Aula 03 - SEEL 2019
Minicurso de Arduino
Funções, Temporizadores

Adriano Rodrigues

25 de outubro de 2019

- Funções;
- Declarações const e #define;
- Temporizadores;

Funções

Funções servem basicamente para descentralizar o código de tal forma que tarefas repetidas possam ser executadas à parte do código principal.

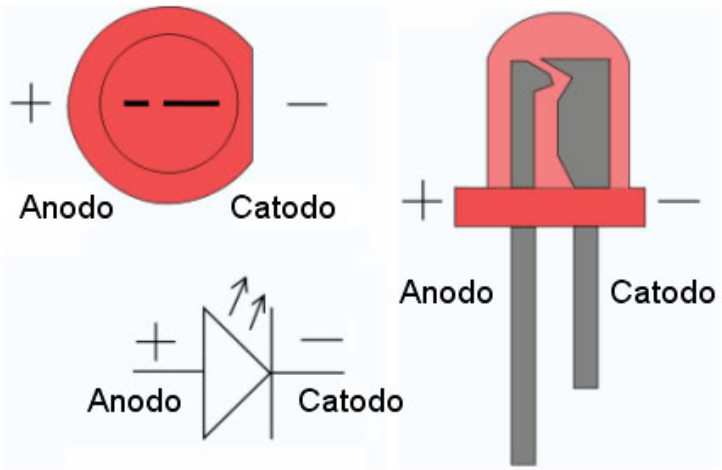
As mesmas podem ser dos tipos: **int**, **float**, **char**... Esses indicam qual tipo de variável será retornado pela função.

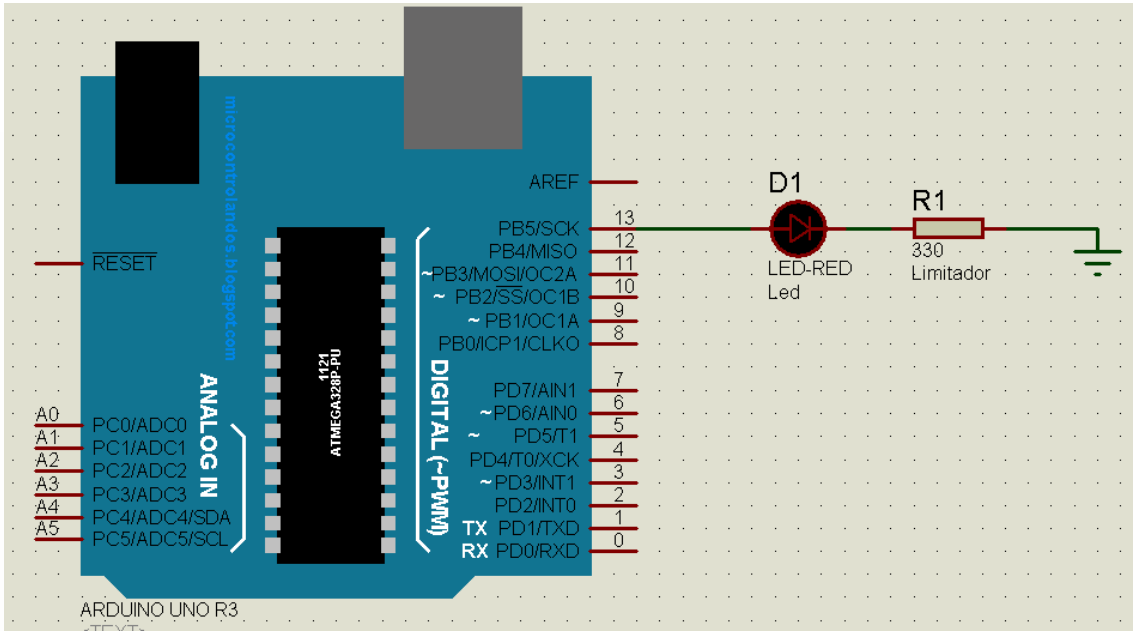
Se a função não retorna nenhum valor, então a mesma será do tipo **void**.

Exercício 1 - Funções

```
1  // Funcao de piscar 2 LEDs - Adriano Rodrigues.
2  #define LED0  13  // Maneira alternativa de declarar constantes.
3  #define LED1  12  // Nao consome memoria do Arduino.
4  boolean E[2]; // Vetor de estados - Variavel GLOBAL.
5  void setup() // Dispensa comentarios.
6  {
7      Serial.begin (9600); pinMode(LED0, OUTPUT); pinMode(LED1, OUTPUT);
8  }
9  void loop()
10 {
11     piscar(LED0, 0); // Pisca o LED zero.
12     piscar(LED1, 1); // Pisca o LEL um.
13     delay(500);
14 }
15
16 // Funcao dedicada a piscar LEDs.
17 void piscar(int LED, int i) // Variaveis LOCAIS.
18 {
19     E[i] = !E[i];              // Alterna o valor do estado.
20     digitalWrite(LED, E[i]); // Escreve no LED.
21 }
```

Montagem do LED





Funções

Armazenar o número da porta associada a um LED em uma variável do tipo inteiro é um grande desperdício de memória.

Existem diversas maneiras otimizadas de definirem-se essas constantes: por meio de **#define** ou por meio de **byte const**, por exemplo.

- #define LEDpin 13

  Nesse caso interface é a responsável por buscar todas as ocorrências da palavra *LEDpin* e subtituí-la pelo valor 13. Ou seja, quando o programador escreve *LEDpin*, o programa entenderá como 13.

- byte const LEDpin $= 13$;

  Dessa forma, você armazena o número 13 na memória do Arduino, porém em apenas 8-bits ao invés de 16-bits. Por ser constante, o compilador acusará erro caso o programador tente enganadamente modificar o seu valor.

Qual o melhor?

- #define LEDpin 13

  Não gasta memória, porém o número só pode ser acessado pelo compilador.

- byte const LEDpin = 13;

  Gasta memória, porém, caso seja necessário acessar esse valor (usando ponteiros, por exemplo), existirá um endereço o qual esse valor pode ser recuperado.

Nas aplicações mais básicas (como as nossas), utiliza-se normalmente o #define. As outras estratégias são melhores aproveitadas quando existe manipulação de valores armazenados nos registradores.

Tarefa 1 - Multitarefa com Períodos Diferentes

- Fazer dois LEDs piscarem. LED0 com período de 1.0 segundo, LED1 com período de 1.2 segundo.

Tarefa 1 - Multitarefa com Períodos Diferentes

- Fazer dois LEDs piscarem. LED0 com período de 1.0 segundo, LED1 com período de 1.2 segundo.
- Não é tão simples, né?

Tarefa 1 - Multitarefa com Períodos Diferentes

- Fazer dois LEDs piscarem. LED0 com período de 1.0 segundo, LED1 com período de 1.2 segundo.
- Não é tão simples, né?
- Vamos já aprender uma forma simples e eficiente!

Temporizadores

Funções úteis para administrar o tempo de processamento:

- delay(*tempo*); $\rightarrow$ *tempo* = int. Espera sem fazer nada por *tempo* ms.
- delayMicroseconds(tempo); $\rightarrow$ *tempo* = int. Espera sem fazer nada por *tempo* μs.
- millis(); $\rightarrow$ Retorna quanto tempo (em ms) se passou desde a última inicialização.
- micros(); $\rightarrow$ Retorna quanto tempo (em μs) se passou desde a última inicialização.

Tarefa 0 - Relógio Simples

- Faça a implementação de um relógio que exibe o tempo na serial.

  Dica: ulitizar a função **millis()**.

Tarefa 0 - Relógio Simples

```
1  // Contador de tempo simples - Adriano Rodrigues.
2  int tempo;
3
4  void setup()
5  {
6       Serial.begin(9600);
7  }
8
9  void loop()
10 {
11      Serial.println(millis());
12      delay(1000);
13 }
14
```

Temporizadores

Como operar duas (ou mais) tarefas que demandam períodos diferentes?

Evitar o uso de delay é a forma mais eficiente de executar diversas *threads*. Vamos relembrar as funções de tempo.

- delay(*tempo*); $\rightarrow$ *tempo* = int. Espera sem fazer nada por *tempo* ms.
- delayMicroseconds(tempo); $\rightarrow$ *tempo* = int. Espera sem fazer nada por *tempo* μs.
- millis(); $\rightarrow$ Retorna quanto tempo (em ms) se passou desde a última inicialização.
- micros(); $\rightarrow$ Retorna quanto tempo (em μs) se passou desde a última inicialização.

delay() vs millis()

```
1  // Codigo COM delay().
2  void loop()
3  {
4       // Digite aqui as instrucoes.
5       delay(500);
6  }
7
8
9  //Codigo SEM delay().
10 unsigned long timer; // Evitar erros de overflow.
11 void loop()
12 {
13      if (millis()-timer >= 500)
14      {
15           timer = millis(); // Atualizar o timer (NAO ESQUECER!).
16           //Digite aqui as instrucoes.
17      }
18 }
```

- Conseguem ver a diferença?

delay() vs millis()

```
1  // Codigo COM delay().
2  void loop()
3  {
4       // Digite aqui as instrucoes.
5       delay(500);
6  }
7
8
9  //Codigo SEM delay().
10 unsigned long timer; // Evitar erros de overflow.
11 void loop()
12 {
13      if (millis()-timer >= 500)
14      {
15           timer = millis(); // Atualizar o timer (NAO ESQUECER!).
16           //Digite aqui as instrucoes.
17      }
18 }
```

- Conseguem ver a diferença?
- Sem delay aproveitamos melhor nosso precioso processamento.

Tarefa 1

- Fazer dois LEDs piscarem. LED0 a cada 1 segundo, LED1 a cada 1.2 segundos.

Tarefa 1 - Acionamento de LEDs Multitask

```
1  // Acionamento de LEDs Multitask - Adriano Rodrigues.
2  #define LED0  13  // Maneira alternativa de declarar constantes.
3  #define LED1  12  // Nao consome memoria do arduino.
4  #define temp0 500
5  #define temp1 600
6  unsigned long timer0, timer1;
7  boolean E[2]; // Vetor de estados - Variavel GLOBAL.
8  void setup() // Dispensa comentarios.
9  {
10      pinMode(LED0, OUTPUT); pinMode(LED1, OUTPUT);
11 }
12 void loop()
13 {
14      if (millis()-timer0>=temp0)
15      {
16           timer0 = millis();    piscar(LED0, 0); // Pisca o LED zero.
17      }
18      if (millis()-timer1>=temp1)
19      {
20           timer1 = millis();    piscar(LED1, 1); // Pisca o LEL um.
21      }
22 }
```

Obrigado pela Participação.

Aula 03 - SEEL 2019
Minicurso de Arduino
Funções, Temporizadores

Adriano Rodrigues

25 de outubro de 2019