01

Controlando o Arduino com código

Transcrição

Olá pessoal!

Vamos continuar construindo nosso **Genius**. Repare no circuito que desenvolvemos desde o capítulo 1, nele conectamos o LED na porta de **5V**. Porém, nós não temos *controle* sobre essa porta o que significa que não temos como alterá-la, assim ela **sempre** irá fornecer 5V. Para termos esse controle de falar para a porta o que ela deve fazer, precisamos usar as *portas digitais*. As portas digitais ficam do lado contrário a de 5V.

Atenção: as portas 0 e 1 não devem ser usadas neste momento, por isso, comece pela porta **2**.

Agora que nosso fio está conectado a **porta 2** e não mais na de 5V, precisamos criar os comandos de ação para essa porta. Para isso, precisaremos *programar* esse comportamento. A linguagem de programação usada no **Arduino** é a **C++**, nós utilizaremos a IDE oficial, que pode ser baixada aqui (https://www.arduino.cc/en/Main/Software).

Antes de começarmos a programar o ideal é que você observe esta página (https://www.arduino.cc/en/Reference/HomePage). Nela estão armazenadas todas as documentações de funções, constantes, variáveis, estruturas e muito mais. Nesse curso tentaremos cobrir tudo o que utilizarmos porém, essa é uma dica valiosa para seus projetos futuros.

Começando a Programar

Ao abrir a IDE do **Arduino** nos deparamos com duas funções: `setup()` e `loop()`. O código dentro de `setup()` será executado uma vez, já o código contido no `loop()`, como o próprio nome insinua, será executado infinitas vezes até pararmos ele.

Queremos comunicar ao **Arduino** que "Nós temos uma conexão na porta 2!". Consultando a documentação encontramos a função `pinMode()`, que recebe a porta e indica se ela é de **saída** ou **entrada**. Vamos falar, então, que nossa **porta 2** é de **saída**, assim, a partir dela emitiremos o sinal para o LED acender.

```
void setup(){
    pinMode(2, OUTPUT);
}
```

Após a codificação, nós compilamos o código para verificar se tudo está em ordem com a sintaxe. Feito isso é necessário verificar se estamos lidando com a placa correta. Vamos em **Ferramentas -> Placa ...** e averiguamos se nosso modelo está selecionado. Depois, logo abaixo, verificamos se o SO achou a porta certa do **Arduino**. Ele deve fazer isso automaticamente.

Feito isso, podemos voltar a programar! No caso, queremos que a luz acenda! Dessa forma, é preciso escrever um sinal para a nossa porta. Usaremos a função `digitalWrite()`, que recebe o sinal e também a porta para a qual enviaremos o sinal. Como desejamos que acenda, enviaremos o **sinal 1**.

```
void setup(){
    pinMode(2, OUTPUT);
    digitalWrite(2,1);
}
```

Agora, já podemos verificar se funciona! Ao compilarmos novamente e ao clicarmos na seta ao lado, que envia o programa para o **Arduino**, veremos a luz acender! Porém, dessa vez, fomos nós que controlamos a saída de sinal. Por exemplo, se quiséssemos acender e apagar o LED em seguida, bastaria repetir a linha e mandar um **sinal 0**.

```
void setup(){
    pinMode(2, OUTPUT);
    digitalWrite(2,1);
    digitalWrite(2,0);
}
```

Compilando e mandando esse código para o **Arduino** apenas aparecerá que a luz ficou apagada. Isso acontece porque logo depois que ela acende, ela também apaga! Só que isso acontece muito rápido! Para resolvermos essa questão podemos inserir um intervalo de tempo entre as duas ações:

```
void setup(){
    pinMode(2, OUTPUT);
    digitalWrite(2,1);
    delay(1000);
    digitalWrite(2,0);
}
```

A função `delay()` recebe como parâmetro o tempo, que deve ser escrito em *milisegundos*. Logo, ao escrever `delay(1000)`, isso significa que o tempo de espera será de *1 segundo* até que a luz se apague. Mas, note que quando usamos **sinal 1** ou **sinal 0** não estamos aplicando códigos muito intuitivos ou legíveis. Portanto, podemos substituir esses nomes por *constantes* do **Arduino**, as chamadas **HIGH** e **LOW** que significam, respectivamente, *ligado* e *desligado*. Observe como fica substituindo isso em nosso código:

```
void setup(){
    pinMode(2, OUTPUT);
    digitalWrite(2,HIGH);
    delay(1000);
    digitalWrite(2,LOW);
}
```

Repare que repetimos o valor **2**, que é a nossa porta digital. Será que outra pessoa ao ler nosso código, entenderia isso de primeira? Acho que não! Por isso, podemos acoplar esse valor **2** em uma variável. Como nosso código está escrito em linguagem C++, é preciso acrescentar um tipo. Esse tipo será *inteiro*, no caso, `int`. Assim, em todo lugar que quisermos representar nossa porta deveremos adicionar o nome dessa variável. Ficará assim:

```
void setup(){
    int ledVermelho = 2;
    pinMode(ledVermelho, OUTPUT);
    digitalWrite(ledVermelho,HIGH);
    delay(1000);
    digitalWrite(ledVermelho,LOW);
}
```

Passando para o loop

Caso surjam dúvidas ao longo do processo basta compilar e testar para verificar se continua funcionando!

Agora, vamos reparar outro *erro* nosso! Queremos que a luz **pisque**. Porém, o código do `setup()` roda apenas uma vez. Logo, fica fácil resolver isso, basta passar a utilizar o código de acender e apagar a luz junto com o `loop()`. Porém, ao fazer isso, a função `loop()` não irá enxergar a variável `ledVermelho`, que ficou no `setup()`. Para solucionar esse problema faremos com que a variável seja **global**, o que significa que ela será "vista" por qualquer função. Além disso, vamos adicionar um tempo para a *luz apagada* também. Vamos então ao código:

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



void setup(){
    pinMode(ledVermelho, OUTPUT);
}


void loop(){
    digitalWrite(ledVermelho,HIGH);
    delay(1000);
    digitalWrite(ledVermelho,LOW);
    delay(500); //espera meio segundo antes de acender novamente
}
```