05

Melhorando nosso código

Transcrição

Diversas vezes não sabemos o porquê do código não funcionar. Para nos auxiliar nessa tarefa existe o `Monitor Serial`. Nele, podemos `printar`, ou seja, escrever o valor de variáveis e outras mensagens que nos guiarão pelo programa.

Para utilizá-lo é preciso, primeiro, iniciá-lo com a função `Serial.begin()`. Essa função recebe como parâmetro a *taxa de transferência* que pode ser verificada no próprio monitor serial, no canto inferior esquerdo. Em nosso caso, essa taxa é de **9600**. Para escrever algo dentro do monitor basta usar a função `Serial.println()` e, por exemplo, se quisermos saber o valor da variável `ledVermelho` a cada iteração do nosso `loop()`, podemos usar o seguinte código:

```
//variável global
int ledVermelho = 2;



void setup(){
    Serial.begin(9600);
    pinMode(ledVermelho, OUTPUT);
}


void loop(){
    Serial.println(ledVermelho);
    digitalWrite(ledVermelho,HIGH);
    delay(1000);
    digitalWrite(ledVermelho,LOW);
    delay(500);
    ledVermelho++; //aumenta o valor de ledVermelho em 1
}
```

Dentro do `Monitor Serial` podemos verificar o valor de `ledVermelho` aumentando. Além disso, a luz para de piscar após a primeira interação. Isso acontece pois o valor de `ledVermelho` muda, conforme podemos verificar, logo, a porta indicada para ligar e desligar é a porta errada.

Utilizando constantes

Com esse pequeno experimento percebemos que, na verdade, o valor de `ledVermelho` não deveria ser uma variável, mas sim uma **constante**. Isso significa que precisamos fazer com que o valor dela **nunca** mude. Podemos, ainda, extrair outras constantes como o `segundo` e o `meio-segundo`. O jeito de *definir* isso no C++ é bem simples. Por convenção, toda constante fica completamente em caixa-alta. Portanto:

```
//int ledVermelho = 2;
//declarando a constante


#define LED_VERMELHO 2
#define UM_SEGUNDO 1000
#define MEIO_SEGUNDO 500




void setup(){
    Serial.begin(9600);
    pinMode(LED_VERMELHO, OUTPUT);
}


void loop(){
    Serial.println(LED_VERMELHO);
    digitalWrite(LED_VERMELHO,HIGH);
    delay(UM_SEGUNDO);
    digitalWrite(LED_VERMELHO,LOW);
    delay(MEIO_SEGUNDO);
}
```

Nos preparando para os próximos LEDs

Nesse momento o código está funcionando como queremos. Porém, nosso jogo pode ter *(e terá)* diversos LEDs. Se formos copiar e colar cada linha de código para cada LED diferente a chance de cometer algum erro é gigante. Por isso, criaremos funções que irão encapsular, ou seja, guardar um código específico para nós. Assim, quando elas forem chamadas executarão o código normalmente. Vamos fazer isso com o código de piscar o LED?

```
//int ledVermelho = 2;
//declarando a constante


#define LED_VERMELHO 2
#define UM_SEGUNDO 1000
#define MEIO_SEGUNDO 500




void setup(){
    Serial.begin(9600);
    pinMode(LED_VERMELHO, OUTPUT);
}


void loop(){
    Serial.println(LED_VERMELHO);
    piscaLed(); //estamos chamando a função
}
```

```
void piscaLed(){
    digitalWrite(LED_VERMELHO,HIGH);
    delay(UM_SEGUNDO);
    digitalWrite(LED_VERMELHO,LOW);
    delay(MEIO_SEGUNDO);
}
```

Perceba que já usamos diversas funções em nosso código, por exemplo, o `pinMode()`. Dentro dos parênteses passamos alguns valores como `LED_VERMELHO` e `OUTPUT`. Podemos mudar a função para que pisque a cor específica do led que indicamos. Para isso, vamos passar parâmetros no `loop()` e declará-los na nossa função mesmo. Fica assim:

```
//int ledVermelho = 2;
//declarando a constante


#define LED_VERMELHO 2
#define UM_SEGUNDO 1000
#define MEIO_SEGUNDO 500



void setup(){
    Serial.begin(9600);
    pinMode(LED_VERMELHO, OUTPUT);
}


void loop(){
    Serial.println(LED_VERMELHO);
    piscaLed(LED_VERMELHO); //estamos chamando a função
}


void piscaLed(int portaLed){
    digitalWrite(portaLed, HIGH);
    delay(UM_SEGUNDO);
    digitalWrite(portaLed, LOW);
    delay(MEIO_SEGUNDO);
}
```