Conceitos Básicos de Programação

Iremos compreender os fundamentos da linguagem utilizada para programar o Arduino. Caso você já conheça alguma linguagem de programação, basta fazer uma "adaptação de linguagem", ou seja, procurar se adaptar a um novo ambiente, pois os fundamentos principais são praticamente os mesmos para a maioria das linguagens.

Variáveis e Constantes

Declaramos como constante o objeto que o seu valor não precisa ser modificado durante todo o processamento. Assim, do início ao fim da execução seu valor permanece inalterado.

Existem duas formas de declarar uma constante:

- Utilizando a palavra reservada CONST. Exemplo:
  const int LED = 10;

- Utilizando a palavra reservada DEFINE. Exemplo:
  #define PINO_A 10

Em todas as linguagens de programação, existem as palavras reservadas. Em C++, algumas dessas palavras são:

- TRUE – Indica um valor lógico;
- FALSE – Indica um valor lógico;
- HIGH – Indica que uma porta está ativada, ou seja, com 5V;
- LOW – Indica que uma porta está desativada, ou seja, com 0V;
- INPUT – Indica que uma porta será utilizada como entrada de dados;
- OUTPUT – Indica que uma porta será utilizada como saída de dados.

As variáveis são objetos que armazenam (posição de memória) dados e seu conteúdo pode variar durante o processamento. Uma variável só pode armazenar um único valor por vez.

Sua declaração é formada por um ou mais caracteres, em que o primeiro caractere é uma letra.

Não é permitido nem para uma constante ou variável, identificadores com caracteres especiais (*, %, @, etc), palavras reservadas ou espaços em branco. Se preferir, poderá utilizar o underline (“_”).

Exemplo:
executar_Led = false;

Tipos de Variáveis

A seguir os tipos de variáveis existentes:

- **bool** - Os valores possíveis são *true* (1) e *false* (0). Ocupa um byte de memória.
  exemplo: bool executar = false;

- **char** - Pode ser uma letra ou um número. A faixa de valores válidos é de -128 a 127. Ocupa um byte de memória.
  exemplo: char caractereA = 65; ou char led = ‘L’;

- **unsigned char** - O mesmo que o char, porém a faixa de valores válidos é de 0 a 255. Ocupa um byte de memória.
  exemplo: unsigned char novoChar = 240;

- **byte** - A faixa de valores é de 0 a 255. Ocupa 1 byte de memória.
  exemplo: byte b = B10010; // B10010 = 18 decimal;

- **Int** - Armazena números inteiros e ocupa 16 bits de memória (2 bytes). A faixa de valores é de -32.768 a 32.767.
  exemplo: int ledPino = 10;

- **unsigned int** - O mesmo que o int, porém a faixa de valores válidos é de 0 a 65.535. Ocupa 2 bytes de memória.
  exemplo: unsigned ledPino = 10;

- **word** - O mesmo que um unsigned int.

exemplo: word x = 1000;

- **long** - Armazena números de até 32 bits (4 bytes). A faixa de valores é de -2.147.483.648 até 2.147.483.647.
  exemplo: long velocidade = 186000;

- **unsigned long** - O mesmo que o long, porém a faixa de valores é de 0 até 4.294.967.295.
  exemplo: unsigned long time;

- **short** - Armazena número de até 16 bits (2 bytes). A faixa de valores é de -32.768 até 32.767.
  exemplo: short ledPino = 10;

- **float** - Armazena valores de ponto flutuante (com vírgula) e ocupa 32 bits (4 bytes) de memória. A faixa de valores é de -3.4028235E+38 até 3.4028235E+38.
  exemplo: float sensorCalibrado = 1.117;

- **double** - O mesmo que o float.
  exemplo: double sensorCalibrado = 1.117;

- **String** – Ocupa 1 Byte + x. Utilizado para coleção de variáveis.
- **Array** - Ocupa 1 Byte + x. Utilizado para coleção de variáveis.

Declaração de Variáveis

A atribuição de valores para as variáveis e constantes são realizados com o uso do operador "**=**".
Exemplos:

```
int efeito = 3;
int contraste = 0;
boolean aumenta = true;
```

Observações:

- O Operador de atribuição não é utilizado com o comando #define;
- A linguagem de programação do Arduino é Case Sensitive, diferenciando letras maiúsculas de minúsculas. Portanto a declaração **int efeito** é diferente de **int Efeito**.

Vetores e Matrizes

Uma variável pode armazenar muitos valores ao longo da execução do programa, porém não ao mesmo tempo. Entretanto, existem alguns tipos de variáveis que podem armazenar mais de um valor, são conhecidas como "variáveis vetoriais". Na programação do Arduino é possível utilizar dois tipos de "variáveis vetoriais": Vetores e Matrizes.

Vetores

Para declarar um vetor é necessário definir o tipo e o tamanho (quantidade de itens). Um vetor é unidimensional, ou seja, uma única dimensão.

Exemplo: Criar um vetor de tamanho 3 (três) do tipo inteiro.

```
int pinLed[3] = {2,3,4};
```

Para atribuir um valor a uma determinada posição do vetor, basta usar o índice, ou seja, a posição onde o valor será armazenado no vetor, lembrando que a primeira posição do vetor é a posição 0 (zero).

Exemplo: Atribuir o valor 2 (dois) para a posição 0 (zero) do vetor:
pinLed[0] = 2;

Para acessar um valor em uma determinada posição do vetor, basta indicar a sua localização (índice).

Exemplo:
pinMode(pinLed[0], OUTPUT);

Vetores de Strings

Quando for trabalhar com vários textos, podemos utilizar um vetor de strings.
O asterisco indica que se trata de um vetor de string.
Exemplo:

```
char* myStrings[]={"string 1", "string 2", "string 3","string 4"};
void setup(){
Serial.begin(9600);
}
void loop(){
for (int i = 0; i <= 3; i++){
Serial.println(myStrings[i]);
delay(500);
}
}
```

Matrizes

Uma matriz é similar a um vetor, porém pode ser formada por duas ou mais dimensões, ou seja, é bidimensional. Este elemento possui um determinado número de linhas e de colunas.
Exemplo: Criar uma matriz 4x4 de char.

```
char teclas[4][4] = {{'1','2','3','A'},
                     {'4','5','6','B'},
                     {'7','8','9','C'},
                     {'*','0','#','D'}};
```

Para atribuir um valor a uma determinada posição da matriz, basta usar o índice (linha e coluna).

Exemplo: Atribuir o valor A para a posição 1 (linha), 4 (coluna) da matriz teclas:
teclas [1] [4] = 'A';

Para acessar um determinado valor em uma posição da matriz, basta usar o índice (linha e coluna).

Exemplo: Ativar a porta 0 (linha), 0 (coluna) da matriz teclas:
digitalWrite(teclas[0][0],HIGH);

Operadores

Além dos dados existentes em um processamento de um programa, temos também os operadores que são elementos que indicam a realização de uma operação sobre esses tipos de dados, gerando um resultado. O termo operador é o elemento que indica a realização da operação.

Operadores Aritméticos

Os operadores aritméticos são utilizados nos cálculos matemáticos, são os responsáveis pelas operações matemáticas realizadas no computador. No próximo quadro (Quadro MD1), iremos mostrar como utilizar os operadores e o grau de prioridade em um cálculo matemático.

Quadro MD1 - Operadores Aritméticos

Operador	Operação	Descrição	Prioridade
+	+x	Manutenção do sinal	1
-	- x	Inversão de sinal	1
++	x++	Incrementa	2
--	x--	Decrementa	2
*	x * y	Multiplicação de x por y	3
/	x / y	Divisão de x por y	3
%	%x	Módulo (resto da divisão inteira)	3
+	x + y	Adição de x com y	4
-	x - y	Subtração de y de x	4
*=,/=,+=,-=, %=		Tipos de Atribuição *= Multiplicação com atribuição /= Divisão com atribuição += Adição com atribuição -= Subtração com atribuição %= Módulo com atribuição	5

Expandida	Reduzida
x = x + y	x += y
x = x – y	x -= y
x = x * y	x *= y
x = x / y	x /= y
x = x % y	x %= y

Operadores Relacionais

Os operadores relacionais (Quadro MD2) permitem que sejam feitas comparações entre dois elementos, tendo como resultado um valor lógico (Verdadeiro ou Falso). Através do resultado da comparação, é possível determinar qual caminho um processamento vai seguir. Os operadores relacionais possuem o mesmo grau de prioridade entre si.

Quadro MD2 - Operadores Relacionais

Operador	Descrição
==	Igual a *true* se o argumento da esquerda possuir o mesmo valor do da direita
!=	Diferente Oposto da Igualdade
>,<	Maior que, Menor que *true* se o argumento da esquerda for maior ou menor que o da direita
>=, <=	Maior ou igual a, menor ou igual a *true* se > ou == for *true*, ou se < ou == for *true*

Operadores Lógicos

Da mesma forma que os operadores relacionais (Quadro MD3), os operadores lógicos

retornam como resultado um dos dois valores lógicos: verdadeiro ou falso.
Os operadores lógicos podem ser: conjunção (AND); disjunção (OR); e negação (NOT).

Quadro MD3 - Operadores Lógicos

Operador	Descrição
&&	AND (E) *true* se ambos os argumentos da esquerda e direita forem *true*
||	OR (OU) *true* se um dos argumentos da esquerda ou direita forem *true*
!	NOT (NÃO) *true* se seu argumento for false ou false se seu argumento for *true*

Operadores Lógicos de conjunção

A conjunção é uma operação lógica que relaciona dois valores lógicos (Quadro MD4) através do operador **&&**.
Para que o resultado lógico de saída seja verdadeiro, é necessário que os valores lógicos de entrada sejam verdadeiros.

Quadro MD4 - Operador Lógico de Conjunção

Condição 1	Condição 2	Condição 1 && Condição 2
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

Exemplo:

```
if (x==2 && y==10) {.....}
```

Essa expressão retornará verdadeiro se ambos os operandos forem verdadeiros, ou seja, **x deverá ser igual a 2 e y deverá ser igual 10**, caso contrário retornará falso.

Operadores Lógicos de Disjunção

A disjunção é uma operação lógica que relaciona dois valores lógicos (Quadro MD5) através do operador **||**.
Para que o resultado lógico de saída seja verdadeiro, é necessário que pelo menos um dos valores lógicos de entrada seja verdadeiro.

Quadro MD5 - Operador Lógico de Disjunção

Condição 1	Condição 2	Condição 1 || Condição 2
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Exemplo:

```
if (x==2 || y==10) {.....}
```

Essa expressão retornará verdadeiro se um ou mais operandos forem verdadeiros, ou seja, **x deverá ser igual a 2 ou y deverá ser igual 10**, caso contrário retornará falso.

Operadores Lógicos de Negação

Negação é uma operação lógica que gera como saída um valor lógico inverso ao valor lógico de entrada (Quadro MD6). É utilizado o operador **!**.

Quadro MD6 - Operador Lógico de Negação

Condição	! Condição
Verdadeiro	Falso
Falso	Verdadeiro

Exemplo:

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

Essa expressão retornará verdadeiro se o operando for falso, ou seja, **x deverá ser igual a 0**, caso contrário retornará falso.

As condições podem aparecer “aninhadas”.
Exemplo:

```
if (x==2 || y==10 && k == 20) {.....}
```

Grau de Prioridades

Os operadores lógicos possuem o seguinte nível de prioridade entre os operadores que devem ser obedecidas (Quadro MD7).

Quadro MD7 - Grau de Prioridade entre Operadores Lógicos

Operador	Operação	Prioridade
!	Negação	1
&&	Conjunção	2
||	Disjunção	3

Se for necessário alterar o grau de prioridade em um determinado processamento lógico, deve-se utilizar os parênteses. Lembrando que os cálculos acontecem da esquerda para a direita.

Exemplo:

```
if (  x==2 && (y==10 || k == 20)  ) {.....}
```

é diferente de:

```
if (  x==2 && y==10 || k == 20  ) {.....}
```

Funções

No Arduino é possível controlar o intervalo de execução dos componentes através das Funções. Link: https://www.arduino.cc/reference/pt/#functions.

Funções de Temporização

delay()
Provoca um pausa na execução do programa até iniciar a próxima instrução. Em geral, utilizamos o valor 1000 milissegundos (1 segundo).
Exemplo: delay(1000).

delayMicroseconds(us)
Provoca uma pausa na execução do programa por uma quantidade especificada de tempo

(em microssegundos). Há mil microssegundos em um milissegundo, e um milhão de microssegundos em um segundo.

millis()
Retorna o número de milissegundos desde que a placa começou a executar o programa atual. Este número retorna a zero após aproximadamente 50 dias.

micros()
Retorna o tempo em microssegundos que a placa está executando o programa. Este número retorna a zero em aproximadamente 70 minutos.

Funções Matemáticas

min(x, y)
Retorna o menor valor entre dois valores.
Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("resultado: ");
  int a = 50;
  int b = 5;
  Serial.println(min(a,b));

}
```

Resultado: 5

max(x, y)
Retorna o maior valor entre dois valores.

abs(x)
Calcula o valor absoluto de um número.
Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("resultado: ");
  int a = -10;
  Serial.println(abs(a));

}
```

Resultado: 10

map(valor, menorLimiteAtual, maiorLimiteAtual, menorAlvo, maiorAlvo)
Remapeia um número inteiro de um intervalo para outro dentro de uma faixa específica.
Parâmetros:
valor: o número a ser mapeado.
menorLImiteAtual: menor limite do intervalo atual do valor.
maiorLimiteAtual: maior limite do intervalo atual do valor.

menorAlvo: menor limite do intervalo alvo.

maiorAlvo: maior limite do intervalo alvo.

Exemplo:

```
int valorLido;
int P_analoP = A3;
int led = 10;
unsigned int pwm;
void setup() {
Serial.begin(9600); //Inicia porta serial
pinMode(led,OUTPUT);
}

void loop() {
valorLido = analogRead(P_analoP);
Serial.println(valorLido);
delay(1000); // Espera um segundo
pwm = map(valorLido,0,1023,0,255);
analogWrite(led,pwm);

}
```

pow(base, expoente)

Retorna o valor de um número (base) elevado a uma potência(expoente).
Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("FUNCAO POW: ");
  Serial.println(pow(5,3));
  delay(5000);

}
```

Resultado: 125.00

sq(x)

Retorna o quadrado de um número (x).
Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("FUNCAO SQ: ");
  Serial.println(sq(4));
  delay(5000);

}
```

Resultado: 16.00

sqrt(x)

Retorna a raiz quadrada de um número (x).

Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("FUNCAO SQRT: ");
  Serial.println(sqrt(16));
  delay(5000);

}
```

Resultado: 4.00

constrain(x, a, b)

Restringe um número para estar dentro de um intervalo.
Parâmetros:
x: O número que deseja restringir;
a: menor valor do intervalo;
b: maior valor do intervalo.

Retorno:
x: se x estiver entre a e b;
a: se x é menor que a;
b: se x é maior que b.

Exemplo:
sensorValor = constrain(sensorValor, 10, 150);
Obs.: Restringe o valor do sensor entre os valores 10 e 150

Funções Caracteres

isAlpha(caractere)

Analisa de o parâmetro é um caractere (uma letra). Parâmetro deverá ser do tipo char e retorna verdadeiro ou falso.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("FUNCAO isAlpha: ");
  if (isAlpha('A')) {
    Serial.print("É um caractere");
  }
  else {
    Serial.print("Não é um caractere");
  }

  delay(5000);

}
```

isAlphaNumeric(parâmetro)

Analisa se o parâmetro é uma letra ou número. Retorna true (verdadeiro) se o parâmetro contém ou uma letra ou um número.

Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("FUNCAO isAlphaNumeric: ");
  if (isAlphaNumeric('a')) {
    Serial.println("É uma letra ou número");
  }
  else {
    Serial.println("Não é uma letra ou número");
  }
  delay(5000);
}
```

Funções Aleatórios

random(max)
Gera número pseudoaleatórios.
Existem duas formas de utilização: random(max) ou random(min, max).
Onde: random(max), max é o maior valor a ser gerado (-1); random(min, max), min é o menor valor do limite e max é o maior valor do limite (-1);
Exemplo:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("FUNCAO random: ");
  Serial.println(random(100));
  Serial.println(random(10,20));
  delay(5000);
}
```

<<<Pesquisar: funções de Conversão>>>

Comentários

Geralmente é necessário comentar as linhas dos códigos de um programa. Utilizaremos duas maneiras:

- Comentário de linha: //.

Exemplo:

```
if (liga == 'H')
// ativar o pino_a
digitalWrite(PINO_A, HIGH);
else
digitalWrite(PINO_A, LOW);
}
```

- Comentário de bloco: /* */, utilizado para comentar mais de uma linha.

Exemplo:

```
if (liga == 'H')
/* ativar o pino_a
 * desativa o pino_a
 */
digitalWrite(PINO_A, HIGH);
else
digitalWrite(PINO_A, LOW);
```

Voltar ao Topo

CONTATOS

e-mail: euderfs@gmail.com