

3.2.2.8 Desempenho da equipe e Escopo de Utilização

O desempenho da equipe será definido com base no histórico de entregas e conclusões de projetos anteriores, ações que iram definir o quanto em pontos a equipe consegue atingir. Através de um sistema de pontuação cada desenvolvedor atinge em média 20 pontos por sprint de duração de duas semanas. Tal desempenho pode ser afetado por diversos fatores, os quais que devem ser trazidos para discussão na reunião de retrospectiva e planejamento. Nesta reunião, deve-se abordar a apresentação de problemas e dificuldade encontrados, buscando assim resoluções, melhorias no processo de desenvolvimento de software e consequentemente objetivando a velocidade de conclusão dos próximos sprints.

O Scrum é uma metodologia destinada a pequenas equipes com menos de dez pessoas. *Schwaber e Beedle* sugerem que a equipe seja composta de cinco a nove integrantes, se mais pessoas estiverem envolvidas no projeto, devem-se formar múltiplas Equipes Scrum.

4. Introdução à Lógica de Programação

4.1. 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.

4.2. Sequência Lógica

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

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

4.3. 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.

4.4 Algoritmo

Um algoritmo é formalmente uma sequencia finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequencia 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 sequencias lógicas.

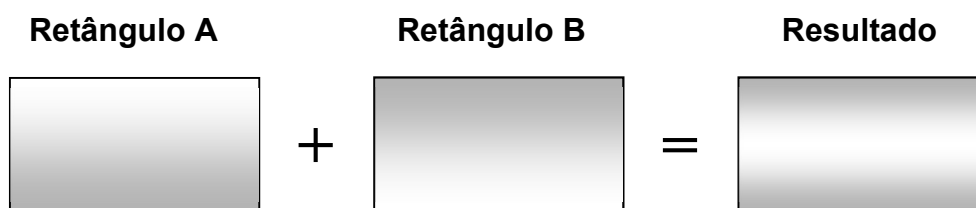
Por exemplo:

“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



4.5 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.

4.6 Exercícios (PEDIR PARA UM ALUNO PARA CADA EXERCÍCIO FAZER NA LOUSA)

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

5. Desenvolvendo Algoritmos

5.1 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.

5.2 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).

5.3 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.



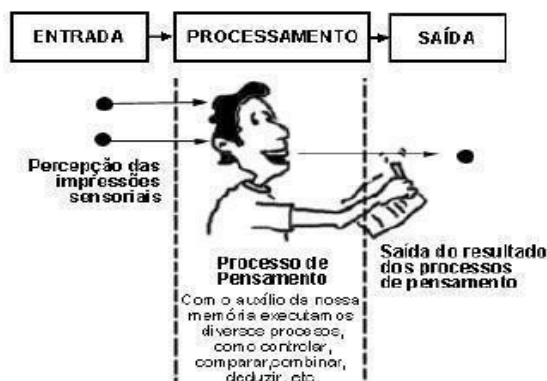
Onde temos:

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:



5.4 Exemplo de Algoritmo

Imagine o seguinte problema: Calcular a média final dos alunos do 9º ano. 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

5.5 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.

P1	P2	P3	P4	Processamen to	Média

Veja o exemplo:

Utilize a tabela abaixo:

Nota da Prova 1
Nota da Prova 2
Nota da Prova 3
Nota da Prova 4

5.6 Exercícios

- 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 $ESTOQUE_MÉDIO = (QUANTIDADE_MÍNIMA + QUANTIDADE_MÁXIMA) / 2$
- 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.

6. Sistema Computacional

Um sistema computacional pode ser visto como uma associação entre dois conceitos utilizados na terminologia de informática:

O hardware, que está associado à parte física do sistema (os circuitos e dispositivos) que suporta o processamento da informação;

O software, que corresponde ao conjunto de programas responsáveis pela execução das tarefas.

Software de sistema (ou sistema operacional) : facilita o acesso aos recursos do computador, através de comandos ou serviços especiais ativados a nível de programa. O sistema operacional administra os arquivos, controla os periféricos e executa utilitários;

- Software utilitário: são programas desenvolvidos para facilitar a realização de certas atividades como detecção de vírus, programas de comunicação em redes, compressão de arquivos, etc...)
- Software aplicativo: são programas desenvolvidos ou adquiridos pelos usuários para algum fim específico (de natureza profissional, educacional ou de lazer – jogos).

Sendo mais específicos, podemos definir que um programa de computador, ou software, é uma sequência de instruções que são enviadas para o computador (hardware). 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.

7. 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 no mercado, como exemplos podemos citar o Arduino, Java, C#, PHP, Python etc.

7.1 Níveis de linguagem de programação

Uma linguagem de programação é um vocabulário e um conjunto de regras gramaticais usadas para escrever programas de computador. Esses programas instruem o computador a realizar determinadas tarefas específicas. Cada linguagem possui um conjunto único de palavras-chaves (palavras que ela reconhece) e uma sintaxe (regras) específica para organizar as instruções dos programas. Os programas de computador podem ser escritos em várias linguagens de programação, algumas diretamente compreensíveis pelo computador e outras que exigem passos de tradução intermediária. As linguagens de programação podem ser divididas em três tipos, com relação à sua similaridade com a linguagem humana:

- Linguagem de máquina;
- Linguagem simbólica;
- Linguagem de alto nível.

7.1.1 Linguagem de máquina (Machine Language)

É a linguagem de mais baixo nível de entendimento pelo ser humano e a única, na verdade, entendida pelo processador (UCP). É constituída inteiramente de números, o que torna praticamente impossível entendê-la diretamente. Cada UCP tem seu conjunto único de instruções que definem sua linguagem de máquina, estabelecido pelo fabricante do chip. Uma instrução típica em linguagem de máquina seria algo como: **0100 1111 1010**. Essa linguagem é também classificada como uma linguagem de primeira geração.

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

7.1.2 Linguagem Simbólica (ASSEMBLY):

É a linguagem de nível imediatamente acima da linguagem de máquina. Ela possui a mesma estrutura e conjunto de instruções que a linguagem de máquina, porém permite que o programador utilize nomes (chamados mnemônicos) e símbolos em lugar de números. A linguagem simbólica é também única para cada tipo de UCP (Unit Central Processor), de forma que um programa escrito em linguagem simbólica para uma UCP poderá não ser executado em outra UCP de uma família diferente. Nos primórdios da programação todos os programas eram escritos nessa linguagem.

Hoje a linguagem simbólica, é utilizada quando a velocidade de execução ou o tamanho do programa executável gerado são essenciais. A conversão da linguagem simbólica para a linguagem de máquina se chama montagem, e é feito por um programa chamado montador (ou assembler). Uma típica instrução em linguagem simbólica seria: ADD A,B. Essa linguagem é também classificada como linguagem de segunda geração, e, assim como a linguagem de máquina, é considerada uma linguagem de baixo nível.

C014 24 FA	BCC	INCH	RECIEVE NOT READY
C016 B6 80 05	LDA A	ACIA+1	GET CHAR
C019 84 7F	AND A	#\$7F	MASK PARITY
C01B 7E C0 79	JMP	OUTCH	ECHO & RTS

```
*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input
```

C01E 8D F0	INHEX	BSR	INCH	GET A CHAR
C020 81 30		CMP A	#'0	ZERO
C022 2B 11		BMI	HEXERR	NOT HEX
C024 81 39		CMP A	#'9	NINE
C026 2F 0A		BLE	HEXRTS	GOOD HEX
C028 81 41		CMP A	#'A	
C02A 2B 09		BMI	HEXERR	NOT HEX

Figura 1: Linguagem Assembly

7.1.3 LINGUAGEM DE ALTO NÍVEL:

São as linguagens de programação que possuem uma estrutura e palavras-chave que são mais próximas da linguagem humana. Tornando os programas mais fáceis de serem lidos e escritos. Esta é a sua principal vantagem sobre as linguagens de nível mais baixo. Os programas escritos nessas linguagens são convertidos para a linguagem de baixo nível através de um programa denominado compilador ou de um interpretador.

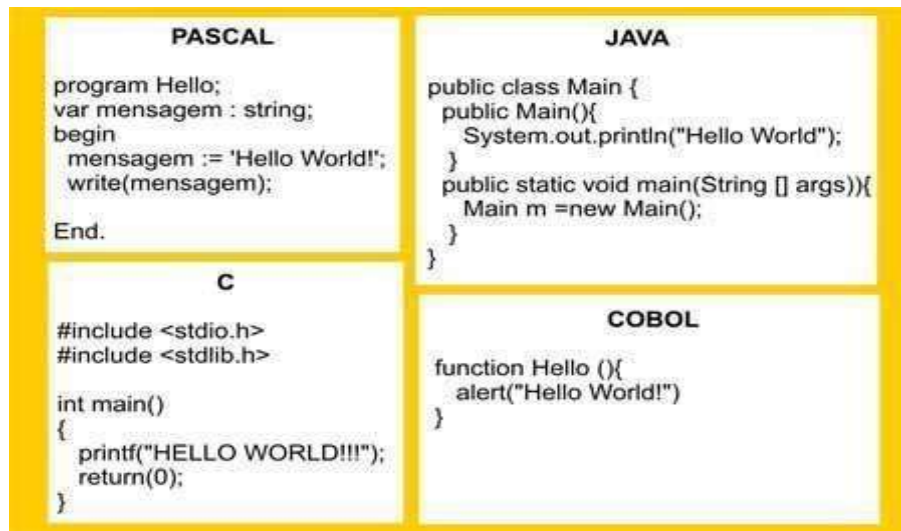


Figura 2: Exemplos de Hello World
Para relaxar ...

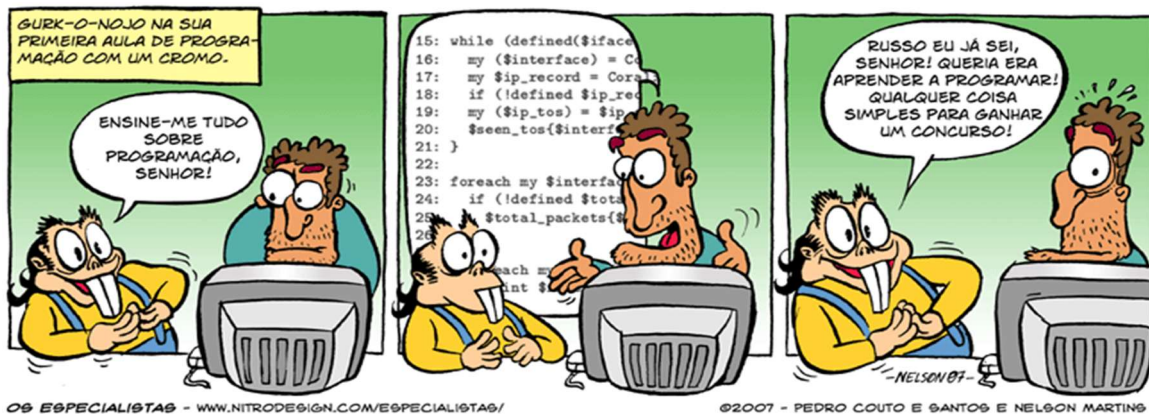


Figura 3: Para relaxar

8. Paradigmas de programação

O número de linguagens de programação existentes atualmente chega a ser da ordem de alguns milhares. Esse número impressionante reflete o esforço no sentido de projetar linguagens que facilitem sempre mais a atividade de programação, tornando-a cada vez mais produtiva. Outro objetivo importante no projeto dessas linguagens é o de favorecer a construção de programas mais eficientes, isto é, que originem programas executáveis rapidamente e que usam relativamente pouca quantidade de memória de um computador, e seguros, isto é, menos sujeitos a erros que possam ocasionar um comportamento da execução do programa diferente daquele que é esperado.

Apesar dessa grande diversidade de linguagens de programação, a maioria delas apresenta, essencialmente, o mesmo conjunto de comandos básicos, embora esses possam apresentar formas diferentes em diferentes linguagens (o termo “instrução” é em geral usado para linguagens de mais baixo nível, enquanto o termo “comando”, em princípio equivalente, é mais usado para linguagens de alto nível). Esse conjunto é constituído de:

- um comando básico — denominado comando de atribuição — usado para armazenar um valor em uma determinada posição de memória;
- comandos para leitura de dados, de dispositivos de entrada, e para escrita de dados, em dispositivos de saída;
- três formas distintas de combinação de comandos:
 - composição sequencial — execução de um comando após outro,
 - seleção (ou escolha condicional) — escolha de um comando para ser executado, de acordo com o resultado da avaliação de uma condição,
 - repetição — execução de um comando repetidas vezes, até que uma condição seja satisfeita.

Em uma linguagem com essas características, um programa consiste em uma sequência de comandos que descreve, em essência, como devem ser modificados os valores armazenados na memória do computador, de maneira que uma determinada tarefa seja realizada. Esse paradigma de programação é denominado paradigma imperativo e linguagens baseadas nesse paradigma são chamadas de linguagens imperativas.

Existem, em contraposição, outros paradigmas de programação, chamados declarativos, nos quais um programa se assemelha mais a uma descrição de o que constitui uma solução de um determinado problema, em lugar de como proceder para obter essa solução.

A linguagem C é uma linguagem que provê suporte ao paradigma imperativo de programação.

8. Usabilidade

Seu novo usuário hoje ao interagir com o seu produto consegue se localizar e entender como tudo funciona rapidamente? No geral, as tarefas são realizadas com agilidade? Você evita os erros conduzindo-o e indicando a melhor forma de utilizar esse produto com dicas visuais e gatilhos simpáticos para as tarefas?

Esses pontos e vários erros ligados a eficiência, eficácia e satisfação durante o uso de um produto são reflexos diretos de uma boa ou má usabilidade.

Usabilidade, em linhas gerais, é um termo usado para definir a facilidade com que as pessoas utilizam uma ferramenta ou interface.

A norma ISO 9241 determina como usabilidade, a capacidade que um sistema interativo oferece a seu usuário em um determinado contexto de operação, para a realização de tarefas com efetividade, eficiência e satisfação.

- EFICÁCIA – capacidade de executar a tarefa de forma correta e completa
- EFICIÊNCIA – são os recursos gastos para conseguir ter eficácia. Sejam eles tempo, dinheiro, produtividade ou memória.
- SATISFAÇÃO – se refere ao nível de conforto que o usuário sente ao utilizar a interface

Lembrando que quando falamos de user experience design e de boas experiências, a USABILIDADE aparece como fator fundamental, no segundo faixa da nossa “pirâmide das boas experiências”, trazendo a questão do “eu consigo usar esse produto?”.

Porque no fim das contas, de nada adianta eu ter uma proposta de valor incrível se meu usuário não conseguir ou tiver dificuldades de manejar o produto/interface, podendo comprometer significativamente a sua percepção de valor e permanência no produto.

E na prática? Como melhorar a usabilidade?

Para colocar em prática, existem muitas regras e diretrizes criadas por profissionais da área de usabilidade ao longo dos anos que visam facilitar a identificação dos erros e acertos mais comuns das interfaces. Uma das listas mais conhecidas é a do Jakob Nielsen, criador das famosas “Heurísticas de Nielsen”.

“(…) dez princípios gerais do design de interface do usuário. São chamados de “heurística” porque estão mais na natureza de regras do que como diretrizes de usabilidade específicos.” Jacob Nielsen

Alguns sites como o Good UI também reúnem uma série de dicas práticas para você verificar onde está cometendo gafes nas interfaces de seu produto.

Conheça as 10 regras essenciais da usabilidade

Confira na íntegra algumas regras que refletem a maior parte dos pontos que você deve se preocupar no que diz respeito a usabilidade e que irão refletir diretamente na experiência do seu usuário:

1- Diálogos Simples e Naturais:

As interfaces de usuários devem ser o mais simples possível. Deve-se apresentar exatamente a informação que o usuário precisa – nem mais nem menos – na hora e lugar exatos onde é necessária.

2- Falar a Linguagem do Usuário:

A terminologia da interface deve ser baseada na linguagem do usuário. Deve ser expressado com palavras, frases e conceitos familiares ao usuário ao invés dos termos originados do sistema.

3- Minimizar a Sobrecarga de Memória do Usuário:

O sistema deve exibir elementos de diálogo para o usuário e permitir que o mesmo faça suas escolhas, sem a necessidade de lembrar de um comando específico.

4- Consistência:

Um mesmo comando ou uma mesma ação terá sempre o mesmo efeito. A mesma operação deverá ser apresentada na mesma localização em todas as telas e deverá ser formatada da mesma maneira para facilitar o reconhecimento.

5- Feedback:

O sistema deverá informar continuamente ao usuário sobre o que ele está fazendo. O tempo de resposta influi no tipo de feedback que deve ser dado ao usuário. Dez segundos (10s) é o limite para manter a atenção do usuário focada no diálogo.

6- Saídas Claramente Marcadas:

De modo a fazer com que o usuário sinta que pode controlar o sistema, deverá ser fácil abortar uma tarefa ou desfazer uma ação.

7- Atalhos:

Os sistemas devem conter atalhos para usuário experiente executar mais rapidamente operações frequentemente utilizadas.

8- Boas mensagens de erro:

As mensagens de erro devem ter linguagem clara e sem código, devem ser precisas e ajudar o usuário a resolver o problema. Não devem intimidar ou culpar o usuário.

9- Prevenir Erros:

Melhor do que possuir boas mensagens, é evitar situações de erro. Conhecer as situações que mais provocam erro e modificar a interface para que estes erros não ocorram.

10- Ajuda e Documentação:

O melhor é que um sistema que seja tão fácil de usar que não necessite de ajuda ou documentação. No entanto, se preciso, esta ajuda deve estar facilmente acessível on-line.

9. 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.

9.1 Constantes

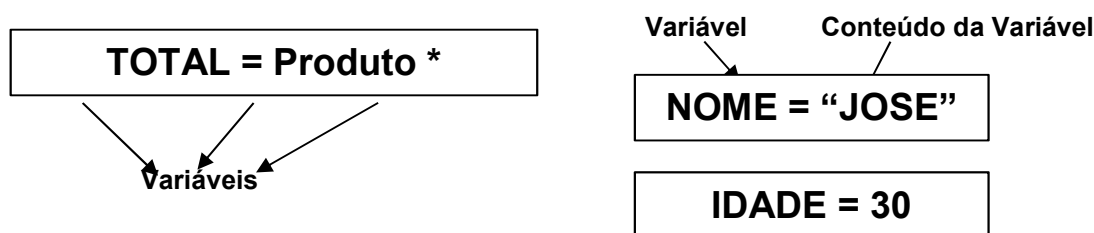
Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.

Exemplo de constantes:

$$\frac{N1 + N2 + N3}{8} \quad \text{Constante}$$

9.2 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:



9.3 Tipos de Variáveis

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

9.3.1 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.

9.3.2 Caracteres

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

9.3.3 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.

9.4 Lógicas

Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso.

10. Introdução ao Arduino

O Arduino faz parte do conceito de hardware e software livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália, em 2005, com o objetivo de criar um dispositivo que fosse utilizado em projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado.

Ele pode ser usado para desenvolver artefatos interativos stand-alone ou conectados ao computador, utilizando diversos aplicativos, tais como: *Adobe Flash*, *Processing*, *Max/MSP*, *Pure Data* ou *SuperCollider*.

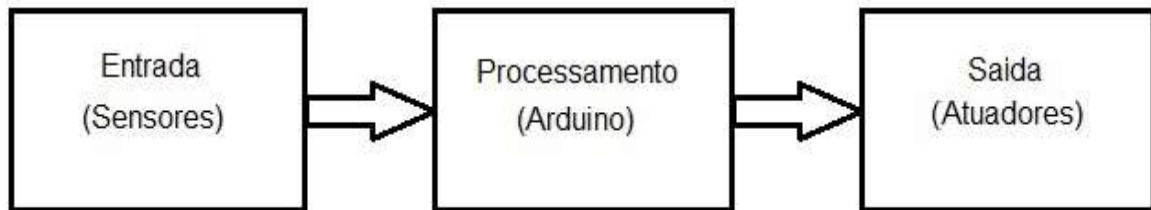
O Arduino foi projetado com a finalidade de ser de fácil entendimento, de fácil programação e de fácil aplicação, além de ser multiplataforma, podendo ser configurado em ambientes *Linux*, *Mac OS* e *Windows*. Além disso, um grande diferencial deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *open-source*, desenvolvendo e divulgando gratuitamente seus projetos.

O equipamento é uma plataforma de computação física: são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondem com ações físicas. Ele é baseado em uma placa microcontrolada, com acessos de Entrada/Saída (I/O), sobre a qual foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma sintaxe similar a das linguagens C e C++.

O Arduino utiliza o microcontrolador Atmega. Um microcontrolador (também denominado MCU) é um computador em um chip, que contém um microprocessador, memória e periféricos de entrada/saída. Ele pode ser embarcado no interior de algum outro dispositivo, que, neste caso, é o Arduino, para que possa controlar suas funções ou ações.

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. Por fim, ele pode ainda atuar no controle ou no acionamento de algum outro elemento eletroeletrônico conectado ao terminal de saída. A

Figura abaixo apresenta um diagrama de blocos de uma cadeia de processamento utilizando o Arduino.



Uma vez que o Arduino é baseado em um microcontrolador e, portanto, é programável, torna-se possível criar diversas aplicações diferentes com uma certa facilidade. Além disso, o próprio equipamento pode ser reutilizado, através de uma nova programação. Por sua vez, a sua programação é simplificada pela existência de diversas funções que controlam o dispositivo, com uma sintaxe similar a de linguagens de programação comumente utilizadas (C e C++).

11. Arduino em Windows

11.1 Placa Arduino e um cabo USB AB



11.2 - Download do software do Arduino

Faça download da última versão do software do Arduino (<http://multilogica-shop.com/Download>). Ao terminar, descompacte o arquivo e mantenha a estrutura de pastas e sub-pastas. Se quiser guarde esta pasta no drive C: do seu computador. Dentro desta pasta existe um arquivo chamado arduino.exe que é o ponto de entrada do programa do Arduino, a IDE (Integrated Development Environment).

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).

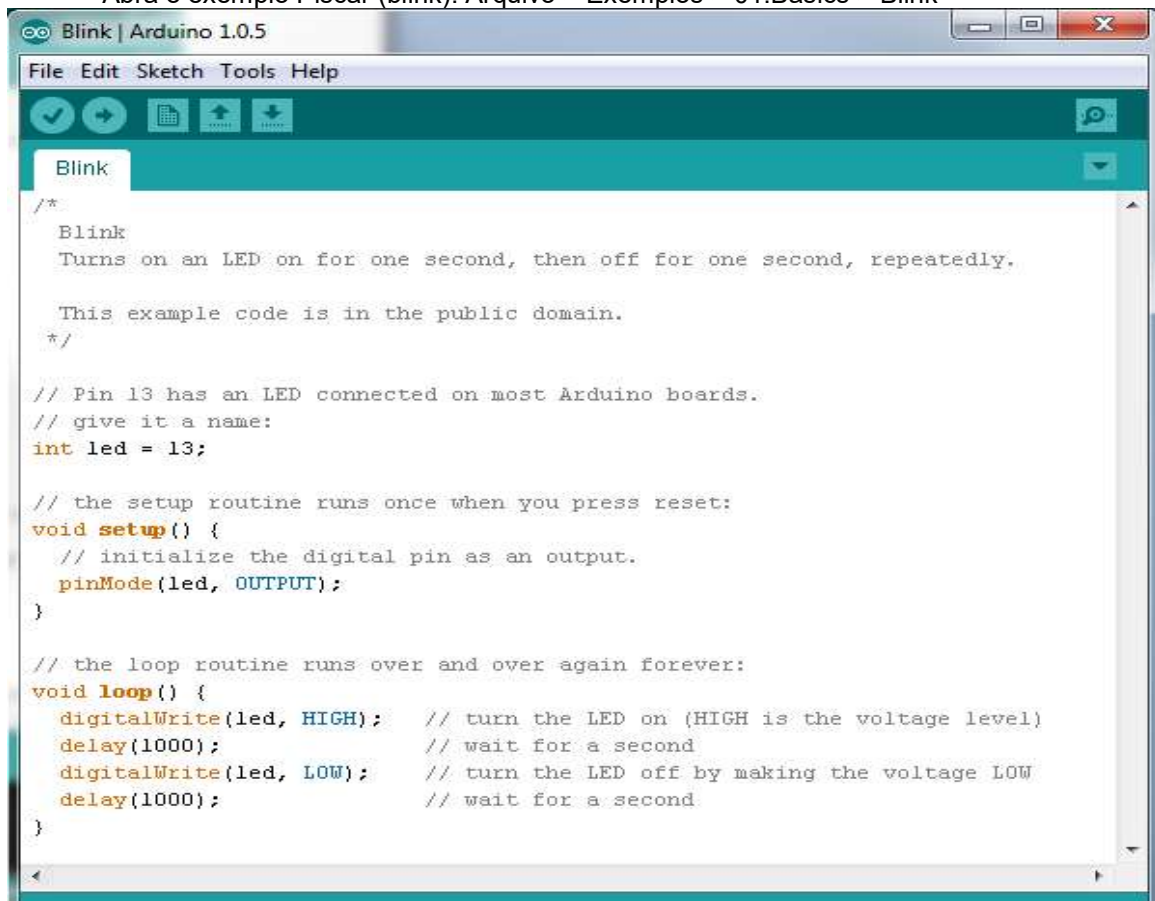
- Clique com o botão da direita em Arduino UNO (COMxx) e escolha a opção Atualizar Driver.
- Depois escolha a opção Instalar de uma lista ou local específico (Avançado), e clique em avançar.
- Finalmente navegue e escolha o driver arduino.inf localizado na pasta Drivers do software do Arduino que você 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 | 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;

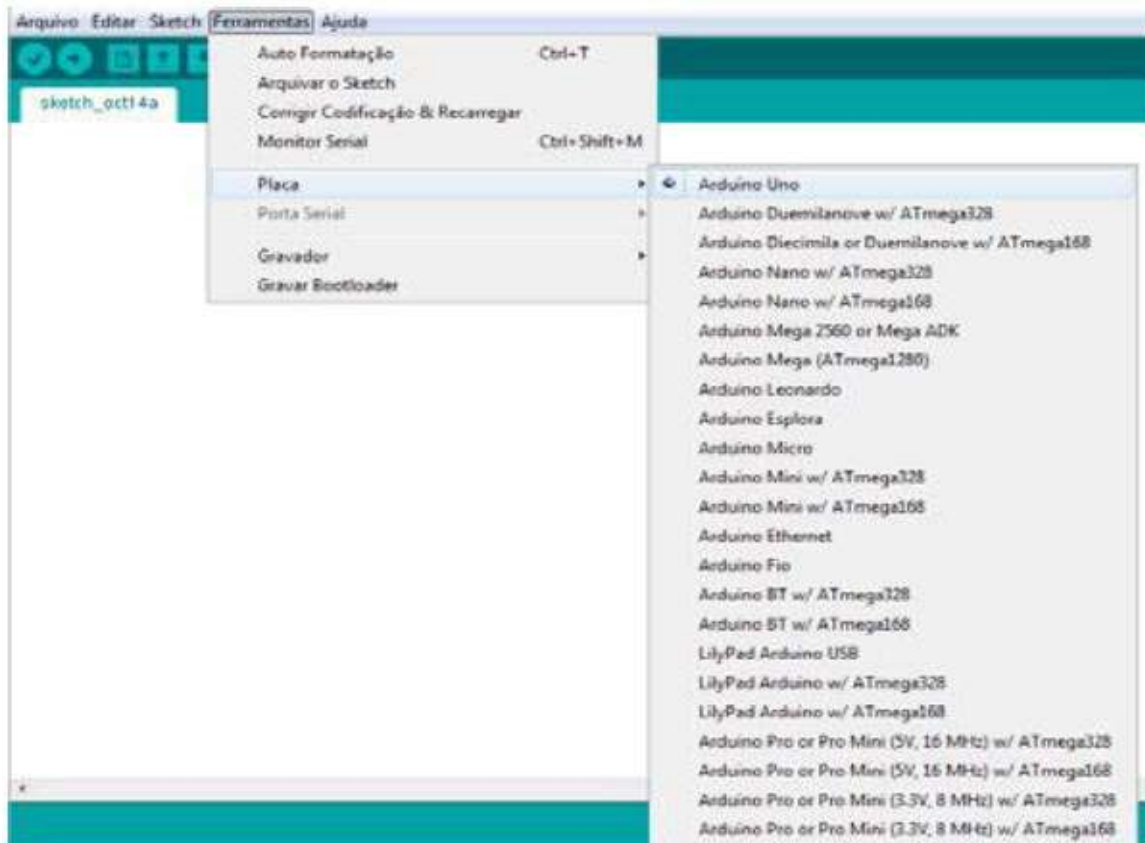
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// 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
}

```

11.7 - Selecione sua placa

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



11.8 - Selecione 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
}
  
```

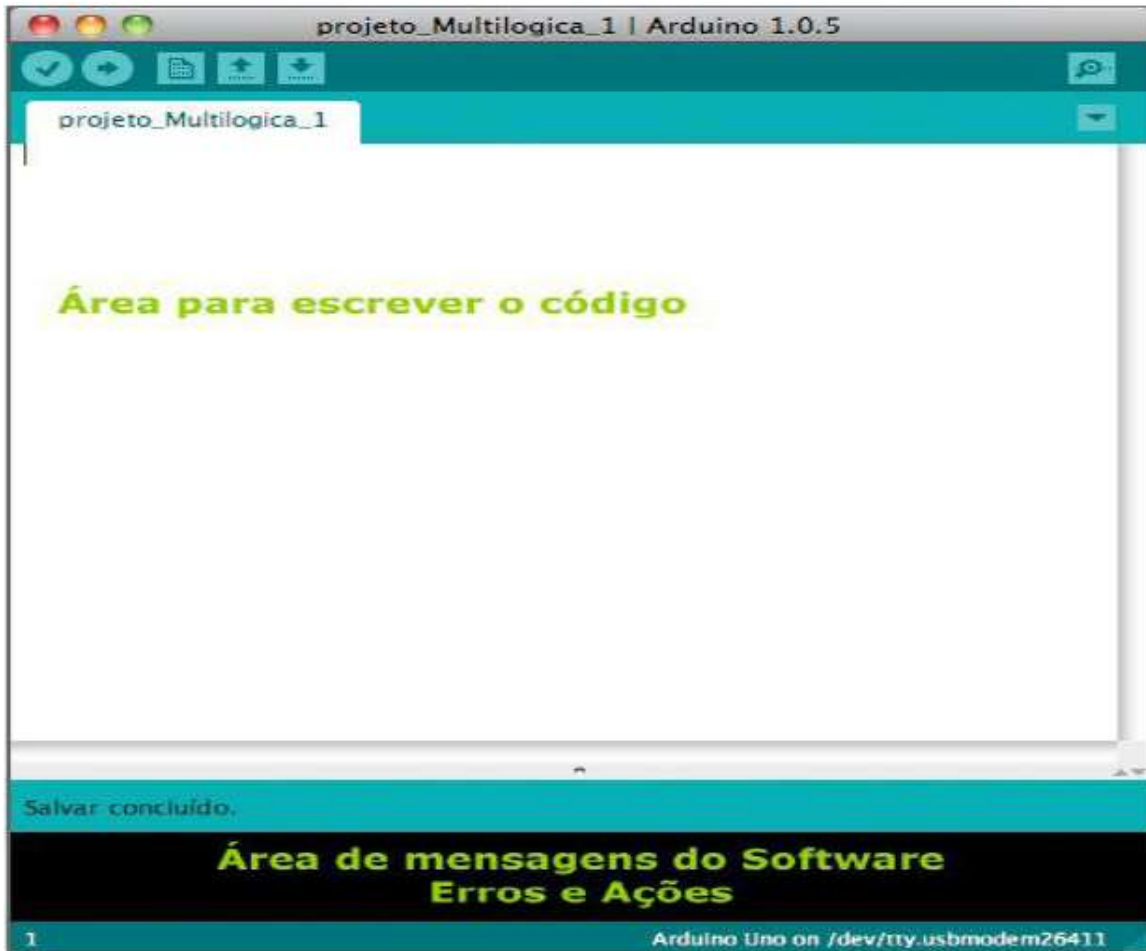
12. Software Arduino



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:



12.1 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.

12.2 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>
```

13. 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.

13.1 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 livrarias, etc. Esta função será executada apenas uma vez após a placa Arduino ser ligada ou ressetada.

```
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(){  
  
}
```

13.2 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';
```

14. 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.

14.1 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.

14.1.1 O que vou aprender?

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

14.1.2 Conhecimentos prévios

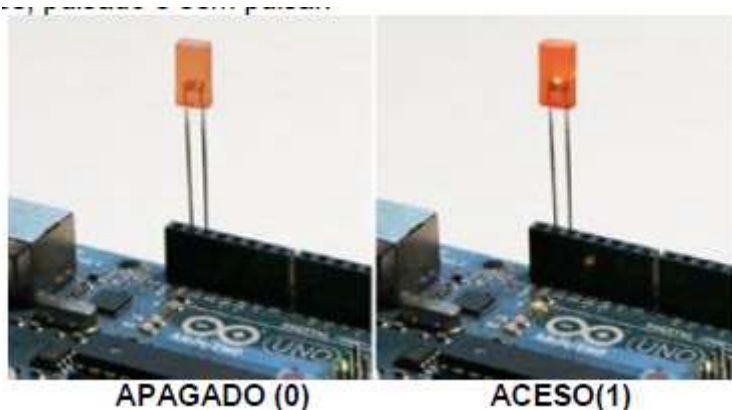
14.1.2.1 Sinal digital

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.



14.1.2.2 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.

14.1.2.3 Função `digitalWrite()`

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

Sintaxe:

`digitalWrite(pin, valor)`

`digitalWrite(9,HIGH)` // o LED na porta 9 acenderá

14.1.2.4 Função `delay()`

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

Sintaxe:

`delay(valor em milissegundos)`

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

14.1.2.5 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.

14.1.2.6 Conexão da placa Arduino com o computador

Item 11.3 desta nota de aula.

14.1.2.7 Material necessário

1 Arduino Uno



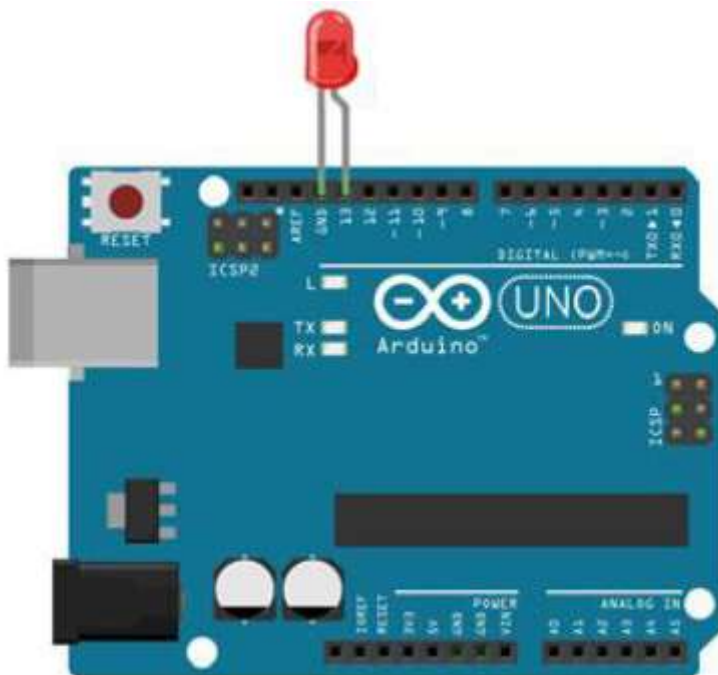
1 LED



1 Cabo USB AB



14.1.2.8 Diagrama



14.1.2.9 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)
}

```

14.1.2.10 Exercício de fixação

A partir do código fonte apresentado neste tutorial, faça as modificações necessárias para que o LED fique:

- 3 segundos aceso e 3 segundos apagado
- 200 milissegundos aceso e 500 milissegundos apagado

15. Controlando um servomotor com potenciômetro no Arduino

Este exemplo faz a leitura de um potenciômetro e move um servo motor de acordo com a leitura.

15.1 O que vou aprender?

- O que é um servomotor;
- O que é uma *protoboard*;
- O que é um potenciômetro;
- PWM;
- Função *analogRead*;
- Função *map()*;
- Biblioteca "Servo.h".

15.2 Conhecimentos prévios

15.2.1 Servomotor

Entre os atuadores temos um motor bem especial. Os servomotores, também chamados de servos, são muito utilizados quando o assunto é robótica. De forma simplificada, um servomotor é um motor na qual podemos controlar sua posição angular através de um sinal PWM.



Dessa forma, um servomotor é um atuador eletromecânico utilizado para posicionar e manter um objeto em uma determinada posição. Para isso, ele conta com um circuito que verifica o sinal de entrada e compara com a posição atual do eixo.

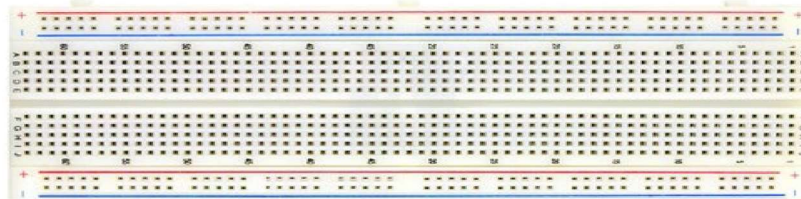
Servomotores geralmente possuem 3 pinos:

- Alimentação positiva (vermelho) – 5V;
- Terra (Preto ou Marrom) – GND;
- (Amarelo, Laranja ou Branco) – Ligado a um pino digital de entrada e saída;

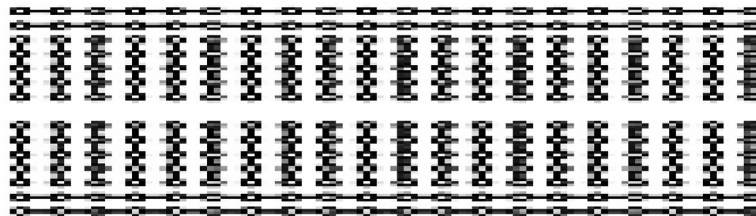
15.2.2 O QUE É UMA PROTOBOARD

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

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



Protoboard



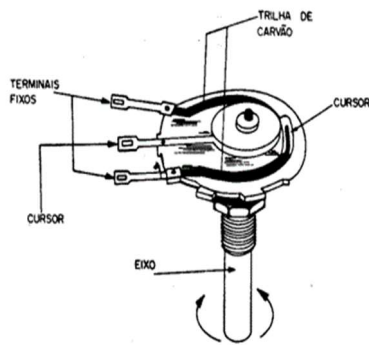
Conexões abertas

15.2.3 Potenciômetro

Um potenciômetro é um tipo especial de resistor de três terminais cuja resistência pode ser ajustada por meio mecânico, girando ou deslizando um eixo móvel, formando assim um divisor de tensão ajustável. Os potenciômetros encontram inúmeras aplicações em vários campos da tecnologia, como por exemplo em amplificadores de áudio, instrumentos musicais eletrônicos, *mixers* de áudio, eletrodomésticos, televisores, equipamentos industriais, joysticks, osciloscópios analógicos, e muitos outros.

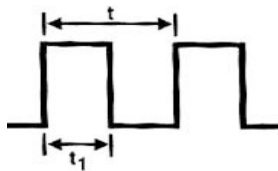


Um potenciômetro consiste em um elemento resistivo, chamado de “pista”, ou “trilha”, e de um cursor móvel, que se movimenta ao longo de um eixo, rotatório ou linear. De acordo com a posição desse cursor ao longo do eixo, a resistência obtida será diferente, dentro de certos limites característicos do componente em questão.



15.2.4 PWM

O nome PWM é uma sigla que significa *Pulse Width Modulation* (Modulação em Largura de Pulso). O PWM trata-se de uma modulação digital / em sinal digital, ou seja, trabalha com somente dois níveis de sinal: *on* (ou '1') e *off* (ou '0'). Em termos simples, o PWM controla quanto tempo um sinal digital fica em *on* em uma determinada janela de tempo fixa (chamada de período). Observe a figura abaixo, onde o tempo em *on* é representado por t_1 e o período por t .



Eletricamente, é possível afirmar que o PWM faz com que a tensão média num dado período possa ser controlada. Quanto mais tempo o sinal fica em *on* no período, maior a tensão média do sinal neste mesmo período. A esta porção de tempo em que o sinal fica em *on* no referido período, é dado o nome de *Duty Cycle*. O *Duty Cycle* é comumente tratado na forma porcentagem, como uma fração do período total do sinal.

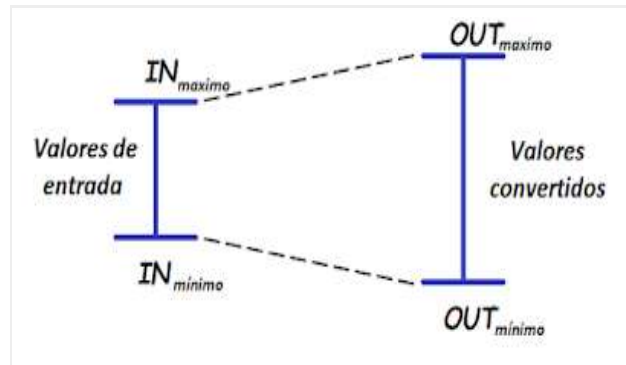
No Arduino, podemos encontrar algumas portas PWM, a quantidade delas depende do modelo do Arduino. Veja abaixo as portas PWM de um Arduino Uno:



Veja que as saídas PWM são as marcadas com um "~".

15.2.5 Função map()

Como o próprio nome diz, a função `map()` do Arduino faz o mapeamento de um intervalo numérico em outro intervalo numérico desejado. Conforme figura 1, notamos que um intervalo numérico que vai de um valor mínimo até um valor máximo, será mapeado para um novo valor de saída definido pelo programador.



A sintaxe da função `map()` é dado por:

```
map(x, IN_min, IN_max, OUT_min, OUT_max);
```

onde temos:

- X - Variável com valor a ser convertido.
- IN_min - Limite inferior do intervalo de entrada
- IN_max - Limite superior do intervalo de entrada
- OUT_min - Limite inferior do intervalo de saída
- OUT_max - Limite superior do intervalo de saída

Essa função pode ser interessante quando queremos modular um pulso usando PWM que será controlado por um potenciômetro no caso de nosso exemplo hoje, lembrando que os valores medidos no potenciômetro podem variar de 0 a 1023.

15.2.6 Função `analogRead()`

Função para leitura de pinos analógicos, que pode ser valores de 0 a 1023. Isso significa que ele mapeará as tensões de entrada entre 0 e a tensão de operação (5V ou 3,3V) em valores inteiros entre 0 e 1023. Em uma UNO Arduino, por exemplo, isso gera uma resolução entre leituras de: 5 volts / 1024 unidades ou 0,0049 volts (4,9 mV) por unidade. Veja a tabela abaixo para obter os pinos utilizáveis, a tensão de operação e a resolução máxima para algumas placas Arduino.

Sintaxe: `analogRead(pino ou variável)`

15.2.7 Biblioteca “Servo.h”

O controle de um servo motor pode ser realizado com o auxílio da biblioteca Servo, que é pré-instalada na IDE do Arduino. Com esta biblioteca é possível controlar até 12 servomotores em um Arduino Uno e 48 em um Arduino Mega. Para utilizar a biblioteca é necessário incluí-la no início do sketch com a diretiva `#include < Servo. h >`. A biblioteca permite a associação de um pino ao motor (de forma semelhante a um `pinMode()`), a escrita de um ângulo em graus para rotacionar o servomotor, a obtenção do último ângulo escrito, dentre outras funções relacionadas conforme abaixo:

Nome da função	Uso	Descrição
Servo	<i>Servo nome</i>	Cria um objeto servo
attach()	<i>attach(pino)</i>	Conecta o servomotor a um pino digital.
write()	<i>write(angulo)</i>	Configura o ângulo de rotação do motor (em graus)
read()	<i>read()</i>	Retorna o último ângulo escrito (em graus)
attached()	<i>attached()</i>	Verifica se o motor está conectado
detach()	<i>detach()</i>	Desconecta o motor do pino

15.3 Material necessário

1 Arduino Uno



1 Micro Servomotor



1 Cabo USB AB

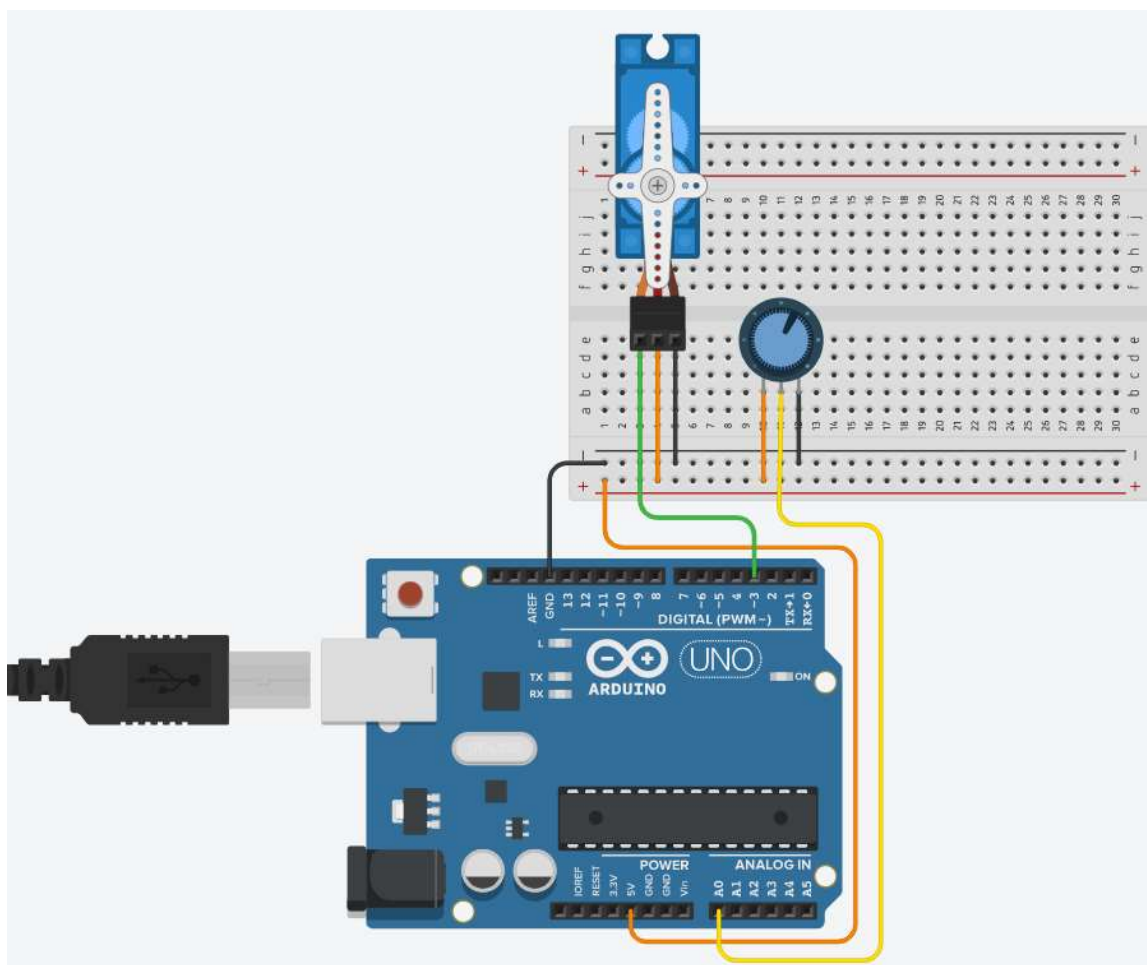


Jumpers

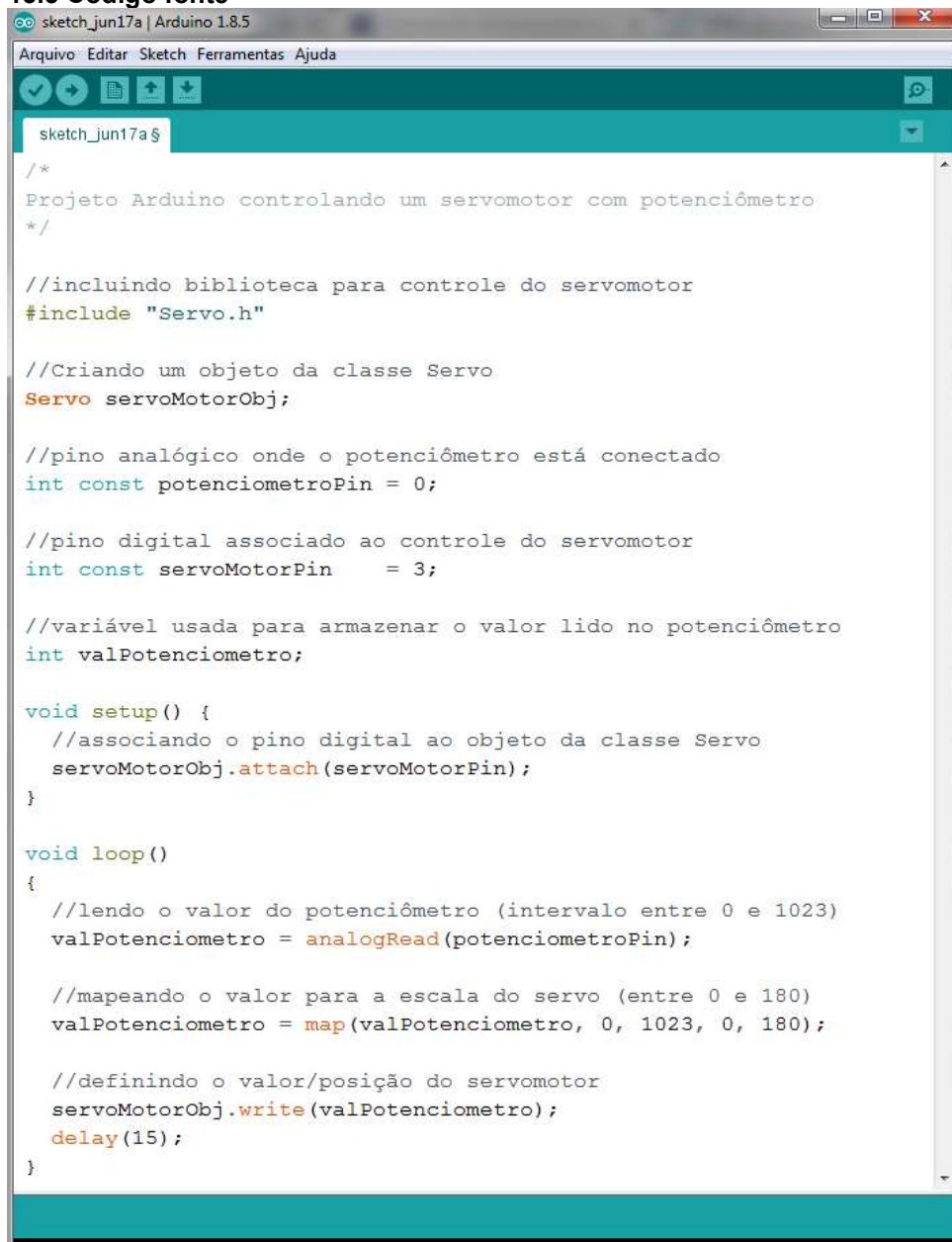


1 Protoboard

15.4 Diagrama



15.5 Código fonte



```

sketch_jun17a | Arduino 1.8.5
Arquivo Editar Sketch Ferramentas Ajuda

sketch_jun17a $

/*
Projeto Arduino controlando um servomotor com potenciômetro
*/

//incluindo biblioteca para controle do servomotor
#include "Servo.h"

//Criando um objeto da classe Servo
Servo servoMotorObj;

//pino analógico onde o potenciômetro está conectado
int const potenciometroPin = 0;

//pino digital associado ao controle do servomotor
int const servoMotorPin = 3;

//variável usada para armazenar o valor lido no potenciômetro
int valPotenciometro;

void setup() {
    //associando o pino digital ao objeto da classe Servo
    servoMotorObj.attach(servoMotorPin);
}

void loop()
{
    //lendo o valor do potenciômetro (intervalo entre 0 e 1023)
    valPotenciometro = analogRead(potenciometroPin);

    //mapeando o valor para a escala do servo (entre 0 e 180)
    valPotenciometro = map(valPotenciometro, 0, 1023, 0, 180);

    //definindo o valor/posição do servomotor
    servoMotorObj.write(valPotenciometro);
    delay(15);
}

```

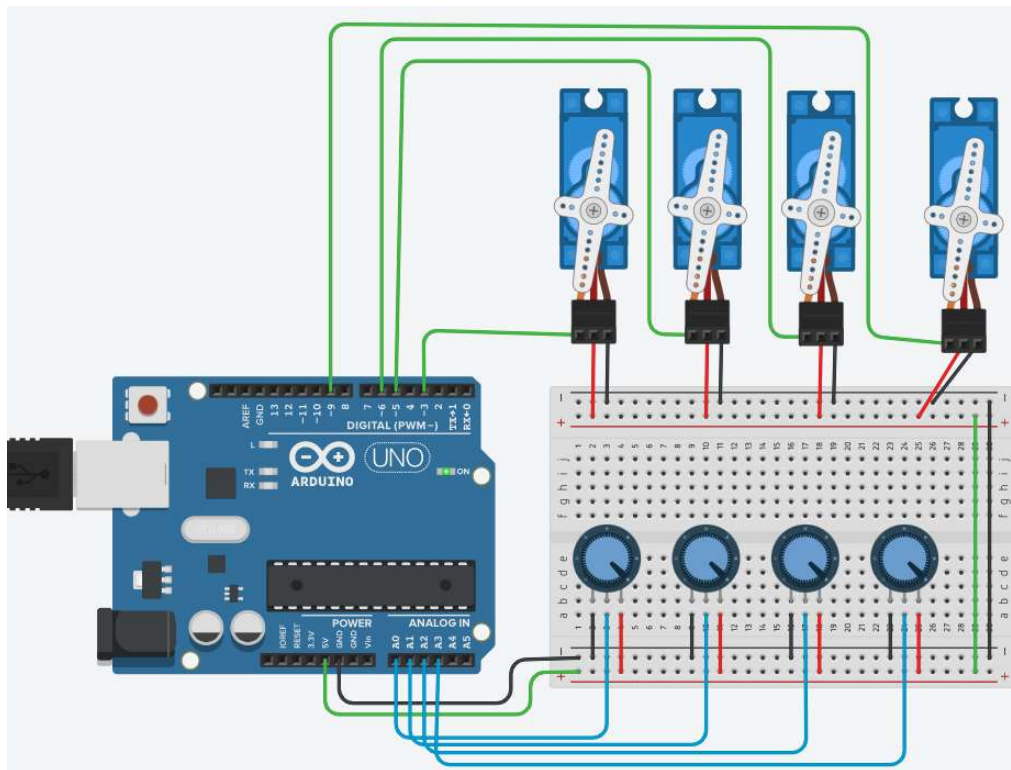
15.6 Exercício proposto

De acordo com a implementação anterior, programem agora 4 servos que tenham movimentações independentes, ou seja, cada servo terá seu respectivo potenciômetro.

15.6.1 Resolução do Exercício

Abaixo temos uma das opções de resolução do exercício, sempre lembrando que a lógica é o fator primordial para a execução correta o objetivo do exercício.

A seguir a montagem sugerida:



Agora temos o código:

```

1 #include "Servo.h"
2
3 //Instância dos 4 motores
4 Servo sM1;
5 Servo sM2;
6 Servo sM3;
7 Servo sM4;
8
9 //Portas analógicas para os potenciômetros
10 int const pPin1 = 0;
11 int const pPin2 = 1;
12 int const pPin3 = 2;
13 int const pPin4 = 3;
14
15 //Pinos PWM dos motores
16 int const sMotorPin1 = 3;
17 int const sMotorPin2 = 5;
18 int const sMotorPin3 = 6;
19 int const sMotorPin4 = 9;
20
21 //Dados recebidos de leitura
22 int valPot1;
23 int valPot2;
24 int valPot3;
25 int valPot4;
26
27

```



```
28 void setup()
29 {
30     //"Atachando" os motores as portas
31     sM1.attach(sMotorPin1);
32     sM2.attach(sMotorPin2);
33     sM3.attach(sMotorPin3);
34     sM4.attach(sMotorPin4);
35 }
36
37 void loop()
38 {
39     //Rotina para o 1° motor
40     valPot1 = analogRead(pPin1); //Leitura do potenciômetros
41     valPot1 = map(valPot1, 0, 1023, 0, 180); //Cálculo para graus
42     sM1.write(valPot1); //Movimentação do motor
43
44     //Rotina para o 2° motor
45     valPot2 = analogRead(pPin2); //Leitura do potenciômetros
46     valPot2 = map(valPot2, 0, 1023, 0, 180); //Cálculo para graus
47     sM2.write(valPot2); //Movimentação do motor
48
49     //Rotina para o 3° motor
50     valPot3 = analogRead(pPin3); //Leitura do potenciômetros
51     valPot3 = map(valPot3, 0, 1023, 0, 180); //Cálculo para graus
52     sM3.write(valPot3); //Movimentação do motor
53
54     //Rotina para o 4° motor
55     valPot4 = analogRead(pPin4); //Leitura do potenciômetros
56     valPot4 = map(valPot4, 0, 1023, 0, 180); //Cálculo para graus
57     sM4.write(valPot4); //Movimentação do motor
58
59     delay(15);
60 }
```

Notem que o código ficou maior, com os motores acrescentados e sua respectiva lógica, iremos na sequência conhecer mais funções, estruturas e lógicas que irão a minimizar o trabalho e dar mais agilidade a nosso desenvolvimento.

16. Fazendo um braço robótico totalmente autônomo

Projeto que visa fazer um braço robótico com suas atividades totalmente autônomas sem intervenção humana direta, para isso criaremos um script com os movimentos simples e deixar o funcionamento por conta da lógica aplicada.

16.1 O que vou aprender?

- Função;
- Estrutura condicional *IF ELSE*;
- Estrutura de repetição *FOR*;

16.2 Função

Uma *função* em programação é simplesmente um bloco de código separado que recebe um nome para execução de uma determinada tarefa. Entretanto, funções também podem receber parâmetros e/ou retornar dados. Abaixo a sintaxe geral:

```
void NOME_DA_FUNCAO (tipo_de_dado PARAMETRO){
    //Comando(s) da função
}
```

Em nosso caso, iremos passar parâmetro(tipo de movimento) para que os motores do braço robótico. Como exemplo abaixo:

```
void esticaMovimento(char tipoMov) {
    if (tipoMov == 'E') {
        for (int x = 0; x <= 63; x++) {
            servoEstica.write(x);
            delay(100);
        }
    } else {
        for (int x = 63; x >= 0; x--) {
            servoEstica.write(x);
            delay(100);
        }
    }
}
```

parâmetro que será recebido pela função

nome da função

16.3 Estrutura condicional IF ELSE

Em programação, uma estrutura chamada de desvio condicional tem por finalidade tomar uma decisão de acordo com o resultado de uma determinada condição (chamada de teste lógico), e então executar um bloco de códigos específico, o qual irá depender do resultado desse teste.

O condicional IF permite executar um bloco de instruções caso o resultado do teste lógico retorne o valor verdadeiro (*true*), e não realiza nenhuma ação caso o teste lógico retorne o valor falso (*false*) indo para o *ELSE* onde obrigatoriamente os comandos nele contidos são executados. Abaixo a sintaxe geral.

```
if (condição){
    //Comando(s) caso True
}else{
    //Comando(s) caso False
}
```

Podemos ver no exemplo em um dos métodos onde chamamos a execução em um dos servos motores de nosso braço. Veja a seguir:

```
void pincaMovimento(char tipoMov) {
    if (tipoMov == 'F') {
        for (int x = 150; x <= 180; x = x + 2) {
            servoPinca.write(x);
            delay(100);
        }
    } else {
        for (int x = 180; x >= 150; x = x - 2) {
            servoPinca.write(x);
            delay(100);
        }
    }
}
```

Caso verdadeiro

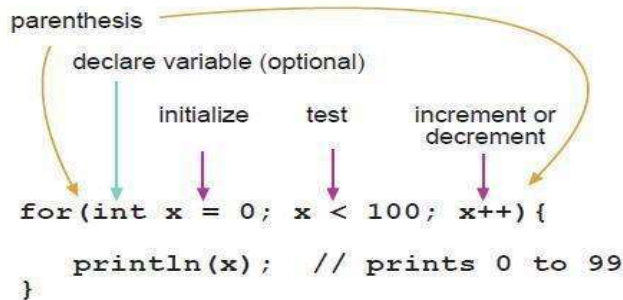
Caso falso

16.4 Estrutura de repetição FOR

A sentença for é utilizada para repetir um bloco de código delimitado por chaves. Um contador com incremento normalmente é usado para controlar e finalizar o loop. A sentença for é útil para qualquer operação repetitiva.

Há três elementos na montagem da estrutura FOR:

```
for ( initialization ; condition ; increment ) {
    //statement(s);
}
```



A inicialização acontece primeiro e exatamente uma vez. Cada vez através do loop, a condição é testada; se é verdade, o bloco de declaração, e o incremento é executada, em seguida, a condição é testada novamente. Quando a condição se torna falsa, o loop termina.

Utilizamos essa estrutura em nosso projeto para que o braço robótico se movimente “grau a grau” tornando mais suave o giro do servo motor. Veja:

```

void baseMovimento(int angulo) {
    for (int x = 90; x <= angulo; x++) {
        servoBase.write(x);
        delay(100);
    }
    for (int x = 180; x >= 90; x--) {
        servoBase.write(x);
        delay(100);
    }
}

```

Estrutura para que o servo se movimente "grau a grau"

16.5 Material necessário

1 Arduino Uno



1 Braço Robótico



4 Micro Servomotores



1 Cabo USB AB

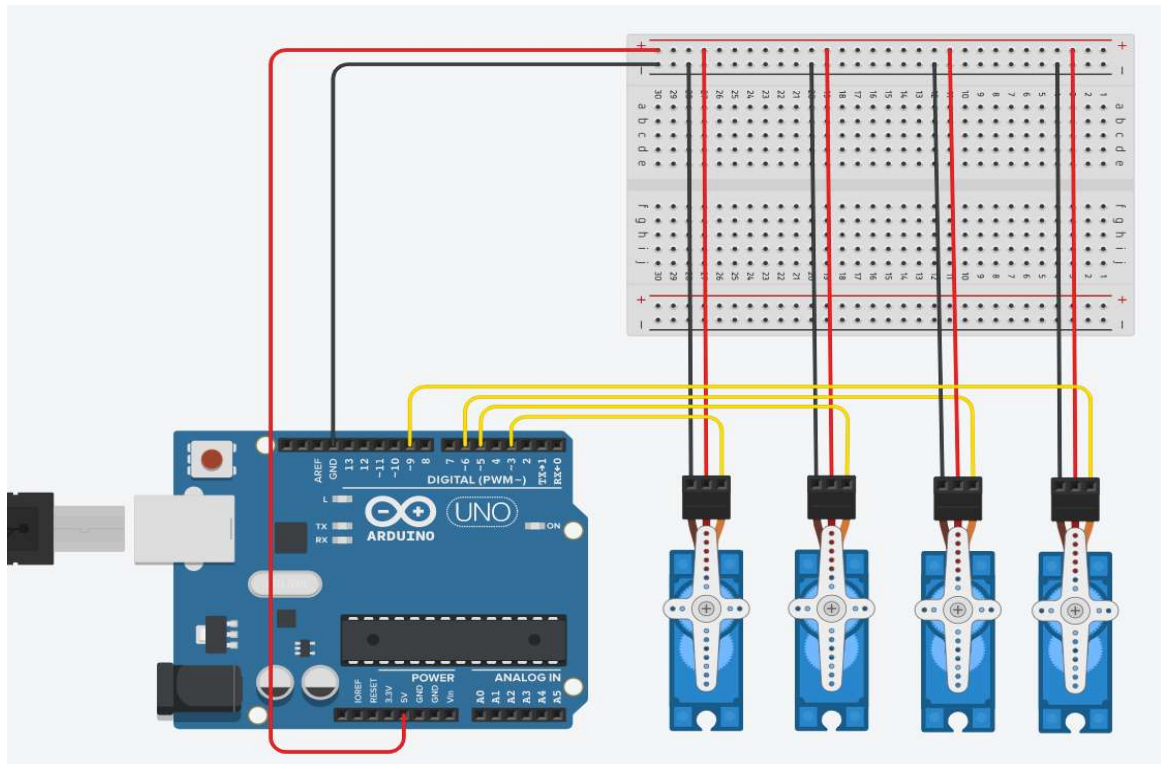


Jumpers



1 Protoboard

16.6 Diagrama



16.7 Código fonte

```

Braco_basico
#include <Servo.h>           //Biblioteca para servo motores

Servo servoEstica;           //Objeto do motor para esticar o braço robótico
Servo servoPinca;            //Objeto do motor para pinçar o braço robótico
Servo servoBase;             //Objeto do motor para girar o braço robótico
Servo servoHorizontal;       //Objeto do motor para subir e descer o braço robótico

int const servoEsticaPin     = 5; //Pino PWM para motor
int const servoPincaPin      = 6; //Pino PWM para motor
int const servoBasePin       = 9; //Pino PWM para motor
int const servoHorizontalPin = 3; //Pino PWM para motor

void setup() {
    servoEstica.attach(servoEsticaPin);
    servoPinca.attach(servoPincaPin);
    servoBase.attach(servoBasePin);
    servoHorizontal.attach(servoHorizontalPin);
}

```

```
void loop() {
    //Script de movimento do braço robótico
    esticaMovimento('E');      //E -> Esticar
    pincaMovimento('A');       //A -> Abrir
    baseMovimento(180);
    horizontalMovimento('D');  //D -> Descer
    pincaMovimento('F');       //F -> Fechar
    esticaMovimento('R');      //R -> Retrair
    horizontalMovimento('S');  //S -> Subir
}

void esticaMovimento(char tipoMov){
    //Função para ESTICAR ou RETRAIR o braço robótico
    if (tipoMov == 'E'){
        for (int x = 0; x <= 63; x++){
            servoEstica.write(x);
            delay(100);
        }
    }else{
        for (int x = 63; x >= 0; x--){
            servoEstica.write(x);
            delay(100);
        }
    }
}

void pincaMovimento(char tipoMov){
    //Função para ABRIR ou FECHAR a pinça do braço robótico
    if (tipoMov == 'F'){
        for (int x = 150; x<= 180; x = x + 2){
            servoPinca.write(x);
            delay(100);
        }
    }else{
        for (int x = 180; x >= 150; x = x - 2){
            servoPinca.write(x);
            delay(100);
        }
    }
}

void baseMovimento(int angulo){
    //Função para GIRAR o braço robótico
    for (int x = 90; x <= angulo; x++){
        servoBase.write(x);
        delay(100);
    }
    for (int x = 180; x >= 90; x--){
        servoBase.write(x);
        delay(100);
    }
}
```



```
void horizontalMovimento(char tipoMov){
  //Função para SUBIR ou DESCER o braço robótico
  if (tipoMov == 'D'){
    for(int x = 0; x <= 100; x++){
      servoHorizontal.write(x);
      delay(100);
    }
  }else{
    for(int x = 100; x <= 0; x--){
      servoHorizontal.write(x);
      delay(100);
    }
  }
}
```

Salvo

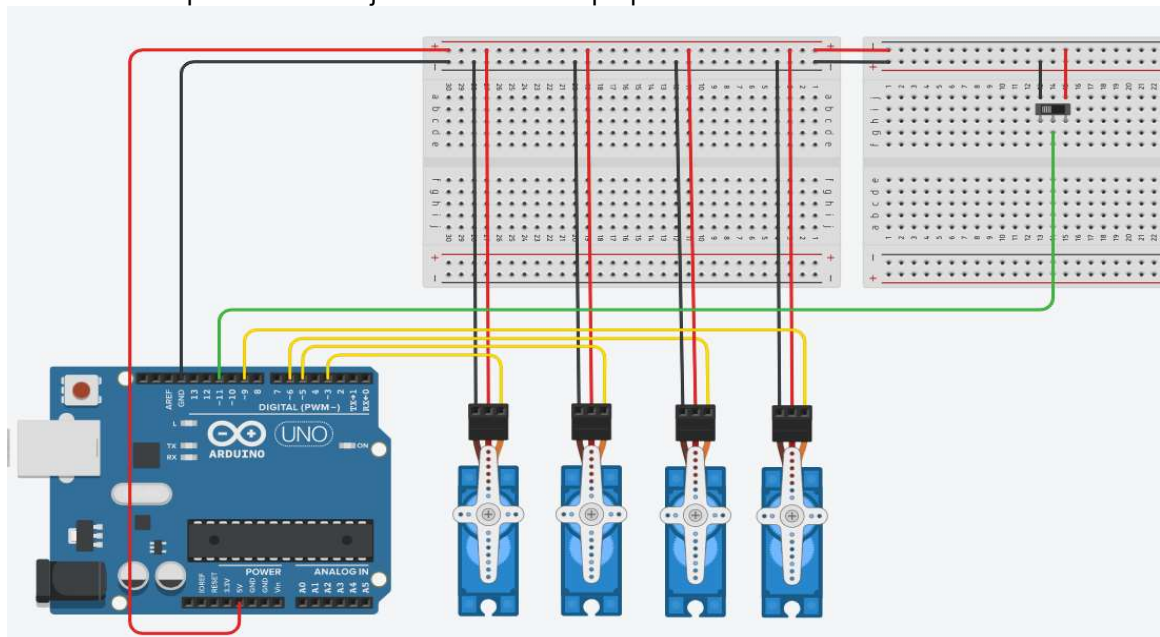
16.8 Exercício

Agora, tentem colocar um interruptor deslizante no código com a seguinte condição, quando o estiver “LIGADO” o braço robótico faz o processo automático, caso contrário, ele fica parado. O interruptor deslizante é esse:



16.8.1. Correção

O diagrama ficaria parecido com isso, mas lembre que sua forma de resolver também é correta desde que atenda o objetivo do exercício proposto:



Agora o código:

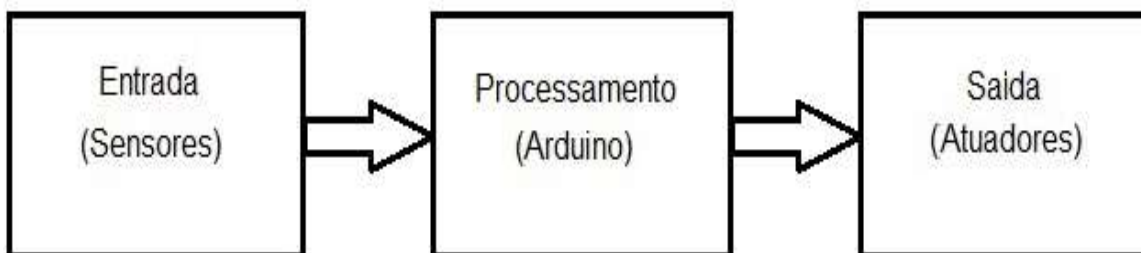

```

1  #include <Servo.h>
2  Servo servoEstica; //cria um novo objeto servo
3  Servo servoPinca;
4  Servo servoBase;
5  Servo servoHorizontal;
6
7  int const servoEsticaPin      = 5;
8  int const servoPincaPin      = 6;
9  int const servoBasePin       = 9;
10 int const servoHorizontalPin = 3;
11
12 int const pinBotao           = 11;
13 int leitura;
14
15 void setup(){
16     servoEstica.attach(servoEsticaPin);
17     servoPinca.attach(servoPincaPin);
18     servoBase.attach(servoBasePin);
19     servoHorizontal.attach(servoHorizontalPin);
20     pinMode(pinBotao, INPUT);
21 }
22
23 void loop(){
24     leitura = digitalRead(pinBotao);
25
26     if (leitura == 1){
27         esticaMovimento('E');          //E -> Esticar
28         pincaMovimento('A');           //A -> Abrir
29         baseMovimento(180);
30         horizontalMovimento('D');      //D-> Descer
31         pincaMovimento('F');           //F - Fechar
32         esticaMovimento('R');          //R -> Retrair
33         horizontalMovimento('S');      //S-> Subir
34     }
35 }
36
37 void esticaMovimento(char tipoMov){
38     if (tipoMov == 'E'){
39         for (int x = 0; x <= 63; x++){
40             servoEstica.write(x);
41             delay(100);
42         }
43     }else{
44         for (int x = 63; x >= 0; x--){
45             servoEstica.write(x);
46             delay(100);
47         }
48     }
49 }
50
51 void pincaMovimento(char tipoMov){
52     if (tipoMov == 'F'){
53         for (int x = 150; x<= 180; x = x + 2){
54             servoPinca.write(x);
55             delay(100);
56         }
57     }else{
58         for (int x = 180; x >= 150; x = x - 2){
59             servoPinca.write(x);
60             delay(100);
61         }
62     }
63 }
64 }

```

17. Sensores e atuadores

Como vimos no início de nossa UC, temos as três fases importantes:



Iremos abordar alguns dispositivos que comportam essas fases (Entrada e Saída).

17.1. Sensores:

Os sensores já são uma tecnologia um tanto antiga utilizada para diferentes fins. Seja para uso doméstico ou até mesmo empresarial, com o tempo novos modelos foram sendo lançados no mercado, sempre trazendo formas mais modernas e atualizadas de se realizar suas principais funções.

No geral, todo tipo de sensor é um dispositivo que detecta determinados estímulos e desencadeia reações específicas a partir disso. O que diferencia um do outro é justamente o tipo de estímulo que eles respondem, que pode variar entre calor, luz, movimentos, pressão etc. A ação também varia de acordo com a finalidade do produto, então existem sensores capazes de iluminar, outros que reforçam a segurança do local, que abrem portas e por aí vai.

A seguir, vamos conhecer quais são os diferentes tipos de sensores, como eles funcionam e qual seria o mais indicado para você, de acordo com suas principais necessidades.

17.1.2. Tipos de sensores

Como explicado anteriormente, o principal diferencial de um sensor é o tipo de estímulo que ele é capaz de detectar, existindo uma boa variedade com diferentes tecnologias. Confira quais são os principais:

17.1.2.1. Sensor mecânico

Esses são um dos mais populares no nosso dia a dia, pois são os responsáveis por detectar qualquer tipo de movimentação ao redor. Através de recursos mecânicos, é possível realizar algumas funções na presença de qualquer pessoa, como acender uma ou várias luzes, abrir portas automaticamente e coisas do tipo. É algo que vemos de forma recorrente em estabelecimentos públicos, como shoppings, por exemplo.

Já no uso doméstico, é mais comumente utilizado para iluminar certos pontos da casa na presença de pessoas, o que pode servir tanto para facilitar o dia a dia quanto para reforçar a segurança, pois também serve como sinal de alerta no caso de invasores.

17.1.2.2. Sensor elétrico

Um sensor elétrico é capaz de detectar qualquer tipo de variação na corrente e tensão elétrica da instalação local, assim sendo capaz de desempenhar funções que, em sua maioria, se resumem a proteger os aparelhos eletrônicos locais de qualquer sobrecarga e instabilidade elétrica.

Esses sensores podem ser encontrados nos protetores eletrônicos, aparelhos dedicados a proteger qualquer eletrônico conectado de variações de energia. Quando ocorre uma queda de eletricidade, eles automaticamente desligam para evitar que os dispositivos sejam danificados com o aumento súbito de tensão.

17.1.2.3. Sensor acústico

Esse tipo de sensor se orienta através do eco propagado na velocidade do som, mais utilizado para medir distâncias. Não é um sensor utilizado para fins domésticos, mas está presente em alguns objetos do dia a dia, como nas câmeras polaroides e em carros mais modernos, que contam com um sistema de estacionamento mais avançado e seguro. O sensor alerta o motorista quando a traseira do carro está muito próxima de alguma parede ou objeto, assim evitando batidas.

17.1.2.4. Sensor magnético

Fabricado com a ajuda de ímãs, os sensores magnéticos conseguem detectar a abertura de portas e janelas através de seus componentes de metal. Quando posicionado em algum ponto de entrada da casa, o sensor dispara um alarme sempre que sentir movimentação naquela porta ou janela, servindo como um excelente recurso contra arrombamentos e invasões.

17.1.2.5. Sensor térmico

Como o nome já entrega, esse sensor é capaz de captar variações na temperatura para diferentes fins. Um simples objeto como um termômetro funciona como um sensor térmico, mas em aplicações estruturais, geralmente são sensores utilizados para regular a temperatura ambiente em salas que precisam se manter em temperaturas intensas.

17.1.2.6. Sensor óptico

O sensor óptico ou fotoelétrico funciona a partir da propagação da luz para detectar pessoas e objetos. É a tecnologia utilizada em portas de elevadores, por exemplo, assim como em outros objetos do dia a dia como mouses e leitores de código de barras. Veículos mais modernos que contam com sensores de ré também utilizam esse recurso.

17.2. Sensores, alguns modelos.

17.2.1 LDR

LDR (do inglês Light Dependent Resistor), em português Resistor Dependente de Luz é um componente eletrônico passivo do tipo resistor variável, mais especificamente, é um resistor cuja resistência varia conforme a intensidade da luz (iluminação) que incide sobre ele. Tipicamente, à medida que a intensidade da luz aumenta, a sua resistência diminui.

O LDR é construído a partir de material semicondutor com elevada resistência elétrica. Quando a luz que incide sobre o semicondutor tem uma frequência suficiente, os fótons que incidem sobre o semicondutor libertam elétrons para a banda condutora que irão melhorar a sua condutividade e assim diminuir a resistência.



Sensor LDR

17.2.2 Sensor de temperatura LM35

O **Sensor de Temperatura LM35** é um sensor de precisão, que apresenta uma saída de tensão linear relativa à temperatura em que ele se encontrar no momento em que for alimentado por uma tensão de 4-20Vdc e GND, tendo em sua saída um sinal de 10mV para cada Grau Celsius de temperatura, sendo assim, apresenta uma boa vantagem com relação aos demais sensores de temperatura calibrados em “KELVIN”,

não necessitando nenhuma subtração de variáveis para que se obtenha uma escala de temperatura em Graus Celsius.

O **LM35** não necessita de qualquer calibração externa ou “*trimming*” para fornecer com exatidão, valores temperatura com variações de 0,25°C ou até mesmo 0,75°C dentro da faixa de temperatura de –55°C à 150°C. Este sensor tem saída com baixa impedância, tensão linear e calibração inerente precisa, fazendo com que o interfaceamento de leitura seja especificamente simples e o custo do sistema como um todo seja significativamente baixo.



Sensor LM35

17.2.3 Sensor Infravermelho Reflexivo de Obstáculo

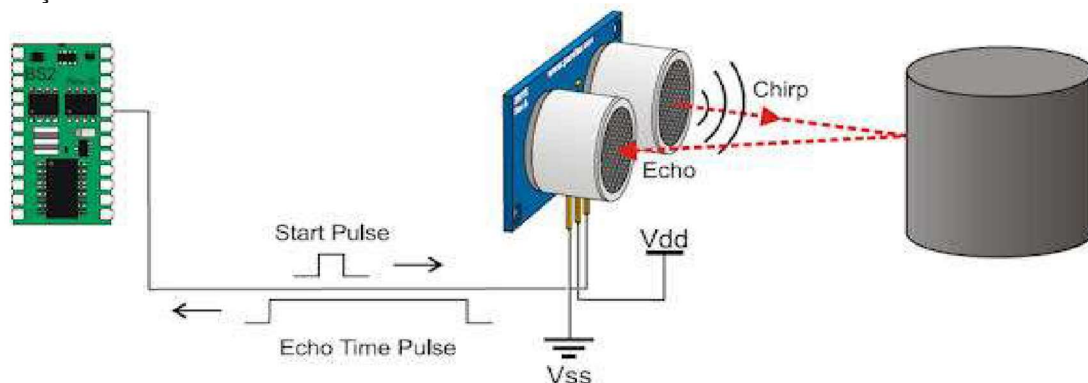
O **Sensor Infravermelho Reflexivo de Obstáculo** funciona através de um sistema de reflexão infravermelho, onde um LED emissor IR e um fototransistor IR ficam lado a lado e quando um obstáculo ou objeto atravessa dentro do raio de ação de ambos o sensor coloca a saída em nível baixo e um LED verde será aceso na placa indicando presença de um obstáculo ou objeto. Este sensor possui um potenciômetro ajustável para controle da sensibilidade da distância de detecção que pode ficar entre 2cm e 30cm. Vale ressaltar que o tamanho e cor do obstáculo ou objeto influenciam na distância de detecção.



Sensor Infravermelho reflexivo

17.2.4 Sensor Ultrassônico

O sensor é usado para medir distâncias. A distância do sensor de ultrassom comum de medição é de cerca de 2 cm a 3-5m. O módulo sensor funciona assim:



O sensor ultrassônico é composto de um emissor e um receptor de ondas sonoras. Podemos compará-los a um alto-falante e um microfone trabalhando em conjunto. Entretanto, ambos trabalham com ondas de altíssima frequência, na faixa dos 40.000 Hz (ou 40KHz). Isto é muito, muito acima do que os nossos ouvidos são capazes de perceber. O ouvido humano

consegue, normalmente, perceber ondas na entre 20 e 20.000 Hz e por isto o sinal emitido pelo sensor ultrassônico passa despercebido por nós.

O sinal emitido, ao colidir com qualquer obstáculo, é refletido de volta na direção do sensor. Durante todo o processo, o aparelho está com uma espécie de “cronômetro” de alta precisão funcionando. Assim, podemos saber quanto tempo o sinal levou desde a sua emissão até o seu retorno.



Sensor ultrassônico

17.2.5 Botão

A **Chave Táctil / Push Button** como também é conhecido, é um dos componentes eletrônicos mais utilizados para prototipagem de projetos. Esta chave é um tipo de interruptor pulsador (conduz somente quando está pressionado).



Push button

17.2.6 Potenciômetro

Como vimos aqui em nossa UC, um potenciômetro ou potenciómetro é um componente eletrônico que possui resistência elétrica ajustável, que possui normalmente três terminais onde a conexão central é deslizante e manipulável via cursor móvel.



Potenciômetro

17.3. Atuadores:

Um atuador é um dispositivo de uma máquina que faz a movimentações ou o controle de cargas e mecanismos. O atuador pode ser operado por energia, seja por fluido pressurizado, ar ou eletricidade/ A energia então é convertida em movimentos ou força para atingir controle.

17.3.1. Buzzer

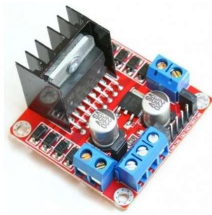
Buzzer é um componente de geração de ruídos sonoros a partir da excitação elétrica de componentes eletromecânicos ou piezoelétricos. As principais aplicações do *Buzzer* são alarmes, campainhas, despertadores, brinquedos e sistemas automatizados.



Buzzer

17.3.2. Ponte H – L298N

O Módulo Driver Ponte H - L298N é um módulo de controle capaz de trabalhar com dois motores DC ou um motor de passo. Ele é baseado no driver L298N, sendo muito utilizado com Arduino e diversos outros microcontroladores.



Ponte H – L298N

17.3.3. 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.



Ponte H – L298N

17.3.4. Servomotor

Como já vimos nessa UC, os atuadores temos um motor bem especial. Os servomotores, também chamados de servos, são muito utilizados quando o assunto é robótica. De forma simplificada, um servomotor é um motor na qual podemos controlar sua posição angular através de um sinal PWM.



Servomotor

Dessa forma, um servomotor é um atuador eletromecânico utilizado para posicionar e manter um objeto em uma determinada posição.

17.3.5. Relé

Os relés são componentes eletromecânicos capazes de controlar circuitos externos de grandes correntes a partir de pequenas correntes ou tensões, ou seja, acionando um relé com uma pilha podemos controlar um motor que esteja ligado em 110 ou 220 volts, por exemplo.

O Módulo Relé 5V 1 Canal permite que a partir de uma plataforma microcontrolada seja possível controlar cargas AC (alternada) de forma simples e prática. Por ter apenas 1 canal, é possível controlar apenas uma carga AC de até 10A. Comumente é utilizado em projetos de automação residencial para controle de lâmpadas, ventiladores e outras saídas que possam ser acionadas através de relé.



Relé

17.3.6. LCD 16x2

O *Display* LCD 16x2 é um tipo de display LCD categorizado como do tipo caractere. O que isso significa? Quer dizer que ele é dividido e identificado pelo número de caracteres possíveis de serem exibidos. No caso do 16x2, significa que ele possui 16 espaços na horizontal (ou seja, 16 caracteres) e 2 linhas.

Existem outros valores como 20x4, 40x2, 8x2, e assim por diante.

Para esse tipo de display, é utilizada a biblioteca *LiquidCrystal*, pois possui diversas funções prontas para controle do display. A comunicação pode ser feita de forma comum (utilizando 12 pinos do microcontrolador como o Arduino) ou através do I2C, assim gastando apenas 2 pinos para conectar de forma correta, e esse é que temos aqui no Senac.



LCD 16x2 com I2C

17.4 Exercício - Atividade em grupo

Orientações para o desenvolvimento da atividade:

- Organizarem os grupos;
- O grupo deverá desenvolver no *Tinkercad* um projeto que seja capaz de, através de um sensor, captar dados do ambiente externo e enviá-lo ao arduino que, por sua vez, será codificado para transmitir esses dados através de atuadores.
- Esse projeto deverá ser pensado para resolver algum problema real, por exemplo: Utilizar o sensor LM35 para que seja possível captar a temperatura de um frigorífico a qual devem ser menores que 3°C, caso passe desta temperatura um alarme e ou um sinalizador deve(m) ser acionado(s) para que seja verificado o ocorrido.
- O problema a ser resolvido deve ser apresentado ao professor;

- Após o perfeito funcionamento do projeto no tinkercad, deverá ser desenvolvido fisicamente (com os componentes reais).

REGRAS:

- Cada grupo deverá escolher um sensor o qual não poderá se repetir entre os grupos;
- Os grupos podem utilizar quantos atuadores achar por necessário de acordo com o que foi pensado para o desenvolvimento;
- Os projetos não podem ser realizados para resolver o mesmo problema real.