Aula 03 - SEEL 2019
Minicurso de Arduino
Funções, Temporizadores e Sensor Infravermelho

Adriano Rodrigues

25 de outubro de 2019

- Funções;
- Declarações const e #define;
- Temporizadores;
- Sensor IR:
  - Calibração do Sensor IR.

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 }
```

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 eneganadamente 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.

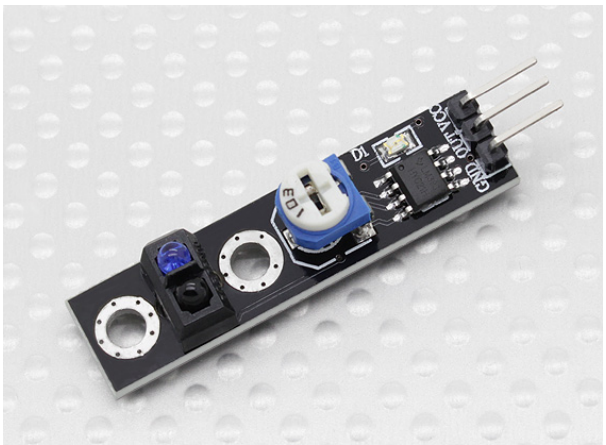
Tarefa 1

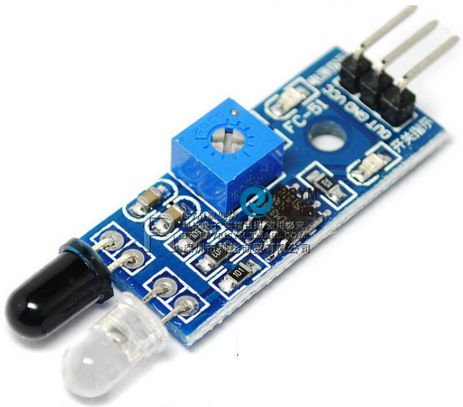
- Fazer dois LEDs piscarem. LED0 a cada 1 segundo, LED1 a cada 1.2 segundos.

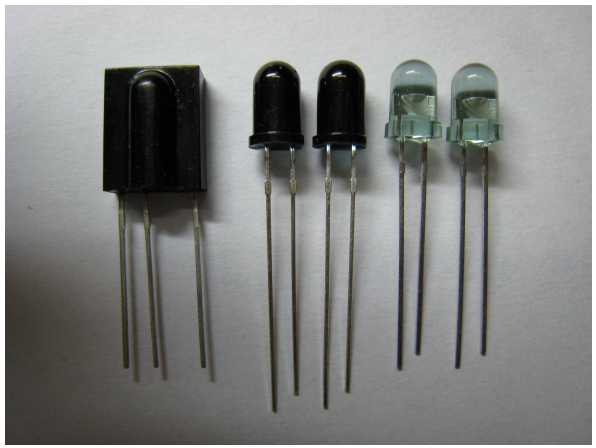
Tarefa 1 - Acionamento de LEDs Multitask

```
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      Serial.begin (9600); 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 }
```

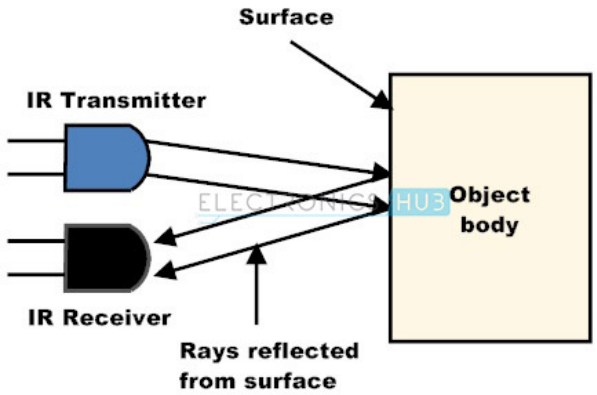
Sensor de Infravermelho

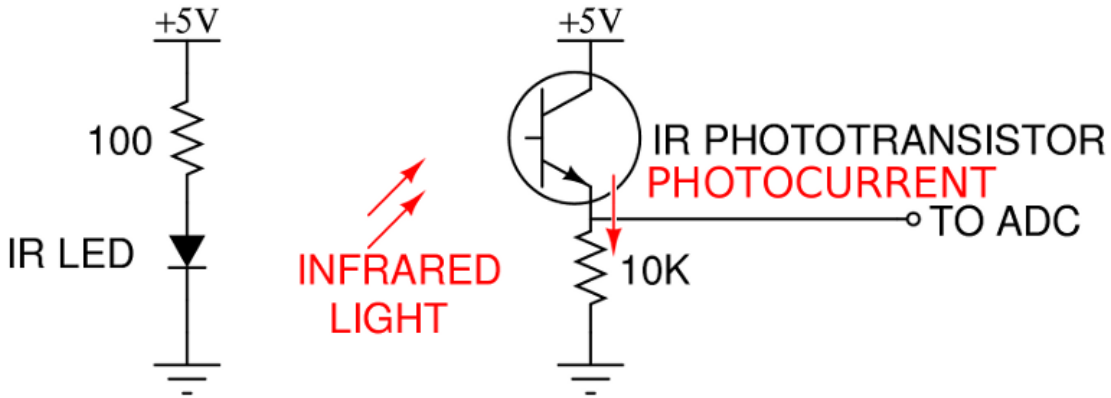






Sensor de Infravermelho





Ligar o terra do receptor na porta digital!

Exercício 2 - Sensor Infravermelho

```
1  // Infrared basico - Adriano Rodrigues.
2  #define temp 100
3  int led = 13, port_sinal = 2; // Porta do LED e do SINAL.
4  boolean sinal = false       ; // Registrador do SINAL.
5  unsigned long timer         ;
6
7  void setup()
8  {
9       pinMode(led, OUTPUT)      ; // Configura porta digital do LED.
10      pinMode(port_sinal, INPUT); // Configura porta digital do SINAL.
11 }
12
13 void loop()
14 {
15      if (millis() - timer >= temp)
16      {
17           timer = millis()                ; // Atualiza timer.
18           sinal = digitalRead(port_sinal); // Le o SINAL.
19           digitalWrite(led, sinal)        ; // Escreve na porta LED.
20      }
21 }
```

Exercicio 3 - IR + analogRead + Serial

```
1  // IR + analogRead + Serial - Adriano Rodrigues.
2  #define temp 100
3  int Leitura;
4  unsigned long timer;
5
6  void setup()
7  {
8      Serial.begin(9600);
9  }
10 void loop()
11 {
12     if (millis() - timer >= temp)
13     {
14         timer = millis()          ;
15         Leitura = analogRead(A0);
16         Serial.println(Leitura) ;
17     }
18 }
```

Tarefa 2 - Calibragem do IR

- Exibir o valores analógicos do infravermelho pela serial;
- Encontrar faixa de corte (sensibilidade) de cada sensor.

Tarefa 2 - Calibragem do IR

```
1  // Calibragem do IR - Adriano Rodrigues.
2  #define temp 100
3  int  corte, Le     ; // Corte e leitura.
4  char char_Le = 'B' ; // Indicador de Branco ou Preto.
5  unsigned long timer;
6  void setup()
7  {
8       Serial.begin(9600);
9  }
10 void loop()
11 {
12      if (millis() - timer >= temp)
13      {
14           timer = millis();
15           Le = analogRead(A0); Serial.print(Le); // Recebe e imprime leitura
16           // Le e imprime corte.
17           corte = analogRead(A2); Serial.print('\t'); Serial.print(corte);
18           if (Le > corte) char_Le = 'B'; // Branco se Le>corte.
19           else char_Le = 'P'           ; // Preto caso contrario.
20           Serial.print('\t'); Serial.println(char_Le);
21      }
22 }
```

Tarefa 2.2 - Calibragem do IR

```
1  #define temp = 100
2  int  corte, Le; char char_Le = 'B'; unsigned long timer;
3  void setup()
4  {
5       Serial.begin(9600);
6  }
7  void loop()
8  {
9       if (millis() - timer >= temp)
10      {
11           timer = millis();
12           Le = analogRead(A0); corte = analogRead(A2); // Recebe leitura e corte.
13           if (Le > corte) char_Le = 'B'; // Branco se leitura>corte.
14           else char_Le = 'P'            ; // Preto caso contrario.
15           imprimir(Le, corte, char_Le) ; // Chama impressao.
16      }
17 }
18 void imprimir(int l, int c, char cl)
19 {
20      Serial.print(l); Serial.print('\t'); Serial.print(c);
21      Serial.print('\t'); Serial.println(cl);
22 }
```

Obrigado pela Participação.

Aula 03 - SEEL 2019
Minicurso de Arduino
Funções, Temporizadores e Sensor Infravermelho

Adriano Rodrigues

25 de outubro de 2019