UFRRJ

UNIVERSIDADE FEDERAL RURAL
DO RIO DE JANEIRO

Introdução à programação física

Um auxílio ao laboratório de física

William Wayn Monteiro

Prefácio

Este material foi produzido com intuito de complementar o conteúdo ministrado no curso de introdução aos microcontroladores durante a semana acadêmica de 2017 do curso de física da Universidade Federal Rural do Rio de Janeiro. Este conteúdo procura instigar os alunos de física a buscarem por alternativas e melhorias aos experimentos que podem ser encontrados nos laboratórios de física experimental I, II, III, IV e moderna.

SUMÁRIO

1 INTRODUÇÃO.

Os microcontroladores foram inventados por volta de 1970 pela empresa Texas Instruments e podemos com certa aproximação os comparar com computadores, pois, os microcontroladores são sistemas computacionais completos, possuindo uma unidade central de processamento, sistemas de clock, memoria de armazenamento de instruções e manipulação de dados, saídas I/O (Input, output) para processar informações do meio externo e firmware para definir as funcionalidades do sistema. Por conta disto e graças a sua grande margem de utilidade, em poucas décadas os microcontroladores passaram a fazer parte de nosso cotidiano, de tal forma que já não conseguimos nos ver sem eles, pois os mesmos já então presentes em praticamente qualquer eletrônico de nosso cotidiano, podemos os encontrar em eletrodomésticos como os micro-ondas presentem em nossas cozinhas, na injeção eletrônica dos carros que presenciamos diariamente, em diversos brinquedos, telefones, etc.

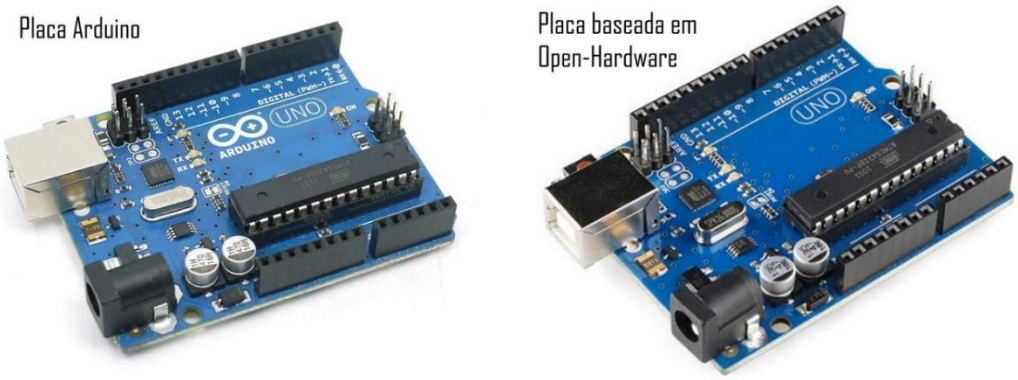
Estes microcontroladores inicialmente apresentavam grande dificuldade de serem manipulados, pois os mesmos precisam ser programados para realizar suas funções e essa programação no início era possível apenas através da linguagem Assembly e posteriormente em c, porem esses códigos não eram triviais pois se tratavam de linguagens de *baixíssimo nível* (bem distante de uma "linguagem de ser humano") e de difícil entendimento para leigos, além disto, esses microcontroladores para serem programados precisavam de uma plataforma especial feita apenas com este intuito e com um preço pouco acessível, tornando ainda mais inviável para um público geral sendo assim um resumo seria, para utilizarmos estes microcontroladores em nossos projetos, deveríamos inicialmente escolher qual microcontrolador usaríamos em nosso projeto, já que existem muitos modelos de microcontroladores assim como a plataforma que faria o upload do código computacional para o microcontrolador, após, programaríamos nossos microcontroladores com o código da função que o mesmo deveria ter, então levaríamos para a plataforma responsável pela compilação e faríamos o upload do código, após levaríamos o microcontrolador para a plataforma que gostaríamos de testar e caso algo não estivesse de acordo, então, deveríamos retirar o mesmo do projeto, levar a plataforma de compilação e o reprogramar, esta dificuldade então, pode ser o um dos principais motivos pelo qual o Arduino ter se tornado tão popular nos últimos anos, basicamente o Arduino é o conjunto de um hardware que usa como "cérebro" um microcontrolador, em um sistema embarcado o que facilita bastante seu uso para múltiplos projetos, juntamente com um software de desenvolvimento e compilação padronizado com uma linguagem de *alto nível* (Bem próxima a nossa linguagem), modulada em Processing, Writing e C. O Arduino foi um projeto que tinha como objetivo de ser um recurso educacional e que fossem baratos, funcionais e principalmente fáceis de programar por um público leigo em computação.

1.1 HARDWARE LIVRE.

O Arduino faz parte de um conceito conhecido como hardware livre que teve sua inspiração em um conceito bem parecido chamado de software livre, o qual se popularizou atraves do sistema operacional Linux, cujo seu código fonte é aberto, possibilitando assim que se faça modificações no codigo do programa original afim de se criar uma nova versão do sistema operacional Linux, achar falhas no sistemas existentes, entre outras possibilidades. Com isto os circuitos eletronicos ou hardwares das empresas open hardware podem ser copiados e editados livremente, já que seus diagramas esquematicos com lista de componentes e outras informações são distribuidos pelo proprio desenvolvedor, sem que haja qualquer taxa de licença, no caso do Arduino a única limitação que existe é que as placas feitas a partir de seu modelo não venham escrito Arduino, sua logo ou a frase “Made in Italy”, principalmente caso não tenha sido feita na italia.

Figura 1 - Arduino original x baseada em open-hardware

(FilipeFlop, 2017)

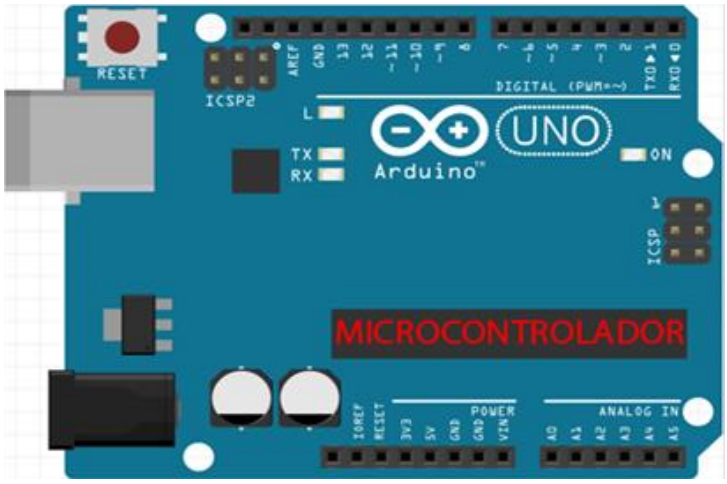


2 CARACTERÍSTICAS DA PLACA ARDUINO

Ao entrarmos no site oficial do Arduino na aba de produtos, link https://www.arduino.cc/en/Main/Products, podemos perceber que existem algumas dezenas de modelos de placas oficiais, onde cada placa apresenta características e finalidades distintas. Por conta disto, esta apostila então ira se guiar no modelo mais vendido e usado atualmente como parâmetro o Arduino Uno o qual possui o microcontrolador ATMega328.

2.1 Hardware do Arduino Uno.

Figura 2 - Imagem Arduino UNO



É possível visualizar no canto superior da Figura 2 os pinos com numeração de 0 a 13, assim como que alguns apresentam o símbolo (~) ao lado de seu número. Estes pinos são os pinos digitais, e os sinais (~), representam os pinos digitais que podem ser saídas PWM.

No canto inferior direito, podemos ver os pinos analógicos que possuem valores de A0 a A5. Ainda na parte inferior da placa é possível visualizar as entradas de potencial zero (GND - ground), assim como as saídas de 5 e 3.3v. Escrito em vermelho, temos o "cérebro" do Arduino, se trata de seu microcontrolador ATMEGA328, o qual iremos falar mais a respeito na sessão a seguir.

2.1.1 Microcontrolador ATMEGA328

Existem diversas versões de placas Arduino no site oficial do Arduino, algumas, por exemplo, com WI-FI ou ethernet LAN integrados, mas essas diferenças podem ser adicionadas ao Arduino através de módulos ou Shields, como veremos no capitulo 3, por conta disso, podemos dizer que a principal diferença entre os Arduinos oficiais que estão atualmente no mercado, diz respeito principalmente ao microcontrolador que o acompanha, então, iremos introduzir uma comparação apenas com três modelos de microcontroladores, para posteriormente detalhar suas funcionalidades através do ATMega328, que é o microcontrolador que acompanha o Arduino Uno.

	ATtiny85 (LilyPad Arduino)	ATMega328 (Arduino Uno)	ATMega1280 (Arduino Mega)
Pinos Digitais I/O	6 (2 podem ser PWM)	14 (6 podem ser PWM)	54 (14 podem ser PWM)
Pinos Analógicos	4	6	16
Memória Flash	8kb	32KB (0.5kb são usadas pelo bootloader)	128kb (4kb são usadas pelo bootloader)
SRAM	512bits	2kb	8kb
EEPROM	512bits	1kb	4kb
Clock Speed	8MHz	16MHz	16MHz
Preço no site Arduino.cc (2017)	US 20	US 25	US 46

- **MEMORIA FLASH** – Nossa memória flash serve para armazenar nossos sketchs e bootloader, o microcontrolador ATMega328 possui 32kb de ***memória flash*** disponível.
    - ***SRAM*** (Static random access memory), Do português “Memoria estática de acesso aleatório”, a SRAM se comporta de forma semelhante as memorias rams dos computadores, são usadas para armazenar e alterar as variáveis de nossos programas. Ao resetarmos o sistema, esses dados são perdidos.
    - ***EEPROM*** (Electrically-erasable programmable read-only memory), EEPROM, é o local onde podemos gravar dados sem perdê-los, mesmo depois de resets, geralmente esse local da memória é usado para gravar constantes como π.
    - ***Clock Speed*** Funciona como um contador de tempo do microcontrolador, e ajuda a ditar a velocidade em que o microcontrolador irá executar suas tarefas. Suas unidades são em Hertz, 1 Mhz (mega hertz) significa que nosso microcontrolador processa 1 milhão de ciclos por segundo.

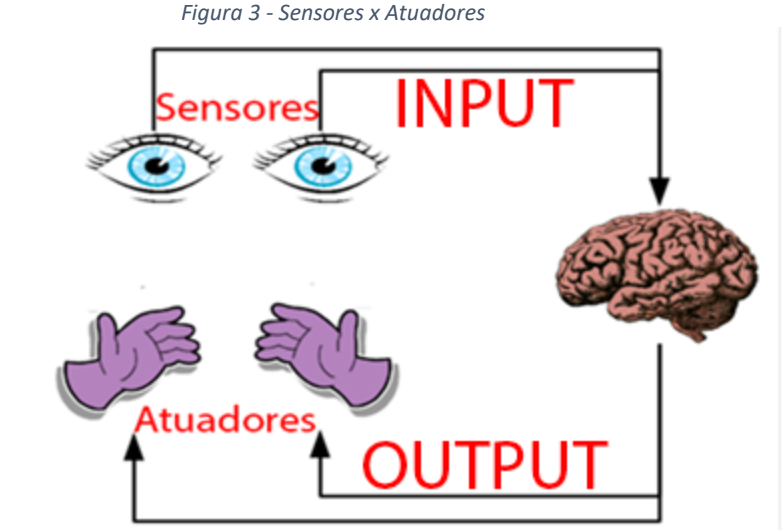
3 SENSORES, ATUADORES, MÓDULOS E SHIELDS.

Para realizarmos nossas tarefas com o Arduino, precisamos juntar conhecimentos de eletrônica e programação. Por conta disto, este capitulo falará a respeito da parte de eletrônica básica que será de extrema necessidade para compreender como se faz a ligação da placa Arduino com o mundo externo, levando em consideração que os alunos possuem um conhecimento prévio mínimo de eletromagnetismo.

3.1 ATUADORES X SENSORES

O principal conceito que precisamos compreender é; O que são os sensores e atuadores, pois, os encontramos em diversos aparelhos eletrônicos de nossos cotidianos e estes serão os responsáveis pela conexão do mundo externo com o microcontrolador.

Figura 3 - Sensores x Atuadores



A Figura 3 - Sensores x Atuadores se trata de um exemplo de um sensor e um atuador, pois, quando enxergamos uma imagem, estamos utilizando nossos sensores óticos, que enviam informações a nosso cérebro o qual decodifica e interpreta os sinais, e com isto pode responder movimentando nossos braços com intuito de interagir ou atuar com o meio, isto é, através de instruções e orientações previas de nosso cérebro.
Em resumo, um Sensor recebe dados externos e envia ao processamento sem interagir com o meio, e o atuador envia dados para interagir com o meio. Em nosso microcontrolador, possuímos pinos digitais e analógicas, as quais podem agir como atuadores ou sensores. Iremos, portanto, diferenciar estes dois conceitos.

3.1.1 Saídas e entradas digitais.

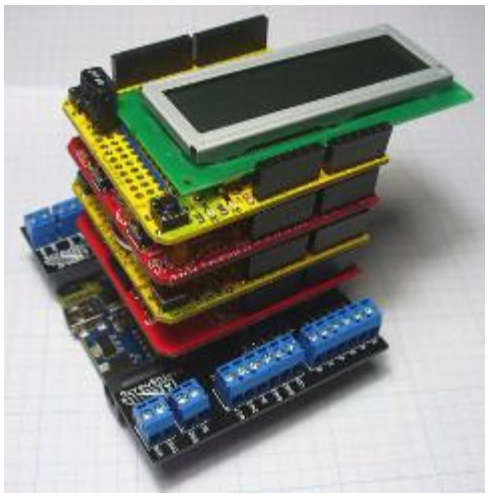
Os pinos digitais podem assumir apenas os valores de 0 ou 1, ligado ou desligado. Com isto podemos por exemplo ligar um led(OUTPUT) ou determinar se um pino está recebendo tensão(INPUT).

3.1.2 Saídas e entradas analógicas.

Os pinos analógicos possuem a possibilidade de enviar dados(OUTPUT) em PWM, que será discutido em breve, desta forma, podemos fazer a tensão que estamos enviando variar. Como também pode receber dados(INPUT) com uma certa sensibilidade, função que será melhor esclarecida em analogRead() pagina 20

3.2 MÓDULOS E SHIELDS.

Figura 4 - Muitos módulos conectados ao Arduino.



Os módulos e Shields do Arduino se tratam de dispositivos de hardware de forma que podemos aumentar suas funções originais, como por exemplo, se queremos controlar motores com Arduino, conectar à internet via WI-FI, conectar a um bluetooth ou receber e enviar dados de GPS, precisaremos de hardwares específicos para estas funções que precisam ser conectados ao Arduino, nossos módulos e Shields podem adaptar estas funções. Os shields levam vantagem sobre os módulos pelo fato de serem hardwares com funções completas, porém, possuem a desvantagem de funcionarem de forma acoplada ao Arduino, conectando todos os pinos os fazendo servir apenas nos modelos de pinagem parecidas, já os módulos são separados e possuem maior flexibilidade de montagem.

4 SOFTWARE ARDUINO.

Na sessão anterior vimos um pouco a respeito do potencial do Arduino, portanto o próximo passo é pensar em como podemos programa-los para executar as tarefas que desejamos, mas, para isto, precisamos inicialmente de um ambiente integrado para desenvolver nosso código, chamado de IDE em inglês.

4.1 IDE ARDUINO

A IDE do Arduino é disponibilizada pelo site Arduino.cc pela empresa Arduino e se trata de um programa onde podemos escrever os códigos de programação (Sketch) e os compilar para o microcontrolador com auxílio do Arduino. Nosso ambiente funciona da seguinte maneira, com nossa placa conectada ao computador pela porta USB, escrevemos nossos códigos no IDE do Arduino, usando uma linguagem baseada em processing, após fazemos o upload deste código para placa Arduino através do botão upload, ao o código será automaticamente traduzido pela IDE Arduino para uma linguagem baixo nível a qual será compilada para o microcontrolador através da porta USB.

Figura 5- IDE Arduino

4.1.1 Toolbar

VERIFY – Verifica se o código Arduino apresenta algum error,
UPLOAD – Traduz, compila e grava o código no microcontrolador.
NEW – Cria uma nova página vazia onde podemos escrever nosso código
OPEN – Abre um exemplo de uma lista do sketchbooks
SAVE – Salva seu código atual
SERIAL MONITOR – Abre o monitor serial

4.1.2 Sketch

Espaço destinado para escrever nossos códigos.

4.1.3 FEEDBACK

Local onde recebemos as mensagens de verificação e compilação do código.

5 PROGRAMAÇÃO NO AMBIENTE ARDUINO.

5.1 ALGORITMO.

Apesar do computador possuir grande capacidade de processamento de informações, o mesmo não apresenta uma "inteligência", os passos que o computador deverá seguir para alcançar determinado objetivo precisam ser pré-estabelecidos por um programador de forma que estas instruções sejam claras e precisas, estas instruções tem o nome de algoritmo e podemos exemplificar como um algoritmo, uma receita de bolo, onde este bolo deverá ser feito por alguém que nunca havia cozinhado na vida, consegue imaginar como esse algoritmo seria?

5.2 ESTRUTURA BASE.

Ao programarmos no Arduino, precisamos seguir sempre um padrão de estrutura a qual está representada e comentada a baixo.

Estrutura base.

```
/*
Estrutura básica; é sempre necessário que nosso setup e
loop estejam presentes no código, mesmo que vazios.
Nesta área inicial do código, antes de compilar, podemos
definir nossas variáveis.
*/

void setup(){
/* Nosso programa ao iniciar seja sendo ligado ou resetado,
ira começa pelo setup, rodará uma vez e irá ao loop.
Geralmente nesta etapa configuramos a maneira com a qual
nossos pinos irão se comportar, INPUT|OUTPUT (os quais em
breve serão esclarecidos) assim como podemos iniciar o
monitor serial.
*/
}

void loop(){
// Nosso loop contém a parte do código que sempre irá se
repetir. Geralmente a usamos para
// acionar ou desativar um dispositivo, realizar a leitura
de um sensor, etc... }
```

5.3 VARIÁVEIS E CONSTANTES.

Uma declaração de variável é definida pela alocação de uma certa informação que tem nome, valor e tipo, em um local da memória.1 O Arduino por padrão acompanha algumas variáveis pré-estabelecidas, as quais não podemos usar como HIGH, LOW, INPUT, OUTPUT, pinMode, digitalWrite, entre outras.

Tabela 1 - Variáveis principais

Variáveis principais		
Tipo	Informações	Exemplo
Char	Utilizado para armazenar 1 caractere e ocupa 1 byte.	`char meuCaracter = 'a';`
Byte	Armazena números inteiros entre 0 a 255, ocupa 1 byte.	`byte umNumero = 223;`
Int	Armazena números inteiros entre 2^{15} e -2^{15}, ocupam 2 bytes.	`int numUm = 1280, numDois;` `// iniciamos uma variável tipo inteira de nome numUm e valor 1280, e uma segunda de mesmo tipo, com nome numDois e sem valor definido.`
Long	Armazena números inteiros entre 2^{31} e -2^{31}, ocupam 4 bytes.	
Float	Armazena números decimais entre $\pm 3.4 \times 10^{38}$, ocupando 4 bytes.	`float decimal = 52.41;`
Double	Armazena números decimais ocupando 8 bytes.	

Boolean	Armazena 1 byte de valor, 0 ou 1,	`boolean falso = false;`
		`// VALOR 0`
		`boolean verdade = true`
		`// VALOR 1`

Constante	Cria uma variável apenas de leitura, imutável.	`#define a = 3` `const float pi = 3.14;`

[1] Traduzido de https://www.arduino.cc/en/Tutorial/Variables

Com ciência de nossas variáveis, podemos realizar algumas manipulações com elas como atribuições de valores, realizar operações aritméticas e logicas.

5.4 ATRIBUIÇÕES

Símbolo		Syntax
=, Atribuição	Podemos através deste comando atribuir valores a uma determinada variável	`int a = 20;` `int b = 14;`
+, Adição		`int c = a+b;` `//Teremos 34 como resposta.`

-, *, /	Subtração, multiplicação e divisão	`int c = a-b;`
		`int c = a+b;`
		`int c = a*b;`
		`int c = a/b;`

++,--	Incrementa ou diminui o valor de uma variável.	`x++; // incrementa x em 1 e retorna o antigo valor de x`
		`++x; // incrementa x em 1 e retorna o novo valor de x`
		`x = 2;`
		`y = ++x; // x == 3, y == 3`
		`y = x++; // x == 2, y == 3`

+=, -=, *=, %=	Permite fazer operações matemáticas de forma abreviada.	`x = 2;`
		`x += 2; // x = x+2`
		`x *= 2; // x = x*2`

5.5 COMPARAÇÃO.

ESTES ELEMENTOS GERALMENTE SÃO UTILIZADOS JUNTO AS ESTRUTURAS DE CONTROLE.

OPERADOR	EXEMPLO	CONDIÇÕES
==, IGUAL A	12 == 13	FALSO
!=, NÃO É IGUAL A	12 != 13	VERDADEIRO
<, MENOR QUE	12 < 13	VERDADEIRO
>, MAIOR QUE	12 > 12	FALSO
<= MENOR OU IGUAL A	12 <= 12	VERDADEIRO
>= MAIOR OU IGUAL A	14 >= 90	FALSO
&&, E	(A > 2 && B > 3)	VERDADEIRO
||, OU	(A > 16 || B > 16)	VERDADEIRO
!, NÃO / CONTRÁRIO.	!(A>B) VERDADE SE O OPERADOR É FALSO.	FALSO

5.6 CONTROLES DE ESTRUTURAS

5.6.1 IF -> (Se, Caso)

Usado em conjunto com os operadores de comparação, testa se certa condição foi atingida, para iniciar as instruções.
Exemplo:

`if (condição) {`
`    INSTRUÇÕES...;`
`}`

5.6.2 IF..ELSE (Se -> (IF)... do contrário -> (ELSE))

IF...ELSE, garante um maior controle sobre os fluxos de comandos de determinado código em comparação com o IF.

Exemplo:

```
if (condição) {
    Ação 1...;
    Ação 2...;
}
else {
    Ação 3...;
    Ação 4...;
}
```

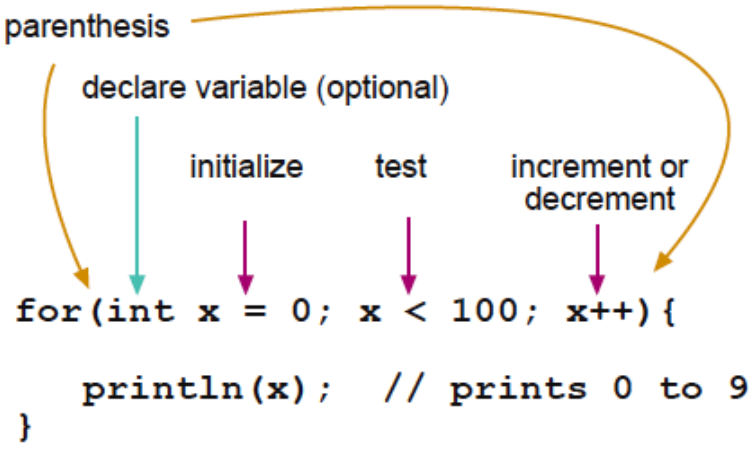
5.6.3 SWITCH CASE (Seleção de casos)

Assim como a estrutura de IF, o switch...case controla o fluxo do programa, porem de forma mais especifica, onde se listam os possíveis casos para determinada variável, quando desejado, usamos um break (que será discutido na página 17) para sair do loop.

Exemplo:

```
switch(variável){
   case 1:
      // faz alguma coisa quando nossa variável tem valor 1
     break;
   case 2:
      // faz alguma coisa quando nossa variável tem valor 2
     break;
   default:
      // default(Padrão) é opcional, é usada para ser executada no caso em que nenhum dos casos sejam validos.
     break;
```

5.6.4 FOR (PARA)

O ciclo for geralmente é utilizado para repetir um certo bloco fechado de instruções, usando um contador e incrementador afim de parar com o loop.
Exemplo:
Para um x = 0 até x < 100, execute o comando println(x) e mude o valor de x para x = x+1 

5.6.5 WHILE (ENQUANTO)

O while faz um loop infinito enquanto as condições no interior dos parênteses seja verdadeira(TRUE).
Exemplo:

`while(condição){`
`// Instruções... ;`
`}`

5.6.6 DO..WHILE (FAÇA ENQUANTO)

Funciona de forma na mesma lógica do while, com a única diferença de que a condição while é testada ao final do código.
Exemplo:

`do {`
`Instruções....~;`
`} while(condição);`

5.6.7 BREAK (PAUSA)

Break é usado para **sair** das condições de do, for, while loop ou da condição de switch, ignorando as condições do loop.
Exemplo:

`for (x= 0; x < 255 ; x++) {`
`  instrução 1... ;`
`  instrução 2...`
`  if (condição 02){ // sai do loop for.`
`  break;`
`  }`
`}`

5.6.8 CONTINUE

A condição de continue funciona de forma análoga ao break, porém, ela funciona pulando algumas condições do loop sem sair da função.
Exemplo:

`for (x= 0; x < 255 ; x++) {`
`  instrução 1... ;`
`  instrução 2... ;`
`  if (condição 02){ // Pode pular alguns valores de x.`
`  continue;`
`  } }`

5.7 FUNÇÕES

5.7.1 O que é uma função.

As funções são grandes aliadas do programador, elas são sub-rotinas e geralmente são usadas para não repetirmos uma mesma operação várias vezes, deixar nosso código mais compacto, além de que se tivermos um código muito grande, caso o mesmo apresente algum erro, será mais fácil a consertar.

Como motivação, imagine que tenhamos no Arduino a leitura de alguns sensores e que precisamos somar seus valores a todo momento, como veremos a seguir, criar essa sub-rotina será um trabalho bem simplificado.

5.7.2 Como criamos funções.

Uma nova função deverá ser escrita abaixo da função loop e podemos cria-la da seguinte forma, o primeiro passo é definir um valor que a mesma irá retornar, o qual pode ser um inteiro (int), uma condição verdadeira ou falsa (boolean) e se estendendo para qualquer variável de nossa tabela de variáveis, além disto, podemos ter, assim

como nossas funções setup() e loop() um valor de retorno vazio (void) que faz com que nossa função retorne nada. Além disto, podemos adicionar parâmetros a nossa função, os quais servem para enviar dados a nossa função chamada, como a e b no exemplo da função chamada soma() que será definida abaixo.

Exemplo:

```
// Exemplo de uso de uma função que soma dois valores.
void setup(){
}

void loop(){
  int x = soma(2,3);
}
int soma(int a, int b){
  return a + b;
}
```

5.7.3 FUNÇÕES EXISTENTES.

O Arduino já vem integrado com algumas funções importantes para seu funcionamento, a seguir iremos exemplificar algumas dessas funções.

5.7.3.1 Comunicação Serial.

5.7.3.1.1 Serial.begin(velocidade)

Comando usado para iniciar uma comunicação serial entre a placa Arduino e o computador ou outros dispositivos, através do cabo USB ou pinos RX e TX

Exemplo:

```
void setup(){
     Serial.begin(9600); // inicia comunicação de dados via porta serial com velocidade de 9600 bits por segundo.
}
```

5.7.3.1.2 Serial.print() e Serial.println()

Estes comandos apenas são usados após termos iniciado a comunicação serial e servem para imprimir dados pela porta serial. O println() ao final de seu envio quebra a linha para começar o novo texto em uma linha diferente, o que não acontece com a função print().

Exemplo:

```
void setup(){
     Serial.begin(9600); // inicia comunicação Serial.
} void setup(){
     Serial.println("Abacate"); // Imprime no Monitor Serial.
}
```

5.7.3.2 Input/Output digital

5.7.3.2.1 pinMode()

Sempre que precisamos usar um pino do Arduino, precisamos informar ao microcontrolador e também definir como será o tipo de sinal que teremos nele, entrada(INPUT) ou saída(OUTPUT), esta função geralmente é feita no início do programa, no interior da estrutura básica setup.

```
void setup(){
  pinMode(Numeração do pino, sinal);
}
```

5.7.3.2.2 digitalWrite()

Os pinos definidos como saída (output), podem receber os valores de HIGH(alto) ou LOW(baixo), o que os possibilita enviar através desse pino um sinal de 5v ou 0v

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

5.7.3.2.3 digitalRead()

Possibilita a leitura de sinal externo em um pino digital definido como INPUT na forma de HIGH ou LOW.

```
void setup(){
  Serial.begin(9600); // Inicia comunicação Serial.
  pinMode(2, INPUT);
}
void loop(){
  if(2 == HIGH){
Serial.println("O pino 2 está com tensão");
}
}
```

5.7.3.3 Input/Output analógico.

5.7.3.3.1 analogRead()

Possibilita a leitura de um pino analógico do Arduino definido como input, porém diferentemente do sinal digital, o sinal analógico possui uma certa sensibilidade definida pela quantidade de bits de nosso microcontroladores que reconhece os valores de tensão externos entre os valores de 0v a 5v, esta sensibilidade é definida pela equação 2^{bits}, no caso do atmega 328 com 10 bits, temos o valor 1024, com isto podemos receber valores inteiros entre 0 a 1023 com 5v de referência.

Figura 6 - Grafico PWM

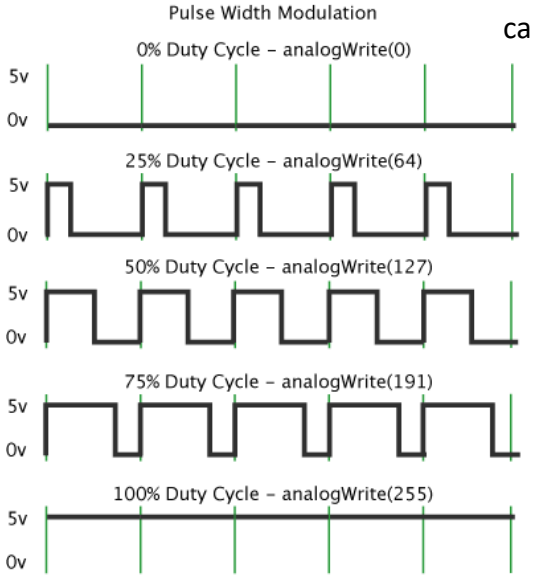
```
int analogPin = A0;      // temos a perna do meio de um potenciômetro conectado no pino A0. As outras extremidades estão conectadas ao ground e +5v
int val = 0;             // declaramos uma variável a ser lida, que possui um valor inicial igual a zero.

void setup()
{
  Serial.begin(9600);    //  Iniciamos a comunicação serial; neste caso ARDUINO -> PC, através do USB
}

void loop()
{
  val = analogRead(analogPin);     // Faz a leitura da tensão de referência no pino.
  Serial.println(val);                 // Imprime os valor de val.}
```

5.7.3.3.2 analogWrite()

Os pinos que apresentam um símbolo (~) na placa Arduino podem ser usadas como modulação de largura de pulso (PWM), em resumo, a placa Arduino é capaz de imprimir a configuração high e low em poucos microssegundos, desta forma, nosso valor de tensão pode ser compreendida como uma média do tempo que o pino está on/off. O valor de referência é 255 para 5v e 0 para 0v. (Arduino, 2017)



```
// Programa para variar o brilho do led de acordo com a tensão em um potenciômetro.

int ledPin = 9;      // LED conectado ao pino digital 9
int analogPin = 3;   // potenciômetro conectado ao pino analógico 3
int val = 0;    // variável para armazenar o valor lido
void setup(){
  pinMode(ledPin, OUTPUT);   // pré-determina o pino como saída
}

void loop() {
  val = analogRead(analogPin);    // lê o pino de entrada
  analogWrite(ledPin, val / 4);   // os valores do analogRead variam de 0 a 1023, os valores do analogWrite variam de 0 a 255 }
```

5.7.3.4 Contagem de tempo

5.7.3.4.1 delay(milissegundos)

Permite pausar o programa por uma quantidade de tempo especifica dada em milissegundos, (1000 milissegundos = 1 segundo)
Exemplo:
`delay(1000);  // Espere 1 segundo.`

5.7.3.4.2 delayMicroseconds(µs)

Funciona de forma análoga ao delay porém, com um intervalo de tempo em microssegundos(1/1000 milissegundos) ou (1/10^6 segundos). Segundo os fabricantes do Arduino, o mesmo funciona bem para intervalos superiores a 3 microssegundos.
Exemplo:
`delayMicroseconds(1000); // espere 1 milissegundo.`

5.7.3.5 Geradores de números aleatórios

5.7.3.5.1 random e randomSeed()

As funções random e randomSeed, possibilitam gerar números pseudo-aleatórios preferencialmente através do ruído de uma porta analógica que não esteja conectada.

Exemplo:

```
long randNumber;
void setup() {
   Serial.begin(9600);  // Iniciado para que visualizado os números aleatórios no monitor serial.
randomSeed(analogRead(0));  //  Um  pino  0  desconectado, poderá gerar sinais analógicos aleatórios detectando ruídos através da função randomSeed(), este sinal irá ser enviado para a função random, afim de a deixar mais aleatória.
}
 void loop() {
  randNumber = random(300);  // sorteará um número entre 0 e 299.
  Serial.println(randNumber);     // irá imprimir o número sorteado no monitor serial.
  randNumber = random(10, 20);  // sorteará um número entre 10 e 19
  Serial.println(randNumber);
  delay(50);
}
```

6 BIBLIOGRAFIA

Arduino. (02 de 09 de 2017). *Arduino.cc*. Fonte: Arduino PWM: https://www.arduino.cc/en/Tutorial/PWM

FilipeFlop. (02 de 09 de 2017). *filipeflop.com*. Fonte: FilipeFlop: filipeflop.com/blog/open-hardware-livre

Figura 3 - Sensores x Atuadores (02 de 09 de 2017)
Fonte: http://www.c2o.pro.br/hackaguas/ar01s02.html

7 LISTA DE FIGURAS