Robótica Elementar

Ensino Prático

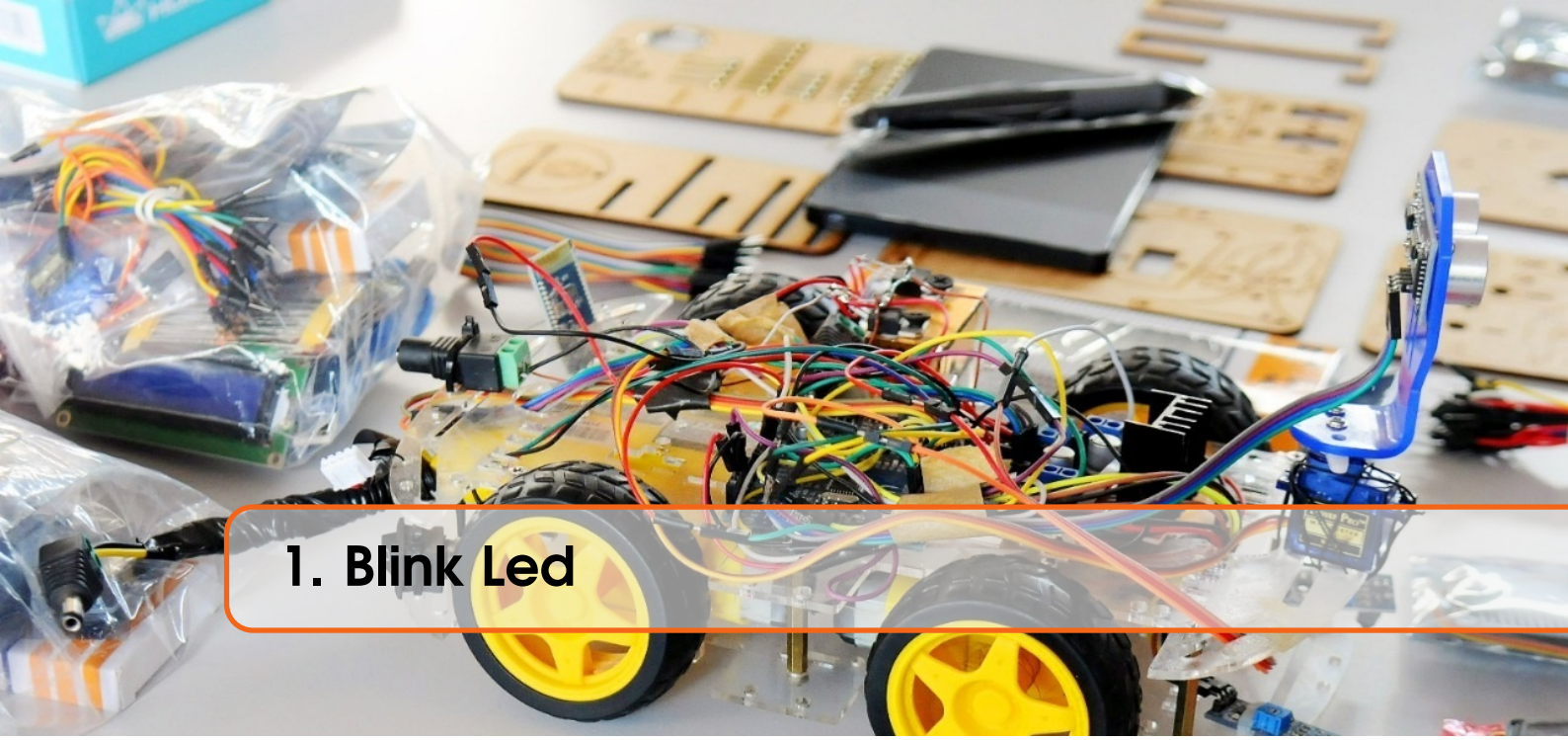
Aratã Saraiva

EDUCAÇÃO INFANTIL

ARATASARAIVA@GMAIL.COM

Primeira impressão, Fevereiro de 2022

Módulo 1



1. Blink Led

1.1 Introdução

Nesta prática você vai aprender a executar seu primeiro projeto com Arduino. Assim como em toda linguagem de programação, o Arduino também possui o Hello World, contudo, o 'Olá Mundo' é executado através de um LED e esse procedimento é conhecido como blink, ou seja, acender e apagar o LED em intervalos pré-definidos.

Para quem está iniciando com Arduino, o blink será provavelmente o primeiro projeto a ser executado, pois é simples e pode ser executado de duas formas: sem necessidade de um esquema de ligação, conforme Figure 1.1; com um esquema de ligação externo ao arduino.

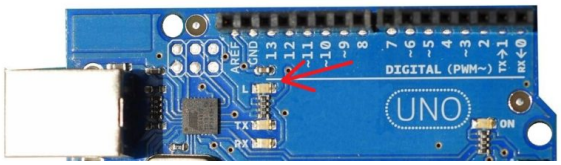


Figure 1.1: LED conectado ao pino digital 13 da placa do arduino que possibilita um blink sem a necessidade de esquema de ligação.

1.1.1 IDE do Arduino

Para instalar a IDE do Arduino acesse o link https://www.arduino.cc/en/software e em seguida retorne para continuar.

Depois de instalado, execute o "arduino.exe" para carregar a IDE (provavelmente um atalho foi criado na sua área de trabalho).

Com a IDE aberta, será mostrado algumas opções na barra de ferramentas da mesma. Abaixo é feito um apanhado geral (das opções relevantes) do menu Arquivo e Ferramentas:

1. Arquivo
    a. Novo: abre uma nova instância da IDE.
    b. Abrir: carrega na IDE projetos salvos no computador.
    c. Exemplos: disponibiliza diversos exemplos que são separados por pastas e bibliotecas. Os exemplos estão em submenus e para abrir algum deles, basta selecionar e clicar.
    d. Fechar: fecha a instância atual.
    e. Salvar: salva as últimas modificações do seu projeto.

f. Salvar como: se o projeto já foi salvo, mas deseja salvar em outro local, essa é a opção indicada.
g. Preferências: configurações gerais da IDE.
h. Sair: encerra definitivamente a IDE.

2. Ferramentas
   a. Monitor serial: executa o terminal serial que auxilia no recebimento e envio de dados para a placa sem a necessidade de recorrer a uma ferramenta externa.
   b. Placa: possibilita selecionar o modelo da placa Arduino que está conectado ao computador.
   c. Porta: possibilita selecionar a porta COM em que o Arduino está recebendo/enviando informações.

Na Figure 1.1 uma breve descrição dos botões relevantes para testes e envio do código para o Arduino:

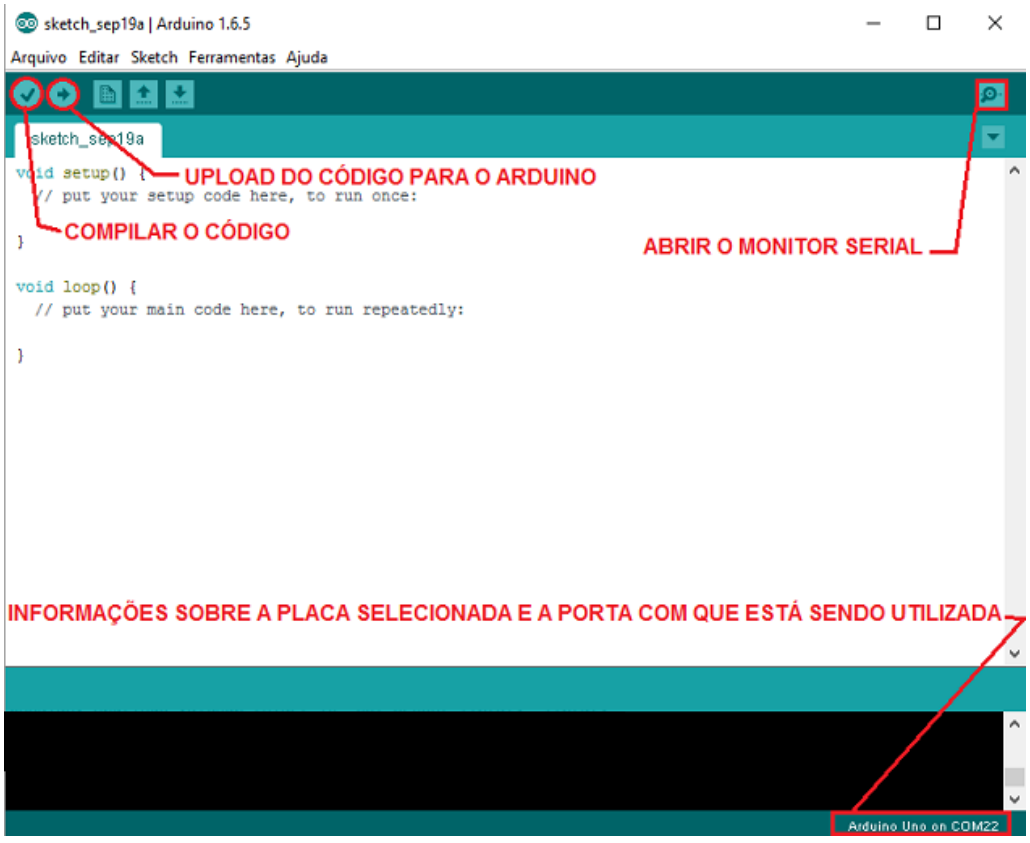


Figure 1.2: Descrição dos botões da IDE do Arduino.

Definindo a placa e a porta na IDE:

Com a IDE instalada e o Arduino instalado no seu Windows, será necessário definir a placa Arduino que você está utilizando e a porta COM em que a placa está conectada. Sem executar esse procedimento, caso você escreva o código ou utilize algum exemplo de código e tente carregar no Arduino o carregamento do código para a placa não será bem sucedido e um erro será mostrado na IDE.

No menu “Ferramentas” selecione a opção “Placas”, e nas opções que abrirem na tela selecione o nome referente a sua placa que está conectada no computador. Veja que na imagem abaixo está selecionado a placa “Arduino Uno”, pois é modelo da placa que está conectado ao computador, porém se sua placa for outro modelo, basta selecionar na lista.

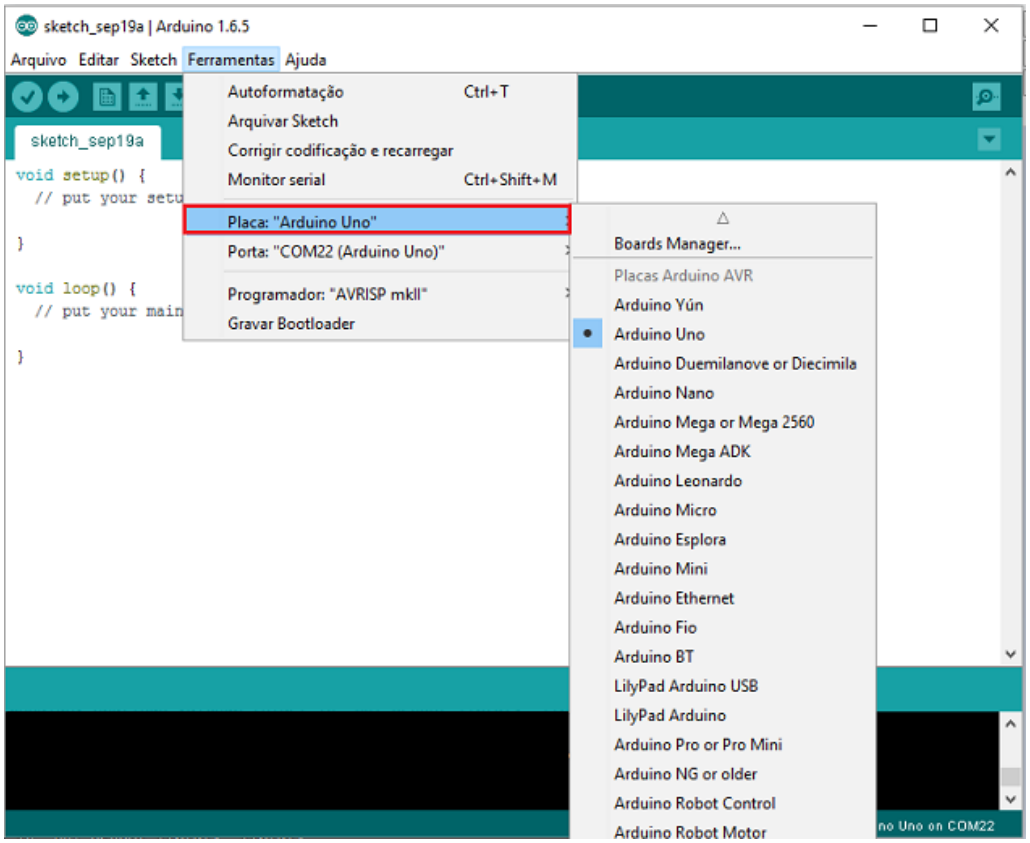


Figure 1.3: Definição da placa na IDE como Arduino UNO.

Em seguida no menu “Ferramentas” seleciona a opção “Portas”. Aparecerá a porta COM1 e a porta em que seu Arduino está conectado, caso ele tenha sido instalado corretamente. Veja que na imagem abaixo está selecionado a porta “COM22”, contudo, no seu computador a porta provavelmente será descrita por outro número:

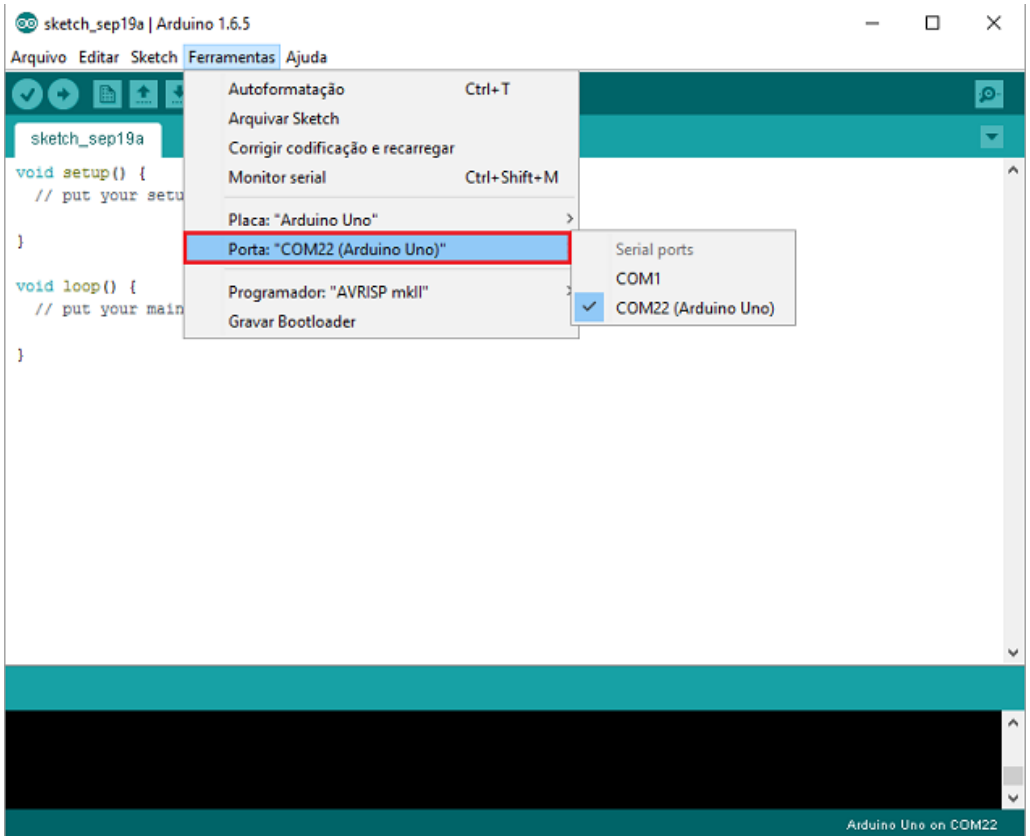


Figure 1.4: Definição da porta na IDE como COM22.

Feito essas configurações sua IDE está pronta para enviar os códigos ao Arduino.

Para um teste rápido, na IDE clique no menu “Arquivo”, selecione a opção “Exemplos”, em seguida “01.Basics” e selecione “Blink”. Uma nova janela da IDE vai abrir. Faça a conferência das informações para garantir que nada foi alterado na placa e na porta. Se atente ao rodapé da IDE que mostra a placa e a porta COM que está configurada na IDE.

Feito a conferência, basta clicar no botão de upload do código para o Arduino e aguardar o código ser carregado na placa. O exemplo Blink faz com que o LED da placa Arduino pisque a cada 1 segundo. Observe na placa o LED (amarelo / verde / vermelho) que está piscando:

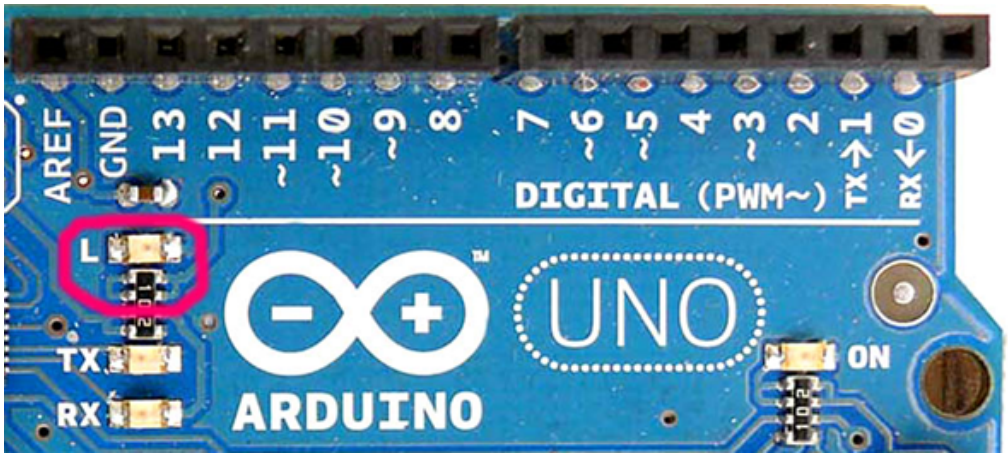


Figure 1.5: LED.

Algumas placas já saem de fábrica com o Blink carregado, portanto, ao ligar o Arduino mesmo

que ele não esteja instalado no computador, o LED da placa vai ficar piscando em um determinado intervalo de tempo.

Sempre que abrir a IDE, faça a conferência das informações para garantir que nada foi alterado. Geralmente, feito as configurações pela primeira vez ela vai se manter sem necessidade de fazer alteração ou reconfigurar. Importante ressaltar que se você instalar um novo Arduino no Windows será necessário fazer o procedimento novamente. Considerando o exemplo das imagens, onde está instalado o Arduino Uno na porta COM22: caso seja conectado e instalado um Arduino Mega 2560 na porta COM15, ao abrir a IDE você deverá fazer os procedimentos para definir a placa e a porta.

Reforçando: sem executar os passos de definição de placa e porta, caso você escreva o código ou utilize algum exemplo de código e tente carregar no Arduino, o carregamento do código para a placa não será bem sucedido e um erro será mostrado na IDE.

1.1.2 Instalando Bibliotecas

Algumas bibliotecas são necessárias para que o código do Arduino seja compilado e carregado corretamente de acordo com os recursos utilizados. Contudo, há bibliotecas que não são nativas da IDE do Arduino e torna-se necessário importar tais bibliotecas de forma manual.

Abra a IDE do Arduino e na barra de ferramentas clique em Sketch > Include Library > Add .ZIP Library...:

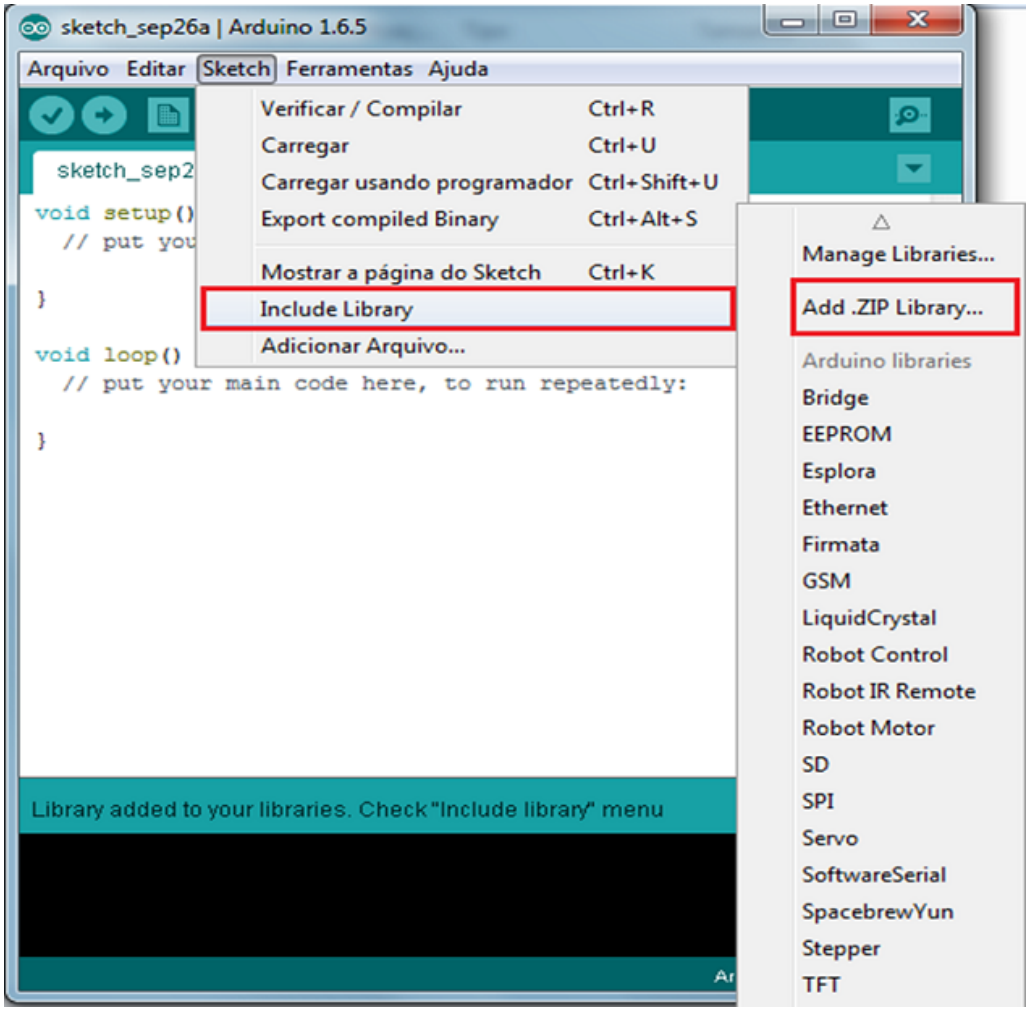


Figure 1.6: Adicionando Biblioteca.

Na tela de seleção de arquivo que vai abrir pesquise a pasta onde se encontra a biblioteca a ser

importada e selecione a mesma (arquivo ZIP) a ser importada e clique em “Abrir”:

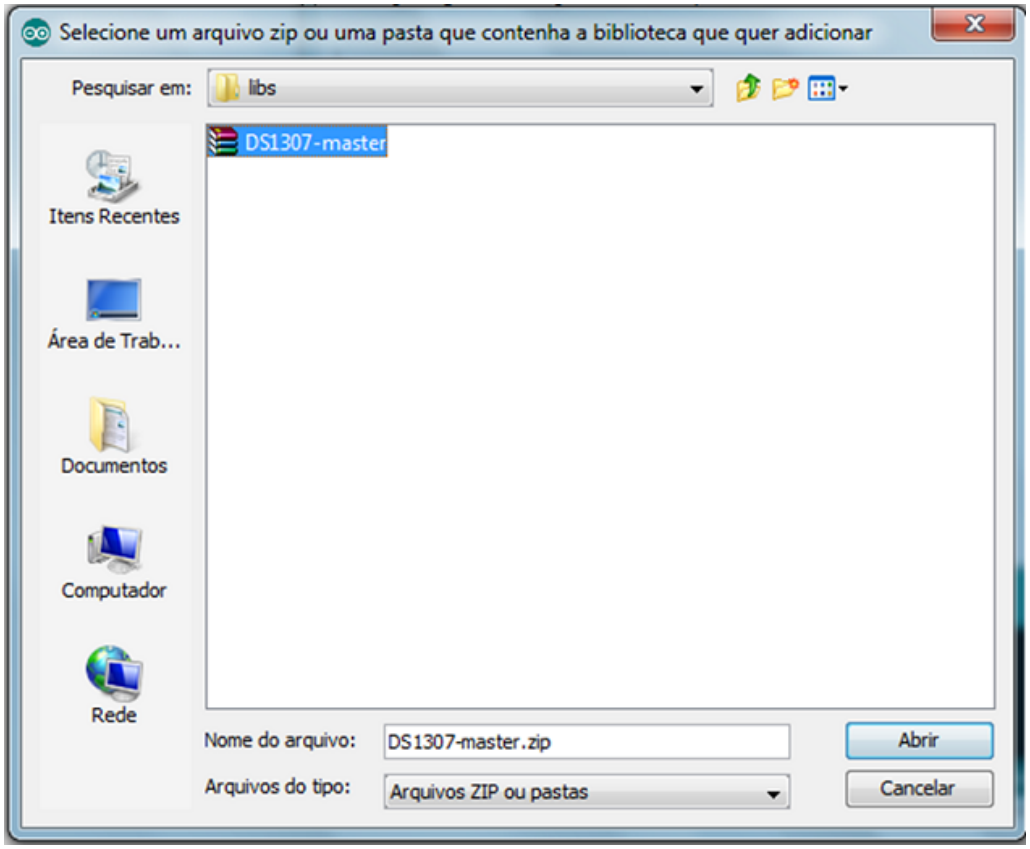


Figure 1.7: Selecionando arquivo ZIP.

Nesse momento a biblioteca já foi importada para sua IDE. Para verificar se foi importada corretamente, vá até a barra de ferramentas da IDE, clique em Arquivo > Exemplos e verifique se a biblioteca importada encontra-se na lista.

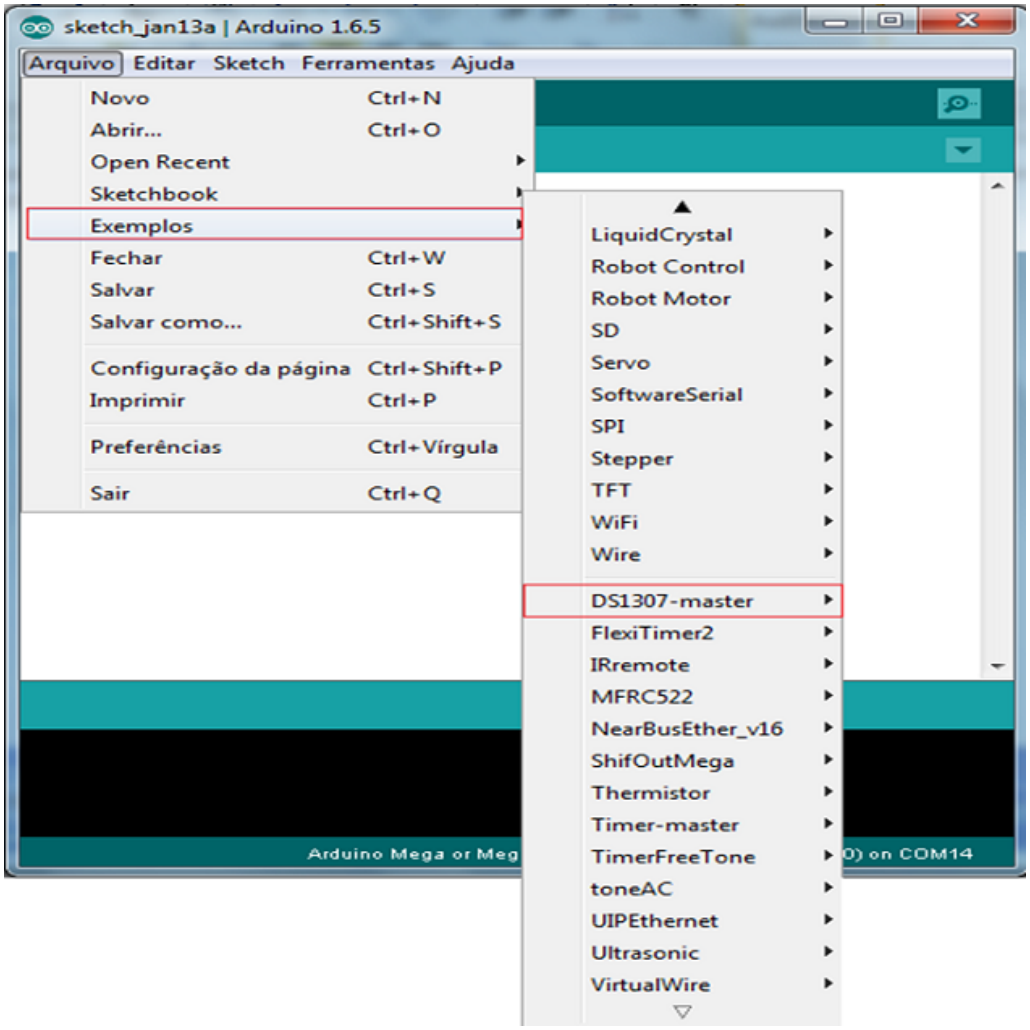


Figure 1.8: Verificando o arquivo ZIP.

Faça esse mesmo procedimento para toda biblioteca que precisar ser importada e prossiga no seu projeto ou prática.

1.1.3 Algoritmo

Abaixo está o código que você deve carregar no Arduino. Conecte a placa ao computador, abra a IDE, confira a placa e a porta selecionada e carregue o código na placa:

Exercise 1.1 Blink Led

```
void setup() {
  pinMode(13, OUTPUT); //DECLARA O PINO 13 COMO SENDO SAÍDA
}
void loop() {
  digitalWrite(13, HIGH); //LIGA O PINO 13 E O LED DA PLACA ACENDE
  delay(1000); //INTERVALO DE 1 SEGUNDO (1000 MILISEGUNDOS = 1 SEGUNDO)
  digitalWrite(13, LOW); //DESLIGA O PINO 13 E O LED DA PLACA APAGA
  delay(1000); //INTERVALO DE 1 SEGUNDO (1000 MILISEGUNDOS = 1 SEGUNDO)
}
```

Basicamente, este código vai ligar e desligar o LED do pino 13 em intervalos de 1 segundo.

No código você pode alterar as linhas ‘delay(1000);’ e dentro dos parênteses remover o 1000 e colocar outro valor para diminuir ou aumentar o intervalo que o LED permanecerá aceso ou apagado. Lembre-se de que o 1000 entre parênteses é igual a 1000 milissegundos que é igual 1 segundo. Portanto, se você deseja que o intervalo seja de 5 segundos, deverá colocar 5000 entre os parênteses.

Vamos executar novamente o blink conforme o esquemático da Figure 1.9:

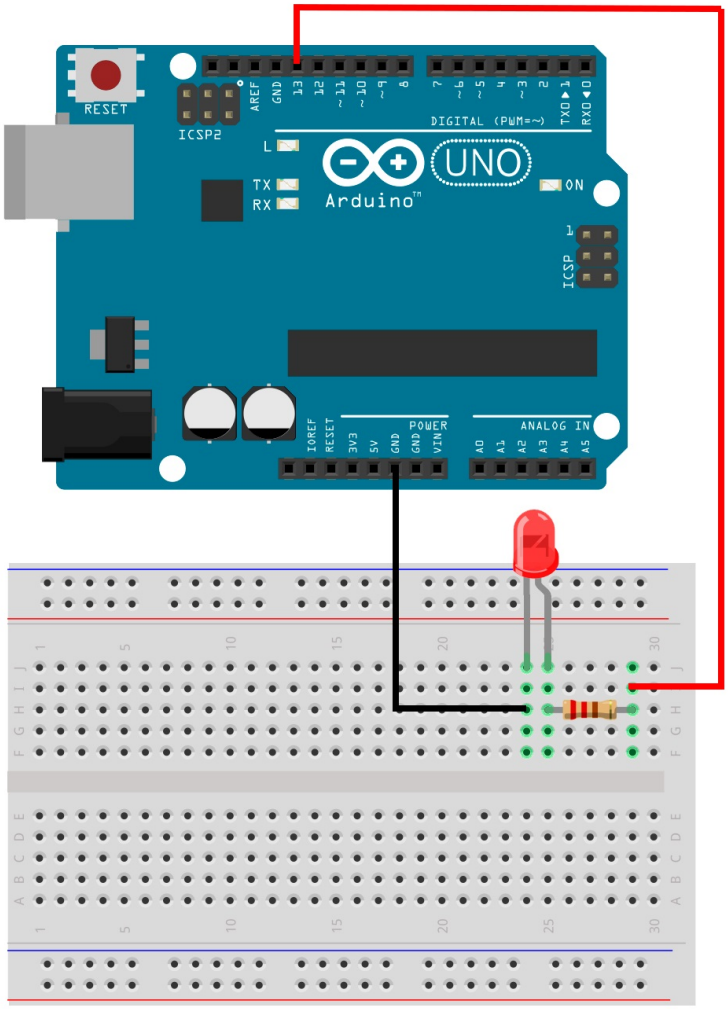


Figure 1.9: Verificando o arquivo ZIP.

1.2 Eletrônica

1.2.1 Arduino

Arduino é uma plataforma de prototipagem eletrônica muito versátil e amplamente utilizada por estudantes, hobbistas e profissionais das mais diversas áreas. O objetivo principal do Arduino é tornar o acesso à prototipagem eletrônica mais fácil, mais barata e flexível. As versões mais simples da palca utilizam um microcontrolador da família Atmel AVR e uma linguagem de programação baseada em C/C++. Com ele é possível criar projetos variados em eletrônica, desde os mais simples até aplicações intermediárias como Internet das Coisas (IoT), Robôs, Sistemas de Automação Residencial ou Industrial, Alarmes e outros.

As funcionalidades do Arduino também podem ser facilmente ampliadas, ou seja, você não precisa trocar a placa principal caso queira expandir os recursos do seu projeto. Basta acrescentar

sensores, módulos e shields para incorporar novas funções. Além disso, depois de programado, o Arduino pode ser utilizado sem a necessidade de um computador, já que o programa instalado na placa permanece em loop, repetindo sem parar, sendo necessário apenas uma fonte de alimentação para que a placa funcione.

O Arduino foi desenvolvido com base no conceito open-source, em tradução literal "código aberto", que significa que o projeto da placa e o firmware podem ser utilizados livremente por outros desenvolvedores e fabricantes. Essa forma de inserção na eletrônica e programação inovou o movimento maker, também conhecido por sua característica "faça você mesmo". A tecnologia e os softwares livres têm promovido uma quarta revolução industrial, que reflete na comunidade maker e no modelo de criação e desenvolvimento de projetos: por ela você idealiza, compartilha e recria outras ideias.

1.2.1.1 Tipos de placa Arduino

O processo de desenvolvimento das tecnologias do Arduino, desde sua primeira versão, produziu um conjunto de placas que podem ser classificadas em função dos recursos e poder de processamento ou controle:

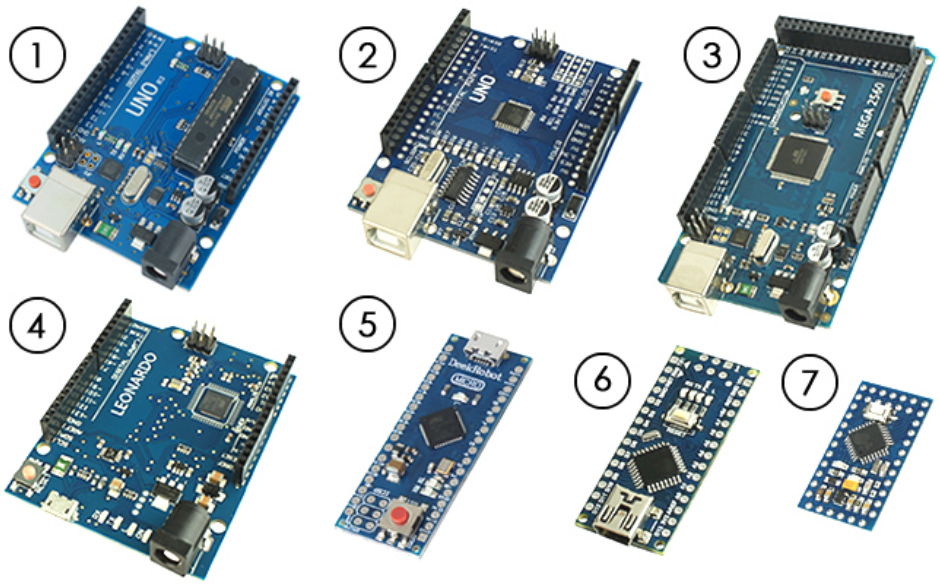


Figure 1.10: Placas Arduino - Modelos Uno, Uno SMD, Mega 2560, Leonardo, Nano, Micro e Pro Micro respectivamente. A ideal para o seu projeto robótico vai depender de quais funcionalidades vai precisar e o tamanho físico necessário.

1.2.2 Protoboard

As protoboads talvez sejam umas das ferramentas mais importes para quem esteja começando com eletrônica e montagem de circuitos, pois com ela é possível montar dezenas de circuitos sem a necessidade de soldar qualquer componente. Segue o seguinte esquema de conexão interna:

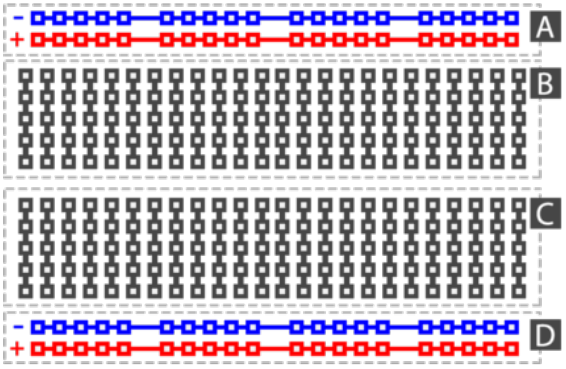


Figure 1.11: Nas seções A e D geralmente são conectados VCC e GND. As seções B e C são utilizadas para conexão dos componentes eletrônicos.

Na Figure 1.11 observa-se que:
- As trilhas vermelhas são utilizadas para ligar o positivo dos componentes (linha de energia);
- As trilhas azuis são utilizadas para ligar o negativo dos componentes ("ground" ou fio terra);
- As trilhas pretas são utilizadas para realizar o contato elétrico entre os componentes desejados.

1.2.3 Fonte

Quando alimentado com 5V a partir da conexão USB, normalmente um Arduino não consome muita energia. O Arduino Uno, por exemplo, consome cerca de 40mA, correspondendo a apenas 200mW. Isso significa que pode facilmente funcionar com uma bateria de 9V durante umas 4 horas.

O consumo de corrente é importante quando o Arduino deve funcionar por um longo período de tempo usando baterias como monitoração remota ou no caso de situações de controle em que o uso de bateria ou energia solar é a única opção.

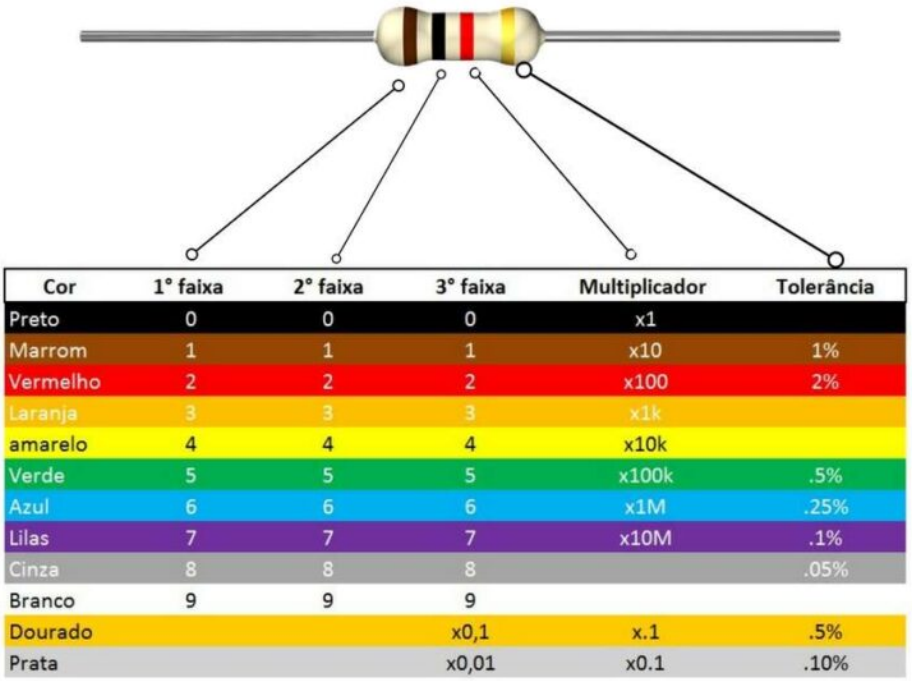
Figure 1.12: Fonte Arduino.

1.2.4 Resistência

O valor do resistor no projeto do blink varia de 120 ohms a 220 ohms dependendo do LED que for utilizar.

A tabela funciona da seguinte forma: a 1°, 2°, 3° faixa são os valores a 4° faixa é um fator multiplicativo que irá multiplicar a sequência encontrada na 1°, 2° e 3° faixa. Na ausência da 3°

faixa se leem somente as duas primeiras, no caso um resistor de 4 faixas a 3° faixa será o fator multiplicador e em um resistor de 5 faixas será a 4° faixa. A ultima faixa sempre vai dizer respeito a tolerância do valor da resistência. Abaixo temos uma foto com um exemplo para exemplificar melhor:



Cor	1° faixa	2° faixa	3° faixa	Multiplicador	Tolerância
Preto	0	0	0	x1	
Marrom	1	1	1	x10	1%
Vermelho	2	2	2	x100	2%
Laranja	3	3	3	x1k	
amarelo	4	4	4	x10k	
Verde	5	5	5	x100k	.5%
Azul	6	6	6	x1M	.25%
Lilas	7	7	7	x10M	.1%
Cinza	8	8	8		.05%
Branco	9	9	9		
Dourado			x0,1	x.1	.5%
Prata			x0,01	x0.1	.10%

Figure 1.13: Tabela de cores da resistência.

1° faixa cor marrom = 1

2° faixa cor preto = 0

3° faixa cor vermelha = 2

4° faixa cor dourado = +-0,5

Logo temos 10×100 que é igual a 1000 que é igual a 1K, isso significa que temos um resistor de 1K com uma tolerância de +-5

1.2.5 Led

O LED inclui dois pinos:

O pino do cátodo(-) precisa ser conectado ao GND (0V).

O pino ânodo(+) é usado para controlar o estado do LED.

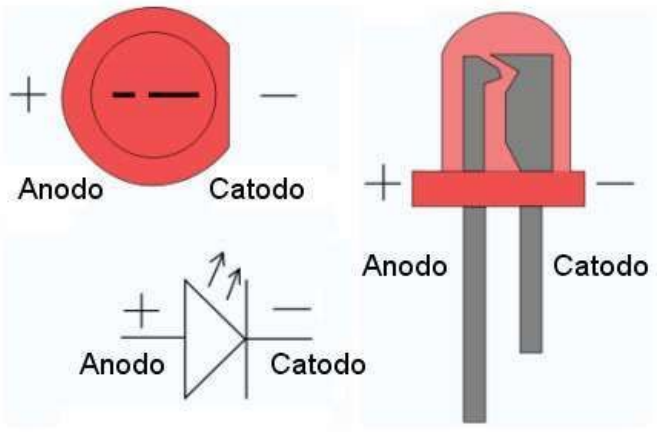


Figure 1.14: Lados de um LED.

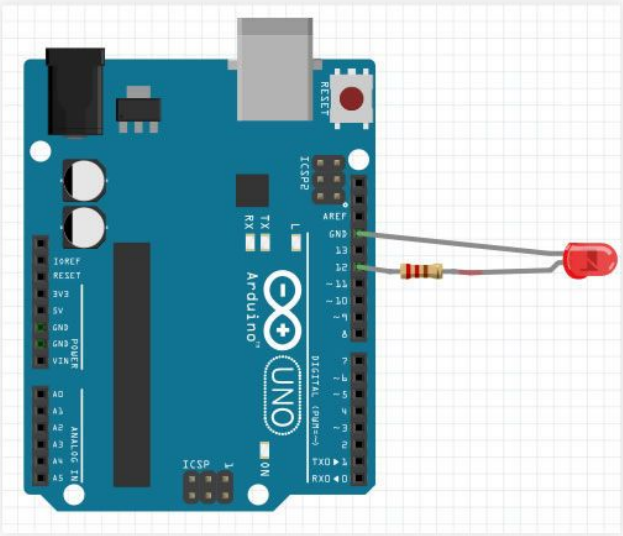


Figure 1.15: Montagem dos componentes sem protoboard.

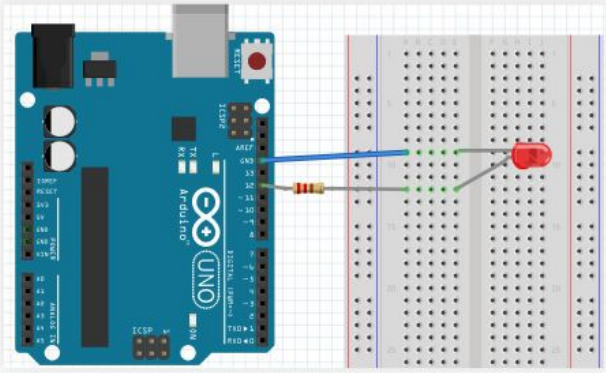


Figure 1.16: Montagem dos componentes com protoboard.

1.3 Software

O Arduino IDE é a opção mais utilizada e mais conhecida pelos desenvolvedores, com este método é possível realizar a programação e fazer o upload de códigos em qualquer lugar, tenha acesso a Internet ou não.

O Arduino também apresenta um Web Editor, é a versão online do Software, sendo facilmente encontrado junto ao site oficial Arduino, porém tem como desvantagem, a necessidade de estar-se conectado à internet para sua utilização tendo em vista que algumas de suas funções utilizam bancos de dados online.

Após acessar o Arduino Web Editor, você terá que realizar o Login em sua conta Arduino ou conta Google para ter acesso aos benefícios desta ferramenta.

1.3.1 Piscar um LED

Ver 1.1.3.

1.3.2 Piscar dois LEDs

Nossos materiais são quase os mesmos do Blink Led, mas desta vez vamos fazer dois circuitos de LED separados (com os mesmos componentes), conforme a figura 1.17 . Então, precisamos de dois LEDs vermelhos e dois resistores de 220Ω. Agora que estamos usando mais componentes, também precisaremos cada vez mais de uma placa de ensaio (Protoboard) — o que facilitará a criação de um circuito limpo e organizado.

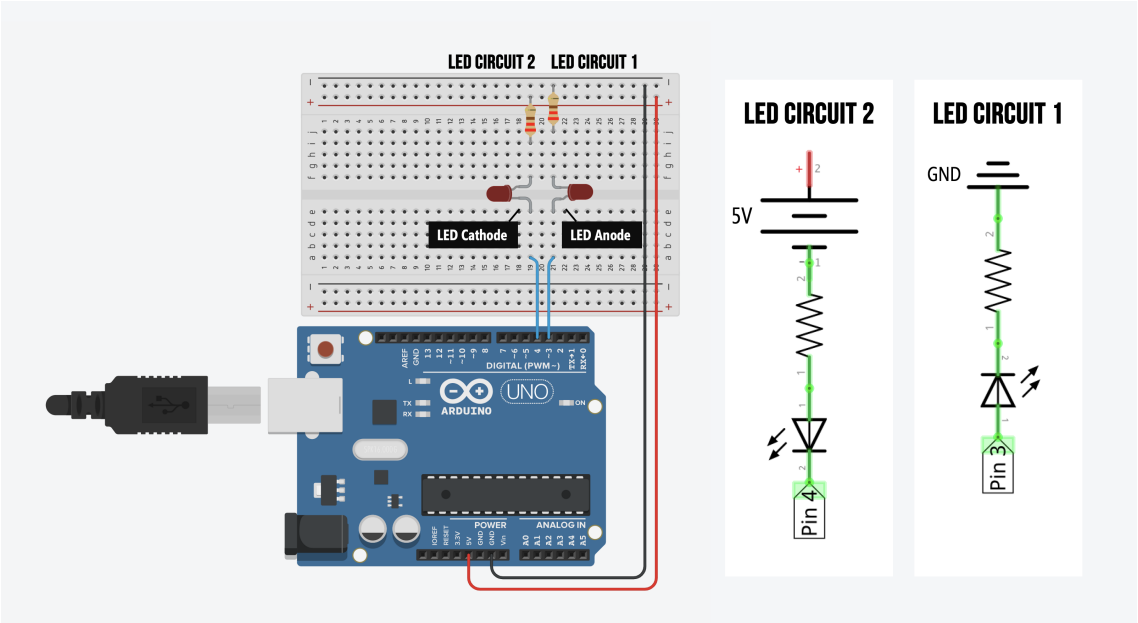


Figure 1.17: Piscar 2 LEDs.

Vamos escrever código para piscar os LEDs ligados aos pinos 3 e 4.

Observe que quando o pino 3 é HIGH(5V), há uma diferença de tensão entre o pino 3 e GND, portanto, a corrente flui do pino 3 para o terra. Quando o pino 4 é HIGH(5V), no entanto, não há diferença de tensão no circuito (do pino 4 a 5V) e, portanto, não há corrente. Esse comportamento é ilustrado na animação abaixo.

Lembre-se de que a corrente sempre flui do potencial de alta tensão para o potencial de baixa tensão.

Exercise 1.2 PISCAR DOIS LEDs

```
// FUNÇÃO DE CONFIGURAÇÃO
void setup() {
// DECLARANDO DOIS PINOS COMO SAÍDA
   pinMode(3, OUTPUT);
   pinMode(4, OUTPUT);
}
// FUNÇÃO DO LOOP
void loop() {
   digitalWrite(3, HIGH); // LED1 ESTADO ALTO
   digitalWrite(4, HIGH); // LED2 ESTADO BAIXO
   delay(1000); // DELAY DE 1 SEGUNDO
   digitalWrite(3, LOW); // LED1 ESTADO BAIXO
   digitalWrite(4, LOW); // LED2 ESTADO ALTO
   delay(1000); // DELAY DE 1 SEGUNDO
}
```

1.3.3 Fazendo um semáforo

A ideia é simular o funcionamento de um semáforo com Arduino. A lógica do funcionamento de um sinal varia um pouquinho dependendo de onde você mora mas de um modo geral segue sempre o mesmo padrão.

O circuito eletrônico é mostrado na figura 1.18, no entanto é apenas uma sugestão e que pode ser modificado.

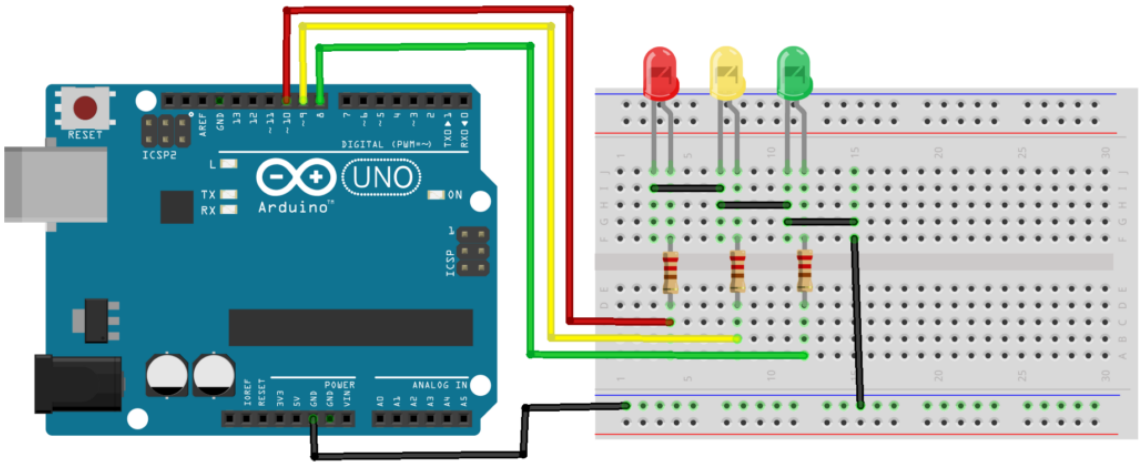


Figure 1.18: Semáforo.

O que vamos fazer nada mais é do que determinar quais leds estarão acesos e quando serão as trocas. Como sugestão segue o algoritmo definido no exercício 1.3.

Exercise 1.3 SEMÁFORO

```
// dando um "nome" para as portas
int vermelho = 10;
int amarelo = 9;
int verde = 8;
```

```
void setup() {
// indicando para o arduíno quais portas vamos usar
pinMode(vermelho, OUTPUT);
pinMode(amarelo, OUTPUT);
pinMode(verde, OUTPUT);
}
void loop() {
// vamos começar do amarelo. Estranho não?
// você vai entender no próximo exercício!
digitalWrite(vermelho, LOW);
digitalWrite(amarelo, HIGH);
digitalWrite(verde, LOW);
// esperamos 2s com o sinal no amarelo delay(2000);
// apagamos o amarelo e ligamos o vermelho
digitalWrite(amarelo, LOW);
digitalWrite(vermelho, HIGH);
// Não precisa desse pois o verde já estava apagado
// digitalWrite(verde, LOW);
// esperamos 5s com o sinal fechado
delay(5000);
// para finalizar, apagamos o vermelho e ligamos o verde
digitalWrite(verde, HIGH);
// não precisa desse pois o amarelo já estava apagado
// digitalWrite(amarelo, LOW);
digitalWrite(vermelho, LOW);
// esperamos 5s com o sinal aberto
delay(5000);
}
```

1.3.4 Fazendo um semáforo modelo cruzamento

Vamos simular agora o funcionamento de semáforos em um cruzamento de ruas. Teremos dois semáforos que atuarão de forma sincronizada. O esquema eletrônico encontra-se na figura 1.19 e o código no exercício 1.4.

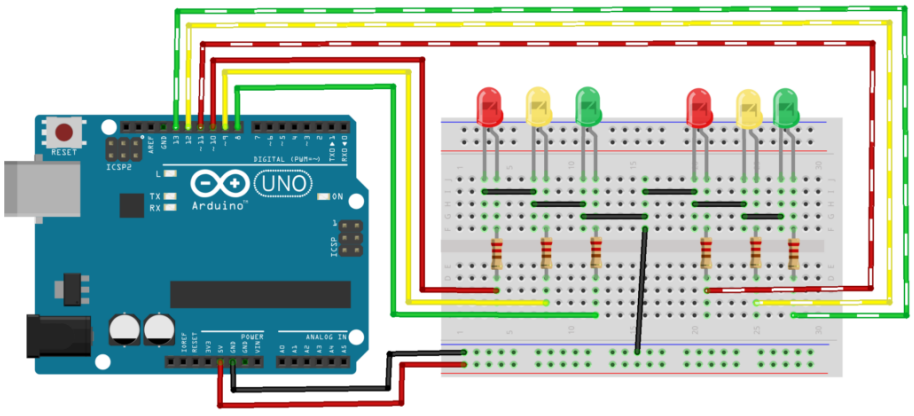


Figure 1.19: Semáforo modelo cruzamento.

Exercise 1.4 SEMÁFORO MODELO CRUZAMENTO

```
// dando um "nome" para as portas
int vermelho1 = 10;
int vermelho2 = 11;
int amarelo1 = 9;
int amarelo2 = 12;
int verde1 = 8;
int verde2 = 13;
void setup() {
// indicando para o arduino quais portas vamos usar
pinMode(vermelho1, OUTPUT);
pinMode(amarelo1, OUTPUT);
pinMode(verde1, OUTPUT);
pinMode(vermelho2, OUTPUT);
pinMode(amarelo2, OUTPUT);
pinMode(verde2, OUTPUT); }
void loop() {
cruzamento();
}
void cruzamento() {
// sinal 1 - amarelo
digitalWrite(vermelho1, LOW);
digitalWrite(amarelo1, HIGH);
digitalWrite(verde1, LOW);
// sinal 2 - amarelo também
digitalWrite(vermelho2, LOW);
digitalWrite(amarelo2, HIGH);
digitalWrite(verde2, LOW);
// esperamos 2s com o sinal no amarelo
delay(2000);
// sinal 1
// apagmos o amarelo e ligamos o vermelho
digitalWrite(amarelo1, LOW);
digitalWrite(vermelho1, HIGH);
// digitalWrite(verde1, LOW);
// sinal 2
// apagmos o amarelo e ligamos o verde
digitalWrite(amarelo2, LOW);
// digitalWrite(vermelho2, LOW);
digitalWrite(verde2, HIGH);
// esperamos 5s com o sinal fechado (1)/ aberto (2)
delay(5000);
// sinal 1
// apagamos o vermelho e ligamos o verde
digitalWrite(verde1, HIGH);
// digitalWrite(amarelo1, LOW);
digitalWrite(vermelho1, LOW);
// sinal 2
// apagamos o verde e ligamos o vermelho
```

```
digitalWrite(verde2, LOW);
// digitalWrite(amarelo2, LOW);
digitalWrite(vermelho2, HIGH);
// esperamos 5s com o sinal aberto
delay(5000);
}
```

1.3.5 Acender um LED com uma push button

O objetivo deste projeto é ligar e desligar um LED com um botão push button. Desta forma, quando o botão for pressionado o LED deverá acender e retornará ao estado desligado quando o botão deixar de ser pressionado. O esquema eletrônico encontra-se na figura 1.20 e o código no exercício 1.5.

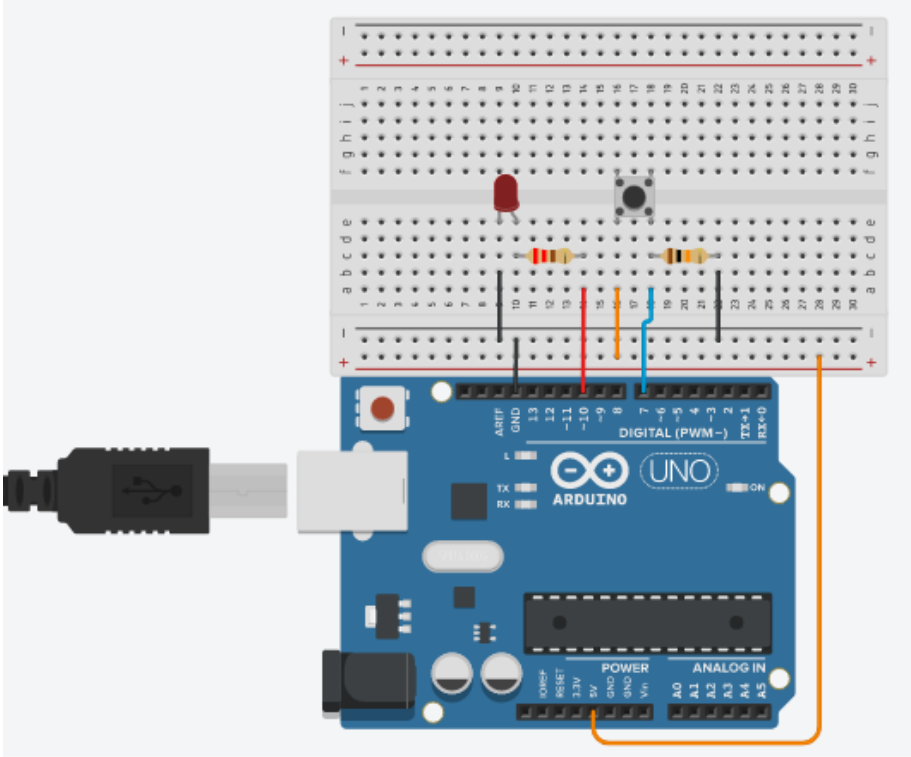


Figure 1.20: Acendendo um LED com uma push button.

Exercise 1.5 ACENDENDO UM LED COM UMA PUSH BUTTON

```
int buttonPin = 7;     //Define buttonPin no pino digital 7
int ledPin = 10;     //Define ledPin no pino digital 10
int estadoButton = 0;//Variável que armazena o estado do botão (ligado/desligado)
void setup(){
   pinMode(ledPin , OUTPUT);     //Define ledPin (pino 10) como saída
   pinMode(buttonPin , INPUT);     //Define buttonPin (pino 7) como entrada
}
void loop(){
   estadoButton = digitalRead(buttonPin);//Lê o valor buttonPin e armazena em estadoButton
   if (estadoButton == HIGH) {     //Se estadoButton for igual a HIGH ou 1
```

```
    digitalWrite(ledPin , HIGH);    //Define ledPin como HIGH, ligando o LED
    delay(100);    //Intervalo de 100 milissegundos
    }
    else {    //Senão = estadoButton for igual a LOW ou 0
    digitalWrite(ledPin, LOW);    //Define ledPin como LOW, desligando o LED
    }
}
```

Vamos entender a lógica de programação deste projeto a partir dos seguintes passos:

1. Declarar as variáveis:

O primeiro passo na construção será a declaração de variáveis. Desta forma, definimos a porta digital 7, em que o botão está conectado, a variável buttonPin, definimos a porta digital 10, em que o LED está conectado, a variável ledPin, e criamos a variável estadoButton, do tipo inteiro, para armazenar o estado (HIGH ou LOW) do botão.

2. Configurar as portas de saída e entrada:

Na função setup() configuraremos as portas de entrada e saída da placa UNO. A porta 10 (ledPin) deve ser configurada como saída e a porta 7 (buttonPin) deve ser configurada como entrada.

3. Realizar a leitura da porta digital:

Na função loop() escreveremos todos os comandos e operações para ligar e desligar o LED com o botão. Iniciamos o loop() realizando a leitura da porta digital 7 (buttonPin), para isso utilizaremos a função digitalRead(buttonPin), e armazenaremos este valor na variável estadoButton.

4. Realizar a comparação

Utilizaremos a lógica do if...else para comparar se a variável estadoButton encontra-se em nível alto ou baixo (chave pressionada ou chave não pressionada). Se a variável estadoButton estiver em nível lógico alto (chave pressionada) o LED será ligado através do comando digitalWrite(ledPin, HIGH);. Senão, o LED deve ser desligado por meio da instrução digitalWrite(ledPin, LOW);.

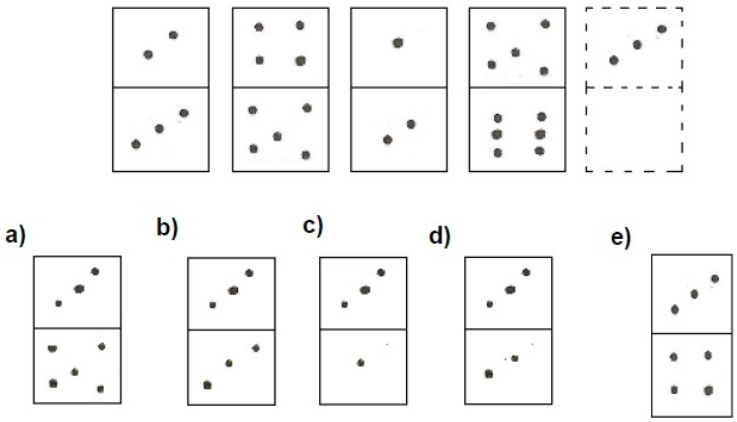
Tente fazer este mesmo projeto substituindo o LED por um buzzer.

1.4 Lógica

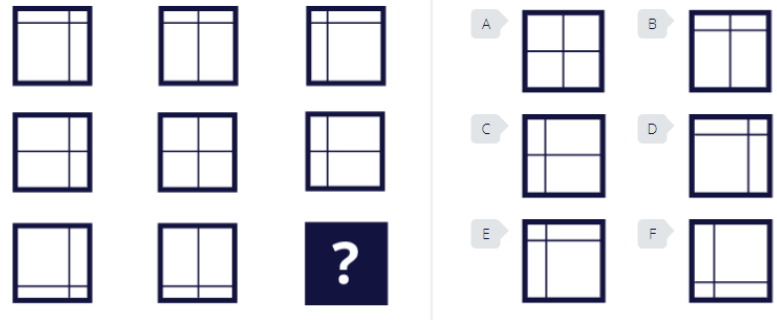
A programação do robô é a parte que mais exige o exercício do raciocínio, pois pede uma forma de pensar lógica e ordenada. No entanto, existem outras formas de aprimorar este pensamento por meio de jogos ou por meio de atividades. A seguir tente resolver as questões.

Assinale a opção que completa a sequência:

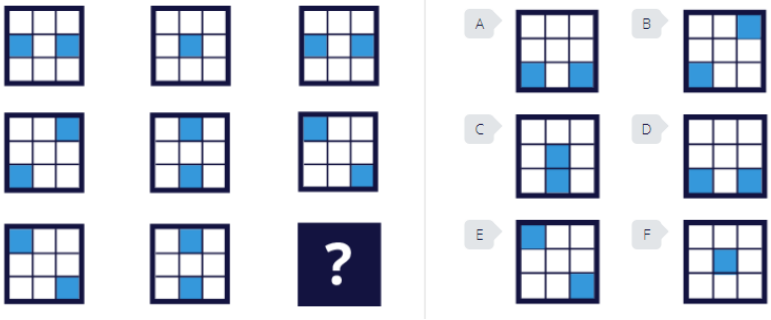
1)



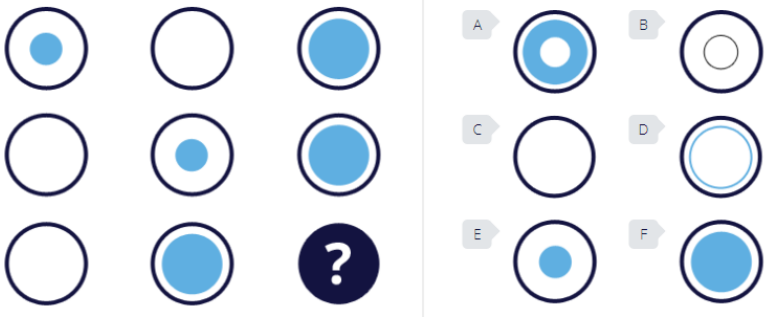
2)



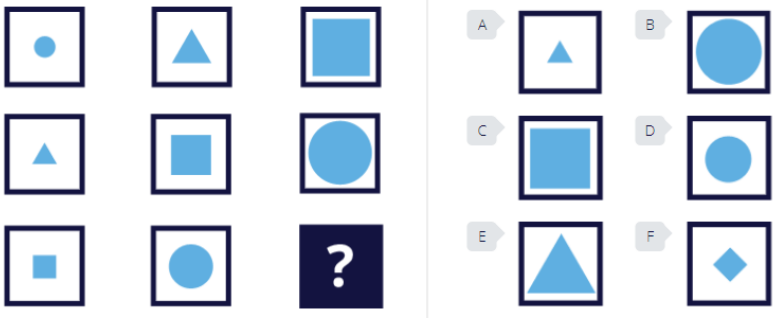
3)



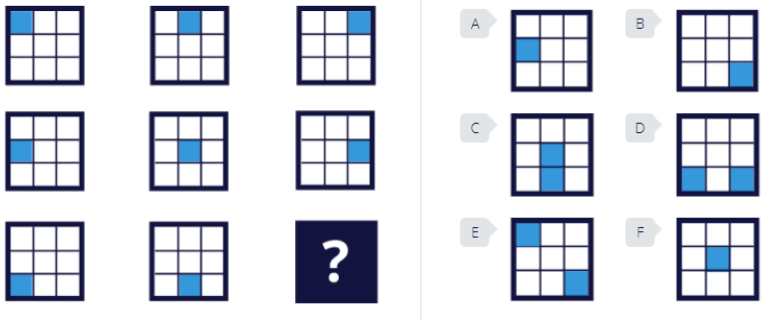
4)



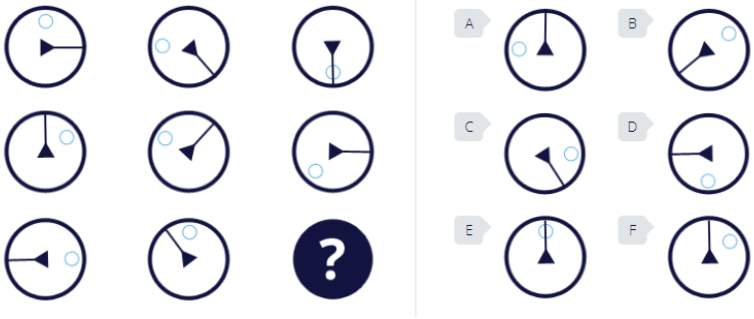
5)



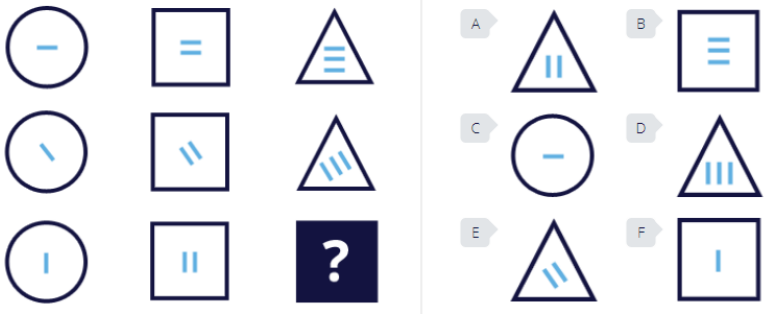
6)



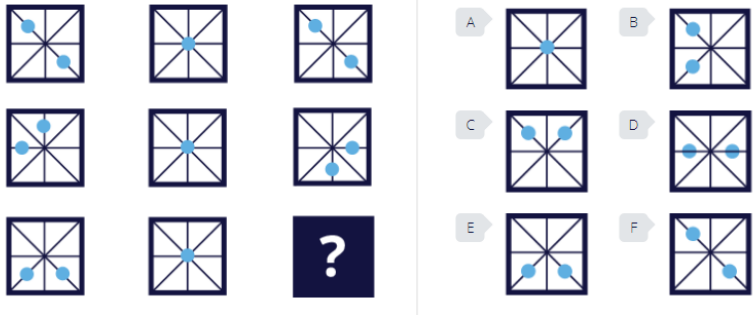
7)



8)

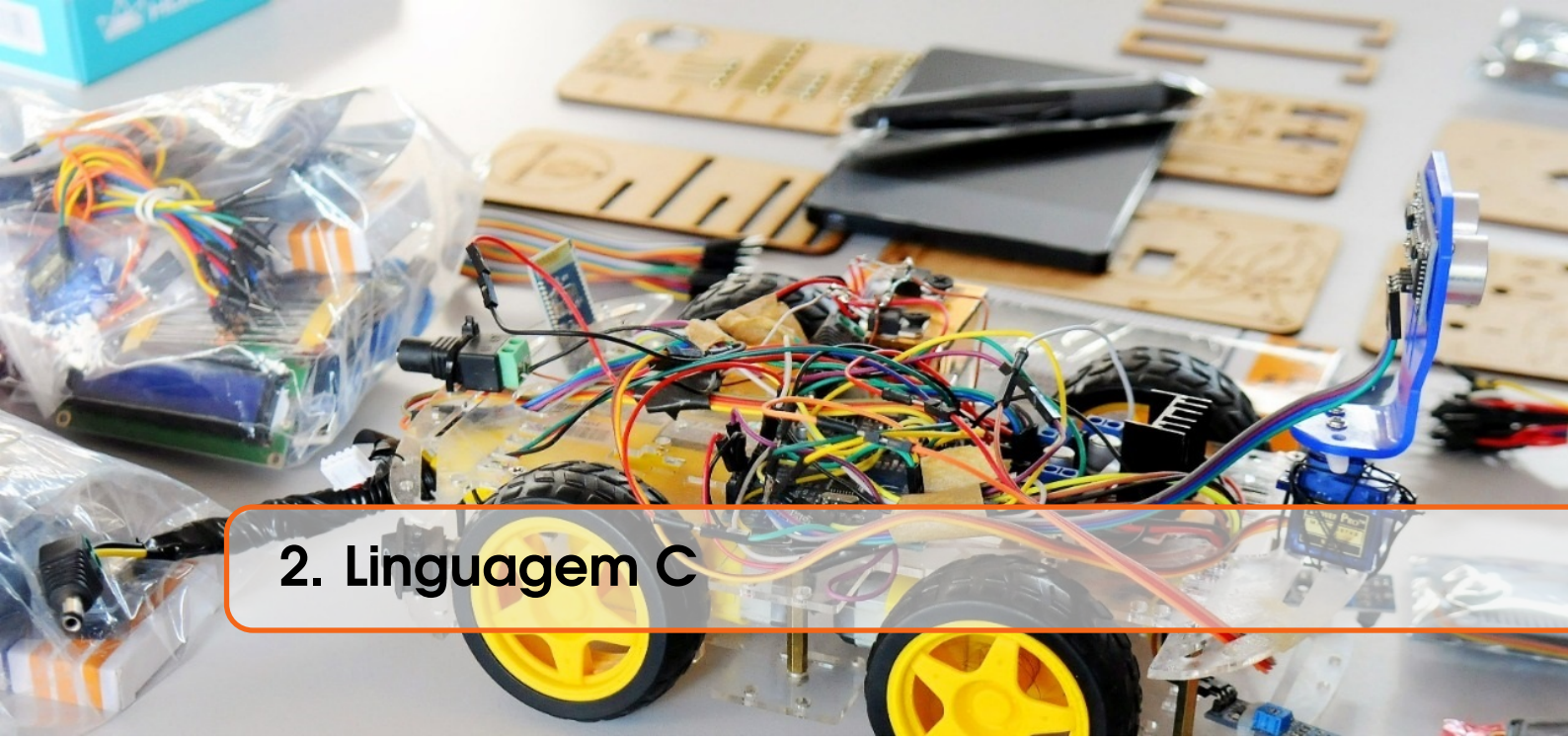


9)



10)


38
37
35
32
28
23
17
10
?
A
38
B
2
C
8
D
18
E
25
F
5



2. Linguagem C

2.1 Introdução

A Linguagem C, criada em 1970 por Dennis Ritchie, é uma evolução da Linguagem B que, por sua vez, foi uma adaptação feita, a partir da Linguagem BCPL, por Ken Thompson. Ela é estreitamente associada ao sistema operacional UNIX, já que as versões atuais do próprio sistema foram desenvolvidas utilizando-se esta linguagem.

Devido ao crescente uso e interesse da comunidade de computação pela linguagem, em 1983, o American National Standards Institute (ANSI), estabeleceu um comitê para prover uma definição moderna e abrangente da linguagem C. O resultado desta comissão foi uma padronização denominada ANSI-C em 1988. A maior contribuição deste trabalho foi a incorporação de uma biblioteca padrão, presentes em todas as variações/versões da linguagem, que fornece funções de acesso ao sistema operacional, entrada e saída formatada, alocação de memória, manipulação de strings (cadeias de caracteres), etc.

2.1.1 Conceitos Básicos

A filosofia básica da linguagem C é que os programadores devem estar cientes do que estão fazendo, ou seja, supõe-se que eles saibam o que estão mandando o computador fazer, e explicitem completamente as suas instruções. Assim, ela alia a elegância e a flexibilidade das linguagens de alto nível (ex: suporte ao conceito de tipo de dados) com o poderio das linguagens de baixo nível (ex: manipulação de bits, bytes e endereços).

O C é uma linguagem de programação de finalidade geral, utilizada no desenvolvimento de diversos tipos de aplicação, como processadores de texto, sistemas operacionais, sistemas de comunicação, programas para solução de problemas de engenharia, física, química e outras ciências, etc. O código-fonte de um programa C pode ser escrito em qualquer editor de texto que seja capaz de gerar arquivos em código ASCII (sem formatação). Como o ambiente de programação utilizado (Turbo C) é para o sistema operacional DOS, estes arquivos devem ter um nome de no máximo 8 caracteres e a extensão “c” (exemplo: NONAME.C).

Após a implementação, o programa-fonte (um ou mais arquivos-fonte) é submetido aos processos de compilação e linkedição para gerar o programa executável (com extensão “exe”). Durante o processo de compilação, cada arquivo-fonte é compilado separadamente, produzindo um arquivo de código-objeto com a extensão “obj”. Estes arquivos-objeto contêm instruções em linguagem de máquina (códigos binários) entendidas somente pelos microprocessadores. Na linkedição, todos os arquivos-objetos pertencentes ao projeto, bem como as bibliotecas declaradas nos códigos-fonte

são processadas em conjunto, visando a produção do arquivo executável correspondente.

Normalmente, tanto o arquivo-objeto quanto o arquivo executável possuem o mesmo nome do arquivo-fonte. Entretanto, quando desejado, o usuário poderá definir diferentes nomes para cada tipo de arquivo.

A linguagem C possui as seguintes características:

• Alta portabilidade inerente da padronização ANSI, ou seja, é possível tomar um códigofonte escrito para uma máquina, compilá-lo e rodá-lo em outra com pouca ou nenhuma alteração;

• Gera programas formados basicamente por funções, o que facilita a modularização e a passagem de parâmetros entre os módulos;

• Inicia a execução a partir da função main(), necessária em todos os programas;

• Uso de chaves ( ) para agrupar comandos pertencentes a uma estrutura lógica (ex: ifelse, do-while, for, etc.) ou a uma função;

• Uso do ponto e vírgula (;) ao final de cada comando;

• É "case sensitive", ou seja, o compilador difere maiúsculas de minúsculas. Assim, se declarar uma variável de nome idade, esta será diferente de Idade, IDADE, etc. Além disso, todos os comandos da linguagem devem ser escritos em minúsculo.

2.1.2 COMENTÁRIOS

Em C, comentários podem ser escritos em qualquer lugar do texto para facilitar a interpretação do algoritmo. Para que o comentário seja identificado como tal, **ele deve ter um /* antes e um */ depois**.

Vale lembrar que, durante a compilação do programa, os comentários são descartados pelo compilador.

Exemplo:

/* esta e´ uma linha de comentário em C */

2.2 Hello World ou Olá Mundo

Desenvolver o código "Hello World" é uma prática comum entre as pessoas que estão aprendendo a programar ou desejam conhecer uma nova linguagem de programação. Normalmente é um dos primeiros contato com as características da linguagem utilizada.

O Hello World é um programa simples, com poucas linhas de código e que retorna a mensagem "Olá, mundo!" na tela. Seu objetivo é facilitar o conhecimento da estrutura básica e o funcionamento da linguagem.

Para compilar os algoritmos aconselho o uso offline do DEV C++ podendo ser feito o download na página (https://www.bloodshed.net/) mas existem compliadores online como o ONLINEGDB (https://www.onlinegdb.com/online_c_compiler).

Exercise 2.1 Olá Mundo

```
#include<stdio.h>
int main (void)
{
   printf("Ola Mundo!!!\n");
   printf("Hello World!!!\n");
   return 0;
}
```

■

Neste exercício 2.1 observa-se que há a presença do #include<stdio.h> que significa a inclusão de um biblioteca. O nome dela é stdio e que significa **st**an**d**art **i**nput **o**utput, padrão de entrada e saída. Esta biblioteca é justamente para o uso da função printf.

Pode-se observar que a estrutura da função se assemelha a forma matemática: $f() = \{()\}$. E que o comando int main informa que é uma função criada do tipo inteira (int) sendo a principal (main).

Uma outra observação é o comando printf que significa uma função de impressão, sendo que esta impressão será feita com um texto na tela: Ola Mundo!!! e Hello World!!!. Estes textos no código estão entre aspas duplas e no final tem um \n. As aspas duplas é obrigatória para existência de um texto no comando printf e o \n é para acrescentar uma linha e é uma abreviação de newline, nova linha.

A existência do return 0 indica que a função retorna ao void (vazio).

2.2.1 Variáveis

Durante a compilação do código-fonte, o compilador C procura alocar o espaço de memória necessário para a execução do programa. Embora o compilador possa determinar os tipos de dados e espaço de memória exigidos para as constantes encontradas no código, no caso das variáveis, isto só é possível se houver a declaração prévia do tipo de dado empregado.

Cada tipo de dado tem uma exigência pré-determinada do tamanho de memória e uma faixa associada de valores permitidos. Por exemplo, um caracter ocupa geralmente 1 byte, enquanto que um inteiro tem normalmente 2 bytes. Entretanto, para garantir a portabilidade do código C, esta suposição não pode ser tomada como verdadeira. Isto porque, o tamanho e a faixa desses tipos de dados variam de acordo com o tipo de processador e com a implementação do compilador C. O padrão ANSI estipula apenas a faixa mínima de cada tipo de dado e não o seu tamanho em bytes.

Há 5 tipos básicos em C, conforme tabela abaixo. Todos os demais tipos são variações destes 5 tipos.

Table 2.1: Variáveis

Tipos	Descrição	Tamanho Aprox.	Faixa Mínima
char	Caractere	8bits = 1 byte	-127 a 127
int	Inteiro	16bits = 2 bytes	-32.767 a 32.767
float	Ponto flutuante	32 bits = 4 bytes	7 dígitos de precisão
double	Ponto flutuante de precisão dupla	64 bits = 8 bytes	10 dígitos de precisão
void	Sem valor		

2.2.2 Bibliotecas

C Standard Library é o nome da biblioteca padrão da linguagem C. Esta biblioteca possui uma série de funções prontas com recursos adicionais e disponíveis para utilização. A biblioteca padrão de C é composta ao todo por 24 arquivos de cabeçalho. Como podemos ver a quantidade de arquivos da biblioteca é bem pequena. A ideia por trás dessa biblioteca é fornecer apenas um conjunto básico de operações, de tal forma que a portabilidade do C ANSI entre diversas plataformas seja relativamente simples.

Além da biblioteca padrão, outras bibliotecas foram desenvolvidas para incorporar outras funcionalidades específicas.

Table 2.2: Bibliotecas e funcionalidades

Nome do arquivo	Descrição do arquivo de cabeçalho
<assert.h>	Implementa ajuda na detecção de erros em versões de depuração de programas.
<complex.h>	Trata de manipulação de números complexos.
<ctype.h>	Funções para conversão de maiúsculas, minúsculas e outros tratamentos de caracteres.
<errno.h>	Teste de códigos de erro reportados pelas funções de bibliotecas.
<fenv.h>	Define várias funções e macros para tratar de exceções em variáveis do tipo ponto flutuante.
<float.h>	Define limites e precisão de variáveis de ponto flutuante.
<inttypes.h>	Trata de conversão precisa entre tipos inteiros.
<iso646.h>	Adiciona a possibilidade de programação usando a codificação de caracteres de acordo com a ISO646.
<limits.h>	Constantes de propriedades específicas de implementação da biblioteca de tipos inteiros, como a faixa de números que pode ser representada (_MIN, _MAX).
<locale.h>	Especifica constantes de acordo com a localização específica, como moeda, data, etc.
<math.h>	Funções matemáticas comuns em computação.
<setjmp.h>	Define as macros setjmp e longjmp, para saídas não locais e tratamento de execeções.
<signal.h>	Implementa definições para receber e fazer o tratamento de sinais.
<starg.h>	Acesso dos argumentos passados para funções com parâmetro variável.
<stdbool.h>	Trata da definição para tipo de dados booleano.
<stdint.h>	Padrões de definição de tipos de dados inteiros.
<stddef.h>	Padrões de definições de tipos.
<stdio.h>	Tratamento de entrada/saída.
<stdlib.h>	Implementa funções para diversas operações, incluindo conversão, alocação de memória, controle de processo, funções de busca e ordenação.
<string.h>	Tratamento de strings.
<tgmath.h>	Implementa facilidades para utilização de funções matemáticas.
<time.h>	Trata de tipos de data e hora.
<wchar.h>	Tratamento de caracteres para suportar diversas línguas.
<wctype.h>	Contém funções para classificação de caracteres wide.

Módulo 2

3. Sensores e Atuadores

3.1 Sensor de Temperatura

O sensor de temperatura é uma tecnologia utilizada para identificar as variações de temperatura em máquinas, equipamentos e em ambientes. Os dados que o sensor fornece sobre a temperatura de determinada máquina ou equipamento são muito importantes para o monitoramento e manutenção deles. Assim, o índice de disponibilidade da máquina é maior.

Resumidamente, os sensores de temperatura ou sondas de temperatura, são dispositivos desenvolvidos para detectar a variação de temperatura em determinado meio.

Nesta seção será abordado um projeto com o sensor de temperatura LM35 que possui alta precisão e alta sensibilidade. Além disso, o sensor tem uma tensão de saída analógica, mede temperaturas na faixa de 0º a 100ºC com uma precisão de ±0,5ºC e a tensão de saída linear é de 10mV/ºC. Para cada 10mV de tensão na saída, representa 1ºC.

Resumidamente suas especificações e características são:

– Circuito integrado: LM35DZ – Tensão de operação: 4 a 20VDC – Corrente de operação: < 60mA – Faixa de medição: 0° a 100° celsius – Precisão: ±0,5° celsius – Sensibilidade: 10mV/ºC – Conexão de saída: analógica

Para obter mais informações técnicas deve-se procurar o datasheet do componente eletrônico.

O circuito eletrônico encontra-se na figura 3.1 e o código no exercício 3.1.

Conectar um LM35 ao Arduino é muito fácil, pois você só precisa conectar 3 pinos. Comece conectando o pino +VS à saída de 5V do Arduino e o pino GND ao aterramento.

Em seguida, conecte o pino do meio a qualquer uma das entradas analógicas do Arduino. Neste caso, usei o pino de entrada analógica *A*0.

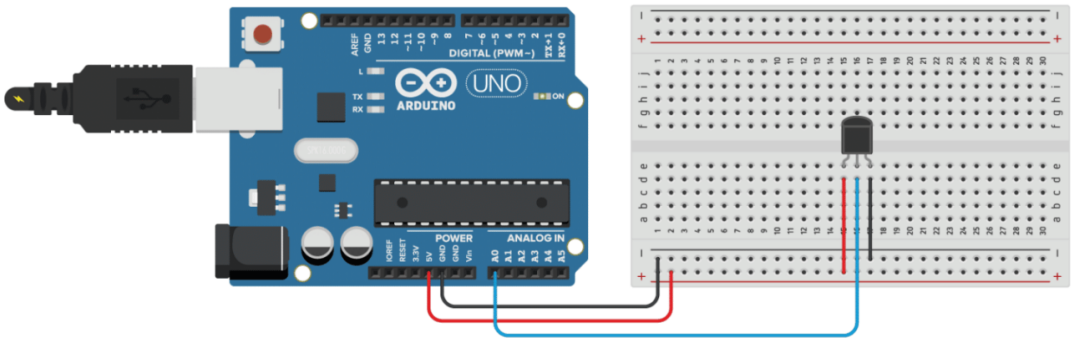


Figure 3.1: Circuito eletrônico do arduino e LM35.

Exercise 3.1 SENSOR DE TEMPERATURA

```
const int sensorPin = A0;     //PINO ANALÓGICO UTILIZADO PELO SENSOR
float temperatura = 0;     //VARIÁVEL DO TIPO FLOAT
void setup(){
   Serial.begin(9600);     //INICIALIZAÇÃO DA SERIAL
}
void loop(){
   temperatura = (analogRead(sensorPin) * 0.0048828125 * 100);     //VARIÁVEL RECEBE
A TEMPERATURA MEDIDA
   Serial.print("Temperatura = ");     //IMPRIME O TEXTO NA SERIAL
   Serial.print(temperatura);     //IMPRIME NA SERIAL A TEMPERATURA MEDIDA
   Serial.println(" C");     //IMPRIME O TEXTO NA SERIAL
   delay(2000);     //INTERVALO DE 2 SEGUNDOS
}
```

Você deve ver a seguinte saída no Monitor Serial (CTRL+SHIFT+M):

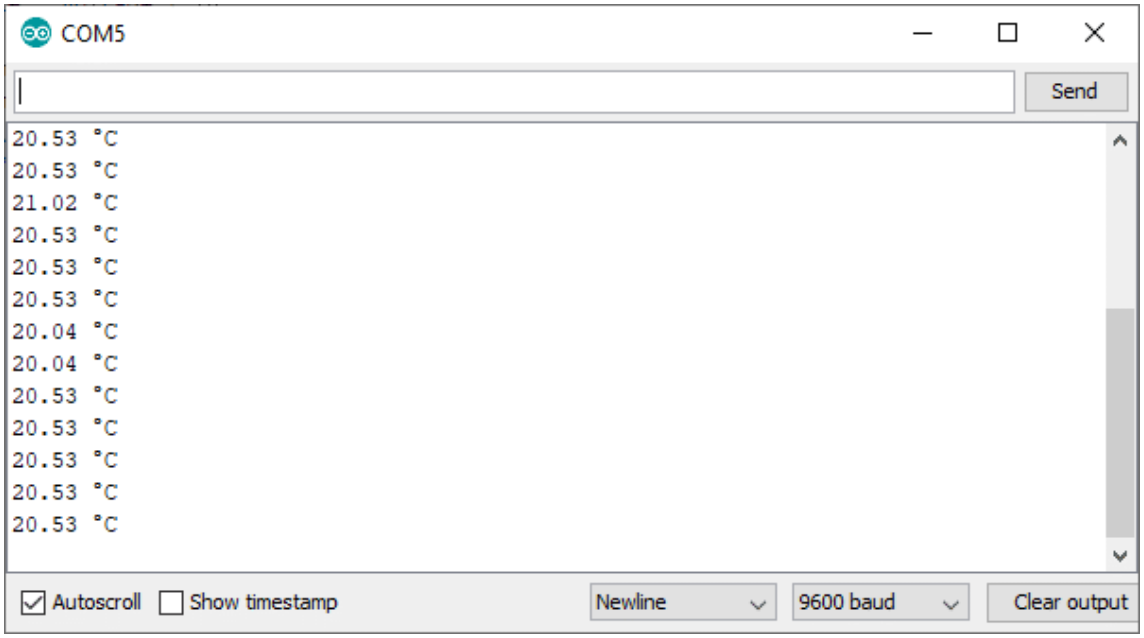


Figure 3.2: Monitoramento serial da temperatura.

Se você quiser fazer um termômetro autônomo que não precise de um computador, pode ser bom saber como exibir as leituras de temperatura em um display LCD.

3.2 Display LCD

O circuito eletrônico encontra-se na figura 3.3 e o código no exercício 3.2.

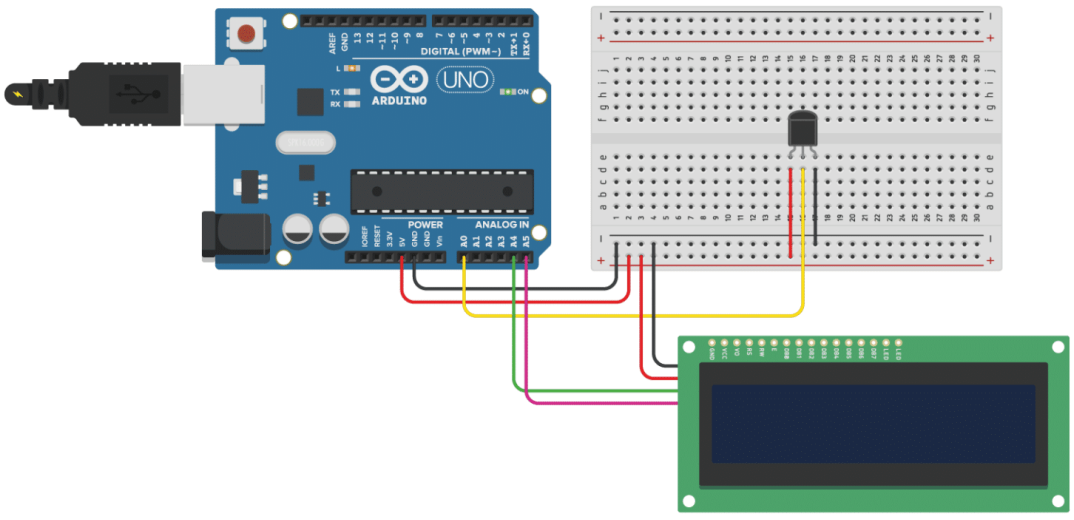


Figure 3.3: Circuito eletrônico do arduino, LM35 e LCD.

Observe que as conexões do LCD I2C são 4:
LCD –> Arduino
GND –> GND

VCC –> 5V
SDA –> A4
SCL –> A5
Para usar um LCD I2C, você precisa instalar a biblioteca LiquidCrystalI2C Arduino.

Exercise 3.2 SENSOR DE NO DISPLAY LCD

```
/* Sensor de temperatura analógico LM35 com LCD I2C */
// Incluir a biblioteca:
#include <LiquidCrystal.h>
// Crie um objeto LCD. Parâmetros: (RS, E, D4, D5, D6, D7):
LiquidCrystal lcd = LiquidCrystal(2, 3, 4, 5, 6, 7);
// Símbolo de grau:
byte Degree[] = {
   B00111,
   B00101,
   B00111,
   B00000,
   B00000,
   B00000,
   B00000,
   B00000
};
// Defina a qual pino do Arduino a saída do LM35 está conectada:
#define sensorPin A0
void setup() {
   // Inicializa o display LCD 16x2
   //lcd.begin (16, 2); para
   // Inicializa o display LCD 20x4
   lcd.begin (20, 4);
   // Liga a luz de fundo do LCD
   lcd.setBacklight(HIGH);
   // Crie um personagem personalizado:
   lcd.createChar(1, Degree);
}
void loop() {
   // Obtenha uma leitura do sensor de temperatura:
   int reading = analogRead(sensorPin);
   // Converta a leitura em tensão:
   float voltage = reading * (5000 / 1024.0);
   // Converta a tensão na temperatura em graus Celsius:
   float temperature = voltage / 10;
   // Imprime a temperatura no LCD;
   lcd.setCursor(0, 0);
   lcd.print("Temperatura:");
   lcd.setCursor(0, 1);
   lcd.print(temperature);
   lcd.write(1); // imprime o caractere personalizado
   lcd.print("C");
   delay(1000); // espere um segundo entre as leituras
}
```

3.3 Sensor de Umidade e Temperatura

O Sensor de Umidade e Temperatura – DHT11 é um dos componentes mais utilizados em projetos que envolva medição de temperatura e umidade ambiente. As características do sensor são:

– Modelo: DHT11 (Datasheet)
– Alimentação: 3,0 a 5,0 VDC (5,5 Vdc máximo)
– Corrente: 200uA a 500mA, em stand by de 100uA a 150 uA
– Faixa de medição de umidade: 20 a 90% UR
– Faixa de medição de temperatura: 0º a 50ºC
– Precisão de umidade de medição: ± 5,0% UR
– Precisão de medição de temperatura: ± 2.0 ºC
– Tempo de resposta: < 5s
– Dimensões: 23mm x 12mm x 5mm (incluindo terminais)

O circuito eletrônico encontra-se na figura 3.3 e o código no exercício 3.2.

O DHT11 possui 4 terminais sendo que somente 3 são usados: GND, VCC e Dados. Se desejar, pode-se adicionar um resistor pull up de 10K entre o VCC e o pino de dados.

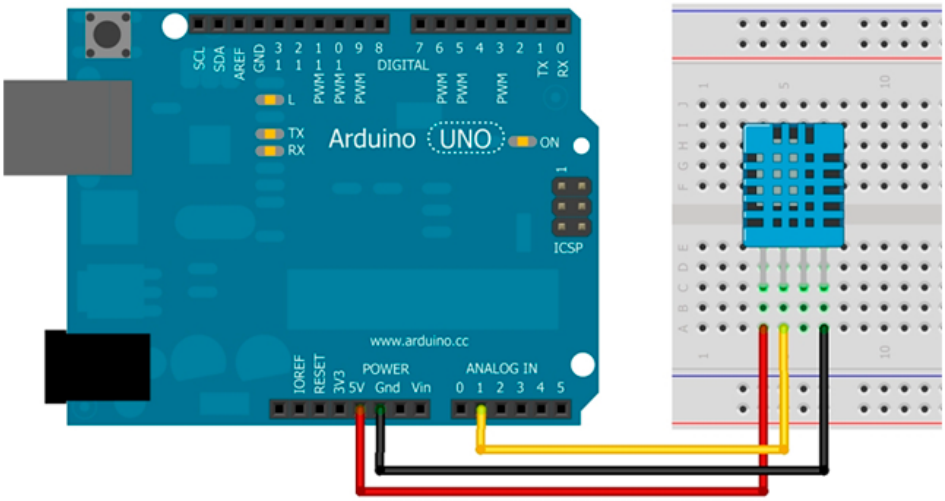


Figure 3.4: Circuito eletrônico do DHT11.

Conecte o pino de dados do DHT11 ao pino 2 do seu Arduino Uno como mostra o código exemplo abaixo, mas você poderá alterar por outro se desejar.

Para facilitar o seu trabalho já existe uma bilioteca que pode ser baixada. Após o download descompacte o arquivo .zip e mova-o para a pasta arduinosketchfolder/libraries/ e reinicie a IDE do Arduino. Não retire o arquivo dht.cpp. e não esqueça de renomear a pasta para “DHT”. Talvez será necessário criar uma sub-pasta da biblioteca caso não exista.

É comum acessar um código já existente seguindo o seguinte passo: Examples->DHT->DHTtester em sua IDE Arduino.

Exercise 3.3 SENSOR DE TEMPERATURA E UMIDADE

```
#include "DHT.h"
#define DHTPIN A1 // pino que estamos conectado
#define DHTTYPE DHT11 // DHT 11
```

```
// Conecte pino 1 do sensor (esquerda) ao +5V
// Conecte pino 2 do sensor ao pino de dados definido em seu Arduino
// Conecte pino 4 do sensor ao GND
// Conecte o resistor de 10K entre pin 2 (dados)
// e ao pino 1 (VCC) do sensor
DHT dht(DHTPIN, DHTTYPE);
void setup()
{
   Serial.begin(9600);
   Serial.println("DHTxx test!");
   dht.begin();
}
void loop()
{
   // A leitura da temperatura e umidade pode levar 250ms!
   // O atraso do sensor pode chegar a 2 segundos.
   float h = dht.readHumidity();
   float t = dht.readTemperature();
   // testa se retorno é valido, caso contrário algo está errado.
   if (isnan(t) || isnan(h))
   {
      Serial.println("Failed to read from DHT");
   }
   else
   {
      Serial.print("Umidade: ");
      Serial.print(h);
      Serial.print("
      Serial.print("Temperatura: ");
      Serial.print(t);
      Serial.println(" *C");
   }
}
```

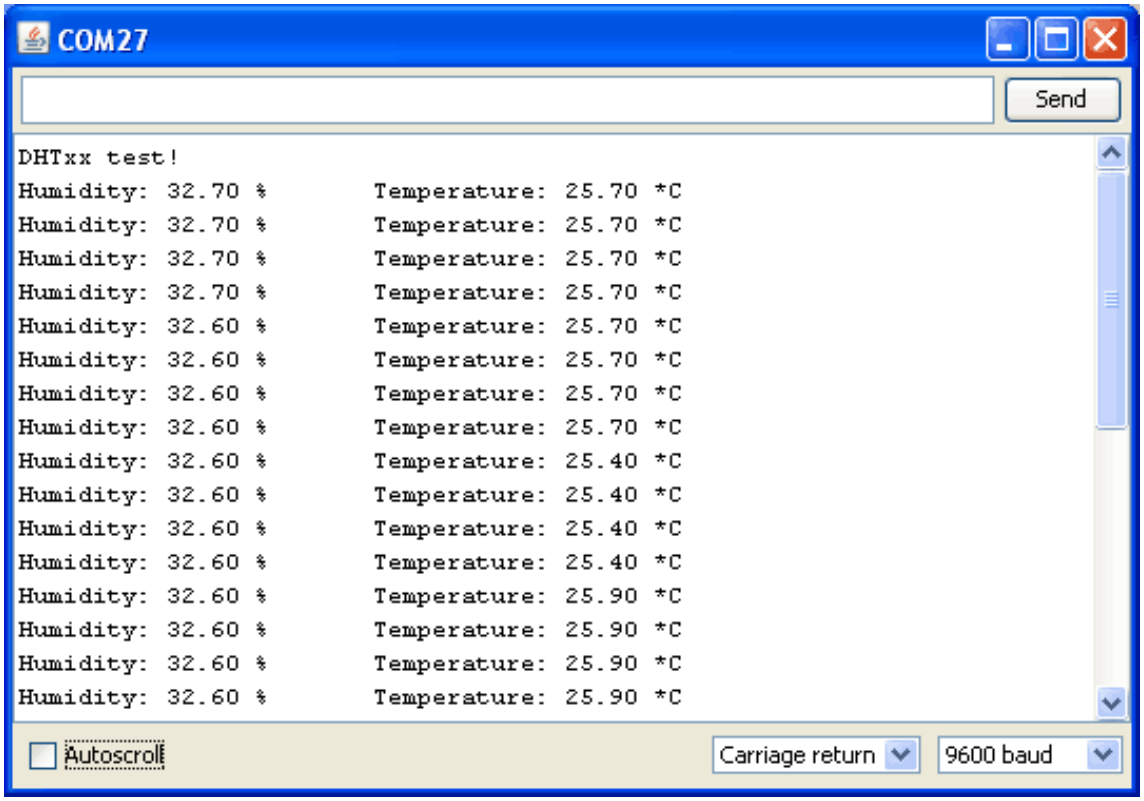


Figure 3.5: Monitoramento serial.

3.4 Sensor de Umidade do solo - higrômetro

Projeto Sensor de Umidade do Solo Arduino é amplamente utilizado com microcontroladores, podendo ser aplicado em sistemas de controle de irrigação, monitorando o nível de umidade da terra no local em que está instalado, vindo a permitir que dispositivos eletrônicos controlem a reposição da água necessária.

É indispensável saber a condição de quantidade de água no solo, essa umidade influencia em construções, plantações e até mesmo na irrigação de um jardim, mas como a umidade do solo costuma ser muito variável, precisamos efetuar medições constantes nesse meio.

Este projeto é muito simples e tem como objetivo automatizar as necessidades específicas de cada pessoa, seja ela a condição do solo para plantação de árvores e plantas em geral até a manutenção e necessidade de irrigação de um jardim. Com sua interface e o uso do Sensor de Umidade de Solo, sua montagem e programação tornam o projeto ainda mais fácil, trabalhando com 2 tipos de comunicação, digital e analógica.

O sensor de umidade do solo consiste em 2 partes: uma sonda que entra em contato com o solo, e um pequeno módulo contendo um chip comparador LM393 (verificar no datasheet), que vai ler os dados que vêm do sensor e enviá-los para o microcontrolador, no nosso caso, um Arduino Uno. Como saída, temos um pino D0, que fica em nível 0 ou 1 dependendo da umidade, e um pino de saída analógica (A0), que possibilita monitorar com maior precisão usando uma porta analógica do microcontrolador.

O módulo tem um led que indica quando a placa está sendo alimentada corretamente, e outro que acende quando a saída digital for acionada. A sensibilidade do módulo é ajustada por meio do potenciômetro existente na placa.

O circuito eletrônico encontra-se na figura 3.6 e o código no exercício 3.4.

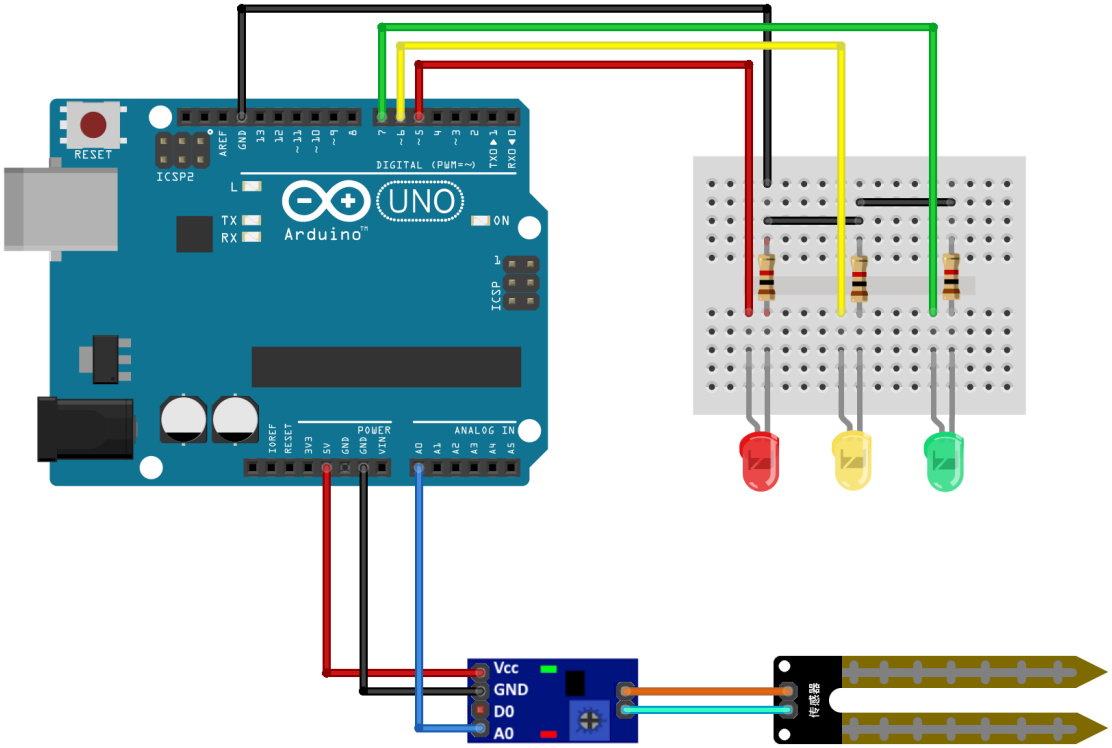


Figure 3.6: Circuito eletrônico do higrômetro.

Exercise 3.4 HIGRÔMETRO

```
#define pino_sinal_analogico A0
#define pino_led_vermelho 5
#define pino_led_amarelo 6
#define pino_led_verde 7
int valor_analogico;
void setup()
{
   Serial.begin(9600);
   pinMode(pino_sinal_analogico, INPUT);
   pinMode(pino_led_vermelho, OUTPUT);
   pinMode(pino_led_amarelo, OUTPUT);
   pinMode(pino_led_verde, OUTPUT);
}
void loop() {     //Le o valor do pino A0 do sensor
   valor_analogico = analogRead(pino_sinal_analogico);
   //Mostra o valor da porta analogica no serial monitor
   Serial.print("Porta analogica: ");
   Serial.print(valor_analogico);
   //Solo umido, acende o led verde
   if (valor_analogico > 0 && valor_analogico < 400)
   {
      Serial.println(" Status: Solo umido");
      apagaleds();
      digitalWrite(pino_led_verde, HIGH);
```

```
    }
  //Solo com umidade moderada, acende led amarelo
    if (valor_analogico > 400 && valor_analogico < 800)
    {
      Serial.println(" Status: Umidade moderada");
      apagaleds();
      digitalWrite(pino_led_amarelo, HIGH);
    }
  //Solo seco, acende led vermelho
    if (valor_analogico > 800 && valor_analogico < 1024)
    {
      Serial.println(" Status: Solo seco");
      apagaleds();
      digitalWrite(pino_led_vermelho, HIGH);
    }
    delay(100);
  }
  void apagaleds()
  {
    digitalWrite(pino_led_vermelho, LOW);
    digitalWrite(pino_led_amarelo, LOW);
    digitalWrite(pino_led_verde, LOW);
  }
```

3.5 Atuadores

3.5.1 Micro Servo Motor

Um Servo Motor comum é um motor que nos possibilita o controle de sua posição. Acompanha um cabo de 3 pinos referente a alimentação/controle e diversos acessórios, conforme figura 3.7,

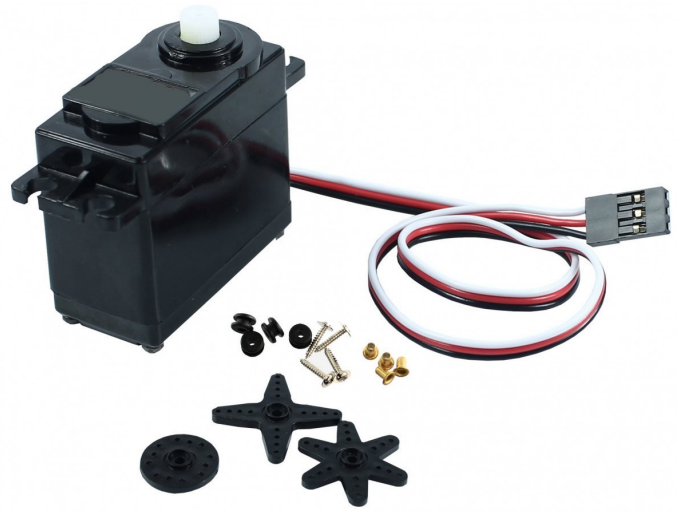


Figure 3.7: Servo Motor.

O circuito eletrônico encontra-se na figura 3.8 e o código no exercício 3.5.

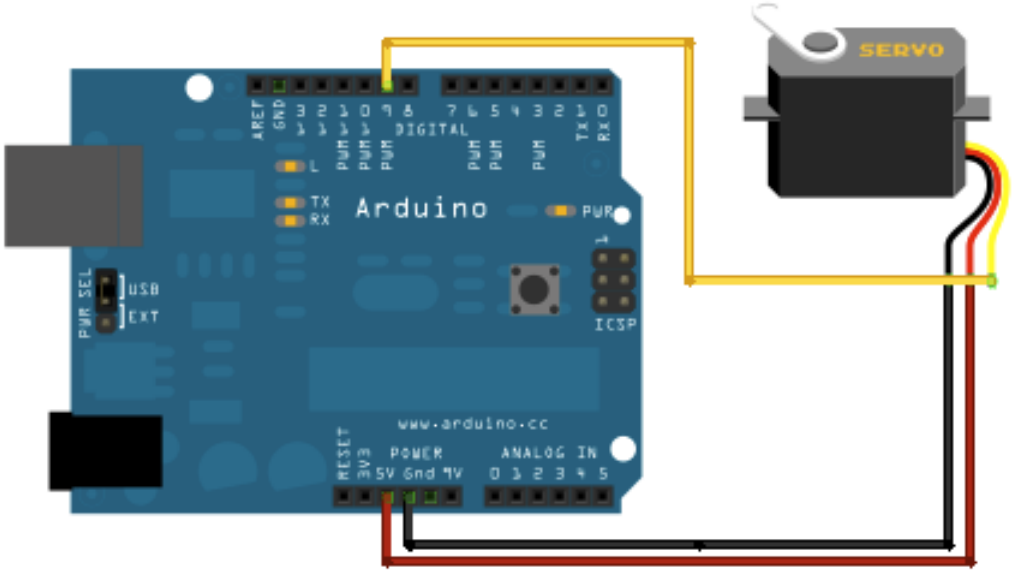


Figure 3.8: Circuito eletrônico do Micro Servo Motor.

Exercise 3.5

```
#include <Servo.h>
#define SERVO 6     // Porta Digital 6 PWM
Servo s;     // Variável Servo
int pos;     // Posição Servo
void setup ()
{
   s.attach(SERVO);
   Serial.begin(9600);
   s.write(0);     // Inicia motor posição zero
}
void loop()
{
   for(pos = 0; pos < 90; pos++)
   {
      s.write(pos);
      delay(15);
   }
   delay(1000);
   for(pos = 90; pos >= 0; pos–)
   {
      s.write(pos);
      delay(15);
   }
}
```

A partir da manipulação deste atuador é possível o aluno iniciar atividades mais complexas tais como na figura

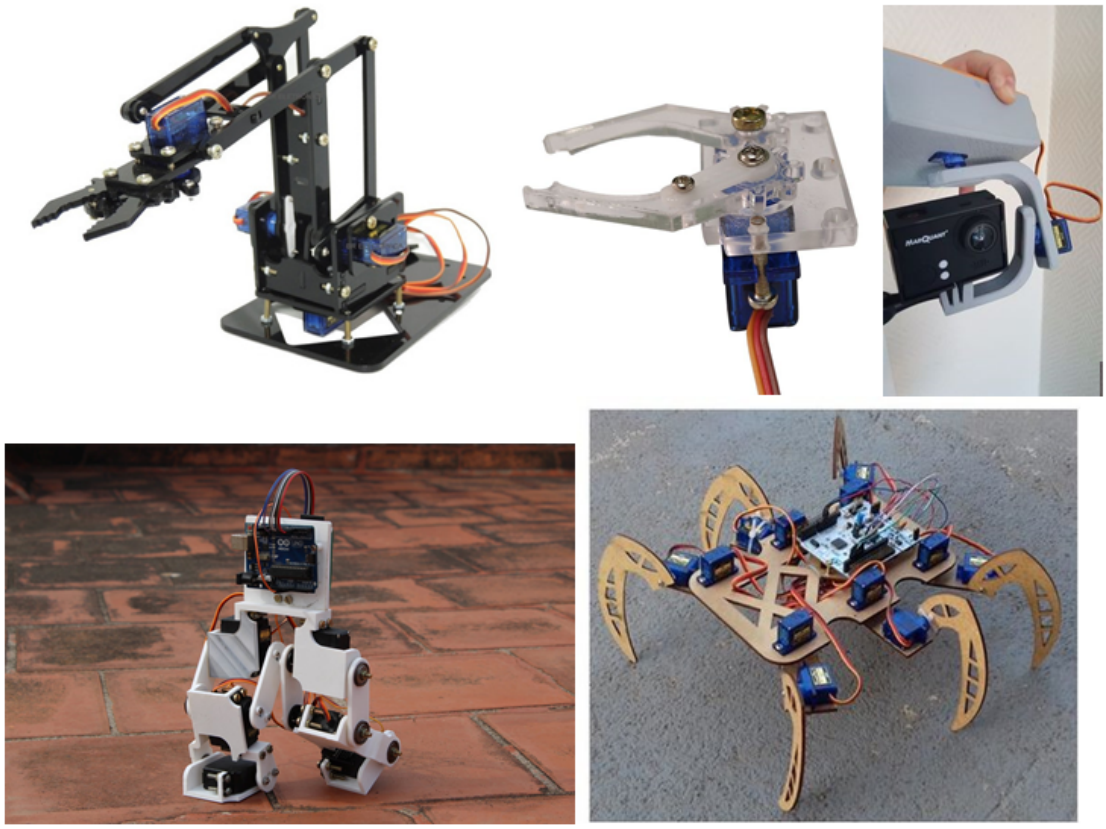


Figure 3.9: Sugestões de atividades com os micro servo motores.

3.5.2 Motor DC e ponte H

Nesta seção será mostrada o controle de até 2 motores DC ou 1 motor de passo com este módulo Ponte H L298N Arduino. Esse módulo é projetado para controlar cargas indutivas como relés, solenóides, motores DC e motores de passo, permitindo o controle não só do sentido de rotação do motor, como também da sua velocidade, utilizando os pinos PWM do Arduino.

As especificações básicas da ponte H são:

Tensão de Operação: 4 35v

Chip: ST L298N (Datasheet)

Controle de 2 motores DC ou 1 motor de passo

Corrente de Operação máxima: 2A por canal ou 4A max

Tensão lógica: 5v

Corrente lógica: 0 36mA

Limites de Temperatura: -20 a +135°C

Potência Máxima: 25W

Dimensões: 43 x 43 x 27mm

Peso: 30g

A figura 3.10 demonstra o funcionamento da Ponte H L298N.

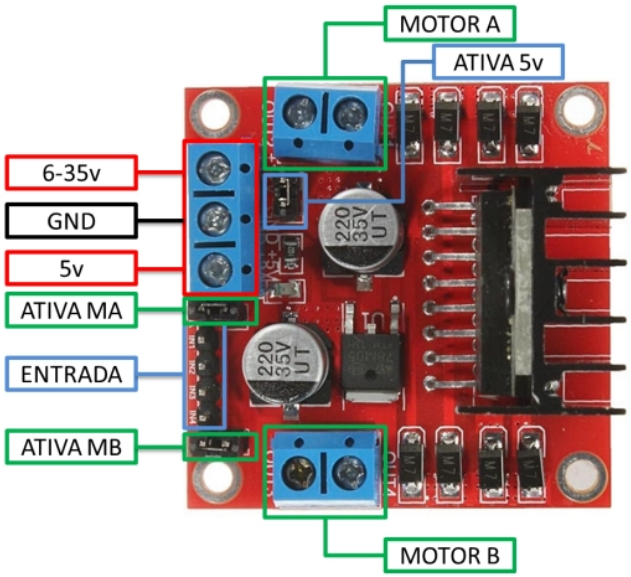


Figure 3.10: Motor DC e ponte H, 2 motores DC 5V.

(Motor A) e (Motor B) se referem aos conectores para ligação de 2 motores DC ou 1 motor de passo

(Ativa MA) e (Ativa MB) – são os pinos responsáveis pelo controle PWM dos motores A e B. Se estiver com jumper, não haverá controle de velocidade, pois os pinos estarão ligados aos 5v. Esses pinos podem ser utilizados em conjunto com os pinos PWM do Arduino

(Ativa 5v) e (5v) – Este Driver Ponte H L298N possui um regulador de tensão integrado. Quando o driver está operando entre 6-35V, este regulador disponibiliza uma saída regulada de +5v no pino (5v) para um uso externo (com jumper), podendo alimentar por exemplo outro componente eletrônico. Portanto não alimente este pino (5v) com +5v do Arduino se estiver controlando um motor de 6-35v e jumper conectado, isto danificará a placa. O pino (5v) somente se tornará uma entrada caso esteja controlando um motor de 4-5,5v (sem jumper), assim poderá usar a saída +5v do Arduino.

(6-35v) e (GND) – Aqui será conectado a fonte de alimentação externa quando o driver estiver controlando um motor que opere entre 6-35v. Por exemplo se estiver usando um motor DC 12v, basta conectar a fonte externa de 12v neste pino e (GND).

(Entrada) – Este barramento é composto por IN1, IN2, IN3 e IN4. Sendo estes pinos responsáveis pela rotação do Motor A (IN1 e IN2) e Motor B (IN3 e IN4).

A tabela abaixo mostra a ordem de ativação do Motor A através dos pinos IN1 e IN2. O mesmo esquema pode ser aplicado aos pinos IN3 e IN4, que controlam o Motor B.

Table 3.1: Ativação do motor

Motor	IN1	IN2
Horário	5V	GND
Anti-Horário	GND	5V
Ponto Morto	GND	GND
Freio	5V	5V

Vamos mostrar dois esquemas de ligação deste módulo ao Arduino Uno, 3.11 e 3.12, que

utilizarão o mesmo programa mostrado no exercício 3.6.

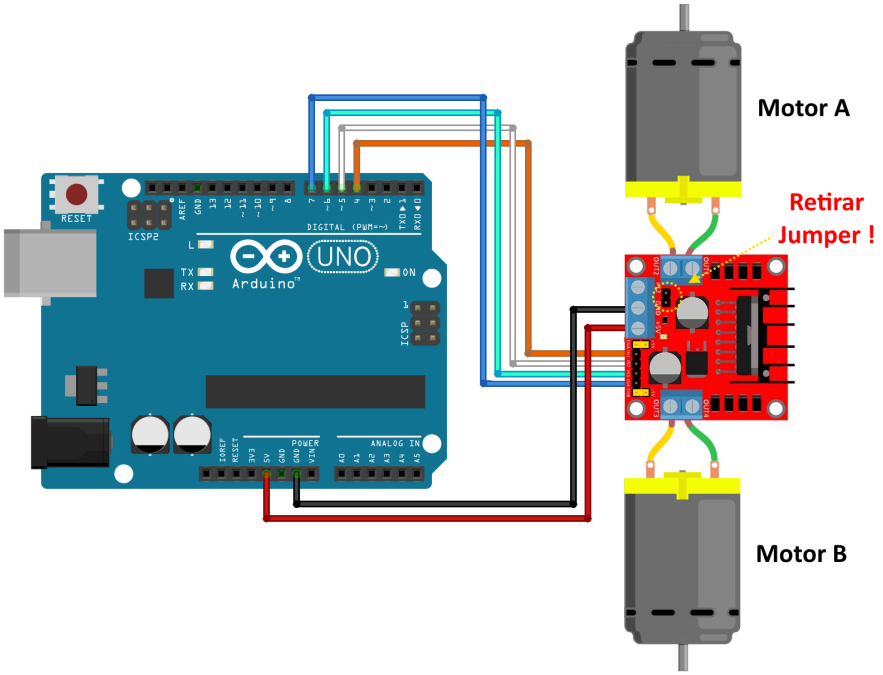


Figure 3.11: Motor DC e ponte H, 2 motores DC 5V.

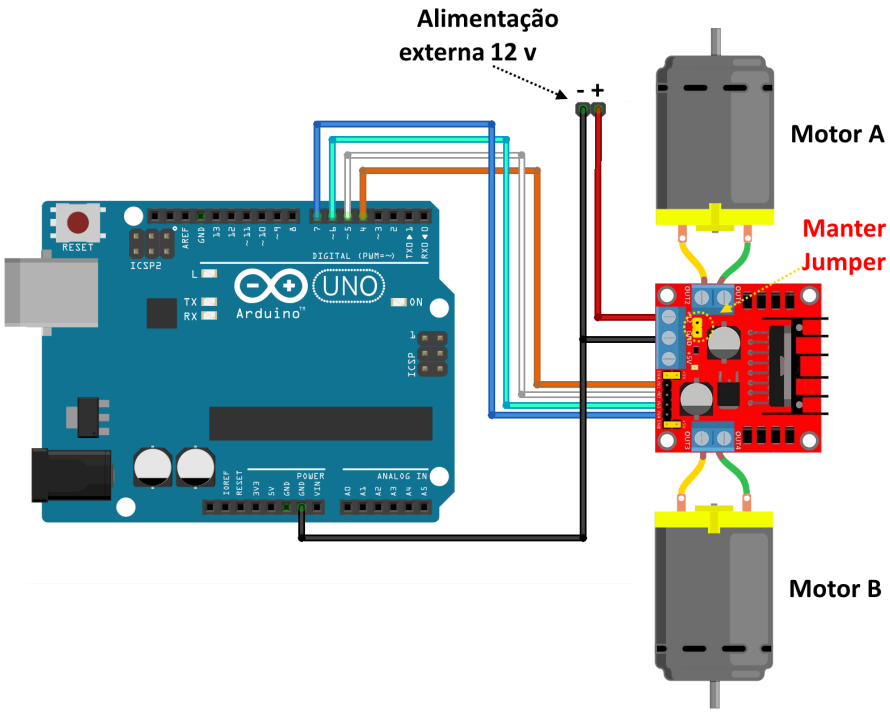


Figure 3.12: Motor DC e ponte H, 2 motores DC de 12V.

Teste o seu módulo carregando o exercício 3.6, que vai servir para os 2 circuitos que mostramos anteriormente. O programa gira o motor A no sentido horário, depois desliga esse motor e gira o motor B no mesmo sentido. Depois, repete esse procedimento no sentido anti-horário.

Exercise 3.6 //Definicoes pinos Arduino ligados a entrada da Ponte H

```
int IN1 = 4;
int IN2 = 5;
int IN3 = 6;
int IN4 = 7;
void setup()
{
  //Define os pinos como saida
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}
void loop()
{
  //Gira o Motor A no sentido horario
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  delay(2000);
  //Para o motor A
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  delay(500);
  //Gira o Motor B no sentido horario
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  delay(2000);
  //Para o motor B
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  delay(500);
  //Gira o Motor A no sentido anti-horario
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  delay(2000);
  //Para o motor A
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  delay(500);
  //Gira o Motor B no sentido anti-horario
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  delay(2000);
  //Para o motor B
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  delay(500);
}
```

3.5.3 Buzzer

O buzzer nada mais é do que um pequeno alto-falante capaz de emitir sons em diversas frequências. O buzzer é normalmente é usado em projetos que necessitam de avisos sonoros, relógios com alarme, e até para reproduzir músicas.

Vamos mostrar o esquema de ligação deste módulo ao Arduino Uno, 3.13, que utilizará o algoritmo mostrado no exercício 3.7. Os valores do buzzer é de 5 volts e o do resistor de 100 ohms.

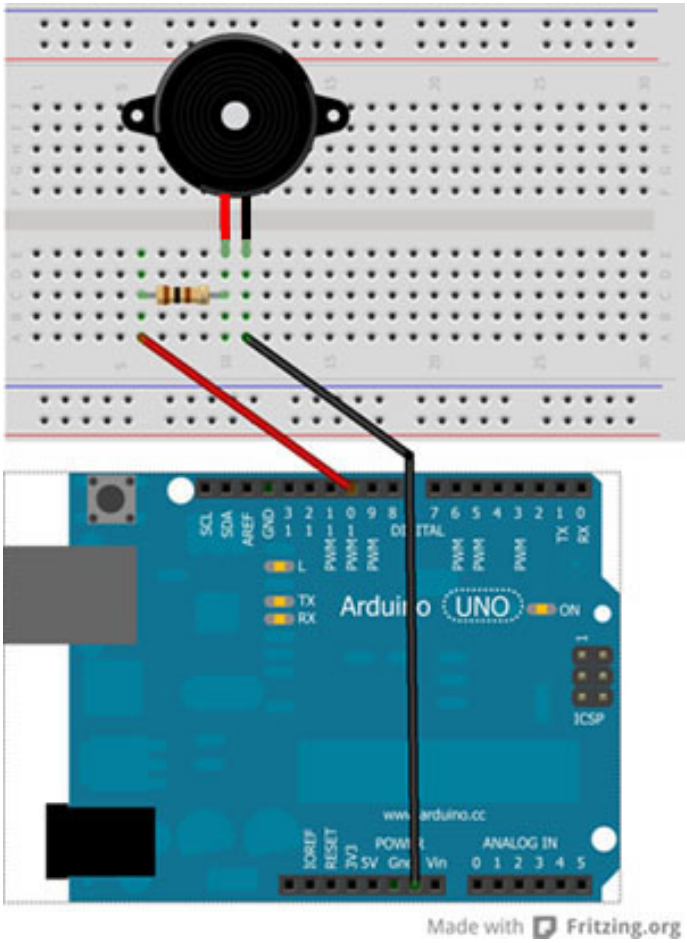


Figure 3.13: Buzzer.

Exercise 3.7

```
//Constante que representa o pino onde o positivo
//do buzzer será ligado.
const int buzzer = 10;
//Método setup, executado uma vez ao ligar o Arduino.
void setup() {
   //Definindo o pino buzzer como de saída.
   pinMode(buzzer,OUTPUT);
}
//Método loop, executado enquanto o Arduino estiver ligado.
void loop() {
   //Ligando o buzzer com uma frequencia de 1500 hz.
   tone(buzzer,1500);
   delay(500);
```

```
    //Desligando o buzzer.
    noTone(buzzer);
    delay(500);
}
```

3.6 Eletrônica

Nesta seção serão abordados conhecimentos básicos de diodo, capacitor e multímetro. Cabe ao aluno procurar no decorrer das atividades de robótica ampliar seu conhecimento.

Este conhecimento possibilita o aluno expandir sua criatividade construindo robôs com componentes eletrônicos novos ou reaproveitados de outros equipamentos e de sucata.

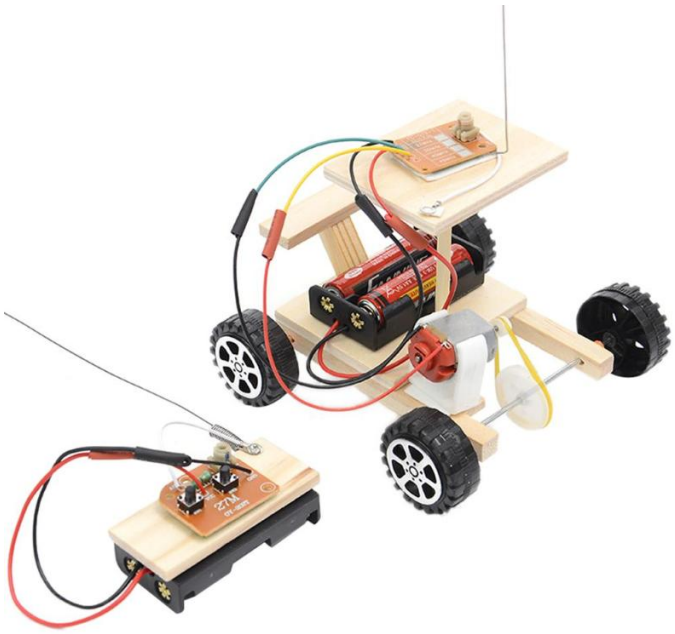


Figure 3.14: Reaproveitamento de materiais na robótica.

3.6.1 Diodos

O diodo é um componente semicondutor passivo (passivo=consome parte da energia para trabalhar) composto de silício ou germânio dopados com diferentes materiais durante seu desenvolvimento, como exemplo tem o da figura 3.15.

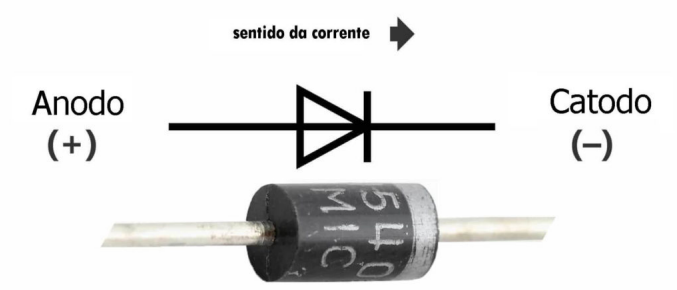


Figure 3.15: Diodo.

Seu consumo para trabalho (uma especie de pedágio, por isso é passivo) é de aproximadamente 0,3V para Germânio e 0,7V para Silício.

Sua principal característica é que ele permite a passagem de corrente com facilidade somente em um sentido que é entre o ânodo para o catodo conforme a figura abaixo.

Para ficar mais fácil gravar o seu funcionamento, é só observar que ele parece com uma seta, onde indica o sentido, já no oposto é possível ver uma "barreira", ou seja, dificuldade de passagem de elétrons.

Polarizado inversamente o diodo não permite passagem de corrente, ou seja, é um circuito aberto, servindo como um bom inibidor de corrente reversa, sem corrente, sem tensão. Diretamente, depois do "pedágio" de 0,3V no caso do Germânio ou 0,7V no caso do Silicio, seu funcionamento é como um circuito fechado, ou seja, "livre" passagem de corrente. Na figura 3.16 mostra a curva do diodo que pode ser observada em seu datasheet.

Figure 3.16: Curva do diodo.

3.6.2 Capacitores

Um dos componentes que figuram entre os mais utilizados dentro da eletrônica são os capacitores, que são capazes de armazenar energia na forma de campo elétrico no seu processo de carga, liberando essa energia no processo de descarga.

São largamente utilizados para diversos fins, tanto em aplicações de corrente contínua, como temporizadores, retificadores e em corrente alternada para correção do fator de potência, filtros passivos, entre outros.

Como os capacitores são usados basicamente para armazenar energia, alguns cuidados devem ser tomados para garantir que trabalhem sempre dentro das suas especificações evitando sobreaquecimento que em alguns casos pode até a causar explosão do componente.

Basicamente, o capacitor consiste em duas placas metálicas condutoras separadas por um isolante dielétrico, sendo que esse dielétrico pode ser encarado como um material isolante qualquer, que em alguns casos dá nome ao capacitor (capacitores cerâmicos, de mica, de poliéster, etc).

Figure 3.17: Capacitores.

Todo capacitor tem um parâmetro denominado capacitância cuja unidade é o Farad (F), que determina quanta carga ele é capaz de armazenar.

Como 1 Farad (1F) é considerado uma capacitância muito grande, o mais comum é vermos componentes com subunidades do Farad, como microFarad (uF), nanoFarad (nF) ou mesmo picoFarad (pF).

A tendência é de quanto maior a capacitância, maior as dimensões do capacitor, aumentando também os cuidados em seu manuseio. Há também uma tensão máxima impressa no capacitor, essencial para garantir a isolação do dielétrico e manter o funcionamento do dispositivo.

Outra característica importante dos capacitores é o seu processo de carga e descarga, que garante que a energia acumulada seja descarregada um tempo depois em outra parte do circuito.

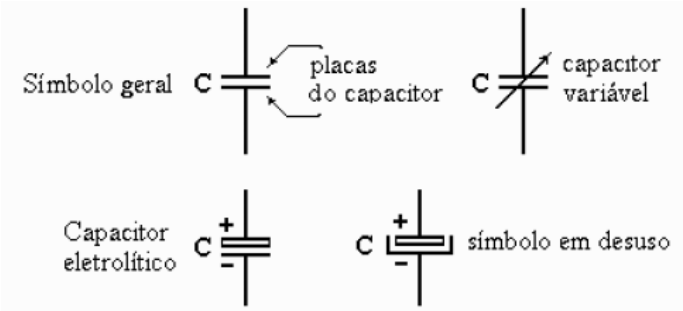


Figure 3.18: Símbolo dos capacitores.

No mercado há diversos tipos de capacitores, sendo os mais comuns: cerâmico, poliéster e eletrolítico.

3.6.2.1 Capacitor Cerâmico

São capacitores apolares, cujo dielétrico é feito de cerâmica. Geralmente possuem um encapsulamento de esfera achatada. Como tratam-se em sua maioria de capacitores muito pequenos, usa-se com uma codificação especial para obter seu valor nominal de capacitância.

Um capacitor cerâmico tem 3 algarismos na sua carcaça, sendo os dois primeiros significativos e o ultimo um multiplicador de base 10. Veja a figura 3.19

Figure 3.19: Capacitor Cerâmico.

Para obter o valor do capacitor, considera-se os dois primeiros algarismos e acrescenta-se tantos zeros quanto forem indicados no terceiro algarismo. Nesse caso, temos 12.000 pF, ou seja, 12 nF de capacitância.

3.6.2.2 Capacitores de Poliéster

São também apolares, geralmente maiores que os de cerâmica e com a capacitância já impressa na resina externa.

Cores	1ª Faixa	2ª Faixa	3ª Multiplicador	4ª Tolerância	5ª Tensão Nominal
PRETO	0	0	-	± 20%	-
MARROM	1	1	x 10	-	-
VERMELHO	2	2	x 100	-	250V
LARANJA	3	3	x 1000	-	-
AMARELO	4	4	x 10000	-	400V
VERDE	5	5	x 100000	-	-
AZUL	6	6	-	-	630V
VIOLETA	7	7	-	-	-
CINZA	8	8	-	-	-
BRANCO	9	9	-	± 10%	-

Figure 3.20: Capacitor Poliéster.

3.6.2.3 Capacitor Eletrolítico

Usado em circuitos de corrente contínua, o capacitor eletrolítico é polarizado, ou seja, há um terminal específico para o positivo e outo para o negativo dentro do circuito. Nesse tipo de capacitor, as informações mais importantes estão impressas na capa plástica que envolve o dispositivo.

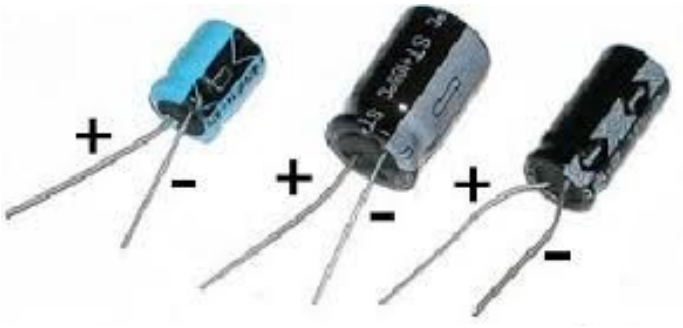


Figure 3.21: Capacitor Eletrolítico.

Inverter essa polarização pode ser perigoso, já que o capacitor corre o risco de sofrer danos, e até explodir, mesmo em condições que seriam adequadas para seu funcionamento na polarização correta.

3.6.3 Multímetros

O Multímetro nada mais é do que um equipamento utilizado para fazer a medição da resistência elétrica, tensão ou corrente contínua e da tensão ou corrente alternada.

Com este aparelho você também consegue medir a capacitância, temperatura, frequência de sinais alternados, dentre outras grandezas elétricas.

Para estabelecer qual é o tipo de medição que deve ser realizada com o Multímetro, basta ficar acionar uma chave rotativa que existe neste aparelho e escolher a medição que deseja fazer.

Atualmente existem dois tipos básicos de Multímetro: Analógico e Digital.

– Analógico Esse é um tipo de multímetro que apresenta um mostrador com diversas escalas graduadas, verificar figura 3.22.

Ele realiza a verificação da leitura utilizando a força eletromagnética existente no seu ponteiro, para interpretar o valor da grandeza que está sendo medida, ou seja, se é resistência, tensão ou mesmo corrente.

– Digital O multímetro digital ou amplificador operacional como também é conhecido no mercado, corresponde a um aparelho de medição mais versátil, que possui um visor digital de cristal líquido e que é muito usado em laboratórios, serviços de campo ou na indústria.

Esse aparelho engloba diferentes instrumentos de medição como o amperímetro, voltímetro, capacímetro, termômetro, ohmímetro, dentre outros.

Figure 3.22: Multímetro analógico e digital.

Pode até não parecer, mas este pequeno aparelho apresenta inúmeras aplicações, conforme figura 3.23.

No carro por exemplo, este aparelho pode ser usar para testar fusíveis, relés, baterias, bonina de ignição, sistema de som e muitas outras coisas.

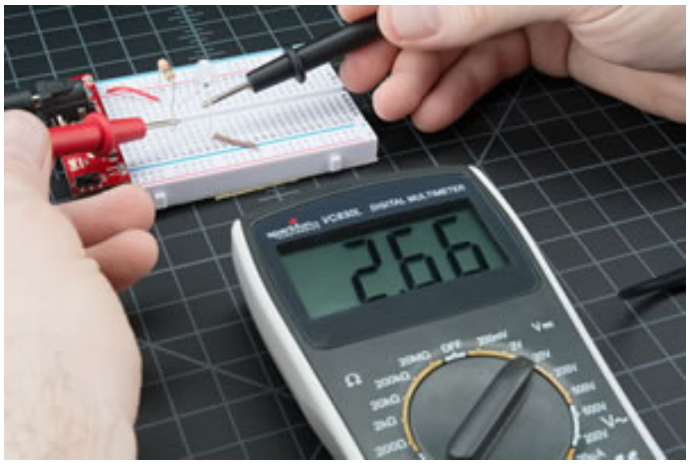
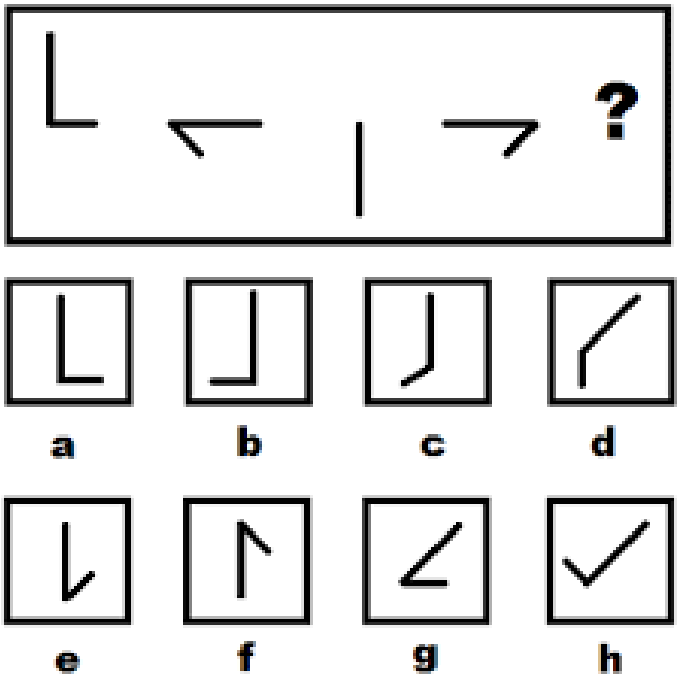


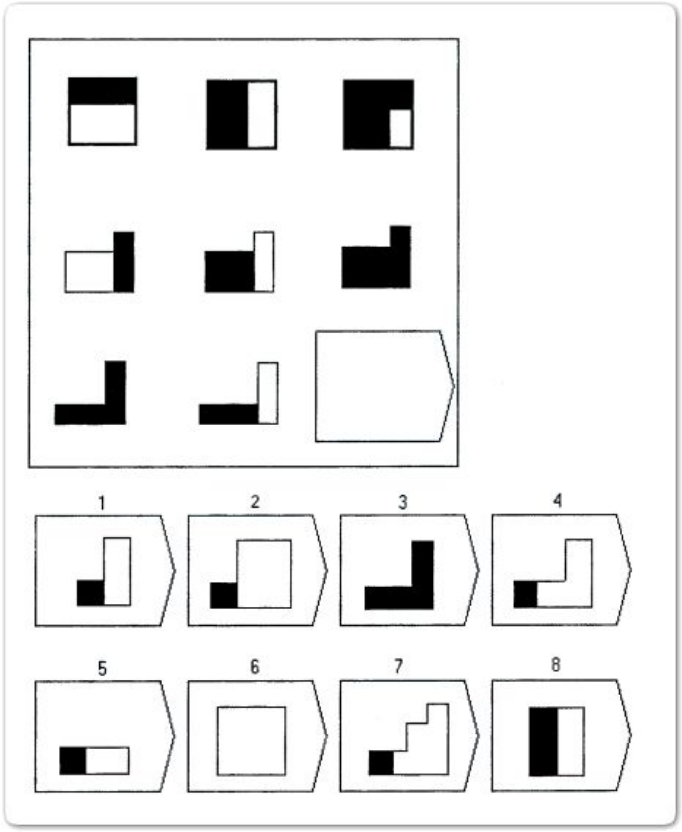
Figure 3.23: Multímetro.

3.7 Lógica

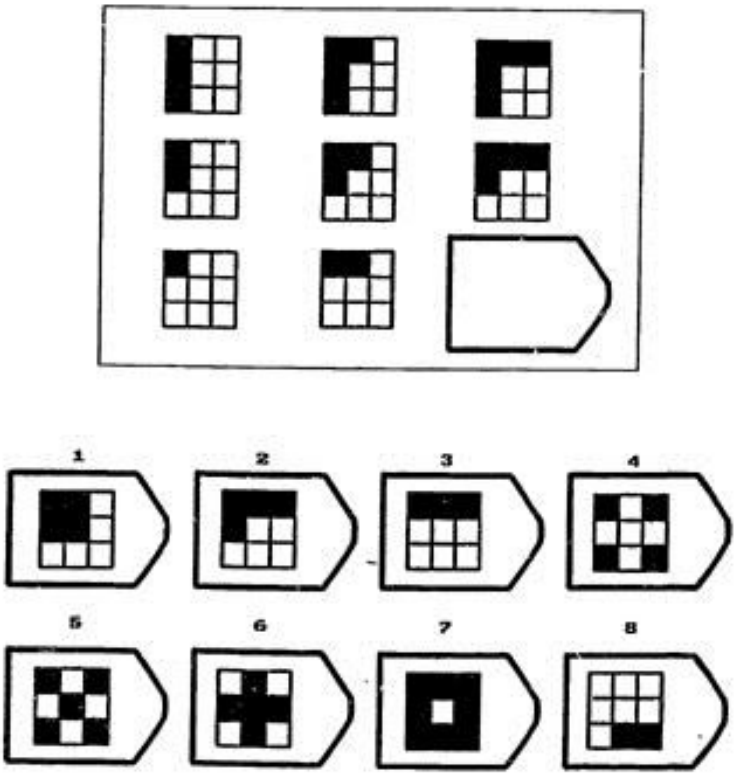
01)



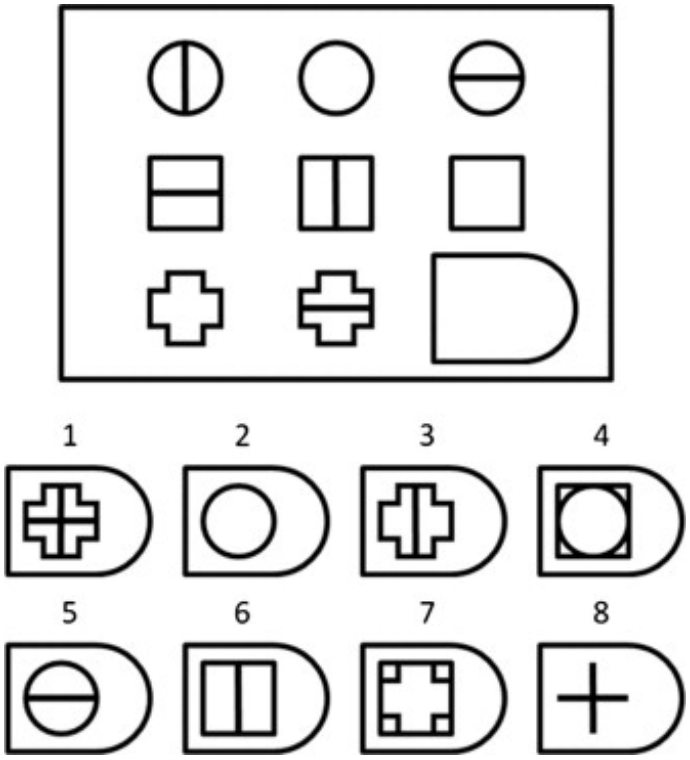
02)



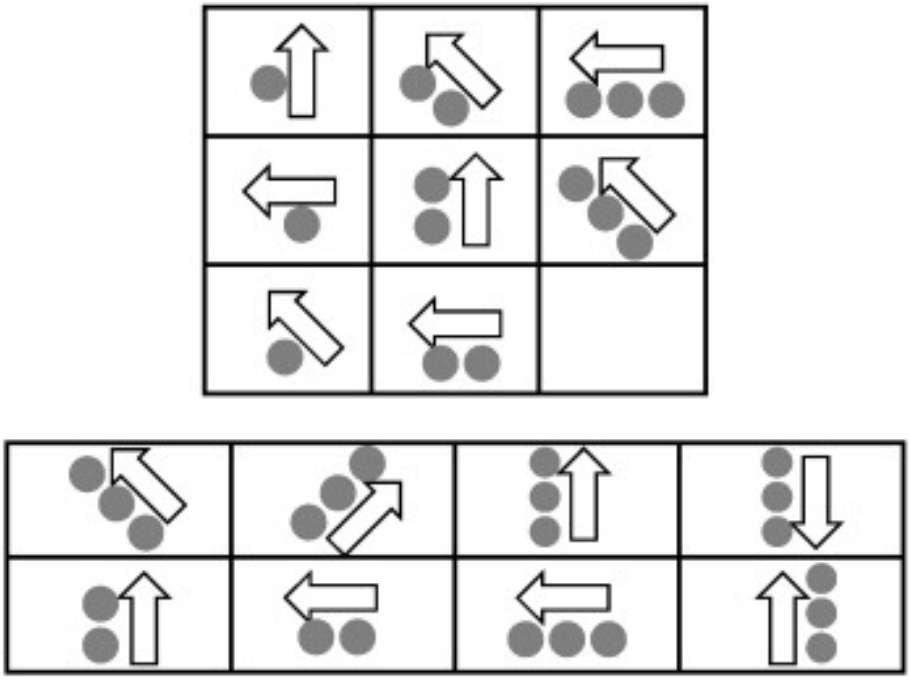
03)



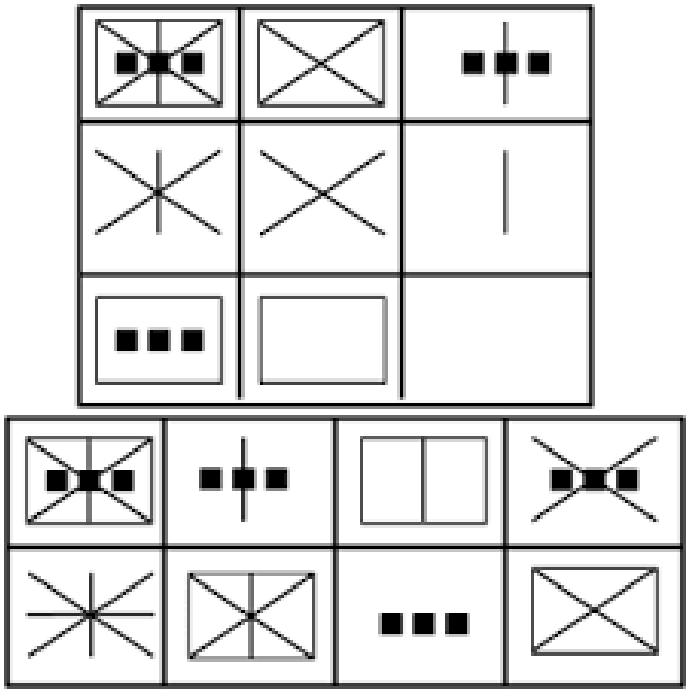
04)



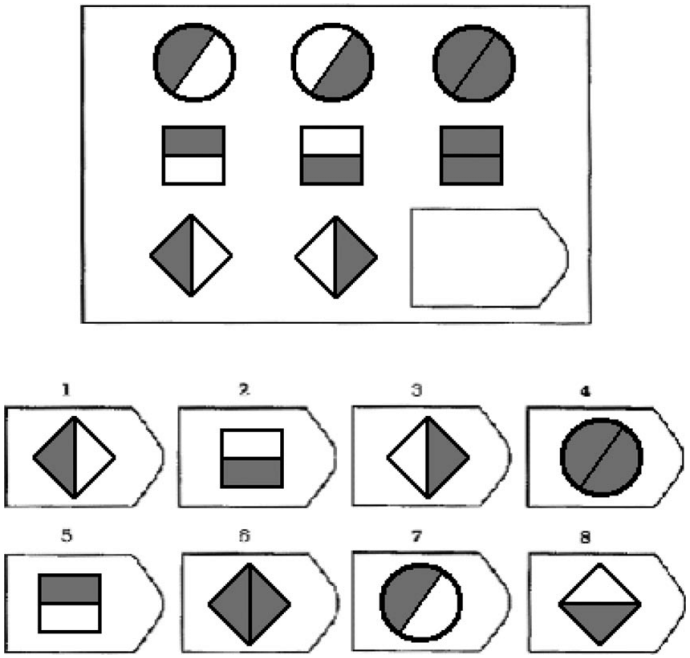
05)



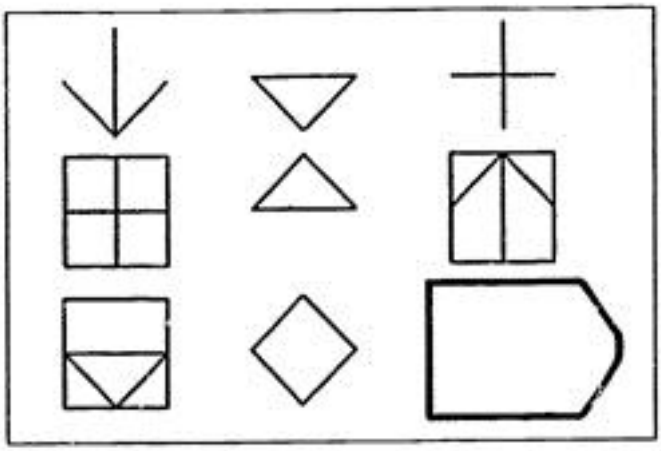
06)

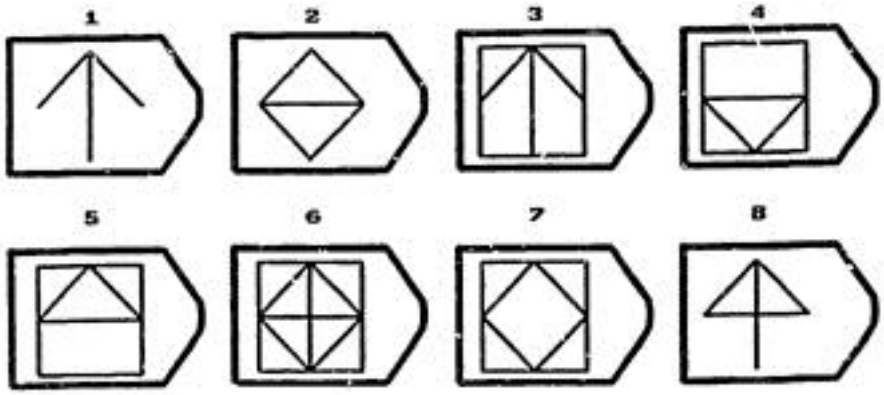


07)

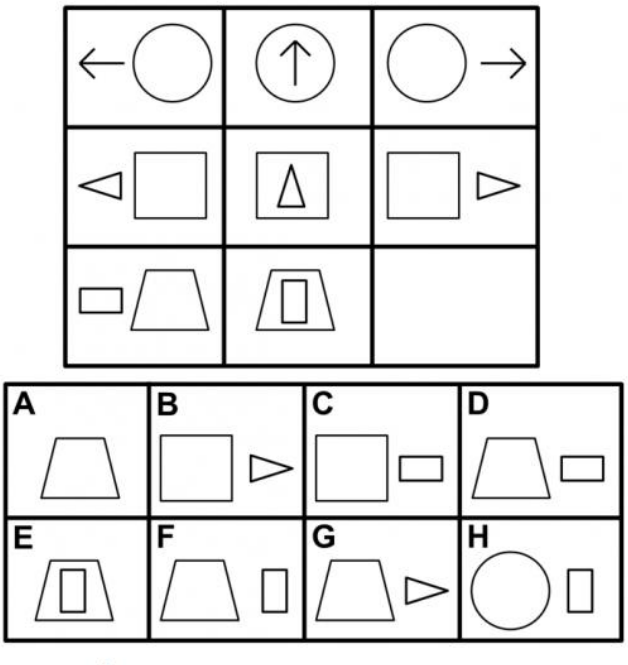


08)

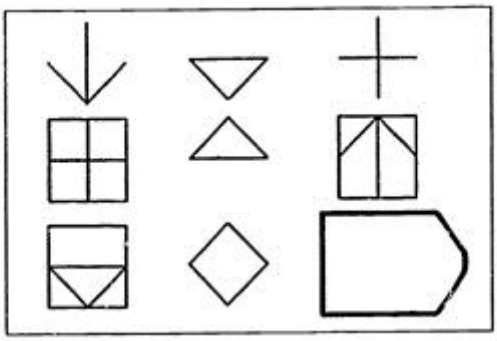


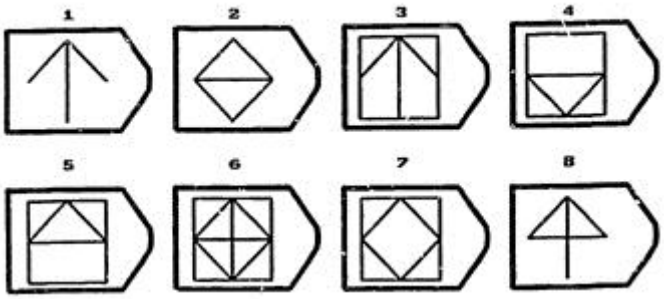


09)



10)




1
2
3
4
5
6
7
8

4. Lógica de Programação - Comandos de repetição e decisão

4.1 Comandos de decisão

Agora, estudaremos as estruturas de repetição, que permitem executar mais de uma vez um mesmo trecho de código. Trata-se de uma forma de executar blocos de comandos somente sob determinadas condições, mas com a opção de repetir o mesmo bloco quantas vezes for necessário. As estruturas de repetição são úteis, por exemplo, para repetir uma série de operações semelhantes que são executadas para todos os elementos de uma lista ou de uma tabela de dados, ou simplesmente para repetir um mesmo processamento até que uma certa condição seja satisfeita.

4.1.1 Comando If-Else

Uma estrutura de decisão if-lse examina uma ou mais condições e decide quais instruções serão executadas dependendo se a condição foi ou não foi.

4.1.1.1 Comando If

O comando if é uma estrutura de decisão muito utilizada.

Em pseudocódigo teríamos algo assim:

Se
(condição)
então
Início
Instruções;
Fim;

Em linguagem C o código equivalente para essa estrutura de decisão é:

```
if (condição)
  {
    instrução ou instruções para condição verdadeira;
  }
```

Exemplo de programa utilizando estrutura de decisão: Elaborar um programa em linguagem C para somar dois números inteiros e mostrar o valor da soma na tela. Caso a soma dos números seja maior que 10 mostrar uma mensagem informativa na tela.

Exercise 4.1 #include <stdio.h>
```
#include <stdlib.h>
int main(void)
{
int A, B, Soma;
printf("Digite um numero inteiro: ");
scanf("%d", &A);
printf("Digite um numero inteiro: ");
scanf("%d", &B);
Soma = A + B;
printf ("O Valor da soma = %d", Soma);
   if(Soma > 10)
   {
      printf("O valor da soma eh maior que 10 \n");
   }
   system ("PAUSE");
   return(0);
}
```
■

Observe que somente será escrito na tela "O valor da soma é maior que 10" SE a condição (Soma > 10) for verdadeira. Caso a condição seja falsa os comandos que estão dentro do if serão ignorados.

4.1.1.2 Estrutura de decisão if else

Agora vamos modificar o código do exemplo anterior com o intuito de enviar uma mensagem informando também quando a soma é menor ou igual a 10.

Para tanto, o pseudocódigo seria:

Se (Soma > 10) **então**
   Escreva("Valor maior que 10");
Senão
   Escreva("Valor menor ou igual a 10");

Sendo assim, vamos executar uma determinada instrução se a condição for satisfeita ou outra instrução quando a condição for falsa.

O senão simboliza a negação da condição, logo está ligado à instrução que será executada quando a condição for falsa.

Em linguagem C temos o código equivalente:

```
if(Soma > 10)
   printf("O valor da soma é maior que 10 \n");
else
   printf("Valor menor ou igual a 10");
```

Observe que o senão em linguagem C é representado pelo comando else.

Veja o código completo:

Exercise 4.2 #include <stdio.h>
```
#include <stdlib.h>
int main(void)
{
   int A, B, Soma;
   printf("Digite um numero inteiro: ");
```

```
    scanf("%d", &A);
    printf("Digite um numero inteiro: ");
    scanf("%d", &B);
    Soma = A + B;
    printf("O Valor da soma = %d\n", Soma);
    if(Soma > 10)
    {
       printf("O valor da soma e maior que 10\n");
    }
    else
    {
       printf("Valor menor ou igual a 10\n");
    }
    system("PAUSE");
    return(0);
}
```

4.1.1.3 Estruturas encadeadas - if-else

Chamamos de estruturas de decisão encadeadas, quando uma estrutura de decisão está localizada dentro do lado falso da outra. Este tipo de estrutura também é conhecida como seleção “aninhada” ou seleção “encaixada”.

Qualquer que seja o termo usado para identificar a estrutura, o importante é que esse formato com uma estrutura de seleção dentro da outra permite fazer a escolha de apenas um entre vários comandos possíveis.

Vejamos um exemplo em pseudocódigo.

Exemplo:

Receber os valores inteiros e verificar qual dos valores é o maior. Emitir uma mensagem caso os valores sejam iguais.

```
Algoritmo MaiorNumero;
Var
   N1,N2: Inteiro;
Inicio
   Escreva("Digite o primeiro número: ");
   Leia(N1);
   Escreva("Digite o segundo número: ");
   Leia(N2);
   Se (N1 == N2) então
      Escreva ("Os números são iguais”);
   Senão
      Se (N1 > N2) então
         Escreva("O maior valor é = ",N1);
      Senão
         Escreva("O maior valor é = ",N2);
Fim.
```

Em linguagem C podemos usar estruturas if – else -if encadeadas para construir o código equivalente.

Exercise 4.3 #include <stdio.h>
#include <stdlib.h>

```
int main (void)
{
   int N1, N2 ;
   printf("Digite o primeiro numero: ");
   scanf("%d", &N1);
   printf("Digite o segundo numero: ");
   scanf("%d", &N2);
   if (N1 == N2)
      printf("Os numeros sao iguais!");
   else
      if (N1 > N2)
         printf("O maior valor e = %d", N1);
      else
         printf("O maior valor e = %d", N2);
   printf("\n");
   system("pause");
   return (0);
}
```

Você pode usar vários if...else um dentro do outro, tudo depende de quantas opções você tem disponíveis. Note bem que neste caso, ao usar if...else encadeado (aninhado) uma condição exclui a outra.

4.1.2 Switch

É uma forma de reduzir a complexidade de vários if ... else encadeados.

É muito utilizado, principalmente para uso em estruturas de menu.

O conteúdo de uma variável é comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado.

Em português estruturado o comando escolha...caso equivale ao switch...case.

```
Escolha (Variável)
Inicio
   Caso (Valor1):
      Instruções;
   Caso (Valor2):
      Instruções;
   Caso (ValorN):
      Instruções;
Fim;
```

Sintaxe do comando switch case em linguagem C

```
switch (variável)
   case constante1:
      Instruções;
   break;
   case constante2:
      Instruções;
   break;
   default
      Instruções;
```

Vamos construir um programa para verificar o dia da semana a fim de exemplificar a utilização do switch...case.

O usuário vai digitar um número e o programa vai retornar o dia da semana equivalente ao número.

Exercise 4.4 DIA DA SEMANA EQUIVALENTE AO NÚMERO

```
#include <stdio.h>
#include <conio.h>
int main (void )
{
   int valor;
   printf ("Digite um valor de 1 a 7: ");
   scanf("%d", &valor);
   switch ( valor )
   {
      case 1 :
      printf ("Domingo\n");
      break;
      case 2 :
      printf ("Segunda\n");
      break;
      case 3 :
      printf ("Terça\n");
      break;
      case 4 :
      printf ("Quarta\n");
      break;
      case 5 :
      printf ("Quinta\n");
      break;
      case 6 :
      printf ("Sexta\n");
      break;
      case 7 :
      printf ("Sabado\n");
      break;
      default :
      printf ("Valor invalido!\n");
   }
   getch();
   return 0;
}
```

4.1.3 Ternários

O operador ternário é uma alternativa para substituir o if...else em algumas situações por ser um comando bem enxuto.

Sintaxe:

Condição ? verdadeiro : falso

Onde:

Condição é a condição que será testada.

Verdadeiro é o que fazer quando a condição for verdadeira.

Falso é o que fazer quando a condição for falsa.

Exemplo de programa em C usando operador ternário:

Exercise 4.5 TERNÁRIO

```
#include <stdio.h>
#include <conio.h>
int main (void )
{
   int numero;
   printf("Digite um numero: ");
   scanf("%d",&numero);
   numero >= 0 ? numero++ : numero–;
   printf("O novo valor de numero e: %d",numero);
   getch();
   return(0);
}
```

Explicação do código

numero >= 0 ? numero++ : numero–;

Neste código se o número for maior ou igual a zero será incrementado, caso contrário será decrementado de uma unidade.

Seria o equivalente a usar if:

if (numero > = 0) Numero ++; else Numero –;

4.2 Estruturas de repetição

Em linguagem C, existem três estruturas de repetição, são elas: for, while e do... while.

Cada uma destas estruturas tem a sua particularidade em termos de funcionamento.

Veremos cada uma delas em subseções separadas.

4.2.1 Comando For

Em pseudocódigo o laço for da linguagem C é equivalente ao comando Para.

Seu funcionamento é simples, como veremos.

Pseudocódigo:

Para (valor_inicial até condição_final passo n) **faça**

Inicio

Instruções;

Fim

onde:

valor_inicial é uma instrução de atribuição do valor inicial do laço para a variável de controle.

condição final é uma condição que controla o laço.

passo é o incremento do laço.

O laço for é uma estrutura muito útil quando se sabe de antemão quantas vezes a repetição deverá ser executada. Este laço utiliza uma variável para controlar a contagem do loop, bem como seu incremento.

Trata-se de um comando bem enxuto, já que própria estrutura faz a inicialização, incremento e encerramento do laço.

Sintaxe:

for(valor_inicial; condição_final; valor_incremento)

instruções;

Exercise 4.6 Laço com contagem decrescente

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
   int contador; //variável de controle do loop
   for(contador = 1; contador <= 10; contador++)
   {
      printf("%d ", contador);
   }
   getch();
   return(0);
}
```

Neste exemplo podemos observar três coisas sobre o funcionamento da estrutura de repetição for:

Primeiramente o contador foi inicializado com um valor determinado

Depois testamos se a condição que envolve o contador é verdadeira (ou seja se contador <= 10), em caso afirmativo, a repetição continua; caso – contrário, ela será encerrada.

A cada nova repetição a variável do contador foi incrementada

4.2.1.1 Laço for decrescente

Em certos casos, ao invés de incrementar, podemos decrementar a variável de controle e construir um laço com contagem decrescente. Para tanto, devemos inicializar a variável de controle com um valor adequado e construir a condição necessária para que o laço seja finalizado.

Exercise 4.7 Laço com contagem decrescente

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
   int contador; //variável de controle do loop
   for(contador = 1; contador >= 1; contador–)
   {
      printf("%d ", contador);
   }
   getch();
   return(0);
}
```

4.2.1.2 Laço for repetir frase

Agora vejamos mais um exemplo de utilização de laço for que repete um comando determinado número de vezes. No código abaixo vamos repetir uma frase 3 vezes.

Exercise 4.8 Repetir uma frase 3 vezes

```
#include <stdio.h>
#include <conio.h>
```

```
int main(void)
{
   int i; //variável de controle do loop
   for(i = 1; i <= 3; i++)
   {
      printf("Robótica Elementar\n");
   }
   getch();
   return(0);
}
```

4.2.2 Comando While

Executa a repetição de um bloco de instruções enquanto uma condição é verdadeira.

Pseudocódigo

A estrutura **Enquanto ... Faça** equivale a estrutura while em linguagem C.

A Sintaxe seria:

Iniciar a variável de controle
Enquanto (condição) faça
Início
Instruções;
Atualizar a variável de controle;
Fim;

Em C a sintaxe seria:

```
while (condição)
{
   Instrução ou bloco de instruções;
}
```

Exercise 4.9 REPETIÇÃO DE 1 A 10

```
#include <stdio.h>
int main(void)
{
   int contador = 1; //declarando e inicializando a variável de controle
   while (contador <= 10) // Testando a condição
   {
      printf("
      contador++; //atualizando a variável de controle
   }
   return 0;
}
```

Inicialmente, a variável de controle denominada contador foi declarada e inicializada com o valor 1.

O teste da condição while é realizado , e como o contador é menor que 10, então a condição é verdadeira.

Como a condição é verdadeira, o programa entra dentro do corpo do laço e executa o printf, exibindo o valor da variável contador.

Após executar o comando printf, o contador é incrementado.

Depois do incremento, a condição é testada novamente e enquanto for verdadeira, os comandos são executados de novo, até que a condição se torne falsa.

A condição falsa faz com que o laço seja encerrado.

4.2.3 Comando do-While

Esta estrutura de repetição, garante que o bloco de instruções seja executado no mínimo uma vez, já que a condição que controla o laço é testada apenas no final do comando.

A diferença entre o comando while e o do... while é justamente o local onde a condição que controla o laço é testada.

No comando while a condição é testada antes do bloco de instruções, e caso a condição seja falsa a repetição não será executada.

No do... while o bloco de comandos é executado pelo menos uma vez de forma obrigatória, independente do resultado da expressão lógica.

Vejamos um exemplo de código usando o comando do ... while

Exercise 4.10

```
#include<stdio.h>
int main(void)
{
   float nota1=0,nota2=0,media=0;
   int resp;
   do
   {
      printf("Digite a primeira nota: ");
      scanf("%f",&nota1);
      printf("Digite a segunda nota: ");
      scanf("%f",&nota2);
      media = (nota1 + nota2)/2;
      printf("Media do aluno = %f\n",media);
      printf("Digite 1 para continuar ou 2 para sair\n");
      scanf("%d", &resp);
   }while (resp==1);
   return 0;
}
```

■