CURSO TÉCNICO DE INFORMÁTICA

DISCIPLINA: AUTOMAÇÃO

AULA 03/04/2020

» *Linguagem de programação*

Uma concepção errônea a respeito do Arduino é que ele tem a sua própria linguagem de programação. Na realidade, ele é programado em uma linguagem denominada C. Essa linguagem já existe desde os primeiros dias da computação. O que o Arduino proporciona é um conjunto simples de comandos de fácil uso escrito em C que você pode usar em seus programas.

Os mais tradicionais poderão dizer que o Arduino usa C++, a extensão orientada a objetos da linguagem C. Rigorosamente falando, isso é verdade, mas o Arduino dispõe de somente 1 ou 2Kb de memória. Como resultado, as práticas encorajadas pela programação orientada a objetos não são uma boa ideia quando aplicadas ao Arduino. Na prática, fora alguns casos especiais, você está de fato programando em C.

Vamos começar modificando o sketch Blink (pisca).

» *Modificando o sketch Blink*

É possível que o Arduino tenha começado a piscar quando você o ligou pela primeira vez. Isso aconteceu porque o Arduino costuma vir com o sketch Blink já instalado.

Se for o caso, então você gostará da ideia de ver se consegue modificar o sketch. Para isso, você pode, por exemplo, alterar a velocidade do pisca-pisca. Vamos exa-

minar o sketch Blink para ver como modificá-lo e fazer o LED piscar mais rapidamente.

A primeira parte do sketch é apenas um comentário dizendo o que o sketch faz. Um comentário não é um trecho executável de código de programa. Quando o sketch está sendo preparado para ser transferido ao Arduino, todos esses "comentários" são removidos. Qualquer coisa entre /* e */ (comentário em bloco) é ignorada pelo computador, mas pode ser lida por você.

```
/*
  Blink
  Liga um LED por um segundo, depois o desliga por um segundo, repetidamente.
  O código desse exemplo é de domínio público.
 */
```

A seguir, há dois comentários em linhas individuais, como os comentários em bloco, exceto que eles começam com //. Esses comentários dizem o que está acontecendo. Nesse caso, o comentário auxilia dizendo que o pino 13 é o pino que usaremos para controlar o pisca-pisca do LED. Esse pino foi escolhido porque já está conectado ao LED "L" que faz parte do Arduino Uno.

```
// Na maioria das placas de Arduino, há um LED conectado ao pino 13.
// Chamaremos esse pino de "led":
int led = 13;
```

A próxima parte do sketch é a função **setup** (inicialização). Todo sketch de Arduino deve ter uma função **setup**. Essa função é executada sempre que o Arduino é submetido a um reset (inicialização) porque o botão Reset foi pressionado (como o comentário afirma) ou porque o Arduino foi ligado.

```
// A rotina de setup é executada quando você pressiona o botão reset:
void setup() {
  // Inicializar o pino digital como sendo uma saída (OUTPUT).
  pinMode(led, OUTPUT);
}
```

A estrutura desse texto é um pouco confusa para quem está começando a programar. Uma *função* é uma seção do código do programa (sketch) que recebeu um nome (nesse caso, o nome é **setup**). Por enquanto, use o texto anterior como modelo e saiba que você deve iniciar o seu sketch com o comando **void setup(){** na primeira linha. A seguir, você deve incluir os comandos que deseja executar, cada um em uma linha terminada com um ponto e vírgula (;). O final da função é assinalado com o símbolo }.

Nesse caso, o único comando que o Arduino executará é o comando **pinMode(led, OUTPUT)**, que faz o pino (led) ser uma saída (OUTPUT).

A seguir vem a parte saborosa do sketch, a função **loop** (laço).

Assim como a função **setup**, todo sketch de Arduino deve conter uma função **loop** (laço de repetição). Diferentemente de **setup**, que é executada uma única vez depois de um reset, a função **loop** é executada repetidas vezes, indefinidamente. Isto é, após a execução de todas as suas instruções, ela volta a ser executada.

Na função **loop**, para acender o LED, devemos executar a instrução **digitalWrite(led,HIGH)**, que "escreve" (write) um nível alto (HIGH) no pino digital "led." A seguir, há uma pausa, ou retardo (delay), na execução do sketch, obtida por meio do comando **delay(1000)**. O valor 1000 significa 1000 milissegundos ou 1 segundo. Então você apaga o LED e espera por outro segundo antes que todo o processo comece novamente.

```
// A rotina loop é executada repetidas vezes, indefinidamente:
void loop() {
  digitalWrite(led, HIGH);       // Acender o LED (o nível da tensão é HIGH (alto)).
  delay(1000);                   // Esperar (delay) um segundo.
  digitalWrite(led, LOW);        // Apagar o LED baixando (LOW) o nível da tensão.
  delay(1000);                   // Esperar um segundo.
}
```

Para modificar esse sketch de modo que o LED pisque mais rapidamente, mude o valor de 1000 para 200 nos dois lugares onde aparece. Essas mudanças são feitas na função **loop**. Desse modo, a sua função será como se mostra a seguir:

```
void loop() {
  digitalWrite(led, HIGH);       // Acender o LED (o nível da tensão é HIGH (ALTO)).
  delay(200);                    // Esperar (delay) um segundo.
  digitalWrite(led, LOW);        // Apagar o LED baixando (LOW) o nível da tensão.
  delay(200);                    // Esperar um segundo.
}
```

Se você tentar salvar o sketch antes de transferi-lo (upload), o IDE do Arduino irá lembrá-lo de que o sketch "só pode ser lido" (read-only), porque se trata de um exemplo. Entretanto, ele dá a opção de salvá-lo como cópia que, então, poderá ser modificada segundo o seu desejo.

Não é necessário fazer isso. Você pode simplesmente fazer upload do sketch sem salvá-lo antes. Se decidir salvar esse ou outros sketches, você os encontrará no menu File | Sketchbook do IDE.

Assim, de qualquer modo, clique novamente no botão Upload e, quando a transferência estiver completa, o Arduino se inicializará automaticamente e o LED deverá começar a piscar muito mais rapidamente.

» *Variáveis*

Uma variável dá um nome a um número. Na realidade, as variáveis são capazes de muito mais, mas por enquanto nós as usaremos com essa finalidade.

Quando você define uma variável em C, deve especificar o tipo da variável. Por exemplo, se deseja que suas variáveis sejam números inteiros, você deverá usar *int*, de *integer* (inteiro). Para definir uma variável de nome **delayPeriod** (período do retardo) com um valor de **200**, você deve escrever:

```
int delayPeriod = 200;
```

Observe que, como **delayPeriod** é um nome, não deve haver espaços entre as palavras do nome. Por convenção, no nome de uma variável constituído por diversas palavras, a primeira palavra começa com letra minúscula e cada palavra seguinte começa com letra maiúscula. Essa forma de escrever é conhecida como *camel case*, ou *bumpy case*.

Vamos aplicar esse recurso ao sketch Blink. Assim, em vez de escrever o valor 200 de retardo, usaremos uma variável que chamaremos de **delayPeriod**:

```
int led = 13;
int delayPeriod = 200;

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

void loop()
{
 digitalWrite(led, HIGH);
 delay(delayPeriod);
 digitalWrite(led, LOW);
 delay(delayPeriod);
}
```

Nos lugares do sketch onde antes havíamos escrito **200**, agora temos **delayPeriod**.

Agora, se quisermos fazer o sketch piscar mais rapidamente, basta alterar o valor de **delayPeriod** em um único lugar.

» *If*

Normalmente, as linhas do programa são executadas sequencialmente, uma depois da outra, sem exceção. No entanto, o que faremos se não quisermos que seja assim? O que fazer se quisermos executar uma parte do sketch somente se uma dada condição for verdadeira?

Um bom exemplo disso seria fazer alguma coisa somente se um botão, conectado ao Arduino, fosse apertado. O código seria algo como:

```
void setup()
{
  pinMode(5, INPUT_PULLUP);
  pinMode(9, OUTPUT);
}

void loop()
{
  if (digitalRead(5) == LOW;
  }
    digitalWrite(9, HIGH);
  }
}
```

Nesse caso, a condição está logo após o comando **if** (se). A condição é de que o valor lido (read) no pino digital 5 deve ter o valor **LOW** (baixo). O símbolo composto de dois sinais de igual (==) é usado para comparar dois valores. Pode ser facilmente confundido com o sinal simples de igual, que atribui valor a uma variável. Se a condição do **if** for verdadeira, os comandos dentro dos sinais de chaves ({}) serão executados. Nesse caso, a ação consiste em fazer com que a saída digital **9** tenha um nível alto (**HIGH**).

Se essa condição não for verdadeira, o Arduino simplesmente continuará com a execução da próxima coisa a ser feita. Nesse caso, é a função **loop** que é executada novamente.

» *Loops*

Além de executar comandos de forma condicional, também pode ser necessário que o sketch execute comandos repetidas vezes de forma contínua. Para tanto, você coloca os comandos dentro da função **loop** do sketch. Isso é o que acontece no exemplo Blink.

Algumas vezes, entretanto, pode ser necessário ser mais específico sobre o número de vezes que você deseja repetir alguma coisa. Você pode conseguir isso com o comando **for**, que é um laço de repetição no qual é permitido usar uma variável de contagem. Por exemplo, vamos escrever um sketch que pisca o LED dez vezes. Mais adiante você verá por que essa abordagem pode ser considerada abaixo do ideal em algumas circunstâncias, mas por enquanto será suficiente.

```
// sketch 01_01_blink_10
int ledPin = 13;
int delayPeriod = 200;
void setup()
{
  pinMode(ledPin, OUTPUT);
}
void loop()

{
  for (int i = 0; i < 10; i++)
  {
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
  }
}
```

O comando **for** define uma variável de nome **i** e atribui o valor inicial **0** a ela. Após o ponto e vírgula (;), aparece o texto **i < 10**. Essa é a condição para permanecer dentro do laço de repetição **for**. Em outras palavras, enquanto **i** for menor que **10**, siga fazendo as coisas que estão dentro das chaves ({}).

A última parte do comando **for** é **i++**. Essa é uma forma abreviada da linguagem C para o comando "i = i + 1", que acrescenta 1 ao valor de **i**. O valor 1 é adicionado a **i** sempre que o **loop** é repetido. Esse é o mecanismo que assegura a saída para fora do **loop**, porque, se ficarmos acrescentando 1 a **i**, acabaremos alcançando um valor maior que 10.

Funções

Funções são uma forma de agrupar um conjunto de comandos de programa em um bloco único. Isso ajuda a dividir o sketch em blocos administráveis, facilitando o seu uso.

Por exemplo, vamos escrever um sketch que faz o Arduino piscar rapidamente 10 vezes no início da execução do sketch e, após, piscar uma vez a cada segundo.

Leia inteiramente a seguinte listagem do sketch. Após, eu explicarei o que está acontecendo.

```
// sketch 01_02_blink_fast_slow
int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
  flash(10, 100);
}
void loop()
{
  flash(1, 500);
}

void flash(int n, int delayPeriod)
{
  for (int i = 0; i < n; i++)
  {
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
  }
}
```

Agora, a função **setup** contém a linha que diz **flash(10, 100);**. Isso significa piscar (flash) **10** vezes com um **delayPeriod** (período de retardo) de **100** milissegundos. Como o comando **flash** não é um comando interno do Arduino, você mesmo deve criar essa função de muita serventia.

A definição da função está no final do sketch. A primeira linha dessa definição de função é

```
void flash(int n, int delayPeriod)
```

Isso diz ao Arduino que você está definindo a sua própria função, que terá o nome **flash** e dois parâmetros, ambos do tipo **int**. O primeiro é **n**, o número de vezes que o LED deve piscar, e o segundo é **delayPeriod**, o tempo entre ligar e desligar o LED.

Esses dois parâmetros podem ser usados somente dentro da função. Assim, **n** é usado no comando **for** para determinar quantas vezes o laço deve ser repetido e **delayPeriod** é usado dentro do comando **delay**.

A seguir, aparece a função **loop**, que também usa a função **flash** que acabamos de ver. Agora, ela é executada com um **delayPeriod** mais longo e faz o LED piscar apenas uma vez a cada repetição do **loop**. Como ela está dentro do **loop**, o resultado é que o LED ficará piscando continuamente de qualquer forma.

» *Entradas digitais*

Para obter o máximo desta seção, você precisará de um pedaço de fio ou clipe de papel. Se você usar um clipe, desentorte-o para deixá-lo reto.

Carregue o seguinte sketch e execute-o:

```
// sketch 01_03_paperclip
int ledPin = 13;
int switchPin = 7;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(switchPin, INPUT_PULLUP);
}

void loop()
{
  if (digitalRead(switchPin) == LOW)
  {
    flash(100);
  }
  else
  {
    flash(500);
  }
}

void flash(int delayPeriod)
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
}
```

Use o pedaço de fio ou clipe para conectar o pino GND ao pino digital 7, como mostrado na Figura 1.13. Você pode fazer isso com o Arduino conectado ao computador, mas somente após ter carregado o sketch. A razão é que, se em algum sketch anterior o pino 7 foi definido como saída, então ele será danificado se for conectado ao pino GND. Como o sketch está definindo o pino 7 como entrada, então podemos ficar seguros.

O que deverá acontecer é o seguinte: quando o clipe de papel está conectado, o LED pisca rapidamente, e quando não está conectado, pisca lentamente.

Vamos dissecar o sketch e ver como ele funciona.

Primeiro, temos uma nova variável, denominada **switchPin** (pino da chave). Essa variável é atribuída ao pino 7. Assim, o clipe de papel atuará como uma chave. Na função **setup**, quando usamos o comando **pinMode** (modo do pino) estamos especificamos que esse pino será uma entrada. O segundo argumento de **pinMode** não é simplesmente **INPUT** (entrada), mas **INPUT_PULLUP** (entrada puxada para cima). Por *default*, isso diz ao Arduino que a entrada ficará normalmente em nível alto (**HIGH**), a menos que seja puxada para baixo (**LOW**) quando a conectamos ao pino GND (com o clipe de papel).

Na função **loop**, usamos o comando **digitalRead** (leitura digital) para testar qual é o valor presente no pino de entrada. Se for **LOW** (clipe conectado), então será chamada a função **flash** com o parâmetro **100** (valor do **delayPeriod**). Com isso, o LED pisca rapidamente.

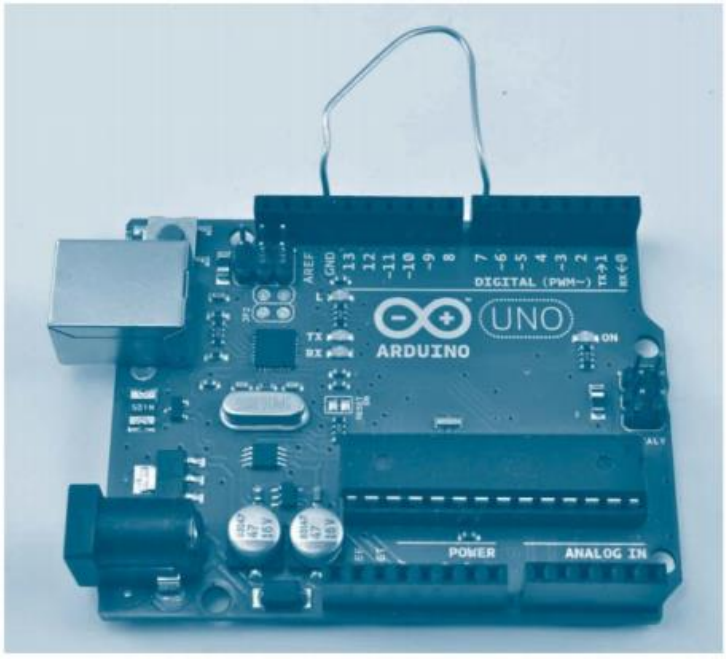

Figura 1.13 Usando uma entrada digital.

Por outro lado, se a entrada for **HIGH**, os comandos da parte **else** do **if** serão executados. Com isso, a mesma função **flash** será chamada, mas com um retardo muito maior, fazendo o LED piscar lentamente.

A função **flash** definida nesse sketch é uma versão simplificada da que você usou no sketch anterior. Ela se limita a piscar uma única vez com o período especificado.

No caso de módulos ligados ao Arduino, as saídas digitais não funcionam como chaves, que produzem apenas um nível. Na realidade, essas saídas produzem ambos os níveis **HIGH** ou **LOW**. Nesse caso, você pode usar **INPUT** em vez de **INPUT_PULLUP** na função **pinMode**.

» *Saídas digitais*

Do ponto de vista de programação, não há muito mais o que dizer sobre saídas digitais. Você já as usou com o pino 13, que está conectado ao LED do Arduino.

A essência de uma saída digital é defini-la como saída dentro da função **setup** usando o seguinte comando:

```
pinMode(outputPin, OUTPUT);
```

Quando quer colocar a saída em nível **HIGH** ou **LOW**, você usa o comando **digitalWrite** (escrita digital):

```
digitalWrite(outputPin, HIGH);
```

» O Monitor Serial

Como o Arduino está conectado ao seu computador por USB, você pode enviar mensagens entre os dois usando um recurso do IDE de Arduino denominado *Serial Monitor* (Monitor Serial).

Para ilustrar, vamos modificar o sketch 01_03 de modo que, quando a entrada digital 7 está em nível LOW, ele envia uma mensagem em vez de alterar a velocidade de pisca-pisca do LED.

Carregue este sketch:

```
// sketch 01_04_serial
int switchPin = 7;

void setup()
{
  pinMode(switchPin, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop()
{
  if (digitalRead(switchPin) == LOW)
  {
    Serial.println("Paperclip connected");
  }
  else
  {
    Serial.println("Paperclip NOT connected");
  }
  delay(1000);
}
```

Agora abra o Monitor Serial no IDE de Arduino clicando no ícone parecido com uma lupa na barra de ferramentas. Imediatamente, você começará a ver algumas

Figura 1.14 O Monitor Serial.
Fonte: do autor.

mensagens, uma a cada segundo (Figura 1.14). As mensagens poderão ser "Paperclip connected" (Clipe de papel conectado) ou "Paperclip NOT connected" (Clipe de papel NÃO conectado).

Desconecte uma das extremidades do clipe de papel e você verá a mensagem mudar.

Como você não está mais usando o LED do Arduino, não precisa mais da variável **ledPin**. Em vez disso, você deve usar o comando **Serial.begin** (iniciar serial) para iniciar as comunicações seriais. O parâmetro é o "baud rate" (taxa de bauds). No Capítulo 13, você encontrará muito mais informações sobre comunicação serial.

Para escrever mensagens e enviá-las ao Monitor Serial, tudo o que você precisa fazer é usar o comando **Serial.println** (imprimir linha serial).

Nesse exemplo, o Arduino está enviando mensagens ao Monitor Serial.

» *Arrays e strings*

Arrays são uma forma de agrupar uma lista de valores. Cada uma das variáveis que você encontrou até agora continha apenas um único valor, geralmente do tipo **int**. Já um array contém uma lista de valores e você pode acessar qualquer um desses valores pela sua posição na lista.

A linguagem C, como a maioria das linguagens de programação, inicia a indexação das posições em 0 em vez de 1. Isso significa que o primeiro elemento é, na realidade, o elemento zero.

Na última seção, quando você conheceu o Monitor Serial, viu um tipo de array. Mensagens como **"Paperclip NOT connected"** ("Clipe de papel NÃO conectado") são denominadas *arrays de caracteres* porque basicamente são coleções de caracteres.

Por exemplo, vamos ensinar o Arduino a dizer algumas "bobagens" por meio do Monitor Serial.

O seguinte sketch tem um array de arrays de caracteres. O sketch escolherá ao acaso um deles e o exibirá no Monitor Serial depois de um tempo aleatório. Esse sketch tem a vantagem adicional de lhe mostrar como produzir números aleatórios (randômicos) com um Arduino.

```
// sketch 01_05_gibberish

char* messages[] = {
                "My name is Arduino",              // "Meu nome é Arduino"
                "Buy books by Simon Monk", // "Compre livros de Simon Monk"
                "Make something cool with me",     // "Faça algo legal comigo"
                "Raspberry Pis are fruity"};       // "Raspberry Pis são saborosos"

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int delayPeriod = random(2000, 8000);
  delay(delayPeriod);
  int messageIndex = random(4);
  Serial.println(messages[messageIndex]);
}
```

As mensagens ou *strings*, como também são denominadas as coleções de caracteres, utilizam um tipo de dado denominado **char***. O sinal * é um apontador para alguma coisa. Trataremos desse tópico avançado sobre apontadores no Capítulo 6. Os sinais [] no fim da declaração de variável indicam que a variável é um array do tipo **char*** e não simplesmente um único caractere do tipo **char***.

Dentro da função **loop**, o valor de **delayPeriod** (período de retardo) é atribuído de forma aleatória entre **2000** e **7999** (o segundo argumento da função "**random**" é exclusivo).[†] Então, uma pausa com essa duração é iniciada por meio da função **delay** (retardo).

A variável **messageIndex** (índice mensagem) também recebe um valor aleatório usando a função **random**. No entanto, dessa vez a função **random** recebe apenas um argumento como parâmetro. Nesse caso, um número aleatório entre 0 e 3 é gerado como índice para a mensagem que será exibida.

Finalmente, a mensagem contida na posição dada pelo índice é enviada ao Monitor Serial. Teste o sketch, não esquecendo de abrir o Monitor Serial.

Entradas analógicas

Os pinos do Arduino de A0 até A5 podem medir a tensão aplicada neles. A tensão deve estar entre 0 e 5V. A função interna do Arduino que faz isso é a **analogRead** (leitura analógica), a qual retorna um valor entre 0 e 1013: o valor 0 corresponde a 0V e 1023, a 5V. Assim, para converter esse número lido em um valor entre 0 e 5, o número lido deve ser dividido por 204,6 (de 1023/5).

Para medir a tensão, o tipo de dado ideal não é **int** porque ele só representa números inteiros. Seria bom conhecer também a parte fracionária da tensão. Para isso, você deverá usar o tipo de dado **float** (flutuante).

Carregue esse sketch de leitura analógica (analog) no Arduino e em seguida instale o clipe de papel entre o pino A0 e o de 3,3V (Figura 1.15).

```
// sketch 01_06_analog
int analogPin = A0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int rawReading = analogRead(analogPin);
  float volts = rawReading / 204.6;
  Serial.println(volts);
  delay(1000);
}
```

Abra o Monitor Serial e uma sequência de números deverá aparecer (Figura 1.16). Os seus valores deverão estar próximos de 3,3.

Se você mantiver uma das extremidades do clipe em A0 e mudar de lugar a outra extremidade inserindo-a em 5V, as leituras passarão para valores em torno de 5V. Se você deslocar esta mesma extremidade para GND, as leituras serão de 0V.

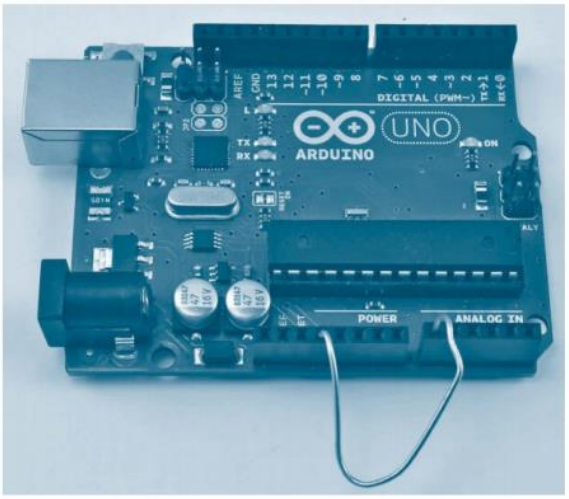

Figura 1.15 Conectando 3,3V a A0.

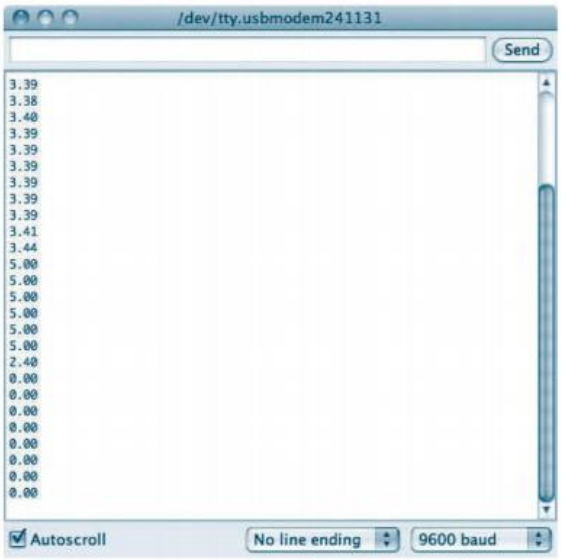

Figura 1.16 Leitura de tensões.

Saídas analógicas

Na realidade, o Arduino Uno não produz saídas verdadeiramente analógicas (para isso você precisaria de um Arduino Due). No entanto, ele tem pinos que são capazes de produzir saídas moduladas por largura de pulso (PWM, *pulse width modulation*). Isso permite que uma forma aproximada de saída analógica seja produzida pelo controle da duração de cada pulso de uma sequência, como você pode ver na Figura 1.17.

Quanto mais tempo o pulso permanecer em nível alto (HIGH), maior será a tensão média do sinal. Como são cerca de 600 pulsos por segundo e a maioria das coisas que você conecta a uma saída PWM responde muito lentamente, o efeito resultante é uma mudança no valor da tensão.

Em um Arduino Uno, os pinos marcados com um pequeno sinal ~ (pinos 3, 5, 6, 9, 10 e 11) podem ser usados como saídas analógicas.

Se você tiver um multímetro, coloque-o na escala de 20V CC (DC). A seguir, conecte a ponteira positiva ao pino digital 6 e a negativa ao pino GND (Figura 1.18). Então, carregue o sketch PWM seguinte:

```
// sketch 01_07_pwm
int pwmPin = 6;

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

void loop()
{
  if (Serial.available())
  {
    int dutyCycle = Serial.parseInt();
    analogWrite(pwmPin, dutyCycle);
  }
}
```

Abra o Monitor Serial e digite um número entre 0 e 255 na janela de entrada de texto na parte superior da tela próximo do botão Send (enviar). A seguir, aperte o botão Send e você verá no seu multímetro que a tensão mudará. Ao enviar um valor 0, deve resultar uma tensão em torno de 0V. O valor 127 corresponde a um valor entre 0 e 5V (2,5V, aproximadamente), e o valor 255 deverá produzir um valor próximo de 5V.

Nesse sketch, a função **loop** começa com um comando **if** (se). A condição para o **if** é **Serial.available()** (serial disponível). Isso significa que, se o Serial Monitor tiver enviado para o Arduino uma mensagem que está disponível e pronta para ser lida, os comandos contidos entre os sinais {} serão executados. Nesse caso, o comando **Serial.parseInt** converte a mensagem que você digitou no Monitor Serial para o tipo **int**, colocando o resultado na variável **dutyCycle** (ciclo de trabalho). O valor inteiro resultante é então usado como argumento de **analogWrite** (escrita analógica) que ativará a saída PWM.

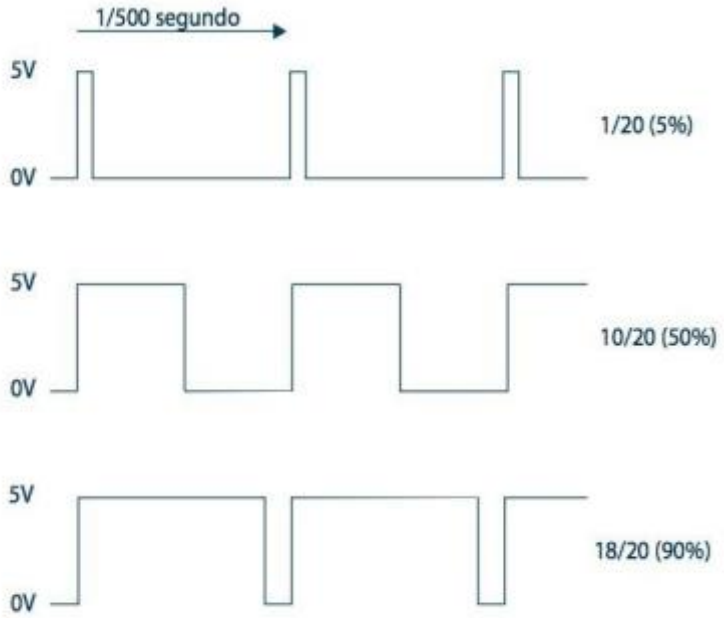

Figura 1.17 Modulação por largura de pulso.

» *Comandos do Arduino*

Na biblioteca do Arduino há um grande número de comandos disponíveis. Uma seleção dos comandos mais comumente usados, juntamente com exemplos, está listada na Tabela 1.2

Tabela 1.1 » Tipos de dados na linguagem C do Arduino

Tipo	Memória (bytes)	Intervalo	Observações
boolean	1	verdadeiro ou falso (0 ou 1)	Usado para representar valores lógicos.
char	1	–128 até +128	Usado para representar um código de caractere ASCII. Por exemplo, A é representado como 65. Normalmente, valores negativos não são usados.
byte	1	0 até 255	Frequentemente usado na comunicação serial de dados como uma unidade simples de dados. Veja o Capítulo 9.
int	2	–32768 até +32767	São valores de 16 bits com sinal.
unsigned int	2	0 até 65536	Pode ser usado para obter mais precisão quando não há necessidade de números negativos. Use com cautela, porque a aritmética que usa o tipo **int** pode produzir resultados inesperados.
long	4	2.147.483,648 até 2.147.483,647	Necessário apenas para representar números muito grandes.
unsigned long	4	0 até 4.294.967,295	*Veja* **unsigned int**.
float	4	–3.4028235E+38 até + 3.4028235E+38	Usado para representar números de ponto flutuante.
double	4	como **float**	Normalmente, seriam 8 bytes com uma precisão maior do que com **float** e um intervalo de representação maior. Entretanto, no Arduino, **double** é o mesmo que **float**.

Tabela 1.2 » Funções da biblioteca do Arduino

Comando	Exemplo	Descrição
	Entrada/saída digital	
pinMode	**pinMode(8, OUTPUT);**	Define o pino 8 como sendo saída. A alternativa é que seja entrada, usando **INPUT** ou **INPUT_PULLUP**.
digitalWrite	**digitalWrite(8, HIGH);**	Faz o nível do pino 8 ser alto. Para que seja baixo, use **LOW** em vez de **HIGH.**
digitalRead	**int i;** **i = digitalRead(8);**	Faz o valor de **i** ser **HIGH** ou **LOW**, dependendo da tensão no pino especificado (neste caso, pino 8).
pulseIn	**i = pulseIn(8, HIGH)**	Retorna a duração em microssegundos do próximo pulso com nível **HIGH** no pino 8.
tone	**tone(8, 440, 1000);**	Faz o pino 8 oscilar em 440 Hz durante 1.000 milissegundos.
noTone	**noTone();**	Corta imediatamente a geração de áudio que estiver em andamento.
	Entrada/saída analógica	
analogRead	**int r;** **r = analogRead(0);**	Atribui um valor a **r** entre 0 e 1023; 0 para 0V, 1023 se pin0 tiver 5V (3,3V no caso de uma placa de 3V).
analogWrite	**analogWrite(9, 127);**	Produz um sinal PWM na saída. O ciclo de trabalho apresenta um valor entre 0 e 255. 255 corresponde a 100%. Esse comando deve ser usado com qualquer pino marcado (~) como PWM na placa de Arduino (3, 5, 6, 9, 10 e 11).
	Comandos de tempo	
millis	**unsigned long l;** **l = millis();**	No Arduino, o tipo de variável **long** é representado com 32 bits. O valor retornado por **millis()** é o número de milissegundos decorridos desde a última inicialização (reset). O valor volta a zero depois de aproximadamente 50 dias.
micros	**long l;** **l = micros();**	*Veja* **millis**, exceto que é em microssegundos decorridos desde a última inicialização (reset). O valor volta a zero depois de aproximadamente 70 minutos.
delay	**delay(1000);**	Retardo de 1.000 milissegundos ou um segundo.

Tabela 1.2 Funções da biblioteca do Arduino (*continuação*)

Comando	Exemplo	Descrição
delayMicroseconds	**delayMicroseconds(100000);**	Retardo de 100.000 microssegundos. Observe que o retardo mínimo é 3 microssegundos e o máximo é em torno de 16 milissegundos.
	Interrupções (veja o Capítulo 3)	
attachInterrupt	**attachInterrupt(1, myFunction, RISING);**	Associa a função **myFunction** (minha função) à ocorrência de uma transição de subida (RISING) na interrupção 1 (D3 em um Uno).
detachInterrupt	**detachInterrupt(1);**	Desabilita interrupções no pino 1 de interrupção.