

Programação em Arduino

Prof. Daniel R. Cassar and James Moraes de Almeida

Capacitação em Arduino

Contents

2	Módulo 2	1
2.1	Portas digitais de <code>INPUT_PULLUP</code> e <code>INPUT_PULLDOWN</code>	1
2.2	Definindo funções no Arduino	1
2.3	Gerando números aleatórios com a função <code>random</code>	5

2 Módulo 2

2.1 Portas digitais de `INPUT_PULLUP` e `INPUT_PULLDOWN`

Além de `INPUT` e `OUTPUT`, existem mais modos de operação das portas digitais do Arduino: o modo `INPUT_PULLUP` e o `INPUT_PULLDOWN`. O `INPUT_PULLUP` opera sempre enviando sinal na porta selecionada, quando um circuito for fechado, a porta detectará e retornará o modo `LOW`, se o circuito não estiver fechado, ela ficará no modo `HIGH`. Já o modo `INPUT_PULLDOWN` funcionará da maneira contrária, quando o circuito estiver fechado a porta fica no modo `HIGH` e quando estiver aberto fica no modo `LOW`.

Esses modos são muito importantes para botões! Com eles você consegue receber os sinais dos botões, sabendo se eles estão pressionados ou não. Para isso, conecte um dos terminais do botão no terra do Arduino (`GND`) e outro em uma porta digital que esteja no modo `INPUT_PULLUP`, assim, quando o botão for pressionado você terá um sinal `LOW` na porta e quando não estiver pressionado, a porta terá um sinal `HIGH`.

2.2 Definindo funções no Arduino

É comum em programação o conceito de código reutilizável: trata-se de um conjunto de comandos que, após definidos, podem ser utilizados em out-

ros locais. Nós já vimos algumas funções embutidas do Arduino: `pinMode`, `digitalWrite`, `digitalRead` e `delay` são alguns exemplos. Estas funções já foram escritas para nós e podemos usar a hora que quisermos. Imagine se cada vez que você fosse configurar uma porta digital você tivesse que escrever todos os comandos da função `digitalWrite`... seria bastante contraproduativo!

Nós podemos definir nossas próprias funções. Na realidade, todo código de Arduino requer a definição de duas funções em particular, uma chamada `setup` e outra chamada `loop`. A sintaxe de definição destas funções nós já vimos no Módulo 1 e está repetida abaixo.

```
1 void setup() {
2     // aqui fica o bloco de configuração
3 }
4
5 void loop() {
6     /* aqui fica o
7        bloco de laço de repetição */
8 }
```

As funções `setup` e `loop` não recebem nenhum argumento¹ e também não retornam nenhuma informação. Nós observamos que estas funções não retornam nenhuma informação pela presença da palavra `void` antes do nome da função. Veja abaixo um exemplo da definição e uso de uma função que retorna um valor inteiro.

```
1 int armazena_numero;
2
3 void setup() {
4     armazena_numero = retorna_oito();
5 }
6
7 void loop() {
8 }
9
10 int retorna_oito() {
```

¹Alguns livros chamam argumentos de parâmetros. Aqui usaremos o termo argumento.

```
11     return 8;
12 }
```

Após rodar este programa, qual você acha que é o valor armazenado na variável `armazena_numero`? Vamos por partes:

1. A variável `armazena_numero` é definida na linha 1 como sendo uma variável inteira, mas sem nenhum valor atribuído a ela.
2. Na linha 4, dentro do bloco de configuração, a variável `armazena_numero` recebe o valor da função `retorna_oito`. Sabemos que `retorna_oito` é uma função pois funções são sempre chamadas com os argumentos dentro de parênteses. Neste caso, como a função tem zero argumentos, não temos nada dentro dos parênteses (mas mesmo assim precisamos escrever os parênteses!). Mas onde está a definição da função `retorna_oito`?
3. A função `retorna_oito` é definida na linha 10. Não tem nenhum problema definir funções após o `void setup` e o `void loop`, mesmo que estas funções sejam chamadas dentro destes blocos de código. Nós sabemos que essa é a definição de uma função e não de uma variável por conta da presença de chaves indicando um bloco de código (isso não existe na definição de variáveis!). Também sabemos que esta função retorna um valor inteiro ao ser executada por conta do `int` antes do nome da função. Se ela não retornasse nada teríamos um `void` no lugar do `int`.
4. Dentro da função `retorna_oito` nós temos um bloco de código delimitado por chaves. Este bloco tem apenas uma linha de código, mas poderia ter quantas fossem necessárias. Vimos acima que esta função tem um retorno; o comando para retornar um valor é o `return` (que significa retornar em inglês). Neste caso específico, a função retorna o valor 8.
5. Sabendo o que acontece quando a função `retorna_oito` é executada, agora podemos responder a pergunta: o valor armazenado na variável

`armazena_numero` ao final da execução do programa será 8.

A função `retorna_oito` definida acima não é muito interessante... ela apenas retorna o valor 8. Mas o importante neste momento é entender como funções com retorno são definidas. Para tornar funções mais interessantes, vamos ver no exemplo abaixo como definir funções que recebem argumentos.

```
1  int armazena_numero;
2
3  void setup() {
4      armazena_numero = multiplica_por_dois(10);
5      acende_led(LED_BUILTIN, 1000);
6  }
7
8  void loop() {
9  }
10
11 int multiplica_por_dois(int numero) {
12     return numero * 2;
13 }
14
15 void acende_led(int porta, int tempo_aceso_ms) {
16     digitalWrite(porta, HIGH);
17     delay(tempo_aceso_ms);
18     digitalWrite(porta, LOW);
19 }
```

Duas funções foram definidas no código acima: `multiplica_por_dois` na linha 11 e `acende_led` na linha 15. Observe que na definição delas temos um parêntese antes de abrir as chaves do bloco de código da função. É dentro deste parêntese que definimos os argumentos das funções, sempre indicando o tipo do argumento e o nome dele. No caso da função ter dois ou mais argumentos, estes são separados por vírgulas.

A função `multiplica_por_dois` recebe um argumento inteiro e retorna o valor da multiplicação deste argumento pelo número 2. Esta função é usada na linha 4 para atualizar o valor da variável `armazena_numero`, que a partir de agora armazena o valor 20.

A função `acende_led` recebe dois argumentos, ambos inteiros, e não retorna nenhum valor. Esta função envia sinal elétrico por uma porta do Arduino (argumento `porta`) por um certo tempo em milisegundos (argumento `tempo_aceso_ms`) e após isso para de enviar o sinal elétrico por esta porta. Esta função é usada na linha 5 do bloco de configuração para acender o LED embutido do Arduino por 1000 milisegundos.

2.3 Gerando números aleatórios com a função `random`

A função `random` do Arduino é usada para sortear números aleatórios. Na verdade, ela é usada para gerar números *pseudoaleatórios*. A realidade é que é bastante difícil sortear números de forma perfeitamente aleatória usando computadores e na grande maioria dos casos nós trabalhamos com um sorteador que finge ser perfeitamente aleatório mas efetivamente não o é (por isso são pseudoaleatórios). Esse fato costuma estar implícito em textos de programação. Então não ficamos sempre dizendo que são números pseudoaleatórios, mas sim avisamos quando for o caso dos números serem números aleatórios de fato.

A função `random` pode receber um ou dois argumentos. Se receber apenas um argumento, então ela irá sortear um número inteiro entre zero e o valor do argumento subtraído de 1. Se receber dois argumentos, então ela irá sortear um número inteiro entre o primeiro argumento e o segundo argumento subtraído de 1. Observação: se o valor da função `random` for armazenado em uma variável, ela deve ser do tipo `long`.

Associado ao processo de gerar números aleatórios existe o conceito da “semente aleatória”. Este conceito está intimamente relacionado ao fato de estarmos trabalhando com números pseudoaleatórios. Esta “semente aleatória” serve para dar o pontapé inicial na função `random` para ela iniciar seus sorteios. Toda a sequência de números sorteados depende desta semente! Se a semente aleatória for sempre o mesmo valor, então a sequência de números produzidos pela função `random` será sempre a mesma. Por conta disso, é interessante sempre escolher uma semente aleatória diferente. A função que configura a semente aleatória é a `randomSeed`. Ela recebe apenas um argumento que é justamente um número inteiro positivo representando a semente aleatória. A forma recomendada na documentação do Arduino para usar a função `randomSeed` é alimentando ela com o valor lido de uma porta analógica que não tenha nenhum componente eletrônico ligado (isto pois existe uma

variação inerente no sistema que altera essa leitura de forma imprevisível).

O código abaixo exemplifica tudo que foi apresentado acima.

```
1 long numero_aleatorio;
2 int porta_analogica_sem_nada_conectado = A0;
3
4 void setup() {
5     Serial.begin(9600);
6     randomSeed(analogRead(porta_analogica_sem_nada_conectado));
7 }
8
9 void loop() {
10     numero_aleatorio = random(0, 1001); // sorteia um número entre 0 e 1000
11     Serial.println(numero_aleatorio);
12     delay(100);
13 }
```