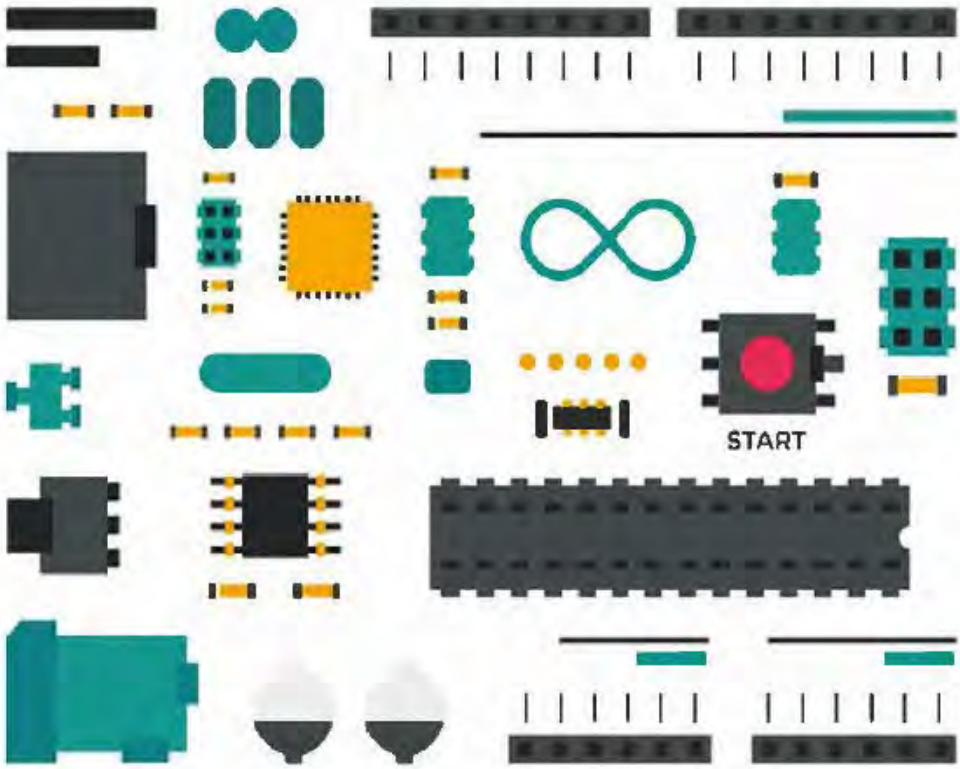
Arduino

Guia para colocar suas ideias em prática



Casa do Código

FERNANDO BRYAN FRIZZARIN

AGRADECIMENTOS

A Deus.

À Cecília Regina Bryan Frizzarin e ao Gilson Waldemar Frizzarin, meus pais, pela criação.

À Priscila Keli de Lima Pinto Frizzarin, ao Eduardo Pinto Frizzarin e ao Bruno Pinto Frizzarin, minha família, pelo amor incondicional.

Ao Prof. Marcos Teixeira, pela amizade e muitos ensinamentos.

Ao Vinicius Donadelli e ao Prof. José Alberto Matioli, pelo incentivo.

À Profª. Vanessa Juliato e à Profª. Magda Silva Rizzeto, por reforçar que o conhecimento é mesmo uma descoberta.

Ao Octavio Nogueira, por parte dos equipamentos e por acreditar.

Ao Victor Badolato Athayde, pelas ilustrações, ideias e debates acalorados.

Ao COTIL/UNICAMP, por me permitir ensinar – eis a profissão mais apaixonante do mundo.

SOBRE O AUTOR

Fernando Bryan Frizzarin, natural de Americana, é técnico em Informática, bacharel em Ciência da Computação e Psicopedagogo, especialista em Redes de Computadores e MBA em Gestão Estratégica de Negócios.

Ele é professor do Magistério Secundário Técnico na Universidade Estadual de Campinas (UNICAMP), professor do ensino superior nas Faculdades Integradas Einstein de Limeira, e supervisor de Informática do Departamento de Água e Esgoto de Americana. Também é voluntário na Fundação Romi no Núcleo de Ensino Integrado, em Santa Bárbara d'Oeste (SP), como consultor para o ensino de robótica no Ensino Fundamental.

Coautor da patente BR1020140270159/2014: *Dispositivo automatizado de liberação controlada*, projeto desenvolvido em conjunto com os alunos Bianca de Mori Machado Amaral e Felipe Ferreira da Silva, incluindo apoio da Arqª Marylis Barbosa de Souza. Esse projeto foi desenvolvido nas disciplinas de Desenvolvimento de Projetos e Tópicos Avançados em Informática no Colégio Técnico de Limeira (UNICAMP), e o depósito feito por meio da Agência de Inovação da UNICAMP (INOVA).

COMO UTILIZAR ESTE LIVRO

Todo autor tem uma preconcepção de como seu livro deveria ser utilizado, e eu não sou exceção para essa "regra", mas acredito que este livro deva ser usado da forma mais criativa que você conseguir. Seja livre para voar.

Aqui estão, basicamente, anotações de aulas de laboratórios apresentados durante vários anos, para várias classes e, sendo assim, para muitos alunos. E tudo isso carrega a vantagem de que as experiências e a teoria foram extensivamente testadas, ajustadas e testadas novamente, sempre na procura de uma sequência e um tom ideal.

Logo, o livro é especialmente direcionado para estudantes, mesmo que não necessariamente das áreas técnicas de informática ou eletrônica digital, interessados em saber como conceber equipamentos e criar soluções para os mais diversos problemas.

Há necessidade de um conhecimento prévio, porém básico, de lógica de programação em qualquer linguagem, já que neste livro a linguagem a ser utilizada será mostrada na teoria e na prática de forma fácil de compreender. A ideia é que se possa usar este livro principalmente como material de consulta para quaisquer aulas que envolvam Arduino.

Você pode deixar este livro em uma estante para consultas esporádicas, ou pode andar com ele em baixo do braço para consultas constantes. Importante mesmo é que ele lhe seja útil e acrescente novos conhecimentos e abra novos caminhos.

Minha formação construtivista e minhas experiências profissionais mostram que o importante é apresentar os conceitos, as ferramentas e os métodos para, em seguida, desafiar o aluno a

encontrar utilidade de tudo isso em seu mundo, dentro do contexto do seu dia a dia, de forma que possa impactar e mudar, para melhor, o seu mundo interior e exterior.

Sumário

Capítulo 1

INTRODUÇÃO

Como acontece com muitos professores, alguns alunos sempre comentam sobre a quantidade de matéria que precisam absorver e copiar: "Nossa professor! É tanta coisa que daria um livro". Essa ideia sempre ficou ecoando, até que ganhou corpo com este texto.

Há também alguns outros professores que sugeriram que as minhas aulas não ficassem “presas” dentro da sala de aula, ou apenas com os alunos que conseguem vir aprender conosco no colégio ou na faculdade. "O conhecimento foi feito para ser espalhado o máximo possível"; incentivos nunca faltaram.

Pois aqui estão as minhas aulas de Arduino, utilizadas principalmente nas aulas de Automação e Controle e Desenvolvimento de Projetos no COTIL/UNICAMP e projetos na Fundação Romi. Logicamente, não foi simplesmente uma transcrição literal, muita coisa foi melhorada ou adaptada para ser apresentada no atual formato.

A união dessas duas disciplinas, Automação e Controle e Desenvolvimento de Projetos, visa responder às observações do crescimento acentuado sobre as exigências de profissionais cada vez mais inovadores, capazes de inovar, inventar e reinventar-se, adaptando-se aos novos desafios impostos pela competição mercadológica.

Espera-se que os profissionais que ingressarão no mercado

daqui para frente não sejam apenas executores de tarefas, mas sim solucionadores de problemas e entregadores de soluções. Ser capaz de observar um dado problema e resolvê-lo com criatividade será cada vez mais pré-requisito para ter acesso aos melhores postos de trabalho e ascender na carreira profissional.

No mundo cada vez mais competitivo em que vivemos onde há sempre a necessidade urgente de um novo produto ou serviço para ganhar novos mercados.

Inovar não é necessariamente inventar, e inventar também não é necessariamente inovar. Inventar também não é descobrir! Descobrir é trazer, à luz da ciência, fatos, objetos ou técnicas antes desconhecidas que representam uma solução para o que já existe. Inventar é, quase invariavelmente, uma solução para um problema, muitas vezes técnico e parte do cotidiano do inventor.

Para saber mais sobre esse assunto, sugiro a leitura da Lei Federal brasileira nº 9.279 de 14 de maio de 1996, que regula os direitos e obrigações relativos à propriedade industrial.

Inventar é um ato de convivência com o problema, muita imaginação e criatividade. Bons inventores dedicam-se em, principalmente, ouvir e observar o mundo e seus problemas. Hoje, é comum a figura do funcionário que passa dias observando o trabalho de outros funcionários, como lidam com as ferramentas (tecnologias) e com os demais colegas para tentar encontrar (inventar) algum método ou ferramenta que melhore e facilite o trabalho.

Normalmente, empresas que mantêm funcionários assim possuem Departamento de Pesquisa e Desenvolvimento (P&D) e contam até com verbas públicas de fomento, já que esse tipo de trabalho gera patentes, royalties e *know-how*.

Já inovação é a concepção de um novo produto ou processo, bem como a agregação de novas funcionalidades que impliquem em melhorias incrementais ao que já existe. Melhorias estas que podem ser no ganho da qualidade, produtividade, usabilidade, longevidade ou competitividade. A inovação começa na invenção, uma vez que implementar uma nova ideia é tão essencial quanto criá-la, porém será assunto para próxima oportunidade.

O objetivo final deste livro é servir como referência para o uso do Arduino de forma a ajudar que o leitor entenda todas suas possibilidades. Não com projetos prontos e, de certo ponto de vista, limitantes, mas com experiência que possam ser interconectadas, combinadas, modificadas e aprimoradas para que a necessidade seja suprida. Ou seja, dar subsídios para que o leitor possa, a partir de pequenas montagens e códigos-fontes simples, liberar toda sua criatividade e, assim, inventar e inovar a partir do que será apresentado.

O ensino de robótica nas escolas de ensino fundamental, de lógica de programação e automação, e nas escolas de ensino médio e técnica, desde o mais cedo possível, traz inúmeras vantagens aos alunos, às escolas e à sociedade. Esta última é a maior beneficiária desse movimento, já que em um futuro próximo colherá os frutos de mentes mais criativas e inovadoras.

Este sim é um excelente investimento.

Capítulo 2

O QUE É ARDUINO

Arduino é uma plataforma formada por um equipamento eletrônico e um ambiente de programação integrado (*Integrated Development Enviroment* - IDE) para prototipagem eletrônica e de software. O equipamento eletrônico da plataforma Arduino consiste em uma placa de circuitos integrados, devidamente equipada com seus componentes eletrônicos e cujo componente central é um microprocessador do tipo AVR da Atmel®.

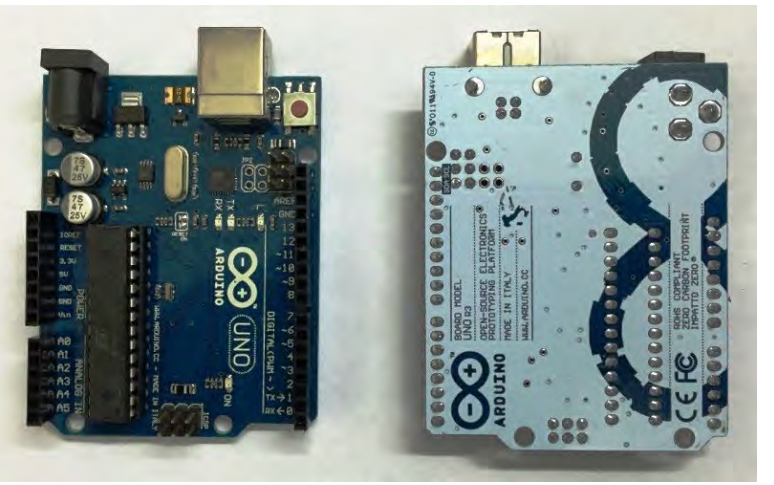


Figura 2.1: Um Arduino UNO frente e verso

O Ambiente de Desenvolvimento Integrado (IDE) é, como veremos com mais detalhes adiante, um software desenvolvido para ser multiplataforma. Ele é escrito na linguagem Java, e automatiza tarefas como depuração, compilação e envio do binário compilado

para o microcontrolador na placa eletrônica Arduino. A linguagem de programação utilizada para a programação para o Arduino é uma variação da Linguagem C, baseada na linguagem Wiring.

Figura 2.2: A tela principal do IDE Arduino

Atualmente, existem 12 versões da placa eletrônica Arduino que podem ser vistas no site https://www.arduino.cc/en/Main/Products. Também existem dezenas de kits e módulos, chamados shields, que podem ser interligados com as placas Arduino de forma a lhe conferir habilidades e funcionalidades adicionais e específicas.

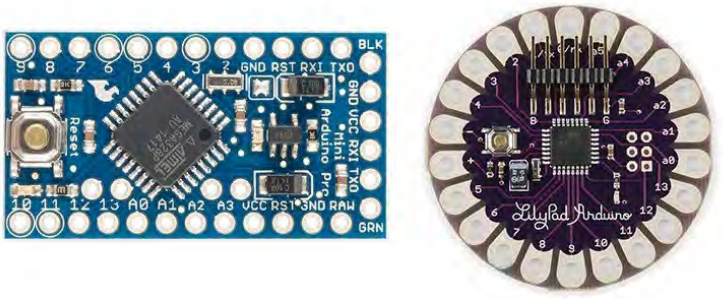


Figura 2.3: Duas outras versões de Arduino, à esquerda o Arduino Mini e à direita o LilyPad Arduino (fonte: arduino.cc)

Para este livro, vou considerar como placa padrão o Arduino UNO R3, mostrado na primeira figura, o mais difundido de todos, principalmente no Brasil, onde é fácil encontrá-lo em lojas e sites especializados.

Todas as variações de Arduino possuem, basicamente, o mesmo esquema de funcionamento, mas com variações nas interfaces, velocidade de processamento, capacidade de memória, armazenamento interno e algumas funcionalidades específicas, como interface de rede com ou sem fio, Bluetooth etc. Porém todas essas "alterações" podem ser incluídas também no Arduino UNO R3 através de módulos (*shields*), inclusive capacidade de armazenamento, exceto o que envolve capacidade de processamento.

2.1 COMPONENTES DE UM ARDUINO UNO R3

Principais componentes

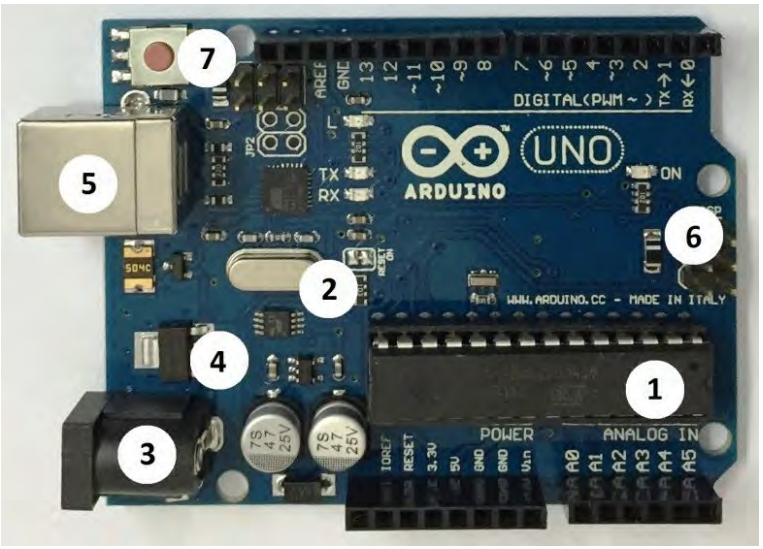


Figura 2.4: Principais componentes do Arduino UNO

1. Microcontrolador Atmel® Atmega 328P.

Figura 2.5: O microcontrolador do Arduino UNO separado da placa

2. Clock de 16 MHz.

3. Conector para alimentação em corrente contínua de 6V até 20V (recomendado de 7V até 12V).

4. Regulador de voltagem.
5. Conector USB com um conversor USB-Serial integrado.
6. Interface ICSP.
7. Botão de Reset.

Interfaces

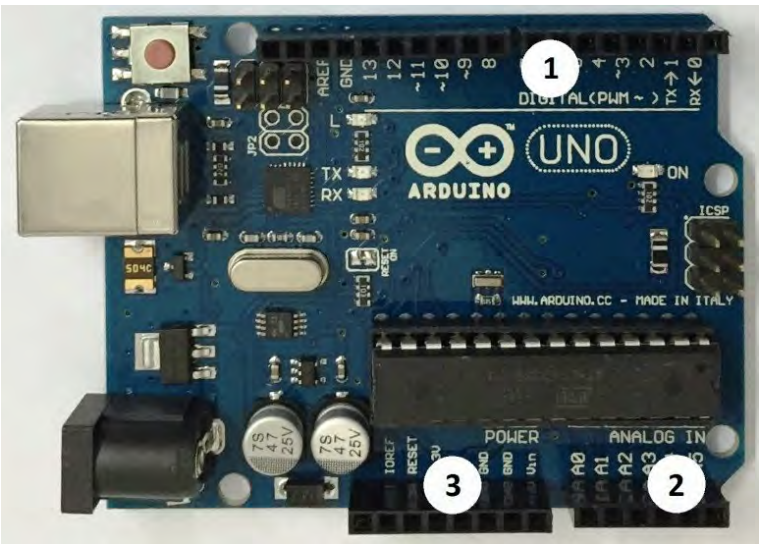


Figura 2.6: Interfaces existentes no Arduino UNO

1. **Pinos digitais**

   São 14 pinos digitais ao todo, sendo que os pinos 0 (RX – receptor) e 1 (TX – transmissor) são usados em compartilhamento à comunicação serial nativa do microcontrolador. Os pinos 3, 5, 6, 9, 10 e 11 com a marcação de uma acentuação til (~) são pinos PWM (*Power-with Modulation*), sendo os demais sem capacidade específica. Deste lado da placa, ainda há um pino GND (terra) e um AREF (tensão de referência para entradas analógicas).

Esses pinos podem assumir os modo de entrada ou saída e apenas dois estados, sendo `HIGH` (logicamente alto) ou `LOW` (logicamente baixo).

2. **Pino analógicos**

   São 6 pinos analógicos ao todo que podem assumir os modos de entrada ou saída, ler e/ou receber variações de tensão e convertê-las em valores digitais.

3. **Pinos de alimentação**

   São 6 os pinos de alimentação, sendo eles:

   - *Vin* – Voltagem de entrada pode-se alimentar o Arduino por esse pino, mas associado a ele não há um regulador de voltagem, portanto, é preciso cuidado;
   - *Gnd* – Terra;
   - *5V* – Saída de 5V a partir do regulador de voltagem;
   - *3,3V* – Saída de 3,3V a partir do regulador de voltagem;
   - *Reset* – Deixando esse pino como estado lógico `LOW` reinicializará o microcontrolador;
   - *IORef* – Voltagem de referência de operação do microcontrolador.

Luzes indicativas

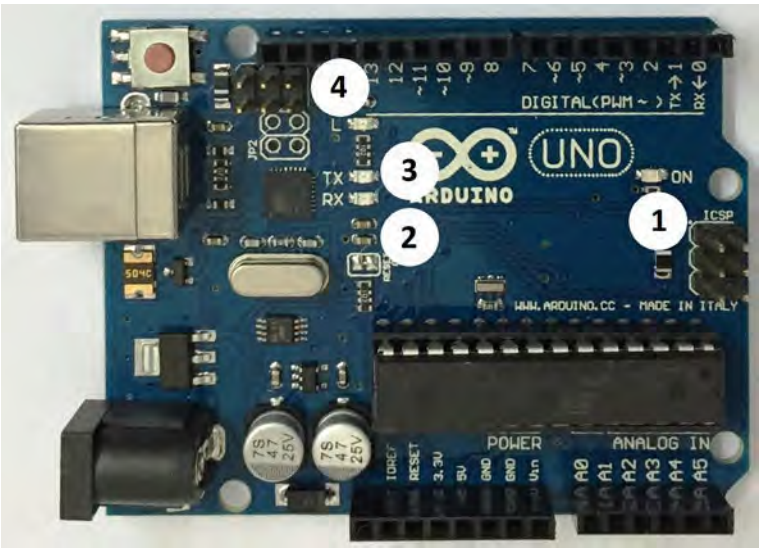


Figura 2.7: Luzes indicativas do Arduino UNO

1. LED indicador de funcionamento, permanece aceso enquanto o Arduino estiver ligado.
2. LED indicador de atividade de comunicação serial para recepção (RX).
3. LED indicador de atividade de comunicação serial para transmissão (TX).
4. LED integrado ao pino digital 13, quando o pino é colocado em estado lógico `HIGH` o LED acenderá, e quando é colocado em `LOW` , o LED apagará, como veremos mais adiante.

Memórias internas

O Arduino possui 4 tipos de memórias internas diferentes para determinados propósitos. Uma delas é a memória Flash, com capacidade de 32 KiB sendo que desse montante 512 bytes são ocupados pelo programa inicialização (*bootloader*) e o restante é

ocupado pelos programas que desenvolvemos e carregamos no microcontrolador.

O bootloader é o primeiro programa a ser executado quando o microcontrolador é ligado, e comanda o carregamento e inicialização do programa carregado por nós no microcontrolador. É o bootloader que possui um conjunto de instruções que permite a gravação da aplicação e sua execução.

Os dados que são criados e manipulados por nossos programas, tais como variáveis e constantes, e são armazenados na memória SRAM, ou *Static Ram* (memória estática de acesso randômico), que possui células de memória que não precisam ser atualizadas de tempos e tempos, poupando assim o consumo de eletricidade.

Há também a memória EEPROM, ou *Electrically-Erasable Programmable Read-Only Memory* (Memória Eletricamente Apagável e Programável apenas para Leitura) com capacida de 1 KiB. A EEPROM é uma memória regravável não volátil, ou seja, não perde os dados quando o equipamento é desligado e normalmente é usada para guardar informações por longo períodos, informações que podem ser o status do equipamento, dados de configurações etc. Como toda memória possui uma vida útil, sendo que o número de (re)gravações é limitado entre 100 mil e 1 milhão de vezes.

2.2 A SEGUIR...

Agora que você sabe que o Arduino é uma placa eletrônica dotada de um microcontrolador, várias interfaces analógicas e digitais, vamos ver as possibilidades que ele nos traz para que tudo funcione da maneira que você desejar.

Capítulo 3

POR QUE USAR ARDUINO?

Arduino é, como será detalhado durante todo o livro, basicamente uma plataforma de prototipagem de baixo custo e pequena curva de aprendizagem. Ele vem de encontro com as filosofias Maker ou hobbistas do *faça-você-mesmo* que, em resumo, significam fazer sistemas físicos interativos pelo uso de software e hardware que possam "sentir" e "responder" ao ambiente onde estão inseridos - seja pela construção de um carrinho de controle remoto que pode ser guiado pelo uso do aparelho celular, matrizes de LED ou até painéis interativos para controle de filas.

Também é possível pensar em Arduino pela ótica da Internet das Coisas (ou *Internet of Things* - IoT) que visa criar equipamentos com capacidade e objetivo de transmitir dados e interagir diretamente com a internet sem a efetiva intervenção humana. Exemplos para esse uso são vastos e passam por estações meteorológicas automáticas, sensores de presença e liberação de acesso até rastreadores veiculares.

Pessoalmente, acredito que é possível agrupar todos esses campos sob a única definição de Computação Física (*Physical Computing*), pois nessa definição podemos incluir praticamente todos os equipamentos e funcionalidades desejadas, como por exemplo, em sistemas de controle de tráfego, automação industrial, residencial e comercial, e jogos.

É nesse contexto, de trazer ao mundo toda criação possível

fazendo uso intensivo de sensores e microcontroladores para traduzir entradas analógicas para um sistema de software e controlar equipamentos eletromecânicos, em que o Arduino está incluso.

Apenas para ilustrar, o Arduino pode ser usado em, mas não somente:

- **Jogos**

  Jogos interativos que usam o corpo ou o movimento do usuário como controle, como máquinas de dança, jogos de tiro em arcade etc.

- **Residências**

  - Controle automático da iluminação e temperatura interna de acordo com o clima e hábitos dos moradores;
  - Geladeiras que "sabem" quando um produto entra ou sai;
  - Vasos que regam plantas automaticamente e alimentadores automáticos de animais;
  - Controle por voz de eletrodomésticos;
  - Regadores automáticos de plantas;
  - Alimentador automático de animais de estimação.

- **Comércio**

  - Gôndolas de supermercado inteligentes que percebem a retirada ou colocação de produtos;
  - Contagem de clientes;
  - Exibição de peças publicitárias apenas com a aproximação de pessoas;
  - Alimentador automático de animais de criação.

- **Indústria**
    - Máquinas para inspeção de peças;
    - Sensores para inspeção de máquinas;
    - Controle automático de temperatura, umidade e iluminação;
    - Controle por voz de máquinas;
    - Rastreamento de ferramentas, equipamentos e produtos.
- **Brinquedos**

  É possível construir miniaturas controladas à distância, acionar motores, criar sons e luzes com o objetivo de entreter crianças, jovens e adultos.
- **Ensino**
    - Simuladores interativos de forças físicas;
    - Medidores de gases, PH, força e peso;
    - Realizar interação entre o aluno e o conteúdo por meio de sensoriamento do ambiente escolar.
- **Tráfego terrestre**
    - Semáforos que alteram a duração do sinal de acordo com a quantidade e fluxo de veículos;
    - Contagem de veículos em estradas, ruas e avenidas e apresentação automática da orientação do motorista de rotas alternativas.
- **Museus**

  Desenvolver exibições mais próximas dos visitantes e mais interativas, do tipo em que a exposição responde à presença ou passagem do visitante, por exemplo.

- **Arte**

  Análogo ao museu, a obra pode mudar e responder ao visitante, à iluminação, à temperatura etc.

Essas são apenas algumas das infinitas criações possíveis com o Arduino e, para iniciar a criar qualquer coisa usando Arduino, você precisará apenas de conhecimento básico de eletrônica, programação e criatividade. Para os dois primeiros, este livro lhe ajudará. Para a criatividade, basta deixar a imaginação voar livre!

Mais à frente, você terá a oportunidade de montar e programar 22 experimentos utilizando o Arduino, de forma a entender o seu funcionamento e sua programação, e permitindo que você, a partir desses exemplos, possa conceber seus próprios projetos.

3.1 A SEGUIR...

Como você percebeu, é possível desenvolver praticamente qualquer coisa usando o Arduino, relativa a automatização, coleta de dados e até mesmo interação com pessoas, animais e plantas. A seguir, você terá todas as informações necessárias sobre o que será necessário para desenvolver as experiências aqui expostas. Essas informações também servirão na hora de você soltar a imaginação no que desejar desenvolver.

Capítulo 4

O QUE SERÁ NECESSÁRIO TER

Ter um Arduino e os componentes dos quais falaremos ao longo do livro será muito útil, mas não tê-lo não será um impeditivo para aprender sobre ele.

Para utilizar e realizar todas as experiências deste livro, será necessário ter basicamente um computador com um bom navegador instalado e acesso à internet. Todas as experiências poderão ser realizadas pelo simulador de Arduino, sem a necessidade de se ter os componentes e placas fisicamente. Porém, o contato físico com esse tipo de equipamento é enriquecedor e traz consigo as noções de tamanho, peso, temperatura, movimento etc.

O simulador em especial permite que sejam vencidos medos e limitações econômicas para começar a estudar e entender o ecossistema do Arduino. Você poderá aprender, estudar e criar seus circuitos sem praticamente qualquer custo.

Minha sugestão inicial é de treinar bastante usando o simulador, mas quando se sentir seguro e com condições, passe a adquirir os equipamentos físicos, começando por um Arduino e, em seguida, pelos demais componentes apresentados neste livro.

Mesmo assim, uma coisa, o simulador, deve ser encarada como complemento da outra, o Arduino físico (e vice-versa), pois ideias surgem a todo o momento, de várias formas e lugares.

4.1 SIMULADOR DE ARDUINO

A Autodesk Inc.® criou e disponibiliza um simulador de circuitos eletrônicos on-line que inclui o Arduino em suas versões Uno R3, DFRduino R3, Uno e Micro, até o momento da edição deste livro. Acredito que a tendência é que, com o tempo, cada vez mais versões de Arduino e componentes eletrônicos sejam incorporados à simulação.

O simulador pode ser acessado em http://123d.circuits.io e será utilizado nas experiências apresentadas.

> Algumas imagens deste livro são capturas de tela da Autodesk, com reimpressão sob cortesia da Autodesk, Inc.

Figura 4.1: Tela principal do simulador

Para poder simular circuitos, é necessário inscrever-se no site. Para isso, use o link *Sign Up* que está no lado direito superior da tela. Você terá de, como primeiro passo, determinar o país de

residência e a data de seu aniversário.

Figura 4.2: Tela para a seleção de país e data de aniversário

Após avançar, você terá a seguinte tela:

Figura 4.3: Janela de login com botão criar conta

Digite seu e-mail e uma senha e, em seguida, clique em criar conta. Você também pode, em vez de criar uma conta, usar suas credenciais do Facebook. Note que as regras para senha são que ela deve conter no mínimo 8 caracteres, com ao menos um número e uma letra.

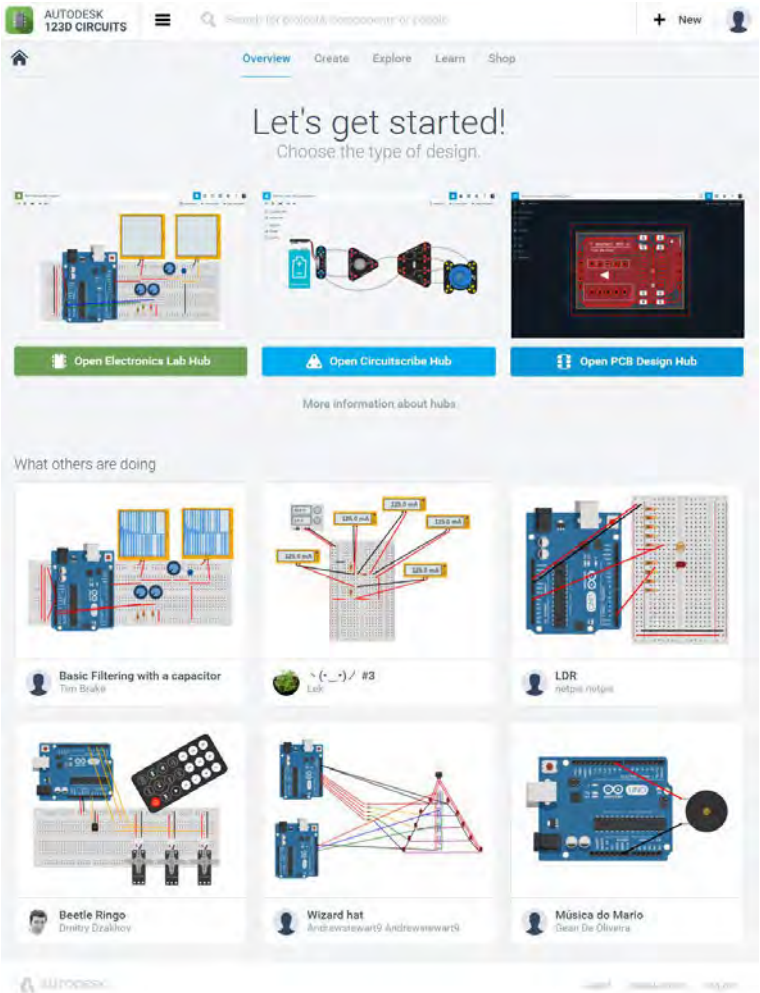


Figura 4.4: Janela de boas vindas

Pronto! Conta criada. À direita, no canto superior, você tem um ícone para editar os detalhes da sua conta.

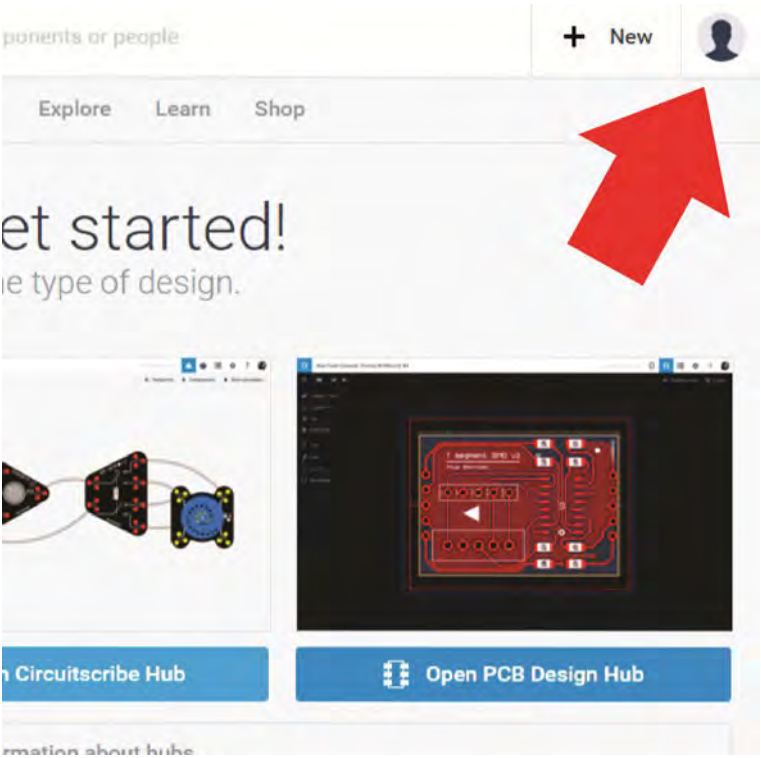


Figura 4.5: Destaque para o ícone da conta

Para começar a editar um novo circuito, clique no link *Create*.

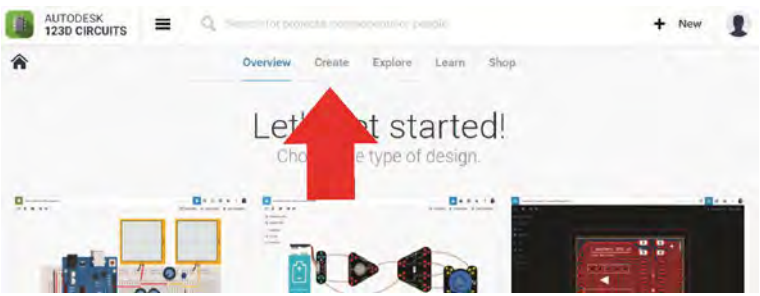


Figura 4.6: Destaque para o link create

Agora, clique no link *Open Electronics Lab Hub*.

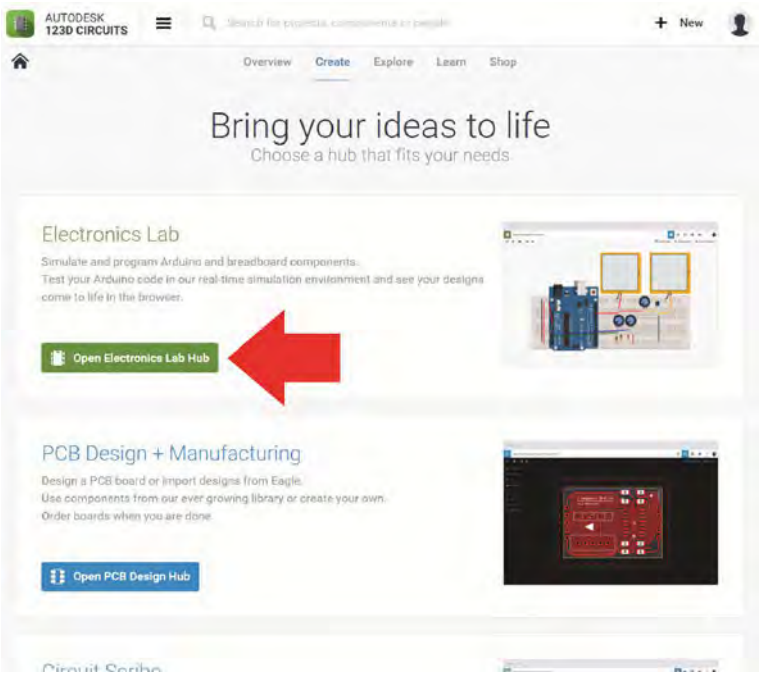


Figura 4.7: Destaque para link open electronics lab hub

Nessa tela de boas-vindas, clique no link *New Electronics Lab*.

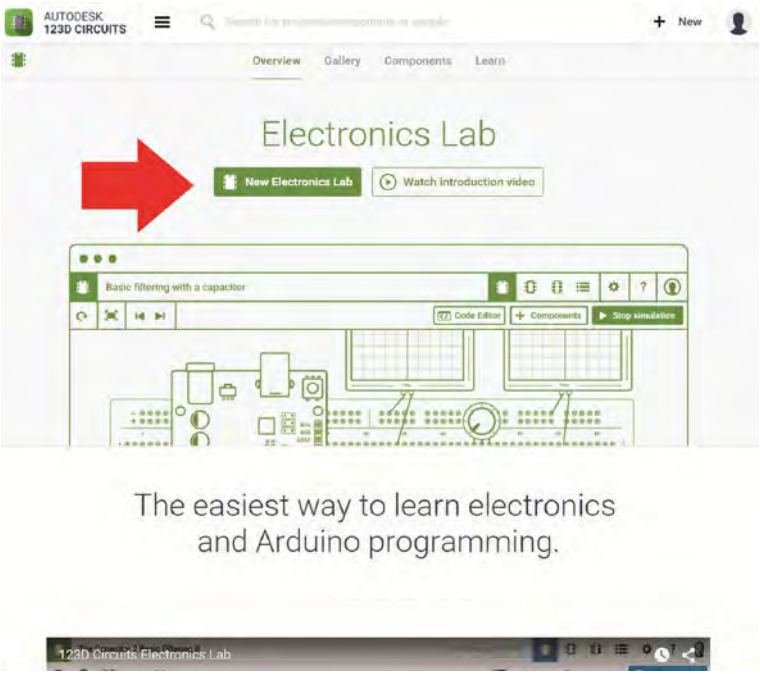


Figura 4.8: Destaque para o link New Electronics Lab

Você será levado ao laboratório de eletrônica e deverá ter uma placa de ensaios na sua tela.

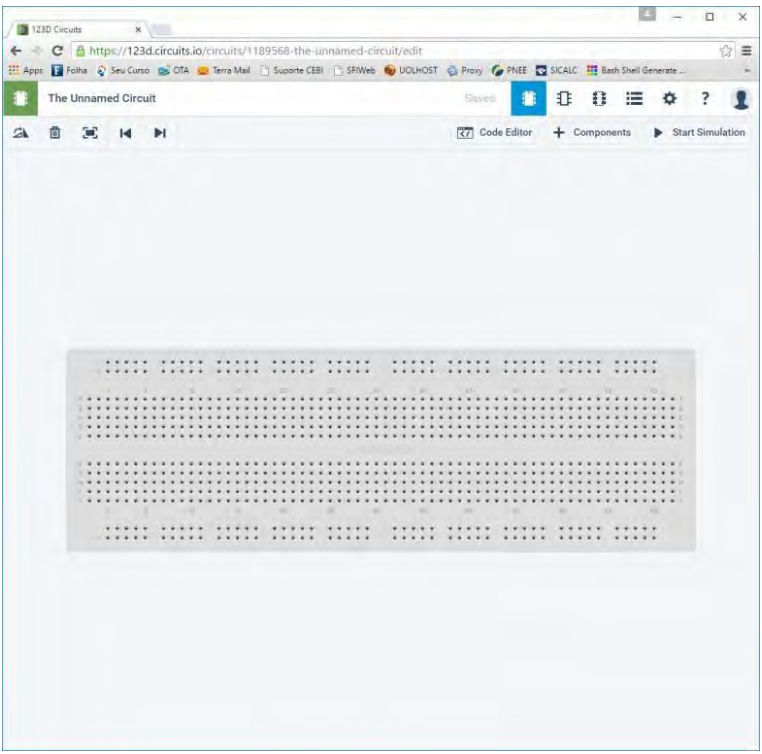


Figura 4.9: Janela com a prot-o-board no meio

As principais funcionalidades disponíveis são:

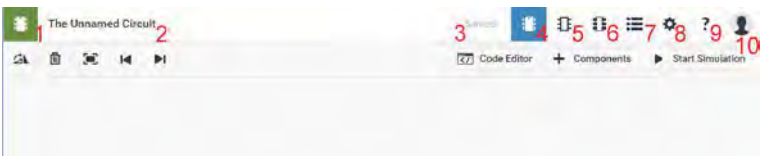


Figura 4.10: Janela com as funcionalidades numeradas da esquerda para a direita

1. Acesso à página principal do seu laboratório, com todos os seus circuitos;

2. Nome do circuito atual;

3. Mensagem de salvamento das alterações realizadas (isso

acontece automaticamente);

4. Visão de laboratório, com a placa de ensaios e demais componentes mostrados como se fossem reais;
5. Visão esquemática, sendo os componentes vistos em formato de esquema utilizando-se a simbologia padrão para eletrônica;
6. Visão de PCB, *Printed Circuit Board* ou Placa de Circuitos Impressos;
7. Lista de materiais;
8. Configurações do seu circuito, tais como nome, resumo tags e convidados a trabalhar nele em conjunto com você;
9. Feedback, onde é possível realizar pesquisas de ajuda;
10. Sua conta, onde é possível ter uma visão geral de todos os circuitos que você já criou e, entre outras coisas, alterar seus dados;

Sempre que acessar o 123D Circuits, você terá a tela onde estão todos seus circuitos. Para voltar a editar seu circuito ou laboratório, basta clicar sobre o link *Edit*, à esquerda do nome dele, quando estiver na página da sua conta.

Figura 4.11: Página com alguns circuitos criados

Há ainda as seguintes ferramentas:

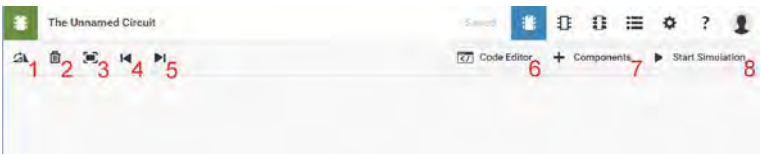


Figura 4.12: Prot-o-board com a segunda linha de ícones numerada da esquerda para a direita

1. Rotacionar o componente selecionado;
2. Apagar o componente selecionado;
3. *Zoom to fit*, ou ajustar o zoom, para que todos os componentes sejam vistos na tela;
4. *Undo*, que desfaz a última alteração a cada clique;
5. *Redo*, que refaz a cada clique última alteração desfeita;
6. *Code editor*, editor de códigos, onde podemos escrever nosso programa para o Arduino;
7. Components, onde podemos selecionar os componentes a serem utilizados na experiência, bastando arrastá-lo sobre o circuito ou placa de ensaios;
8. *Start Simulation*, que faz com que a simulação do circuito tenha início.

Também é possível editar as propriedades dos componentes na janela que aparece automaticamente quando clicamos sobre eles. Cada componente tem propriedades específicas desde simplesmente o nome até a resistência, tensão, corrente e cor, entre outros;

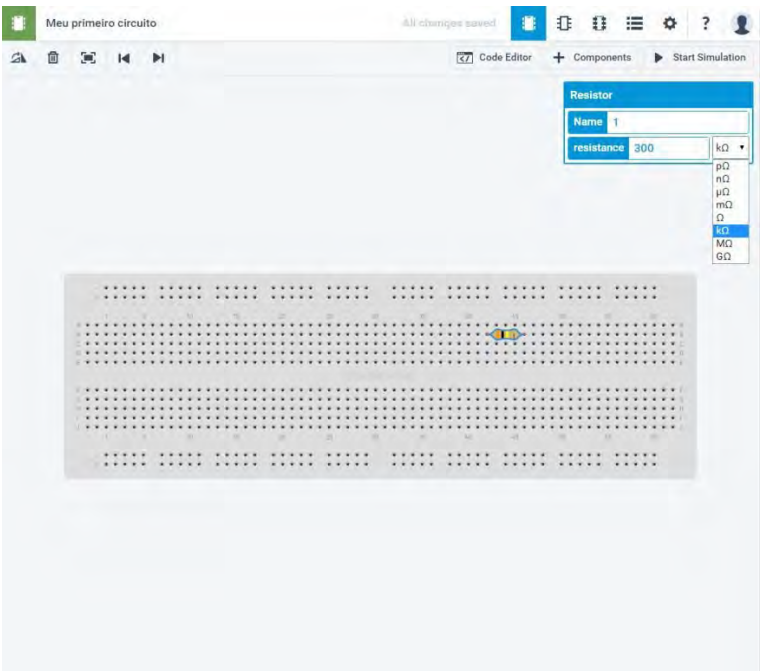


Figura 4.13: Componente qualquer com a janela de propriedades aparecendo

Antes de começar, vale a pena explorar um pouco o simulador, mesmo que sem saber muito para que cada componente serve. Não se preocupe, durante as experiências você vai compreender melhor o uso deles. Arraste-os para a placa de ensaios, mude propriedades, altere entre as visões e as propriedades do circuito. Familiarize-se com o ambiente para que possamos continuar.

Logo você saberá criar circuitos usando o simulador.

4.2 IDE ARDUINO

Se você possui um Arduino e não quer usar o simulador, também será possível realizar todas as experiências constantes deste livro. Basta ter instalado no seu computador o IDE (*Integrated Development Enviroment*, ou Ambiente de Desenvolvimento

Integrado) do Arduino instalado no seu computador.

O IDE é um editor de códigos-fonte que tem como funções compilar e enviar o programa ao Arduino de forma automatizada, dispensando a necessidade de conhecimento de códigos e comandos complexos e muitas vezes longos, que normalmente seriam executados caso o IDE não existisse.

Para obter o IDE Arduino, basta entrar em www.arduino.cc.

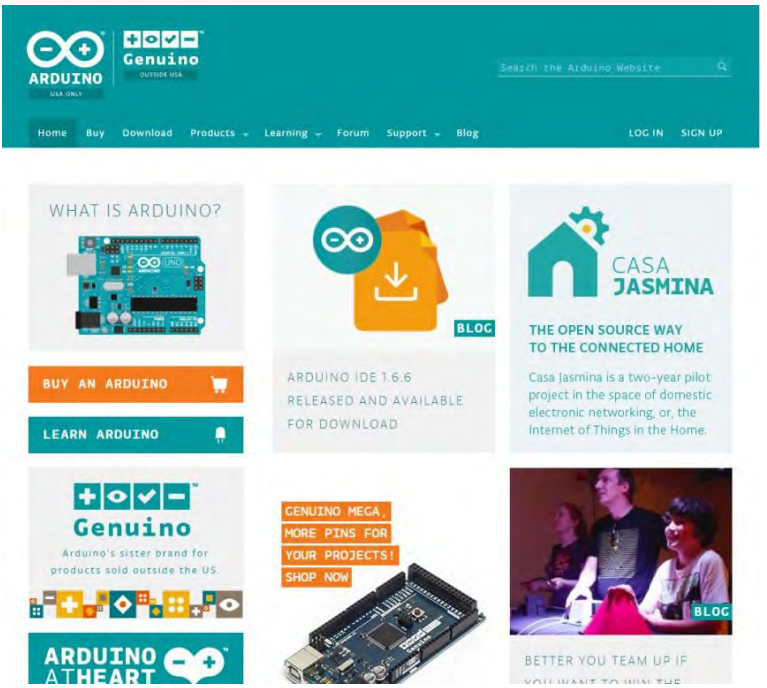


Figura 4.14: O site http://arduino.cc

Clique no link *Download* na porção superior da página.

Existem instalações disponíveis para sistemas operacionais Windows, Linux e Mac OS X. Basta selecionar qual é o sistema e realizar o download. Porém, para a versão Linux, você deve optar pela versão para sistemas de 32-bits ou 64-bits. Em todos os casos, o

driver do conversor USB-Serial presente no Arduino está incluso.

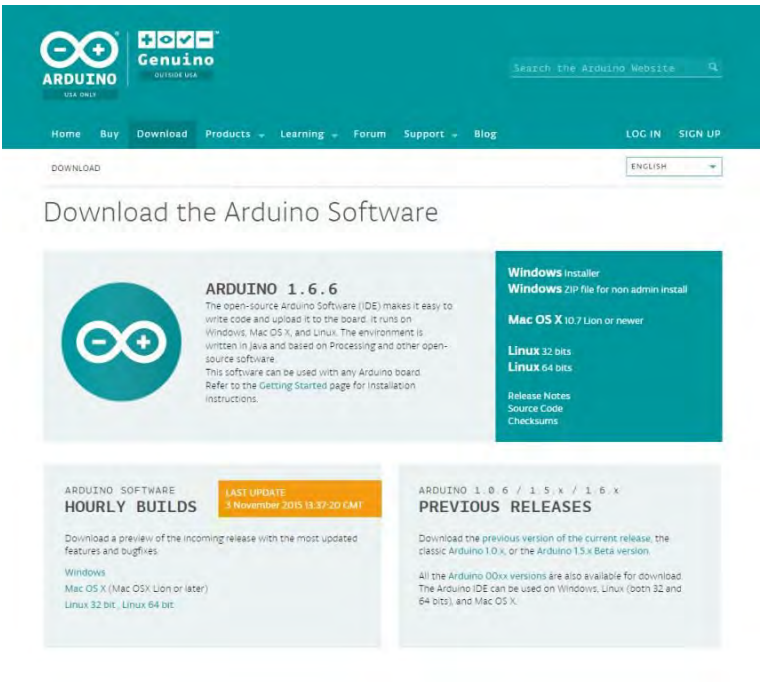


Figura 4.15: O site http://arduino.cc

Para todas as opções de download, você verá uma tela para efetuar uma doação à Arduino Software, que é o grupo que mantém a IDE e algumas bibliotecas. A doação não é obrigatória, mas se você optar por realizá-la, clique no valor e no link *Contribute & Download*. Se não for realizar uma doação, clique em *Just Download*.

A instalação no Windows

No download, dê preferência para a versão *Windows Installer*. Nela virá tudo incluso, e a instalação será mais simples. Após o download realizado, clique duas vezes sobre o arquivo baixado para iniciar a instalação.

A primeira tela será para você conhecer os termos de licença do Arduino Software - não é muito longo e, apesar de estar apenas em inglês, acho interessante lê-lo. Em seguida, clique no botão *I Agree*.

Na tela *Installation Options*, você pode selecionar se deseja ou não instalar o driver USB, criar ou não o ícone no menu iniciar e na área de trabalho, e associar arquivos com extensão `.ino` ao IDE Arduino. Sugiro deixar todas as opções selecionadas.

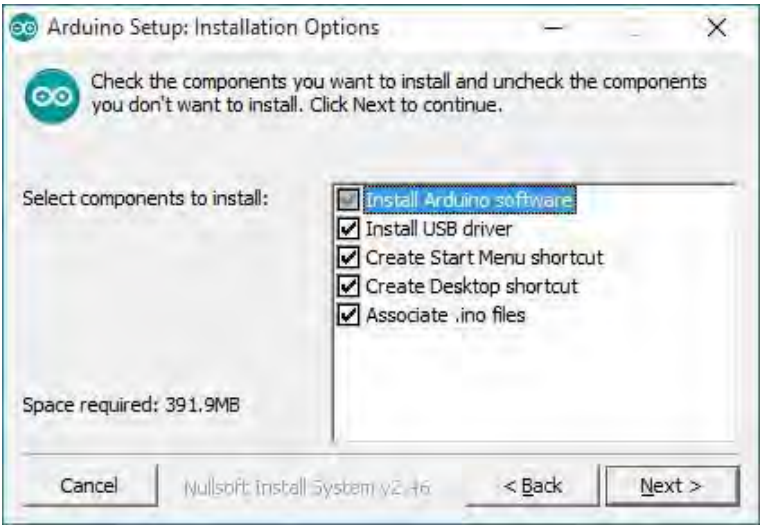


Figura 4.16: A janela Installation Options

Na tela seguinte (*Installation Folder*), você pode selecionar outro local onde o software será instalado ou deixar no local sugerido. É importante saber onde a instalação ocorreu no caso de desejar incluir bibliotecas de terceiros na IDE.

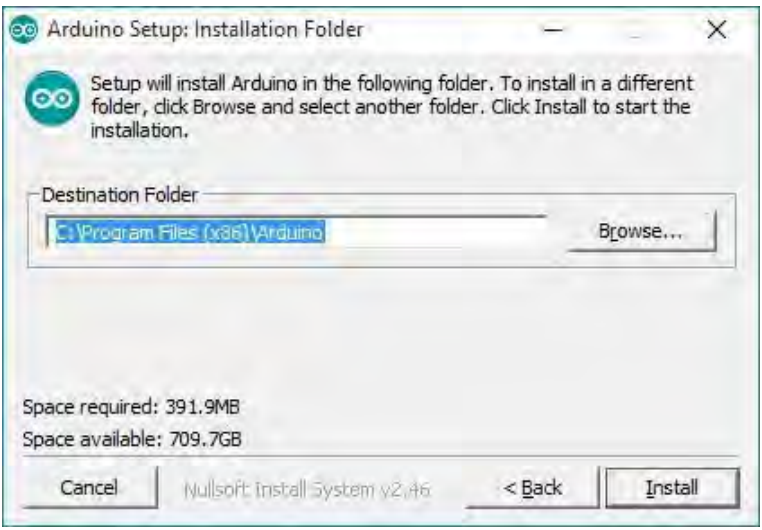


Figura 4.17: A janela Installation Folder

Após clicar no botão *Install*, a instalação finalmente ocorrerá. Após seu fim, clique no botão *Close*. Na sua área de trabalho, caso você não tenha alterado essa opção, você terá o ícone da IDE Arduino.

Figura 4.18: O ícone da IDE Arduino

Pronto, a instalação para Windows está pronta!

A instalação no Linux

Como existem várias distribuições Linux hoje em dia, vou considerar para esse passo a passo a distribuição Debian com ambiente gráfico GNOME. Mas se você estiver usando qualquer outra distribuição, não deve haver quaisquer problemas e a instalação deve ocorrer normalmente.

Antes de continuar, certifique-se de que seu sistema operacional esteja atualizado e que você possua a senha do superusuário (`root`).

Localize a aplicação Terminal do seu Linux ou alterne entre a interface gráfica e terminal usando `CTRL + ALT + F1` - você pode voltar para a interface gráfica novamente usando `CTRL + ALT + F7` .

No terminal digite o comando `su` e em seguida a senha do superusuário `root` .

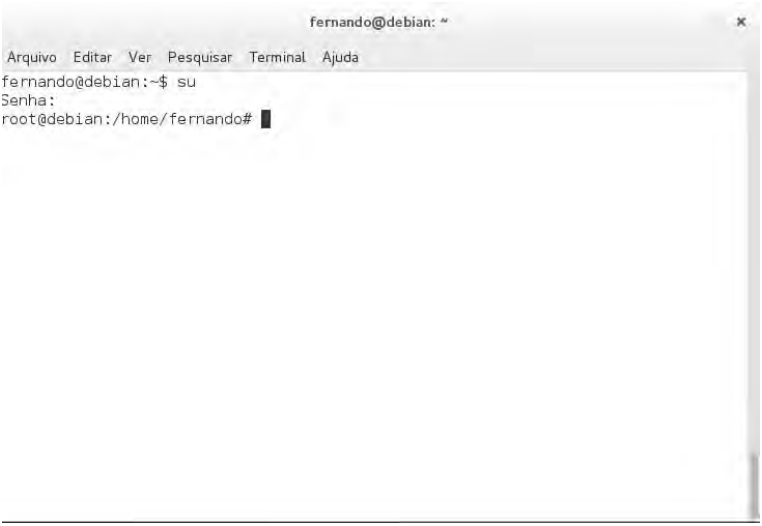


Figura 4.19: O comando su solicitado a digitação de uma senha

Para o Debian, execute a seguinte linha de comandos:

```
apt-get install arduino
```

Para distribuições derivadas do Red Hat, como o Fedora ou o CentOS, execute a seguinte linha de comando:

```
dnf install arduino
```

Serão informadas várias dependências e softwares que serão instalados e, em seguida, a pergunta se deseja continuar. Digite *S* de Sim, e pressione *ENTER*.

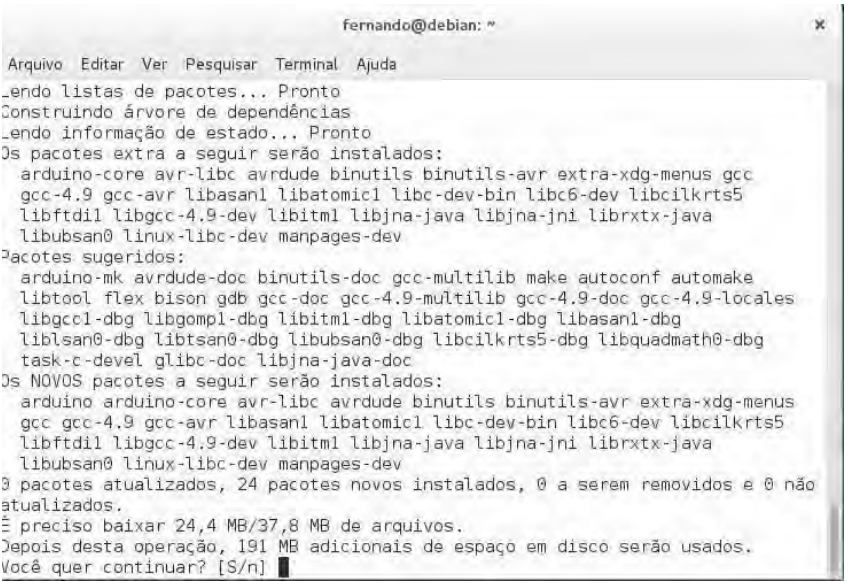


Figura 4.20: O momento de decidir em continuar a instalação ou não

A instalação vai iniciar, basta esperar um pouco.

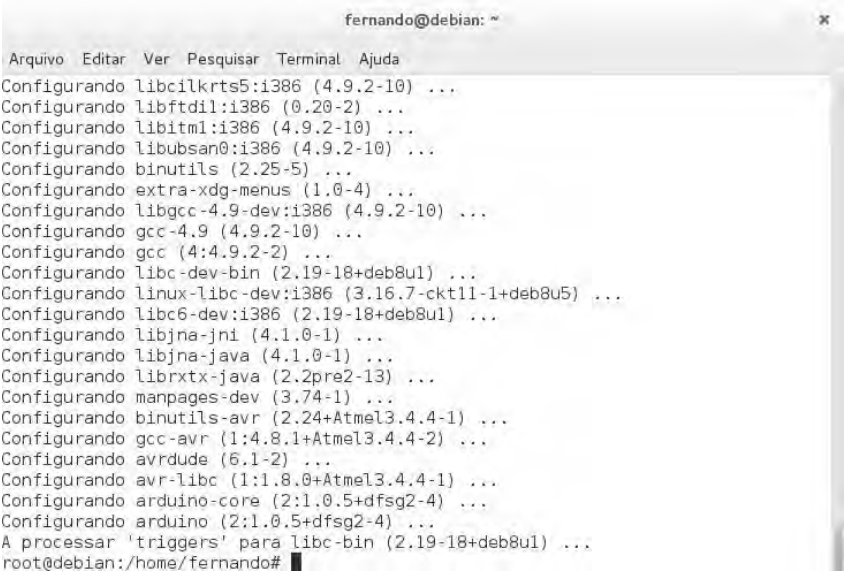


Figura 4.21: Instalação concluída

Após feito isso, você encontrará na área de trabalhos o ícone para o Arduino IDE.

A Instalação no Mac OS X

A instalação do IDE Arduino nos computadores com o sistema operacional Mac OS X é a mais simples de todos os três sistemas operacionais apresentados neste livro.

Faça o download do arquivo para esse sistema operacional. Após o download completo, copie o arquivo para sua Área de Trabalho ou outra localização que achar mais pertinente.

Clique duas vezes para abrir o programa. Apenas no primeiro acesso o arquivo será verificado:

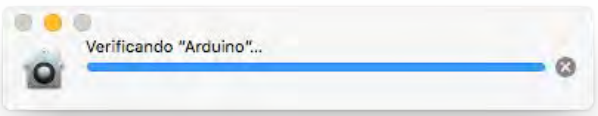


Figura 4.22: Tela de verificação do arquivo

Depois, será exibida uma tela de confirmação, clique em abrir:

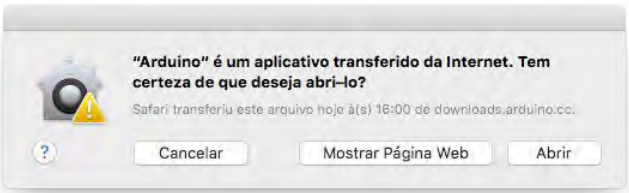


Figura 4.23: Tela de confirmação no Mac OS X

Pronto! Instalado e funcionando. Nos próximos acessos, todas essas confirmações não serão solicitadas.

4.3 A SEGUIR...

Independente de você escolher utilizar a IDE Arduino ou o simulador da Autodesk Inc., é de extrema importância que você saiba o que o IDE pode proporcionar na hora de facilitar a programação. O simulador usa os mesmos conceitos do IDE Arduino, portanto, a seguir, ele será mostrado em profundidade para que você possa entendê-lo e compreenda o seu funcionamento para que possa dominá-lo a partir do seu uso.

Capítulo 5

ENTENDENDO O IDE ARDUINO

Figura 5.1: A tela splash do Arduino IDE

Como já mostrado anteriormente, o IDE é um ambiente para edição de códigos-fonte, onde podemos escrever nossos programas, compilá-los, enviá-los ao Arduino e, se necessário, realizar comunicação serial entre o Arduino e o computador, independente do sistema operacional que você estiver usando.

A seguir, veja a imagem da tela do IDE:

Figura 5.2: O IDE Arduino

Para este livro, usarei a versão 1.6.6 do IDE, mas não acredito que haja grandes alterações para as próximas versões, a não ser melhorias de estabilidade, alguma funcionalidade e desempenho.

Ter em mente todas as funcionalidades do IDE vai ajudar na hora de utilizá-lo, portanto, a seguir estão todos os menus, suas opções e atalhos, seguidos de uma breve explicação da funcionalidade de cada um.

5.1 MENU ARQUIVO E SUAS OPÇÕES

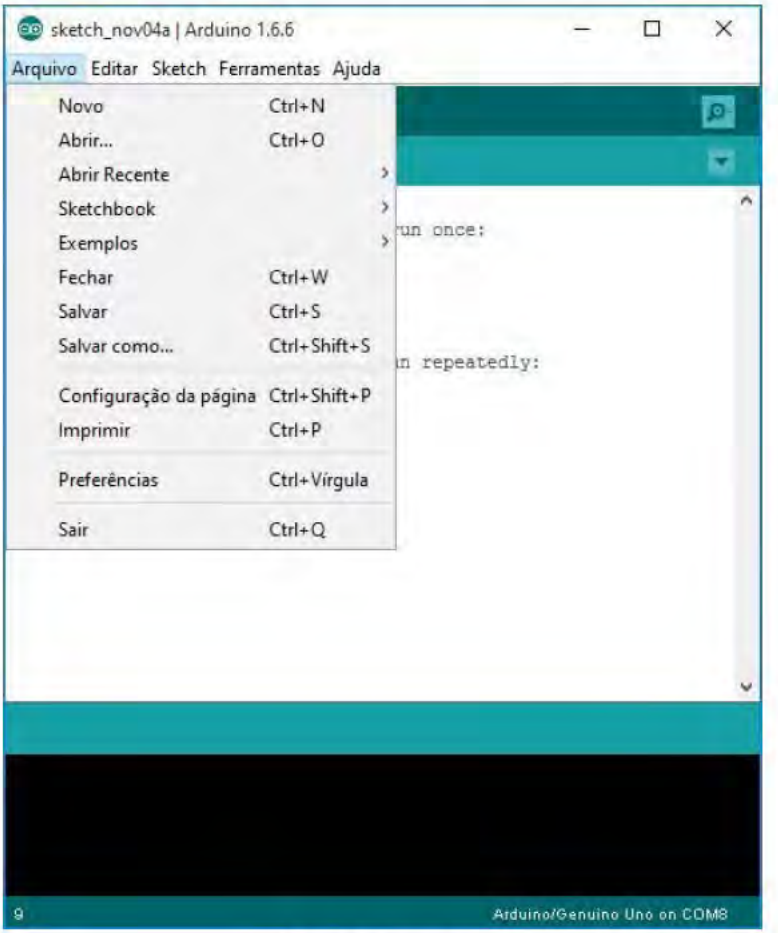


Figura 5.3: O menu Arquivo

Além das funções bem conhecidas, são importantes as tratadas a seguir:

- Opção **Novo** (`CTRL + N`) - Ela criará um novo código-fonte, ou sketch, para já com as funções `setup()` e `loop()` pré-declaradas, mas em branco. Isso é especialmente importante para os iniciantes, já que

assim é praticamente impossível esquecer de uma delas.

- Opção **Sketchbook** - Como o IDE Arduino trabalha com o conceito de **Sketchbook**, que é um local padrão para que você grave seus sketches, sendo que um Sketchbook pode conter um ou mais sketches. Ela mostra todos os projetos que foram utilizados recentemente.

- Opção **Exemplos** - Ela mostrará todos os sketches preexistentes no IDE. Vale a pena dar uma passada por todos eles para verificar as suas funcionalidades e técnicas de programação que foram utilizadas.

- Opção **Preferências** (`CTRL + VÍRGULA`) - Nela existem todas as opções para modificar as configurações do ambiente (veja figura a seguir). Por exemplo, é possível alterar o idioma, tamanho da fonte, mostrar número de linhas etc.

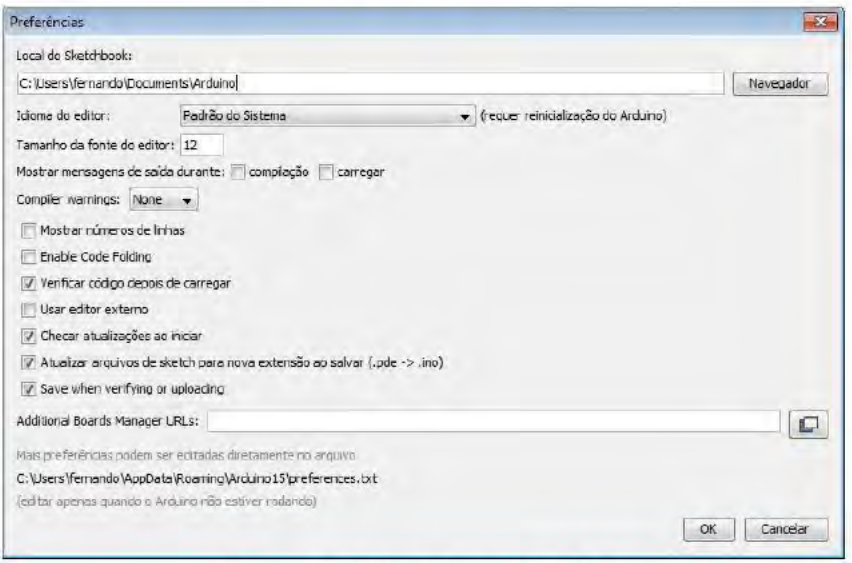


Figura 5.4: A tela de preferências do IDE Arduino

5.2 MENU EDITAR E SUAS OPÇÕES

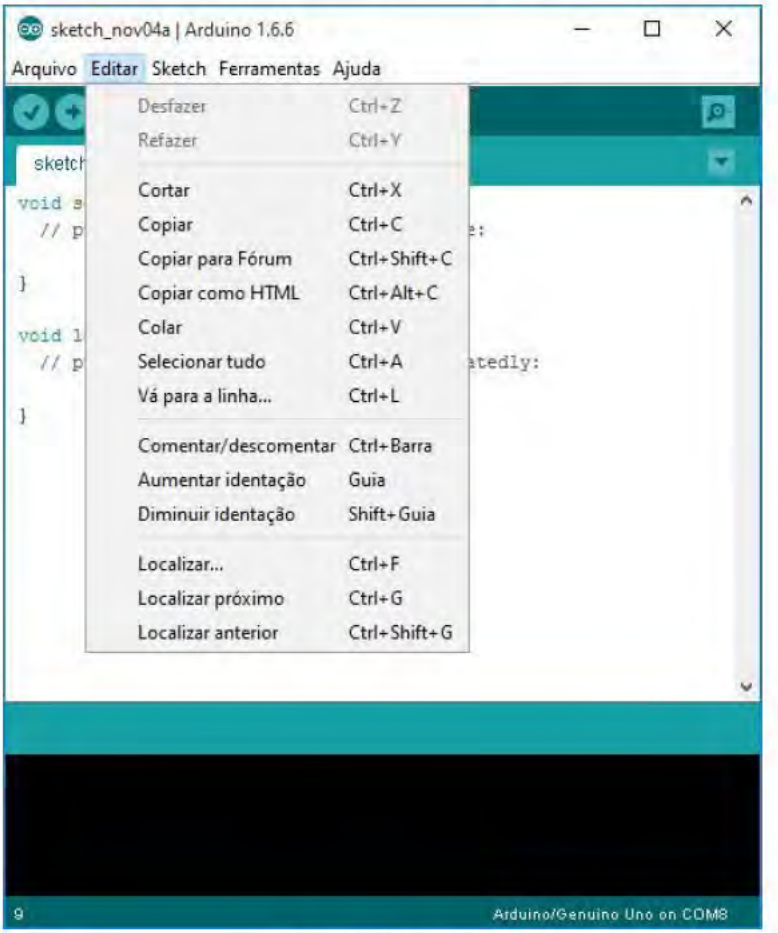


Figura 5.5: O menu Editar

Assim como o menu **Arquivo**, além das funções bem conhecidas do menu Editar, são importantes as seguintes:

- **Copiar para Fórum** (`CTRL + SHIFT + C`) - Com essa opção, você poderá copiar o trecho do código-fonte que estiver selecionado e guardar na área de transferência do sistema operacional, juntamente com as tags

`[code]` e `[/code]` para que possa ser colado em fóruns que normalmente usam essas tags para especificar códigos-fontes em suas páginas.

- **Copiar como HTML** (`CTRL + ALT + C`) - Essa opção vai copiar o trecho do código-fonte que estiver selecionado e guardará na área de transferência do sistema operacional, juntamente com as tags HTML de formatação de parágrafo e fonte para que possa ser colado em páginas na WEB.
- **Vá para a linha** (`CTRL + L`) - Com ela, é possível digitar o número de uma linha e o cursor será posicionado no início dessa linha.
- **Comentar/Descomentar** (`CTRL + Espaço`) - Usando essa opção, você adiciona, caso não exista, a marcação de comentário no código-fonte. Caso a marcação já exista, ela remove-a.
- **Aumentar endentação** (`TAB`) - Adiciona dois espaços em branco no começo da linha a partir da esquerda. Útil para organizar o código-fonte.
- **Diminuir endentação** (`SHIFT + TAB`) - Remove dois espaços em branco, caso existam, no começo da linha a partir da esquerda. Útil para organizar o código-fonte.

5.3 MENU SKETCH E SUAS OPÇÕES

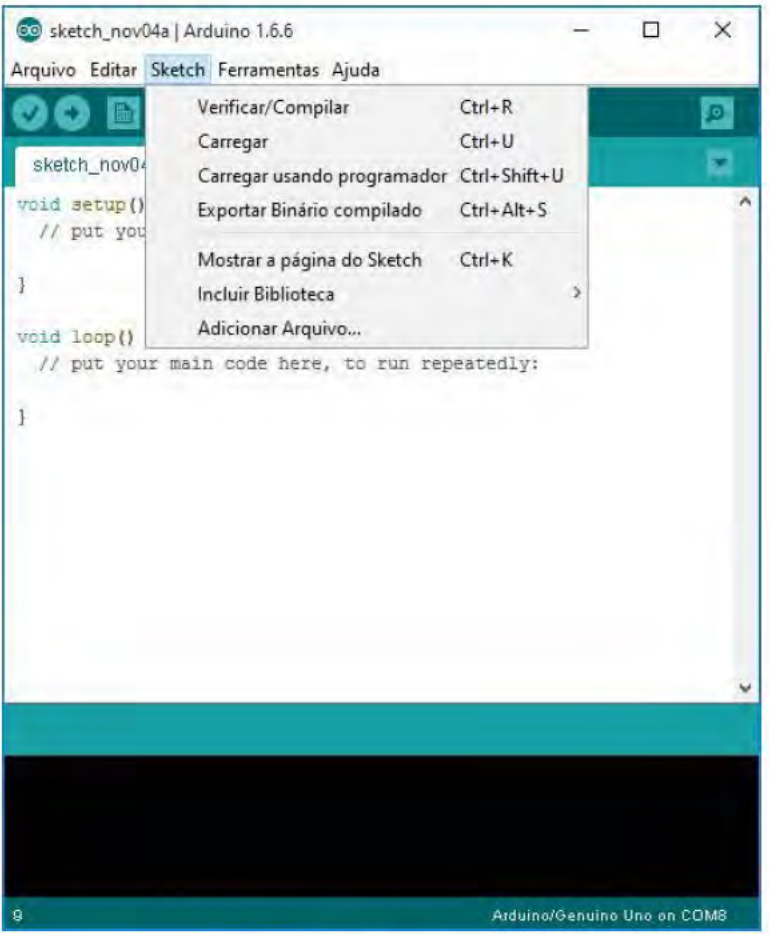


Figura 5.6: O menu Sketch

O menu **Sketch** possui opções importantes para a compilação e carregamento dos programas para o Arduino, que são as seguintes:

- **Verificar/Compilar** (`CTRL + R`) - Essa opção verifica se há erros no código-fonte e executa a sua compilação.
- **Carregar** (`CTRL + U`) - Esta enviará o sketch compilado para a placa Arduino.
- **Carregar usando o programador** (`CTRL + SHIFT +`

U) - Ela vai executar o carregamento de programa usando diretamente o programador AVR.

- **Exportar Binário Compilado** (CTRL + ALT + S) - Cria o arquivo binário que seria carregado ao Arduino na mesma pasta onde está salvo o sketch. Com esse arquivo, você pode carregar para o Arduino usando métodos alternativos ao IDE.
- **Mostrar a página do Sketch** (CTRL + K) - Mostra/abre a pasta onde o sketch foi salvo.
- **Incluir Biblioteca** - Com ela, é possível selecionar entre as bibliotecas padrão do Arduino Software para que sejam inseridas no código-fonte.
- **Adicionar arquivo** - Permite adicionar um arquivo, normalmente de código-fonte também, ao arquivo atual e na posição do cursor.

5.4 MENU FERRAMENTAS E SUAS OPÇÕES

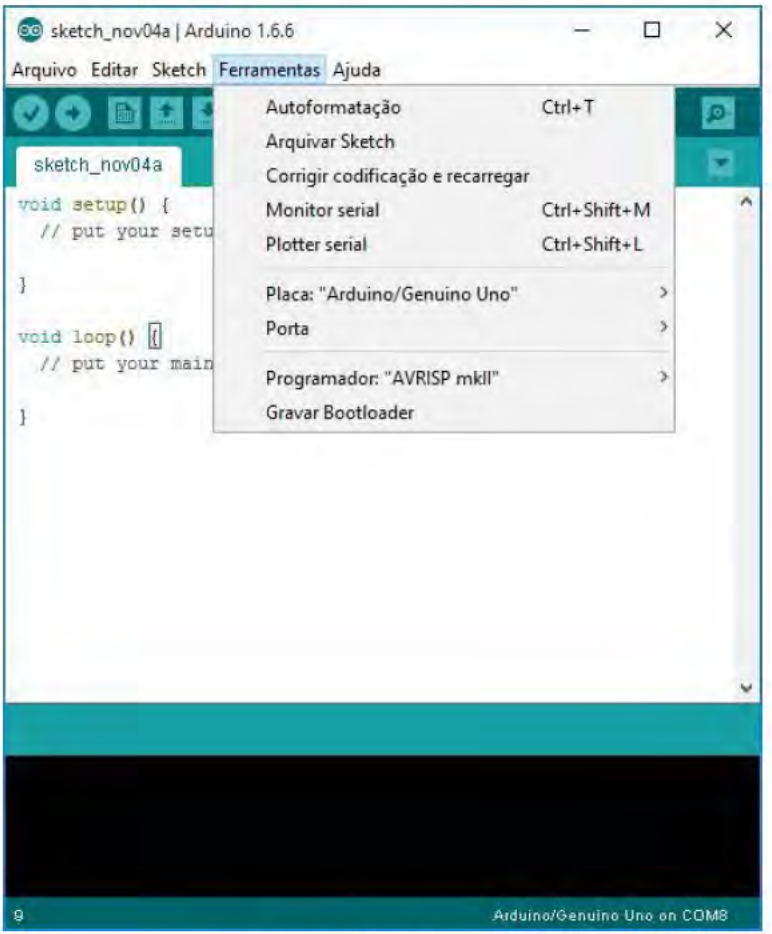


Figura 5.7: O menu Ferramentas

- **Autoformatação** (CTRL + T) - Essa opção realiza, em todo o código-fonte, indentação (adicionar dois espaços antes da linha depois de abrir chaves e remover os dois espaços depois de fechar chaves).
- **Arquivar sketch** - Esta grava todos os arquivos do sketch em uma pasta compactada (formato ZIP).
- **Corrigir codificação e carregar** - Esta verifica alguns

erros no código-fonte e corrige-os, como por exemplo, falta de abertura ou fechamento de chaves. Também executa indentação.

- **Monitor Serial** (`CTRL + SHIIFT + M`) - Abre a interface de comunicação serial entre o computador e o Arduino (veja figura seguinte).

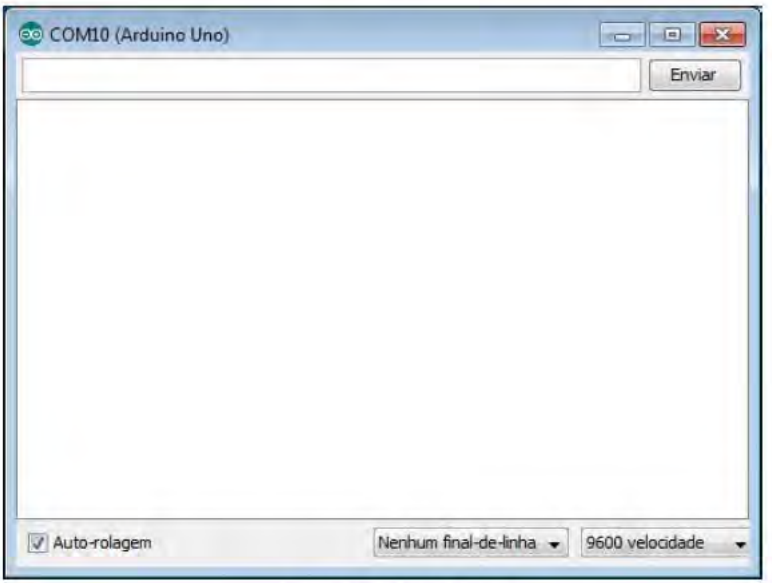


Figura 5.8: O Serial Monitor

- **Plotter Serial** (`CTRL + SHIFT + L`) - Com essa opção, é permitido que os dados recebidos serialmente do Arduino sejam mostrados graficamente na tela, e não somente em formato de texto, como no Monitor Serial.
- **Placa** - Ela permite selecionar qual a placa Arduino a ser utilizada junto com o IDE.
- **Porta** - Permite selecionar a porta COM em que a placa Arduino está conectada (porta COM virtual –

conversor USB-Serial embarcado no Arduino).

- **Programador** - Ela permite selecionar o programador a ser usado para compilar e carregar o sketch para a placa Arduino, caso deseje usar algum que não seja o padrão já pré-selecionado.
- **Gravar bootloader** - Com ela, é possível gravar o bootloader para a placa Arduino caso seja necessário, como por exemplo, no caso de troca do microcontrolador.

Sobre o Monitor Serial, é preciso ressaltar as opções de que ele é composto:

- **Área de entrada de dados:** o que for digitado aqui será enviado serialmente ao Arduino após pressionar *ENTER* ou clicar no botão *Enviar*.
- **Botão Enviar:** envia o que estiver na área de entrada de dados ao Arduino.
- **Área de resposta:** onde é mostrado o retorno que o Arduino eventualmente envia ao computador, o que dependerá da programação que você realizar nele. Caso a área de resposta seja preenchida, o texto mais antigo automaticamente rola para cima.
- **Lista de seleção para indicar o tipo de terminador de linha a ser utilizado:** o que pode depender da programação que estiver sendo usada no Arduino.
- **Lista de seleção com todas as velocidades de transmissão de dados possíveis entre o Arduino e o computador:** também depende da programação que você fizer no Arduino, mas deve ser igual em seu

programa e no Monitor Serial para que os dados sejam inteligíveis.

5.5 MENU AJUDA E SUAS OPÇÕES

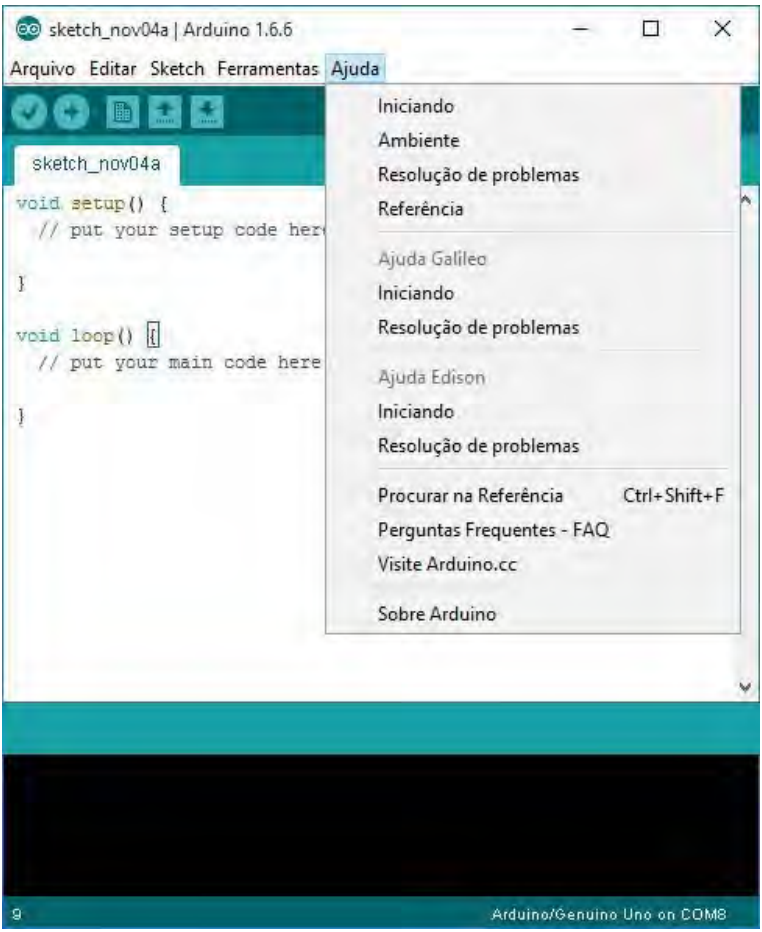


Figura 5.9: O menu Ajuda

No menu **Ajuda**, você encontrará ligações para as páginas de internet correspondentes ao suporte geral do IDE e da Linguagem para Arduino.

5.6 A SEGUIR...

Agora que você já sabe todas as opções disponíveis no IDE Arduino, que possui muitas ferramentas super úteis na hora de ajudar a programar, a seguir serão apresentadas uma referência da linguagem C e várias experiências para colocarmos tudo em prática. Você terá todas as principais funções e comandos necessários para qualquer programa a ser desenvolvido.

Capítulo 6

COMO COLOCAR TUDO EM PRÁTICA

Consideraremos que você possa querer montar todas as experiências, tanto usando uma placa de ensaios como o simulador mostrado anteriormente.

Para montar as experiências, siga os passos descritos em cada uma dela, considerando o esquema mostrado a seguir.

6.1 NA PLACA DE ENSAIOS OU PROT-O-BOARD

A placa de ensaios (ou *prot-o-board*) é uma placa com orifícios que se conectam entre si por um anteparo metálico contido dentro dela, como mostrado nas figuras seguintes. Essa é uma operação bem simples, basta ter atenção e conferir sempre as ligações antes de ativar ou inicializar qualquer coisa.

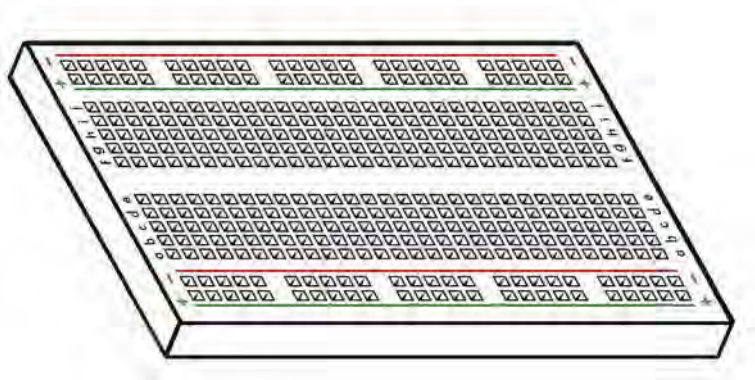


Figura 6.1: A placa de ensaios ou prot-o-board (Cortesia: Victor Badolato Athayde)

Para usá-la, conectamos os componentes e fios nos orifícios de maneira que formem contato entre eles nos terminais desejados, considerando as ligações entre eles como vemos na figura:

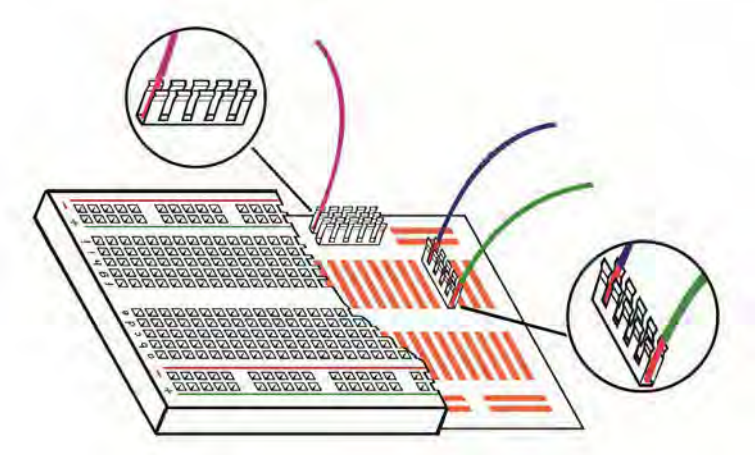


Figura 6.2: Os contatos dentro da placa de ensaios (Cortesia: Victor Badolato Athayde)

Veja um exemplo:

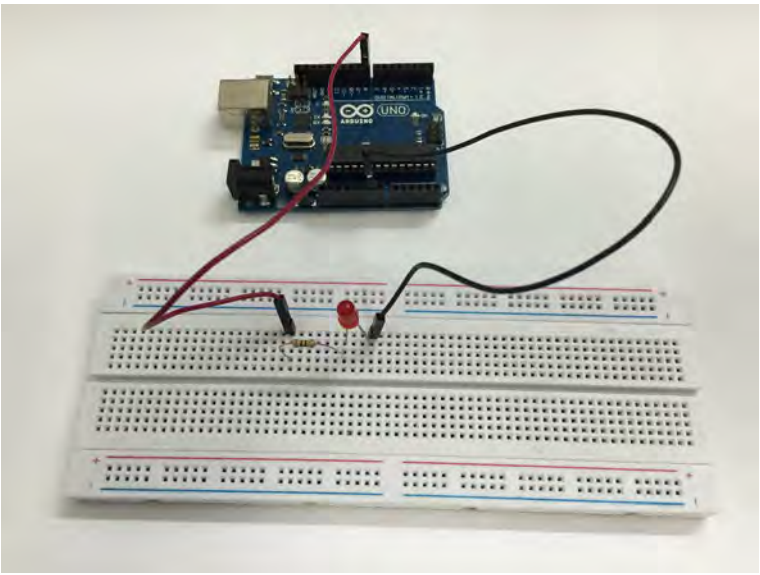


Figura 6.3: Um resistor conectado a um LED e ambos conectados em um Arduino usando uma placa de ensaios

6.2 NO SIMULADOR

No simulador, a mecânica é a mesma. Para ligar os fios virtuais, clique sobre o orifício da placa de ensaios, e depois no pino ou outro orifício que desejar. Não é necessário manter o botão do mouse pressionado, clique uma vez em um ponto e estique o fio virtual até o outro ponto.

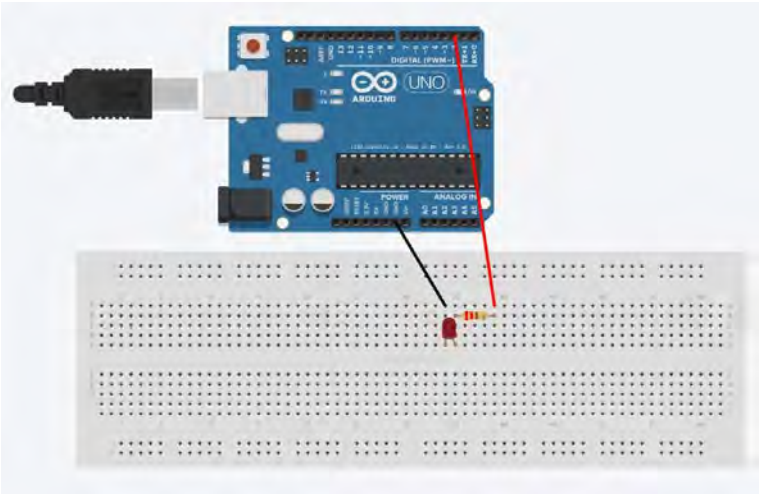


Figura 6.4: Um resistor conectado a um LED e ambos conectados em um Arduino usando o simulador

Para que os componentes fiquem ligados, basta colocar os terminais deles sobre os orifícios da placa de ensaios virtual, que tem os contatos iguais à placa de ensaios real, mostrados anteriormente.

6.3 A SEGUIR...

A seguir, vamos ao que interessa: colocar as mãos na massa! Adiante você encontrará tudo o que precisa saber sobre a linguagem para o Arduino e como utilizá-la para realizar várias experiências em que terá a oportunidade de exercitar os conceitos por trás da tecnologia e compreendê-lo.

Capítulo 7

PROGRAMANDO O ARDUINO

A *linguagem C* é a base para a programação para Arduino. Apesar de ter alguns comandos específicos para o microcontrolador, é basicamente a mesma utilizada em programação para computadores. Se você tiver alguma familiaridade com programação usando linguagem C em computadores, não terá qualquer dificuldade, e mesmo se você não tiver, também não terá problemas, porque as experiências serão autoexplicativas.

Há diversas formas de se pensar em qual seria o ponto inicial para se aprender a usar o Arduino para criar coisas. Poderia ser a partir da eletrônica geral do equipamento, ou a partir da linguagem utilizada para programá-lo. Resolvi seguir o segundo caminho.

Conhecendo bem a linguagem de programação e como usá-la para atuar sobre o hardware, os conceitos eletrônicos ficarão mais fáceis de fixar, e a lógica de como tudo funciona abrirá portas que demorariam mais se o foco fosse a eletrônica.

Aliás, considero que o Arduino é essencialmente um exercício de programação, já que seus módulos e a facilidade de encaixá-los e colocá-los para funcionar deixa a eletrônica trivial. Pois bem, vamos programá-lo!

7.1 SINTAXE BÁSICA PARA A LINGUAGEM

Toda linguagem de programação tem seus "detalhes" mais básicos. A linguagem de programação para o Arduino, que como já vimos é baseada na linguagem C, também tem esses detalhes, que veremos a seguir.

Ponto e vírgula

O ponto e vírgula (`;`) é usado para terminar as linhas de comandos, mas há exceções. Por exemplo, não é utilizado depois de declaração de funções e nem na declaração das estruturas de controle ou repetição.

Esquecê-la causará um erro de compilação e esse erro poderá ou não apontar esse esquecimento diretamente. Você pode conferir o uso de ponto-e-vírgula em todas as experiências.

Chaves

Indicam o início quando `{` , e término quando `}` , de blocos de comandos. Delimita a porção de código para uma função, estrutura de seleção ou repetição. Considero como o símbolo ou comando mais importante de toda a linguagem C. Você pode conferir o uso de chaves em todas as experiências.

Comentários

Eles são essenciais para manter o código-fonte do seu programa organizado e garantir que você vai entendê-lo depois.

Usar duas barras `//` indica que delas até o final da linha é um comentário. Também, usar `/*` e `*/` indica que tudo o que estiver escrito entre barra e asterisco e depois asterisco e barra é um comentário.

Veja um exemplo:

```
/* Todo este texto
é considerado como
comentário */

void setup() {
    // esta linha é um comentário
}

void loop() {
    // esta linha é um comentário
}
```

7.2 A SEGUIR...

O que foi mostrado aqui será usado para qualquer programa na linguagem C para o Arduino. É o básico e indispensável. A seguir, começaremos a estudar as estruturas da linguagem colocando-as em prática por meio das experiências propostas.

CAPÍTULO 8

ESTRUTURA PRINCIPAL DA LINGUAGEM

Toda linguagem de programação tem seus componentes básicos e essenciais. Vimos anteriormente os básicos e aqui veremos os essenciais. A linguagem C para computadores, por exemplo, tem a obrigatoriedade da função `main()`, que é o que torna um programa executável depois de compilado. Para o Arduino, a função `main()` não existe, mas no seu lugar, temos outras duas: `setup()` e `loop()`, detalhadas na sequência.

setup()

A função `setup()` é a primeira coisa a ser executada quando o Arduino é ligado, e é executada apenas uma vez. É usada para inicializar variáveis, modos de atuação dos pinos, declaração de objetos e uso de bibliotecas.

Você verá a função `setup()` em todas as experiências.

loop()

Após executar a função `setup()`, é a função `loop()` que passa a ser executada do primeiro ao último comando. Depois ela volta para o primeiro comando e repete tudo novamente infinitamente - ou melhor, enquanto o Arduino estiver ligado.

Caso o Arduino seja desligado, a função `setup()` será

executada novamente e depois a função `loop()` entrará em cena. Para exemplificar bem esse funcionamento, sugiro realizarmos a experiência a seguir: apenas acender o LED embutido no pino digital 13.

8.1 EXPERIÊNCIA Nº 01 - ACENDER O LED DO PINO 13

Essa é a experiência mais básica que se pode fazer com o Arduino, por isso considera-se que seja o "*Hello World*" ("Olá Mundo!"). A ideia aqui é conhecer o Arduino, seu Ambiente de Programação e sua linguagem de programação na prática.

Como você já viu anteriormente, o Arduino possui integrado ao seu pino digital 13 um LED que pode ser controlado bastando enviar o sinal alto (`HIGH`) para acendê-lo, ou sinal baixo (`LOW`) para apagá-lo, e é exatamente o que faremos.

O que é necessário

1 x Arduino UNO

Esquema de montagem

Não há necessidade de qualquer componente exceto o próprio Arduino.

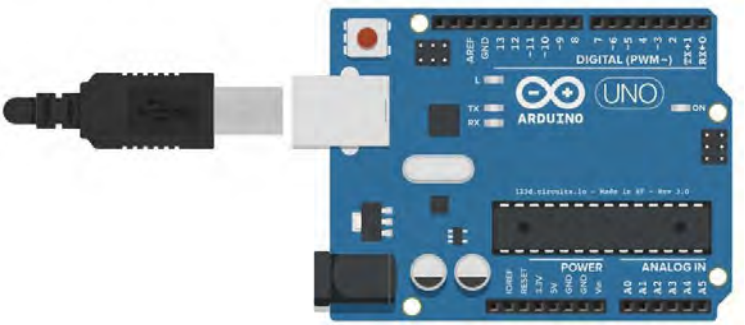


Figura 8.1: Esquema de montagem da experiência nº 01

Programação

```
void setup() {
    pinMode(13,OUTPUT);
}

void loop() {
    digitalWrite(13,HIGH);
}
```

O que foi feito

Na função `setup()`, ajustamos o modo de operação do pino digital 13 do Arduino para saída usando o comando `pinMode(13,OUTPUT)`. Dessa forma, poderemos colocar ou não tensão nele usando as constantes `HIGH` e `LOW`.

Na função `loop()`, que é executada infinitamente enquanto o Arduino estiver ligado, como já vimos anteriormente, usamos o comando `digitalWrite(13,HIGH)` para indicar que haverá saída de tensão positiva de 5V (`HIGH`) pelo pino digital 13 do Arduino.

Resultado esperado

Após fazer upload para o Arduino, o programa fará com que o LED do pino digital 13 permaneça aceso o tempo todo.

Ótimo, você acabou de verificar como um programa para Arduino deve funcionar. Primeiro, a função `setup()` entra em ação, é executada apenas uma vez logo que o Arduino é ligado. Em seguida, é a vez da função `loop()`, que ficará rodando em quanto o Arduino estiver ligado.

Vamos tentar uma segunda experiência, dessa vez fazendo com que o LED do pino digital 13 mude de estado.

8.2 EXPERIÊNCIA Nº 02 - PISCAR O LED DO PINO 13

Esta é a segunda mais básica experiência que se pode fazer com o Arduino, e consiste em uma continuação da experiência anterior.

Na primeira experiência, conseguimos ligar o LED do pino digital 13. Agora, vamos desligá-lo também a cada período de tempo que definiremos na programação. Com isso, o LED piscará.

O que é necessário

1 x Arduino UNO

Esquema de montagem

Não há necessidade de qualquer componente exceto o próprio Arduino.

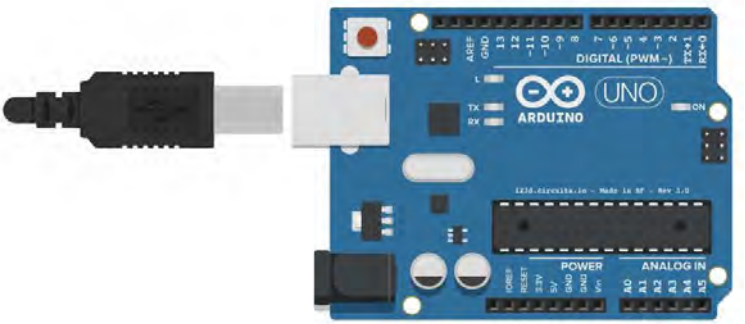


Figura 8.2: Esquema de montagem da experiência nº 02

Programação

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

void loop() {
    digitalWrite(LED_BUILTIN,HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN,LOW);
    delay(1000);

}
```

O que foi feito

Na função `setup()`, ajustamos o modo de operação do pino digital 13 como saída, usando o comando `pinMode(13,OUTPUT)`. Na função `loop()`, usamos o comando `digitalWrite(13,HIGH)` para colocar no pino digital 13 um valor de saída alto (`HIGH`), correspondente a 1. Com isso, ele apresentará tensão positiva de 5 volts, fazendo com que o LED acenda.

Com o comando `delay(1000)`, o microcontrolador entrará em estado de pausa na execução do programa durante um segundo, já que, como vimos anteriormente na referência da linguagem C

para o Arduino, o tempo determinado entre parênteses para esse comando deve ser em milissegundos. Ou seja, são 1.000 milissegundos, o que é igual a 1 segundo.

Continuando, colocamos a linha de comando `digitalWrite(13,LOW)` onde colocamos no pino digital 13 um valor de saída baixo (`LOW`), correspondente a 0. Com isso, ele apresentará tensão neutra (0 volts) e isso apagará o LED.

Com a repetição do comando `delay` , teremos mais uma pausa de um segundo antes de o programa retornar novamente ao início da função `loop()` e permanecer executando todo o código indeterminadamente até que o Arduino seja desligado, ou receba nova programação.

Resultado esperado

Após fazer upload para o Arduino, o programa fará com que o LED do pino digital 13 permaneça aceso por um segundo e apagado também por um segundo.

A seguir, veja as constantes que você encontrou no código-fonte dessas duas primeiras experiências:

- `HIGH`

  Indica que no pino no modo saída apresentará uma tensão acima de 3 volts para referência de 5V e acima de 2 volts para referência de 3,3 volts. Caso o pino esteja no modo entrada, ele poderá receber como leitura as mesmas tensões de saída.

- `LOW`

  Indica que no pino no modo saída apresentará uma tensão abaixo de 3 volts para referência de 5V e abaixo

de 2 volts para referência de 3,3 volts. Caso o pino esteja no modo entrada ele poderá receber como leitura as mesmas tensões de saída.

- `INPUT`

  Indica que o pino será utilizado como entrada. Ou seja, poderá realizar uma leitura de tensão.

- `OUTPUT`

  Indica que o pino será utilizado como saída. Ou seja, apresentará uma tensão de saída.

- `LED_BUILTIN`

  Substitui o valor do pino onde o LED integrado está presente. Normalmente o número 13. Para fixar bem vamos repetir a mesma experiência anterior, mas usando um LED externo e não o que está embutido no pino digital 13 do Arduino:

8.3 EXPERIÊNCIA Nº 03 - PISCAR UM LED EXTERNO

O princípio dessa experiência é o mesmo das duas anteriores: controlar um LED. Mas agora ligaremos um LED a uma placa de ensaios (*prot-o-board*) junto a um resistor, e vamos controlá-lo por meio de um pino digital qualquer do Arduino. O desenho da montagem sugere que o LED seja vermelho, mas pode ser da cor de sua preferência.

O pino digital utilizado pode variar de acordo com sua vontade, apenas é preciso ter atenção de ajustá-lo corretamente na programação. Isso vale o mesmo para o pino GND, que pode ser qualquer um dos três disponíveis no Arduino UNO.

O que é necessário

1 x Arduino UNO

1 x LED

1 x Resistor de 220Ω

Esquema de montagem

Coloque o LED na placa de ensaios como mostrado na figura a seguir, lembrando do que foi exposto anteriormente sobre como usar a placa de ensaios.

Usando um cabinho, ligue o terminal negativo do LED a um dos pinos GND disponíveis no Arduino. Para identificar no LED qual é o terminal negativo e qual é o positivo, saiba que o negativo é o mais curto, já o positivo é o mais comprido.

Ligue o terminal positivo do LED a um resistor de, pelo menos, 220Ω como na figura a seguir. Usando um cabinho, ligue o outro terminal do resistor ao pino digital 2 do Arduino:

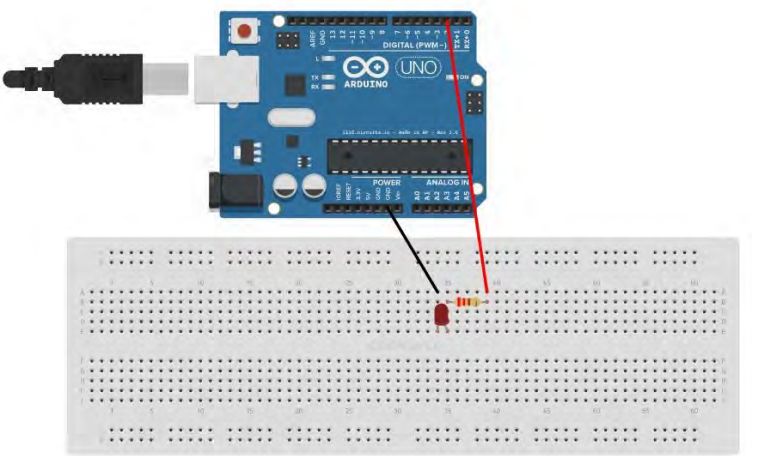


Figura 8.3: Esquema de montagem da experiência nº 03

Programação

```
void setup() {
    pinMode(2,OUTPUT);
}

void loop() {
    digitalWrite(2,HIGH);
    delay(1000);
    digitalWrite(2,LOW);
    delay(1000);

}
```

O que foi feito

Note que a programação é idêntica à da experiência anterior, apenas mudamos o número do pino digital que vamos utilizar, mas vamos lá novamente.

Na função `setup()`, ajustamos o modo de operação do pino digital 2 como saída usando o comando `pinMode(2,OUTPUT)`. Na função `loop()`, usamos o comando `digitalWrite(2,HIGH)` para colocar no pino digital 2 um valor de saída alto (`HIGH`), correspondente a 1. Com isso, ele apresentará tensão positiva de 5 volts, fazendo com que o LED acenda.

Com o comando `delay(1000)`, o microcontrolador entrará em estado de pausa na execução do programa durante um segundo, já que, como vimos anteriormente na referência da linguagem C para o Arduino, o tempo determinado entre parênteses para esse comando deve ser em milissegundos. Ou seja, 1.000 milissegundos, que é igual a 1 segundo.

Continuando, colocamos a linha de comando `digitalWrite(2,LOW)` onde colocamos no pino digital 2 um valor de saída baixo (`LOW`), correspondente a 0. Com isso, ele apresentará tensão neutra (0 volts) e isso apagará o LED;

Com a repetição do comando `delay` , teremos mais uma pausa de um segundo antes de o programa retornar novamente ao início da função `loop()` e permanecer executando todo o código indeterminadamente até que o Arduino seja desligado, ou receba nova programação.

Resultado esperado

Após fazer upload para o Arduino, o programa fará com que o LED conectado ao pino digital 2 permaneça aceso e apagado por um segundo.

Agora que está claro e dominado como acionar LEDs usando o Arduino, vamos criar um sequência de 3 LEDs externos na próxima experiência.

8.4 EXPERIÊNCIA Nº 04 - SEQUENCIAL COM 3 LEDS

Seguindo o aumento de complexidade nessa experiência, vamos controlar 3 LEDs em uma montagem. A lógica é a mesma para apenas um LED, mas as possibilidades são quase infinitas do que se pode fazer em relação à sequência de acender e apagar os LEDs.

Sugestões de projetos não faltam, tais como um semáforo simples, com as três luzes que compõem um. Saltar de três para seis LEDs não deve ser muito difícil, basta um pouco de paciência e tempo para entender melhor a programação, e construir um cruzamento com dois semáforos também pode ser um exercício interessante.

Vale a pena parar um pouco nesta experiência e soltar a imaginação para exercitar o controle de LEDs externos ao Arduino.

Na figura da montagem, as cores sugeridas para os LEDs são

vermelho, verde e azul, mas você pode usar todos da mesma cor ou quaisquer cores que encontrar ou estiverem disponíveis. No caso do semáforo, você pode trocar o azul por um LED amarelo, por exemplo.

O que é necessário

1 x Arduino UNO

3 x LEDs

3 x Resistores de 220Ω

Esquema de montagem

Coloque os LEDs na placa de ensaios conforme mostrado na figura do esquema de montagem. Usando um cabinho, ligue o terminal negativo do primeiro LED (sugiro que cada LED tenha uma cor diferente, então esse pode ser o vermelho), que como vimos na experiência nº 03, é o terminal mais longo, a um dos pinos GND disponíveis no Arduino. Em seguida, faça o mesmo com o terminal negativo do segundo LED (verde, se seguir a minha sugestão) ao terminal negativo do primeiro LED (vermelho), e finalmente ligue o terminal negativo do terceiro LED (azul) ao terminal negativo do segundo LED (verde).

Agora ligue o terminal positivo do primeiro LED a um resistor de pelo menos 220Ω e, em seguida, ligue o outro terminal do resistor, usando um cabinho, ao pino digital 4 do Arduino.

Seguindo o mesmo esquema do primeiro LED, ligue o terminal positivo do segundo LED a um resistor de pelo menos 220Ω e ligue o outro terminal do resistor, usando um cabinho, ao pino digital 3 do Arduino.

Finalizando, ligue o terminal positivo do terceiro LED a um

resistor de pelo menos 220Ω e ligue o outro terminal do resistor, usando cabinho, ao pino digital 2 do Arduino.

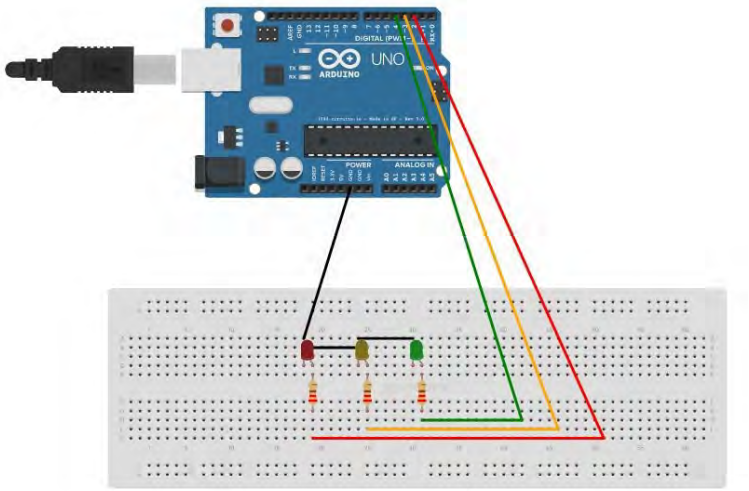


Figura 8.4: Esquema de montagem da experiência nº 04

Programação

```
void setup() {
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    pinMode(4,OUTPUT);
}

void loop() {
    digitalWrite(2,HIGH);
    delay(1000);
    digitalWrite(2,LOW);
    digitalWrite(3,HIGH);
    delay(1000);
    digitalWrite(3,LOW);
    digitalWrite(4,HIGH);
    delay(1000);
    digitalWrite(4,LOW);
}
```

O que foi feito

Na função `setup()` , como você já deve ter notado que é de praxe, ajustamos os pinos 2, 3 e 4 como saída usando os comandos `pinMode(2,OUTPUT)` , `pinMode(3,OUTPUT)` e `pinMode(4,OUTPUT)` . Isso nos permitirá colocar ou não tensão neles.

Já na função `loop()` usamos os comandos `digitalWrite` para colocarmos nos pinos digitais 2, 3 e 4 um valor de saída alto (`HIGH`), correspondente a 1. Com isso, eles apresentarão tensão positiva de 5 volts e isso acenderá os LEDs. Como já vimos na experiência anterior, usamos `delay(1000)` para fazer uma pausa na execução do programa de um segundo.

Na sequência, usamos o comando `digitalWrite` novamente para colocarmos nos pinos digitais 2, 3 e 4 um valor de saída baixo (`LOW`), correspondente a 0. Com isso, eles apresentarão tensão neutra (0 volts) e isso apagará os LEDs.

Resultado esperado

O “segredo” mesmo está na sequência em que esses comandos são colocados, isso fará com que o primeiro LED (vermelho) acenda, então será aguardado um segundo, o primeiro LED (vermelho) apagará e o segundo LED (verde) acenderá. Será aguardado outro segundo, o segundo LED (verde) apagará e o terceiro LED (azul) acenderá, então mais um segundo, o terceiro LED (azul) apagará e o loop estará completo, voltando a acender novamente o primeiro LED (vermelho). Teremos assim um sequencial.

Continuando na mesma linha de experiências, vamos acionar um LED RGB.

8.5 EXPERIÊNCIA Nº 05 - CONTROLANDO UM

LED RGB

Para controlar um LED RGB, não há praticamente nada diferente do que controlar três LEDs independentes cada um com uma cor, no caso, vermelho, verde e azul. As principais diferenças são que, como no LED RGB, as cores estão montadas no mesmo invólucro, tornando possível acendê-las em diferentes combinações. E com a mistura das luzes geradas, são possíveis várias tonalidades de cores. Basta lembrar das aulas da escola sobre as cores primárias e como é possível criar outras cores a partir delas.

Para essa experiência, também usamos três pinos PWM (*Pulse-with Modulation*) para alterar a intensidade de cada cor do LED. Já sobre os pinos PWM, veremos mais detalhes adiante.

O que é necessário

1 x Arduino UNO

1 x LED RGB

3 x Resistores 220Ω

Esquema de montagem

Coloque o LED RGB na placa de ensaios conforme mostra a próxima figura. Usando um cabinho, ligue o terminal negativo do LED (terminal mais longo) a um dos pinos GND do Arduino.

Em seguida, ligue o terminal vermelho do LED (veja na figura) a um resistor de pelo menos 220Ω, e ligue o outro terminal do resistor, usando um cabinho, ao pino digital 6 do Arduino.

Agora, ligue o terminal verde do LED (veja na figura) a um resistor de pelo menos 220Ω e ligue o outro terminal do resistor, usando cabinho, ao pino digital 5 do Arduino. Finalmente, ligue o

terminal azul do LED (veja na figura) a um resistor de pelo menos 220Ω, e ligue o outro terminal do resistor, usando cabinho, ao pino digital 3 do Arduino.

Caso não tenha certeza de quais são as cores dos terminais no LED, não se preocupe, depois de colocá-lo para funcionar será fácil identificá-los e ajustar o que for necessário.

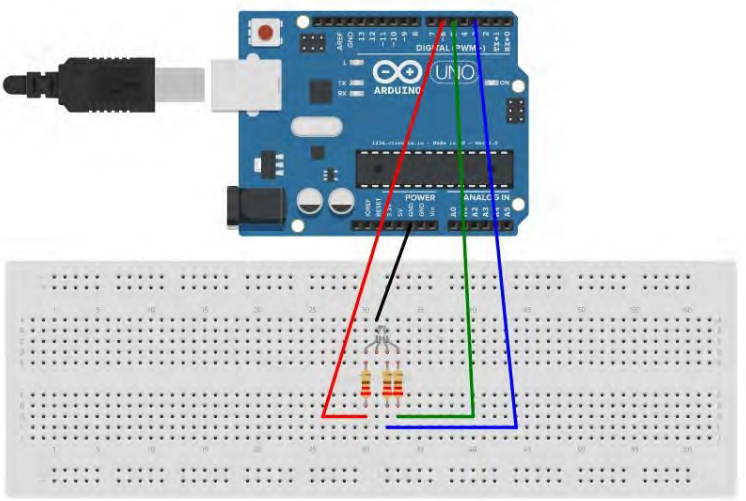


Figura 8.5: Esquema de montagem da experiência nº 05

Programação

```
void setup() {
    pinMode(3,OUTPUT);
    pinMode(5,OUTPUT);
    pinMode(6,OUTPUT);
}

void loop() {
    analogWrite(3,255);
    analogWrite(5,255);
    analogWrite(6,255);
}
```

O que foi feito

Na função `setup()`, ajustamos os pinos 3, 5 e 6 para saída usando os comandos `pinMode(3,OUTPUT)`, `pinMode(5,OUTPUT)` e `pinMode(6,OUTPUT)`. Assim poderemos colocar ou não tensão neles.

Já na função `loop()` usamos o comando `analogWrite` para colocarmos nos pinos digitais 3, 5 e 6 um valor decimal para saída 255. Ele pode variar de 0 (intensidade mínima da cor) até 255 (intensidade máxima da cor).

Escolhi deixar fixo em 255 para mostrar todas as cores acesas na intensidade máxima, para que você possa compreender primeiro como tudo isso funciona.

Resultado esperado

Você pode gerar as cores primárias do LED (vermelho, verde e azul) colocando 255 no respectivo pino e 0 nos demais. Para gerar qualquer outra cor, basta mudar a intensidade entre os pinos.

8.6 EXERCITE

Projete e construa um conjunto de semáforos em um cruzamento. Você terá de controlar dois semáforos de veículos com 3 luzes (vermelho, amarelo e verde) e mais dois de pedestres com 2 luzes (vermelho e verde).

Faça tudo sincronizado, inclusive dando tempo suficiente para o pedestre atravessar enquanto dos dois semáforos de veículos permanecem vermelho.

8.7 A SEGUIR...

Após executar as cinco experiências propostas neste capítulo,

você já deve estar ansioso por expandir seu horizonte e quer ir além de controlar LEDs. Mas saiba que essa prática é superimportante e vale a pena parar um pouco nela até que todos os conceitos estejam bem assimilados.

A seguir, você terá um detalhamento de todas as funções para os pinos digitais do Arduino.

Capítulo 9

AS FUNÇÕES PARA PINOS DIGITAIS

Uma das interfaces do Arduino são os pinos digitais, que são capazes de escrever (enviar) uma tensão alta (`HIGH`) de 5V ou manter a ausência de tensão (`LOW`) com 0V. Por essa variação, como já pode ser visto nas experiências anteriores, é possível controlar equipamentos e, como veremos adiante, receber valores de sensores.

Na sequência, estão todas as funções para a manipulação dos pinos digitais do Arduino.

pinMode()

Configura um determinado pino para comportar-se como entrada (`INPUT`) ou saída (`OUTPUT`).

Veja um exemplo de sintaxe:

```
pinMode(pino,modo de operação);
```

Veja exemplos de uso:

```
pinMode(13,OUTPUT);

pinMode(13,INPUT);
```

digitalWrite()

Se o pino for configurado para saída (`OUTPUT`), o comando `digitalWrite` poderá ajustar a voltagem do pino para 5V (`HIGH`) ou 0V (`LOW`).

Veja um exemplo de sintaxe:

```
digitalWrite(pino,valor);
```

Veja exemplos de uso:

```
digitalWrite(13,HIGH);

digitalWrite(13,LOW);
```

digitalRead()

Lê o valor atual de um pino especificado, que pode ser `LOW` ou `HIGH` .

Sintaxe:

```
digitalRead(pino);
```

Exemplos:

```
int valor = digitalRead(4);
```

OK, aproveitei para incluir todas as funções para pinos digitais do Arduino. A `digitalWrite` já vimos na prática. Agora, para a `digitalRead` , dê uma olhada na experiência seguinte.

9.1 EXPERIÊNCIA Nº 06 - USANDO UM BOTÃO

Um dos dispositivos analógicos mais usados para interagir com o Arduino são os botões, ou chaves. Botões são utilizados para permitir e/ou interromper a passagem de sinal elétrico temporariamente.

O que é necessário

1 x Arduino UNO

1 x Botão

1 x Resistor de 10KΩ

Esquema de montagem

Guiando-se pela figura a seguir, ligue um resistor de 10KΩ entre o pino GND e o pino digital 2 do Arduino, usando dois cabinhos. Também usando cabinhos, ligue o botão entre o pino digital 2 e o pino 5V do Arduino.

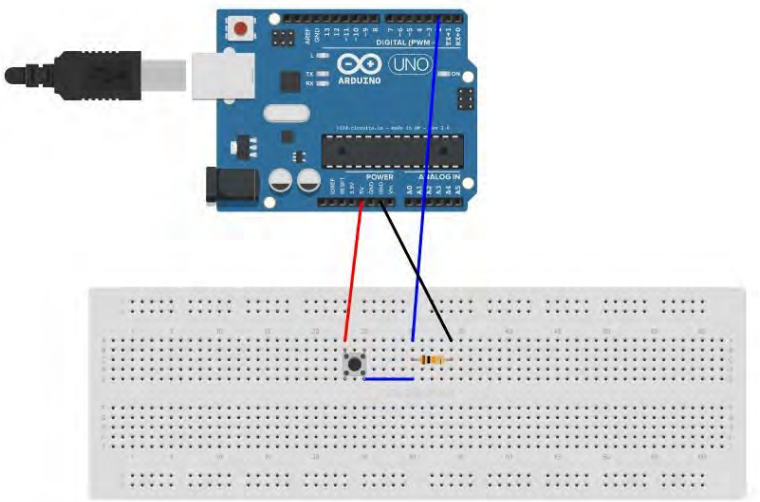


Figura 9.1: Esquema de montagem da experiência nº 06

Programação

```
int led = 13;
int botao = 2;
int press = 0;

void setup() {
    pinMode(led,OUTPUT);
    pinMode(botao,INPUT);
}
```

```
void loop() {
    press = digitalRead(botao);
    if(press == HIGH) {
        digitalWrite(led,HIGH);
    } else {
        digitalWrite(led,LOW);
    }
}
```

O que foi feito

Declaramos duas variáveis do tipo inteiro para indicar em qual pino digital está ligado o LED - que será aceso ou apagado pela atuação do botão (`int led = 13`) - e para indicar em qual pino digital está conectado o botão (`int botao = 2`). Uma terceira variável guardará o estado do pressionamento do botão (`int press = 0`).

Na função `setup()` , ajustamos o pino digital do LED para saída usando `pinMode(led,OUTPUT)` , e o pino digital do botão para entrada usando `pinMode(botao,INPUT)` . Agora, na função `loop()` , lemos o estado do pino digital do botão e armazenamos o que for lido na variável `press` , destinada a isso, usando `press = digitalRead(botao)` .

Então, basta verificar se o estado do botão for `HIGH` (ou seja, está pressionado), acendemos o LED; caso contrário (o estado do botão for `LOW`), apagamos o LED. Tudo isso é feito usando `if(press == HIGH)` para ver se o botão está pressionado, `digitalWrite(led,HIGH)` para acender o LED, e `digitalWrite(led,LOW)` para apagar o LED.

Resultado esperado

Depois de compilar e carregar o programa no Arduino, se você pressionar o botão, o LED do pino 13 acenderá. E quando você

soltar o botão, ele deve apagar.

9.2 EXERCITE

Faça com que um LED pisque uma vez a cada segundo e em seguida adicione um botão ao projeto. Quando você apertar o botão, o LED deve parar de piscar. Apertando novamente, ele deve voltar a piscar.

9.3 A SEGUIR...

Com a linguagem C básica para o Arduino já na cabeça e com domínio sobre os pinos digitais, chegou a hora de aprofundarmos mais na linguagem. A seguir, estarão todos os tipos de dados possíveis na linguagem, além de detalhes sobre variáveis e conversores entre os tipos de dados.

Capítulo 10

TIPOS DE DADOS, VARIÁVEIS E CONVERSORES

Como você viu nas experiências anteriores, onde usamos variáveis, elas precisam ser declaradas, e o modo correto de se fazer isso é indicar o tipo de dados que elas vão armazenar, seguido no nome pelo qual serão identificadas.

Saber declará-las corretamente é importantíssimo, pois temos disponíveis quantidades muito limitadas de memórias para utilizar no Arduino. Por isso, na sequência, temos todos os tipos de dados e a quantidade de memória que cada variável de cada tipo de dado ocupa, além dos conversores para que seja possível intercambiar dados entre variáveis de tipos diferentes.

Além disso, é preciso dar nomes óbvios para suas variáveis, por exemplo, para uma variável que guardará o resultado de uma soma, o nome mais óbvio é mesmo `soma`.

Declarações de funções também podem ser precedidas por um tipo de dado, e isso indicará qual é o retorno que essas funções proverão. Por exemplo, `int soma()` indica que a função `soma` retorna um número inteiro depois de, provavelmente, realizar uma soma. Para os nomes de funções, use a mesma lógica que para os nomes de variáveis.

Os tipos de dados possíveis para variáveis são:

- `void`

  Apenas usado em declaração de funções, indica que a função não retornará valores, como por exemplo, nas funções `setup()` e `loop()`. Você verá que será utilizado em todas as experiências.

- `boolean`

  Pode armazenar apenas dois valores, sendo `true` (verdadeiro) ou `false` (falso). Ocupa 1 byte de memória.

- `char`

  Pode armazenar apenas um caractere. Pode receber um caractere entre aspas simples ('), ou um número que indique um caractere na tabela ASCII (*American Standard Code For Information Interchange* ou Tabela Americana para Intercâmbio de Informação).

  No caso de usar um número para indicar um caractere ASCII, você precisa ter em mente que os caracteres não são imprimíveis. Ou seja, do 0 ao 31, que são caracteres de controle, não aparecerão na tela do Monitor Serial. Ocupa 1 byte de memória.

- `byte`

  Pode armazenar um número entre 0 e 255. Também pode receber um número binário precedido pelo formatador de binário, como:

```
byte x = B1100; // 1100 em binário é igual ao número
12 em decimal
```

Ocupa 1 byte de memória.

- `int`

  É o tipo primário para armazenamento de números. Pode armazenar números inteiros entre -32.768 e 32.676.

  Pode ser usado com o modificador `unsigned int` para armazenar apenas números inteiros positivos entre 0 e 4.294.967.295. Ocupa 2 bytes de memória no Arduino UNO, e 4 bytes no Arduino Due.

- `word`

  Pode armazenar apenas números inteiros positivos. Ocupa 2 bytes de memória no Arduino.

- `long`

  Pode armazenar números inteiros entre -2.147.483.648 e 2.147.483.647. Se utilizado com o modificador `unsigned` , vai armazenar apenas números inteiros positivos entre 0 e 4.294.967.295. Ocupa 4 bytes de memória.

- `short`

  Pode armazenar números inteiros entre -32.768 e 32.767. Ocupa 2 bytes de memória.

- `float`

  Pode armazenar números fracionários entre -3,4028235 x 10+38 e 3,4028235 x 10+38. Tem precisão de apenas 6 ou 7 dígitos depois da vírgula. Ocupa 4 bytes de memória.

- `double`

  Pode armazenar exatamente o dobro de uma variável `float` . Ocupa 4 bytes de memória no Arduino Uno, e 8 bytes no Arduino Due.

- `string`

  Pode armazenar vários caracteres até o total disponível de memória. Deve receber o conjunto de caracteres entre aspas ("), como por exemplo, `"Arduino"` - considerado uma string. Ocupa a quantidade de caracteres em bytes de memória.

- `array`

  Tipo conhecido por vetor unidimensional (vetor) ou multidimensional (matriz). Pode ser de qualquer um dos tipos de dados anteriores, menos `void` .

```
int vetor[8];        // indica um vetor de 8 posiçõe

int matriz[3][3];    // indica uma matriz de 3 linha
s e 3 colunas
```

10.1 VARIÁVEIS

Escopo de variável

As variáveis possuem uma propriedade que chamamos de escopo, que é algo como a validade delas dentro de um programa. Basicamente, você pode considerar que o escopo de uma variável é o bloco de código que estiver entre chaves. Caso alguma variável seja declarada fora de qualquer chave, antes mesmo da declaração da função `setup()` ou `main()` , ela é considerada global, ou seja, valerá para todo o programa.

Veja um exemplo:

```
int x = 0;    // esta será uma variável global

void setup() {
    int y = 0;    // esta é uma variável local à função setup(),
                  // não poderá
                  // ser usada fora dela
}

void loop() {
    int q = 0;    // esta é uma variável local à função loop(),
                  // não poderá
                  // ser usada fora dela
    for(int i=0;i<10;i++) {        // a variável i é local para a
                                   // estrutura for, pois foi
                                   // declarada dentro dela
        // aqui vão comandos
        // a serem repetidos pela
        // estrutura for
    }
}
```

- `static`

  O qualificador `static` indica que a variável será apenas visível e modificável na função em que for declarada.

- `volatile`

  O qualificador `volatile` indica que o valor da variável poderá ser alterado em qualquer ponto do programa. É uma diretiva para o compilador armazenar o conteúdo da variável na memória RAM, e não nos registradores de armazenamento.

- `const`

  Esse qualificador modifica o comportamento da variável tornando-a apenas para leitura, ou seja, torna o identificador como constante. Uma constante tem seu

valor definido na declaração e não pode ser alterado em nenhum momento dentro do programa.

O display de 7-segmentos são basicamente 7 LEDs que, quando acionados na sequência correta, apresentam o desenho de números. É possível criar várias outras formas neles, basta usar a imaginação. Vamos entender o display de 7-segmentos, e depois faremos outra experiência para utilizarmos alguns tipos de dados apresentados anteriormente.

10.2 EXPERIÊNCIA Nº 07 - USANDO UM DISPLAY DE 7-SEGMENTOS

Display de 7 segmentos é um mostrador comumente utilizado para mostrar números. Normalmente, são usados em conjunto com um conversor Binário/BCD (*Binary Coded Decimal*). São baratos e construídos por 7 LEDs que formam seus segmentos e o ponto decimal (DP).

Display de 7 segmentos podem ser de ânodo comum, que possui um pino que é o polo negativo comum a todos os segmentos que devem ser conectados ao polo positivo para serem acesos, ou cátodo comum, que possui um pino que é o polo positivo comum a todos os segmentos que devem ser conectados ao polo negativo para serem acesos.

Os segmentos são identificados por letras de A até G, mais o DP (ponto decimal). Para formar números e algumas letras, basta acender ou apagar os segmentos:

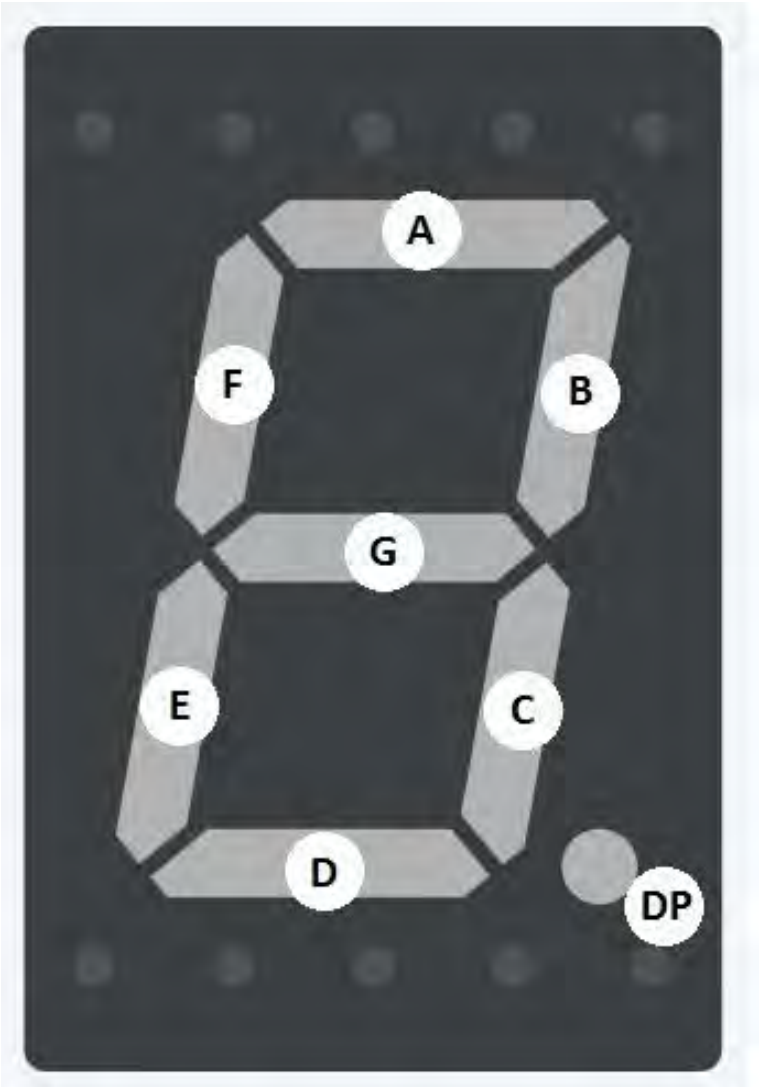


Figura 10.1: Um display de 7-segmentos com seus segmentos identificados

Para trabalhar com displays de 7-segmentos, usamos os mesmos conhecimentos que já temos em trabalhar com LEDs.

O que é necessário

1 x Arduino

1 x Display de 7-segmentos cátodo comum

Esquema de montagem

Orientando-se pela imagem a seguir e usando um cabinho, ligue os pinos comuns (5V) a um resistor de 220Ω e ao pino 5V do Arduino. Agora, também usando um cabinho, ligue o pino do segmento A ao pino digital 2 do Arduino.

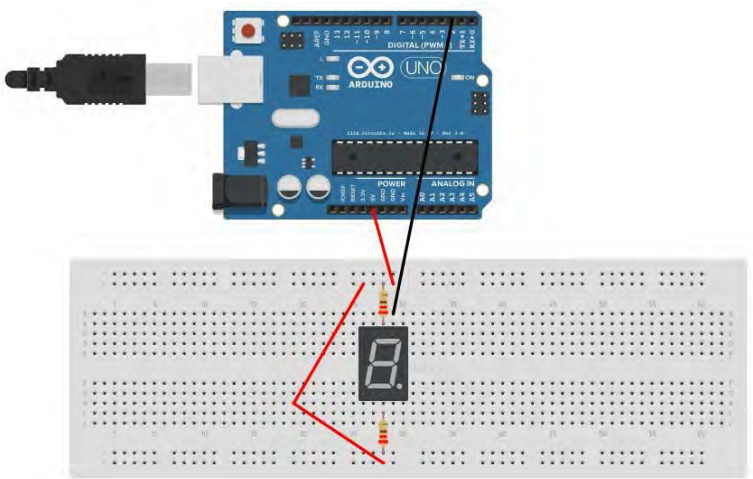


Figura 10.2: Esquema de montagem da experiência nº 07

Programação

```
void setup() {
    pinMode(2,OUTPUT);
}

void loop() {
    digitalWrite(2,HIGH);
    delay(1000);
    digitalWrite(2,LOW);
    delay(1000);
}
```

O que foi feito

Você deve ter percebido que usamos uma programação idêntica para acender e apagar um LED, e o segredo é esse mesmo. Para criar números nos displays de 7-segmentos, basta acender e apagar os segmentos corretos na hora certa.

Agora que você entendeu a "mágica", na próxima experiência vamos mostrar números no display.

Resultado esperado

O segmento A do display ficará aceso por um segundo e apagado por mais um segundo, piscando como fizemos com o LED nas primeiras experiências. Sabendo todo o funcionamento, espera-se que você consiga ligar todos os outros segmentos para formar números.

Espaço para anotações

Para exemplificar o uso dos tipos de dados, especificamente de vetores e matrizes, execute a experiência a seguir.

10.3 EXPERIÊNCIA Nº 08 - MOSTRANDO NÚMEROS DE 0 A 9 EM UM DISPLAY DE 7-SEGMENTOS

Na experiência anterior, vimos como é simples trabalhar com um display de 7-segmentos. Agora, vamos à montagem e a um código completo para um contador de zero a nove.

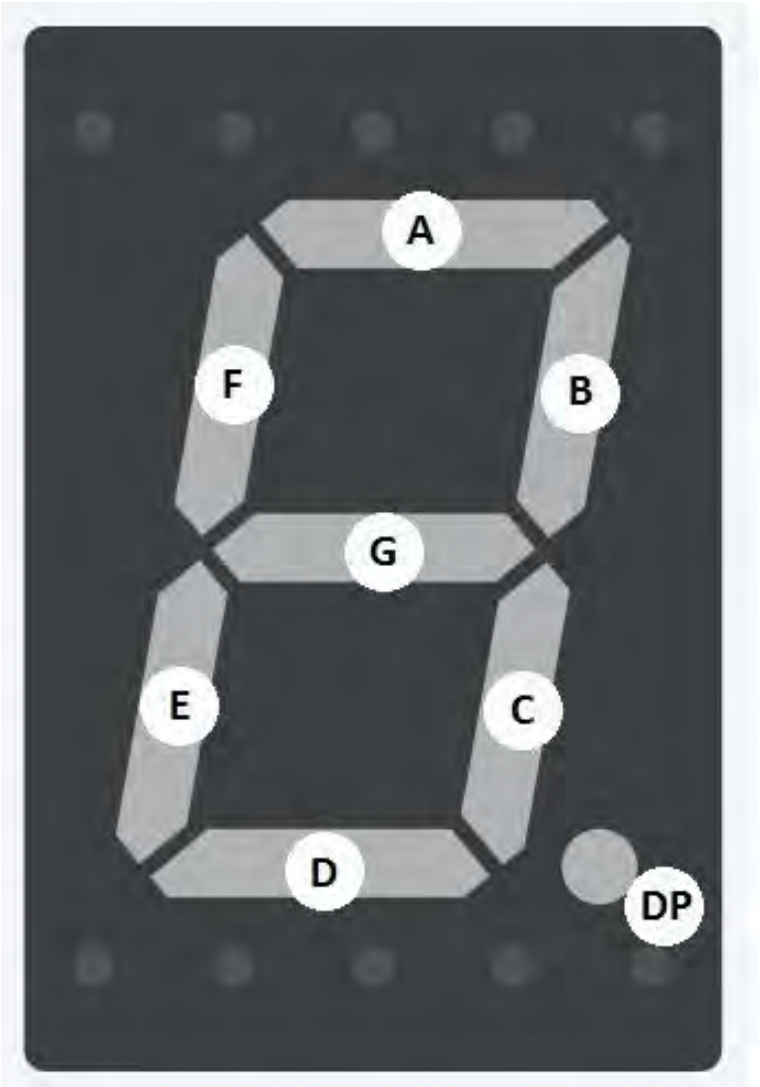


Figura 10.3: Um display de 7-segmentos com seus segmentos identificados

O que é necessário

1 x Arduino

1 x Display de 7-segmentos cátodo comum

Esquema de montagem

O primeiro passo é ligar os pinos comuns (5V) a um resistor de 220Ω e ao pino 5V do Arduino. Depois, basta ligar os pinos dos segmentos aos pinos digitais do Arduino. Vamos usar o seguinte esquema:

- O segmento A ao pino digital 3 do Arduino;
- O segmento B ao pino digital 4 do Arduino;
- O segmento C ao pino digital 5 do Arduino;
- O segmento D ao pino digital 6 do Arduino;
- O segmento E ao pino digital 7 do Arduino;
- O segmento F ao pino digital 8 do Arduino;
- O segmento G ao pino digital 9 do Arduino.

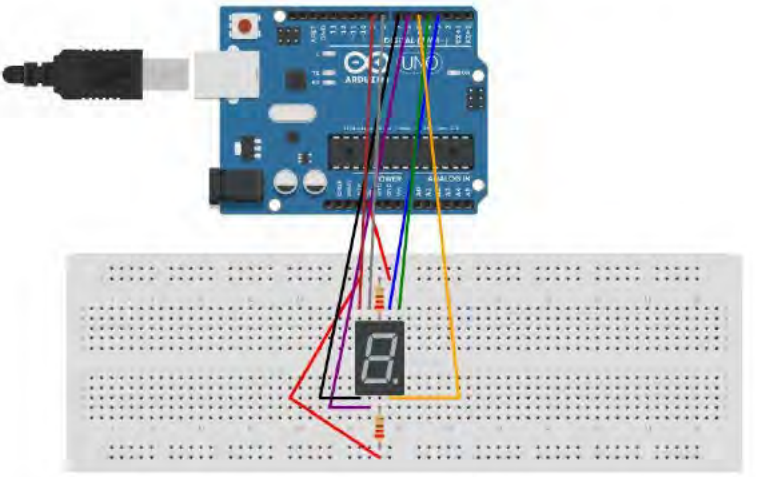


Figura 10.4: Esquema de montagem da experiência nº 08

Programação

```
int a = 3, b = 4, c = 5, d = 6, e = 7, f = 8, g = 9;

int num[10][7] = {
        {a,b,c,d,e,f},     // Zero
        {b,c},             // Um
        {a,b,e,d,g},       // Dois
        {a,b,c,d,g},       // Três
        {b,c,f,g},         // Quatro
        {a,c,d,f,g},       // Cinco
        {a,c,d,e,f,g},     // Seis
        {a,b,c},           // Sete
        {a,b,c,d,e,f,g},   // Oito
        {a,b,c,f,g}        // Nove
     };

void setup() {
    pinMode(a,OUTPUT);
    pinMode(b,OUTPUT);
    pinMode(c,OUTPUT);
    pinMode(d,OUTPUT);
    pinMode(e,OUTPUT);
    pinMode(f,OUTPUT);
    pinMode(g,OUTPUT);
}

void loop() {
    for(int i=0;i<10;i++) {
        apaga();
        numero(i);
        delay(1000);
    }
}

void apaga() {
    digitalWrite(a,HIGH);
     digitalWrite(b,HIGH);
     digitalWrite(c,HIGH);
     digitalWrite(d,HIGH);
     digitalWrite(e,HIGH);
     digitalWrite(f,HIGH);
     digitalWrite(g,HIGH);
}

void numero(int n) {
  for(int i=0;i<7;i++) digitalWrite(num[n][i],LOW);
}
```

O que foi feito

As variáveis `int a = 3, b = 4, c = 5, d = 6, e = 7, f = 8, g = 9` indicarão quais pinos digitais estão ligados a quais segmentos do display. Em seguida, declaramos uma matriz com 10 linhas e 7 colunas. Cada coluna será a sequência de segmentos a ser acesa para `{b,c}` criar um número do display. Cada linha será um número na ordem de zero até nove, usando a seguinte declaração:

```
int num[10][7] = {
        {a,b,c,d,e,f},    // Zero
        {b,c},            // Um
        {a,b,e,d,g},      // Dois
        {a,b,c,d,g},      // Três
        {b,c,f,g},        // Quatro
        {a,c,d,f,g},      // Cinco
        {a,c,d,e,f,g},    // Seis
        {a,b,c},          // Sete
        {a,b,c,d,e,f,g},  // Oito
        {a,b,c,f,g}       // Nove
    };
```

Na função `setup()`, ajustamos os pinos digitais como saída usando as variáveis nas quais estão identificados. Na função `loop()`, usando a estrutura de repetição `for(int i=0;i<10;i++)`, contamos de 0 até 9 usando a variável `i` que indicará qual a linha da matriz que será lida.

Chamamos em seguida a função `apaga()`, que apaga todos os segmentos do display, e depois a função `numero(i)`, passando como parâmetro a variável `i` e indicando o número que deve ser criado no display.

Aguardamos um segundo para que seja possível visualizar o número no display usando o comando `delay(1000)`. Em seguida, declaramos a função `apaga()`, que não terá retorno (`void`), e que coloca o valor `HIGH` em cada pino digital correspondente a cada segmento, fazendo com que o LED desse segmento seja apagado

A função `numero()` também não possui retorno (`void`), mas recebe como parâmetro um número inteiro, que deve ser entre 0 e 9

e que será mostrado no display de 7-segmentos. Para isso, ela usa a estrutura de repetição `for(int i=0;i<7;i++)` para indicar a coluna da matriz que deve ser lida. Usando o conjunto entre linha (recebida por parâmetro na variável `n`) e coluna (variável `i` contada pela estrutura de repetição `for`), coloca no pino digital correspondente o valor `LOW`, acendendo o segmento desejado. A função `numero()` também não possui retorno (`void`), mas recebe como parâmetro um número inteiro, que deve ser entre 0 e 9 e que será mostrado no display de 7-segmentos. Para isso, ela usa a estrutura de repetição `for(int i=0;i<7;i++)` para indicar a coluna da matriz que deve ser lida. Usando o conjunto entre linha (recebida por parâmetro na variável `n`) e coluna (variável `i` contada pela estrutura de repetição `for`), coloca no pino digital correspondente o valor `LOW`, acendendo o segmento desejado.

Resultado esperado

Depois de compilar e enviar seu programa ao Arduino, o display de 7-segmentos começará a acender seus segmentos criando números de zero até nove.

10.4 CONVERSORES DE DADOS

- `char()`

  Converte um valor para o tipo de dado `char`. Veja um exemplo de uso:

  ```
  char teste;
  int numero = 63;
  teste = char(numero);
  ```

- `byte()`

  Converte um valor para o tipo de dado `byte`. Veja um exemplo de uso:

```
byte teste;
int numero = 51;
teste = byte(numero);
```

- `int()`

  Converte um valor para o tipo de dado `int` .

```
int teste;
char letra = '1';
teste = int(letra);
```

- `word()`

  Converte um valor para o tipo de dados `word` .

```
long teste;
char letra = '1';
teste = word(letra);
```

- `long()`

  Converte um valor para o tipo de dado `long` .

```
long teste;
int numero = 10;
teste = long(numero);
```

- `float()`

  Converte um valor para o tipo de dado `float` .

```
float teste;
int numero = 10;
teste = float(numero);
```

10.5 EXERCITE

Adicione um botão ao circuito da experiência que mostra números de zero a dez no display de 7-segmentos. Quando você apertar o botão, a contagem deve zerar. Lembre-se de que a contagem deve ir apenas até nove e depois voltar até zero

novamente.

10.6 A SEGUIR...

Depois dos tipos de dados possíveis para a linguagem C para Arduino, a seguir você encontrará as estruturas de desvio de fluxo e repetição de execução, além de todos os operadores comparativos, matemáticos, lógicos e de bits disponíveis para a linguagem.

Capítulo 11

OPERADORES E ESTRUTURAS DE SELEÇÃO E REPETIÇÃO

Os operadores podem ser de comparação, matemáticos e lógicos. Eles são componentes centrais de qualquer linguagem de programação e precisam ser bem compreendidos para que sejam bem utilizados.

As estruturas de seleção e repetição vão definir como será o funcionamento de seu programa, desviando o fluxo de funcionamento ou repetindo trechos de código. Devido à importância de todos, recomendo uma olhada com bastante atenção.

11.1 OPERADORES DE COMPARAÇÃO

- == - Igual a;
- != - Diferente de;
- < - Menor que;
- > - Maior que;
- <= - Menor ou igual.

- `>=` - Maior ou igual.

11.2 OPERADORES MATEMÁTICOS

- `+` - Soma. `a = 1 + 2`, `a` será 3.
- `-` - Subtração. `a = 3 - 1`, `a` será 2.
- `*` - Multiplicação. `a = 2 * 2`, `a` será 4.
- `/` - Divisão. `a = 6 / 2`, `a` será 3.
- `%` - Resto da divisão entre inteiros. `a = 3 % 2`, `a` será 1.
- `++` - Incremento em um. `a++` é o mesmo que `a = a + 1`.
- `--` - Decremento em um. `a--` é o mesmo que `a = a - 1`.
- `+=` - Adição composta. `a += 2` é o mesmo que `a = a + 2`.
- `-=` - Subtração composta. `a -= 2` é o mesmo que `a = a - 2`.
- `*=` - Multiplicação composta. `a *= 2` é o mesmo que `a = a * 2`.
- `/=` - Divisão composta. `a /= 2` é o mesmo que `a = a / 2`.
- `%=` - Divisão entre inteiros composta. `a %= 2` é o mesmo que `a = a % 2`.

11.3 OPERADORES LÓGICOS

- `&&`

  Realiza a operação lógica **E** (*AND*) entre dois valores booleanos. O resultado é verdadeiro (*true*) apenas se os dois valores forem verdadeiros. Veja um exemplo de uso:

```
boolean A = true;     // Verdadeiro
boolean B = false;     // Falso
boolean C = A && B;     // C = Falso
```

- `||`

  Realiza a operação lógica **OU** (*OR*) entre dois valores booleanos. O resultado é verdadeiro (*true*) se um dos dois valores for verdadeiro.

```
boolean A = true;     // Verdadeiro
boolean B = false;     // Falso
boolean C = A || B;     // C = Verdadeiro
```

- `!`

  Realiza a operação lógica **NÃO** (*NOT*) em um valor booleano. O resultado é verdadeiro (*true*) se o valor for falso (*false*), e falso se o valor for verdadeiro.

```C
boolean A = true;     // Verdadeiro
boolean C = !A;       // C = Falso
```

11.4 OPERADORES BIT-A-BIT

- `&`

  Realiza a operação *bit-a-bit* **E** (*AND*) entre todos os bits de dois bytes. O resultado de uma operação *AND* é 0 quando ambos os bits são 0; 0 quando um dos bits é 1 e o outro é 0; e 1 quando ambos os bits são 1. Veja um

exemplo de uso:

```
int A = 9;          // 9 decimal = 1001 binário
int B = 7;          // 7 decimal = 0111 binário
int C = A & B;      // C = 1 decimal = 0001 binário
```

- `|`

  Realiza a operação *bit-a-bit* **OU** (*OR*) entre todos os bits de dois bytes. O resultado de uma operação *OR* é 0 quando ambos os bits são 0; 1 quando um dos bits é 1 e o outro é 0; e também 1 quando ambos os bits são 1.

```
int A = 1;          // 1 decimal = 0001 binário
int B = 10;          // 10 decimal = 1010 binário
int C = A | B;      // C = 11 decimal = 1011binário
```

- `^`

  Realiza a operação *bit-a-bit* **OU-Exclusivo** (*XOR*) entre todos os bits de dois bytes. O resultado de uma operação *XOR* é 0 quando os bits são iguais, e 1 quando os bits são diferentes.

```
int A = 5;          // 5 decimal = 0101 binário
int B = 7;          // 7 decimal = 0111 binário
int C = A ^ B;      // C = 1 decimal = 0010 binário
```

11.5 ESTRUTURAS DE SELEÇÃO

`if ... else`

Usado em conjunto com os operadores de comparação para realizar um desvio no fluxo de funcionamento do programa. Se a condição especificada for verdadeira, o bloco de comandos será executado; caso contrário, o bloco de comandos após a palavra `else` será executado. O uso de `else` , ou seja, do segundo bloco de comandos é opcional.

Veja um exemplo de sua sintaxe:

```
if(condição) {
    // bloco de comandos
} else {
    // bloco de comandos
}
```

Veja um exemplo de uso:

```
if(a > 10) {
    // este bloco de comandos será executado se o valor
    // da variável a for maior que 10
}

if(a != 10) {
    // este bloco de comandos será executado se o valor
    // da variável a for diferente que 10
} else {
    // este bloco de comandos será executado se o valor
    // da variável a NÃO for diferente que 10
}
```

`switch ... case`

Utilizado para seleção de uma variável e comparação com um valor ou um conjunto de valores. Executa o bloco de comandos até encontrar o comando `break` que para a execução. Possui a diretiva `default` para caso a variável não coincida com nenhum dos valores especificados nos casos.

Veja um exemplo de sua sintaxe:

```
switch(variável) {
    case condição:
        // bloco de comandos
        break;
}
```

Veja um exemplo de uso:

```
switch(valor) {
    case 1:
        // executa este bloco de comandos caso valor
        // seja igual a 1
```

```
        break;
    case 2:
        // executa este bloco de comandos caso valor
        // seja igual a 2
        break;
    case 3..9:
        // executa este bloco de comandos caso valor
        // seja maior ou igual a 3 ou menor ou igual
        // a 9
        break;
```

11.6 ESTRUTURAS DE REPETIÇÃO

for

Faz com que um bloco de comandos seja repetido conforme a condição e o incremento especificado. A estrutura de repetição `for` consiste em três parâmetros: a inicialização da variável de controle, uma condição e um incremento:

Veja um exemplo de sua sintaxe:

```
for(inicialização;condição;incremento) {
    // bloco de comandos
}
```

Veja um exemplo de uso:

```
for(int x = 1; x<=10; x++) {
    // executará este bloco de comandos 10 vezes
}

for(int x = 2; x <= 100; x += 2) {
    // executará este bloco de comando 50 vezes,
    //começando em 2, contando até 100 com incremento de 2
}

for(int x = 10; x >= 1; x--) {
    // executará este bloco de comando 10 vezes
    // contando regressivamente
}
```

while

Estrutura de repetição com teste no início. Ela é diferente do `for`, que controla a variável contadora com o `while`, pois você terá de controlá-la. Repete um bloco de comandos enquanto a condição especificada seja verdadeira. Caso a variável de controle seja inicializada com um valor que torne a condição falsa, o laço de repetição não acontecerá nenhuma vez.

Veja um exemplo de sintaxe:

```
while(condição) {
    // bloco de comandos
}
```

Veja um exemplo de uso:

```
int i = 1;
while(i <= 10) {
    // bloco de comandos
    i++;   // incrementa a variável - irá contar de 1 até 10
}

int i = 100;
while(i > 1) {
    // bloco de comandos
    i--;   // decrementa a variável - irá contar de 100 até 1
}
```

`do while`

Estrutura de repetição com teste no final. Repete um bloco de comandos enquanto a condição especificada seja verdadeira. A grande diferença entre o `do ... while` e o `while` é que, caso a variável de controle seja inicializada com um valor que torne a condição falsa, o laço de repetição acontecerá pelo menos uma vez.

Sintaxe:

```
do {
    // bloco de comandos
} while(condição);
```

Exemplo:

```
int i = 1;
do {
    // bloco de comandos
    i++;     // incrementa a variável - irá contar de 1 até 10
} while (i <= 10);

int i = 100;
do {
    // bloco de comandos
    i--;     // decrementa a variável - irá contar de 100 até 1
} while(i > 1);
```

break

Interrompe a execução de qualquer estrutura de repetição e também no caso da estrutura de seleção `switch...case` .

Sintaxe:

```
break;
```

Exemplo:

```
for(int x = 1; x<=10; x++) {
    // bloco de comandos
    if(x == 5) {
        break;        // interrompe a execução
    }
}

int i = 100;
while(i > 1) {
    // bloco de comandos
    i--;
    if(i == 3) {
        break;        // interrompe a execução
    }
}
```

continue

Faz com que a estrutura de repetição volte ao primeiro comando a partir do ponto onde está o comando `continue` .

Sintaxe

```
continue;
```

Exemplo:

```
for(int x = 1; x<=10; x++) {
    // bloco de comandos
    if(x == 5) {
        continue;
    }
    // bloco não executado no momento que o continue
    // acontecer
}

int i = 1;
while(i < 100) {
    // bloco de comandos
    i++;
    if(i == 7) {
        continue;
    }
    // bloco não executado no momento que o continue
    // acontecer
}
```

Para exemplificar as estruturas apresentadas até agora podemos executar mais algumas experiências.

11.7 EXPERIÊNCIA Nº 09 - OS PINOS PWM (PULSE-WITH MODULATION)

PWM significa *Pulse-with Modulation* (Pulso com modulação). Em um pino comum sem PWM, são aceitos apenas níveis alto e baixo (`HIGH` e `LOW`), ou seja, ligado (5V) ou desligado (0V). Com um pino com PWM, é possível realizar uma modulação de sinal, ou seja, alternar entre os modos `HIGH` e `LOW` rapidamente, causando como resultado um efeito de variação de tensão através da intermitência do sinal.

A frequência no Arduino UNO é de aproximadamente 490 Hz por pino PWM. Um pino PWM pode ser usado para, por exemplo, variar o brilho de um LED ou a velocidade de um motor. Mas note

que ligar um motor diretamente em um pino PWM pode causar danos ao Arduino. Será necessária uma interface para isso.

O que é necessário

1 x Arduino

1 x LED

1 x Resistor de 220Ω

Esquema de montagem

Usando um cabinho, ligue o terminal positivo do LED (o mais curto) ao pino digital 3 do Arduino. Ligue o terminal negativo do LED a um resistor de 220Ω no mínimo e, em seguida, usando um cabinho, ligue o outro terminal do resistor ao GND do Arduino.

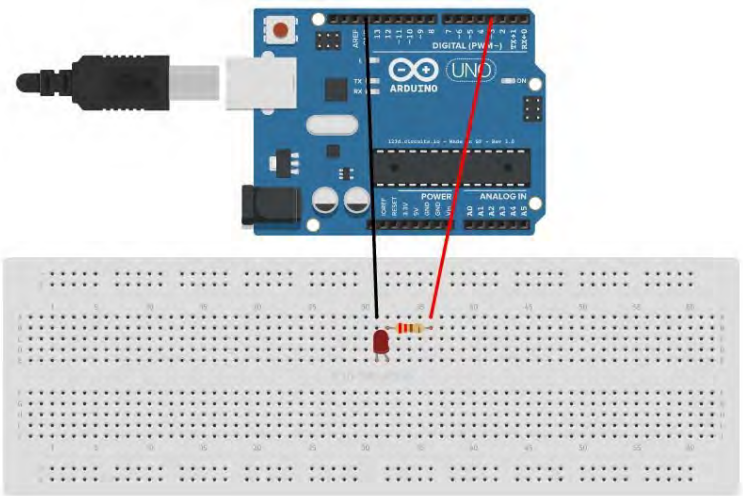


Figura 11.1: Esquema de montagem da experiência nº 09

Programação

```
int led = 3;
```

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

void loop() {
    int i;
    for(i=0;i<255;i++) {
        analogWrite(led,i);
        delay(50);
    }
}
```

O que foi feito

Como prevemos que o LED pode ser colocado em qualquer pino digital, apesar de eu sempre sugerir um, começamos com `int led = 3` , que é a variável com o valor do pino digital onde estará o LED.

Na função `setup()` , ajustamos o pino do LED para saída com `pinMode(led,OUTPUT)` . Já na função `loop()` , usamos `int i` para declarar uma variável contadora para o laço de repetição `for` .

Usamos `for(i=0;i<255;i++)` como laço de repetição que contará todos os possíveis valores para um pino PWM, ou seja, de zero até 255, onde 0 equivale ao estado LOW e 255 ao estado HIGH.

Como se trata de um pino PWM, a escrita para ele é analógica. Portanto, usamos `analogWrite(led,i)` passando para o pino o valor da variável contadora. Usamos `delay(50)` para uma pequena pausa de 50 milissegundos apenas para conseguirmos distinguir o efeito de acender e apagar do LED.

Resultado esperado

O LED deve acender vagarosamente, variando a luminescência de apagado até totalmente aceso. Note que o LED não possui muita precisão e, em determinado momento, ficará totalmente aceso,

mesmo antes de o laço de repetição terminar. Ao terminar o laço de repetição, o LED apagará e começará a acender novamente.

Depois de tantas luzes, vamos para um pouco de som nas duas experiências seguidas, mas seguindo a mesma linha de estruturas da linguagem apresentadas anteriormente e o uso de pinos PWM.

11.8 EXPERIÊNCIA Nº 10 - FAZENDO BARULHO COM UM BUZZER

Um buzzer é um componente eletrônico composto de uma membrana vibratória em um invólucro, muitas vezes plástico. Ao excitá-lo eletricamente, a membrana vibra e produz ruído. Alternando a frequência de excitação, é possível conseguir alguns tons.

O buzzer, quando ligado, produz ruído em 1kHz (quilo hertz), um bip mais ou menos equivalente à letra **i**. Esse componente é muito utilizado em alarmes de rádios-relógios, bips de centrais de alarme ou painéis numéricos para atendimento ao público.

Como basta alimentá-lo para que o ruído seja produzido, veja que o código é a mesma coisa que usar um simples LED.

O que é necessário

1 x Arduino

1 x Buzzer 5V

Esquema de montagem

Usando um cabinho, ligue o pino positivo (+) do buzzer no pino digital 9 (PWM) e, em seguida, usando outro cabinho, ligue o pino negativo do buzzer no pino GND do Arduino.

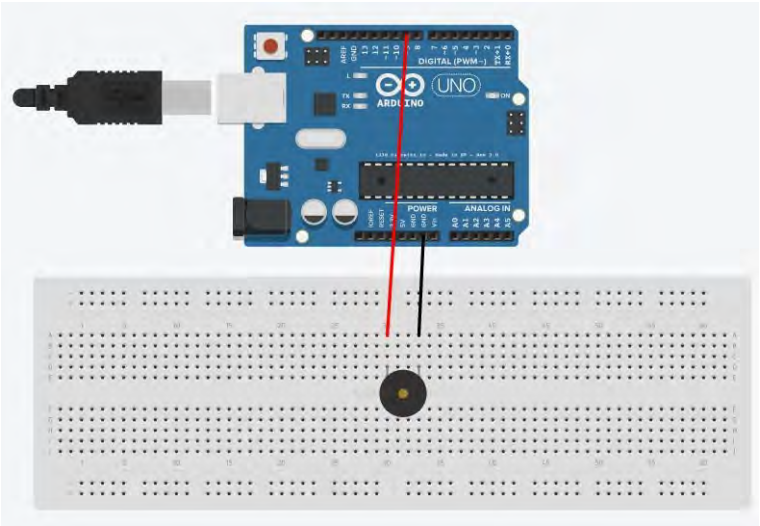


Figura 11.2: Esquema de montagem da experiência nº 10

Programação

```
void setup() {
    pinMode(9,OUTPUT);
}

void loop() {
    digitalWrite(9,HIGH);
    delay(150);
    digitalWrite(9,LOW);
    delay(3000);
}
```

O que foi feito

Na função `setup()`, usamos o já bem conhecido `pinMode(9,OUTPUT)` para ajustar o pino do buzzer como saída. Já na função `loop()` usamos `digitalWrite(9,HIGH)` para colocar o pino digital do buzzer em HIGH, e `delay(150)` para deixar o buzzer emitindo som por 150 milissegundos. Depois usamos `digitalWrite(9,LOW)` para colocar o pino digital do buzzer em LOW para que pare de emitir som, e `delay(3000)` para que o

silêncio dure 3 segundos.

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, o buzzer emitirá um bit curto de 150 milissegundos e esperará três segundos para emiti-lo novamente.

11.9 EXPERIÊNCIA Nº 11 - FAZENDO BARULHO COM UM ALTO-FALANTE

Agora que você já sabe como funciona um buzzer e os pinos PWM, deve ter imaginado em gerar tons, como notas musicais. Isso é possível, mas como o buzzer vibra apenas na faixa de 1 Hz aproximadamente, é melhor usar um alto-falante, que é capaz de reproduzir frequências mais variadas.

Como sabemos que cada nota musical possui uma frequência específica e uma duração específica, podemos usar o comando `tone` para gerar tons na frequência da nota musical, e o comando `delay` para mantê-la durante o tempo necessário.

A tabela de notas *x* frequência é:

Nota	Nome	Frequência
C	Dó	261 Hz
D	Ré	294 Hz
E	Mi	329 Hz
F	Fá	349 Hz
G	Sol	392 Hz
A	Lá	440 Hz
B	Sí	523 Hz

O que é necessário

1 x Arduino

1 x alto-falante de no máximo 4Ω

Esquema de montagem

Usando os próprios fios do alto-falente (se ele tiver), ou usando cabinhos, ligue o pino positivo do alto-falante no pino digital 3 (PWM) e, em seguida, ligue o pino negativo do alto-falante no pino GND do Arduino.

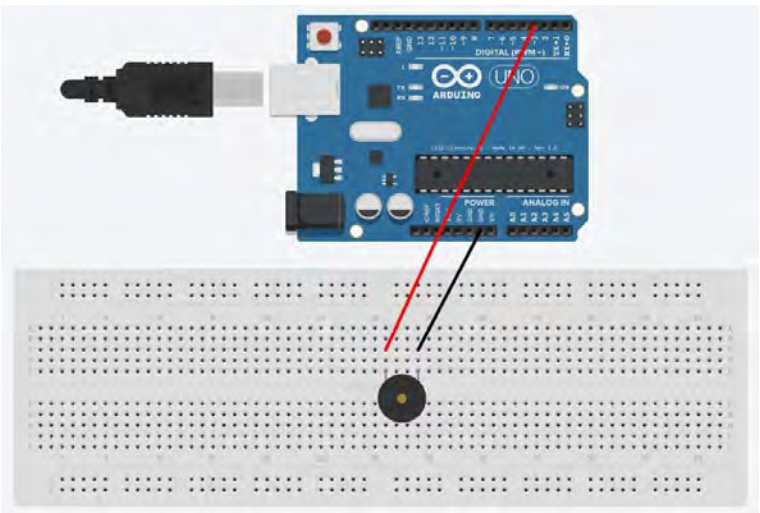


Figura 11.3: Esquema de montagem da experiência nº 11

Programação

```
void setup() {
    pinMode(3,OUTPUT);
}

void loop() {
    int C = 262, D = 294, E = 330, F = 349, G = 392, A = 440, B =
523;
    int P = 0;
```

```
    int i;
    int ode[] = {
        E,E,F,G,G,F,E,D,C,C,D,E,E,D,P,D,P,
        E,E,F,G,G,F,E,D,C,C,D,E,D,C,P,C,P,
        D,D,E,C,D,E,P,F,E,C,D,E,P,F,E,D,C,D,P,
        E,E,D,G,G,F,E,D,C,C,D,E,D,C,P,C};
    int tempo[] = {
        1,1,1,1,1,1,1,1,1,1,1,1.5,1,.5,1,1,
        1,1,1,1,1,1,1,1,1,1,1,1.5,1,.5,1,1,
        1,1,1,1,1,1,.5,1,1,1,1,1,.5,1,1.5,1,.5,1,1,1,
        1,1,1,1,1,1,1,1,1,1,1,1,1.5,.125,1,1};
    for(i=0;i<sizeof(ode)/2;i++) {
        if(ode[i] != 0) tone(3,ode[i]); else noTone(3);
        delay(tempo[i] * 500);
        noTone(3);
    }
    noTone(3);
    delay(10000);
}
```

O que foi feito

Depois de ajustar o pino digital como saída na função `setup()` na função `loop()`, declaramos 7 variáveis para conter as frequências das notas musicais (`C`, `D`, `E`, `F`, `G`, `A` e `B`), e uma que recebe zero e servirá para indicar uma pausa (`P`).

O vetor `ode` recebe a sequência de notas musicais para a música a ser tocada (Ode para Joy), e o vetor `tempo` recebe a sequência de tempo de cada nota musical, sendo que a posição na nota no vetor `ode` corresponde à posição do vetor `tempo`.

Com o `for(i=0;i<sizeof(ode)/2;i++)`, criamos um laço de repetição para ler o vetor `ode`. Como cada posição do vetor ocupa 2 bytes, para saber a quantidade de posições, basta dividir o tamanho dele por dois, ou seja, pelo tamanho de memória ocupada pelo tipo de dado `int`.

Em `if(ode[i] != 0) tone(3,ode[i]); else noTone(3);`, dizemos que se a nota é diferente de zero (pausa - variável P), toca a nota, senão para imediatamente de emitir qualquer som. Já o

comando `delay(tempo[i] * 500)` faz o tempo execução de cada nota. Depois disso, é preciso interromper a emissão de tom no pino digital 3, e fazemos isso usando `noTone(3)` .

Finalmente, com `delay(10000)` , esperamos 10 segundos antes de recomeçar a execução da música novamente.

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, uma versão para a música Ode to Joy começará a ser executada, ao término serão dados 10 segundos e a música será re-executada.

11.10 EXERCITE

Usando o buzzer e as funções corretas, tente reproduzir alguma música simples que seja fácil de reconhecer, como por exemplo, "batatinha quando nasce" ou "ciranda cirandinha". Deixe a música dentro da função `setup()` para que seja executada apenas uma vez.

11.11 A SEGUIR...

Depois exercitar bastante praticamente toda a linguagem C para o Arduino, mas usando somente os pinos digitais, chegou a hora de praticarmos também usando os pinos analógicos, que são capazes de enviar e receber valores e tensões entre 0V (`0`) e 5V (`1024`).

Capítulo 12

FUNÇÕES PARA PINOS ANALÓGICOS

Além dos pinos digitais, outra interface do Arduino são os pinos analógicos que, diferentemente dos primeiros, são capazes de escrever (enviar) ou ler (receber) uma tensão entre 0V e 5V. Ou seja, podem enviar qualquer variação entre essas duas tensões.

A tensão a ser enviada é uma proporção aproximada entre os valores decimais apresentados à interface e à tensão de referência do Arduino. Em outras palavras, se o valor decimal for `0` , a tensão será 0V e, se o valor decimal for `1204` , a tensão será 5V. Sendo assim, se o valor decimal apresentado for `256` , a tensão será aproximadamente 2,5V.

Na sequência, veja todas as funções para a manipulação dos pinos analógicos do Arduino.

`analogRead ()`

Lê o valor atual de um pino analógico. Os pinos analógicos do Arduino possuem um conversor analógico-digital de 10 bits, isso quer dizer que o comando `analogRead` vai converter voltagens entre 0V e 5V para valores decimais entre 0 e 255, sendo 0 sempre desligado e 255 sempre ligado.

Veja um exemplo da sintaxe:

```
analogRead(pino);
```

Veja um exemplo do uso:

```
int valor = analogRead(A0);
```

analogWrite()

Escreve um valor analógico para um pino analógico ou digital PWM. Os valores que podem ser escritos para os pinos podem variar de 0 até 1023.

Sintaxe:

```
analogWrite(pino);
```

Exemplo:

```
analogWrite(6);    // pino digital PWM

analogWrite(A2);    // pino analógico
```

Os pinos analógicos são usados para ler e escrever valores analógicos, ou seja, discretos e não apenas zeros e uns. Um bom exemplo de uso desses pinos é o potenciômetro. Vejamos a seguir.

12.1 EXPERIÊNCIA Nº 12 - USANDO UM POTENCIÔMETRO

Existem dois tipos de resistores: os fixos e os que são variáveis. Os resistores fixos são os que conhecemos bem em circuitos eletrônicos, e os variáveis são denominados potenciômetros ou reostatos. À medida que um cursor deslizante gira, o seu ponto de contato com o elemento resistivo muda, variando assim a resistência entre os terminais.

Com Arduino, podemos converter a tensão resultante entre um pino de alimentação e um pino analógico em valor decimal com o uso de um potenciômetro.

O que é necessário

1 x Arduino

1 x Potenciômetro de 1 KΩ

Esquema de montagem

Usando dois cabinhos, os terminais laterais do potenciômetro serão ligados aos pinos GND e 5V do Arduino. Usando um terceiro cabinho, o terminal central do potenciômetro será ligado ao pino analógico A0 do Arduino.

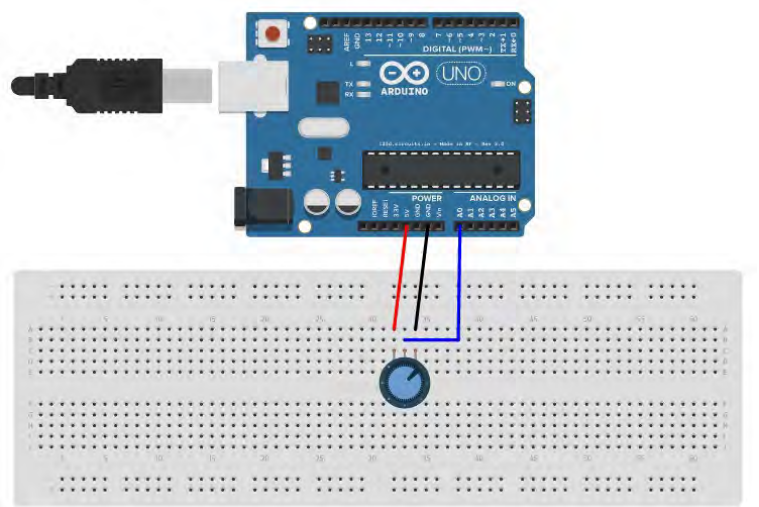


Figura 12.1: Esquema de montagem da experiência nº 12

Programação

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

void loop() {
    Serial.println(analogRead(A0));
    delay(10);
}
```

O que foi feito

Na função `setup()` , iniciamos a comunicação serial usando `Serial.begin(9600)` para acompanharmos a leitura do potenciômetro via Monitor Serial. Com `Serial.println(analogRead(A0))` , lemos o valor lido no pino analógico A0 do Arduino e enviamos via conexão serial para o computador. Usamos também o `delay(10)` para uma espera de apenas 10 milissegundos apenas para os números não passarem tão rápido.

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, abra o Monitor Serial e mova o cursor do potenciômetro. Você deverá ver os números mudarem de acordo com o movimento realizado no potenciômetro, resultante da variação da resistência.

Outro bom exemplo é o uso de um sensor de luminosidade.

12.2 EXPERIÊNCIA Nº 13 - USANDO UM SENSOR LUMINOSIDADE

Um dos objetivos da Computação Física é "sentir" o mundo e interagir com ele por meio de equipamentos digitais. Usando um sensor de luminosidade, é possível realizar essa tarefa, sendo que podemos medir a quantidade de luz em um ambiente e realizar alguma tarefa a partir dessa medição.

O que é necessário

1 x Arduino

1 x LDR 10KΩ

1 x Resistor de 1KΩ

Esquema de montagem

Guiando-se pela figura a seguir, ligue um resistor de 10KΩ entre o pino GND e o pino digital 2 do Arduino, usando dois cabinhos. Também usando cabinhos, ligue o LDR entre o pino digital 2 e o pino 5V do Arduino.

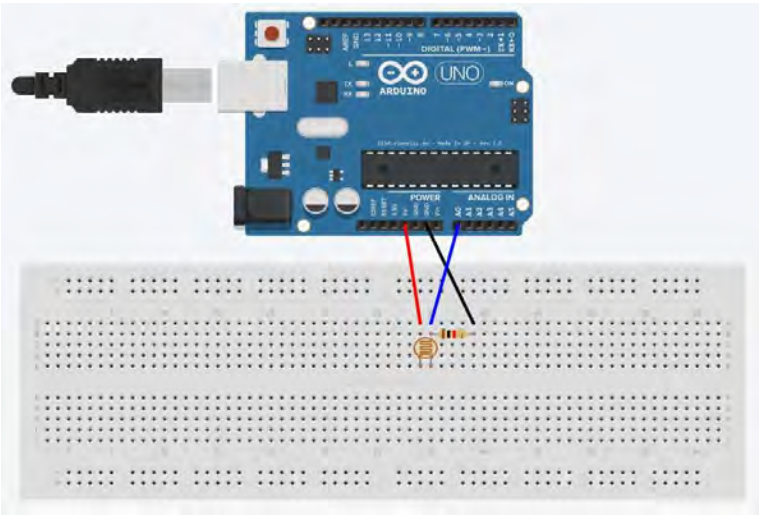


Figura 12.2: Esquema de montagem da experiência nº 13

Programação

```
int valor = 0;

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

void loop() {
    valor = analogRead(A0);
    Serial.println(valor);
    delay(500);
}
```

O que foi feito

Primeiro, inicializamos a variável que receberá a leitura com zero com `int valor = 0`. O próximo passo foi inicializar a comunicação serial na função `setup()` usando `Serial.begin(9600)`.

Com `valor = analogRead(A0)`, lemos o valor do pino analógico A0 do Arduino e o valor lido ficará armazenado na variável `valor`. Usamos `Serial.println(valor)` para imprimir o valor lido do pino analógico no Monitor Serial. No final, um `delay` para conseguirmos ver o valor na tela.

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, abra o Monitor Serial e faça com que o LDR receba mais ou menos luminosidade, e verifique que o valor impresso muda proporcionalmente a quantidade de luz.

No simulador, clique sobre o LDR para selecionar a quantidade de luz a ser simulada.

12.3 EXERCITE

Faça um alarme para detectar a porta aberta da geladeira. Funda os circuitos do LDR e do buzzer, e crie um programa que, ao detectar luz, emita um alarme sonoro. Esse pequeno e simples equipamento pode ser deixado dentro da geladeira e, caso a porta fique aberta, permitindo que alguma luz entre, ele emitirá o alarme. Faça um alarme para detectar a porta aberta da geladeira. Funda os circuitos do LDR e do buzzer, e crie um programa que, ao detectar luz, emita um alarme sonoro. Esse pequeno e simples equipamento pode ser deixado dentro da geladeira e, caso a porta fique aberta,

permitindo que alguma luz entre, ele emitirá o alarme.

12.4 A SEGUIR...

Muitos projetos dependem da contagem de tempo para atuar sobre dispositivos diversos, geração de sons de aviso ou alerta a quem for o operador do equipamento. Portanto, a seguir você terá a descrição de todas as funções para essas tarefas.

CAPÍTULO 13

FUNÇÕES ESPECIAIS

Como mencionado anteriormente, há algumas funções para geração de ruídos de alerta ou aviso para quem estiver operando o equipamento criado. Para gerar esse tipo de sinal, usamos os pinos digitais PWM do Arduino.

As funções `tone()` e `noTone()` fazem parte das funções especiais da linguagem. Veja a seguir a especificação para elas e também para as funções de contagem de tempo:

`tone()`

Gera uma onda na frequência especificada em um pino até que a duração seja cumprida, ou até que a função `noTone()` seja executada. A frequência mínima em um Arduino Uno é de aproximadamente 31 Hz e a máxima de aproximadamente 65.535 Hz.

Veja um exemplo da sintaxe:

```
tone(pino,frequência);

tone(pino, frequência, duração);
```

Veja um exemplo de uso:

```
tone(6,240);        // pino digital PWM 6, frequência de 240 Hz

tone(6,240,500);    // pino digital PWM 6, frequência de 240 Hz
                    // por 500 milissegundos
```

`noTone()`

Para a geração da onda criada pelo comando `tone` em um pino específico.

Sintaxe:

```
noTone(pino);
```

Exemplos:

```
noTone(6);
```

13.1 FUNÇÕES DE TEMPO

`millis()`

Retorna um número do tipo `unsigned long` contendo a quantidade de milissegundos em que o programa atual encontra-se em execução. Consegue contar até aproximadamente 50 dias, depois retorna a zero e recomeça a contagem. Lembre que em um segundo há 1.000 milissegundos.

Sintaxe:

```
millis();
```

Exemplos:

```
unsigned long tempo = millis();
```

`micros()`

Retorna um número do tipo `unsigned long` contendo a quantidade de microssegundos em que o programa atual encontra-se em execução. Consegue contar até aproximadamente 70 minutos, depois retorna a zero e recomeça a contagem. Note que em um segundo há 1.000.000 microssegundos.

Sintaxe:

```
micros();
```

Exemplos:

```
unsigned long tempo = micros();
```

delay()

Pausa a execução do programa por uma quantidade de milissegundos determinada.

Sintaxe:

```
delay(tempo);
```

Exemplos:

```
delay(1000);    // pausa o programa por um segundo

delay(13000);    // pausa o programa por 13 segundos
```

delayMicroseconds()

Pausa a execução do programa por uma quantidade de microssegundos determinada.

Sintaxe:

```
delayMicroseconds(tempo);
```

Exemplos:

```
delayMicroseconds(1000000);    // pausa o programa por um
                              // segundo

delayMicroseconds(13000000);    // pausa o programa por 13
                                // segundos
```

13.2 EXPERIÊNCIA Nº 14 - PROGRAMANDO UM RELÓGIO

Um bom exemplo para o uso das funções de tempo é programar um relógio. Você poderá ver o tempo passando no Monitor Serial e, se desejar, recorrer às experiências do display de 7-segmentos e tentar montar um relógio "de verdade".

O que é necessário

1 x Arduino

Esquema de montagem

Não há necessidade de qualquer componente exceto o próprio Arduino.

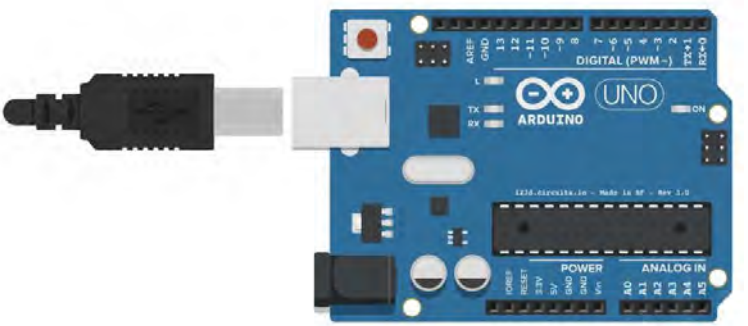


Figura 13.1: Esquema de montagem da experiência nº 14

Programação

```
int seg=0, min=0, hor=0;

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

void loop() {
    static unsigned long ult_tempo = 0;
    int tempo = millis();
    if(tempo - ult_tempo >= 1000) {
        ult_tempo = tempo;
```

```
        seg++;
    }
    if(seg>=60) {
        seg = 0;
        min++;
    }
    if(min>=60) {
        min = 0;
        hor++;
    }
    if(hor>=24) {
        hor=0;
        min=0;
    }
    Serial.print(hor);
    Serial.print(":");
    Serial.print(min);
    Serial.print(":");
    Serial.println(seg);

}
```

O que foi feito

Primeiro, precisamos criar variáveis para armazenar as horas, minutos e segundos. Fazemos isso com `int seg=0, min=0, hor=0` , sendo que, para segundos, usaremos a variável `seg` , para minutos a variável `min` , e para horas a variável `hor` .

Já na função `setup()` , inicializamos a comunicação serial com `Serial.begin(9600)` . Na função `loop()` , usamos `static unsigned long ult_tempo = 0` para declarar e inicializar a variável que será usada para a contagem de tempo, e com `int tempo = millis()` declaramos a variável `tempo` que recebe o retorno da função `millis()`

Com a estrutura de seleção `if(tempo - ult_tempo >= 1000)` , verificamos se, caso já tenham se passado 1.000 milissegundos (1 segundo), incrementamos a quantidade de segundos.

É preciso armazenar a última passagem de tempo, já que precisamos de algum valor para comparar o tempo atual e o que já passou. Fazemos isso com `ult_tempo = tempo` e, em seguida, é só incrementar a quantidade de segundos (`seg++`).

Usamos `if(seg>=60)` para verificar se já se passaram 60 segundos (1 minuto). Caso positivo, incrementamos a quantidade de minutos, e é preciso zerar os segundos para recomeçar a contagem deles. Fazemos isso com `seg = 0` e `min++` .

De forma similar ao que fazemos para os segundos, usamos `if(min>=60)` para verificar se já tenham se passado 60 minutos (1 hora). Caso positivo, incrementamos a quantidade de horas e é preciso zerar os minutos para recomeçar a contagem deles. Fazemos isso com `min = 0` e `hor++` .

Finalmente, só nos falta contar a quantidade de dias, então usamos `if(hor>=24)` para verificar se já se passaram 24 horas (1 dia). Se sim, é preciso zerar as horas e os minutos para recomeçar a contagem deles. Fazemos isso com `hor=0` e `min = 0` .

Depois de tudo isso, basta mostrar o tempo passando no Monitor Serial. Para isso, imprimimos os números usando `Serial.print(hor)` , `Serial.print(":")` , `Serial.print(min)` , `Serial.print(":")` e `Serial.println(seg)` .

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, abra o Monitor Serial e você verá a contagem de tempo.

13.3 EXERCITE

Junte a experiência deste capítulo com o uso de botões, e crie

uma maneira de ajustar as horas, minutos e segundos usando, pelo menos, três botões: um para incrementar os números, outro para decrementar, e o terceiro para selecionar a opção de qual variável deseja ajustar. Use o Monitor Serial para mostrar mensagens da opção selecionada e os valores ajustados.

13.4 A SEGUIR...

Apesar de a experiência de programar um relógio poder render várias outras ideias, principalmente, como já sugerido, unida às experiências que usam displays de 7-segmentos - e você pode explorá-las, se quiser -, já podemos continuar os estudos. A seguir, você encontrará como manipular números e caracteres utilizando todas as funções matemáticas disponíveis para a linguagem C para Arduino.

Capítulo 14

FUNÇÕES MATEMÁTICAS

As funções matemáticas e de caracteres são especialmente úteis quando é necessário realizar cálculos de dados recebidos de sensores, outros equipamentos ou mesmo Arduino. Também devem ser consideradas na hora de interagir com o usuário para mostrar valores inteligíveis.

`min()`

Retorna o menor valor entre dois números especificados.

Veja um exemplo de sintaxe:

```
min(valor,valor);
```

Veja um exemplo de uso:

```
int x = min(2,7);    // x será igual a 2
```

`max()`

Retorna o maior valor entre dois números especificados.

Sintaxe:

```
max(valor,valor);
```

Exemplo:

```
int x = max(2,7);    // x será igual a 7
```

`abs()`

Retorna o módulo de um valor (valor absoluto).

Sintaxe:

```
abs(valor);
```

Exemplos:

```
int x = abs(30);    // x será igual a 30

int x = abs(-30);
```

constrain()

Verifica se um número pertence a uma faixa de valores especificada.

Sintaxe:

```
constrain(valor,inicio da faixa, fim da faixa);
```

Exemplos:

```
int resultado = constrain(10,1,20); // retorna 10, pois 10
                                    // está dentro da faixa

int resultado = constrain(10,20,30);    // retorna 20, pois 10
                                        // é menor que o início
                                        // da faixa

int resultado = constrain(40,20,30);    // retorna 30, pois 40
                                        // é maior que o início
                                        // da faixa
```

map()

Retorna a relação entre um número e duas faixas de valores. Valor será uma relação entre a primeira faixa de valores (mínimo e máximo) sobre a segunda faixa de valores (mínimo e máximo).

Sintaxe:

```
map(valor, de menor valor, de maior valor, para menor valor, para
maior valor);
```

Exemplos:

```
int x = map(10,0,10,100,200);   // x será igual a 200

int x = map(0,0,10,100,200);    // x será igual a 100
```

pow()

Realiza potenciação com um número em relação a um expoente.

Sintaxe:

```
pow(base,expoente);
```

Exemplos:

```
float x = pow(3,2); // realiza 3² sendo assim x será igual
                    // a 9

float x = pow(4,-2);    // realiza 4-2 sendo assim x será
                        // igual a 2
```

sqrt()

Calcula a raiz quadrada de um número.

Sintaxe:

```
sqrt(número);
```

Exemplos:

```
double x = sqrt(4); // calcula a raiz quadrada de 4,
                    // sendo assim x será igual a 2

double x = sqrt(8); // calcula a raiz quadrada de 8,
                    // sendo assim x será igual a
                    // aproximadamente 2.82842712
```

sin()

Calcula o seno de um ângulo em radianos.

Sintaxe:

```
sin(ângulo);
```

Exemplos:

```
float x = sin(90);  // calcula o seno de 90 graus radianos,
                    // x será aproximadamente 0.89

float x = sin(45);  // calcula o seno de 45 graus radianos,
                    // x será aproximadamente 0.85
```

`cos()`

Calcula o cosseno de um ângulo em radianos.

Sintaxe:

```
cos(ângulo);
```

Exemplos:

```
float x = cos(90);  // calcula o cosseno de 90 graus radianos,
                    // x será aproximadamente -0.45

float x = cos(45);  // calcula o cosseno de 45 graus radianos,
                    // x será aproximadamente 0.53
```

`tan()`

Calcula a tangente de um número em radianos.

Sintaxe:

```
tan(ângulo);
```

Exemplos:

```
float x = tan(90);  // calcula o tangente de 90 graus radianos,
                    // x será aproximadamente -2.00

float x = tan(45);  // calcula o tangente de 45 graus radianos,
                    // x será aproximadamente 1.62
```

14.1 FUNÇÕES PARA CARACTERES

isAlphaNumeric()

Retorna verdadeiro (1 - TRUE) se o caractere for alfanumérico; caso contrário, retorna falso (0 - FALSE).

Veja um exemplo de sintaxe:

```
isAlphaNumeric(caractere);
```

Veja um exemplo de uso:

```
boolean t = isAlphaNumeric('K');    // retorna 1 - TRUE
```

isAlpha()

Retorna verdadeiro (1 - TRUE) se o caractere for uma letra; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isAlpha(caractere);
```

Exemplos:

```
boolean t = isAlpha('K');    // retorna 1 - TRUE

boolean t = isAlpha('1');    // retorna 0 - FALSE
```

isAscii()

Retorna verdadeiro (1 - TRUE) se o caractere for um ASCII; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isAscii(caractere);
```

Exemplo:

```
boolean t = isAscii('K');    // retorna 1 - TRUE
```

isWhitespace()

Retorna verdadeiro (1 - TRUE) se o caractere for um espaço em branco; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isWhitespace(caractere);
```

Exemplos:

```
boolean t = isWhitespace(' ');    // retorna 1 - TRUE

boolean t = isWhitespace('K');    // retorna 0 - FALSE
```

isControl()

Retorna verdadeiro (1 - TRUE) se o caractere for um de controle; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isControl(caractere);
```

Exemplo:

```
boolean t = isControl('K');    // retorna 0 - FALSE
```

isDigit()

Retorna verdadeiro (1 - TRUE) se o caractere for um número; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isDigit(caractere);
```

Exemplos:

```
boolean t = isDigit('1');    // retorna 1 - TRUE

boolean t = isDigit('K');    // retorna 0 - FALSE
```

isGraph()

Retorna verdadeiro (1 - TRUE) se o caractere for um imprimível; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isGraph(caractere);
```

Exemplo:

```
boolean t = isDigit('1');    // retorna 1 - TRUE
```

isLowerCase()

Retorna verdadeiro (1 - TRUE) se o caractere for uma letra minúscula; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isLowerCase(caractere);
```

Exemplos:

```
boolean t = isDigit('a');    // retorna 1 - TRUE

boolean t = isDigit('A');    // retorna 0 - FALSE
```

isPrintable()

Retorna verdadeiro (1 - TRUE) se o caractere for um imprimível; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isPrintable(caractere);
```

Exemplo:

```
boolean t = isPrintable('K');    // retorna 1 - TRUE
```

isPunct()

Retorna verdadeiro (1 - TRUE) se o caractere for um de pontuação; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isPunct(caractere);
```

Exemplos:

```
boolean t = isPunct(';');    // retorna 1 - TRUE

boolean t = isPunct('A');    // retorna 0 - FALSE
```

isSpace()

Retorna verdadeiro (1 - TRUE) se o caractere for um espaço em branco; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isSpace(caractere);
```

Exemplos:

```
boolean t = isSpace(' ');    // retorna 1 - TRUE

boolean t = isSpace('P');    // retorna 0 - FALSE
```

isUpperCase()

Retorna verdadeiro (1 - TRUE) se o caractere for uma letra maiúscula; caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isUpperCase(caractere);
```

Exemplos:

```
boolean t = isUpperCase('B');    // retorna 1 - TRUE

boolean t = isUpperCase('b');    // retorna 0 - FALSE
```

isHexadecimalDigit()

Retorna verdadeiro (1 - TRUE) se o caractere for um número

hexadecimal (de 0 até 9, de A até F); caso contrário, retorna falso (0 - FALSE).

Sintaxe:

```
isHexadecimalDigit(caractere);
```

Exemplos:

```
boolean t = isHexadecimalDigit('C');    // retorna 1 - TRUE

boolean t = isHexadecimalDigit('X');    // retorna 0 - FALSE
```

14.2 FUNÇÕES PARA NÚMEROS ALEATÓRIOS

`randomSeed()`

Inicializa o gerador de números pseudoaleatórios do Arduino. É importante gerar a sequência pseudoaleatório do `randomSeed` para que o comando random possa retornar um número aleatório a cada execução em um programa.

Veja um exemplo de sintaxe:

```
randomSeed(número);
```

Veja um exemplo de uso:

```
randomSeed(10000);
```

`random()`

Retorna um número aleatório entre zero e o número máximo, ou entre o número mínimo e o número máximo.

Sintaxe:

```
random(número máximo);

random(número mínimo, número máximo);
```

Exemplos:

```
long num = random(300);      // retorna um número aleatório
                             // entre zero e 300

long num = random(100,200); // retorna um número aleatório
                             // entre 100 e 200
```

14.3 EXERCITE

Crie um programa que calcule o seno de todos os ângulos entre 0 e 360 graus (use uma estrutura de repetição) e mostre o resultado no Monitor Serial. Feche o Monitor Serial e abra o Plotter Serial para vê-lo mostrar a curva correspondente ao cálculo do seno. Repita o mesmo para as funções do cosseno e da tangente para ver a diferença na curva.

14.4 A SEGUIR...

Agora você já deve saber como armazenar dados em variáveis, convertê-los e tratá-los com as mais diversas funções apresentadas até aqui. A seguir, você terá a descrição das funções para comunicação serial do Arduino. Com elas, é possível comunicar-se com sensores, máquinas e equipamentos que usem esse tipo de comunicação e também realizar transmissão de dados entre Arduinos.

Capítulo 15

FUNÇÕES PARA COMUNICAÇÃO

O grande mote hoje em dia é comunicar-se, transmitir e receber os dados mais variados possíveis e transformá-los em atuação sobre equipamentos, para que seja possível criar sistemas que interajam com o mundo físico. Assim que se faz a Internet das Coisas, por exemplo.

Como já mencionei anteriormente, podemos pensar em Arduino pela ótica da Internet das Coisas (*Internet of Things*, ou IoT), que visa criar equipamentos com capacidade e objetivo de transmitir dados e interagir diretamente com a internet, sem a efetiva intervenção humana. Dentre as vastas possibilidades, citei como exemplo estações meteorológicas automáticas, sensores de presença, eletrodomésticos automatizados e capazes de interagir um com outros, liberação de acesso e rastreadores veiculares.

Para esse tipo de funcionalidade, existem muitas funções de comunicação serial para o Arduino, que serão apresentadas na sequência.

`Serial()`

Utilizado para comunicação entre um Arduino e um computador, ou qualquer outro equipamento que seja capaz de realizar (receber e enviar) comunicação serial. Para o Arduino Uno,

os pinos 0 e 1 são compartilhados com os pinos de comunicação serial do microcontrolador que, por sua vez, também estão conectados ao conversor USB-Serial. Portanto, deve-se evitar usar esses pinos nas experiências para não prejudicar ou causar qualquer interferência na comunicação.

Para realizar a comunicação serial você pode usar os comandos a seguir:

`Serial.begin()`

Inicia a comunicação serial com a velocidade selecionada. As velocidades selecionadas são: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200 bauds.

Veja um exemplo de sintaxe:

```
Serial.begin(velocidade);
```

Veja um exemplo de uso:

```
Serial.begin(9600);

Serial.begin(115200);
```

`Serial.avaliable()`

Retorna o número de caracteres existentes para leitura via porta serial.

Sintaxe:

```
Serial.avaliable();
```

Exemplo:

```
if(Serial.available > 0) {  // se houverem caracteres
                            // para serem lidos
    // ler caracteres
}
```

`Serial.print()`

Envia dados para a porta serial em formato humanamente inteligível. Pode receber um formatador de dados para indicar se os dados deverão ser mostrados como binários, octais, decimais, hexadecimais ou números fracionários. Se o formatador for omitido, os dados são mostrados literalmente.

Sintaxe:

```
Serial.print(caracteres, formatador);
```

Exemplos:

```
Serial.print(78, BIN);      // mostrará 1001110
Serial.print(78, OCT);      // mostrará 116
Serial.print(78, DEC);      // mostrará 78
Serial.print(78, HEX);      // mostrará 4E
Serial.println(1.23456,0);  // mostrará 1
Serial.print(1.23456,2);    // mostrará 1.23
Serial.print(1.23456,4);    // mostrará 1.2346
```

`Serial.println()`

Envia dados para a porta serial em formato humanamente inteligível. Pode receber um formatador de dados para indicar se os dados deverão ser mostrados como binários, octais, decimais, hexadecimais ou números fracionários. Se o formatador for omitido, os dados são mostrados literalmente.

Ao final da linha insere um caractere para retorno de carro (*carriage return*) e avanço de linha (*line feed*), equivalente ao `ENTER`.

Sintaxe:

```
Serial.println(caracteres, formatador);
```

Exemplos:

```
Serial.println(78, BIN);    // mostrará 1001110
```

```
Serial.println(78, OCT);    // mostrará 116
Serial.println(78, DEC);    // mostrará 78
Serial.println(78, HEX);    // mostrará 4E
Serial.println(1.23456,0);  // mostrará 1
Serial.println(1.23456,2);  // mostrará 1.23
Serial.println(1.23456,4);  // mostrará 1.2346
```

`Serial.write()`

Envia dados, como uma string, por exemplo, binariamente para a porta serial. Os dados são enviados byte a byte para a comunicação serial, e os caracteres representam o valor numérico. Para mostrar os caracteres legivelmente, use o `Serial.print` ou `Serial.println` . Também retorna a quantidade de bytes que foram enviados.

Sintaxe:

```
Serial.write(valor);

serial.write(string);
```

Exemplos:

```
Serial.write(45);   // envia apenas um byte
                    // com o valor 45

int qtde = Serial.write("Arduino"); // envia a string
                                    // Arduino e retorna 7
```

`Serial.writeln()`

Envia dados, como uma string, por exemplo, binariamente para a porta serial. Os dados são enviados byte a byte para a comunicação serial, e os caracteres representam o valor numérico. Para mostrar os caracteres legivelmente, use o `Serial.print` ou `Serial.println` .

Ao final da linha, insere um caractere para retorno de carro (*carriage return*) e avanço de linha (*line feed*), equivalente ao `ENTER` . Também retorna a quantidade de bytes que foram

enviados.

Sintaxe:

```
Serial.writeln(valor);

Serial.writeln(string);
```

Exemplos:

```
Serial.writeln(45);    // envia apenas um byte com o valor 45

int qtde = Serial.writeln("Arduino");   // envia a string
                                        // Arduino e retorna 7
```

`Serial.read()`

Lê dados pela porta serial. Retorna -1 caso não haja bytes para serem lidos.

Sintaxe:

```
Serial.read();
```

Exemplos:

```
if(Serial.available() > 0) {
    int qtde = Serial.read();   // armazenará o byte lido
                                // na variável qtde
}
```

Bom, vamos colocar as mãos na massa. A experiência nº 15 serve para lidar com a conexão serial entre o Arduino e o computador. Um pouco mais adiante, vamos executar uma experiência para recebermos dados do computador no Arduino. Esta será a experiência nº 16.

15.1 EXPERIÊNCIA Nº 15 - ENVIAR DADOS DO ARDUINO PARA O COMPUTADOR PELA COMUNICAÇÃO SERIAL

Quando conectamos o Arduino ao computador por meio do cabo e porta USB, imediatamente é criada uma porta serial virtual no computador de forma que possamos enviar nossos sketches para a placa. Da mesma maneira, essa porta serial virtual pode ser usada para enviar dados para o Arduino e também receber dados gerados pelo Arduino.

Nesta experiência, vamos gerar dados com o Arduino e enviá-los serialmente para o computador. Para ver os dados gerados, utilizaremos o Monitor Serial do Ambiente de Programação Arduino. Com isso, você pode enviar quaisquer dados para o computador ou qualquer outro equipamento que possua comunicação através de portas seriais.

O que é necessário

1 x Arduino

Esquema de montagem

Não há necessidade de montagens, apenas conecte o seu Arduino ao seu computador usando o cabo e a porta USB.

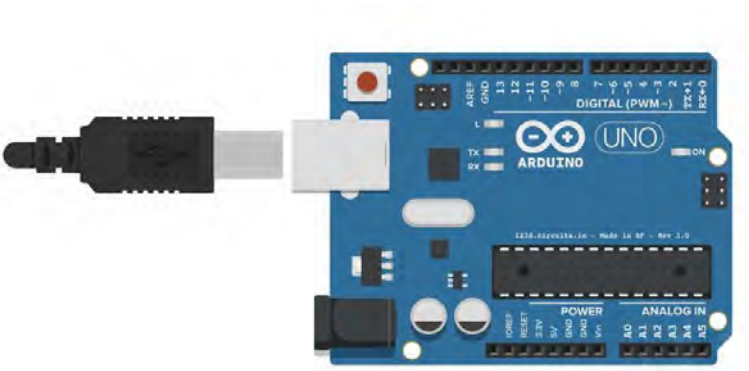


Figura 15.1: Esquema de montagem da experiência nº 15

Programação

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

void loop() {
    Serial.write("Oi!\n");
    delay(1000);
}
```

O que foi feito

Como o comando `Serial.begin(9600)` na função `setup()`, iniciamos a comunicação serial com taxa de 9.600 bauds. Na função `loop()`, usamos o comando `Serial.write("Oi!\n")` para enviar a sequência de caracteres `Oi!` para a porta serial, seguido do caractere newline (`\n`), que corresponde a um `ENTER`.

Coloquei uma pausa de um segundo usando o comando `delay`, para que tudo não fique muito rápido.

Resultado esperado

Após compilar e enviar ao Arduino, abra o Monitor Serial. Você verá a palavra "Oi!" ser recebida a cada um segundo:

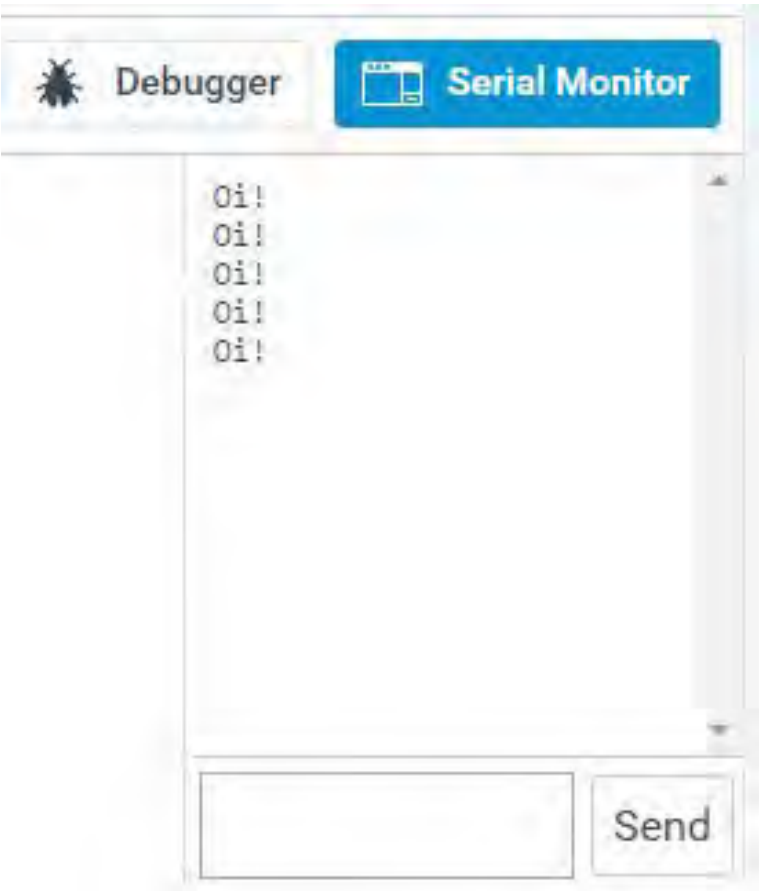


Figura 15.2: Monitor Serial com a mensagem recebida

15.2 EXPERIÊNCIA Nº 16 - RECEBER DADOS DO COMPUTADOR PELA CONEXÃO SERIAL E CONTROLAR UM LED

Podemos tanto receber quanto enviar dados do Arduino pela porta serial (vimos isso na experiência anterior), o que será o fluxo de dados partindo do Arduino para o computador, neste caso. Mas você pode usar isso para enviar dados serialmente para qualquer equipamento que possua suporte a esse tipo de comunicação.

Com essa experiência, vamos enviar um caractere para o Arduino e, se esse caractere for `a` (a letra **a** minúscula), o LED do pino 13 será aceso; caso seja o caractere `b` (a letra **b** minúscula), o LED do pino 13 será apagado. Qualquer outro caractere não terá qualquer ação sobre o LED do pino 13.

O que é necessário

1 x Arduino UNO

Esquema de montagem

Não há necessidade de montagens, apenas conecte o seu Arduino ao seu computador usando o cabo e a porta USB.

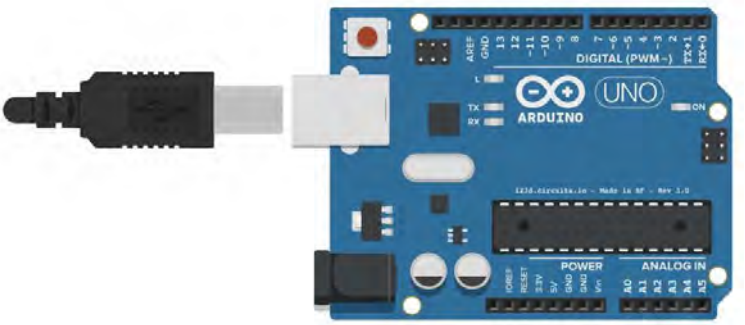


Figura 15.3: Esquema de montagem da experiência nº 16

Programação

```
int recebido = 0;
int led = 13;

void setup() {
    Serial.begin(9600);
    pinMode(led,OUTPUT);
}

void loop() {
```

```
    if(Serial.available() > 0) {
        recebido = Serial.read();
        if(recebido == 'a') {
            digitalWrite(led,HIGH);
        }
        if(recebido == 'b') {
            digitalWrite(led,LOW);
        }
    }
}
```

O que foi feito

Primeiro, declaramos uma variável global do tipo inteiro, chamada `recebido`, que é inicializada com zero. Essa variável armazenará os bytes recebidos pela comunicação serial.

Com `int led = 13`, inicializamos com 13 a variável que armazenará em qual pino está o LED que vamos manipular. A variável será do tipo inteiro e terá como nome `led`. Usando o comando `Serial.begin(9600)`, iniciamos a comunicação serial com 9.600 bauds e, em seguida, ajustamos o pino digital 13 (usando a variável `led`) para saída, usando o comando `pinMode(led,OUTPUT)`.

Com a estrutura de seleção `if(Serial.available() > 0)`, verificamos se há dados a serem recebidos pela porta serial, no caso, o comando `Serial.available()`, que retorna a quantidade de bytes disponíveis para serem recebidos. Ou seja, se retornar qualquer número maior que zero, é sinal de que há caracteres disponíveis.

Se houver dados a serem lidos pela comunicação serial, usamos `recebido = Serial.read()` para ler um caractere da porta serial e armazenar na variável `recebido`. Essa variável é do tipo inteiro, pois armazenará o valor ASCII do caractere.

Na sequência, usamos a estrutura de seleção `if` como

`if(recebido == 'a')` para verificar se o caractere recebido foi uma letra `a` minúscula. Note que são usadas aspas simples em vez de aspas para indicar um byte. Aspas são usadas para indicar cadeira de caracteres, ou seja, Strings.

Caso a letra `a` minúscula que tenha sido recebida, usamos `digitalWrite(led,HIGH)` para acender o LED do pino digital 13. Com `if(recebido == 'b')`, verificamos se o caractere recebido foi a letra `b` minúscula. Caso a letra `b` tenha sido recebida, usamos `digitalWrite(led,LOW)` para apagar o LED do pino digital 13.

Resultado esperado

Após compilar e enviar ao Arduino, abra o Monitor Serial do ambiente de programação. Se você enviar a letra `a` (minúsculo), o LED conectado ao pino 13 vai acender e, caso você envie a letra `b` (minúsculo), o LED conectado ao pino 13 apagará.

Agora vamos unir a comunicação serial e LEDs, os resultados serão fantásticos, veja na experiência seguinte.

15.3 EXPERIÊNCIA Nº 17 - CONTROLAR 3 LEDS PELA SERIAL

A montagem para essa experiência é basicamente igual à experiência de número 4, porém na programação poderemos escolher o LED que queremos acender usando dados enviados pela porta serial.

Vamos escolher um caractere, que será um número, para cada LED. Quando recebido, vai acendê-lo caso esteja apagado, e apagá-lo caso esteja aceso.

O que é necessário

1 x Arduino UNO

3 x LED

3 x Resistores de 220Ω

Esquema de montagem

O primeiro passo, como sempre, é colocar os LEDs na placa de ensaios. Oriente-se pela figura a seguir.

Usando um cabinho, ligue o terminal negativo do primeiro LED ao um dos pinos GND disponíveis no Arduino. Repita o mesmo para os dois outros LEDs. Lembre-se de que o terminal negativo é o mais longo em um LED.

Agora ligue o terminal positivo do primeiro LED a um resistor de pelo menos 220Ω e, em seguida, usando um cabinho, ligue o outro terminal do resistor ao pino digital 4 do Arduino.

Vamos repetir a mesma coisa para os demais LEDs, então, ligue o terminal positivo do segundo LED a um resistor de pelo menos 220Ω. Depois ligue o outro terminal do resistor ao pino digital 3 do Arduino, usando um cabinho.

Finalmente, ligue o terminal positivo do terceiro LED a um resistor de pelo menos 220Ω, e depois ligue o outro terminal do resistor, usando um cabinho, ao pino digital 2 do Arduino.

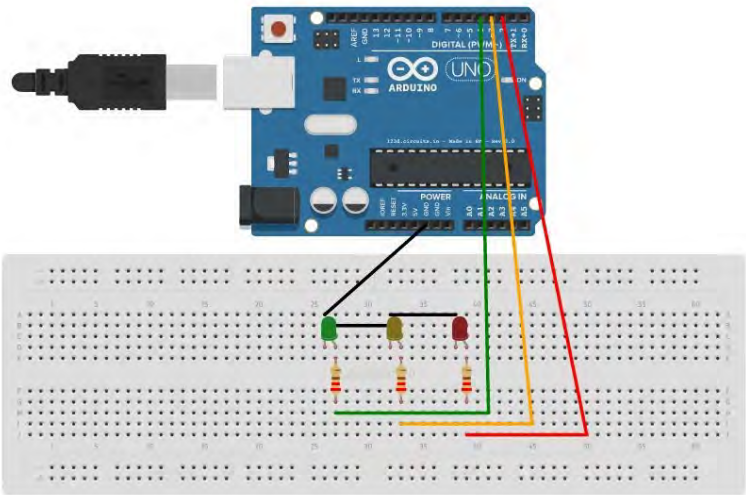


Figura 15.4: Esquema de montagem da experiência nº 17

Programação

```
int recebido = 0;
int dois = HIGH;
int tres = HIGH;
int quatro = HIGH;

void setup() {
    Serial.begin(9600);
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    pinMode(4,OUTPUT);
    digitalWrite(2,dois);
    digitalWrite(3,tres);
    digitalWrite(4,quatro);
}

void loop() {
    if(Serial.available() > 0) {
        recebido = Serial.read();
        if(recebido == '2') {
            if(dois == HIGH) dois = LOW;
            else dois = HIGH;
            digitalWrite(2,dois);
        }
        if(recebido == '3') {
            if(tres == HIGH) tres = LOW;
```

```
            else tres = HIGH;
            digitalWrite(3,tres);
        }
        if(recebido == '4') {
            if(quatro == HIGH) quatro = LOW;
            else quatro = HIGH;
            digitalWrite(4,quatro);
        }
    }
}
```

O que foi feito

Como já fizemos nas experiências anteriores, começamos a função `setup()` declarando uma variável chamada `recebido` para receber o byte que virá pela comunicação serial. Ela deve ser inicializada com zero.

Continuando, declaramos três variáveis do tipo inteiro, uma para cada LED, que serão inicializadas com o valor inicial `HIGH`, ou seja, os LEDs começarão acesos. Mais adiante, elas terão seus valores alternados entre `LOW` e `HIGH` novamente, permitindo acender ou apagar o LED. As variáveis terão como nome `dois`, `tres` e `quatro` para lembrar-nos dos pinos digitais onde os LEDs estão conectados.

Como receberemos dados pela comunicação serial, precisamos inicializá-la, o que fazemos com `Serial.begin(9600)`. Também precisamos ajustar os pinos digitais 2, 3 e 4 para saída. Fazemos isso com `pinMode(2,OUTPUT)`, `pinMode(3,OUTPUT)` e `pinMode(4,OUTPUT)`. Ainda na função `setup()`, acendemos os três LEDs enviando o conteúdo das variáveis `dois`, `tres` e `quatro` que são `HIGH` para eles.

Precisamos saber se há dados da comunicação serial para serem lidos. Fazemos isso usando a estrutura de seleção `if(Serial.available() > 0)` e conjunto com o comando `Serial.available()`, que retorna o número de bytes disponíveis

para serem recebidos. Ou seja, se retornar qualquer número maior que zero, é sinal de que há caracteres disponíveis.

Com `recebido = Serial.read()` , lemos um caractere da porta serial e armazenamos seu valor na variável recebida. Essa variável é do tipo inteiro, pois vai armazenar o valor ASCII do caractere.

A partir desse ponto, começamos a verificar qual LED deve ser aceso, caso esteja apagado; ou apagado, caso esteja aceso. O número a ser recebido é igual ao pino digital em que o LED está conectado, assim fica mais fácil de memorizar.

Por exemplo, caso o caractere recebido seja o número 2, verificado com `if(recebido == '2')` , e o conteúdo da variável `dois` seja `HIGH` (LED aceso), então ele deve ser apagado; caso contrário, ele deve ser aceso. Isso acontece na estrutura de seleção `if(dois == HIGH) dois = LOW; else dois = HIGH` .

Note que o valor `HIGH` e `LOW` é atribuído à variável `dois` , correspondente ao LED. Para efetivamente acender ou apagar, precisamos enviar esse valor ao pino digital; fazemos isso usando `digitalWrite(2,dois)` . Na sequência, realizamos as mesmas operações com os LEDs que estão nos pinos digitais 3 e 4.

Resultado esperado

Após compilar e enviar ao Arduino, abra o Monitor Serial do ambiente de programação. Se você enviar o número 2 e o LED conectado ao pino 2 estiver aceso, ele vai apagar; caso contrário, ele vai acender. Se você enviar o número 3 e o LED conectado ao pino 3 estiver aceso, ele apagará; caso contrário, ele acenderá. E se você enviar o número 4 e o LED conectado ao pino 4 estiver aceso, ele vai apagar; caso contrário, ele acenderá.

15.4 EXERCITE

Faça a experiência de acender o LED do pino 13 usando um botão em dois Arduinos. Depois, modifique a experiência, usando as funções de comunicação serial, e faça com que o botão de um Arduino controle o LED do outro, e vice-versa.

15.5 A SEGUIR...

Juntar todas as experiências mostradas neste capítulo com as anteriores trará à luz várias possibilidades de projetos, ainda mais e unidas às que serão apresentadas a seguir, que são as experiências que usam as bibliotecas adicionais para a linguagem C para Arduino.

Capítulo 16

BIBLIOTECAS ADICIONAIS

As bibliotecas estendem o conjunto de comandos da linguagem C para Arduino, de forma que novas funcionalidades sejam possíveis, e outras, que antes requeriam grande esforço de programação, sejam ainda mais fáceis.

As que estão incluídas no IDE Arduino são:

EEPROM

Possui funções para ler e gravar dados em armazenamento permanente. Essa biblioteca em especial é especialmente útil, portanto, vamos testá-la na experiência nº 18, mais à frente.

Servo

Possui funções para controlar Servo Motores. Controlar servos é muito divertido e pode abrir a imaginação para a criação de robôs diversos. Logo, vamos experimentá-la também na experiência nº 19, mais adiante.

SoftwareSerial

Possui funções que permitem comunicar-se serialmente usando pinos que não sejam os pinos padrão de comunicação serial, como por exemplo, no Arduino UNO os pinos 0 e 1. É mais fácil de entender testando, o que será feito na experiência nº 20.

Ethernet

Possui funções para conectar o Arduino à rede Ethernet e Internet usando um Shield Ethernet.

Firmata

Possui funções para comunicar-se com aplicações em um computador usando o protocolo serial padrão.

GSM

Possui funções para conectar-se à rede GSM/GPRS usando um Shield GSM.

LiquidCrystal

Possui funções para controlar displays de LCD.

SD

Possui funções para ler e gravar em cartões do tipo SD.

SPI

Possui funções para comunicação entre equipamentos usando a Serial Peripheral Interface.

Stepper

Possui funções que permitem controlar motores de passo.

16.1 TFT

Possui funções quer permitem desenhar, imprimir imagens e figuras em telas TFT (*Thin Film Transistor*, ou Transistor de

Película Fina).

WiFi

Possui funções para conectar o Arduino à rede Ethernet e Internet usando um Shield WiFi. Depende de *shield* externo.

Wire

Possui funções para o uso das duas interfaces Wire: TWI e I2C.

16.2 EXPERIÊNCIA Nº 18 - COMO USAR A MEMÓRIA EEPROM

Como já vimos, o microcontrolador no Arduino possui uma memória *Electrically-Erasable Programmable Read-Only Memory* (EEPROM), ou em português, memória programável somente para leitura apagável eletricamente.

Essa é uma memória que armazena dados mesmo sem alimentação elétrica. Esses dados podem ser gravados e apagados, mas essas operações podem ser feitas por um número de vezes limitado, algo entre 100 mil e 1 milhão de vezes.

Os microcontroladores ATmega328 possuem EEPROM de 1024 bytes, já o ATmega168 e o Atmega8 possuem 512 bytes.

Essa memória é importante quando necessitamos armazenar, por exemplo, configurações do equipamento, que devem ser mantidas mesmo depois que ele for desligado, para que no próximo uso as configurações não tenham que ser refeitas. A gravação na EEPROM é feita utilizando-se a biblioteca `EEPROM.h` , e enviando byte a byte para a memória.

Nessa experiência, criaremos um contador que será gravado na

EEPROM que continuará a contar do mesmo número, mesmo se o Arduino for desligado e religado.

Primeiro, verificamos se os dados que estão na EEPROM são os que esperamos que fossem: na primeira posição da memória (posição zero), é um sinalizador que inventamos e, na segunda posição (posição um), é o valor do nosso contador, que será incrementado em um a cada segundo.

Você deve reparar que acessar a memória EEPROM é muito simples, basta referenciar-se a ela como se fosse um vetor do tamanho da quantidade de bytes disponíveis nela, no caso do Arduino UNO R3, de zero até 1023 (1 KiB). A comunicação serial entre o computador e o Arduino será usada como interface para conferirmos o que está acontecendo.

O que é necessário

1 x Arduino

Esquema de montagem

Não há necessidade de qualquer componente exceto o próprio Arduino.

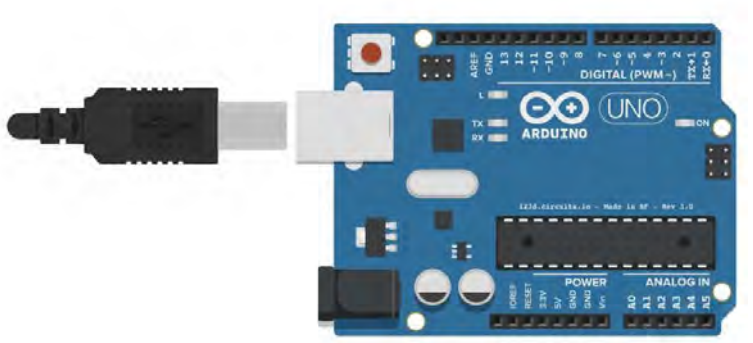


Figura 16.1: Esquema de montagem da experiência nº 18

Programação

```
#include <EEPROM.h>

int contador = 0;

void setup() {
    Serial.begin(9600);
    if(EEPROM[0] != 9) {
        EEPROM[0] = 9;
        EEPROM[10] = 0;
    } else {
        contador = EEPROM[10];
    }
}

void loop() {
    Serial.println(contador);
    contador++;
    EEPROM[10] = contador;
    delay(1000);
}
```

O que foi feito

Primeiro, importamos a biblioteca EEPROM que implementa os comandos e interfaces para facilitar o uso da memória usando `#include <EEPROM.h>`. Em seguida, usamos `int contador = 0` para declarar e inicializar a variável `contador`, que armazenará o valor e será incrementado. Já na função `setup()`, usamos `Serial.begin(9600)` para iniciar a comunicação serial entre o computador e o Arduino.

Como o acesso à memória EEPROM funciona como se ela fosse um vetor predefinido com o nome `EEPROM` (em maiúsculas), usamos `if(EEPROM[0] != 9)` para verificar se há o valor 9 na primeira posição da memória (`0`). Caso não haja o valor 9 nessa posição, quer dizer que nosso contador está sendo usado pela primeira vez e deve ser inicializado.

Com `EEPROM[0] = 9`, colocamos o valor 9 na primeira posição

da EEPROM, indicando que ele já foi inicializado uma vez. Com `EEPROM[1] = 0` na segunda posição da memória, vai o valor do contador iniciado com zero. Bom, se o contador já tiver sido iniciado alguma vez, o código depois do `else` é que será executado.

Usamos `contador = EEPROM[1]` para ler o conteúdo da segunda posição da memória para a variável contador, e `Serial.println(contador)` para mostrar o conteúdo da variável `contador` no Monitor Serial. Com `contador++` , incrementamos em um o contador e, com `EEPROM[1] = contador` , armazenamos na memória o valor da variável `contador` . Finalmente, um `delay(1000)` para pausar por um segundo.

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, abra o Monitor Serial e você verá o contador ser incrementado um a cada segundo. Se você resetar o Arduino, ou desligá-lo e ligá-lo novamente, o contador continuará do número imediatamente seguinte ao que estava quando foi resetado ou desligado.

Note que:

1. Se você não abrir o Monitor Serial, o contador funcionará mesmo assim. Portanto, você não verá os números sendo incrementados;
2. Essa experiência funcionará apenas no Arduino real, no simulador ainda não há implementação das memórias internas do microcontrolador.

16.3 EXPERIÊNCIA Nº 19 - ACIONANDO UM SERVO MOTOR

Um servo motor é um motor capaz de reagir com movimento proporcional a um pulso aplicado a ele. O tamanho do pulso determina a posição do eixo.

Ao contrário dos motores comuns, ele tem liberdade, normalmente, de apenas 180 graus e possui grande precisão quanto à posição. Porém, saiba que existem servos motores que podem ter giro de 260 graus.

São compostos de 3 partes básicas:

1. Sistema atuador, sendo o motor propriamente dito. Normalmente acionado por corrente contínua. O torque, velocidade, material das engrenagens e caixa de redução são determinantes das características de um servo motor.

2. Sensor, que é um potenciômetro ligado ao eixo do motor. O valor da resistência elétrica do sensor (potenciômetro) determina a posição do eixo do motor. A qualidade do sensor influi na precisão, durabilidade e estabilidade do servo motor.

3. Circuito de controle recebe o comando e determina a posição do eixo do motor em relação à resistência elétrica do sensor. O comando normalmente é um pulso cujo tamanho determina a posição.

Figura 16.2: Um mini servo motor

O que é necessário

1 x Arduino

1 x Mini servo motor de 5V

Esquema de montagem

Usando um cabinho ou fazendo a ligação diretamente com o cabo disponível no motor, ligue o fio de alimentação +5V do servo motor (geralmente vermelho) no 5V do Arduino. Em seguida, também usando um cabinho ou fazendo a ligação diretamente com o cabo disponível no motor, ligue o fio terra GND do servo motor (geralmente marrom ou preto) ao GND do Arduino.

Finalmente, usando o mesmo esquema dos dois fios anteriores, ligue o fio de pulso (geralmente branco ou amarelo/laranja) ao pino digital 2 do Arduino.

Figura 16.3: Esquema de montagem da experiência nº 19

Programação

```
#include<Servo.h>

Servo mm;

void setup() {
    mm.attach(2);
    pinMode(13,OUTPUT);
}

void loop() {
    mm.write(1);
    digitalWrite(13,HIGH);
    delay(2000);
    mm.write(180);
    digitalWrite(13,LOW);
    delay(2000);
}
```

O que foi feito

O uso de bibliotecas e criação de objetos é semelhante ao uso da comunicação serial, mas, obviamente, dessa vez será para acionar um servo motor.

Com `#include<Servo.h>`, adicionamos ao programa a biblioteca `Servo.h`, que possui os comandos específicos para controlar servo motores, como geração do pulso no tamanho correto para girá-lo quantos graus forem necessários ou desejados. Com o comando `Servo mm`, declaramos o objeto `mm` como sendo

um servo motor.

Na função `setup()` com a linha `mm.attach(2)` , indicamos que o fio de sinal do servo estará conectado ao pino digital 2 do Arduino. Declarei o pino digital 13 como saída para usarmos o LED desse pino para *debug*. Ou seja, verificar se a experiência está em funcionamento.

Na função `loop()` , o comando `mm.write(1)` indica ao servo motor para que ele se posicione em 1 grau. Em seguida, acendemos o LED do pino digital 13 com `digitalWrite(13,HIGH)` .

Aguardamos dois segundos e, em seguida, com o comando `mm.write(180)` , indicamos ao servo motor para que ele se posicione em 180 graus. Depois, apagamos o LED do pino digital 13 mais dois segundos de espera para podermos perceber o movimento do motor.

Resultado esperado

Depois de compilar e enviar o programa ao Arduino, o servo motor deve começar a se movimentar colocando primeiro o eixo na posição de 1°, aguardar dois segundos e reposicioná-lo em 180°, aguardar mais dois segundos e repetir toda a ação. Antes de posicionar o motor em 1°, o LED do pino 13 deverá acender, e depois de reposicioná-lo em 180°, o LED do pino 13 deve apagar.

16.4 EXPERIÊNCIA Nº 20 - REALIZANDO COMUNICAÇÃO SERIAL ENTRE ARDUINOS

Em vários projetos, são necessários dois ou mais Arduinos, seja pela falta de pinos disponíveis, necessidade de maior processamento, gerenciamento de vários sensores ou vários outros motivos. É possível interligar vários Arduinos de várias formas, por

exemplo, ligação serial com fio, ligação serial sem fio, e até mesmo através de sinal comum em um pino (`HIGH` ou `LOW`). Vamos usar comunicação serial com fio para essa experiência.

O cabo serial (3 fios) entre os Arduinos pode ter entre 15 e 30 metros, dependendo da taxa de transmissão de dados que for usada. As taxas suportadas são (bit/s): 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 e 115200.

O que é necessário

2 x Arduino

Esquema de montagem

Usando um cabinho, ligue o pino digital 10 de um Arduino ao pino digital 11 do outro Arduino. Com um segundo cabinho, ligue o pino digital 11 de um Arduino ao pino digital 10 do outro Arduino. Com um terceiro cabinho, ligue o GND de um Arduino ao GND do outro Arduino.

Se preferir, em vez de três fios separados, você pode usar um cabo de três vias.

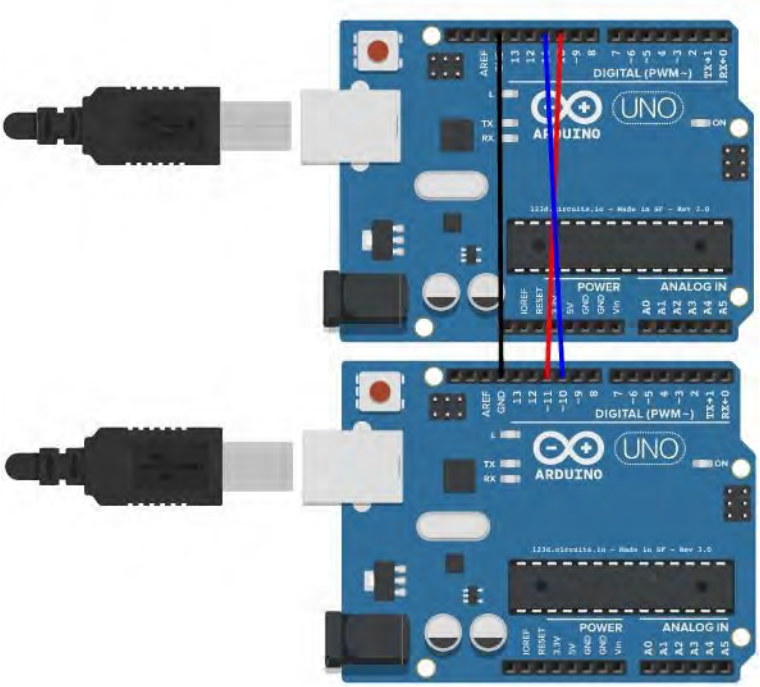


Figura 16.4: Esquema de montagem da experiência nº 20

Programação

```
#include <SoftwareSerial.h>

SoftwareSerial outro_arduino(10,11); // RX TX
int recebido = 0;

void setup() {
     Serial.begin(9600);
     outro_arduino.begin(4800);
}

void loop() {
     if(Serial.available() > 0)  {
          outro_arduino.write(Serial.read());
    }
     if(outro_arduino.available() > 0) {
          Serial.write(outro_arduino.read());
    }
}
```

O que foi feito

A primeira coisa feita foi importar a biblioteca SoftwareSerial usando `#include <SoftwareSerial.h>` , que é uma biblioteca padrão do ambiente Arduino que possibilita criar uma porta serial em dois pinos digitais quaisquer.

Com a biblioteca importada, é preciso criar um objeto indicando quais os pinos que serão utilizados para receber e enviar dados. Fazemos isso com `SoftwareSerial outro_arduino(10,11)` . O pino digital de recepção (RX) será o 10, e o pino digital para transmissão (TX) será o 11. Esses pinos serão usados para a comunicação entre os Arduino.

Na função `setup()` , a variável `recebido` receberá o que for recebido pela comunicação serial, e `Serial.begin(9600)` iniciará a comunicação serial entre o Arduino e o computador. Já `outro_arduino.begin(4800)` iniciará a comunicação serial entre os Arduinos pela porta serial criada nos pinos digitais 10 e 11.

Como a ideia é enviar caracteres de um Arduino para o outro usando o Monitor Serial, primeiro verificamos se há alguma coisa a ser lida vindo do computador. Caso haja, enviamos o que for recebido para a comunicação entre os Arduinos. Fazemos isso usando `if(Serial.available() > 0)` , para ver se há dados recebidos pela comunicação serial entre o Arduino e o PC, e `outro_arduino.write(Serial.read())` , para enviar o que foi lido para o outro Arduino.

Agora verificamos se há caracteres a serem lidos da comunicação entre os Arduino. Caso haja, enviamos o que for lido para o computador usando a conexão serial correspondente. Usamos `if(outro_arduino.available() > 0)` para ver se há dados recebidos pela comunicação entre os Arduinos pela porta serial criada nos pinos digitais 10 e 11, e

`Serial.write(outro_arduino.read())` para enviar o que foi lido para a comunicação serial entre o Arduino e o computador.

Resultado esperado

Compila e carregue o mesmo programa em ambos os Arduinos. No caso do uso do simulador, você deve selecionar um Arduino, enviar, e depois selecionar o outro Arduino e enviar. Você deve abrir um Monitor Serial para cada um deles e, ao enviar dados de Monitor Serial, eles serão mostrados no Monitor Serial do outro Arduino, como se fosse um bate-papo, por exemplo.

16.5 EXERCITE

Una a experiência em que usamos um potenciômetro com a que controlamos um servo motor, e faça com que o potenciômetro passe a controlar a posição do servo motor. Quando você mover o potenciômetro para um lado, o motor deve mover-se para o mesmo lado.

16.6 A SEGUIR...

Para terminar, a seguir você verá como fazer para acionar motores DC (corrente contínua) usando o Arduino. Quando esses motores são unidos com tudo o que vimos anteriormente, os projetos ganham vida!

Capítulo 17

ACIONANDO MOTORES DC

Os pinos digitais e analógicos do Arduino podem ser utilizados para acionar uma grande variedade de equipamentos. Entretanto, eles não possuem potência para alimentar motores, por exemplo, já que o máximo de corrente fornecida ou drenada por esses pinos é de 40mA.

A seguir, veja duas experiências de como lidar com esse tipo de problema.

17.1 EXPERIÊNCIA Nº 21 - ACIONANDO UM MOTOR DC USANDO RELÊ

O primeiro passo, ou o último, dependendo do ponto vista, para montar um robô com rodas é saber controlar os motores para movimentá-lo. Com o Arduino, isso é relativamente simples e necessita de poucas peças e componentes eletrônicos baratos.

Porém, é preciso entender que o Arduino não possui corrente nem potência suficiente em seus pinos para controlar um motor DC (corrente contínua). Por isso, nem pense em ligar qualquer motor diretamente ao seu Arduino, pois isso pode "matá-lo". Para resolver isso, você pode usar um relê para isolar o circuito do motor do circuito do Arduino.

O que é necessário

1 x Arduino

1 x Relê 5V na bobina e pelo menos 9V nos contatos

1 x Motor DC 9V

Esquema de montagem

Usando cabinhos, ligue um terminal da bobina do relê no GND do Arduino. Em seguida, ligue o outro terminal da bobina do relê no pino digital 7 do Arduino, e ligue o negativo da bateria ao pino da chave do relê.

Agora, ligue um terminal do motor (a polaridade apenas influencia na direção de rotação do motor) no outro pino da chave do relê e, finalmente, ligue o positivo da bateria ao outro terminal do motor.

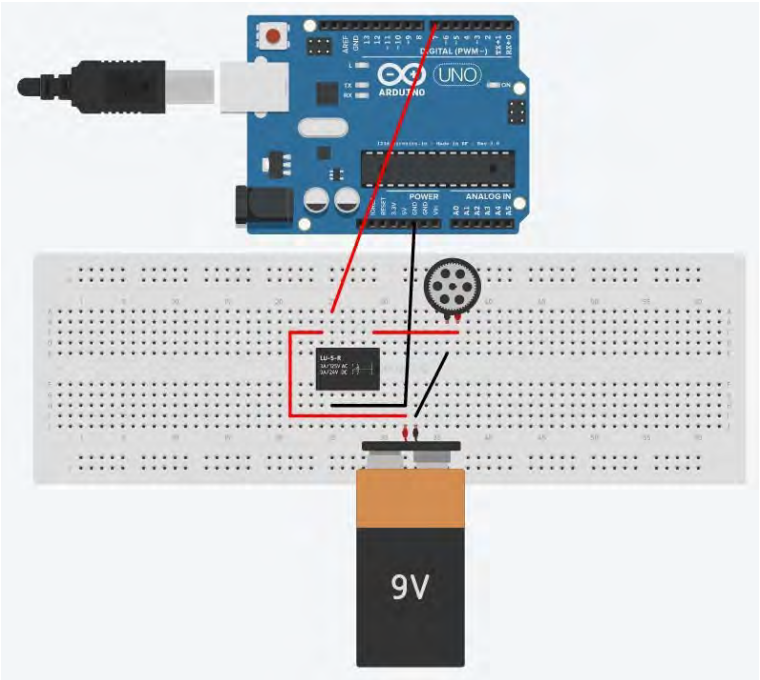


Figura 17.1: Esquema de montagem da experiência nº 21

Programação

```
int rele = 7;

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

void loop() {
  digitalWrite(rele,HIGH);
  delay(1000);
  digitalWrite(rele,LOW);
  delay(1000);
}
```

O que foi feito

Primeiro, com `int rele = 7`, definimos como valor 7 a variável inteira `rele`, pois ela será utilizada para indicar quem qual

pino digital está ligado o rele. Na função `setup()`, definimos o pino digital 7 (variável `rele`) como saída usando `pinMode(rele,OUTPUT)`.

Já na função `loop()`, acionamos o rele (`digitalWrite(rele,HIGH)`), aguardamos um segundo (`delay(1000)`) em que o motor ficará em movimento e, em seguida, desligamos o rele (`digitalWrite(rele,LOW)`), fazendo com que o motor pare, também aguardando um segundo (`delay(1000)`) em que o motor ficará parado.

Resultado esperado

Depois de compilar e enviar o programa para o Arduino, o motor deverá ficar um segundo ligado (girando) e um segundo desligado (parado).

17.2 EXPERIÊNCIA Nº 22 - ACIONANDO UM MOTOR DC USANDO TRANSISTOR

Essa é uma segunda versão para a experiência anterior que, em vez de usar um relê, usa um transistor para a tarefa de chaveamento para o motor.

O que é necessário

1 x Arduino

1 x Transistor TIP 120

1 x Motor DC 9V

Esquema de montagem

Usando cabinhos, ligue o terminal base do transistor ao pino

digital 7, depois ligue o terminal emissor do transistor a um terminal do motor, e ligue o terminal coletor do transistor ao positivo da bateria.

Agora, ligue o outro terminal do motor ao negativo da bateria e, finalmente, ligue o negativo da bateria ao GND do Arduino.

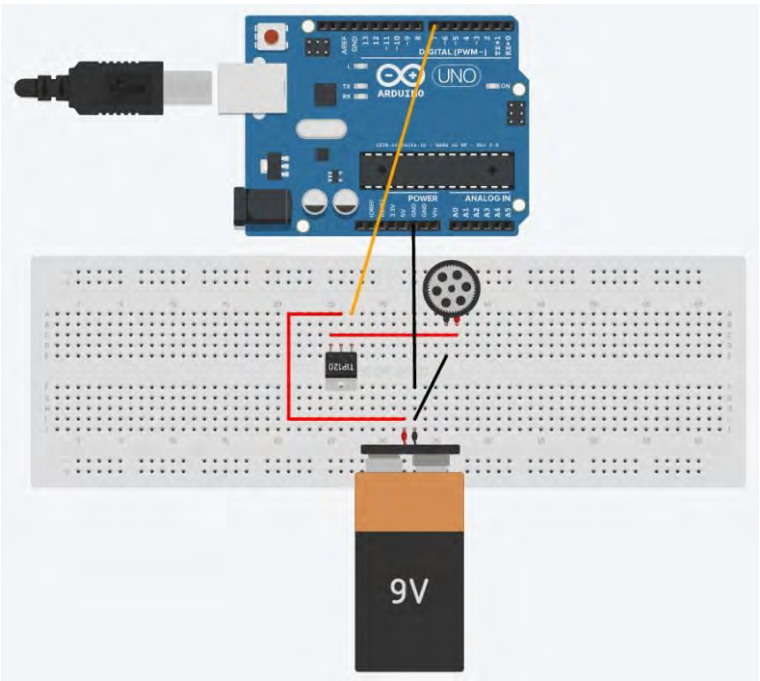


Figura 17.2: Esquema de montagem da experiência nº 22

Programação

```
int transistor = 7;

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

void loop() {
  digitalWrite(transistor,HIGH);
  delay(1000);
  digitalWrite(transistor,LOW);
```

```
  delay(1000);
}
```

O que foi feito

Primeiro, com `int transistor = 7` , definimos como valor 7 a variável inteira `transistor` , pois ela será usada para indicar qual pino digital está ligado o transistor.

Na função `setup()` , definimos o pino digital 7 (variável `transistor`) como saída usando `pinMode(transistor,OUTPUT)` . Já na função `loop()` , acionamos o transistor (`digitalWrite(transistor,HIGH)`), aguardamos um segundo (`delay(1000)`) em que o motor ficará em movimento e, depois, desligamos o transistor (`digitalWrite(transistor,LOW)`), fazendo com que o motor pare, também aguardando um segundo (`delay(1000)`) em que o motor ficará parado.

Resultado esperado

Basicamente, o que muda entre essa experiência e a que aciona um LED é a eletrônica, só que, em vez de um LED acender, é um motor que gira.

Depois de compilar e enviar o programa para o Arduino, o motor deverá ficar um segundo ligado (girando) e um segundo desligado (parado).

17.3 EXERCITE

Junte a experiência de controle de um motor DC, tanto por relê quanto por transistor, com a experiência de controlar LEDs e a do botão. Faça um programa que inicie com um LED vermelho aceso e, quando o botão for pressionado, o motor comece a girar, o LED

vermelho apague e um LED verde acenda. Quando o botão for solto, o motor deve parar de girar, o LED verde deve apagar e o LED vermelho deve acender.

17.4 CONCLUINDO

Com tudo o que foi apresentado, você poderá estudar e compreender o funcionamento do Arduino e sua lógica para programação. Consciente dos mecanismos básicos, você pode agora desenvolver os mais variados projetos, seja os sugeridos nas experiências ou qualquer outro.

É importante praticar e manter-se praticando o tempo todo, pois isso trará desafios que lhe impulsionarão para encontrar soluções e, conforme crescer a complexidade desses desafios, novos horizontes serão mostrados.

O simulador é uma grande ferramenta que pode garantir um começo mais suave, sem receios de danificar qualquer componente. Mas depois de ganhar confiança, procure usar o Arduino e os demais componentes reais. Tê-los nas mãos lhe dará senso de peso e tamanho.

Não tema em errar, aprenda com seus erros. O Arduino é o equipamento mais caro de todos da lista de materiais que usamos durante o livro e, ainda assim, não passa de algumas dezenas de reais. Os demais costumam custar centavos e sempre menos de R$ 10,00.

Desafie-se!

Para terminar, a seguir você terá uma lista de materiais utilizados em todas as experiências apresentadas no livro, com ela você pode programar-se para adquirir todos os componentes e experimentá-los conforme sua imaginação.

Capítulo 18

APÊNDICE A - LISTA DE MATERIAIS

Para a realização de todas as experiências mostrada durante o livro, você pode optar por usar o simulador de Arduino disponibilizado pela Autodesk. Mas caso deseje realizar todos usando os equipamentos e componentes físicos, a seguir está uma lista de tudo do que você precisará, as devidas quantidades e uma pequena descrição da utilização do componente.

1. **Arduino UNO R3** - Original ou clone - 2 unidades.
2. **Prot-o-board** - Placa de ensaios - 1 unidade.
3. **LED** - Tipo de diodo que ilumina quando a eletricidade passa por ele. Tem apenas uma cor - 3 unidades.
4. **Resistor 220Ω** - Resiste ao fluxo de energia elétrica em um circuito, resultando na alteração da voltagem e a corrente - 3 unidades.
5. **Resistor 10KΩ** - Resiste ao fluxo de energia elétrica em um circuito, resultando na alteração da voltagem e a corrente - 1 unidade.
6. **Resistor 1KΩ** - Resiste ao fluxo de energia elétrica em um circuito, resultando na alteração da voltagem e a corrente - 1 unidade.

7. **LED RGB** - Tipo de diodo que ilumina quando a eletricidade passa por ele. Consegue gerar qualquer cor - 1 unidade.

8. **Botão** - Chave momentânea que permite a passagem de eletricidade quando mantido pressionado - 1 unidade.

9. **Potenciômetro** - Resistor variável. Quando os pinos laterais do potenciômetro são conectados à voltagem e à terra, o pino central vai fornecer a diferença em voltagem com relação à rotação do cursor - 1 unidade.

10. **Mini Servo Motor** - Motor capaz de controlar a posição do eixo de rotação baseando-se no comprimento de um sinal - 1 unidade.

11. **Display de 7-segmentos** - É uma forma de display capaz de mostrar números decimais composto por 7 LEDs (segmentos) e, em alguns casos, mais o ponto (DP) - 1 unidade.

12. **Buzzer** - Dispositivo de sinal audível que pode ser mecânico, eletromecânico ou piezelétrico - 1 unidade.

13. **Speaker** - Dispositivo de sinal audível que pode ser mecânico, eletromecânico ou piezelétrico - 1 unidade.

14. **Relê 5V** - Chave mecânica operada eletricamente - 1 unidade.

15. **Motor DC 9V** - Tipo de motor elétrico movido por corrente contínua - 1 unidade.

16. **Suporte para bateria 9V** - Suporte para acoplar a bateria de 9 volts - 1 unidade.

17. **Bateria 9V** - Acumulador elétrico - 1 unidade.

18. **Transistor TIP120** - Equipamento eletrônico capaz, mas não somente, de ter o mesmo funcionamento de um relê, mas sem

o componente mecânico - 1 unidade.

19. **LDR 10KΩ** - O diodo dependente de luz (*Light Dependent Resistor*) é um componente de resistência variável que fornece a diferença em voltagem com relação à quantidade de luz incidente sobre ele - 1 unidade.

Capítulo 19

APÊNDICE B - EXERCÍCIOS RESOLVIDOS

19.1 EXERCÍCIO DO CAPÍTULO 8. ESTRUTURA PRINCIPAL DA LINGUAGEM

Projete e construa um conjunto de semáforos em um cruzamento. Você terá de controlar dois semáforos de veículos com 3 luzes (vermelho, amarelo e verde), e mais dois de pedestres com 2 luzes (vermelho e verde).

Faça tudo sincronizado, inclusive dando tempo suficiente para o pedestre atravessar enquanto dos dois semáforos de veículos permanecem vermelho.

Sugestão de montagem

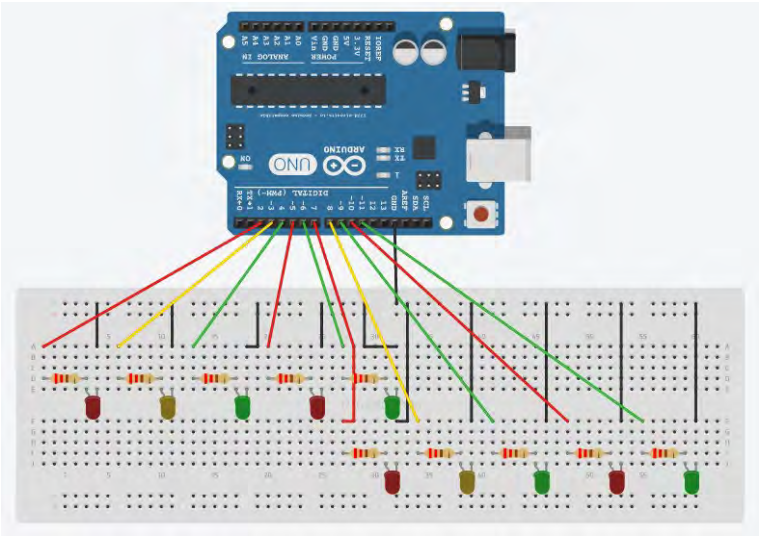


Figura 19.1: Sugestão de montagem do exercício do capítulo 08

Sugestão de programação

```
// Definindo os pinos
int vei1_red = 2;
int vei1_yel = 3;
int vei1_gre = 4;
int ped1_red = 5;
int ped1_gre = 6;
int vei2_red = 7;
int vei2_yel = 8;
int vei2_gre = 9;
int ped2_red = 10;
int ped2_gre = 11;

void setup() {
    // Ajustando o modo dos pinos
    pinMode(vei1_red, OUTPUT);
    pinMode(vei1_yel, OUTPUT);
    pinMode(vei1_gre, OUTPUT);
    pinMode(ped1_red, OUTPUT);
    pinMode(ped1_gre, OUTPUT);
    pinMode(vei2_red, OUTPUT);
    pinMode(vei2_yel, OUTPUT);
    pinMode(vei2_gre, OUTPUT);
    pinMode(ped2_red, OUTPUT);
    pinMode(ped2_gre, OUTPUT);
```

```
}

void loop() {
    // Acendendo e apagando luzes
    digitalWrite(ped1_gre, HIGH);
    digitalWrite(ped1_red, LOW);
    digitalWrite(ped2_red, HIGH);
    digitalWrite(vei1_red, HIGH);
    digitalWrite(vei2_gre, HIGH);
    delay(3000);
    digitalWrite(vei2_gre, LOW);
    digitalWrite(vei2_yel, HIGH);
    delay(3000);
    digitalWrite(vei2_yel, LOW);
    digitalWrite(vei2_red, HIGH);
    digitalWrite(ped1_gre, HIGH);
    digitalWrite(ped2_gre, HIGH);
    digitalWrite(ped2_red, LOW);
    delay(3000);
    digitalWrite(vei1_red, LOW);
    digitalWrite(vei1_gre, HIGH);
    digitalWrite(ped1_gre, LOW);
    digitalWrite(ped1_red, HIGH);
    delay(3000);
    digitalWrite(vei1_gre, LOW);
    digitalWrite(vei1_yel, HIGH);
    delay(3000);
    digitalWrite(ped1_gre, HIGH);
    digitalWrite(ped2_gre, LOW);
    digitalWrite(ped2_red, HIGH);
    digitalWrite(vei1_yel, LOW);
    digitalWrite(vei2_red, LOW);
}
```

19.2 EXERCÍCIO DO CAPÍTULO 9. AS FUNÇÕES PARA PINOS DIGITAIS

Faça com que um LED pisque uma vez a cada segundo e, em seguida, adicione um botão ao projeto. Quando você apertar o botão, o LED deve parar de piscar. Apertando novamente, ele deve voltar a piscar.

Sugestão de montagem

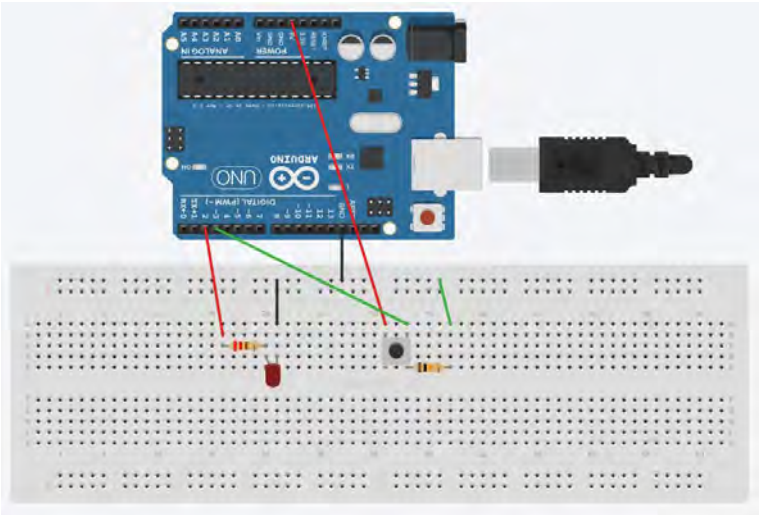


Figura 19.2: Sugestão de montagem do exercício do capítulo 09

Sugestão de programação

```
int led = 2;
int botao = 3;
int press = 0;
int status = 0;

void setup() {
    pinMode(led,OUTPUT);
    pinMode(botao,INPUT);
}
void loop() {
    press = digitalRead(botao);
    // while ao invés de if porque o botão
    // pode ser segurado apertado
    while((press == HIGH) && (status == 0)) {
        status = 1;
        press = LOW;
    }
    while((press == HIGH) && (status == 1)) {
        status = 0;
        press = LOW;
    }
    if(status == 0) {
        digitalWrite(led,HIGH);
        delay(500);
        digitalWrite(led,LOW);
```

```
        delay(500);
    }
}
```

19.3 EXERCÍCIO DO CAPÍTULO 10. TIPOS DE DADOS, VARIÁVEIS E CONVERSORES

Adicione um botão ao circuito da experiência que mostra números de zero a dez no display de 7-segmentos. Quando você apertar o botão, a contagem deve zerar.

Lembre-se de que a contagem deve ir apenas até nove e depois voltar até zero novamente.

Sugestão de montagem

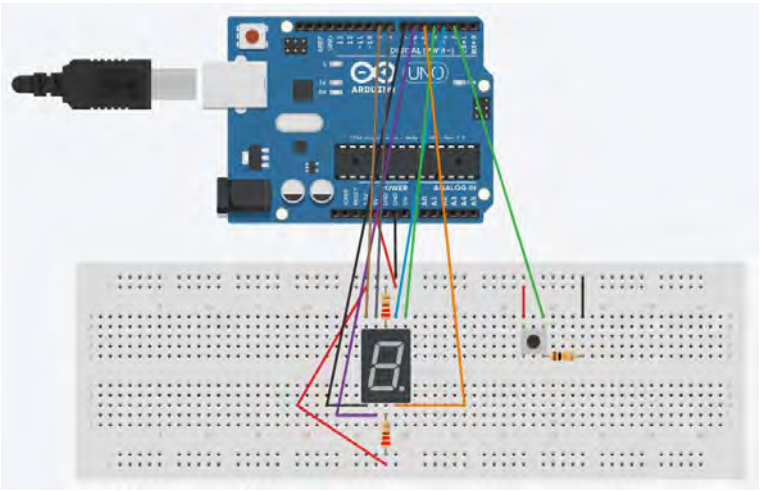


Figura 19.3: Sugestão de montagem do exercício do capítulo 10

Sugestão de programação

```
int botao = 2;
int press = 0;
int status = 0;
int a=3, b=4, c=5, d=6, e=7, f=8, g=9;
```

```
int num[10][7] = {
    {a,b,c,d,e,f},      // 0
    {b,c},              // 1
    {a,b,e,d,g},        // 2
    {a,b,c,d,g},        // 3
    {b,c,f,g},          // 4
    {a,c,d,f,g},        // 5
    {a,c,d,e,f,g},      // 6
    {a,b,c},            // 7
    {a,b,c,d,e,f,g},    // 8
    {a,b,c,f,g}         // 9
};

void setup() {
    pinMode(a,OUTPUT);
    pinMode(b,OUTPUT);
    pinMode(c,OUTPUT);
    pinMode(d,OUTPUT);
    pinMode(e,OUTPUT);
    pinMode(f,OUTPUT);
    pinMode(g,OUTPUT);
}

void loop() {
    for(int i=0;i<10;i++) {
        apaga();
        numero(i);
        delay(1000);
        press = digitalRead(botao);
        if(press == HIGH) break;
    }
}

void apaga() {
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,HIGH);
    digitalWrite(g,HIGH);
}

void numero(int n) {
    for(int i=0;i<7;i++) digitalWrite(num[n][i],LOW);
}
```

19.4 EXERCÍCIO DO CAPÍTULO 11. OPERADORES E ESTRUTURAS DE SELEÇÃO E REPETIÇÃO

Usando o buzzer e as funções corretas, tente reproduzir alguma música simples que seja fácil de reconhecer, como por exemplo, "batatinha quando nasce" ou "ciranda cirandinha". Deixe a música dentro da função `setup()` para que seja executada apenas uma vez.

Sugestão de montagem

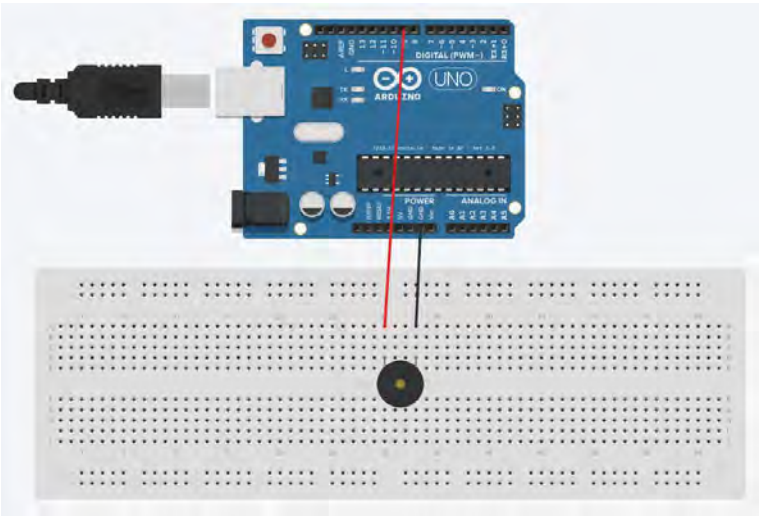


Figura 19.4: Sugestão de montagem do exercício do capítulo 11

Sugestão de programação

```
int som = 9;
int Do = 262;
int Re = 294;
int Mi = 330;
int Fa = 349;
int Sol = 392;
int La = 440;
```

```
int Si = 494;
int Do1 = 277;
int Re1 = 311;
int Fa1 = 370;
int Sol1 = 415;
int La1 = 466;
int Do2 = 523;
int Re2 = 587;
int Mi2 = 659;

void setup () {
    pinMode (som, OUTPUT);
    tone (som, Mi);
    delay (200);
    noTone (som);
    tone (som, Fa);
    delay (200);
    noTone (som);
    tone (som, Sol);
    delay (200);
    noTone (som);
    tone (som, Sol);
    delay (200);
    noTone (som);
    tone (som, Sol);
    delay (200);
    noTone (som);
    tone (som, Fa);
    delay (200);
    noTone (som);
    tone (som, Mi);
    delay (200);
    noTone (som);
    tone (som, La);
    delay (200);
    noTone (som);
    tone (som, La);
    delay (200);
    noTone (som);
    delay (200);
    tone (som, Re);
    delay (200);
    noTone (som);
    tone (som, Mi);
    delay (200);
    noTone (som);
    tone (som, Fa);
    delay (200);
```

```
noTone (som);
tone (som, Fa);
delay (200);
noTone (som);
tone (som, Fa);
delay (200);
noTone (som);
tone (som, Mi);
delay (200);
noTone (som);
tone (som, Re);
delay (200);
noTone (som);
tone (som, Sol);
delay (200);
noTone (som);
delay (200);
tone (som, Mi);
delay (200);
noTone (som);
tone (som, Sol);
delay (200);
noTone (som);
tone (som, Sol);
delay (200);
noTone (som);
tone (som, Sol);
delay (200);
noTone (som);
tone (som, Fa);
delay (200);
noTone (som);
tone (som, Mi);
delay (200);
noTone (som);
tone (som, Do2);
delay (200);
noTone (som);
tone (som, Si);
delay (200);
noTone (som);
tone (som, La);
delay (200);
noTone (som);
tone (som, Sol);
delay (200);
noTone (som);
tone (som, Fa);
```

```
    delay (200);
    noTone (som);
    tone (som, Mi);
    delay (200);
    noTone (som);
    tone (som, Re);
    delay (200);
    noTone (som);
    tone (som, Do);
    delay (200);
    noTone (som);
}

void loop () {
}
```

19.5 EXERCÍCIO DO CAPÍTULO 12. FUNÇÕES PARA PINOS ANALÓGICOS

Faça um alarme para detectar a porta aberta da geladeira. Funda os circuitos do LDR e do buzzer, e crie um programa que, ao detectar luz, emita um alarme sonoro. Esse pequeno e simples equipamento pode ser deixado dentro da geladeira e, caso a porta fique aberta permitindo que alguma luz entre, ele emitirá o alarme.

Sugestão de montagem

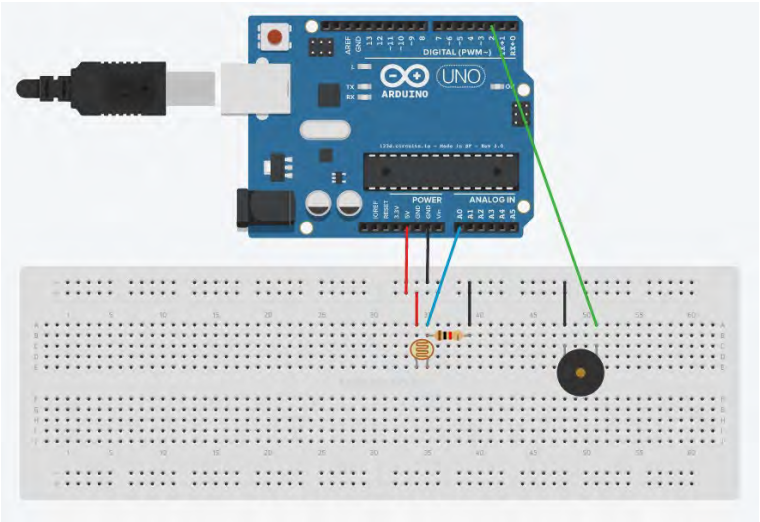


Figura 19.5: Sugestão de montagem do exercício do capítulo 12

Sugestão de programação

```
int valor = 0;

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

void loop() {
    valor = analogRead(0);
    Serial.println(valor);
    if(valor >= 600) {      // Limiar de luz
        tone(2,1000);       // para disparar o alarme
        delay(300);
        noTone(2);
        delay(300);
    }
}
```

19.6 EXERCÍCIO DO CAPÍTULO 13. FUNÇÕES ESPECIAIS

Junte a experiência deste capítulo com o uso de botões, e crie

uma maneira de ajustar as horas, minutos e segundos usando, pelo menos, três botões: um para incrementar os números, outro para decrementar, e o terceiro para selecionar a opção de qual variável deseja ajustar. Use o Monitor Serial para mostrar mensagens da opção selecionada e os valores ajustados.

Sugestão de montagem

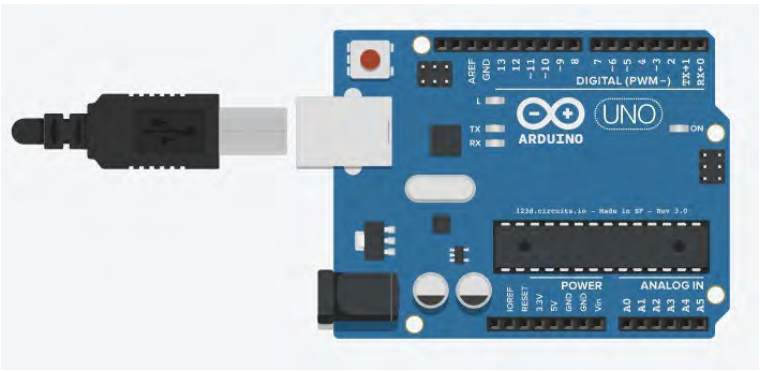


Figura 19.6: Sugestão de montagem do exercício do capítulo 13

Sugestão de programação

```
int seg=0, min=0, hor=0;
int bup = 12, bdown = 11, bsele = 10;
int psele = 0, pup = 0, pdown = 0;
int mostra_menu = 1, opcao = 0, teste = 0;

void setup() {
    Serial.begin(9600);
    pinMode(bup, OUTPUT);
    pinMode(bdown, OUTPUT);
    pinMode(bsele, OUTPUT);
}

void loop() {
    // Primeiro menu
    psele = digitalRead(bsele);
    if(psele == HIGH) {
        while(teste == 0) {
            if(mostra_menu == 1) {
                Serial.println("Pressione bsele novamente para sel
```

```
ecionar");
                Serial.println("Pressione bup e bdown para navegar'
);
                if(opcao == 0)
                    Serial.println("Ajustar a hora?");
                if(opcao == 1)
                    Serial.println("Ajustar os minutos?");
                if(opcao == 2)
                    Serial.println("Ajustar os segundos?");
                mostra_menu = 0;
            }
            psele = digitalRead(bsele);
            pup = digitalRead(bup);
            pdown = digitalRead(bdown);
            if(psele == HIGH) {
                teste = 99;
                psele = LOW;
            }
            if(pup == HIGH) {
                opcao++;
                mostra_menu = 1;
                pup = LOW;
                if(opcao > 2)
                    opcao = 0;
            }
            if(pdown == HIGH) {
                opcao--;
                mostra_menu = 1;
                pdown = LOW;
                if(opcao < 0)
                    opcao = 2;
            }
        }
    }
    // ------------
    // Segundo menu
    mostra_menu = 1;
    while(teste == 99) {
        if(mostra_menu == 1) {
            if(opcao == 0)
                Serial.println("Ajustando as horas bsele encerra")
;
            if(opcao == 1)
                Serial.println("Ajustando as minutos bsele encerra'
);
            if(opcao == 2)
                Serial.println("Ajustando as segundos bsele encerr
a");
```

```
            Serial.println("bup para incrementar e bdown para decr
ementar");
            mostra_menu = 0;
        }
        psele = digitalRead(bsele);
        pup = digitalRead(bup);
        pdown = digitalRead(bdown);
        if(psele == HIGH) {
            teste = 0;
            psele = LOW;
            break;
        }
        if(pup == HIGH) {
            // while evita o botão pressionado por muito tempo
            while(pup == HIGH) pup = digitalRead(bup);
            if(opcao == 0) hor++;
            if(opcao == 1) min++;
            if(opcao == 2) seg++;
            if(hor > 23) hor = 0;
            if(min > 59) min = 0;
            if(seg > 59) seg = 0;
            pup = LOW;
            Serial.print(hor);
            Serial.print(":");
            Serial.print(min);
            Serial.print(":");
            Serial.println(seg);
        }
        if(pdown == HIGH) {
            while(pdown == HIGH) pdown = digitalRead(bdown);
            if(opcao == 0) hor--;
            if(opcao == 1) min--;
            if(opcao == 2) seg--;
            if(hor < 0) hor = 23;
            if(min < 1) min = 59;
            if(seg < 1) seg = 59;
            pdown = LOW;
            Serial.print(hor);
            Serial.print(":");
            Serial.print(min);
            Serial.print(":");
            Serial.println(seg);
        }
    }
    // ------------
    static unsigned long ult_tempo = 0;
    int tempo = millis();
    if(tempo - ult_tempo >= 1000) {
```

```
        ult_tempo = tempo;
        seg++;
    }
    if(seg>=60) {
        seg = 0;
        min++;
    }
    if(min>=60) {
        min = 0;
        hor++;
    }
    if(hor>=24) {
        hor=0;
        min=0;
    }
    Serial.print(hor);
    Serial.print(":");
    Serial.print(min);
    Serial.print(":");
    Serial.println(seg);
}
```

19.7 EXERCÍCIO DO CAPÍTULO 14. FUNÇÕES MATEMÁTICAS

Crie um programa que calcule o seno de todos os ângulos entre 0 e 360 graus (use uma estrutura de repetição) e mostre o resultado no Monitor Serial. Feche o Monitor Serial e abra o Plotter Serial para vê-lo mostrar a curva correspondente ao cálculo do seno. Repita o mesmo para as funções do cosseno e da tangente para ver a diferença na curva.

Sugestão de montagem

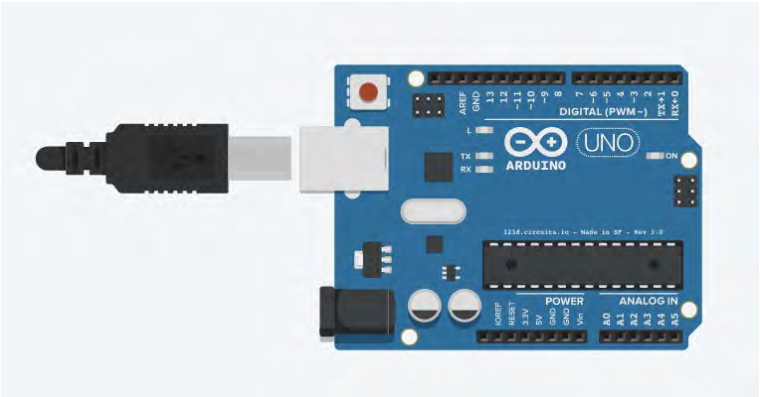


Figura 19.7: Sugestão de montagem do exercício do capítulo 14

Sugestão de programação

```
float pi = 3.14159;

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

void loop() {
    for(int i=0;i<360;i++) {
        // Tem que converter os graus para radianos
        Serial.println(sin(i * (pi / 180)));
    }
}
```

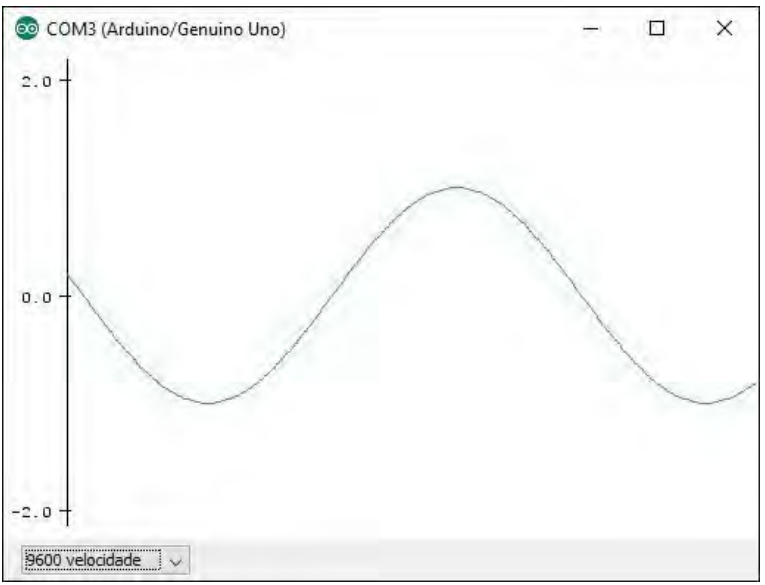


Figura 19.8: O resultado no Plotter Serial do exercício do capítulo 14

19.8 EXERCÍCIO DO CAPÍTULO 15. FUNÇÕES PARA COMUNICAÇÃO

Faça a experiência de acender o LED do pino 13 usando um botão em dois Arduinos. Depois, modifique a experiência, usando as funções de comunicação serial, e faça com que o botão de um Arduino controle o LED do outro e vice-versa.

Sugestão de montagem

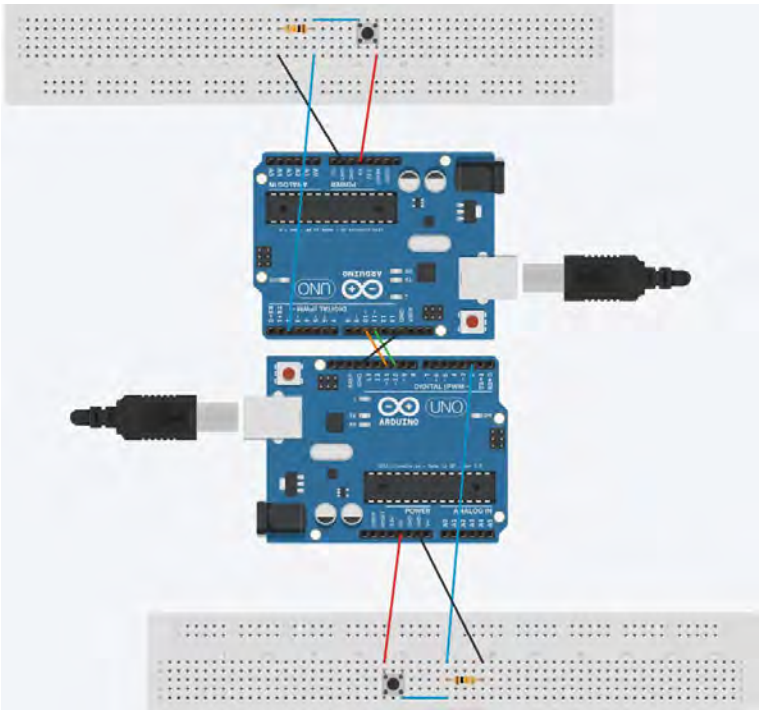


Figura 19.9: Sugestão de montagem do exercício do capítulo 15

Sugestão de programação

```
#include <SoftwareSerial.h>

SoftwareSerial outro_arduino(10,11); // RX TX

int recebido = 0;
int led = 13;
int botao = 2;
int press = 0;

void setup() {
    pinMode(led,OUTPUT);
    pinMode(botao,INPUT);
    Serial.begin(9600);
    outro_arduino.begin(4800);
}
void loop() {
    press = digitalRead(botao);
    if(press == HIGH) {
```

```
        digitalWrite(led,HIGH);
        outro_arduino.write("1");
    } else {
        digitalWrite(led,LOW);
        outro_arduino.write("0");
    }
    if(Serial.available() > 0)  {
        outro_arduino.write(Serial.read());
    }
    if(outro_arduino.available() > 0) {
        recebido = outro_arduino.read();
        Serial.println(recebido);
        if(recebido == 49) {
            digitalWrite(led,HIGH);
            delay(10);
        } else {
            digitalWrite(led,LOW);
            delay(10);
        }
    }
}
```

19.9 EXERCÍCIO DO CAPÍTULO 16. BIBLIOTECAS ADICIONAIS

Una a experiência em que usamos um potenciômetro com a que controlamos um servo motor, e faça com que o potenciômetro passe a controlar a posição do servo motor. Quando você mover o potenciômetro para um lado, o motor deve mover-se para o mesmo lado.

Sugestão de montagem

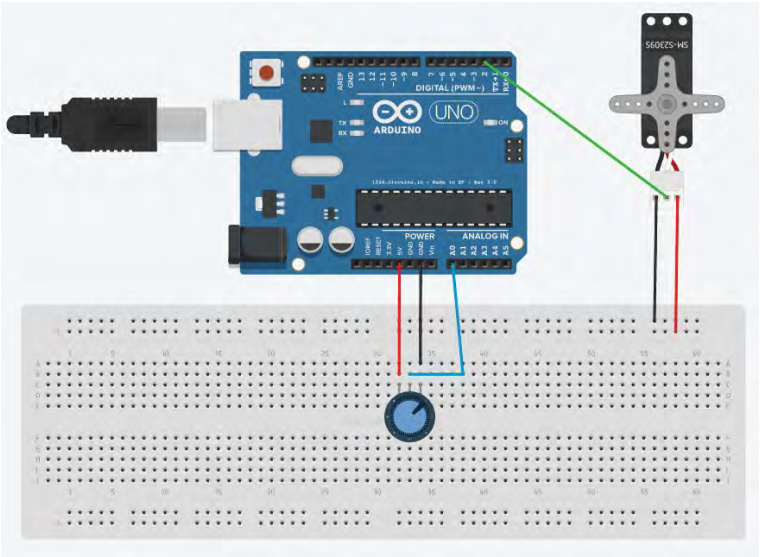


Figura 19.10: Sugestão de montagem do exercício do capítulo 16

Sugestão de programação

```
#include <Servo.h>

Servo srv;
int pos = 0;

void setup() {
    Serial.begin(9600);
    srv.attach(2);
}

void loop() {
    // De 0 a 1024 do potenciômetro para
    // 0 a 180 graus do servo motor
    pos = map(analogRead(A0), 0, 1024, 0, 180);
    srv.write(pos);
}
```

19.10 EXERCÍCIO DO CAPÍTULO 17. ACIONANDO MOTORES DC

Junte a experiência de controle de um motor DC, tanto por relê quanto por transistor, com a experiência de controlar LEDs e a do botão. Faça um programa que inicie com um LED vermelho aceso e, quando o botão for pressionado, o motor comece a girar, o LED vermelho apague e um LED verde acenda. Quando o botão for solto, o motor deve parar de girar, o LED verde deve apagar e o LED vermelho deve acender.

Sugestão de montagem

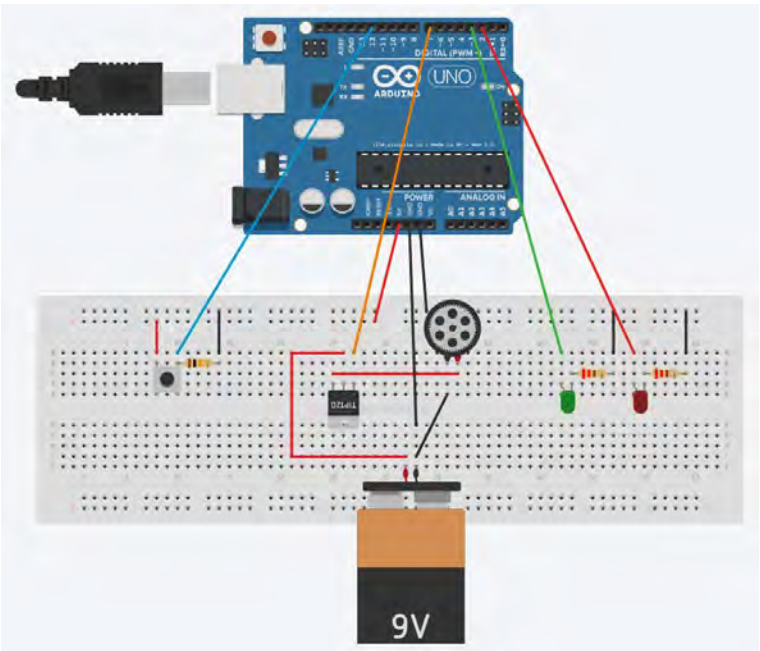


Figura 19.11: Sugestão de montagem do exercício do capítulo 17)

Sugestão de programação

```
int rele = 7;
int led_r = 2;
int led_b = 3;
int botao = 12;
int press = 0;
```

```
void setup() {
    pinMode(rele,OUTPUT);
    pinMode(led_r,OUTPUT);
    pinMode(led_b,OUTPUT);
    pinMode(botao,INPUT);
}

void loop() {
    press = digitalRead(botao);
    if(press == HIGH) {
        press = LOW;
        digitalWrite(led_r,LOW);
        digitalWrite(led_b,HIGH);
        digitalWrite(rele,HIGH);
    } else {
        digitalWrite(led_r, HIGH);
        digitalWrite(led_b, LOW);
        digitalWrite(rele,LOW);
    }
}
```