Arquitetura de Computadores I (LAB)

2025/2

Nicolas Ramos Carreira

Sumário

Capítulo 1

Intuito

O intuito deste documento é explicar sobre o que será feito nas aulas de laboratório da disciplina de Arquitetura de Computadores I. Neste documento provavelmente estarão explicações de componentes elétricos e alguns conceitos físicos, então se prepare!

Capítulo 2

Fundamentos Fisicos da Computação

2.1 Carga elétrica

- A carga elétrica faz com que a matéria experimente uma força de atração ou repulsão
- A medida da carga elétrica é Coulombs (C)
- Para fazer uma analogia, podemos dizer que a a carga elétrica seria a água

2.2 Corrente elétrica

- A corrente elétrica é o movimento da carga elétrica (ou seja, representa uma taxa que mede a intensidade do fluxo elétrico)
- A medida da corrente elétrica é feita em Amperes (A)
- Em equações, costuma-se utilizar: $1A = 1C/s$ (1 Amper = 1 Coulumb/-segundo)
- Para fazer uma analogia, podemos dizer que a corrente elétrica seria: FLUXO DA ÁGUA EM UM PONTO/segundo

2.3 Tensão(voltagem)

- A tensão influencia a taxa com a qual uma carga flui através de um circuito. É a diferença de potencial elétrico entre dois pontos. Só iremos ter corrente elétrica se houver diferença de potencial.
- A tensão é medida em Volts (V)

- Para fazer uma analogia, Imagine uma fonte de energia (uma BOMBA D'ÁGUA).Essa bomba aumenta a PRESSÃO DA ÁGUA e aumenta a velocidade do fluxo. Essa pressão é a TENSÃO

2.4 Resistência

- É uma medida da dificuldade da corrente de fluir por um condutor
- É medida em Ohms (Ω)
- Uma analogia possível seria comparar com o diâmetro do cano

2.5 Analogia(para um entendimento claro)

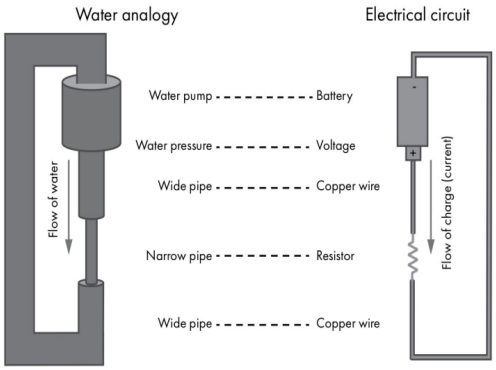


Figura 2.1: Water pump = Bomba d'água; Narrow pipe = tubo estreito

2.6 Lei de Ohm

- A lei de Ohm vai nos dizer que a corrente que flui entre dois pontos é igual à voltagem dividida pela resistência: I = V/R
- Seguindo com a analogia, se você aumentar a pressão da água (V), a quantidade de água que flui (I) também aumenta, desde que o entupimento (R) continue o mesmo. Por outro lado, Se o entupimento (R) for maior, a quantidade de água que flui (I) diminui, mesmo que a pressão (V) continue a mesma.
- CUIDADO AC X DC: A DC representa a corrente continua, que é quando A corrente flui em apenas uma direção. A maioria das pilhas e baterias usa corrente contínua (guarde esta informação). Por outro lado, temos a AC que representa a corrente alternada, onde os polos se invertem várias

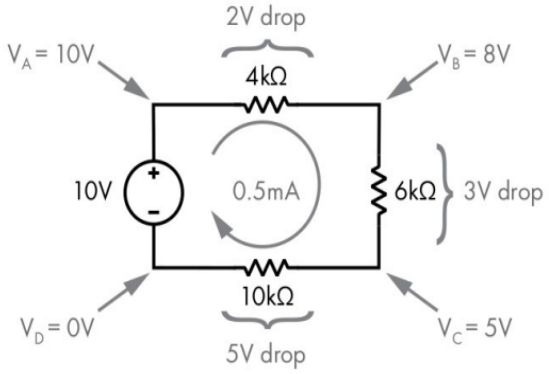
vezes para que a corrente continue circulando. A energia que chega nas tomadas das nossas casas é AC. É importante ter cuidado com isso pois no laboratório, durante as atividades, utilizaremos componentes que utilizam a CORRENTE CONTINUA, então não inverta o polo do capacitor na hora de encaixá-lo, pois como vimos, na corrente continua os polos não se invertem.

2.7 Circuitos elétricos (conceitos básicos)

- É um conjunto de vários componentes (veremos sobre esses componentes mais afrente) conectados de forma a permitir que a corrente possa fluir em um loop, a partir da fonte de energia, passando pelos elementos do circuito, e de volta à fonte de energia
- Um circuito começa e termina no mesmo lugar.
- Em circuitos DC, o terra (GND) é considerado 0V e serve como um ponto de referência. Pode ser a terra propriamente dita, ou o polo negativo de uma bateria, por exemplo.

2.8 Lei de Kirchhoff

- A soma das voltagens em um circuito é 0
- Isso significa que se uma bateria fornece 9V para um circuito, os vários componentes do circuito devem "consumir" esses 9V. No caso da imagem abaixo, os resistores fazem esse papel



Acima, os resistores irão consumir a voltagem durante o circuito, até que seja 0V. Mas afinal, como 4kΩ vira 2V, por exemplo? Para isso, usamos a fórmula V = I X R (proveniente da fórmula I = V/R). Assim teremos que:

- Corrente (I): O circuito tem uma corrente de 0.5 mA (0.0005 Amperes). corrente é a mesma em todos os resistores nesse tipo de circuito.
- Resistência (R): A resistência do resistor em questão é de 4k Ohms (4.000 Ohms).

Com isso, bastará fazer: V = (0.0005 A)×(4000 Ohms), que será igual a 2V. Assim, faremos isso com todos os resistores, resultando em: 2V (o primeiro que fizemos), 3V e 5V. Ao somar a voltagem de todos eles, teremos 10V. Ou seja, os resistores irão consumir a voltagem por completo, atendendo a Lei de Kirchhoff.

Um detalhe é que se tivermos um circuito onde queremos descobrir qual resistor temos que colocar para que ele consuma nossa voltagem, nós usamos a fórmula R = V/I, onde vamos substituir em V quantos volts queremos que esse resistor "gaste". Em I, teremos a corrente que vai passar por esse resistor. Em circuitos simples em série, a corrente é a mesma em todos os componentes.

Capítulo 3

Cicuito real

3.1 O que é Arduíno?

Arduino é uma plataforma para auxiliar com projetos de eletronica e programação. A ideia é que você tivesse uma coisa barata, funcional (que desse para instalar coisas de verdade) e que fosse fácil de aprender. Essa plataforma é quase todo baseado em uma plaquinha. Veja a foto abaixo:

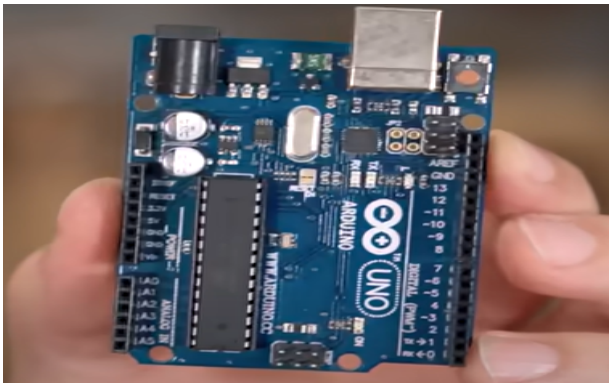


Figura 3.1: Arduino UNO R3

Essa plaquinha tem microcontroladores (a parte preta larga da placa que fica quase no meio) que funciona mais ou menos como um computadorzinho bem pequeno que é capaz de processar e armazenar coisas e também tem uma linguagem de programação, onde você consegue escrever coisas, criar sisteminhas que você coloca dentro e faz com que a placa te obedeça.

E por que o arduino deu tão certo? Pois foi criado com a ideia de hardware livre. Qualquer um pode olhar como funciona seu circuito, comprar os componentes, montar seu próprio arduino e se quiser até vender seu próprio arduino. Isso faz com que o arduino seja barato, sendo assim usado amplamente em pro-

jetos, o que contribui para a disponibilização de vários materiais de arduino e com que várias pessoas possam ter contato com ele.

Uma observação é que se pode utilizar o arduíno pela internet a partir do site TinkerCad. Se você comprar a placa, você precisa conectar ela via USB ao computador e baixar o softwere Arduino IDE, onde você bota o código que você vai fazer para enviar para a placa.

Se você quiser ligar a placa sem o computador (isso acontece depois que você ja colocou o programa que voc e fez nela), você conecta um adptador de bateria de 9V na placa por meio da entrada redonda e conecta o adaptador a bateria, aí você consegue ligar ela assim. Veja as duas entradas abaixo:

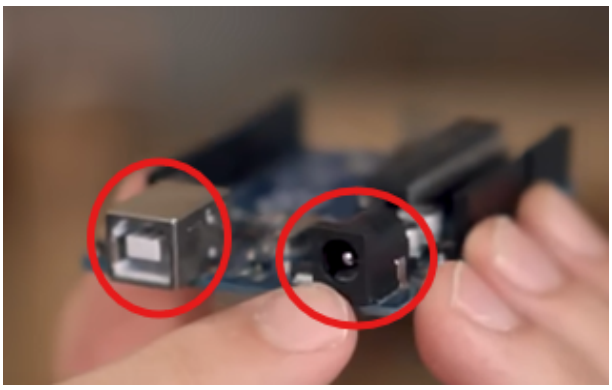
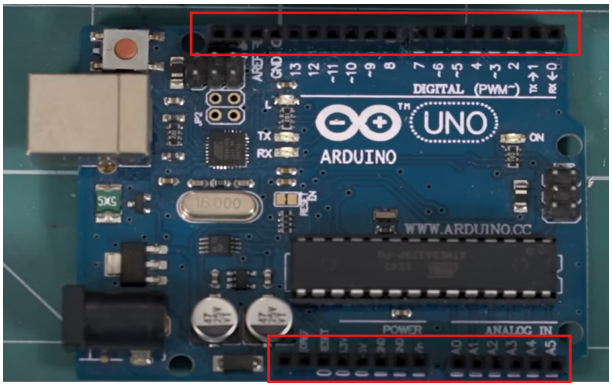


Figura 3.2: Entrada USB e entrada redonda (da esquerda para a direita)

O arduíno ainda possui as portas de comunicação (circulado em vermelho na imagem abaixo), que serve para espetar os fios, componentes e até outras placas (é como se fosse aquela parte lateral do computador onde tem entradas usb e de rede, por exemplo). Veja:



Essas conexões que podemos fazer ao espetar os componentes é chamada de PINOS, sendo que podemos ter pinos de entrada e de saída. Mas como

funciona isso? Suponha que você queira instalar um sensor de presença no arduíno que quando alguém chega perto, você acende um LED. O sensor irá fornecer informações para o arduino, então você coloca ele na entrada. Os pinos de saída, você usará para acender o led, pois está saindo informações do arduíno (você que define os pinos de entrada e saída no código).

Alguns desses pinos tem a sigla GND, o que significa que é o TERRA.

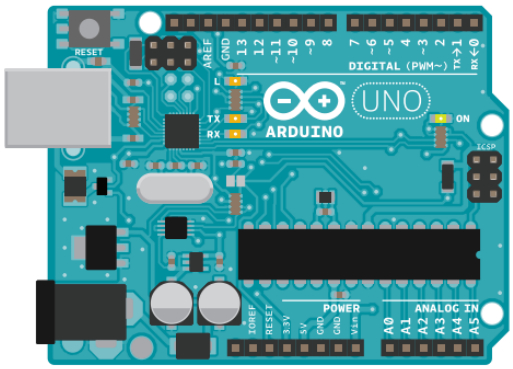
Os pinos que ficam na parte "DIGITAL"são binários, só aceitam 0 e 1. Os pinos que ficam em ANALOG"são mais espertas e identificam valores intermediários.

Na parte de POWER (que é a que a gente vai usar bastante nas aulas de lab) nós temos algumas indicações de Volts, o que significa que o arduino pode ser usado para alimentar componentes

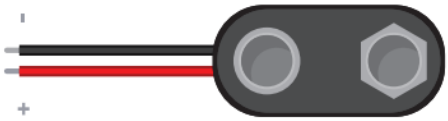
3.2 Principais componentes eletrônicos e ferramentas

Aqui, falaremos sobre os principais componentes e ferramentas que são utilizados em projetos arduínos. Os componentes eletronicos basicamente controlam os movimentos dos eletrons. É importante ressaltar que será dado apenas um panorama geral. Iremos aprofundar conforme formos estudando os sisteminhas. Vamos a eles:

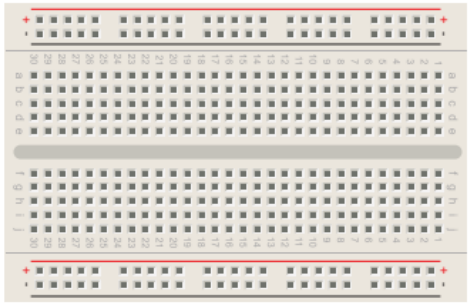
- Arduino Uno - É uma placa de desenvolvimento do microcontrolador que será o centro dos nossos projetos. É a peça central que orquestra e controla o funcionamento de todos os outros componentes.



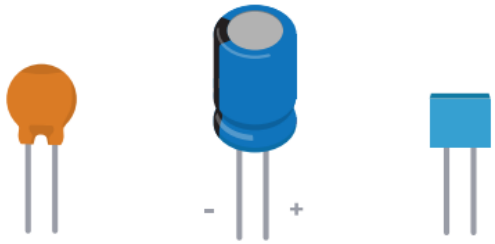
- Snap de bateria - É o cabo que serve para conectar uma bateria (geralmente de 9V) a um circuito, permitindo que a energia dela chegue aos componentes. (não veremos tanto por agora pois iremos conectar via usb direto no computador)



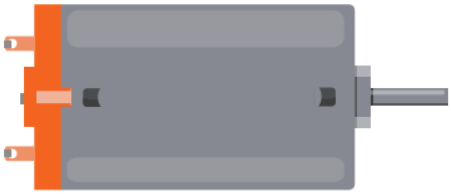
- Protoboard - É uma placa com pequenos furos que permite montar circuitos eletrônicos temporariamente, sem precisar soldar nada. Você encaixa os componentes e os fios nos furos, e pode montar, desmontar e remontar o circuito quantas vezes quiser. Os furos são interligados internamente, formando "trilhos"que levam a energia de um ponto a outro. A região do meio é divida em colunas verticais (isso com a placa deitada, como na imagem abaixo), ou seja, os 5 buraquinhos da coluna estão conectados. Na região das bordas os buraquinhos estão conectados de forma horizontal.



- Capacitores - Um capacitor armazena energia elétrica por um curto período de tempo e pode liberá-la rapidamente. Ao usar capacitores, é importante ter atenção, pois eles são polarizados (tem o polo negativo e positivo) e se conectarmos os polos no lugar errado ele pode explodir. A perna do polo negativo é destacado com uma faixa. Existem os capacitores ceramicos (o da esquerda na imagem abaixo), eletroliticos (o do meio na imagem abaixo) e de poliester (o da direita na imagem abaixo).



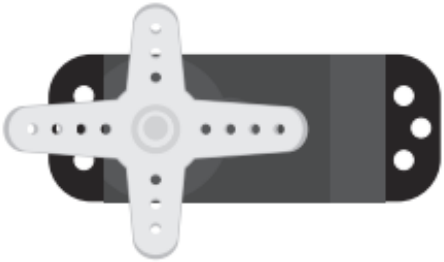
- Motor DC - Um pequeno motor elétrico que converte energia elétrica em movimento (rotação). Por exemplo, em um carrinho de controle remoto, é ele quem faz as rodas se moverem.



- Diodo - Permite que a corrente elétrica flua em apenas uma direção. Ele bloqueia o fluxo no sentido oposto. O lado negativo do diodo é destacado por uma faixa.

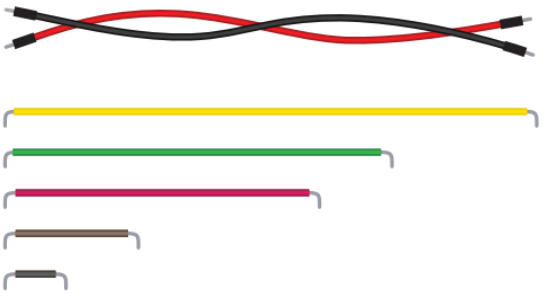
- Fotoresistor - É um componente que muda a sua resistencia conforme a quantidade de luz que é incidida sobre ele. Quanto mais luz, menos resistencia.

- Servo Motor - É um tipo de motor giratório que só pode rotacionar em 180 graus. É controlado pelo envio de pulsos eletricos do Arduino. Esses impulsos informam ao motor para qual posição ele deve se mover

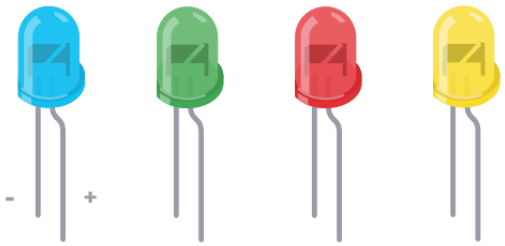


- Sensor de temperatura- Altera sua saída de tensão dependendo da temperatura do componente. As pernas externas conectam-se à alimentação e ao aterramento. A tensão no pino central muda conforme esquenta ou esfria.

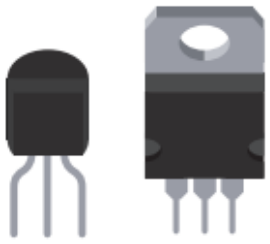
- Fios de ligação - São os "cabos"que você usa para conectar os componentes na protoboard. Eles criam os caminhos para a eletricidade viajar de um componente para outro na protoboard.



- LEDs - É basicamente uma lâmpada. Irá emitir luz quando a corrente elétrica passa por ela. Para identificar o polo positivo pode: identificar a perna mais comprida do LED; Usar o multimetro ou identificar a parte mais gordinha do plastico do LED

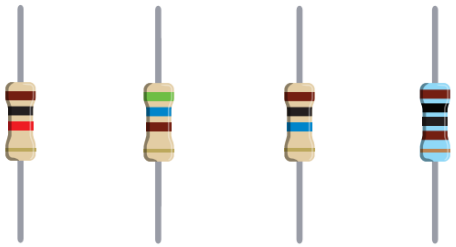


- Transistor - Funciona como um interruptor eletronico, ou seja, conseguimos ligar e desligar coisas eletronicamente. Tem 3 pernas, onde duas delas servem para a corrente fluir (o coletor e o emissor) e a terceira faz o controle (é a base).



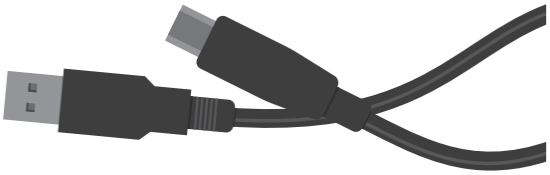
- Botão - Um tipo de interruptor que só funciona enquanto é pressionado. Quando você o solta, ele retorna à sua posição original, interrompendo o circuito.

- Resistores - Ele controla e diminui o fluxo de eletricidade, como uma torneira que reduz a passagem da água. É essencial para proteger componentes como os LEDs de receberem muita corrente e queimarem. Os resistores possuem faixas coloridas que servem para indicar quantos ohms tem o resistor (qual a resitencia dele). Inclusive: site para você ver quantos ohms tem um resistor a partir de sua sequencia de cores.



- Potenciometros - É basicamente um resistor que você consegue variar a resistência.

- USB – É o cabo que conectaremos do computador para a nossa placa de arduíno para fornecer energia para ela e programar nela futuramente



- Multimetro -

3.3 Circuitos básicos (para entendimento)

Capítulo 4

Programação com o Arduíno

4.1 Introdução

Quando você escreve um código em arduíno, ele sempre segue a seguinte estrutura:

```
void setup() {

        // aqui ficam as configuracoes que rodam 1 vez so
}

void loop() {
        // aqui fica o programa que repete para sempre
}
```

- setup(): roda apenas uma vez quando a placa é ligada ou reiniciada. Usada para configurações iniciais (ex: definir pino como saída ou entrada).
- loop(): roda continuamente em um ciclo infinito. É aqui que o Arduino executa o programa principal.

Veja um exemplo a respeito disso:

```
void setup() {
        pinMode(13, OUTPUT); // Configura o pino 13 como saida
}
```

```
void loop() {
        digitalWrite(13, HIGH); // Liga LED
        delay(1000);            // Espera 1s
        digitalWrite(13, LOW);  // Desliga LED
        delay(1000);
}
```

4.2 Pinos

Entender bem os pinos do Arduino é essencial porque eles são a forma de comunicação entre o microcontrolador e o mundo externo (sensores, LEDs, motores, etc.). O Arduino UNO tem 14 pinos digitais e 6 pinos analogicos. Veremos sobre eles a seguir.

4.2.1 Pinos digitais (0 a 13)

- Podem ser configurados como entrada (INPUT) ou saída (OUTPUT) com pinMode. Exemplo: pinMode(pino, OUTPUT/INPUT).
- Como saída (OUTPUT) → podem enviar sinais HIGH (5V) ou LOW (0V). Exemplo: acender um LED → digitalWrite(13, HIGH);
- Como entrada (INPUT) → Podem ler se há tensão (HIGH = 1) ou não (LOW = 0) em um sensor ou botão. Exemplo: ler botão → int estado = digitalRead(2);

Uma observação é que os pinos 3, 5, 6, 9, 10 e 11 têm suporte a PWM (tem um “ ” ao lado do número). Isso permite simular sinais analógicos (ex: controlar brilho de LED ou velocidade de motor).

4.2.2 Pinos analógicos (A0 a A5)

- Usados principalmente como entrada
- Leem sinais analógicos (0 a 5V) e convertem para valores digitais de 0 a 1023 através do conversor ADC (10bits). Exemplo: analogRead(A0); → retorna algo entre 0 e 1023.
- Muito usados com sensores que dão saída variável, como sensor de temperatura, LDR (fotoresistor), potenciômetro etc.
-

4.2.3 Pinos de alimentação

- 5V → saída de 5V para alimentar sensores e módulos.
- 3.3V → saída de 3.3V (para componentes que não suportam 5V).
- 3.3V → saída de 3.3V (para componentes que não suportam 5V).
- Vin → entrada de tensão quando você alimenta o Arduino com fonte externa (7 a 12V).

4.3 Programando no arduíno

O arduíno utiliza basicamente a linguagem C, então muitas tópicos nós já sabemos, então vou passar de forma rapida em alguns deles.

4.3.1 Variaveis

Conceito

Como bem sabemos, variaveis são espaços alocados na memória onde armazenamos valores. No Arduino, você usa variáveis para armazenar dados que seu programa precisa, como a leitura de um sensor, o tempo de um atraso, o estado de um LED (ligado ou desligado), ou o valor de uma porta analógica.

As variaveis como também sabemos, podem assumir um tipo e cada tipo de variável tem um tamanho específico (quantos bytes ela ocupa na memória) e um intervalo de valores que pode armazenar. Dessa forma, os principais tipos do arduino e a suas utilidades são:

- int (inteiro): É o tipo mais comum. Armazena números inteiros, tanto positivos quanto negativos. É ideal para contadores ou para armazenar o número de um pino digital. Exemplo: int numeroDeCliques = 0;
- float (ponto flutuante): Usado para números que têm casas decimais. É perfeito para leituras de sensores que fornecem valores fracionários, como temperatura ou umidade. Exemplo: float temperatura = 25.5;
- boolean (booleano): Armazena apenas dois valores possíveis: true (verdadeiro) ou false (falso). É muito útil para armazenar o estado de algo, como "ligado"ou "desligado". Exemplo: boolean ledEstaLigado = false;

Declaração e atribuição

A sintaxe para declarar e atribuir valores a variaveis em arduino segue a mesma da linguagem C:

$$\text{tipo nomeDaVariavel} = \text{valor};$$

Você pode declarar a variável e atribuir um valor depois, se preferir:

tipo nomeDaVariavel;
nomeDaVariavel = valor;

Exemplo

4.3.2 Operadores

Sabemos que os operadores são símbolos que realizam operações em valores e variáveis. Pense neles como as ferramentas básicas que você usa para fazer cálculos, comparar valores e manipular dados no seu código. No arduino os operadores são essenciais para quase tudo o que você faz em um programa, desde somar dois números até verificar se um sensor atingiu um certo valor. Falaremos então sobre cada tipo de operador

Operador aritmético

Esses são para operações matemáticas básicas.

- (Adição): Soma dois valores. Exemplo: int soma = 5 + 3; (soma será 8).
- (Subtração): Subtrai um valor do outro. Exemplo: int diferenca = 10 - 4; (diferença será 6).
- (Multiplicação): Multiplica dois valores. Exemplo: int produto = 6 * 2; (produto será 12).
- (Divisão): Divide um valor pelo outro. Exemplo: int quociente = 15 / 3; (quociente será 5).
- (Módulo): Retorna o resto de uma divisão. É útil para verificar se um número é par ou ímpar. Exemplo: int resto = 7 porct 2; (resto será 1).

Operador de atribuição

Usados para atribuir um valor a uma variável.

- = (Atribuição simples): Atribui o valor do lado direito à variável do lado esquerdo. Exemplo: int ledPin = 13;
- Atribuições combinadas: Combinam uma operação com a atribuição, tornando o código mais curto.
  - +=: Adição e atribuição. Exemplo: x += 5; é o mesmo que x = x + 5;
  - -= : Subtração e atribuição. Exemplo: x -= 2; é o mesmo que x = x - 2;
  - *=: Multiplicação e atribuição. Exemplo: x *= 3; é o mesmo que x = x * 3;
  - /=: Divisão e atribuição. Exemplo: x /= 4; é o mesmo que x = x / 4;

Operador de comparação

Usados para comparar dois valores. O resultado de uma comparação é sempre um valor booleano: true (verdadeiro) ou false (falso).

- == (Igual a): Verifica se os valores são iguais. Atenção: use dois sinais de igual. Um único = é para atribuição. Exemplo: if (leitura == 100) ...
- != (Diferente de): Verifica se os valores são diferentes. Exemplo: if (estadoBotao != HIGH) ...
- > (Maior que): Exemplo: if (temperatura > 30) ...
- < (Menor que): Exemplo: if (luz < 50) ...
- >= (Maior ou igual a): Exemplo: if (contador >= 10) ...
- <= (Menor ou igual a): Exemplo: if (tempo <= 500) ...

Operador lógico

Usados para combinar ou inverter resultados de expressões lógicas (as que retornam true ou false, como comparações por exemplp).

- && (E lógico): Retorna true se ambas as condições forem verdadeiras. Exemplo: if (leitura > 100 && leitura < 500) ...
- || (Ou lógico): Retorna true se pelo menos uma das condições for verdadeira. Exemplo: if (pin13 == LOW || pin12 == LOW) ...
- ! (Não lógico): Inverte o resultado. Se a condição for true, ele a torna false, e vice-versa. Exemplo: if (!sensorAtivo) ... (Isso é o mesmo que if (sensorAtivo == false) ...).

Exemplos

4.3.3 Operações condicionais

Como sabemos, condicionais são estruturas de controle que permitem que o seu programa tome decisões. Elas verificam se uma ou mais condições são verdadeiras ou falsas e, com base nisso, executam um bloco de código específico. No Arduino, isso é fundamental para fazer seu projeto interagir com o ambiente, como acender um LED quando um botão é pressionado, ou ligar um motor quando a temperatura sobe.

if

O if é a estrutura condicional mais básica e usada. Ela executa um bloco de código apenas se a condição for verdadeira. Sua sintaxe é:

```
if (condicao) {
        codigo
}
```

OBS: condicao é uma expressão que retorna um valor booleano (true ou false). Geralmente, você usa os operadores de comparação (==, !=, <, >) aqui.

if..else

O if...else oferece uma alternativa: ele executa um bloco de código se a condição for verdadeira, e um bloco diferente se a condição for falsa. Sua sintaxe é:

```
if (condicao) {
        codigo condicao verdadeira
}
else {
        codigo condicao falsa
}
```

if..else if..else

Essa estrutura é usada para testar múltiplas condições em sequência. A primeira condição que for verdadeira terá seu bloco de código executado, e o resto da estrutura será ignorado. Se nenhuma condição for verdadeira, o bloco else final será executado. Sua sintaxe é:

```
if (condicao1) {
        Codigo se a condicaoo1 for verdadeira
}
else if (condicao2) {
        Codigo se a condicaoo2 for verdadeira (e a 1 for falsa)
}
else {
        Codigo se nenhuma das condicoes anteriores for verdadeira
}
```

Switch..case

4.3.4 Estruturas de repetição

Estruturas de repetição (ou "loops") são blocos de código que permitem executar um conjunto de instruções repetidamente, até que uma condição específica seja atendida. Elas são essenciais para automatizar tarefas que precisam ser feitas várias vezes, como ler um sensor continuamente, piscar um LED por um certo número de vezes, ou percorrer um array de dados.

No Arduino, o próprio loop() principal é uma estrutura de repetição infinita que se executa indefinidamente, mas dentro dele, você pode usar outras estruturas para criar repetições mais controladas.

While

O loop while executa um bloco de código enquanto uma condição for verdadeira. A condição é verificada antes de cada execução do bloco. Se a condição começar como falsa, o código dentro do while nunca será executado. Sua sintaxe é:

```
while (condicao) {

        codigo a ser executado repetidamente
}
```

Importante: Você deve garantir que algo dentro do loop mude a condição em algum momento, caso contrário, o loop se tornará infinito e travará o seu programa.

do..while

O loop do...while é semelhante ao while, mas com uma diferença crucial: o código dentro do loop é executado pelo menos uma vez antes que a condição seja verificada. Sua sintaxe é

```
do {
        codigo a ser executado pelo menos uma vez
} while (condicao);
```

for

O loop for é ideal quando você sabe exatamente quantas vezes a repetição precisa ocorrer. Ele é mais compacto e geralmente mais fácil de ler para repetições com um número fixo de vezes. Sua sintaxe é

```
for (inicializacao; condicao; incremento) {
        codigo a ser executado repetidamente
}
```

Onde:

- inicializacao: Executada apenas uma vez, no início do loop. Geralmente é onde você declara e inicializa uma variável contadora.
- condicao: Testada antes de cada iteração. Se for verdadeira, o loop continua. Se for falsa, o loop é encerrado.
- incremento: Executado após cada iteração. Geralmente é onde você aumenta ou diminui a variável contadora.

4.3.5 Monitor serial

Conceito

O Monitor Serial é uma ferramenta essencial na IDE (Ambiente de Desenvolvimento Integrado) do Arduino. Pense nele como uma "janela de comunicação"que permite que o seu computador e a placa Arduino troquem dados. Ele é usado principalmente para:

- Depuração (Debugging): É a forma mais fácil de "ver"o que está acontecendo dentro do seu programa. Você pode imprimir o valor de variáveis, a leitura de sensores, ou mensagens de status para entender se o código está se comportando como esperado.
- Visualizar Dados: Se o seu projeto estiver coletando dados (como temperatura, umidade, ou leituras de sensores de distância), você pode usar o Monitor Serial para ver esses valores em tempo real no seu computador.
- Enviar Comandos: Você pode digitar comandos no Monitor Serial e enviá-los para o Arduino. Isso permite controlar seu projeto a partir do computador, como ligar e desligar um LED digitando "ligar"ou "desligar".

Ou seja, o Serial do Arduino é como o terminal em um programa de linha de comando (CLI) que nós fazemos em em C, por exemplo.

Como usar?

Para usar o Monitor Serial, você precisa de duas coisas:

1. Configurar a comunicação serial no seu código.
2. Abrir a janela do Monitor Serial na IDE do Arduino.

Passo 1: Configurando o Código

Você precisa de apenas duas linhas de código para começar a usar o Monitor Serial.

- Serial.begin(velocidade);

  - Esta linha é colocada no bloco setup().
  - Ela inicializa a comunicação serial.
  - velocidade é a taxa de transmissão de dados em bits por segundo (baud rate). O valor mais comum é 9600, mas é crucial que a velocidade configurada no código seja a mesma que a escolhida no Monitor Serial.

- Serial.print(); e Serial.println();

  - Estas linhas são usadas para enviar dados do Arduino para o computador.
  - Serial.print() imprime o dado e mantém o cursor na mesma linha.
  - Serial.println() imprime o dado e move o cursor para a próxima linha, o que é ótimo para exibir cada informação em uma nova linha.
  - Você pode imprimir texto, números, o valor de variáveis, etc., dentro dos parênteses.

Passo 2: Abrindo e usando a janela

Depois de carregar o código na placa, siga estes passos:

1. Clique no ícone de lupa no canto superior direito da IDE do Arduino, ou use o atalho Ctrl + Shift + M.
2. Uma nova janela se abrirá.
3. No canto inferior direito, você verá um menu suspenso para a taxa de transmissão. Certifique-se de que ele esteja definido para o mesmo valor que você usou no Serial.begin() (neste caso, 9600).
4. O Monitor Serial começará a exibir a mensagem e os valores que o seu código está enviando.

Dicas e considerações importantes

- Taxa de Transmissão (Baud Rate): Se a velocidade do código e a do Monitor Serial não coincidirem, você verá caracteres estranhos ou lixo eletrônico.
- Apenas Texto: O Monitor Serial não exibe gráficos ou imagens. Ele é feito para comunicação em texto.
- Alternativas: Existem ferramentas mais avançadas, como o Serial Plotter na IDE do Arduino, que transformam dados numéricos em gráficos, o que é excelente para visualizar leituras de sensores ao longo do tempo. Você pode acessá-lo no menu Ferramentas > Serial Plotter.

Exemplo

4.3.6 Funções

Conceito

Funções são blocos de código que executam uma tarefa específica e podem ser chamados a qualquer momento no seu programa. Elas ajudam a organizar o código, torná-lo mais legível, reutilizável e fácil de dar manutenção.

Pense em uma função como uma "caixa"que recebe uma ou mais informações (chamadas de parâmetros), faz um trabalho com elas e, opcionalmente, retorna um resultado. Existem dois tipos principais de funções:

1. Funções nativas: Aquelas que já vêm com a linguagem do Arduino, como digitalWrite(), analogRead(), delay(), etc.
2. Funções personalizadas: Aquelas que você cria para o seu próprio projeto.

Por que usar

Usamos funções no arduino (e de modo geral) porque:

- Organização: Quebra um programa grande em partes menores e mais fáceis de entender.
- Reutilização: Se você precisa piscar um LED em diferentes partes do seu código, você pode criar uma função piscarLed() e chamá-la quantas vezes quiser, em vez de repetir as mesmas linhas de código.
- Legibilidade: Dar um nome descritivo a uma função (lerTemperatura()) torna o código mais claro do que ter um monte de linhas misturadas.

Como criar

Para criar uma função, sua sintaxe é:

```
tipoDeRetorno nomeDaFuncao(parametro1, parametro2, ...) {

        codigo que a funcao executa
        return valor; —> se retornar algo
}
```

Onde:

- tipoDeRetorno: Define o tipo de dado que a função vai retornar (ex: int, float, boolean). Se a função não retornar nada, use void.
- nomeDaFuncao: Um nome claro e descritivo para a função.

- parametros: Uma lista de variáveis que a função espera receber. São opcionais.
- return: O comando para enviar um valor de volta para onde a função foi chamada. Só é usado se o tipoDeRetorno não for void.

Um detalhe é que as funções que criamos ficam acima da parte de void setup()

Exemplo

4.3.7 Bibliotecas

Bibliotecas são coleções de código pré-escrito que estendem as funcionalidades do seu Arduino. Elas contêm funções e classes prontas para uso, permitindo que você adicione recursos complexos ao seu projeto sem ter que escrever todo o código do zero.

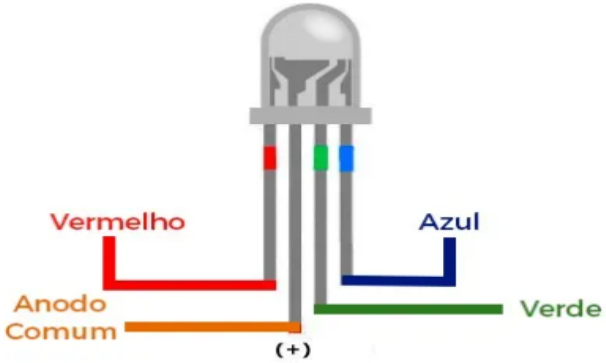
4.3.8 Alguns exemplos

4.4 Uso de sensores e Saídas Analógicas

4.4.1 LED RGB e Fotoresistores

Uso do LED RGB

Um LED RGB é um único componente que contém três LEDs separados: um vermelho (R), um verde (G) e um azul (B). Veja a imagem abaixo:



Ao controlar a intensidade de cada uma dessas cores primárias, você pode misturá-las para criar uma gama enorme de cores. O controle da intensidade é feito usando a técnica de PWM (Modulação por Largura de Pulso, que falaremos em breve) em cada pino de cor.

Uso de fotorresistores

Um fotoresistor, também chamado de LDR (Light-Dependent Resistor), é um tipo de sensor que muda sua resistência de acordo com a quantidade de luz que incide sobre ele.

4.4.2 Modulação por Largura de Pulso (PWM)

PWM

É uma técnica usada para simular saídas analógicas a partir de saídas digitais. O Arduino não consegue fornecer uma voltagem variável (ele só fornece 0V ou 5V), mas com o PWM ele pode ligar e desligar a saída muito rapidamente. O olho humano percebe esse "pisca-pisca"rápido como se a luminosidade fosse menor.

Ciclo de trabalho (Duty cicle)

É a porcentagem de tempo em que o sinal PWM fica em nível alto (HIGH).

- Um ciclo de trabalho de 0% significa que o sinal está sempre em LOW, e o LED fica apagado.
- Um ciclo de trabalho de 50% significa que o sinal fica em HIGH por metade do tempo e em LOW pela outra metade, o que resulta em um brilho médio para o LED.
- Um ciclo de trabalho de 100% significa que o sinal está sempre em HIGH, e o LED fica com o brilho máximo.

4.4.3 Funções analogRead() e analogWrite()

Já chegamos a falar um pouco dessas funções anteriormente, mas vamos revê-las porque será importante neste momento:

- analogRead(pino): Lê o valor de uma porta analógica (de A0 a A5) e retorna um número inteiro de 0 a 1023. Esse valor é proporcional à voltagem lida (0 = 0V, 1023 = 5V). É a função usada para ler dados de sensores como o fotoresistor
- analogWrite(pino, valor): Escreve um valor PWM em uma porta digital que suporte essa função (normalmente identificadas com um ~). O valor pode ser um número de 0 a 255.
  - analogWrite(pino, 0) equivale a um ciclo de trabalho de 0%, deixando o LED apagado.
  - analogWrite(pino, 127) equivale a um ciclo de trabalho de 50%, deixando o LED com brilho médio.
  - analogWrite(pino, 255) equivale a um ciclo de trabalho de 100%, deixando o LED com brilho máximo.