquina, também referido como binário executável.

Um mesmo programa pode ser escrito em muitas linguagens mas, para ficarmos no âmbito da programação de microcontroladores, duas linguagens são comumente usadas, o *Assembly* e o C, como já dissemos. Note que nenhuma das duas é usada pelo Arduino, que usa um tipo de C++. Mas nada lhe impediria de programar para um

Atmega, sem fazer uso do ambiente do Arduino, diretamente em Assembly. Claro que você precisaria conhecer a linguagem. De outra forma, poderia também programar em C. Veja, então, que é possível programar para o mesmo dispositivo em diversas linguagens mas que essa comodidade só existe porque, ao final do processo, existe um compilador, um programa que traduz, ou dizendo de forma mais técnica, compila o código escrito numa dada linguagem para a linguagem de máquina específica do microcontrolador em questão. O microcontrolador não entende C ou Assembly ou qualquer outra linguagem, entende apenas linguagem de máquina, sequências de zeros e uns e é papel do compilador converter o programa escrito em uma dada linguagem para a linguagem de máquina.

Do que foi dito, fica claro que para cada tipo de microcontrolador existe um compilador dedicado. Da mesma forma, um compilador que compila C para um determinado microcontrolador, pode não ser capaz de compilar outra linguagem. Por fim, o processo de compilar é complexo, envolvendo o *link* de bibliotecas, alocações de memória etc. Assunto para um livro inteiro.

Bibliotecas ou *Librarys*

Mesmo fazendo o uso de uma linguagem sofisticada para escrever um programa, há uma série de tarefas que serão tediosas e complicadas de se programar, tarefas em geral rotineiras e repetitivas. Um exemplo comum é termos de fazer com que nosso programa envie dados para o sistema de I/O, quando precisa se comunicar com o mundo externo ao microcontrolador. Ou quando precisamos acionar um temporizador interno ao microcontrolador. Via de regra, realizar esses tipos de tarefas envolve um bocado de

código e de conhecimento do *hardware* envolvido. Pensando nisso, programadores preparam rotinas especialmente elaboradas para realizar essas tarefas repetitivas e muito úteis, permitindo que façamos uso de tais funcionalidades de forma muito simplificada. Essas rotinas aparecem na forma de funções e são colocadas na biblioteca padrão que, em geral, é instalada junto com o ambiente de desenvolvimento. A integração entre essas funções e a linguagem é tão intensa que é comum confundirmos a linguagem com a biblioteca de funções. Tenha em mente, contudo, que as funções que usamos, provenientes da biblioteca, não são, rigorosamente falando, parte de linguagem. Em nosso primeiro exemplo tivemos oportunidade de lidar com algumas funções da biblioteca padrão do Arduino, como *digitalWrite()*, *setup()* e *loop()*. Graças ao uso delas foi fácil lidar com o programa e fácil entender quem faz o que. Além disso, graças a elas os programas tornam-se muito mais curtos do que seriam sem o uso da tais bibliotecas. Haverá programas nos quais teremos que solicitar, no início do programa, a inclusão de bibliotecas que não estão disponíveis por padrão para o sistema. E você poderá, no futuro, até criar suas próprias funções e disponibilizá-las na biblioteca, contribuindo para a comunidade de programadores.

Bootloader

Outro termo hermético. Mas não é difícil entender do que se trata e para que serve. Como eu já disse, os compiladores geram código binário executável, que precisa ser colocado na memória de programa do microcontrolador. Bem, o compilador está rodando em seu computador e o código binário gerado está armazenado em seu disco rígido, ou em qualquer outra mídia. Para que se possa realizar a transferência desse binário entre seu computador e a

memória do microcontrolador pode-se usar um circuito eletrônico especialmente desenhado para isso, denominado gravador de programas. Esse gravador terá de ser compatível com o microcontrolador em questão e deverá ser controlado por um *software* dedicado, instalado no computador hospedeiro. Nesse caso, não há necessidade de *Bootloader*, o gravador se encarrega da operação, bastando que você coloque o *chip* do microcontrolador no soquete do gravador e execute o programa de gravação no seu computador. Mas existe uma outra opção, usada pelo Arduino, que possibilita essa transferência bastando que haja um sistema de comunicação entre o seu computador e o microcontrolador e um programa que já está instalado na memória do próprio microcontrolador e que gerencia essa transferência. No nosso caso, esse sistema de comunicação usa a porta usb e se vale de um *chip* na placa do Arduino dedicado a isso. E o programa gerenciador da transferência é denominado *Bootloader*. É ele que torna possível à placa do Arduino transferir nossos *sketches* para a memória de programa do Atmega328P. Sem um *Bootloader*, seria necessário o uso de um gravador de programas compatível com os microcontroladores da Atmega®. Portanto, graças ao *Bootloader*, instalado pelo fabricante na memória do Atmega328P no momento da montagem do Arduino, não precisamos de um gravador externo, ficando a tarefa a cargo do próprio Arduino. Isso torna o processo de programar e testar muito mais ágil e menos sujeito a erros e falhas.

Depuração

Depuração é o processo de testar e, eventualmente, corrigir erros, também conhecidos como *bugs*, nos programas. Os erros podem ser, de modo geral, de dois tipos: lógicos ou de sintaxe. Os de

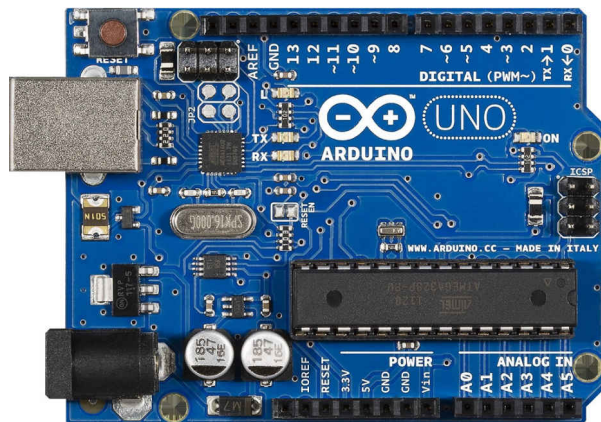
sintaxe são provenientes de digitação incorreta, troca de letras, coisas assim e são acusados pelo próprio compilador, sendo por isso de fácil solução. Já os de lógica, esses são mais complicados porque o compilador não tem meios de saber que você cometeu um erro lógico. Por exemplo, se você deseja que o *led* pisque a uma frequência de 10 Hz mas, por descuido, codificou para 100 Hz ou numa tomada de decisão inverteu as instruções, isso são erros lógicos e somente você mesmo pode percebê-los. Esse tipo de erro é mais difícil de detectar, sobretudo se o código for longo. Nesse momento, os comentários ao programa podem ser decisivos porque é comum que se passem semanas, até meses, ao longo da preparação de um programa. Não espere que vá se lembrar do motivo pelo qual codificou determinado trecho dessa ou daquela maneira, 45 dias depois de tê-lo feito. Comente tudo e os dividendos virão quando precisar ler seu próprio código em busca de erros lógicos. Sem mencionar que outras pessoas podem precisar ler seu programa e é boa prática comentar o código, facilitando a vida dos outros programadores e a sua mesma. Comente, sempre.

OceanofPDF.com

Capítulo 3

Recursos do Arduino. E para ele!

De modo a poder tirar proveito da plataforma Arduino, é crucial que você conheça os recursos que estão disponíveis nela e para ela. São muitos e entre eles está o sistema de I/O, fundamental em qualquer programa, por mais elementar que seja. Observe na imagem a seguir a placa do Arduino UNO, vista por cima.



Acompanhe pela imagem a descrição da pinagem.

Pinos 0 a 13

Esses são pinos de comunicação - I/O - digital. Podem ser configurados em seu *sketch* para atuarem tanto como pinos de entrada - INPUT - quanto como de saída - OUTPUT. Lembra do conteúdo da função *setup()*, no primeiro exemplo que analisamos? Dê uma olhada lá, na função *digitalWrite()*, para refrescar e fazer a conexão.

Pinos A0 a A5

Esses pinos podem atuar como pinos de entrada de dados analógicos e seu funcionamento ficará claro à medida em que avançarmos.

Pinos 3, 5, 6, 9, 10, e 11

Pertencem ao conjunto dos pinos digitais, mas podem ser configurados em seu *sketch* para funcionarem como pinos de saída analógica.

Analógico versus Digital

Se você já lida habitualmente com eletrônica, conhece bem o significado dos termos analógico e digital. Mas pode não ser seu caso e como a distinção e significado desses termos é de fundamental importância para nosso tema vamos fazer aqui uma pequena digressão e discutir brevemente o assunto.

Dizemos que uma grandeza é analógica quando podemos fracionar essa grandeza o quanto desejarmos. Por exemplo, temperatura é uma grandeza analógica. Imagine uma sala onde podemos tomar as medidas de temperatura em diversos pontos da mesma. É de se

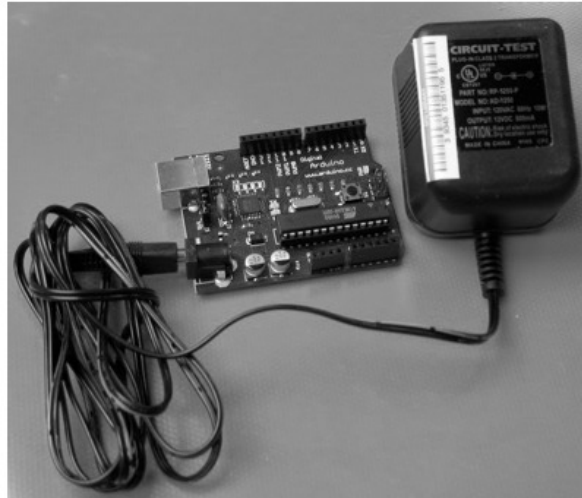
esperar que a cada medida encontremos valores ligeiramente diferentes, dentro de uma certa faixa. Quer dizer, há uma variação e entre dois valores, mesmo próximos, sempre podemos imaginar um outro que seja intermediário. Dizemos que a faixa de valores é contínua, suave. Outro exemplo é o de um circuito eletrônico, no qual podemos encontrar valores de tensão os mais variados. Dizemos nesse caso que o circuito é um circuito analógico. Contudo, há circuitos em que isso não acontece, nos quais predominam, em praticamente todos os pontos, apenas dois valores de tensão, digamos por exemplo, 5 Volts e 0 Volts, não havendo valores intermediários. Agora estamos falando de uma grandeza digital, e o circuito é dito ser um circuito digital. Percebeu a diferença?

Voltando aos pinos do Arduino, quando dizemos que um dado pino é digital estamos querendo dizer que esse pino espera lidar com apenas dois possíveis estados, um deles denominado alto - HIGH - e outro denominado baixo - LOW. Para nossos propósitos, HIGH será associado a 5 Volts enquanto LOW será associado a 0 Volts. Mas quando dizemos que um dado pino é do tipo analógico, então sabemos que esse pino está apto a lidar com sinais que podem exibir qualquer valor dentro de uma faixa, que em nosso caso vai de 0V a 5V, incluindo aí toda uma gama de valores intermediários. Espero que você tenha percebido o que representa essa classificação. É importante frisar que, rigorosamente falando, nenhum equipamento é absolutamente digital, sempre existe alguma parte analógica e normalmente, quando dizemos que um dado circuito ou equipamento é digital, estamos nos referindo ao tipo de tratamento que é dado aos dados que são manipulados no equipamento, como por exemplo, numa TV digital. O Arduino é um dispositivo digital, mas contém circuitos analógicos também. É

considerado dispositivo digital devido ao fato de que a maior parte de suas funções lida com dados digitais e se comportam de forma digital. Afinal, um microcontrolador contém uma CPU, sendo assim um dispositivo lógico digital, perfeitamente preparado para fazer controle digital. Devido a isso o Arduino contém tantas entradas e saídas do tipo digital e, também por isso, mesmo os pinos que podem adotar comportamento analógico, serem por padrão digitais, necessitando configuração explícita na função *setup()* de seu *sketch* para mudarem de comportamento, adotando uma postura analógica.

Alimentação do Arduino

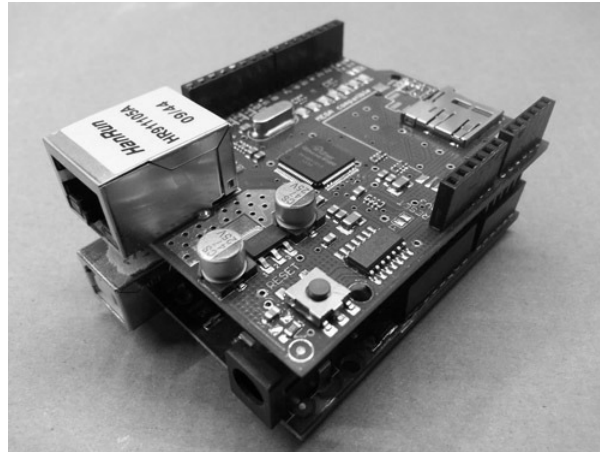
Temos aqui um assunto de grande importância. Quando você conecta o Arduino via porta usb, ele é alimentado pelos 5V fornecidos pela porta. Para efeito de teste usando o *led* L, isso é mais do que suficiente. Contudo o ideal, sobretudo em aplicações mais robustas envolvendo a alimentação de outros componentes como sensores, mais *leds*, transistores etc, é fazer uso de uma fonte de alimentação externa, que pode ser ligada usando-se o conector adequado que existe na placa do Arduino. De acordo como as especificações do fabricante, o valor da tensão da fonte externa ideal deve recair na faixa que vai de 7V a 12V. Um bom valor é 9V, que pode até ser proveniente de uma bateria, embora o ideal seja um adaptador de tomada com capacidade de corrente coerente com o consumo previsto, pelo menos uns 500 mA. Um Ampère se possível. O plugue do adaptador deve ser compatível com o da placa, com 2,1 mm e de centro positivo. Atenção a isso. A figura a seguir mostra um adaptador desse tipo conectado à placa.



Quando você pluga o adaptador de alimentação externo ao Arduino o sistema automaticamente passa a utilizá-lo, em detrimento da alimentação proveniente da porta usb do seu computador.

***Shields* - placas de extensão para Arduino**

A palavra *shield* aparece com frequência quando o assunto é Arduino. *Shield* é um nome genérico para uma placa compatível, a nível de pinagem, com o Arduino e que estende as funcionalidades dele. Por exemplo, um Arduino UNO não tem recursos internos para realizar conexões usando o padrão *Ethernet*, usado para comunicação na internet. Então, quando se precisa desenvolver uma aplicação que faça uso desse tipo de funcionalidade faz-se uso de um *Ethernet Shield*, uma placa contendo todo o *hardware* necessário para implementação do protocolo TCP/IP. Essa placa é encaixada nos conectores do UNO, por cima. Veja a figura a seguir, exibindo uma *Ethernet Shield* conectada a um Arduino.



Há *Shields* para as mais diversas aplicações, desde as mais simples, como a de apenas lhe garantir espaço adicional para prototipagem, até sofisticadas como, por exemplo, realizar funções de GPS. E a lista cresce sem parar, já que os desenvolvedores, oficiais ou não, criam novas placas incessantemente. As *Shields* representam um recurso pra lá de valioso para os desenvolvedores porque lhes possibilitam concentrar-se nas aplicações apenas do ponto de vista do *software*, reduzindo o impacto que representa desenvolver *hardware*, sobretudo para aqueles menos afeitos à eletrônica. E novamente ganha-se em confiabilidade já que as placas, em geral, são de muito boa qualidade e testadas no mercado.

Material de apoio e prototipagem

Sejamos bem realistas: não é viável, e nem divertido, aprender programação de microcontroladores sem que se realize a montagem de circuitos de teste, chamados de protótipos. E normalmente tais protótipos são feitos com o auxílio de uma placa especial, denominada *protoboard*. Veja na figura abaixo o aspecto de um *protoboard* típico, encontrado em lojas de material eletrônico e na internet.



Observe os orifícios na superfície. Neles você pode encaixar os terminais de componentes e fios e dessa forma construir circuitos rapidamente, sem precisar realizar soldagens. E modificar os circuitos é questão de um piscar de olhos, tudo fácil, limpo e organizado. Contudo, para que não haja problemas, é preciso que você entenda corretamente o modo pelo qual o *protoboard* está estruturado, ou seja, de que forma os orifícios estão interligados. Não é difícil. Primeiro entenda que o *protoboard* está espelhado ao longo da linha divisória central horizontal e que a região superior, acima da linha, não tem qualquer tipo de conexão com a região de baixo. Agora observe apenas a região superior, acima da linha divisória central. Nela temos duas partes:

- ✦ A parte mais ao alto, chamada de barramento de alimentação. Nele vemos duas linhas horizontais de orifícios e os orifícios de cada linha estão todos conectados entre si, mas cada uma dessas linhas não tem conexão com a outra. Observe na imagem a região sombreada na parte mais alta do protoboard.

- ✦ Logo abaixo do barramento de alimentação temos uma segunda região e nela as conexões entre os orifícios se dão no sentido vertical, em colunas. Observe uma dessas colunas sombreada na figura. Mas veja, cada coluna é totalmente independente das demais, não havendo qualquer contato elétrico entre colunas. Bem, tudo se repete da mesma forma na parte de baixo do *protoboard*, abaixo da linha divisória central. Mesmo que você nunca tenha

montado um protótipo antes, verá que graças ao *protoboard* a tarefa é fácil e, cá entre nós, divertida. Não se intimide. Ao longo desse livro faremos algumas montagens, passo a passo, tudo ilustrado. Mas antes precisamos conhecer rapidamente mais alguns itens básicos, componentes eletrônicos simples e sempre presentes nas montagens.

Leds

São nossos principais auxiliares, presença frequente nos protótipos, já que nos permitem um obter um retorno visual. Não é a toa que o primeiro programa a ser feito por quem inicia o estudo seja o acendimento de um *led*. A figura mostra um, típico.



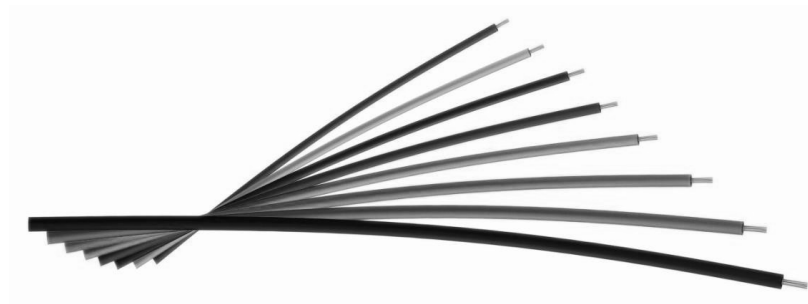
Resistor

Outro componente muito usado é o resistor. São elementos limitadores de corrente, cruciais para que não haja sobrecarga nos pinos do Arduino, o que poderia danificá-lo. Além disso resistores desempenham papel importante quando é necessário construir divisores de tensão e coisas assim. Esse não é um livro de eletrônica básica e não poderemos adentrar muito em detalhes, mas você pode consultar um bom livro ou fazer uma busca na internet. A figura ilustra o aspecto de um resistor comum, do tipo que usaremos em nossas montagens.



Fios ou cabinhos

Para finalizar essa pequena introdução aos materiais de primeira viagem, você precisará de pedaços de fios, conhecidos como cabinhos. Nada de especial, apenas pequenos pedaços de fio flexível, de espessura adequada para os orifícios do *protoboard*. É que embora o *protoboard* já nos forneça meios de conectar componentes, ainda assim existe a necessidade de se fazer ligações como, por exemplo, entre o *protoboard* e o Arduino. Nesse hora os cabinhos são indispensáveis.



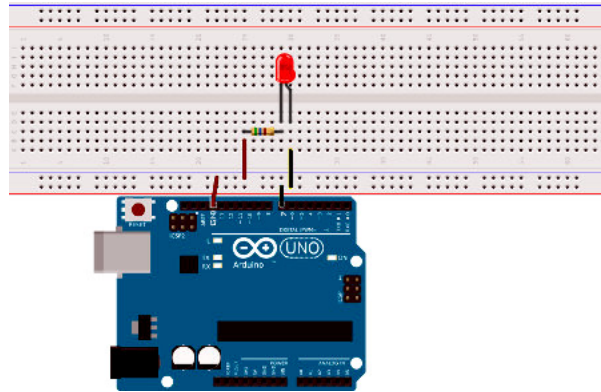
Capítulo 4

Montando um protótipo do Blink

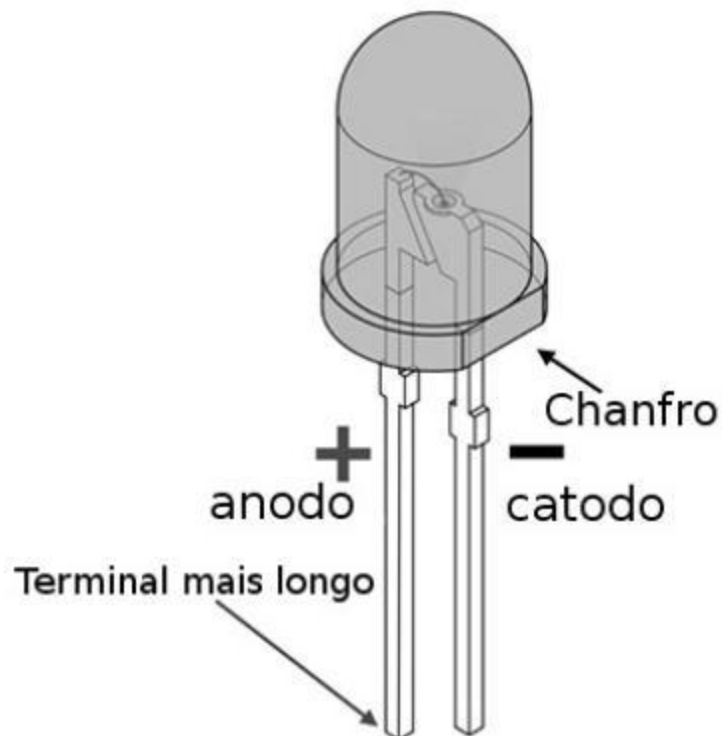
Você já sabe como funciona o *sketch* de exemplo Blink. Ele usa o *led* interno L, na placa do Arduino. Nossa proposta agora é construir um protótipo, usando um *proto board*, um *led* e um resistor, que funcione da mesma maneira que o exemplo que conhecemos, mas agora usando um periférico externo, ou seja, faça o *led* que montaremos no *proto board* piscar na frequência que desejarmos. Esse exercício lhe capacitará a montar seus protótipos com segurança, pois lhe fornecerá todas as informações de que precisa para esse tipo de tarefa e servirá como um treino valioso, principalmente se for sua primeira montagem. Para seguir adiante você vai precisar dos materiais listados abaixo e lembre-se de que no [site](#) você encontra a listagem do programa e imagens dos esquemas de montagem.

1. Fonte de alimentação estável e regulada, na faixa de 7V a 12V e capacidade de fornecer corrente de, pelo menos, 500 mA.
2. Um led comum, de qualquer cor.
3. Um resistor de carbono, no valor de 560Ω com qualquer potência.

4. Um *protoboard*.
5. Pedacos de fio cabinho tendo em torno de 10 cm de comprimento.
6. Seu Arduino UNO.



Como eu disse antes, o único componente polarizado aqui é o *led*. Observe a figura a seguir para entender essa polarização.



Compare o *led* real que você tem em mãos com o apresentado na figura e certifique-se de que o catodo seja conectado ao resistor e

o anodo ao pino 7 do Arduino. Use o chanfro no *led* para se orientar ou, se o *led* for absolutamente novo, o terminal mais longo. A seguir temos a listagem do código para nosso experimento:

```
/* Introdução à Programação de Arduino - Vol. 1
```

```
* Erich Georg
```

```
* Sketch faz o mesmo que o exemplo BLINK,
```

```
* usando outro pino e um protoboard
```

```
* para a montagem do lede do resistor.
```

```
* O led será conectado ao pino 7 do Arduino.
```

```
* A função do resistor de 560 ohm é limitar
```

```
* a corrente pelo led, evitando sobrecarga
```

```
* no pino 7.
```

```
*/
```

```
// Função setup()
```

```
void setup() {
```

```
// inicializa pino digital usado pelo led:
```

```
pinMode( 7 , OUTPUT);
```

```
}
```

```
// Agora o laço onde toda a ação acontece
```

```
// de maneira ininterrupta.
```

```
void loop() {
```

```
// energiza o pino 7 - liga led
```

```
digitalWrite( 7 , HIGH);
```

```
// delay de um segundo;
```

```
delay( 1000);
```

```
// desenergiza o pino 7 - apaga o led
```

```
digitalWrite( 7 , LOW);
```

```
// delay de um segundo
```

```
delay( 1000);
```

```
}
```

Se compararmos esse novo *sketch* com aquele que examinamos antes, do BLINK que é fornecido como exemplo no IDE do Arduino, perceberemos facilmente que se trata do mesmo código, com ligeiras alterações. Na verdade, a única coisa que mudou foi o pino controlado, que agora passa a ser o pino 7. Essa mudança é reflexo da montagem que fizemos, como podemos facilmente ver pela figura. Observe que a escolha do pino 7 é feita na função *setup()*.

Bem, hora de colocar a mão na massa. Faça a montagem de seu protótipo de acordo com o que aparece na figura, certificando-se de que o *led* não seja montado invertido pois, caso isso ocorra, ele não acenderá. Para essa prática, caso você não disponha ainda da fonte externa de alimentação, pode alimentar o Arduino usando apenas a porta usb, já que o consumo do circuito externo é muito pequeno. Uma vez realizada a montagem e com tudo duplamente conferido, abra seu ambiente de desenvolvimento, digite o código de acordo com a listagem, compile e carregue no seu Arduino, da forma como foi ensinado no capítulo 2. Observe então o *led* no *protoboard* piscando a intervalos de um segundo. Faça várias experiências alterando os valores de *delay* e observe. O que acontece quando você usa valores muito baixos para o argumento da função *delay()*? Você é capaz de encontrar uma explicação para o comportamento do *led* nesse caso? Agora experimente escolher outro pino digital e refaça a montagem, é fácil e rápido graças ao uso do *protoboard*. Então edite o *sketch* para refletir sua nova configuração de circuito e teste novamente.

Bem, estamos chegando ao fim desse primeiro volume, mas apenas começando a jornada. Nos volumes que virão na sequência avançaremos no estudo com a apresentação de técnicas de programação aplicada ao controle digital, técnicas que irão lhe

fornecer, progressivamente, ferramentas valiosas para dominar a programação de dispositivos lógico digitais, úteis não apenas à programação do Arduino mas aplicáveis à programação de qualquer outra plataforma para desenvolvimento de controle digital.

OceanofPDF.com

Glossário

- Alias - segundo nome, apelido. Em informática é frequentemente usado para se referir a um endereço eletrônico alternativo.
- Ampère - André-Marie Ampère, físico francês do século XVIII, fez grandes contribuições à eletricidade. Em sua homenagem a unidade de corrente elétrica foi batizada de Ampère.
- Analógico - diz-se daquilo que é contínuo, em oposição ao digital, discreto.
- Assembly - linguagem de programação, composta de mnemônicos que tem estreita relação com o código de máquina de um determinado processador.
- Atmega328P - microcontrolador fabricado pela Atmel©
- Atmel© - um dos maiores fabricantes de microcontroladores do mundo.
- Sistema binário - sistema de numeração contendo dois dígitos, 0 e 1. Um código que pode ser expresso como combinações de zeros e uns.

- Bootloader - rotina gerenciadora de carregamento de programas para a memória do microcontrolador.
- C - linguagem de programação concebida nos anos 70 por Dennis Ritchie.
- C++ - linguagem de computação orientada a objetos, criada por Bjarne Stroustrup.
- Compilador - programa responsável por converter um código em uma dada linguagem para código binário.
- Digital - diz-se da grandeza que só pode assumir dois estados.
- Ethernet - tecnologia usada para implementação de *hardware* de comunicação de rede.
- Frequência - número de vezes que um dado evento se repete num dado intervalo de tempo.
- Hertz – Hz - físico alemão do século 19, que fez importantes descobertas relacionadas à propagação eletromagnética. Em sua homenagem a unidade de frequência foi batizada de Hertz.
- Hexadecimal - sistema de numeração que faz uso de 16 dígitos, muito usado em computação por permitir uma notação mais compacta que os sistemas binário e decimal.
- HTML - Hyper Text Markup Language, linguagem na qual as páginas de internet são codificadas.
- I/O - Input/output, o mesmo que entrada e saída.
- ICSP - In Circuit Serial Programming, técnica para se programar um microcontrolador sem retirá-lo do circuito onde opera.

- IDE - *Integrated Development Enviroment*, o mesmo que ambiente de desenvolvimento integrado.
- Java - linguagem de programação concebida para dar suporte à internet das coisas.
- Led - Diodo emissor de luz.
- Linux - sistema operacional gratuito, estável e confiável. Excelente para programadores.
- Loop - o mesmo que laço, rotinas que se repetem.
- Milisegundo - milésima parte do segundo.
- Ohm - unidade de resistência elétrica, em homenagem a Georg Simon Ohm, físico alemão que revelou ao mundo a Lei de Ohm.
- Open source - código aberto, programas cujos códigos podem ser modificados livremente pela comunidade.
- Palavra reservada - palavra com significado especial numa dada linguagem, de uso restrito.
- Parâmetro funcional - dado que uma função recebe como argumento para poder executar o código que faz parte do corpo da função.
- Porta COM - nos sistemas Windows®, nomenclatura com que as portas seriais são identificadas.

programa - sequência de instruções que realizam determinada tarefa, executadas por uma unidade central de processamento.

- Protoboard - matriz contendo orifícios e contatos para montagem rápida de protótipos eletrônicos.

- PWM - técnica pela qual se pode gerar sinais elétricos modulados, muito usada para realizar controle de potência.
- Python - linguagem de programação de alto nível.
- Resistor - componente passivo limitador de corrente.
- S.O. - abreviatura de Sistema Operacional.
- Sensor - Transdutor de entrada de sistemas de controle. Dispositivo capaz de detectar determinado tipo de grandeza física, como calor, luz etc.
- Shield - No jargão do Arduino, placa que estende a capacidade do mesmo.
- Sketch - o mesmo que programa, no linguajar do Arduino.
- Software - programas que rodam num sistema.
- Soquete - receptáculo apropriado para encaixe de componentes.
- TCP/IP - protocolo de comunicação em rede.
- Temporizador - dispositivo que marca tempo. Módulo quase sempre presente no interior de microcontroladores.
- USB - *universal serial bus*, ou barramento serial universal. Um barramento serial que substitui o antigo com vantagens, como a de permitir a conexão de grande número de dispositivos sem que se precise reiniciar o computador.
- Volt – V - unidade de tensão ou ddp elétrica, em homenagem a Alessandro Volta, físico italiano do século XVIII que deu grande

contribuição à teoria da eletricidade.

- ZIP - algoritmo de compactação de arquivos.

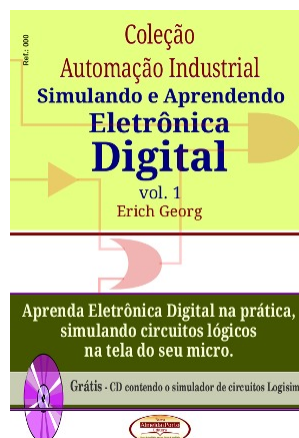
OceanofPDF.com

Outros livros do mesmo autor

- Programando o PIC para Automação e Controle - 2 volumes



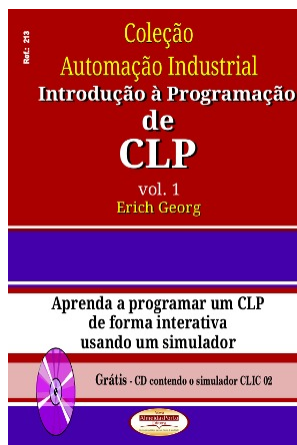
- Simulando e Aprendendo Eletrônica Digital - 2 volumes



- Sensores - Tipos, Princípios e Aplicações - 2 volumes



- Introdução à Programação de CLP - 2 volumes



- Curso Básico de Programação de Microcontroladores PIC - 5 volumes



Visite nosso site

[EEPROM - eletro-eletrônica, programação e mecatrônica.](#)

e conheça mais sobre nossas publicações, acompanhe nossos tutoriais, dicas e blog.

[OceanofPDF.com](#)