

Competências, Habilidades e Bases Tecnológicas da disciplina de Programação e Algoritmos.

I.2 PROGRAMAÇÃO E ALGORITMOS					
Função: Elaboração de programas utilizando linguagens de programação					
Classificação: Execução					
Atribuições e Responsabilidades					
<ul style="list-style-type: none">Implementar algoritmos em linguagem de programação, utilizando ambientes de desenvolvimento de acordo com as necessidades.					
Valores e Atitudes					
<ul style="list-style-type: none">Estimular a organização.Incentivar atitudes de autonomia.Fortalecer a persistência e o interesse na resolução de situações-problema.					
Competências			Habilidades		
1. Implementar algoritmos de programação.			1.1 Elaborar algoritmos.		
2. Utilizar linguagem de programação em ambiente de desenvolvimento.			2.1 Codificar programas, utilizando técnica de programação estruturada.		
Orientações					
<ul style="list-style-type: none">Detalhamento das Bases Tecnológicas - Anexo 1					
Bases Tecnológicas					
Comandos da linguagem de programação					
Programação estruturada					
Programação modular					
Tipos de dados estruturados					
Carga horária (horas-aula)					
Teórica	00	Prática em Laboratório*	120	Total	120 Horas-aula
Teórica (2,5)	00	Prática em Laboratório* (2,5)	100	Total (2,5)	100 Horas-aula
<p>* Possibilidade de divisão de classes em turmas, conforme o item 4.8 do Plano de Curso.</p> <p>* Todos os componentes curriculares preveem prática, expressa nas habilidades, relacionadas às competências. Para este componente curricular está prevista divisão de classes em turmas.</p>					
Para ter acesso às titulações dos Profissionais habilitados a ministrarem aulas neste componente curricular, consultar o site: http://www.cpscetec.com.br/crt/					

1-) Comunicação de alunos com alunos e professores:

- Um e-mail para a sala é de grande valia para divulgação de material, notícias e etc.
- Criação de grupo nas redes sociais também é interessante.

2-) Uso de celulares:

Para o bom andamento das aulas, recomendo que utilizem os celulares em vibracall, para não atrapalhar o andamento da aula.

3-) Material das aulas:

A disciplina trabalha com NOTAS DE AULA que são disponibilizadas ao final de cada aula, e calendários de PTCC estarão disponíveis no site: www.marcoscosta.eti.br, na página principal.

4-) E-mail da disciplina:

A disciplina terá um e-mail de envio das atividades, dúvidas e assim por diante, assim sendo todas as atividades deverão ser enviadas para tal e-mail, se forem enviadas atividades de tal disciplina para e-mail não mencionado, a atividade será desconsiderada.

O e-mail desta disciplina é: **1_dsn_pa@outlook.com**

5-) Prazos de trabalhos e atividades:

Toda atividade solicitada terá uma data limite de entrega, de forma alguma tal data será postergada, ou seja, se não for entregue até a data limite a mesma receberá menção I, isso tanto para atividades entregues de forma impressa ou enviadas ao e-mail da disciplina. No caso de PTCC o calendário será seguido, e no dia marcado de determinada atividade só receberão menção os alunos **presentes**, caso não estejam a menção para aquela atividade será I.

6-) Qualidade do material de atividades:

- Impressas ou manuscritas:

Muita atenção na qualidade do que será entregue, atividades sem grampear, faltando nome e número de componentes, rasgadas, amassadas, com rebarba de folha de caderno e etc. serão desconsiderados por mim.

- Digitais:

Ao enviarem atividades para o e-mail da disciplina, **SEMPRE** no assunto deverá ter o nome da atividade que está sendo enviada, e no corpo do e-mail deverá ter o(s) nome(s) do(s) integrante(s) da atividade, sem estar desta forma a atividade será DESCONSIDERADA.

7-) Menções e critérios de avaliação:

Na ETEC os senhores serão avaliados por MENÇÃO, onde temos:

- MB - Muito Bom;
- B - Bom;
- R - Regular;
- I - Insatisfatório.

Cada trimestres teremos as seguintes formas de avaliação:

- 1 avaliação teórica;
- 1 avaliação prática (a partir do 2º trimestre, e as turmas do 2º módulo em diante);
- Seminários;
- Trabalhos teóricos e/ou práticos;
- Assiduidade;
- Outras que se fizerem necessário.

1. INTRODUÇÃO A LÓGICA DE PROGRAMAÇÃO

Em nossa disciplina iremos trabalhar a ideia do Arduino como ferramenta de aplicação da Lógica de Programação que será estudada nesse primeiro módulo, para isso teremos uma base de microprocessadores, microcontroladores, lógica de programação, uma base de eletrônica e por fim programação em Arduino.

A lógica de programação é uma disciplina que está presente em grande parte da grade de cursos técnicos e de graduação voltados para a área de informática, bem como nas engenharias elétrica, mecânica e mecatrônica, através disso vê-se a necessidade de fazer com que ela tenha o seu conceito teórico aplicado de forma prática em níveis que vão além de softwares criados para computador. A robótica pode ter grande parceria no ensino da lógica de programação, pois através dela o aluno será introduzido no mundo da automação, criando, inovando e contribuindo para o desenvolvimento da ciência desde o primeiro momento de um curso técnico ou de graduação, visto que a matéria de lógica de programação normalmente é ministrada nos primeiros períodos do curso. Para alunos que fazem cursos voltados para engenharia, quando cursarem essa disciplina praticando através de conceitos da robótica em plataformas livres como o Arduino será mais fácil assimilar conteúdos de outras matérias como eletrônica, física, mecânica e mecatrônica.

1.1 COMPUTADOR

Um computador é, de forma simplificada, uma máquina que processa instruções. Essas instruções são processadas no "cérebro" do computador, que se chama microprocessador. Todo computador possui pelo menos um microprocessador. O Arduino, por exemplo, nada mais é do que um computador muito pequeno, e ele utiliza um microprocessador do modelo ATmega. Alguns microprocessadores, como o ATmega, também são chamados de microcontroladores.

1.2 PROGRAMA DE COMPUTADOR

Um programa de computador, ou software, é uma sequência de instruções que são enviadas para o computador. Cada tipo de microprocessador (cérebro) entende um conjunto de instruções diferente, ou seja, o seu próprio "idioma". Também chamamos esse idioma de linguagem de máquina. As linguagens de máquina são, no fundo, as únicas linguagens que os computadores conseguem entender, só que elas são muito difíceis para os seres humanos entenderem. É por isso nós usamos uma coisa chamada linguagem de programação. No caso de sistemas como o Arduino (os chamados sistemas embarcados), o software que roda no microprocessador é também chamado de firmware.

1.3 LINGUAGEM DE PROGRAMAÇÃO

Nós seres humanos precisamos converter as nossas ideias para uma forma que os computadores consigam processar, ou seja, a linguagem de máquina. Os computadores de hoje (ainda) não conseguem entender a linguagem natural que nós usamos no dia a dia, então precisamos de um outro "idioma" especial para instruir o computador a fazer as tarefas que desejamos. Esse "idioma" é uma linguagem de programação, e na verdade existem muitas delas. Essas linguagens de programação também são chamadas de linguagens de programação de alto nível. A linguagem de programação utilizada no Arduino é a linguagem C++ (com pequenas modificações), que é uma linguagem muito tradicional e conhecida.

2. MICROPROCESSADORES

2.1. HISTÓRICO

Embora as primeiras gerações de computadores tivessem obtido grande sucesso nas décadas de 50 e 60, apresentavam alguns inconvenientes: o tamanho e a velocidade. Um impacto tecnológico viria a reduzir as dimensões dos computadores ao mesmo tempo em que os tornariam mais rápidos: o surgimento dos *microprocessadores*.

Um *microprocessador* é um circuito integrado ("chip") capaz de executar instruções, tendo com sua principal parte a Unidade Central de Processamento (CPU). Com o avanço tecnológico na área da microeletrônica, outras características vêm sendo incorporadas ao longo das últimas décadas aos microprocessadores, como unidades de gerenciamento de memória, memória cache, coprocessador numérico, etc, tornando-os cada vez mais complexos.

A origem dos microprocessadores data de 1971, quando a Intel Corporation lançou no mercado o microprocessador 4004, denominado originalmente como "calculadora em um único chip", podendo ser considerado como o primeiro processador de propósito geral. Possuía em torno de 3.000 transistores e logo surgiram aplicações para ele. A partir desta nova tecnologia surgiriam as calculadoras mais modernas, os computadores pessoais (PC), as "workstations", e atualmente os microprocessadores vêm derrubando a última fronteira na área dos computadores: os "mainframes".

2.2. O QUE É UM MICROPROCESSADOR

O microprocessador é um dispositivo lógico programável em um único chip de silício, concebido sob a tecnologia VLSI (circuito integrado em alta escala). Ele age sob o controle de um programa armazenado em memória, executando operações aritméticas, lógica booleana, tomada de decisão, além de entrada e saída, permitindo a comunicação com outros dispositivos periféricos.

2.3. O MICROCOMPUTADOR PESSOAL (PC) E AS "WORKSTATIONS"

O microcomputador pessoal surgiu a partir da evolução dos microprocessadores, correspondendo a um sistema computacional completo, constituído pelo microprocessador, memória primária de armazenamento de dados e programas, memória secundária de armazenamento (disco magnético rígido e flexível, ótico, fita magnética), dispositivos convencionais de entrada (teclado, mouse) e saída (monitor, impressora, caixas de som).

Uma workstation apresenta como principais características, além das citadas para um microcomputador, a alta capacidade de processamento de operações matemáticas em ponto flutuante, a alta resolução gráfica e a maior capacidade de armazenamento.

Para converter um programa escrito em uma linguagem de alto nível para linguagem de máquina, nós utilizamos uma coisa chamada compilador. A ação de converter um programa para linguagem de máquina é chamada compilar. Para compilar um programa, normalmente se utiliza um ambiente de desenvolvimento (ou IDE, do inglês Integrated Development Environment), que é um aplicativo de computador que possui um compilador integrado, onde você pode escrever o seu programa e compilá-lo. No caso do Arduino, esse ambiente de desenvolvimento é o Arduino IDE. O texto contendo o programa em uma linguagem de programação de alto nível também é conhecido como o código fonte do programa.

2.4 ALGORITMO (PROGRAMA)

Um algoritmo, ou simplesmente programa, é uma forma de dizer para um computador o que ele deve fazer, de uma forma que nós humanos conseguimos entender facilmente. Os algoritmos normalmente são escritos em linguagens de programação de alto nível. Isso se aplica a praticamente qualquer computador, inclusive o Arduino, onde um algoritmo também é conhecido como sketch. Para simplificar, a partir de agora nós vamos nos referir aos algoritmos, programas ou sketches simplesmente como "programas". Um programa é composto de uma sequência de comandos, normalmente escritos em um arquivo de texto.

2.5. APLICAÇÕES DE MICROPROCESSADORES

No nosso dia-a-dia nos deparamos com inúmeras aplicações de microprocessadores, sendo que na maioria das vezes de forma despercebida.

Pode-se citar, apenas a título de exemplo: o relógio digital/despertador, calculadoras, alarmes antifurto de residências e automóveis, o controle de injeção de combustível em automóveis, os eletrodomésticos como micro-ondas e máquinas de lavar-louças, videocassetes, etc. Também não podemos deixar de mencionar os microcomputadores, hoje presentes não só no ambiente de trabalho (escritórios e linhas de produção), mas também em muitas residências.

3. MICROCONTROLADORES

3.1 INTRODUÇÃO

Atualmente um grande número de microcontroladores, integrados em diversos equipamentos, exercem um papel importante no dia a dia das pessoas. Despertar ao som de um CD Player programável, tomar café da manhã preparado por um micro-ondas digital, e ir ao trabalho de carro, cuja injeção de combustível é microcontrolada, são apenas alguns exemplos.

O mercado de microcontroladores apresenta-se em franca expansão, ampliando seu alcance principalmente em aplicações residenciais, industriais, automotivas e de telecomunicações. Segundo dados da National Semiconductor (1997), uma residência típica americana possui 35 produtos baseados em microcontrolador. Estima-se que, em 2010, em média uma pessoa interagiria com 250 dispositivos com microcontroladores diariamente.

Em um passado recente, o alto custo dos dispositivos eletrônicos limitou o uso dos microcontroladores apenas aos produtos domésticos considerados de alta tecnologia (televisão, vídeo e som). Porém, com a constante queda nos preços dos circuitos integrados, os microcontroladores passaram a ser utilizados em produtos menos sofisticados do ponto de vista da tecnologia, como máquinas de lavar, micro-ondas, fogões e refrigeradores. Assim, a introdução do microcontrolador nestes produtos cria uma diferenciação e permite a inclusão de melhorias de segurança e de funcionalidade. Alguns mercados chegaram ao ponto de tornar obrigatório o uso de microcontroladores em determinados tipos de equipamentos, impondo um pré-requisito tecnológico. Muitos produtos que temos disponíveis hoje em dia, simplesmente não existiriam, ou não teriam as mesmas funcionalidades sem um microcontrolador. É o caso, por exemplo, de vários instrumentos biomédicos, instrumentos de navegação por satélites, detectores de radar, equipamentos de áudio e vídeo, eletrodomésticos, entre outros.

Entretanto, o alcance dos microcontroladores vai além de oferecer algumas facilidades. Uma aplicação crucial, onde os microcontroladores são utilizados, é na redução de consumo de recursos naturais. Existem sistemas de aquecimento modernos que captam a luz solar e, de acordo com a demanda dos usuários, controlam a temperatura de forma a minimizar perdas. Um outro exemplo, de maior impacto, é o uso de microcontroladores na redução do consumo de energia em motores elétricos, que são responsáveis pelo consumo de, aproximadamente, 50%

de toda eletricidade produzida no planeta. Portanto, o alcance dessa tecnologia tem influencia muito mais importante em nossas vidas, do que se possa imaginar.

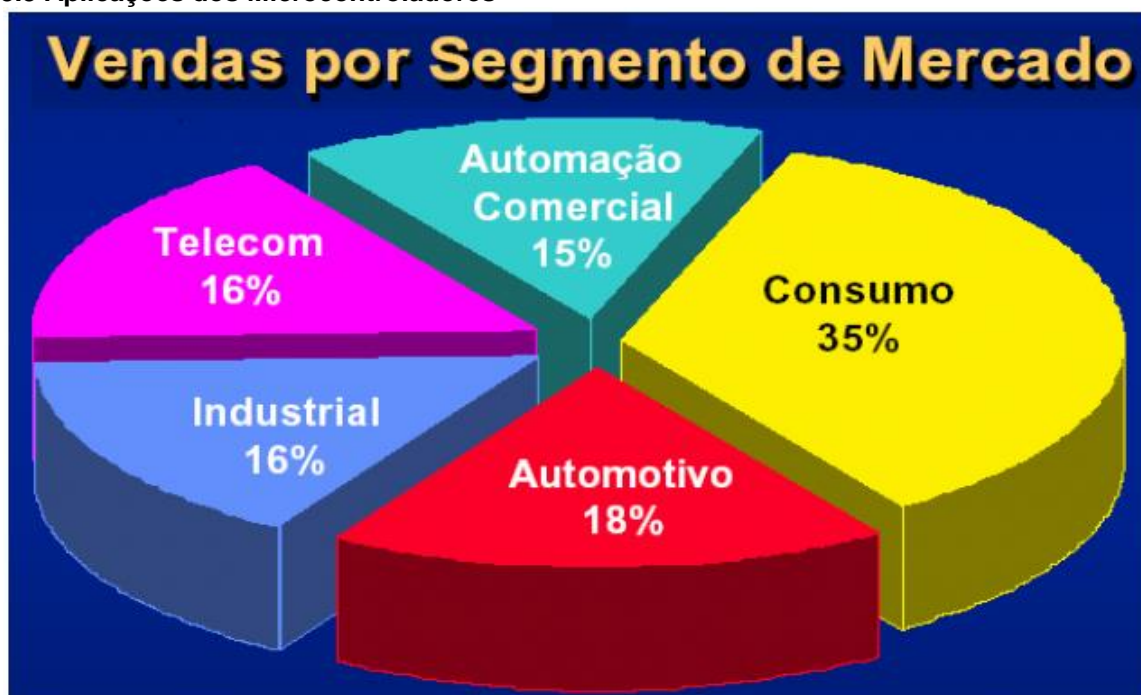
O universo de aplicações dos microcontroladores, como já mencionado, esta em grande expansão, sendo que a maior parcela dessas aplicações é em sistemas embarcados. A expressão “sistema embarcado” (do inglês embedded system) se refere ao fato do microcontrolador ser inserido nas aplicações (produtos) e usado de forma exclusiva por elas. Como a complexidade desses sistemas cresce vertiginosamente, o software tem sido fundamental para oferecer as respostas as necessidades desse mercado. Tanto e, que o software para microcontroladores representa uma fatia considerável do mercado de software mundial. Segundo Edward Yourdon (consultor na área de computação, pioneiro nas metodologias de engenharia do software e programação estruturada) a proliferação dos sistemas embarcados, juntamente com o advento da Microsoft, são os responsáveis pela retomada do crescimento da indústria de software nos Estados Unidos da America.

3.2. DEFINIÇÃO DE MICROCONTROLADOR

Um microcontrolador e um sistema computacional completo, no qual estão incluídos uma CPU (Central Processor Unit), memória de dados e programa, um sistema de clock, portas de I/O (Input/Output), além de outros possíveis periféricos, tais como, módulos de temporização e conversores A/D entre outros, integrados em um mesmo componente. As partes integrantes de qualquer computador, e que também estão presentes, em menor escala, nos microcontroladores são:

- Unidade Central de Processamento (CPU)
- Sistema de clock para dar sequencia as atividades da CPU
- Memória para armazenamento de instruções e para manipulação de dados
- Entradas para interiorizar na CPU informações do mundo externo
- Saídas para exteriorizar informações processadas pela CPU para o mundo externo
- Programa (firmware) para definir um objetivo ao sistema.

3.3 Aplicações dos Microcontroladores



4. OS MICROCONTROLADORES NA ATUALIDADE:

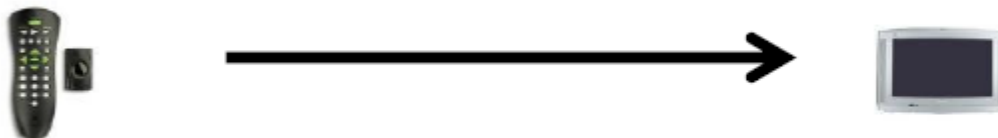
As principais áreas de atuação são:

- Área Automobilística
- Automação
- Segurança
- Controle de Tráfego
- Médica

- Entretenimento
- Robótica

5. EXEMPLO PRÁTICO DE UTILIZAÇÃO DE UM MICROCONTROLADOR

- Podemos citar de início o controle remoto de uma TV



- Outro exemplo corriqueiro é o despertador



- Outro exemplo é a parte de segurança



- Casa Inteligente



- Taxímetro



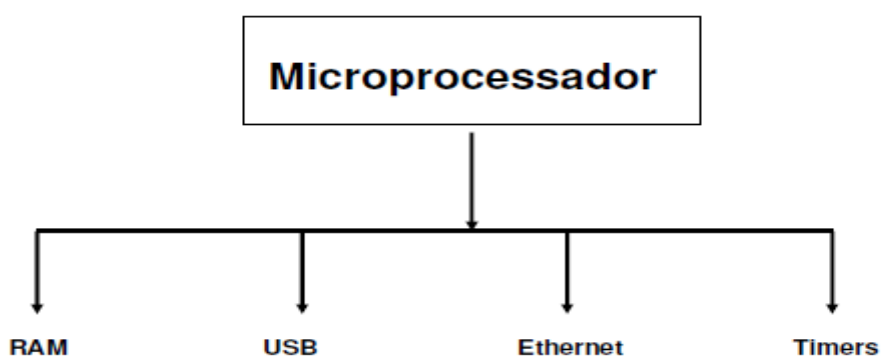
- Indústrias



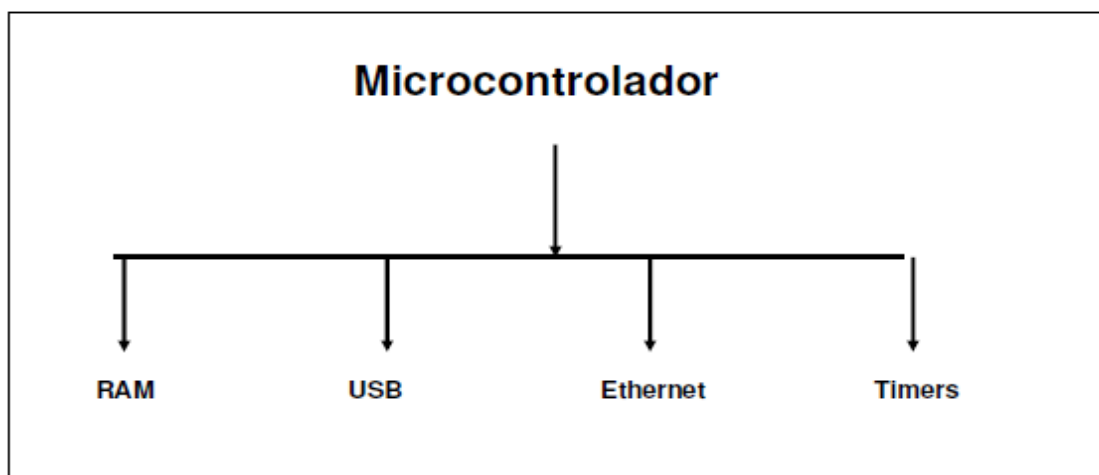
6. DIFERENÇAS ENTRE MICROCONTROLADOR E MICROPROCESSADOR

- Tanto os Microcontroladores como os Microprocessadores possuem uma ULA (unidade lógica e aritmética)
- A ULA de um processador convencional de fato é muito mais poderosa se comparada a uma ULA de um microcontrolador.
- A ULA do microcontrolador é menos poderosa, porém em uma única pastilha já temos todos os recursos para o funcionamento do mesmo.

Arquitetura de um Microprocessador



Arquitetura de um Microcontrolador



7. ÁREA DE ATUAÇÃO DO MICROCONTROLADOR E DO MICROPROCESSADOR

Os microprocessadores são utilizados em aplicações onde são requeridos cálculos matemáticos complexos e com muita velocidade



Já os microcontroladores são utilizados de forma dedicada, por exemplo em eletrodomésticos, onde a velocidade de processamento não é tão alta.



8. LÓGICA

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequência lógica para o desenvolvimento. Então o que é lógica?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.

8.1 SEQUENCIA LÓGICA

Estes pensamentos, podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Sequencia Lógica são passos executados até atingir um objetivo ou solução de um problema.

8.2 INSTRUÇÕES

Na linguagem comum, entende-se por instruções **“um conjunto de regras ou normas definidas para a realização ou emprego de algo”**. Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar.

Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica. Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc.

É evidente que essas instruções tem que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las. Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

Instruções são um conjunto de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar.

8.3 ALGORITMO

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento. Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas.

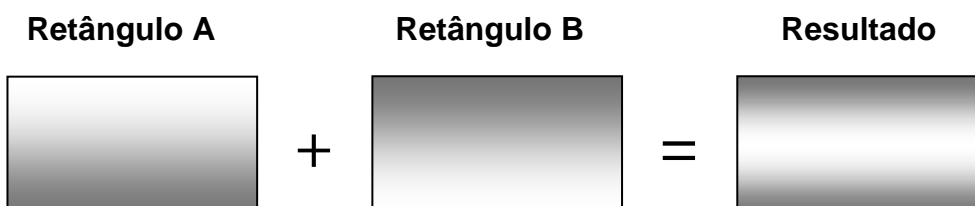
Por exemplo:

“Chupar uma bala”.

- 1) Pegar a bala;
- 2) Retirar o papel;
- 3) Chupar a bala;
- 4) Jogar o papel no lixo;

“Somar dois números quaisquer”.

- 1) Escreva o primeiro número no retângulo A
- 2) Escreva o segundo número no retângulo B
- 3) Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C



8.4 PROGRAMAS

Os programas de computadores nada mais são do que algoritmos escritos numa linguagem de computador (Pascal, C, Cobol, Java, Visual Basic entre outras) e que são interpretados e executados por uma máquina, no caso um computador. Notem que dada esta interpretação rigorosa, um programa é por natureza muito específico e rígido em relação aos algoritmos da vida real.

8.5 FAÇA AS ATIVIDADES EM DUPLA ABAIXO DE FORMA MANUSCRITA PARA A PRÓXIMA AULA.

- 1) Crie uma sequência lógica para tomar banho;
- 2) Faça um algoritmo para somar dois números e multiplicar o resultado pelo primeiro número;
- 3) Descreva a sequência lógica para trocar um pneu de um carro;
- 4) Faça um algoritmo para trocar uma lâmpada.

9. DESENVOLVENDO ALGORITMOS

9.1 PSEUDOCÓDIGO

Os algoritmos são descritos em uma linguagem chamada **pseudocódigo**. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo Java, estaremos gerando código em Java. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

9.2 REGRAS PARA CONSTRUÇÃO DO ALGORITMO

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- ✓ Usar somente um verbo por frase;
- ✓ Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática;
- ✓ Usar frases curtas e simples;
- ✓ Ser objetivo;
- ✓ Procurar usar palavras que não tenham sentido dúbio (duplo).

9.3 FASES

No capítulo anterior vimos que ALGORITMO é uma sequência lógica de instruções que podem ser executadas. É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

Como fazer arroz doce ou então calcular o saldo financeiro de um estoque, entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais.



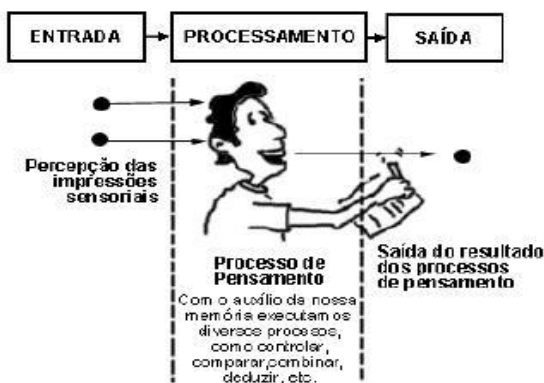
Onde temos:

Entrada: São os dados de entrada do algoritmo.

Processamento: São os procedimentos utilizados para chegar ao resultado final.

Saída: São os dados já processados.

Analogia com o homem:



9.4 EXEMPLO DE ALGORITMO

Imagine o seguinte problema: Calcular a média final dos alunos da 8ª Série. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde:

$$\text{Média Final} = \frac{P1 + P2 + P3 + P4}{4}$$

Para montar o algoritmo proposto, faremos três perguntas:

a) *Quais são os dados de entrada?*

R: Os dados de entrada são P1, P2, P3 e P4

b) *Qual será o processamento a ser utilizado?*

R: O procedimento será somar todos os dados de entrada e dividi-los por 4

c) *Quais serão os dados de saída?*

R: O dado de saída será a média final

Algoritmo

Receba a nota da prova1

Receba a nota de prova2

Receba a nota de prova3

Receba a nota da prova4

Some todas as notas e divida o resultado por 4

Mostre o resultado da divisão

9.5 TESTE DE MESA

Após desenvolver um algoritmo ele deverá sempre ser testado. Este teste é chamado de **TESTE DE MESA**, que significa, seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

Veja o exemplo:

Utilize a tabela abaixo:

Nota da Prova 1

Nota da Prova 2

Nota da Prova 3

Nota da Prova 4

P1	P2	P3	P4	Processamento	Média

9.6 EXERCÍCIOS

Os exercícios abaixo poderão ser realizados em duplas e deverá ser entregue até a próxima aula (24/08/2017) de forma escrita em papel.

1) Identifique os dados de entrada, processamento e saída no algoritmo abaixo:

- Receba código da peça
- Receba valor da peça
- Receba Quantidade de peças
- Calcule o valor total da peça (Quantidade * Valor da peça)
- Mostre o código da peça e seu valor total

2) Faça um algoritmo para “Calcular o estoque médio de uma peça”, sendo que
 $\text{ESTOQUE_MÉDIO} = (\text{QUANTIDADE_MÍNIMA} + \text{QUANTIDADE_MÁXIMA}) / 2$

- 3) A imobiliária Imóbilis vende apenas terrenos retangulares. Faça um algoritmo para ler as dimensões de um terreno e depois exibir a área do mesmo, realize o teste de mesa.
- 4) Faça um algoritmo para ler o salário de um funcionário e aumentar em 15%. Após o aumento, desconte 8% de impostos. Imprima o salário inicial, o salário com o aumento e o salário final, realize o teste de mesa.
- 5) Escreva um algoritmo para ler o nome e a idade de uma pessoa, e exibir quantos dias de vida ela possui. Considere sempre anos completos, e que um ano possui 365 dias. Ex: uma pessoa com 19 anos possui 6935 dias de vida, realize o teste de mesa.


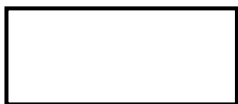

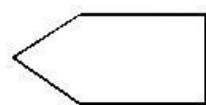
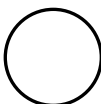
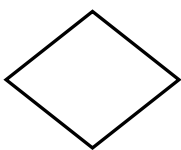
10. DIAGRAMA DE BLOCO

10.1 O QUE É UM DIAGRAMA DE BLOCO?

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento. Com o diagrama podemos definir uma sequência de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

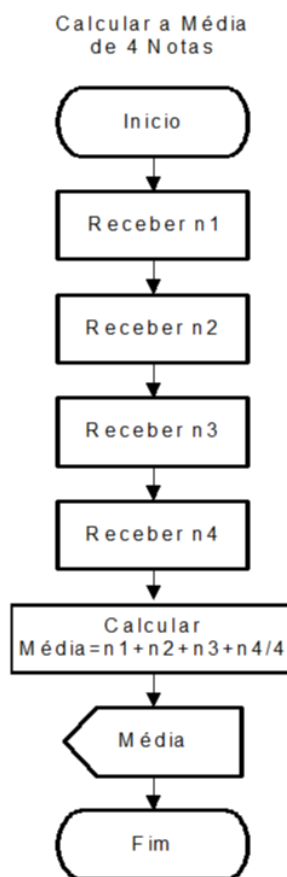
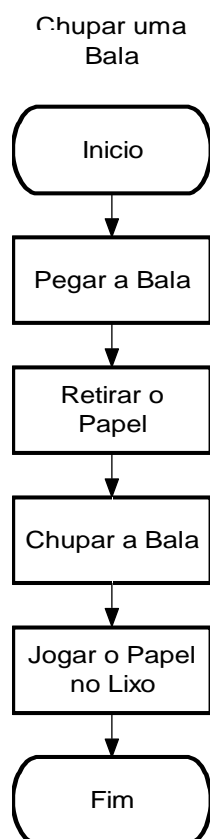
10.2 SIMBOLOGIA

Existem diversos símbolos em um diagrama de bloco. No decorrer do curso apresentaremos os mais utilizados. Veja no quadro a seguir alguns dos símbolos que iremos utilizar:

Símbolo	Descrição	Função
	Terminal	Símbolo que indica o início ou o fim de um processamento. <i>Ex: início do algoritmo</i>
	Processamento	Símbolo de processamento em geral. <i>Ex: cálculo de dois números</i>
	Entrada de dado manual	Símbolo que indica a entrada de dados através do teclado. <i>Ex: digitar a nota da prova 1</i>
	Exibir	Símbolo que mostra informações ou exibe resultados. <i>Ex: mostre o resultado do cálculo</i>
	Conectar	Símbolo utilizado para conectar duas partes do digrama de bloco.
	Comparar	Símbolo utilizado para comparação entre expressões. Ex: código <= 1000 (sim/não)

Dentro do símbolo sempre terá algo escrito, pois somente os símbolos não nos dizem nada. Veja no exemplo a seguir:

Exemplos de Diagrama de Bloco:



Veja que no exemplo da bala seguimos uma sequência lógica somente com informações diretas, já no segundo exemplo da média utilizamos cálculo e exibimos o resultado do mesmo.

10.3 EXERCÍCIOS

1) Construa um diagrama de blocos que :

- Leia a cotação do dólar
- Leia um valor em dólares
- Converta esse valor para Real
- Mostre o resultado

2) Desenvolva um diagrama que:

- Leia 4 (quatro) números
- Calcule o quadrado para cada um
- Somem todos e
- Mostre o resultado

3) Construa um algoritmo para pagamento de comissão de vendedores de peças, levando-se em consideração que sua comissão será de 5% do total da venda e que você tem os seguintes dados:

- Identificação do vendedor
- Código da peça
- Preço unitário da peça
- Quantidade vendida

E depois construa o diagrama de blocos do algoritmo desenvolvido, e por fim faça um teste de mesa.

- 4) Faça um programa que receba três números, calcule e mostre a multiplicação desses números.
- 5) Faça um programa que receba duas notas, calcule e mostre a média ponderada dessas notas, considerando peso 2 para a primeira nota e peso 3 para a segunda nota.

11. CONSTANTES, VARIÁVEIS E TIPOS DE DADOS

Variáveis e constantes são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado.

Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Um programa deve conter declarações que especificam de que tipo são as variáveis que ele utilizará e as vezes um valor inicial. Tipos podem ser por exemplo: inteiros, reais, lógicos, caracteres, etc. As expressões combinam variáveis e constantes para calcular novos valores.

11.1 CONSTANTES

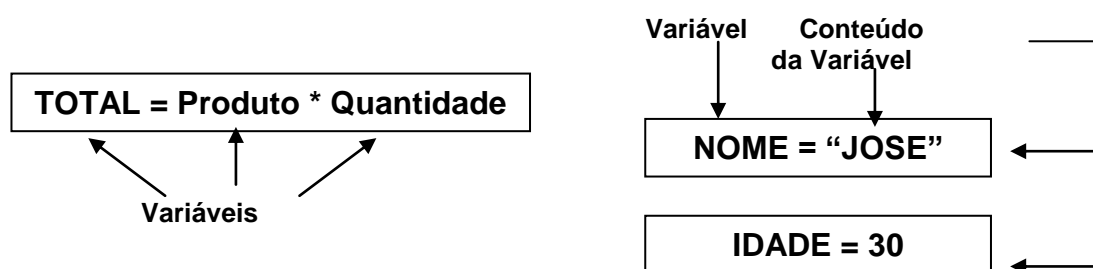
Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.

Exemplo de constantes:

$$\frac{N1 + N2 + N3}{3} \leftarrow \text{Constante}$$

11.2 VARIÁVEIS

Variável é a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante. Exemplos de variáveis:



11.3 TIPOS DE VARIÁVEIS

As variáveis e as constantes podem ser basicamente de quatro tipos: Numéricas, caracteres, alfanuméricas ou lógicas.

11.3.1 Numéricas

Específicas para armazenamento de números, que posteriormente poderão ser utilizados para cálculos. Podem ser ainda classificadas como Inteiras ou Reais. As variáveis do tipo inteiro são para armazenamento de números inteiros e as Reais são para o armazenamento de números que possuam casas decimais.

11.3.2 Caracteres

Específicas para armazenamento de conjunto de caracteres que não contenham números (literais). Ex: nomes.

11.3.3 Alfanuméricas

Específicas para dados que contenham letras e/ou números. Pode em determinados momentos conter somente dados numéricos ou somente literais. Se usado somente para armazenamento de números, não poderá ser utilizada para operações matemáticas.

11.3.4 Lógicas

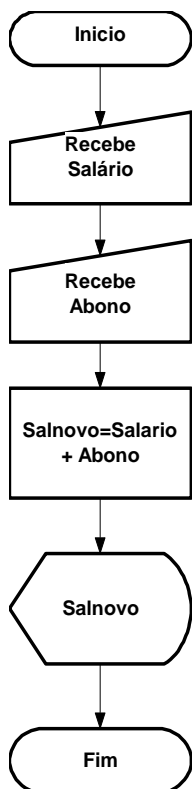
Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso.

11.4 DECLARAÇÃO DE VARIÁVEIS

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo numéricas, lógicas e literais.

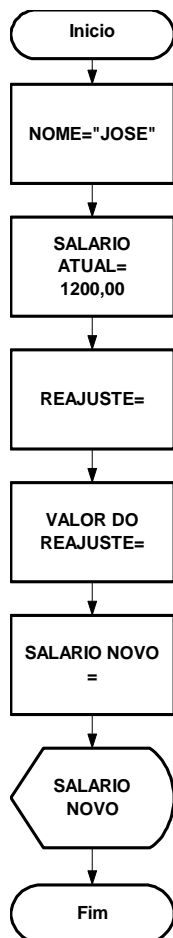
11.5 EXERCÍCIOS

- 1) O que é uma constante? Dê dois exemplos.
- 2) O que é uma variável? Dê dois exemplos.
- 3) Faça um teste de mesa no diagrama de bloco abaixo preenchendo a tabela abaixo com os dados do teste:



Salário	Abono	Salvo
600,00	60,00	660,00
350,00		
980,00		
1500,00		

4) Sabendo-se que José tem direito a 15% de reajuste de salário, complete o diagrama abaixo e simule um teste de mesa:



Nome	Salário	Reajuste	VALOR	Salário Novo
José	1200,00	15%	180,00	1380,00
Antônio	900,00	15%		
Suely	1750,00	15%		

5. OPERADORES

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos três tipos de operadores:

- Operadores Aritméticos
- Operadores Relacionais
- Operadores Lógicos

5.1 OPERADORES ARITMÉTICOS

Os operadores aritméticos são os utilizados para obter resultados numéricos. Além da adição, subtração, multiplicação e divisão, podem utilizar também o operador para exponenciação. Os símbolos para os operadores aritméticos são:

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**

Hierarquia das Operações Aritméticas

1º () Parênteses

2º Exponenciação

3º Multiplicação, divisão (o que aparecer primeiro)

4º + ou – (o que aparecer primeiro)

Exemplo:

TOTAL = PREÇO * QUANTIDADE
1 + 7 * 2 ** 2 - 1 = 28
3 * (1 - 2) + 4 * 2 = 5

5.2 OPERADORES RELACIONAIS

Os operadores relacionais são utilizados para comparar **String** de caracteres e números. Os valores a serem comparados podem ser caracteres ou variáveis.

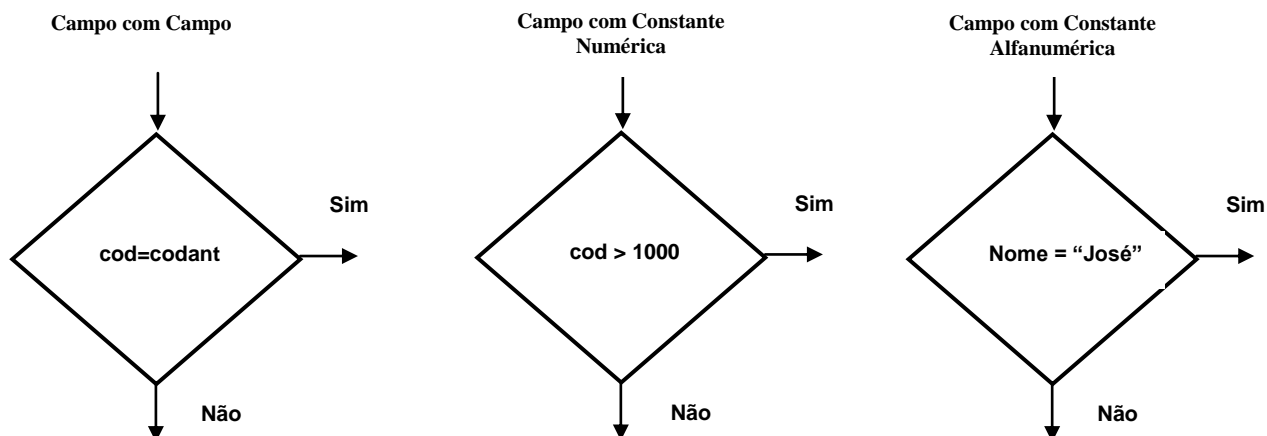
Estes operadores sempre retornam valores lógicos (verdadeiro ou falso/ True ou False). Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses. Os operadores relacionais são:

Descrição	Símbolo
Igual a	=
Diferente de	<>
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Exemplo: Tendo duas variáveis A = 5 e B = 3. Os resultados das expressões seriam:

Expressão	Resultado
A = B	Falso
A <> B	Verdadeiro
A > B	Verdadeiro
A < B	Falso
A >= B	Verdadeiro
A <= B	Falso

Símbolo Utilizado para comparação entre expressões:



5.3 OPERADORES LÓGICOS

Os operadores lógicos servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso. Os operadores lógicos são:

E	AND
OU	OR
NÃO	NOT

AND/E

Uma expressão AND é verdadeira se todas as condições forem verdadeiras

OR/OU

Uma expressão OR é verdadeira se pelo menos uma condição for verdadeira

NOT/NÃO

Uma expressão NOT inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.

A tabela abaixo mostra todos os valores possíveis criados pelos três operadores lógicos (AND, OR e NOT).

1º Valor	Operador	2º Valor	Resultado
T	AND	T	T
T	AND	F	F
F	AND	T	F
F	AND	F	F
T	OR	T	T
T	OR	F	T
F	OR	T	T
F	OR	F	F
T	NOT		F
F	NOT		T

Exemplos:

Suponha que temos três variáveis A = 5, B = 8 e C = 1
Os resultados das expressões seriam:

Expressões			Resultado
A = B	AND	B > C	Falso
A <> B	OR	B < C	Verdadeiro
A > B	NOT		Verdadeiro
A < B	AND	B > C	Verdadeiro
A >= B	OR	B = C	Falso
A <= B	NOT		Falso

5.4 EXERCÍCIOS

1) Tendo as variáveis SALARIO, IR e SALLIQ, e considerando os valores abaixo. Informe se as expressões são verdadeiras ou falsas.

SALARIO	IR	SALLIQ	EXPRESSÃO	V ou F
100,00	0,00	100	(SALLIQ >= 100,00)	
200,00	10,00	190,00	(SALLIQ < 190,00)	
300,00	15,00	285,00	SALLIQ = SALARIO - IR	

2) Sabendo que A=3, B=7 e C=4, informe se as expressões abaixo são verdadeiras ou falsas.

- a) $(A+C) > B$ ()
- b) $B \geq (A + 2)$ ()
- c) $C = (B - A)$ ()
- d) $(B + A) \leq C$ ()
- e) $(C+A) > B$ ()

3) Sabendo que A=5, B=4 e C=3 e D=6, informe se as expressões abaixo são verdadeiras ou falsas.

- a) $(A > C) \text{ AND } (C \leq D)$ ()
- b) $(A+B) > 10 \text{ OR } (A+B) = (C+D)$ ()
- c) $(A \geq C) \text{ AND } (D \geq C)$ ()

5.5 OPERAÇÕES LÓGICAS

Operações Lógicas são utilizadas quando se torna necessário tomar decisões em um diagrama de bloco.

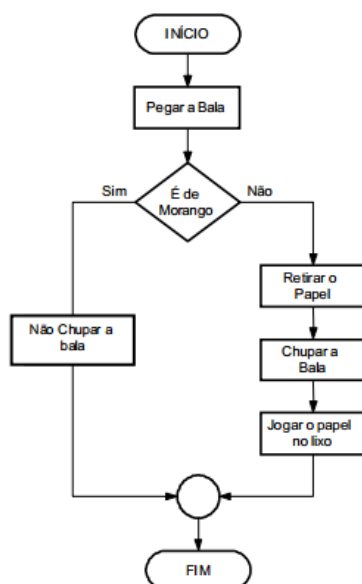
Num diagrama de bloco, toda decisão terá sempre como resposta o resultado VERDADEIRO ou FALSO.

Como no exemplo do algoritmo “CHUPAR UMA BALA”. Imaginemos que algumas pessoas não gostem de chupar bala de Morango, neste caso teremos que modificar o algoritmo para:

“Chupar uma bala”.

- Pegar a bala
- A bala é de morango?
- Se sim, não chupe a bala
- Se não, continue com o algoritmo
- Retirar o papel
- Chupar a bala
- Jogar o papel no lixo

Exemplo: Algoritmo “Chupar Bala” utilizando diagrama de Blocos



5.6 EXERCÍCIOS

- 1) Elabore um diagrama de blocos que leia um número. Se positivo armazene-o em A, se for negativo, em B. No final mostrar o resultado.
- 2) Construa um diagrama de blocos para ler uma variável numérica N e imprimi-la somente se a mesma for maior que 100, caso contrário imprimi-la com o valor zero.

6. ESTRUTURA DE DECISÃO E REPETIÇÃO

Como vimos no capítulo anterior em “Operações Lógicas”, verificamos que na maioria das vezes necessitamos tomar decisões no andamento do algoritmo. Essas decisões interferem diretamente no andamento do programa. Trabalharemos com dois tipos de estrutura. A estrutura de Decisão e a estrutura de Repetição

6.1 COMANDOS DE DECISÃO

Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estruturas de programas que não são totalmente sequenciais. Com as instruções de SALTO ou DESVIO pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores. As principais estruturas de decisão são: “**Se Então**”, “**Se então Senão**” e “**Caso Selecione**”

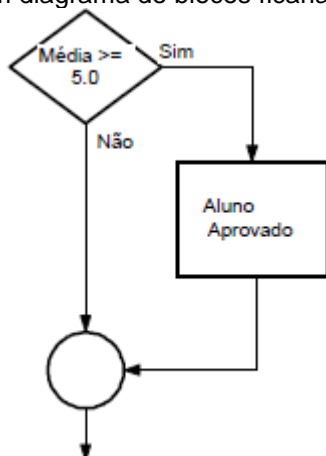
6.1.1 SE ENTÃO / IF ... THEN

A estrutura de decisão “SE/IF” normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando SE/IF então execute determinado comando.

Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5.0, veja no exemplo de algoritmo como ficaria.

SE MEDIA >= 5.0 ENTÃO ALUNO APROVADO

Em diagrama de blocos ficaria assim:



Em Visual Basic

```
IF MEDIA >= 5 Then  
    Text1 = "APROVADO"  
ENDIF
```

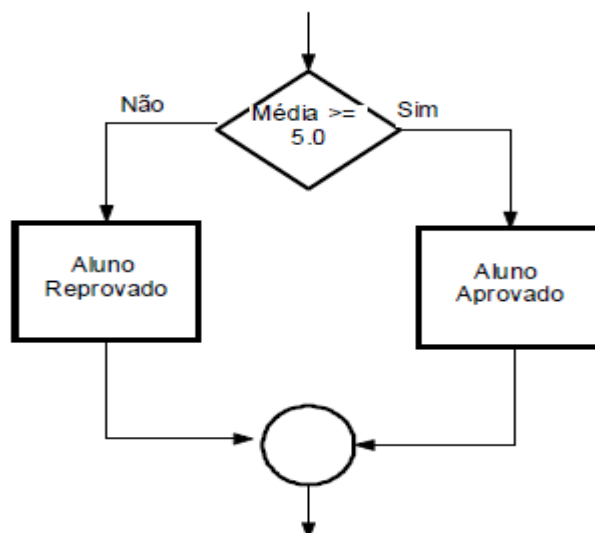
6.1.2 SE ENTÃO SENÃO / IF ... THEN ... ELSE

A estrutura de decisão “SE/ENTÃO/SENÃO”, funciona exatamente como a estrutura “SE”, com apenas uma diferença, em “SE” somente podemos executar comandos caso a condição seja verdadeira, diferente de “SE/SENÃO” pois sempre um comando será executado independente da condição, ou seja, caso a condição seja “verdadeira” o comando da condição será executado, caso contrário o comando da condição “falsa” será executado

Em algoritmo ficaria assim:

```
SE MÉDIA >= 5.0 ENTÃO
    ALUNO APROVADO
SENÃO
    ALUNO REPROVADO
```

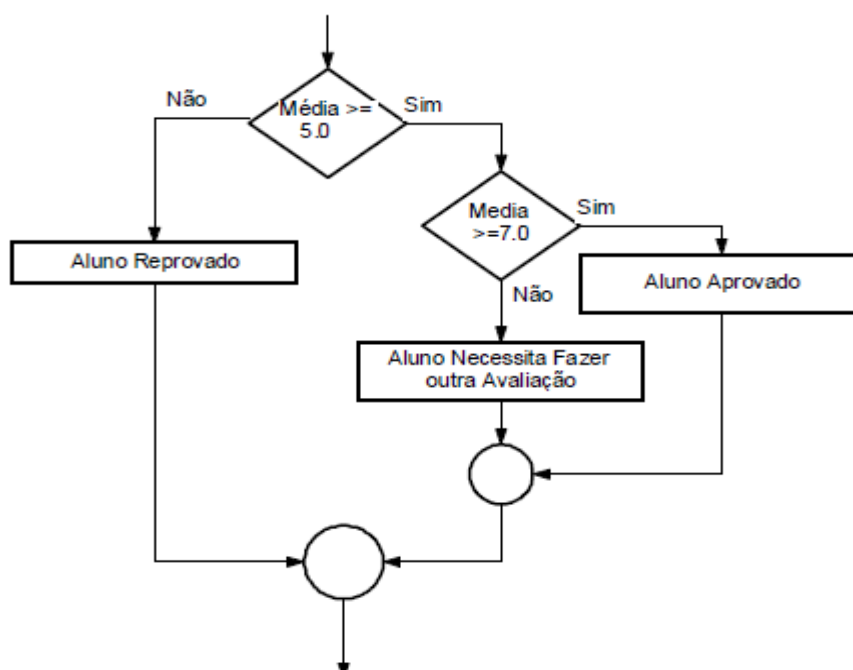
Em diagrama



Em Visual Basic

```
IF MEDIA >= 5 Then
    Text1 = "APROVADO"
ELSE
    Text1 = "REPROVADO"
ENDIF
```

No exemplo acima está sendo executada uma condição que, se for verdadeira, executa o comando “APROVADO”, caso contrário executa o segundo comando “REPROVADO”. Podemos também dentro de uma mesma condição testar outras condições. Como no exemplo a seguir:



Em Visual Basic

```
IF MEDIA >= 5 Then
    IF MEDIA >= 7.0 then
        Text1 = "Aluno APROVADO"
    ELSE
        Text1 = "Aluno Necessita fazer outra Avaliação"
    ENDIF
ELSE
    Text1 = "Aluno REPROVADO"
ENDIF
```

6.1.4 EXERCÍCIOS

1) João Papo-de-Pescador, homem de bem, comprou um microcomputador para controlar o rendimento diário de seu trabalho. Toda vez que ele traz um peso de peixes maior que o estabelecido pelo regulamento de pesca do estado de São Paulo (50 quilos) deve pagar uma multa de R\$ 4,00 por quilo excedente. João precisa que você faça um diagrama de blocos que leia a variável P (peso de peixes) e verifique se há excesso. Se houver, gravar na variável E (Excesso) e na variável M o valor da multa que João deverá pagar. Caso contrário mostrar tais variáveis com o conteúdo ZERO.

2) Elabore um diagrama de bloco que leia as variáveis C e N, respectivamente código e número de horas trabalhadas de um operário. E calcule o salário sabendo-se que ele ganha R\$ 10,00 por hora. Quando o número de horas exceder a 50 calcule o excesso de pagamento armazenando-o na variável E, caso contrário zerar tal variável. A hora excedente de trabalho vale R\$ 20,00. No final do processamento imprimir o salário total e o salário excedente.

3) Desenvolva um diagrama que:

- Leia 4 (quatro) números;
- Calcule o quadrado de cada um;
- Se o valor resultante do quadrado do terceiro for ≥ 1000 , imprima-o e finalize;
- Caso contrário, imprima os valores lidos e seus respectivos quadrados.

7.2 COMANDOS DE REPETIÇÃO

Utilizamos os comandos de repetição quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado.

Trabalharemos com modelos de comandos de repetição:

- Enquanto x, processar (**Do While ...Loop**);
- Até que x, processar ... (**Do Until ... Loop**);
- Processar ..., Enquanto x (**Do ... Loop While**);
- Processar ..., Até que x (**Do ... Loop Until**)
- Para ... Até ... Seguinte (**For ... To ... Next**)

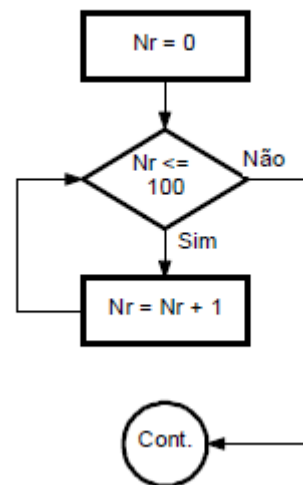
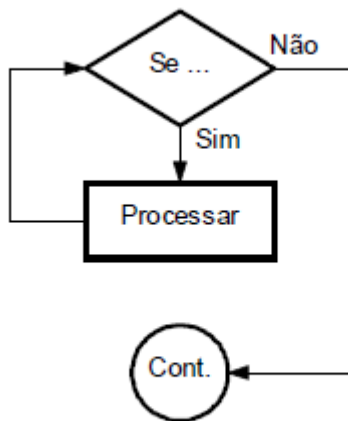
7.2.1 ENQUANTO X, PROCESSAR (DO WHILE ... LOOP)

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação.

Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

Em diagrama de bloco a estrutura é a seguinte:

Exemplo de Contador



Em Visual Basic:

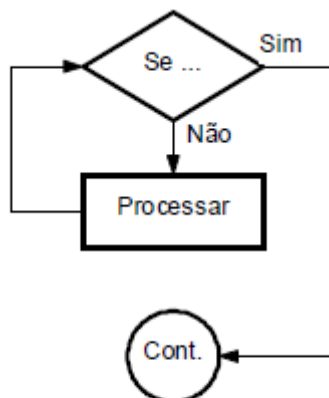
```

Nr = 0
Do While Nr <= 100
    Nr = Nr + 1
Loop
    
```

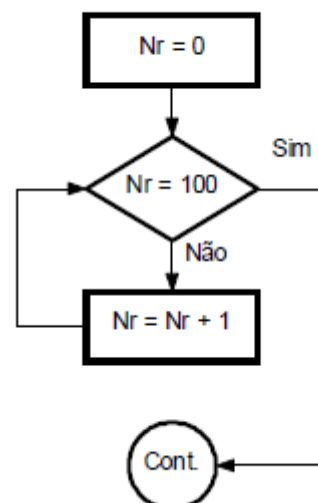
7.2.2 Até que x, processar ... (Do Until ... Loop)

Neste caso, o bloco de operações será executado até que a condição seja satisfeita, ou seja, somente executará os comandos enquanto a condição for falsa.

Em diagrama de bloco:



Exemplo de Até Diagrama



Em Visual Basic:

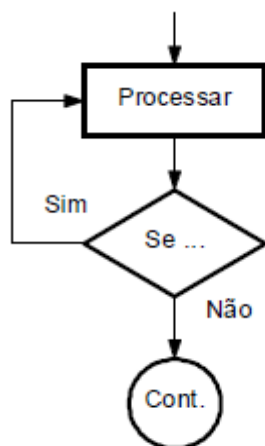
```

Nr = 0
Do Until Nr = 100
    Nr = Nr + 1
Loop
Label1.Caption = Nr
    
```

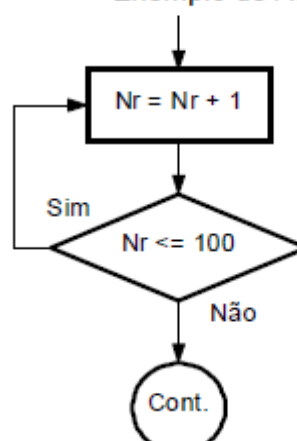

7.2.3 PROCESSAR ... ENQUANTO X (DO ... LOOP WHILE)

Neste caso primeiro são executados os comandos, e somente depois é realizado o teste da condição. Se a condição for verdadeira, os comandos são executados novamente, caso seja falso é encerrado o comando DO.

Em diagrama de bloco:



Exemplo de Até Diagrama



Em Visual Basic:

Nr = 0

Do

Nr = Nr + 1

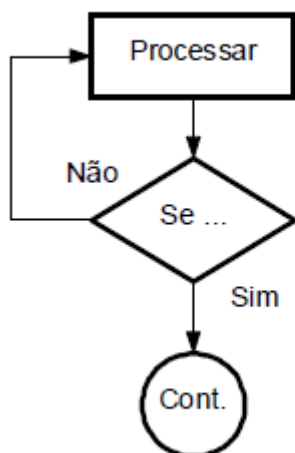
Loop While Nr <= 100

Label1.Caption = Nr

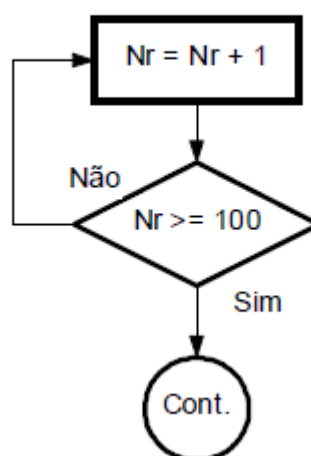
7.2.4 PROCESSAR ... ATÉ QUE X (DO ... LOOP UNTIL)

Neste caso, executa-se primeiro o bloco de operações e somente depois é realizado o teste de condição. Se a condição for verdadeira, o fluxo do programa continua normalmente. Caso contrário é processado novamente os comandos antes do teste da condição.

Em diagrama de Bloco



Exemplo de Do Loop - Until



Em Visual Basic:

```
nr = 0
```

```
Do
```

```
    nr = nr + 1
```

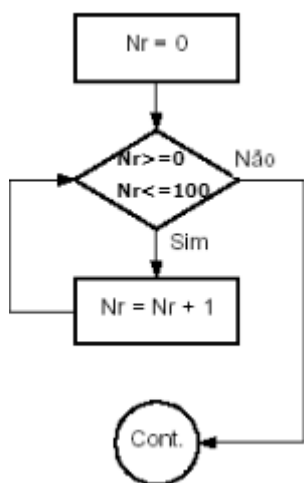
```
Loop Until nr >= 100
```

```
Label1.Caption = nr
```

7.2.5 PARA ... ATÉ ... SEGUINTE (For ... To ... Next)

Neste caso, é usada quando uma sequência de comandos precisa ser repetida um número de vezes pré-fixado.

Em diagrama de Bloco



Em Visual Basic:

```
nr = 0
```

```
For nr = 1 to 100
```

```
    Label1.Caption = nr
```

```
Next nr
```

7.2.6 EXERCÍCIOS:

1) Faça um diagrama que determine o maior entre **N** números. A condição de parada é a entrada de um valor 0, ou seja, o algoritmo deve ficar calculando o maior até que a entrada seja igual a 0 (ZERO).

2) Faça um diagrama que conte de 1 a 100 e a cada múltiplo de 10 emita uma mensagem: "Múltiplo de 10".

8. COMPUTAÇÃO FÍSICA

A computação física significa a construção de sistemas interativos físicos mediante o uso de software e hardware que integrados podem sentir e responder ao mundo analógico. Embora esta definição seja ampla o suficiente para englobar aspectos como os sistemas inteligentes de controle de tráfego de automóveis ou os processos de automatização em fábricas, em um sentido mais amplo a computação física é uma estrutura criativa para a compreensão da relação entre os seres humanos e o mundo digital.

Na prática, frequentemente este termo descreve desenhos de projetos DIY(**Do It Yourself** - do inglês *faça você mesmo*) ou objetos que utilizam sensores e microcontroladores

para traduzir entradas analógicas a sistemas baseados em software, ou controlar dispositivos eletromecânicos como motores, servos, iluminação ou outro hardware.

Outras implementações de computação física trabalham com o reconhecimento de voz, que captam e interpretam as ondas sonoras através de microfones ou outros dispositivos de detecção de ondas sonoras, também a visão por computador, que aplica algoritmos aos vídeos detectados por algum tipo de câmera. Interfaces táteis são também um exemplo de computação física.

O prototipado (criar montagens rápidas com ajuda de uma protoboard e componentes básicos de eletrônica) tem um papel importante na computação física. Ferramentas como o Arduino e o Fritzing são úteis para designers, artistas, estudantes e hobistas porque ajudam a elaborar protótipos rapidamente.



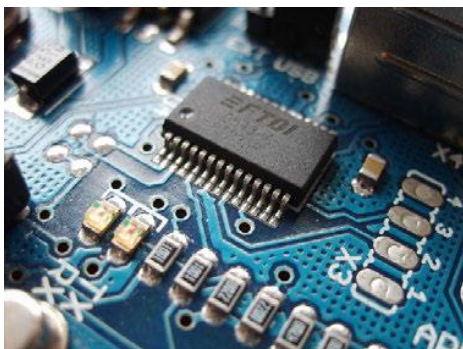
Arduino Uno

9. ELETRÔNICA

Numa definição mais abrangente, podemos dizer que a eletrônica é o ramo da ciência que estuda o uso de circuitos formados por componentes elétricos e eletrônicos, com o objetivo principal de representar, armazenar, transmitir ou processar informações além do controle de processos e servo mecanismos.

Sob esta ótica, também se pode afirmar que os circuitos internos dos computadores, os sistemas de telecomunicações, os diversos tipos de sensores e transdutores estão, todos, dentro da área de interesse da eletrônica.

Divide-se em analógica e em digital porque suas coordenadas de trabalho optam por obedecer estas duas formas de apresentação dos sinais elétricos a serem tratados. Também é considerada um ramo da eletricidade que, por sua vez, é um ramo da Física onde se estudam os fenômenos das cargas elétricas elementares, as propriedades e comportamento do elétron, fótons, partículas elementares, ondas eletromagnéticas, etc.



Circuito integrado

9.1 VOLTAGEM

Voltagem é a magnitude física que, em um circuito elétrico, impulsiona os elétrons ao longo de um condutor. Isto é, conduz a energia elétrica com maior ou menor potência. Voltagem e Volt são termos em homenagem a Alessandro Volta que, em 1800, inventou a pilha voltaica e a primeira bateria química.

A voltagem é um sinônimo de tensão e de diferença de potencial. Em outras palavras, a voltagem é o trabalho por unidade de carga exercida pelo campo elétrico sobre uma partícula para que esta se mova de lugar ao outro. No Sistema Internacional de Unidades, esta diferença de potencial é medida em volts (V), e isto determina a categorização em "baixa" ou "alta voltagem".

9.2 CORRENTE ELÉTRICA

A corrente elétrica é o fluxo ordenado de partículas portadoras de carga elétrica, ou também, é o deslocamento de cargas dentro de um condutor, quando existe uma diferença de potencial elétrico entre as extremidades.

A unidade padrão no Sistema Internacional de Unidades para medir a intensidade de corrente é o ampere. Para medir a corrente, pode-se utilizar um amperímetro.

Uma corrente elétrica, já que se trata de um movimento de cargas, produz um campo magnético, um fenômeno que pode ser usado como um eletroímã, sendo este o princípio de funcionamento de um motor.

9.3 RESISTÊNCIA

Resistência elétrica é a capacidade de um corpo qualquer se opor à passagem de corrente elétrica mesmo quando existe uma diferença de potencial aplicada. É medida em ohms (Ω). Resistores são componentes que têm por finalidade oferecer uma oposição à passagem de corrente elétrica, através de seu material. A essa oposição damos o nome de resistência elétrica. Causam uma queda de tensão em alguma parte de um circuito elétrico, porém jamais causam quedas de corrente elétrica, apesar de limitar a corrente. Isso significa que a corrente elétrica que entra em um terminal do resistor será exatamente a mesma que sai pelo outro terminal, porém há uma queda de tensão.

Utilizando-se disso, é possível usar os resistores para controlar a tensão sobre os componentes desejados.



Resistores

9.4 SISTEMAS ELETRÔNICOS

Um sistema eletrônico é um conjunto de circuitos que interagem entre si para obter um resultado.

Uma forma de entender os sistemas eletrônicos consiste em dividi-los em entradas, saídas e processamento de sinais, conforme esquema a seguir:



As entradas, ou inputs, são sensores eletrônicos ou mecânicos que tomam os sinais (em forma de temperatura, pressão, umidade, contato, luz, movimento, pH, etc.) do mundo físico e converte em sinais de corrente ou voltagem.

As saídas, ou outputs, são atuadores, ou outros dispositivos que convertem os sinais de corrente ou voltagem em sinais fisicamente úteis como movimento, luz, som, força ou rotação,

entre outros. Exemplos de saídas são motores, LEDs ou sistemas de luzes que acendem automaticamente quando escurece ou um buzzer que gere diversos tons.

O processamento será o local onde os dados absorvidos na entrada receberão uma espécie de tratamento, onde passará pela parte que chamamos “inteligente” do processo, onde de acordo com regras pré-estabelecidas os dados serão processados e enviados às saídas.

Como exemplo imaginamos um aparelho de TV. A entrada é um sinal recebido por uma antena ou um cabo. Os circuitos integrados do interior do aparelho extraem a informação sobre brilho, cor e som deste sinal. Os dispositivos de saída são a tela LCD, que converte os sinais eletrônicos em imagens visíveis, e as caixas de som, que emitem o som.

Outro exemplo pode ser um circuito que controle a temperatura de um ambiente. Um sensor de temperatura e um circuito integrado são os responsáveis por converter um sinal de entrada em um nível de voltagem apropriado. Se a temperatura registrada do ambiente é muito alta, este circuito enviará a informação a um motor para que este ligue um ventilador que resfriará o local.



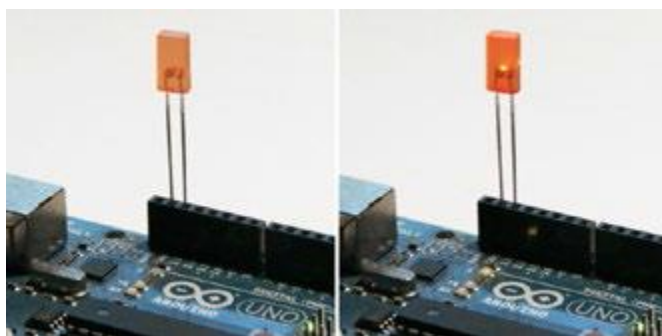
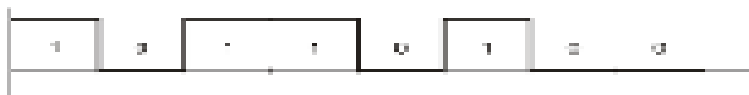
Exemplos de entrada-processamento-saída

As entradas e saídas de um sistema eletrônico serão consideradas como sinais variáveis. Em eletrônica se trabalha com variáveis que são tomadas na forma de tensão ou corrente, que podem simplesmente ser chamados de sinais. Os sinais podem ser de dois tipos: digital ou analógico.

9.5 VARIÁVEL DIGITAL

Também chamadas de variáveis discretas, se caracterizam por ter dois estados diferentes e portanto também podem ser chamadas de binárias (em lógica seria valores Verdadeiro (V) e Falso (F), ou poderiam ser 1 ou 0 respectivamente).

Um exemplo de um sinal digital é o interruptor da campainha da sua casa, porque ele tem somente dois estados, pulsado e sem pulsar.



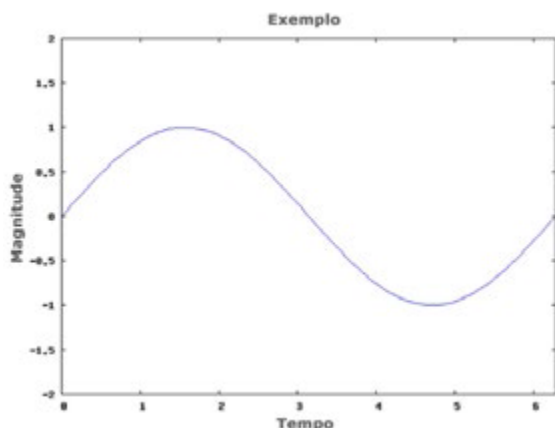
Exemplos de sinais digitais

9.6 VARIÁVEL ANALÓGICA

São aquelas que podem tomar um número infinito de valores compreendidos entre dois limites. A maioria dos fenômenos da vida real são sinais deste tipo (som, temperatura, luminosidade, etc.).

Um exemplo de sistema eletrônico analógico é de um palestrante, que se preocupa em amplificar o som da sua voz para que seja escutado por uma grande audiência. As ondas de

som que são analógicas na sua origem são capturadas por um microfone e convertidas em uma pequena variação analógica de tensão, denominada sinal de áudio.



Exemplo de variáveis analógicas

9.7 ENTRADA/SAÍDA DIGITAL



Entrada (Botão)



Saída (LED)

9.8 ENTRADA/SAÍDA ANALÓGICA



Entrada (LDR)



Entrada (Potenciômetro)



Saída (Moto)

10 INTRODUÇÃO AO ARDUINO

Desde que o Arduino Project teve início em 2005, mais de 150.000 placas Arduino foram vendidas em todo o mundo. O número de placas-clone não oficiais sem dúvida supera o de placas oficiais, assim, é provável que mais de 500 mil placas Arduino e suas variantes tenham sido vendidas. Sua popularidade não para de crescer, e cada vez mais pessoas percebem o incrível potencial desse maravilhoso projeto de fonte aberta para criar projetos interessantes de forma rápida e fácil, com uma curva de aprendizagem relativamente pequena.

A maior vantagem do Arduino sobre outras plataformas de desenvolvimento de microcontroladores e a facilidade de sua utilização; pessoas que não são da área técnica podem, rapidamente, aprender o básico e criar seus próprios projetos em um intervalo de tempo relativamente curto. Artistas, mais especificamente, parecem considera-lo a forma perfeita de criar obras de arte interativas rapidamente, e sem conhecimento especializado em eletrônica. Há uma grande comunidade de pessoas utilizando Arduinos, compartilhando seus códigos e diagramas de circuito para que outros os copiem e modifiquem. A maioria dessa comunidade também está muito disposta a auxiliar outros desenvolvedores. Você descobrirá que o Fórum do Arduino é o melhor local para buscar por respostas rápidas.

Entretanto, apesar da enorme quantidade de informação disponível aos iniciantes na Internet, a maioria desses dados está espalhada em várias fontes, fazendo com que seja complicado rastrear as informações necessárias.

10.1 OPEN SOURCE HARDWARE

Open Source Hardware consiste em dispositivos físicos de tecnologia concebidos e oferecidos pelo movimento de design aberto. Tanto o software livre como o open source hardware são criados sob o movimento de cultura open source e aplica este conceito a uma variedade de componentes. O termo normalmente significa que a informação sobre o hardware é facilmente reconhecida. O design no hardware (ou seja, desenhos mecânicos, esquemas, lista de materiais, dados de layout do PCB, código fonte e dados de layout de circuitos integrados), além do software livre que aciona o hardware, estão todos liberados com a abordagem livre e open source.



Logotipo

10.2 SOFTWARE LIVRE

Software livre é o software que é distribuído juntamente com o seu código-fonte, e é liberado sob os termos que garantem aos usuários a liberdade de estudar, adaptar/modificar e distribuir o software. O software livre é muitas vezes desenvolvido em colaboração entre programadores voluntários como parte de um projeto de desenvolvimento de software open source.

A Free Software Foundation considera um software como livre quando atende aos quatro tipos de liberdade para os usuários:

Liberdade 0: A liberdade para executar o programa, para qualquer propósito;

Liberdade 1: A liberdade de estudar o software;

Liberdade 2: A liberdade de redistribuir cópias do programa de modo que você possa ajudar ao seu próximo;

Liberdade 3: A liberdade de modificar o programa e distribuir estas modificações, de modo que toda a comunidade se beneficie.

Os usuários deste tipo de software são livres porque não precisam pedir permissão e não estão vinculados a licenças proprietárias restritivas.

A Open Source Initiative (OSI) - Iniciativa pelo Código Aberto - é uma organização dedicada a promover o software de código aberto ou software livre. Ela foi criada para incentivar uma aproximação de entidades comerciais com o software livre. Sua atuação principal é a de certificar quais licenças se enquadram como licenças de software livre, e promovem a divulgação do software livre e suas vantagens tecnológicas e econômicas. A OSI, assim como muitos membros da comunidade, considera que o software é, em primeiro lugar, uma ferramenta, e que o mérito dessa ferramenta deve ser julgado com base em critérios técnicos. Para eles, o software livre no longo prazo é economicamente mais eficiente e de melhor qualidade e, por isso, deve ser incentivado. Além disso, a participação de empresas no ecossistema do software livre é considerada fundamental, pois são as empresas que viabilizam o aumento no desenvolvimento, implantação e uso do software livre.

10.3 O QUE É ARDUINO?

O Arduino é um projeto totalmente aberto de protótipos de eletrônica baseados numa plataforma de hardware e software flexível e de fácil utilização. É destinado a artistas, designers, hobbyistas e qualquer tipo de pessoa interessada em criar objetos ou ambientes interativos. É um projeto que engloba software e hardware e tem como objetivo fornecer uma plataforma fácil para prototipação de projetos interativos, utilizando um microcontrolador. Ele faz parte do que chamamos de computação física: área da computação em que o software interage diretamente com o hardware, tornando possível integração com sensores, motores e outros dispositivos eletrônicos.

O Arduino pode perceber o ambiente por receber informação de uma grande variedade de sensores, e pode estimular o ambiente controlando luzes, motores, e outros atuadores.

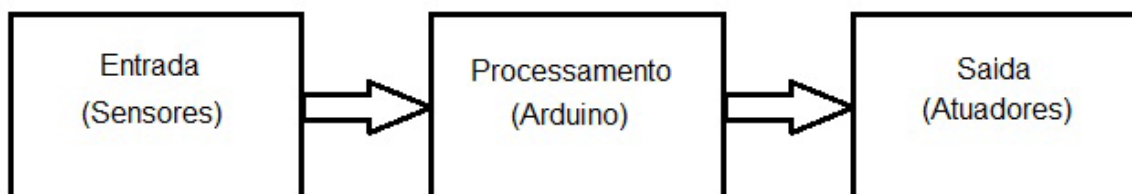
A parte de hardware do projeto, uma placa que cabe na palma da mão, é um computador como qualquer outro: possui microprocessador, memória RAM, memória flash (para guardar o software), temporizadores, contadores, dentre outras funcionalidades. Atualmente, o projeto está na versão Uno, porém muitos Arduinos encontrados hoje são da versão Duemilanove (2009, em italiano), que possui um clock de 16 MHz, 2 Kb de memória RAM, 32 Kb de memória flash, 14 portas digitais e 6 entradas analógicas.

O microcontrolador em que se baseia (ATMEL) é programável usando a linguagem Arduino (baseada em C/C++), e também aceita código diretamente em C/C++. Normalmente, é necessário construir os circuitos para as entradas e saídas do Arduino, o que permite flexibilidade e uma grande variedade de soluções para um mesmo problema. Muitos projetos para Arduino estão disponíveis na internet (o próprio site do Arduino mantém um fórum e um blog para os usuários do sistema), facilitando o aprendizado e a troca de informações entre os construtores. Os projetos em Arduino podem ser únicos ou podem comunicar com outros circuitos, ou até mesmo com outros softwares em um computador (por exemplo, Java, Flash, Processing, MaxMSP).

As placas podem ser montadas à mão ou serem compradas montadas, e o software pode ser obtido gratuitamente.

Como o Arduino é um projeto aberto, diversas empresas fabricam e disponibilizam suas placas no mercado, como o Freeduino, Seeduino, Roboduino, Pinguino e os brasileiros Severino e Brasuino.

Em resumo, o Arduino é um kit de desenvolvimento, que pode ser visto como uma unidade de processamento capaz de mensurar variáveis do ambiente externo, transformadas em um sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada. De posse da informação, ele pode processá-la computacionalmente. Porém, ele pode ainda atuar no controle ou no acionamento de algum outro elemento eletroeletrônico conectado ao terminal de saída. A Figura a seguir apresenta um diagrama de blocos de uma cadeia de processamento utilizando o Arduino.



10.4 PLACA DO ARDUINO UNO

A placa Arduino UNO já está em sua terceira revisão e você pode baixar seu esquema elétrico em formato PDF no site do Arduino, ou até mesmo todos os arquivos do projeto para edição. Ela tem duas camadas apenas e várias características interessantes de projeto. A seguir serão apresentadas as principais características do seu hardware.

10.4.1 ALIMENTAÇÃO DA PLACA ARDUINO UNO

A placa pode ser alimentada pela conexão USB ou por uma fonte de alimentação externa, conforme exibido na figura a seguir:

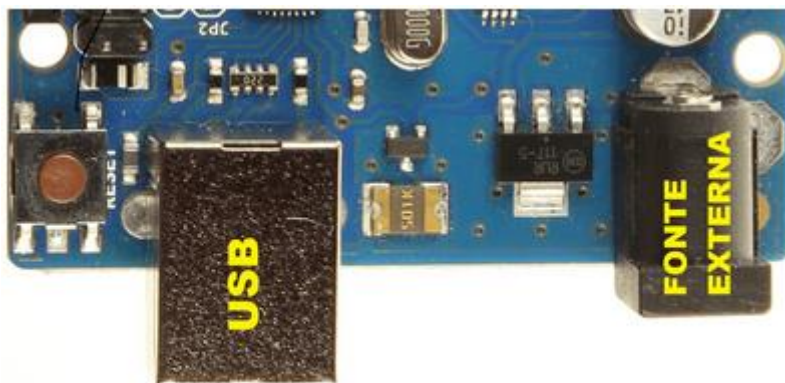


Figura 1 - Alimentação da placa Arduino UNO

A alimentação externa é feita através do conector Jack com positivo no centro, onde o valor de tensão da fonte externa deve estar entre os limites 6V. a 20V., porém se alimentada com uma tensão abaixo de 7V., a tensão de funcionamento da placa, que no Arduino Uno é 5V, pode ficar instável e quando alimentada com tensão acima de 12V, o regulador de tensão da placa pode sobreaquecer e danificar a placa. Dessa forma, é recomendado para tensões de fonte externa valores de 7V. a 12V.

A seguir são exibidos os conectores de alimentação para conexão de shields e módulos na placa Arduino UNO:

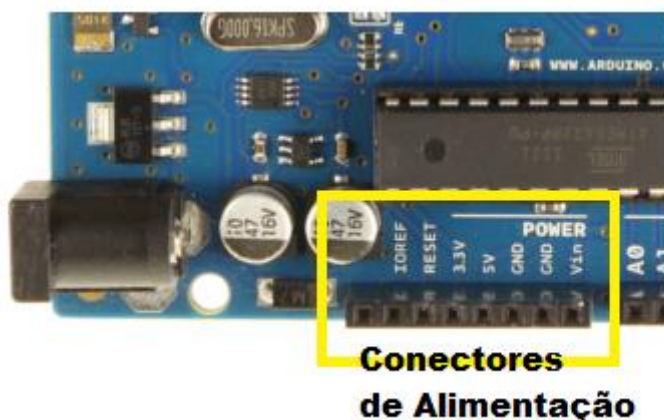


Figura 2 – conectores de alimentação

IOREF - Fornece uma tensão de referência para que shields possam selecionar o tipo de interface apropriada, dessa forma shields que funcionam com a placas Arduino que são alimentadas com 3,3V. podem se adaptar para ser utilizados em 5V. e vice-versa.

RESET - pino conectado a pino de RESET do microcontrolador. Pode ser utilizado para um reset externo da placa Arduino.

3,3 V. - Fornece tensão de 3,3V. para alimentação de shield e módulos externos. Corrente máxima de 50 mA.

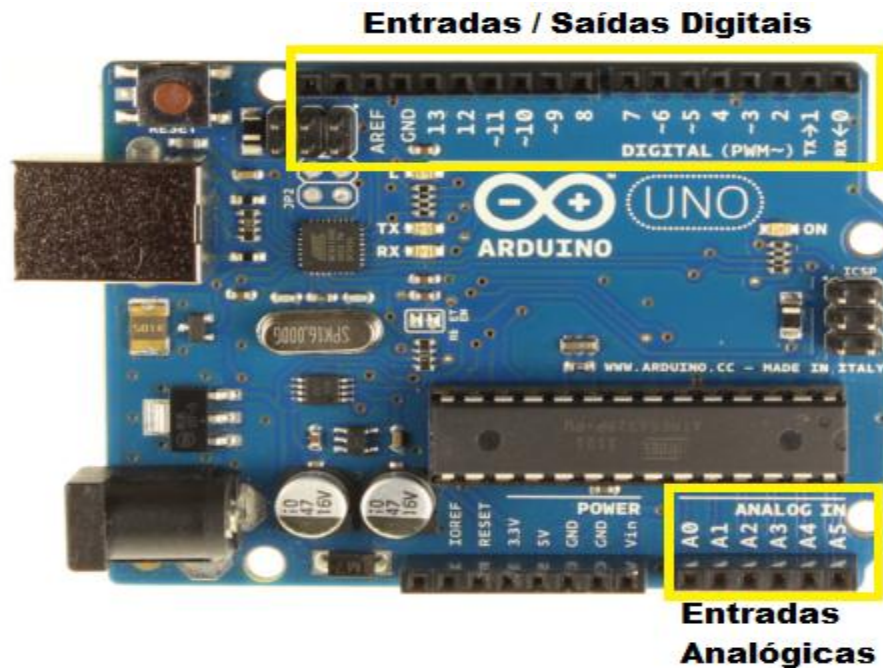
5 V - Fornece tensão de 5 V para alimentação de shields e circuitos externos.

GND - pinos de referência, terra.

VIN - pino para alimentar a placa através de shield ou bateria externa. Quando a placa é alimentada através do conector Jack, a tensão da fonte estará nesse pino.

10.4.2 ENTRADAS E SAÍDAS DO ARDUINO UNO

A placa Arduino UNO possui pinos de entrada e saídas digitais, assim como pinos de entradas e saídas analógicas, abaixo é exibido a pinagem conhecida como o padrão Arduino:



Sem dúvida a placa Arduino UNO é uma ótima ferramenta para quem está começando. É uma ferramenta simples e possui um hardware mínimo, com várias características interessantes de projeto. Sua conectividade USB e facilidade em programar é, sem dúvida nenhuma, um grande atrativo.

É importante lembrar que a placa Arduino não possui a facilidade de debugar em tempo real, como outras placas de desenvolvimento. Não é possível colocar breakpoints, consultar variáveis ou mesmo parar o firmware em tempo real para conferir endereços de memória ou variáveis.

11. ARDUINO EM WINDOES

11.1 PLACA ARDUINO E UM CABO USB AB



11.2 – DOWNLOAD DA IDE DO ARDUINO

Faça download da última versão do software do Arduino (<https://www.arduino.cc/en/main/software>).

11.3 – CONECTANDO O ARDUINO

O Arduino Uno isolado usa a energia do computador através da conexão USB, não sendo necessária energia externa. Conecte a placa Arduino ao computador usando o cabo USB AB. O LED verde de energia (PWR) deve acender.

11.4 – INSTALANDO OS DRIVERS

Drivers para Arduino Uno ou Arduino Mega 2560 com Windows 7, Vista ou XP:

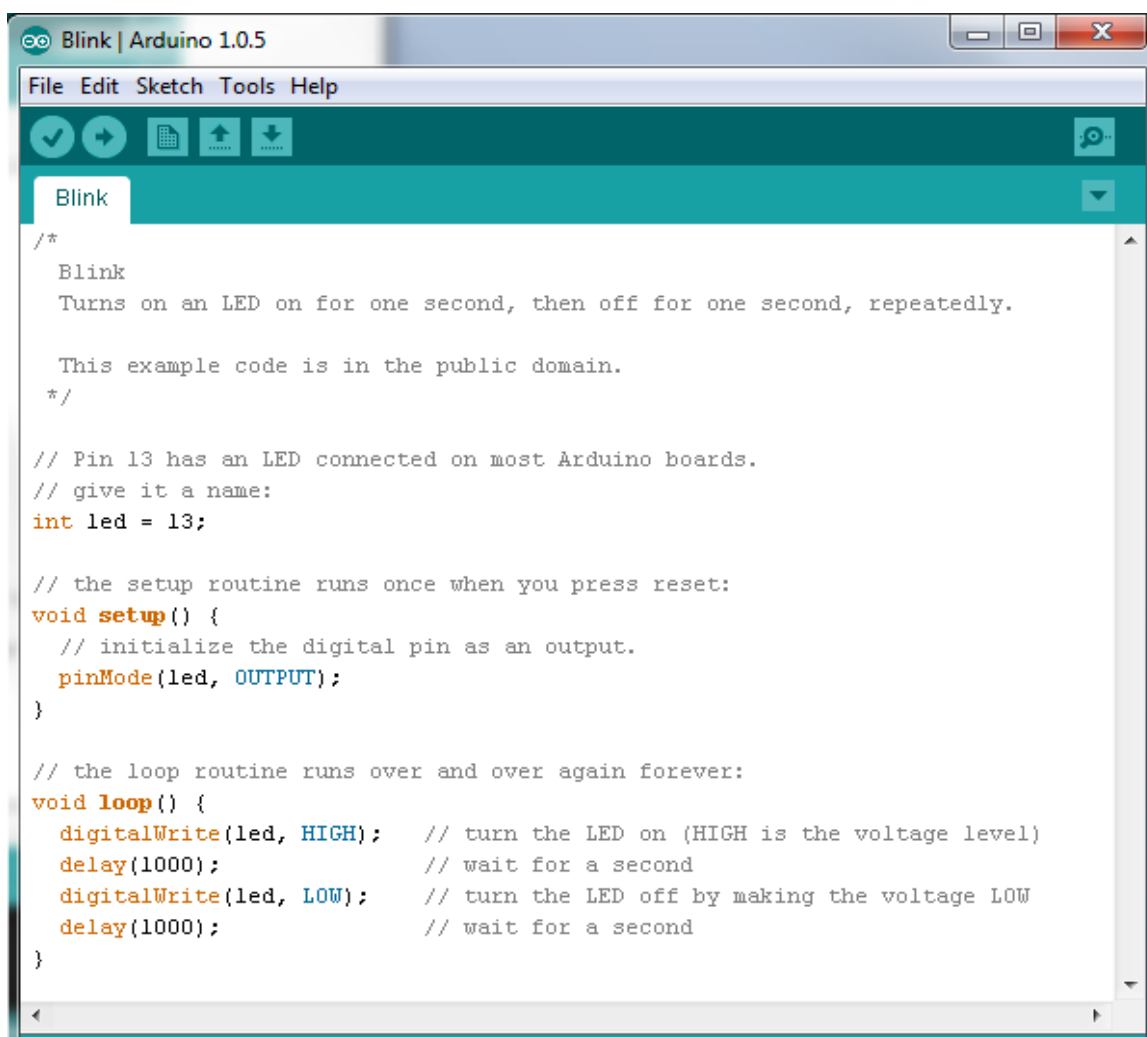
- Conecte a placa ao computador e aguarde o Windows iniciar o processo de instalação do driver. Depois de alguns momentos o processo vai falhar. Clique em concluir e dispense a ajuda do assistente.
- Clique no Menu Principal e abra o Painel de Controle.
- Dentro do Painel de Controle, navegue até Sistema e Segurança. Na sequência clique em Sistema, selecione Hardware e depois clique em Gerenciador de Dispositivos.
- Procure por Portas (COM & LPT), onde você deve ver uma opção Arduino UNO (COMxx).
- Clique com o botão da direita em Arduino UNO (COMxx) e escolha a opção Atualizar Driver.
- Depois escolha a opção Instalar de uma lista ou local específico (Avançado), e clique em avançar.
- Finalmente navegue e escolha o driver arduino.inf localizado na pasta Drivers do software do Arduino que você baixou.
- O Windows vai finalizar a instalação do driver a partir deste ponto.

11.5 – ABRINDO O PROGRAMA ARDUINO

Clique duas vezes na aplicação do Arduino, o arquivo arduino.exe. Caso o programa carregue com o idioma que não é da sua preferência você pode alterar na sessão de preferências do programa.

11.6 – EXEMPLO PISCAR

Abra o exemplo Piscar (blink): Arquivo > Exemplos > 01.Basics > Blink



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

  This example code is in the public domain.
  */

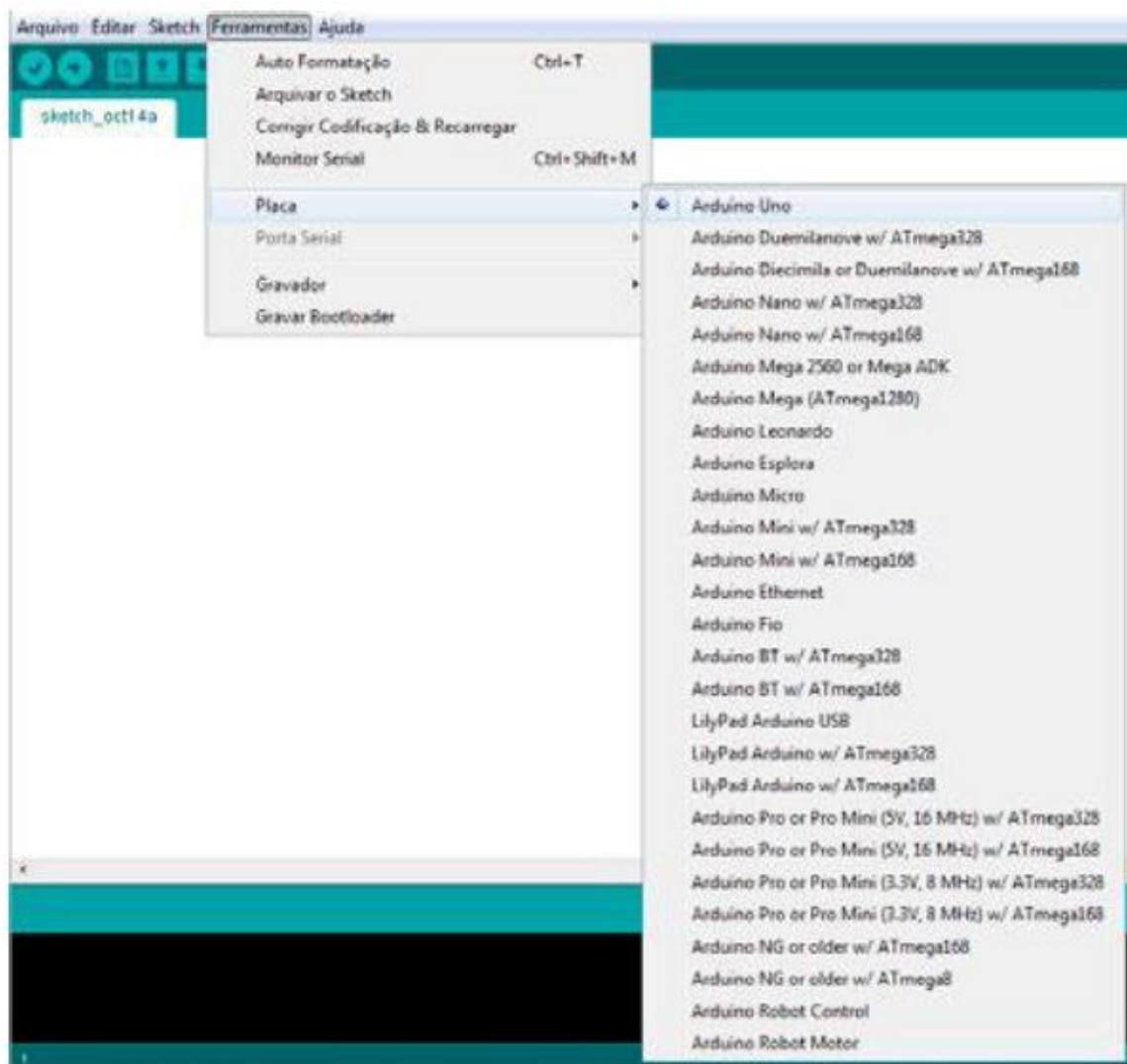
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

11.7 – SELECIONANDO SUA PLACA

Você deve selecionar qual a sua placa Arduino: Ferramentas > Placa > Arduino Uno.



11.8 – SELECIONANDO A PORTA

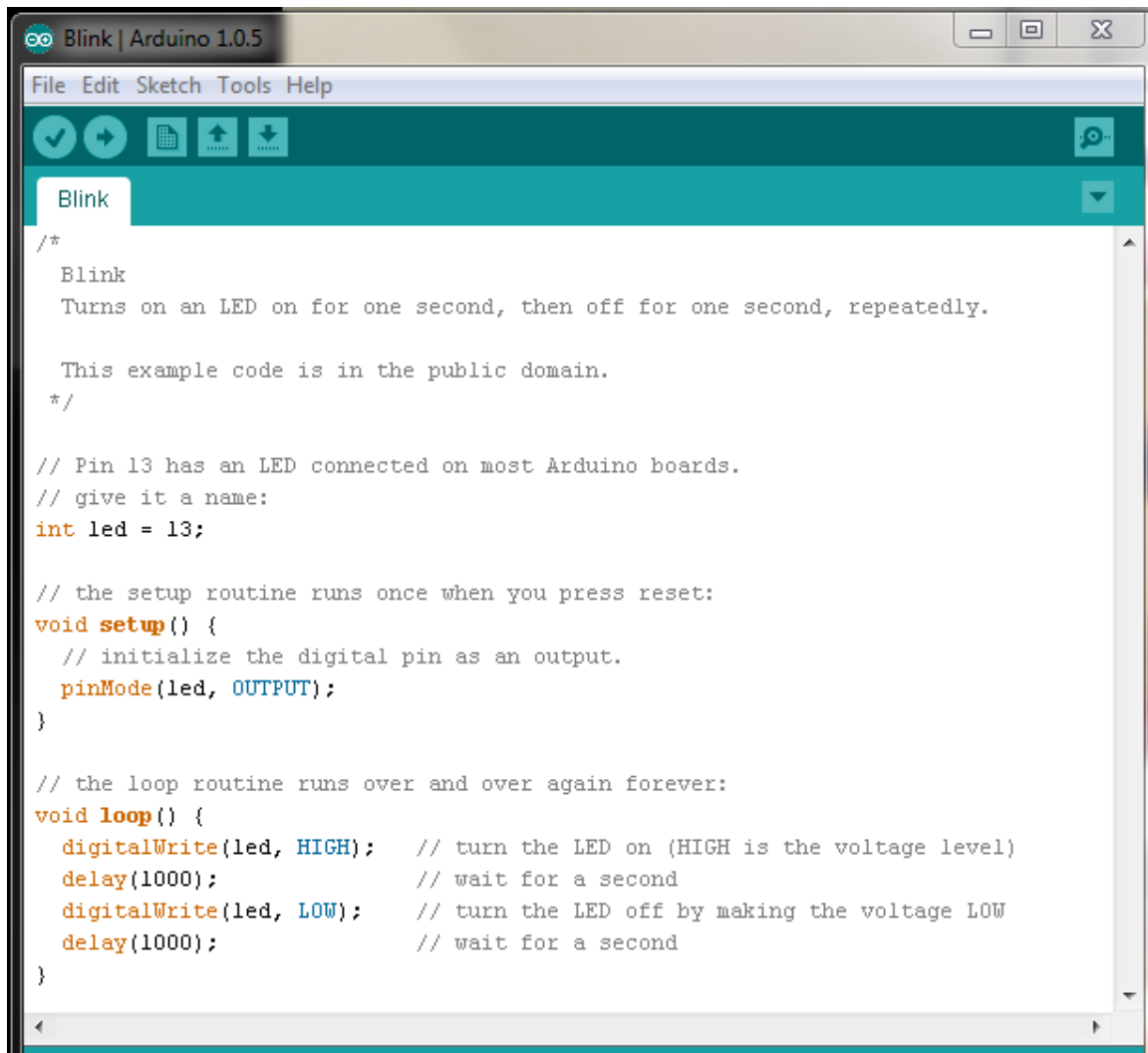
Selecione agora a porta serial que conectará o Arduino: Ferramentas > Porta Serial. Você deve selecionar a mesma porta que utilizou para configurar o sistema no passo 4.

11.9 – CARREGUE O PROGRAMA

Agora simplesmente clique no botão Carregar da janela do programa. Espere alguns segundos. Você deve ver os LEDs RX e TX da placa piscarem. Se o processo foi executado normalmente você verá uma mensagem de “Transferência concluída”.

Depois de alguns segundos você verá o LED do pin 13 piscar em laranja. Neste caso, parabéns! Seu Arduino está pronto e instalado.

Se você tiver problemas na instalação pode acessar a página oficial do Arduino (<http://arduino.cc/en/Guide/Troubleshooting>) com algumas soluções.



```

Blink | Arduino 1.0.5
File Edit Sketch Tools Help
Blink
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

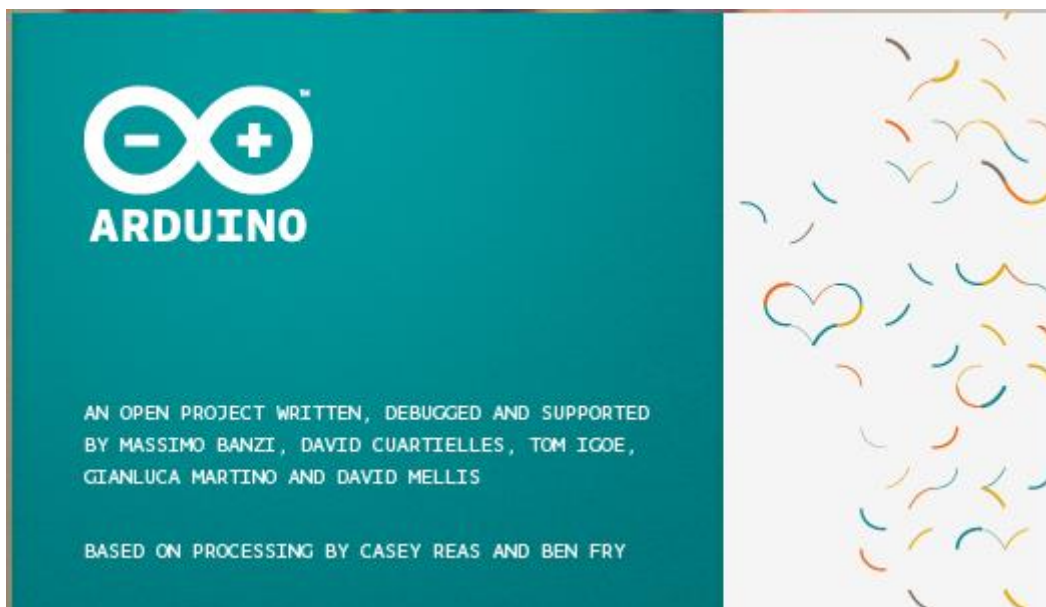
  This example code is in the public domain.
  */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

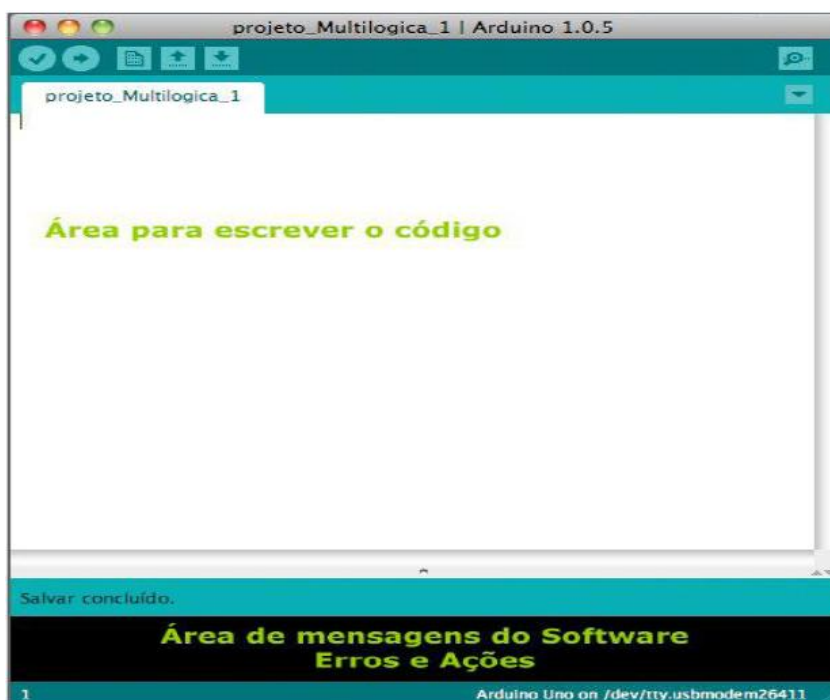
13. Software Arduino



Para executar o programa entramos na pasta do Arduino guardada no computador e procuramos o ícone. Clique duas vezes para abrir o programa.

O programa do Arduino também é conhecido como IDE Arduino (Integrated Development Environment) pois além do entorno de programação consiste também em um editor de código, um compilador e um depurador.

Espaço de trabalho:



13.1 SKETCHES

Softwares escritos usando Arduino são chamados de Sketches. Estes Sketches são escritos no editor de texto da IDE do Arduino e são salvos com a extensão de arquivo .ino. Este editor tem características de cortar/colar e para buscar/substituir texto. A área de mensagem dá feedback ao salvar e exportar arquivos e também exibe informações de erros ao compilar Sketches. O canto direito inferior da janela exibe a placa atual e a porta serial. Os botões da barra de ferramentas permitem que você verifique, carregue, crie, abra e salve Sketches ou abra o monitor serial.

Nota: Nas versões do IDE antes de 1.0 os Sketches são salvos com a extensão .pde. É possível abrir esses arquivos com a versão 1.0, mas você será solicitado a salvar o Sketch com a extensão .ino.



Verificar

Verifica se seu código tem erros.



Carregar

Compila seu código e carrega para a placa Arduino.



Novo

Cria um novo Sketch.



Abrir

Apresenta um menu de todos os sketches já existentes.



Salvar

Salva seu Sketch.



Monitor Serial

Abre o monitor serial.

13.2 BIBLIOTECA ARDUINO

O ambiente Arduino pode ser estendido através da utilização de bibliotecas, assim como a maioria das plataformas de programação. Bibliotecas fornecem funcionalidades extras para uso em sketches. Por exemplo, para trabalhar com hardware ou manipulação de dados.

Algumas bibliotecas já vêm instaladas com a IDE Arduino, mas você também pode fazer download ou criar a sua própria.

Para usar uma biblioteca em um sketch, selecione em sua IDE Arduino:

Sketch> Importar Biblioteca.

Dentro da programação você inclui as funcionalidades de uma biblioteca já existente a partir do comando:

```
#include <LiquidCrystal.h>
```

14. PROGRAMANDO O ARDUINO

Arduino se programa em uma linguagem de alto nível semelhante a C/C++ e geralmente tem os seguintes componentes para elaborar o algoritmo:

- Estruturas (veremos a seguir)
- Variáveis (veremos a seguir)
- Operadores booleanos, de comparação e aritméticos (no decorrer do curso)
- Estrutura de controle (no decorrer do curso)
- Funções digitais e analógicas (no decorrer do curso)

Para mais detalhes visite a Referência da linguagem de programação Arduino (<http://multilogica-shop.com/Referencia>), em português.

Veja a referência estendida (<http://arduino.cc/en/Reference/HomePage?from=Reference.Extended>) para características mais avançadas da linguagem Arduino e a página das bibliotecas (<http://arduino.cc/en/Reference/Libraries>) para interação com tipos específicos de hardware, no site oficial do Arduino.

14.1 ESTRUTURAS

São duas funções principais que deve ter todo programa em Arduino.

A função `setup()` é chamada quando um programa começa a rodar. Use esta função para inicializar as suas variáveis, os modos dos pinos, declarar o uso de bibliotecas, etc. Esta função será executada apenas uma vez após a placa Arduino ser ligada ou resetada.

```
setup(){  
  
}
```

Após criar uma função `setup()` que declara os valores iniciais, a função `loop()` faz exatamente o que seu nome sugere, entra em looping (executa sempre o mesmo bloco de código), permitindo ao seu programa fazer mudanças e responder. Use esta função para controlar ativamente a placa Arduino.

```
loop(){  
  
}
```

14.2 VARIÁVEIS

Variáveis são expressões que você pode usar em programas para armazenar valores como a leitura de um sensor em um pino analógico. Aqui destacamos algumas:

- Variáveis Booleanas

Variáveis booleanas, assim chamadas em homenagem a George Boole, podem ter apenas dois valores: verdadeiro (true) e falso (false).

```
boolean running = false;
```

- Int

Inteiro é o principal tipo de dado para armazenamento numérico capaz de guardar números de 2 bytes. Isto abrange a faixa de -32.768 a 32.767 (valor mínimo de -2^{15} e valor máximo de $(2^{15}) - 1$).

```
int ledPin = 13;
```

- Char

Um tipo de dado que ocupa 1 byte de memória e armazena o valor de um caractere ASCII. Caracteres literais são escritos entre aspas.

```
char myChar = 'A';
```

15. VAMOS PRATICAR?

A ideia agora é iniciarmos o manuseio do arduino com a Linguagem de Programação, para isso vamos usar exemplos de aplicações envolvendo o hardware e o software onde teremos a entrada, o processamento e a saída de dados.

15.1 HELLO WORD

Este exemplo mostra a experiência mais simples que você pode fazer com um Arduino para verificar uma saída física: **piscar um LED**.

Quando você está aprendendo a programar, na maioria das linguagens de programação, o primeiro código que você escreve diz "Hello World" na tela do computador. Como a placa Arduino não tem uma tela substituiremos esta função fazendo piscar um LED.

15.1.1 O QUE VOU APRENDER?

- Ativar uma saída digital
- Acender um LED em ON/OFF
- Temporizar um sinal de saída
- Sintaxe de um programa Arduino

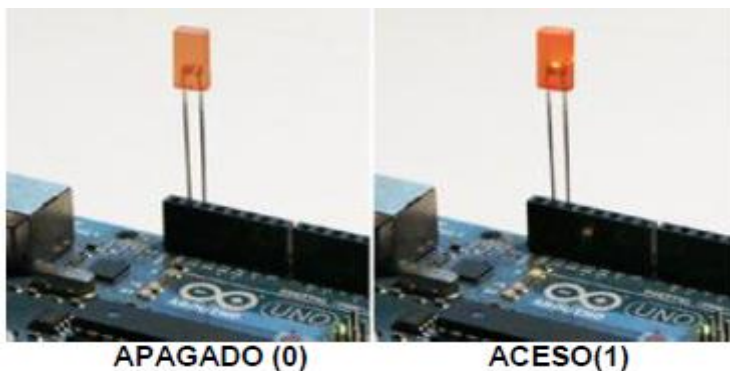
15.1.2 CONHECIMENTOS PRÉVIOS**15.1.2.1 SINAL DIGITAL**

As entradas e saídas de um sistema eletrônico serão consideradas como sinais variáveis. Em eletrônica se trabalha com variáveis que são tomadas na forma de tensão ou corrente, que podem simplesmente ser chamados de sinais.

Os sinais podem ser de dois tipos: digital ou analógico.

Em nosso caso como o que nos interessa no momento é o sinal digital, ele se caracteriza por ter dois estados diferentes e portanto também podem ser chamadas de binárias (em lógica seria valores Verdadeiro (V) e Falso (F), ou poderiam ser 1 ou 0 respectivamente).

Um exemplo de um sinal digital é o interruptor da campainha da sua casa, porque ele tem somente dois estados, pulsado e sem pulsar.

**15.1.2.2 FUNÇÃO pinMode()**

Configura o pino especificado para que se comporte ou como uma entrada (input) ou uma saída (output).

Sintaxe:

```
pinMode(pin, mode)
```

```
pinMode(9, OUTPUT); // determina o pino digital 9 como uma saída.
```

15.1.2.3 FUNÇÃO digitalWrite()

Escreve um valor HIGH (ligar) ou um LOW (desligar) em um pino digital.

Sintaxe:

```
digitalWrite(pin, valor)
```

15.1.2.3 FUNÇÃO delay()

É um temporizador, onde o valor passado(em milisegundos) será o tempo em que o Arduino não irá executar atividades até que este tempo passe.

Sintaxe:

delay(valor em milisegundos)

delay(1000); // neste caso o Aduino ficará 1 segundo sem executar atividades.

15.1.2.4 POLARIDADE DE UM LED

O LED (Light Emitting Diode) é um diodo que emite luz quando energizado. Os LED's apresentam muitas vantagens sobre as fontes de luz incandescentes como um consumo menor de energia, maior tempo de vida, menor tamanho, grande durabilidade e confiabilidade. O LED tem uma polaridade, uma ordem de conexão. Ao conectá-lo invertido não funcionará corretamente. Revise os desenhos para verificar a correspondência do negativo e do positivo.

São especialmente utilizados em produtos de microeletrônica como sinalizador de avisos. Também é muito utilizado em painéis, cortinas e pistas de led. Podem ser encontrados em tamanho maior, como em alguns modelos de semáforos ou displays.

15.1.2.5 MATERIAL NECESSÁRIO

1 Arduino Uno



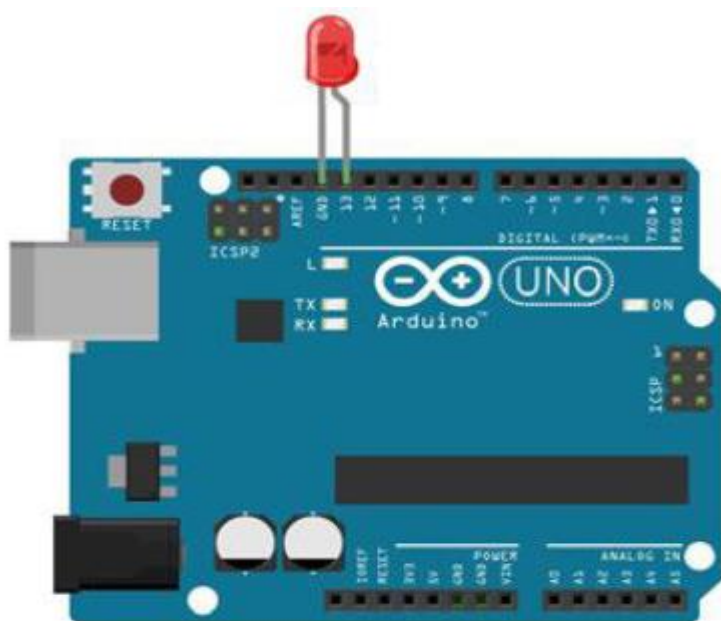
1 LED



1 Cabo USB AB



15.1.2.6 DIAGRAMA



15.1.2.7 CÓDIGO FONTE

No programa a seguir, o primeiro comando é o de inicializar o pino 13 como saída através da linha pinMode(13, OUTPUT);

No loop principal do código, você liga o LED com esta linha de comando:

`digitalWrite(13, HIGH);`

Este comando direciona 5 volts ao pino 13 e o acende. Você desliga o LED com o seguinte comando:

`digitalWrite(13, LOW);`

Este comando retira os 5 volts do pino 13, voltando para 0 e desligando o LED. Entre desligar e ligar você precisa de tempo suficiente para que uma pessoa veja a diferença, então o comando `delay()` informa o Arduino não fazer nada durante 1000 milissegundos, ou um segundo. Quando você usa o comando `delay()`, nada mais acontece neste período de tempo.



```
/*
Piscar
Acende um LED por um segundo, e depois apaga pelo mesmo tempo, repetidamente.
*/
// Estabeleça um nome para o pino 13:
int led = 13;
// Se executa cada vez que o Arduino inicia:
void setup() {
  // Inicializa o pino digital como saída.
  pinMode(led, OUTPUT);
}
// A função loop() continua executando enquanto o Arduino estiver alimentado,
// ou até que o botão reset seja acionado.
void loop() {
  digitalWrite(led, HIGH); // Acende o LED
  delay(1000); // Aguarda um segundo (1s = 1000ms)
  digitalWrite(led, LOW); // Apaga o LED
  delay(1000); // Aguarda um segundo (1s = 1000ms)
}
```

15.2 SOM NO ARDUINO

Agora nós iremos ver como emitir sons com o arduino.

15.2.1 O QUE VOU APRENDER

- O que é um Buzzer;
- Função `tone()`.
- O que é uma Protoboard

15.2.2 O QUE É UM BUZZER

O buzzer é um transdutor ou espécie de alto-falante que já possui um oscilador interno para produzir sons (semelhantes ao de uma sirene). Os transdutores são dispositivos de alta impedância, fabricados com um material à base de uma cerâmica, denominada titanato de bário. Eles são muito sensíveis podendo ser usados como fones ou pequenos alto-falantes em sistemas de aviso ou alarmes.

BUZZER



15.2.3 FUNÇÃO tone()

A função `tone()` possui 2 sintaxes: `tone(pino, frequência)` e `tone(pino, frequência, duração)`, onde `pino` referencia qual é o pino que irá gerar a frequência (ligado ao positivo do buzzer), a frequência é definida em hertz e a duração (opcional) é em milissegundos. Caso opte pela sintaxe sem duração é necessário usar a função `noTone(pino)` para parar a frequência enviada pelo pino definido

Exemplo:

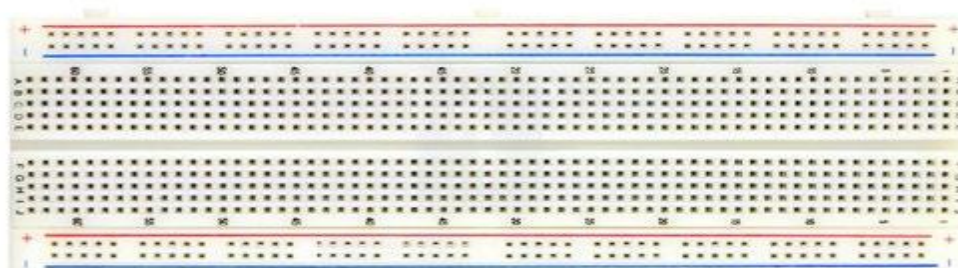
`tone(pino, frequência, duração)`

onde a `frequencia` do tom é setada em hertz, e a `duração`, em milissegundos.

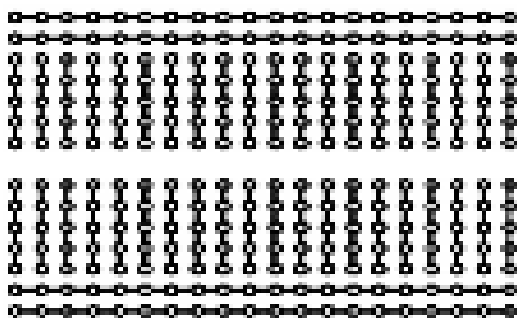
15.2.4 O QUE É UMA PROTOBOARD

É uma placa reutilizável usada para construir protótipos de circuitos eletrônicos sem solda.

Uma protoboard é feita por blocos de plástico perfurados e várias lâminas finas de uma liga metálica de cobre, estanho e fósforo.



Protoboard



Conexões abertas

15.2.4 MATERIAL NECESSÁRIO

1 Arduino Uno



1 Buzzer



1 Cabo USB AB

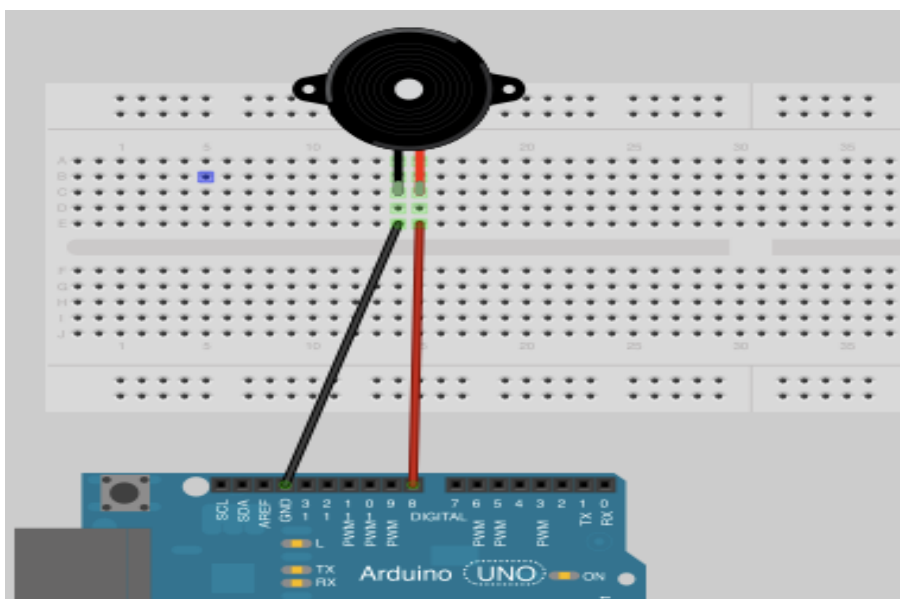


Jumpers



1 Protoboard

15.2.5 DIAGRAMA



15.2.6 CÓDIGO FONTE

```
Buzzer | Arduino 1.0.5
File Edit Sketch Tools Help
Buzzer $
//-----
//Programa: Som no arduino
//O Objetivo é mostrar a utilização de som com o Arduino
//-----

int buz = 8; //define variavel buz como pino 8

void setup(){
  pinMode(buz,OUTPUT); //define buz como saída digital
}

void loop(){
  //play tone
  tone(buz,2999,800); // esta linha de código é usada especialmente para o buzzer
                      //o seu funcionamento é : o pino(buz = 8),
                      //a frequência e o tempo de duração
                      // em milissegundos

  delay(1000); //delay (ms) neste caso será de 1 segundo
}
```

15.3 Botão

O botão é um componente que conecta dois pontos do circuito quando está pressionado. Neste exemplo quando o botão está pressionado o LED se acende.

15.3.1 O que vou aprender?

- Cabear um circuito
- Condicional if/else
- Estado de um botão
- Ler uma entrada digital e escrever uma saída digital

15.3.2 Conhecimentos prévios

15.3.2.1 Função digitalRead()

Lê o valor de um pino digital especificado, HIGH ou LOW.

Sintaxe:

```
digitalRead(pin)
buttonState = digitalRead(9); // Leitura do estado de um botão no pino 9.
```

15.3.2.3 Condicional If else (Se e Senão)

15.3.2.3.1 Condicional If else

If, que é usado juntamente com um operador de comparação, verifica quando uma condição é satisfeita, como por exemplo um input acima de um determinado valor. O formato para uma verificação if é:

```
if (algumaVariavel > 50) {
    // faça alguma coisa
} else{
    // faça outra coisa
}
```

O programa checa se algumaVariavel (colocar acentos em nomes de variáveis não é uma boa ideia) é maior que 50. Se for, o programa realiza uma ação específica. Colocado de outra maneira, se a sentença que está dentro dos parêntesis é verdadeira o código que está dentro das chaves roda; caso contrário o programa salta este bloco de código, e vai para o **else**, que é a negação do IF, realizando outra tarefa.

A sentença que está sendo verificada necessita o uso de pelo menos um dos operadores de comparação:

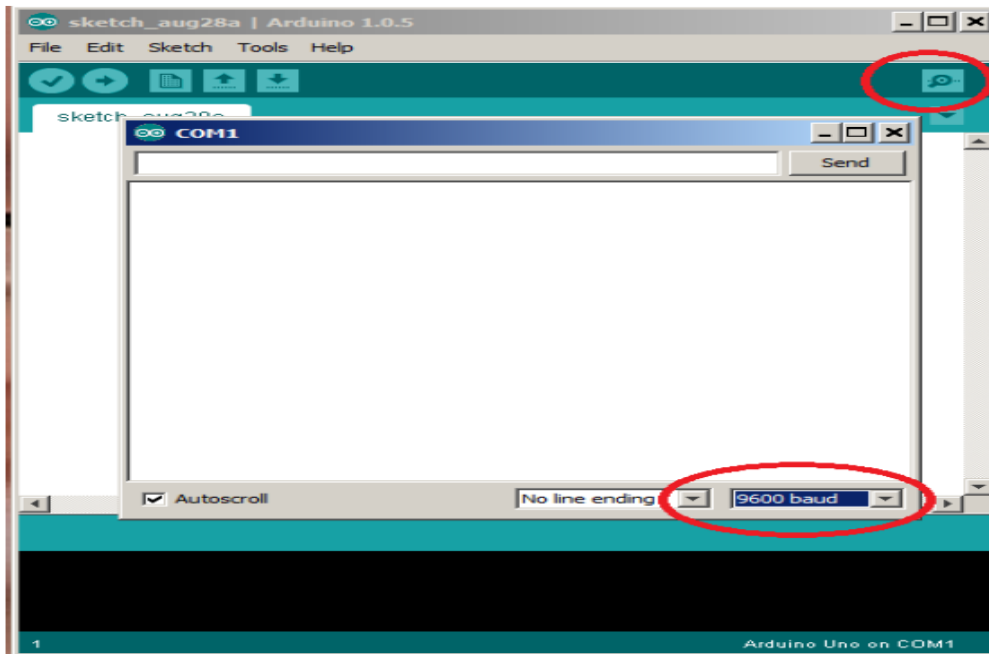
```
x == y (x é igual a y)
x != y (x é não igual a y)
x < y (x é menor que y)
x > y (x é maior que y)
x <= y (x é menor ou igual a y)
x >= y (x é maior ou igual a y)
```

15.3.3 Serial.print

Exibe dados seriais sendo enviados da placa Arduino para o computador. Para enviar dados para a placa, digite o texto e clique no botão "enviar" ou pressione enter.

A comunicação entre a placa Arduino e seu computador pode acontecer em várias velocidades padrão pré-definidas. Para que isso ocorra é importante que seja definida a mesma velocidade tanto na Sketch quanto no Monitor Serial.

Na Sketch esta escolha é feita através da função Serial.begin. E no Monitor Serial através do menu drop down do canto inferior direito.



15.3.4 Material necessário

1 Arduino Uno

1 Botão

1 LED

1 Resistor 10kΩ

1 Cabo USB AB

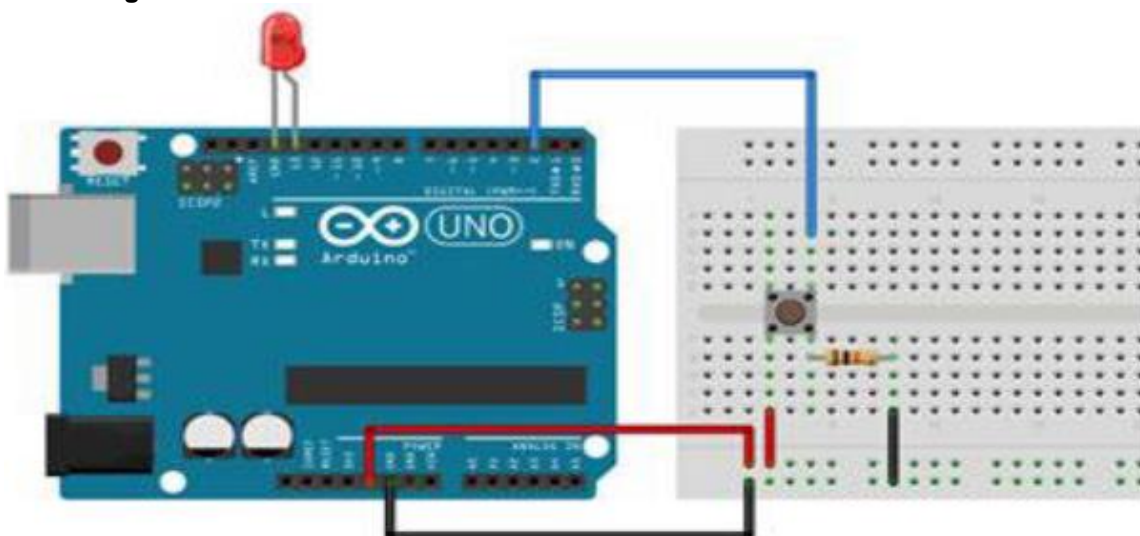


Jumpers

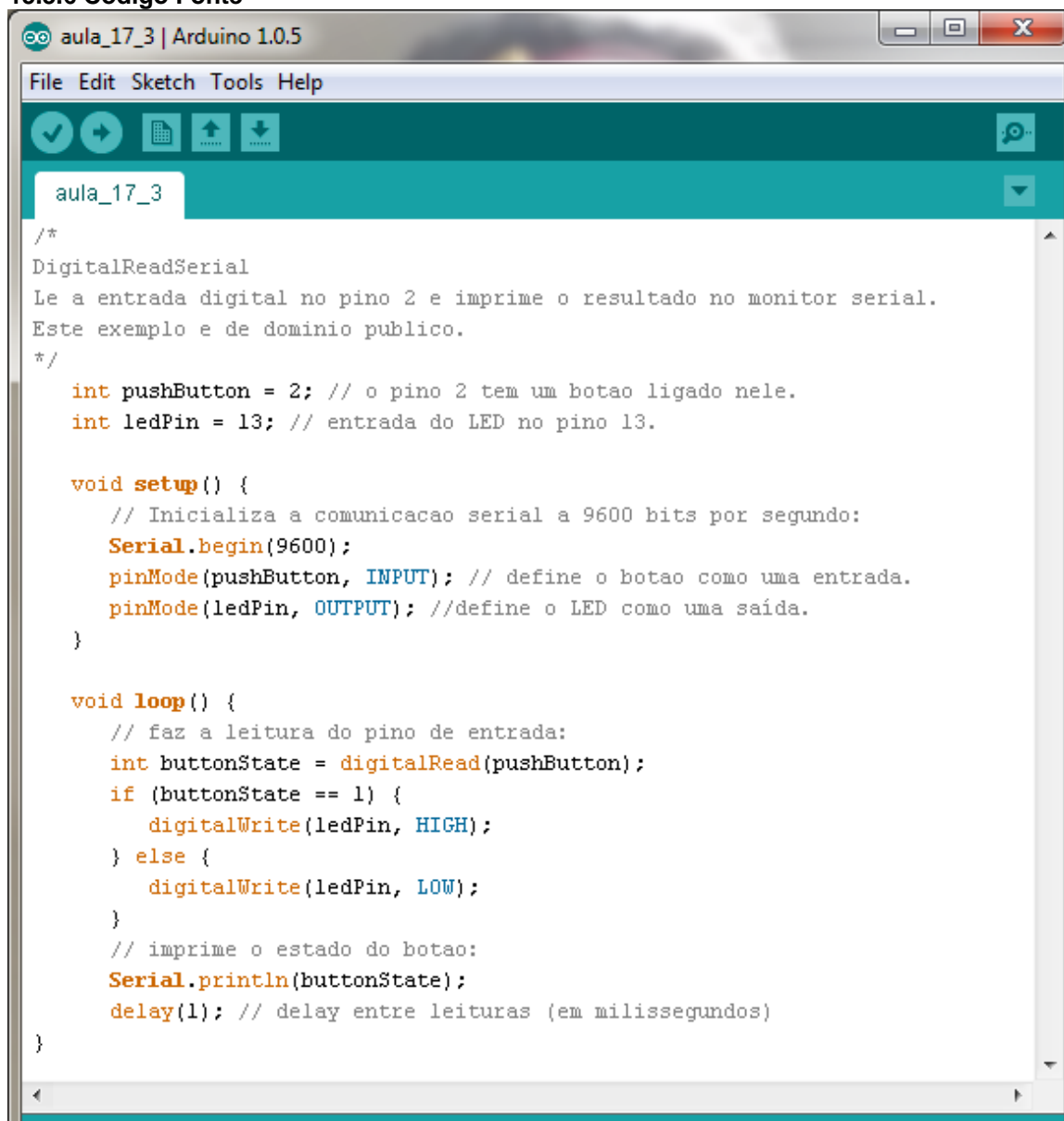


1 Protoboard

15.3.5 Diagrama



15.3.6 Código Fonte



```
/*
DigitalReadSerial
Le a entrada digital no pino 2 e imprime o resultado no monitor serial.
Este exemplo e de dominio publico.
*/

int pushButton = 2; // o pino 2 tem um botao ligado nele.
int ledPin = 13; // entrada do LED no pino 13.

void setup() {
  // Inicializa a comunicacao serial a 9600 bits por segundo:
  Serial.begin(9600);
  pinMode(pushButton, INPUT); // define o botao como uma entrada.
  pinMode(ledPin, OUTPUT); //define o LED como uma saída.
}

void loop() {
  // faz a leitura do pino de entrada:
  int buttonState = digitalRead(pushButton);
  if (buttonState == 1) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
  // imprime o estado do botao:
  Serial.println(buttonState);
  delay(1); // delay entre leituras (em milissegundos)
}
```

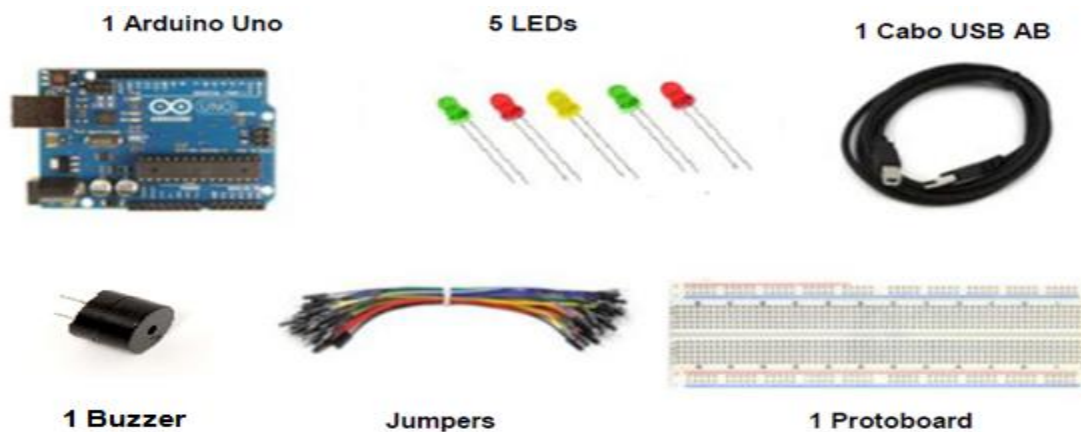
15.3.7 Exercício:

De acordo com os exercícios anteriores, monte um esquema que contenha um buzzer e que o som seja emitido somente se o botão for pressionado. Realizem a atividade para corrigirmos em aula, me chamem ao terminar.

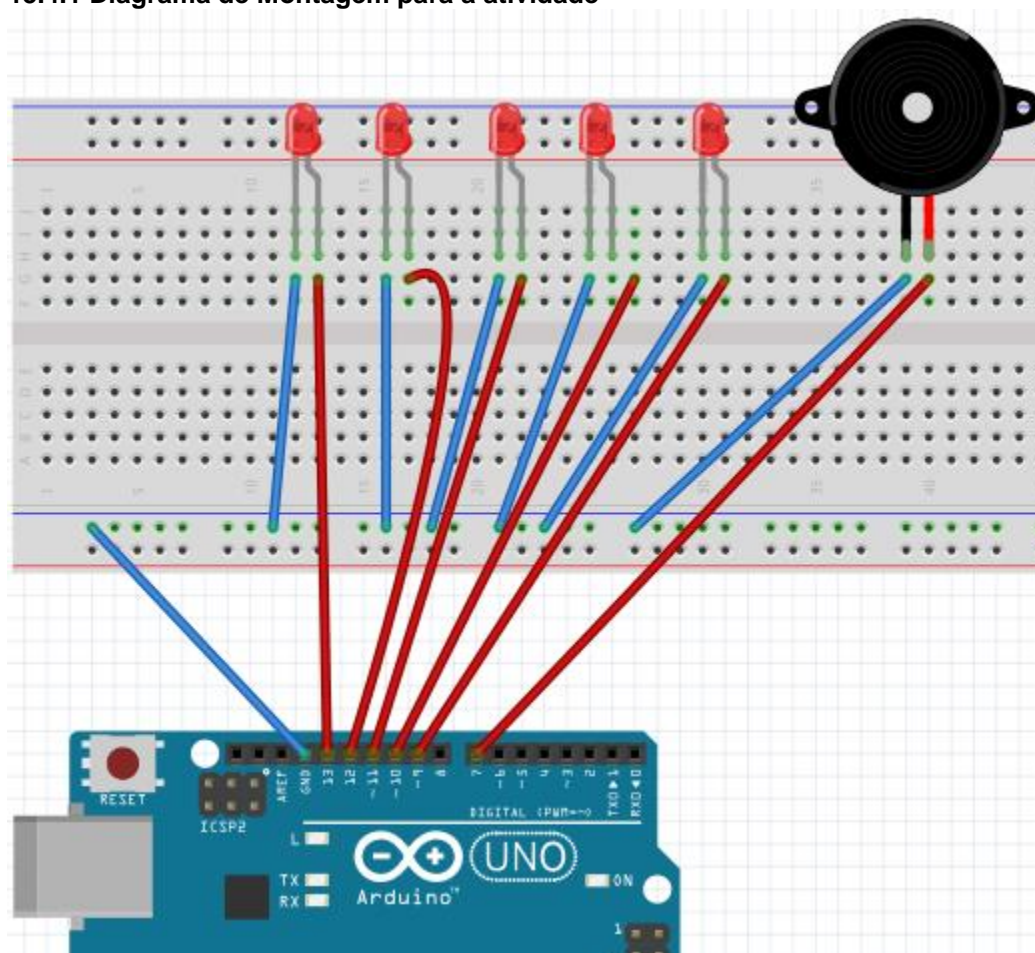
15.4 Atividade prática

Em grupo de até 5 componentes, elaborem um programa em Arduino que atenda o descrito a seguir:

15.4.1 Material necessário para atividade



15.4.1 Diagrama de Montagem para a atividade



15.4.1 Descrição da atividade

De acordo o material fornecido monte na Protoboard um esquema onde os 5 LEDs e o Buzzer se comuniquem com o Arduino.

Após isso, crie um programa em Arduino que siga exatamente esses passos que são informados abaixo:

- Acenda todos os LEDs de uma só vez;
- Emita um som no Arduino que dure 1 segundo;
- Apague o 5º LED aceso;
- Aguarde dois segundos;
- Apague o 4º LED aceso;
- Aguarde dois segundos;
- Apague o 3º LED aceso;
- Aguarde dois segundos;
- Apague o 2º LED aceso;
- Aguarde dois segundos;
- Apague o 1º LED aceso;
- Aguarde dois segundos;
- Pisque duas vezes todos os LEDs;
- Aguarde 2 segundos;
- Emita som no Arduino em três (3) frequências diferentes com duração de 2 segundos

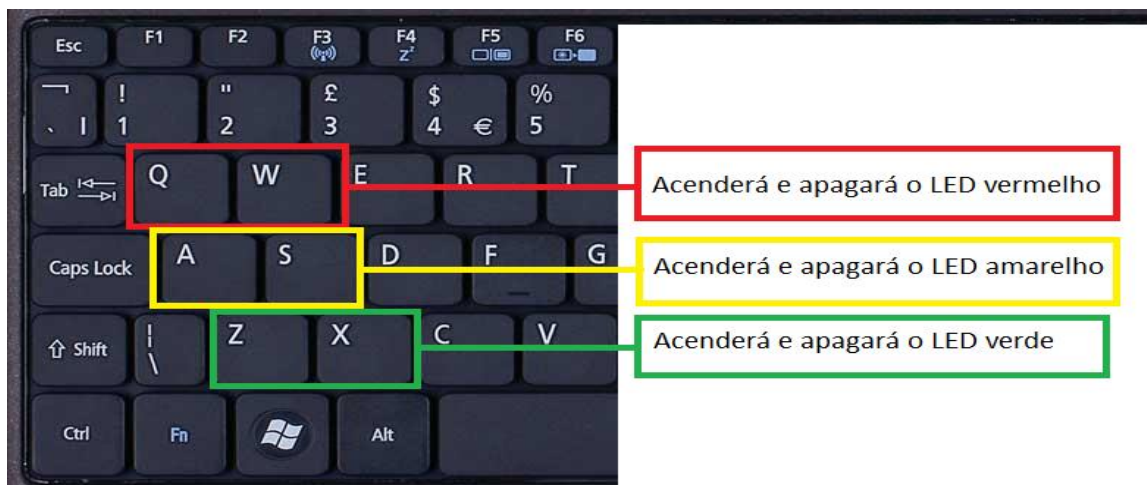
cada;

- Acenda o 1º LED;
- Aguarde 1 segundo;
- Emita um som de 1 segundo de duração
- Acenda o 2º LED;
- Aguarde 1 segundo;
- Emita um som de 1 segundo de duração;
- Acenda o 3º LED;
- Aguarde 1 segundo;
- Emita um som de 1 segundo de duração;
- Acenda o 4º LED;
- Aguarde 1 segundo;
- Emita um som de 1 segundo de duração;
- Acenda o 5º LED;
- Aguarde 1 segundo;
- Emita um som de 1 segundo de duração;
- Apague todos os LEDs.

Após a realização da montagem do Diagrama na Protoboard e a Codificação correta do sistema, chamem o professor para averiguação do código.

15.5 Leitura Serial de uma Entrada Digital

Este exemplo mostra como monitorar o estado de LEDs através da comunicação serial entre seu Arduino e o computador usando um teclado e se comunicando através da USB.



- Q : Acenderá o LED vermelho;
- W : Apagará o LED vermelho;

- A : Acenderá o LED amarelo;
- S : Apagará o LED amarelo;
- Z : Acenderá o LED verde;
- X : Apagará o LED verde;

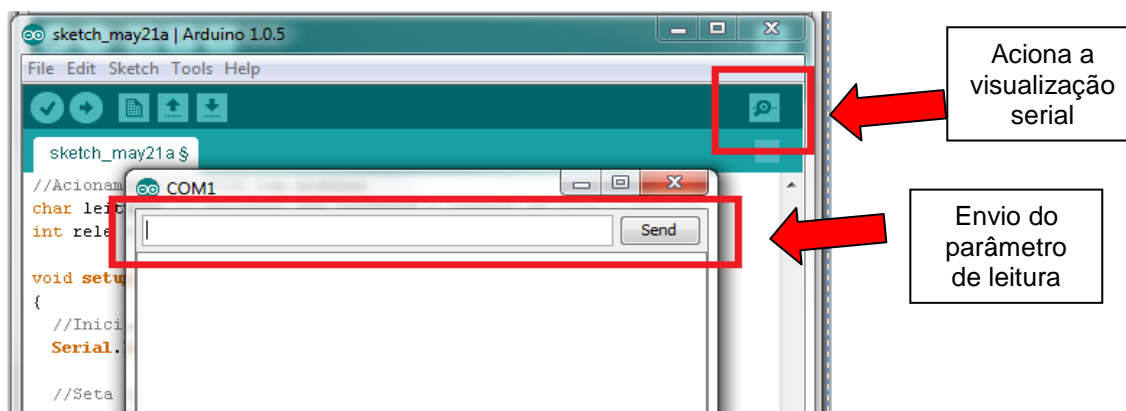
15.5.1 O que vou aprender?

- Função `serial.read()`;
- Operadores booleanos;
- Variável do tipo `char`;
- Ver dados pelo computador;
- Monitor Serial;
- Ler uma entrada digital

15.5.2 Conhecimentos prévios

15.5.2.1 Função `serial.read()`

Lê o byte mais recente apontado no buffer de entrada da serial. Neste exemplo de aula, a lâmpada acenderá ou apagará de acordo com um código enviado através do computador, sem nenhum contato físico com um botão ou controle remoto ligado diretamente ao Arduino.



15.5.2.2 Operadores booleanos

Estes operadores podem ser usados dentro da condição em uma sentença `if`.

- && (“e” lógico)

Verdadeiro apenas se os dois operandos forem verdadeiros, ou seja, a primeira condição e a segunda forem verdadeiras. Exemplo:

```
if (digitalRead(2) == 1 && digitalRead(3) == 1) { // ler dois interruptores
    // ...
}
```

é verdadeiro apenas se os dois interruptores estiverem fechados.

- || (“ou” lógico)

Verdadeiro se algum dos operandos for verdadeiro, ou seja, se a primeira ou a segunda condição for verdadeira. Exemplo:

```
if (x > 0 || y > 0) {
    // ...
}
```

é verdadeiro apenas se x ou y forem maiores que 0.

- ! (negação)

Verdadeiro apenas se o operando for falso. Exemplo:

```
if (!x) {
    // ...
}
```


é verdadeiro apenas se x for falso (ou seja, se x for igual a 0).

15.5.2.3 Variável do tipo char

Tipo de dados que recebe 1 byteletras, números e caracteres especiais (@#%&).

15.5.3 Material necessário

1 Arduino Uno

3 LEDs

3 Resistores 10kΩ

1 Cabo USB AB

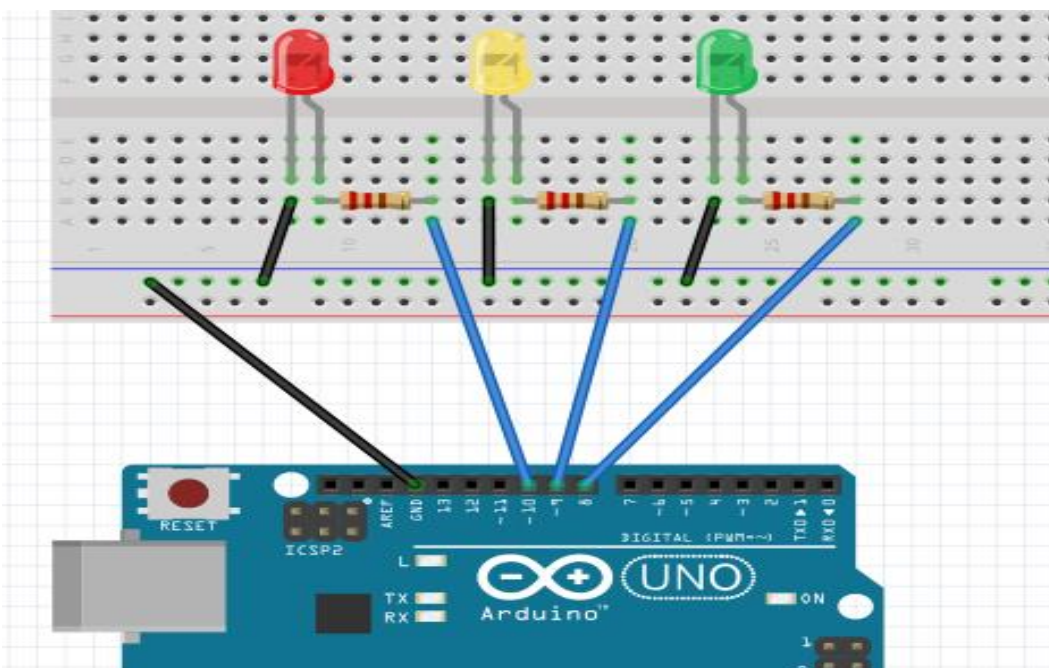


Jumpers



1 Protoboard

15.5.3 Diagrama



15.5.4 Código Fonte



```
teclado | Arduino 1.0.5
Arquivo Editar Sketch Ferramentas Ajuda

teclado $
/*-----
Controle de LEDs através do teclado
*/
int ledVermelho = 10; //Variavel do LED Vermelho
int ledAmarelo = 9; //Variavel do LED Amarelo
int ledVerde = 8; //Variavel do LED Verde
char leitura; //Variavel para armazenar o que foi digitado

void setup() {
  Serial.begin(9600); //Inicializa a comunicação serial
  pinMode(ledVermelho, OUTPUT); //Setando o Arduino
  pinMode(ledAmarelo, OUTPUT); //Setando o Arduino
  pinMode(ledVerde, OUTPUT); //Setando o Arduino
}

void loop() {
  while (Serial.available() > 0) { //Verifica se há conexão com a serial
    leitura = Serial.read(); //Lê o dado vindo da serial no computador e
                           //armazena na variavel leitura
    //As duas || é a operação booleana OU
    if (leitura == 'Q' || leitura == 'q')
    {
      // nesse caso acende o LED VERMELHO
      digitalWrite(ledVermelho, HIGH);
    } else if (leitura == 'W' || leitura == 'w') {
      // nesse caso apaga o LED VERMELHO
      digitalWrite(ledVermelho, LOW);
    } else if (leitura == 'A' || leitura == 'a') {
      // nesse caso acende o LED AMARELO
      digitalWrite(ledAmarelo, HIGH);
    } else if (leitura == 'S' || leitura == 's') {
      // nesse caso apaga o LED AMARELO
      digitalWrite(ledAmarelo, LOW);
    } else if (leitura == 'Z' || leitura == 'z') {
      // nesse caso acende o LED VERDE
      digitalWrite(ledVerde, HIGH);
    } else if (leitura == 'X' || leitura == 'x') {
      // nesse caso apaga o LED VERDE
      digitalWrite(ledVerde, LOW);
    }
  }
}
```

15.5.5 Exercício para próxima aula:

Utilizando a implementação anterior, crie uma espécie de piano, utilizando o teclado uma frequência sugerida para as notas musicais são:

- DO: 262
- RE: 294
- MI: 329
- FA: 349
- SOL: 392
- LA: 440
- SI: 494

Exercício para avaliação em aula, terminem e me chamem para correção.

15.6 Leitura de um sinal analógico no computador e controlando o piscar de um LED.

Neste projeto, você deve construir um circuito que possibilite a leitura de um valor analógico (0 a 1023) fornecido por um potenciômetro, enviando-o através de uma comunicação serial no computador, e piscar o LED de acordo com este parâmetro.

A comunicação serial no computador é vista em uma tela à parte, que pode ser acessada pelo atalho Ctrl+Shift+M (Serial monitor) na IDE do Arduino.

15.6.1 O que vou aprender?

- O que é um potenciômetro;
- A função `analogRead()`

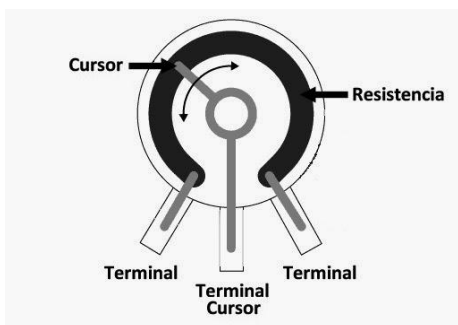
15.6.2 O que é um potenciômetro?

Um potenciômetro é um dispositivo relativamente simples. É uma resistência elétrica ajustável manualmente, que utiliza três terminais. Pode ser usado para medir posição, direção, corrente, tensão etc.



Potenciômetro de uma única volta

O potenciômetro é uma resistência que pode ser ajustada por intermédio de um curso, o qual esta em contato com uma resistência ligada a dois terminais. O curso ou contato móvel tem sua saída ligada ao terminal cursor. O contato móvel do potenciômetro se desloca do valor zero (resistência mínima) ao valor máximo (resistência máxima).



Potenciômetro aberto

15.6.3 A função `analogRead()`

A função `analogRead()`, tem por objetivo retornar a leitura analógica do dispositivo conectado na porta informada

15.6.4 Material necessário.

1 Arduino Uno

1 potenciômetro 10k

1 LED

1 Resistor

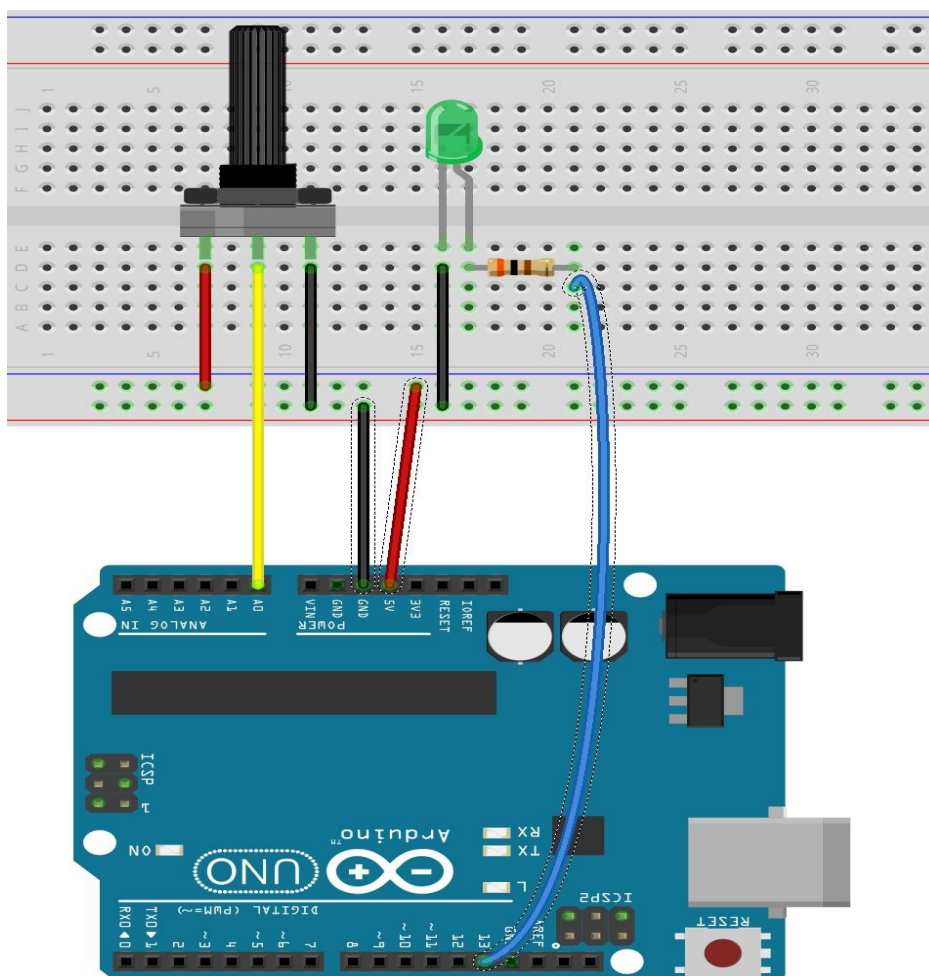


Cabo USB AB

Jumpers

1 Protoboard

15.6.5 Diagrama



15.6.6 Código fonte



```

sketch_jun11a | Arduino 1.0.5
File Edit Sketch Tools Help

sketch_jun11a$
//Código para controlar a intensidade de piscar LED
//usando potenciômetro

const int LedPin = 13;      //Pino digital onde está o LED
const int PotPin = A0;      //Pino analógico onde está o terminal do potenciômetro

void setup()
{
  Serial.begin(9600);        //Porta serial para debugar o código
  pinMode(LedPin, OUTPUT);   //Modo do pino do LED
}

void loop()
{
  //Valor de entrada do potenciômetro, logo este valor irá variar de 0 a 1023.
  int PotValue = analogRead(PotPin);

  digitalWrite(LedPin, HIGH);    //Liga o LED

  //Tempo de espera até desligar o LED. Aqui usaremos diretamente o valor do potenciômetro,
  //o que indica que o LED ficará ligado caso o valor seja 0 e piscará (aproximadamente)
  //a cada 1 segundo (ou 1023 milissegundos)
  delay(PotValue);
  digitalWrite(LedPin, LOW);     //Deliga o LED

  //Novamente o delay, agora para ligar o LED
  delay(PotValue);

  //Mostra na porta serial o valor do potenciômetro
  Serial.print("Valor: ");
  Serial.println(PotValue);
}

Done compiling.

Binary sketch size: 3.074 bytes (of a 32.256 byte maximum)

15 Arduino Uno on COM1
  
```

15.6.7 – Exercício

A este ultimo exemplo realizado, acrescentem ao projeto como se pede, usem a condicional IF para resolver este problema:

1º - Acrescentem um buzzer que emita o som por um tempo de acordo com o valor lido no potenciômetro;

2º - Acrescentem 3 LEDs, e que de acordo com a leitura do potenciômetro vão acendendo e apagando, conforme abaixo:

- Se a leitura do potenciômetro for menor que 350, acenda 1 LED;
- Se a leitura do potenciômetro for maior ou igual a 350, e menor a 600 acenda 2 LEDs;
- Se a leitura do potenciômetro for maior ou igual a 600 e menor que 900, acenda 3 LEDs;
- Se a leitura do potenciômetro for maior que 900, acenda os 4 LEDs;

A seguir forneço o diagrama para este exercício, que deverá ser entregue para avaliação, me chamem ao terminar.

