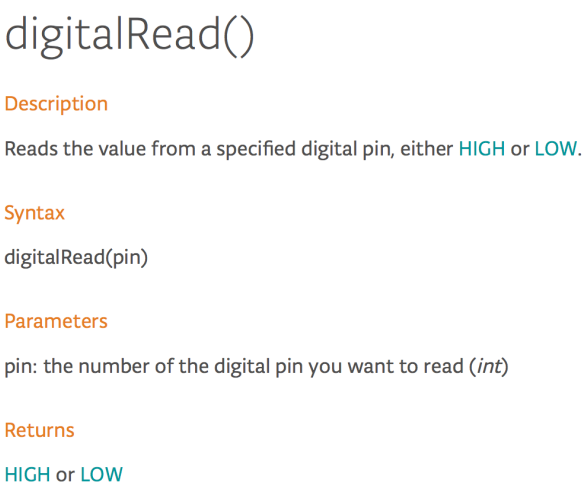
Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Automação e Sistemas
Engenharia de Controle e Automação
Introdução à Informática para Controle e Automação - DAS 5334

Aula 5 — Uso de Funções[1]

1 Funções

Uma função é uma sequência de comandos para a qual se dá um nome e que realiza uma computação. Já usamos diversas funções, por exemplo, `delay()`, `pinMode()`, `digitalRead()`, e `digitalWrite()`. Essas funções já estão definidas, isto é, já tiveram especificado o nome da sequência de comandos e, como vimos, a computação realizada está documentada na página de referência da linguagem Arduino. Veja que nesses casos não nos interessou saber quais os comandos da função, mas apenas o que ela faz e como usá-la. Em todos os nossos programas também modificamos ou estabelecemos a sequência de comandos para duas funções, `loop()` e `setup()`, que são necessárias em todo o programa para Arduino. No decorrer da disciplina conheceremos mais funções e aprenderemos, inclusive, a criar novas funções.

Para vermos mais detalhes do que constitui uma função, considere a documentação apresentada para a função `digitalRead()` na documentação do Arduino:



digitalRead()

Description

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax

digitalRead(pin)

Parameters

pin: the number of the digital pin you want to read (*int*)

Returns

HIGH or LOW

Podemos identificar os seguintes elementos que constituem uma função:

- um nome: `digitalRead` no exemplo;
- parâmetro(s): no exemplo o parâmetro é identificado por `pin` e é do tipo inteiro;
- uma computação: desconhecida por nós no caso do `digitalRead`; e
- opcionalmente, um retorno: no caso do `digitalRead`, o retorno é o valor no pino indicado; funções como `setup()` e `delay()` não tem retorno.[2]

Quando usamos uma função, dizemos que a função foi chamada. Chamadas de funções consistem tipicamente do nome da função seguido de argumentos entre parênteses. Os argumentos são expressões e devem combinar com o tipo do parâmetro definido para a função. Nem toda função exige argumento(s). Nem toda função devolve um valor. Por exemplo:

```
1  delay(250*4);
2  int nivel;
3  nivel = digitalRead(A3);
```

[1] Baseado em conteúdo do livro Think Python 2nd Edition by Allen B. Downey e conteúdo da página oficial do Arduino.

[2] Em algumas linguagens nomes como rotina, subrotina ou procedimento são preferidos ao se referir a funções computacionais que não retornam um valor.

Na linha 1 é feita uma chamada à função `delay()`. Essa chamada da função usa como argumento o resultado da expressão `250*4`, que é um número inteiro, e pára a execução do programa pelo número de milisegundos passado como argumento. No exemplo, o programa ficaria parado por 1000 milisegundos (1 segundo) antes de seguir para a próxima instrução. A função `delay()` não devolve nenhum resultado. Já na linha 3, a função `digitalRead()` usa como argumento o pino do qual se deseja a leitura. Essa função lê o valor do pino correspondente e devolve esse valor (`HIGH` ou `LOW`). Como a função devolve um valor, ela pode ser usada numa expressão aritmética e podemos, portanto, atribuir o resultado a uma variável, como na linha 3. Vimos, na Aula 2 o uso de funções que devolvem valores em expressões booleanas.

Na aula de hoje veremos mais algumas funções do Arduino disponíveis na página de referência da linguagem. Em particular, veremos como usar entradas analógicas, saídas PWM e algumas funções matemáticas/trigonométricas.

2 Entradas analógicas

Na Aula 2 já tratamos de entradas e saídas, mas nos concentramos nas entradas e saídas digitais. A placa Arduino Micro é dotada de 12 pinos que podem ser usados como entradas analógicas. Isto significa que ao invés de obtermos leituras do tipo `HIGH` ou `LOW`, podemos obter leituras de valores de tensão entre 0 V e 5 V. Isso é muito conveniente, pois diversos dispositivos como, por exemplo, sensores, fornecem sinais nessa faixa de valores. Atenção apenas para o fato que a placa Arduino Micro é um dispositivo digital. Assim, a leitura de valores analógicos é feita por meio de um conversor analógico digital (A/D). Isso significa, no caso do Arduino Micro, que poderemos obter 1024 valores entre 0 V e 5 V. Isto implica uma resolução de aproximadamente 0,49 mV (mais do que suficiente para nossos propósitos).

A Figura 1 mostra a pinagem do Arduino. Os pinos `A0` a `A5` e os pinos `A6` a `A11` podem ser usados como entradas analógicas. Notar que os pinos `A6` a `A11` são compartilhados com os pinos digitais. Para usar esses pinos como entrada analógica, não é necessário configurá-los, como ocorria no caso digital. De fato, assumiremos, pelo menos por enquanto, que os pinos podem assumir apenas uma função em cada aplicação para evitar erros.

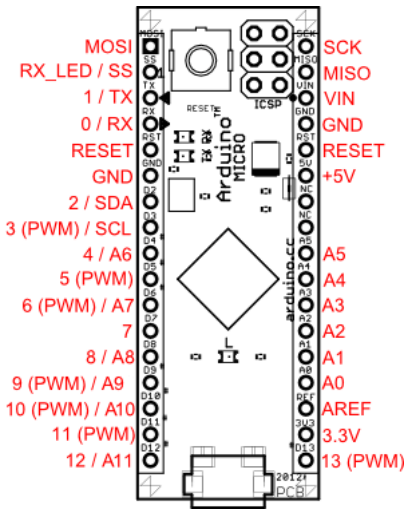


Figura 1: Pinagem da placa Arduino Micro (Fonte: Arduino).

A obtenção de valores analógicos é feita de maneira bastante conveniente pelo uso da função `analogRead()`. Essa função recebe apenas um parâmetro, a identificação do pino do qual se deseja obter a leitura, e devolve um valor inteiro entre 0 e 1023. A tensão no pino pode ser obtida, por exemplo, por regra de três.

```
1  int   valor  = 0;
2  float tensao = 0;
3
4  valor  = analogRead(A0);
5  tensao = (5.0/1024)*valor;
```

Nas linha 1 e 2 declaramos duas variáveis, uma do tipo `int` e uma do tipo `float`. Na linha 4 fazemos a leitura do valor do pino analógico de interesse e o atribuímos à variável `valor`. Na linha 5 transformamos o valor inteiro obtido em tensão. Veja que foi usado o número 5.0 ao invés de 5 para assegurar uma divisão real.

3 Saídas PWM

A placa Arduino Micro não possui saídas analógicas propriamente ditas. Porém, possui saídas PWM (*Pulse-width Modulation*). Os pinos identificados na Figura 1 com `(PWM)` podem ser usados para gerar ondas quadradas com os valores 0 V e 5 V em que a largura do pulso é controlada pelo programa. Apesar de as ondas quadradas geradas não serem de fato um sinal analógico, podem ser usadas para controlar a velocidade de um motor, o brilho de um LED e até mesmo gerar um sinal de tensão analógico por meio de eletrônica apropriada conectada ao respectivo pino. Notar na figura que o texto `(PWM)` aparece sempre ao lado da identificação do pino correspondente à operação digital. Assim, para usar o pino 9 como saída PWM, deve-se usar o identificador `9` como argumento da função `pinMode()` e não `A9`, isto é, `pinMode(9,OUTPUT)`.

Um sinal PWM é gerado pela função `analogWrite()`. Essa função recebe como parâmetro o pino desejado (identificador para operação digital) e o valor da razão cíclica (*duty cycle*), isto é, a largura do pulso em relação a um período da onda quadrada. Esse último é um valor inteiro entre 0 e 255.

```
1 valor = analogRead(A0);
2 analogWrite(9, valor / 4);
```

Na linha 1 um valor analógico é lido do pino `A0`. Na linha 2 o valor lido é dividido por 4 e é usado para gerar uma onda quadrada (PWM) no pino `9`. Como `valor` é um número inteiro entre 0 e 1023, a razão cíclica da onda quadrada, nesse exemplo, é proporcional à tensão no pino `A0`. Ou seja, se a tensão no pino `A0` for 5 V, o valor obtido será 1023 e a largura do pulso será dada por 255, o que equivale a uma razão cíclica de 100% e a um sinal sempre em 5 V. Se o a tensão no pino for 2,5 V, o valor obtido será 127, o que equivale a uma razão cíclica de 50% e a um sinal que fica metade do período em 5 V e metade do período em 0 V.

4 Funções matemáticas e trigonométricas

Além dos operadores matemáticos que permitem realizar operações como adição e subtração, a linguagem Arduino oferece diversas funções matemáticas que podem ser úteis na elaboração de programas em Arduino. Todas essas funções estão documentadas na página de referência da linguagem Arduino. Por comodidade, a seguir algumas delas são descritas com exemplos.

As funções `min()` e `max()` recebem dois números como argumentos e devolvem, respectivamente, o menor e o maior deles. Se estivermos monitorando um processo com dois sensores de temperatura em que os valores foram armazenados nas variávies `temp1` e `temp2`, podemos querer saber o menor ou o maior valor medido:

```
1 float temperatura;
2 temperatura = min(temp1, temp2); //temperatura = max(temp1, temp2);
```

A função `abs()` recebe um número como argumento e devolve o seu valor absoluto, isto é, o seu módulo. Por exemplo:

```
1 int x;
2 x = abs(-4);
```

No exemplo, o valor da variável `x` após o comando de atribuição da linha 2 é 4.

A função `constrain()` permite forçar que um número esteja dentro de uma faixa de valores. Recebe como argumentos o número que se deseja limitar, o limite mínimo e o limite máximo da faixa, e devolve o número limitado. Por exemplo:

```
1 int x;
2 x = constrain(1,-4, 8);
```

No exemplo, o valor da variável `x` após o comando de atribuição da linha 2 é o próprio 1, uma vez que está na faixa de valores estabelecidos ($-4 \leq 1 \leq 8$).

A função `map()` permite mapear um valor em uma faixa de valores para uma nova faixa de valores. Recebe como argumentos o número que deve ser mapeado, os limites mínimo e máximo da faixa atual, e os limites mínimo e máximo da faixa desejada. Devolve o valor mapeado. Por exemplo:

```
1 int x;
2 x = map(-4, -8, 0, 0, 8);
3 int val = analogRead(0);
4 val = map(val, 0, 1023, 0, 255);
```

No exemplo, o valor da variável `x` após o comando de atribuição da linha 2 é 4 e o valor de `val` é "reescalonado" da faixa 0–1024 (vindo da leitura analógica) para a faixa 0–255.

A função `pow()` recebe como argumentos dois números, base e expoente, e devolve o resultado da exponenciação. Por exemplo:

```
1  int x;
2  x = pow(2,4);
```

No exemplo, o valor da variável `x` após o comando de atribuição da linha 2 é 16.

A função `sqrt()` recebe um número e devolve a raiz quadrada desse número. Por exemplo:

```
1  int x = sqrt(9);
```

No exemplo, o valor da variável `x` após o comando de atribuição da linha 2 é 3.

As funções `sin()`, `cos()`, e `tan()` recebem como argumento o ângulo em radianos e devolvem, respectivamente, o seno, o cosseno e a tangente do ângulo. Por exemplo:

```
1  int x = sin(3.1415);
```

No exemplo, o valor da variável `x` após o comando de atribuição da linha 2 é 0.

5 *Debugging*

Já vimos vários aspectos da depuração de programas (tipos de erro, execução passo a passo, etc.). Uma forma simples de acompanhar a execução de um programa e detectar um erro é simplesmente mostrar no display, saída serial, ou num LED o valor de uma variável. Por exemplo, podemos acompanhar a evolução do valor da variável `ultimoValorA3` mostrando seu valor na saída serial junto com algumas mensagens indicando por onde a execução passou.

```
1  ....
2  int ultimoValorA3 = LOW;
3
4  void loop() {
5    lcd.clear();
6
7    Serial.print(ultimoValorA3);     // <-- mensagens de debug
8    Serial.print(" != ");            //     para ver os valores
9    Serial.println(digitalRead(A3));   //     envolvidos no if que segue
10
11   if (ultimoValorA3 != digitalRead(A3)) {
12     Serial.println(" ** mudou    "); // <-- mensagem de  debug
13     ....
14   } else {
15     Serial.println(" ** não mudou"); // <-- mensagem de  debug
16     ....
17   }
18   ....
19 }
```

6 Problemas

A seguir resolveremos os problemas desta aula. Acesse o circuito da Estufa e faça uma cópia em sua própria conta com um clique no botão `Copiar e Tinker`. Esse circuito (Figura 2) é bastante simples comparado com o circuito da aula anterior. Além do *display*, são três LEDs amarelos conectados aos pinos `6`, `9` e `10` da placa e um potenciômetro conectado ao pino `A8`. Notar que os três pinos aos quais os LEDs estão conectados podem ser usados no modo PWM. O potenciômetro permite ajustar o nível de tensão no pino `A8` entre 0 e 5 V, o que corresponde à leitura de valores de 0 a 1023 com o uso de `analogRead()`.

Problema 1

A produção de alimentos a partir da década de 1970 passou a depender fortemente do desenvolvimento científico e tecnológico. O uso de estufas tem papel fundamental na produção de diversos alimentos como, por exemplo, o tomate. O crescimento saudável de uma planta depende fortemente de iluminação. De fato, a luz afeta o desenvolvimento da planta pela sua intensidade, sua qualidade e sua quantidade. A intensidade de luz define a qualidade da planta, por exemplo, em termos de nutrientes que carrega. A qualidade da luz refere-se ao espectro luminoso e afeta a morfologia da planta. Por fim, a

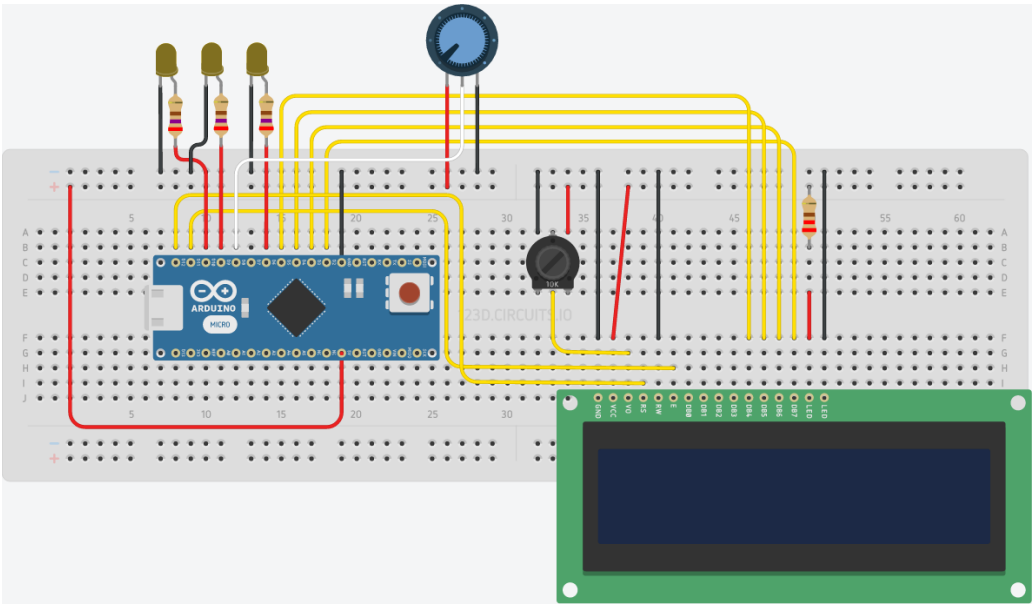


Figura 2: Circuito da Aula 5.

quantidade de luz, isto é, o número de horas que a planta é exposta à luz, afeta a geração de flores e frutos.

Neste exercício, vamos considerar a luminosidade ideal como sendo a linha cheia da Figura 3. Há períodos onde o ideal é não ter luz (às 24 horas) e períodos onde o ideal é ter luminosidade máxima (às 12 horas). Nos outros momentos, a quantidade de luz ideal é determinada pela curva vista na figura. Porém, em função de nebulosidade, a quantidade de luz solar disponível pode ser reduzida. Na figura, a linha tracejada indica um exemplo em que a quantidade de luz solar disponível é inferior à ideal em função da nebulosidade. Na estufa, vamos instalar uma lâmpada para compensar a falta de luz solar. A potência dessa lâmpada deve ser tal que a soma entre a luz solar disponível e a luz emitida pela lâmpada seja a luminosidade ideal.

Para simular essa situação no circuito desta aula, usaremos um potenciômetro para indicar o "nível" de nebulosidade, um LED para representar a luminosidade ideal, um LED para representar a luminosidade com nebulosidade e um LED para representar a lâmpada. O programa final deve compensar a luz solar alterando a potência do LED em função da nebulosidade. O problema é relativamente complexo e vamos resolvê-lo em partes.

a) Simular a passagem do tempo (eixo da abcissa no gráfico). Cada execução da função `loop()` deve contar como a passagem de uma hora. Dicas: ver variável acumuladora e usar a técnica de debug desta aula para companhar o valor desta varável. Os valores dessa variável no decorrer da execução devem ser: 0, 1, 2, 3, ... 23, 0, 1, 2, 3, ... 23, 0, ...

b) Calcular a luminosidade ideal conforme a hora do dia e acionar o LED correspondente. Dica: usar a função `sin()` para dar o formato da curva. A função `map()` funciona só com inteiros, o que não é adequado nesse problema. Uma possibilidade seria implementar uma função equivalente que funcione com números reais:

```
float rescale(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Lembre que o argumento da função seno deve estar entre 0 e 2π, enquanto a hora está entre 0 e 23, e seu resultado, que deve estar na escala 0 a 100% de potência para o LED. A Tabela 1 tem os valores calculados para esse LED nas horas do dia.

c) Calcular a luminosidade atual, que é influenciada pelo potenciômetro, e acionar o LED correspondente. Dica: usar também a função seno com um fator de redução dado pelo potenciômetro.

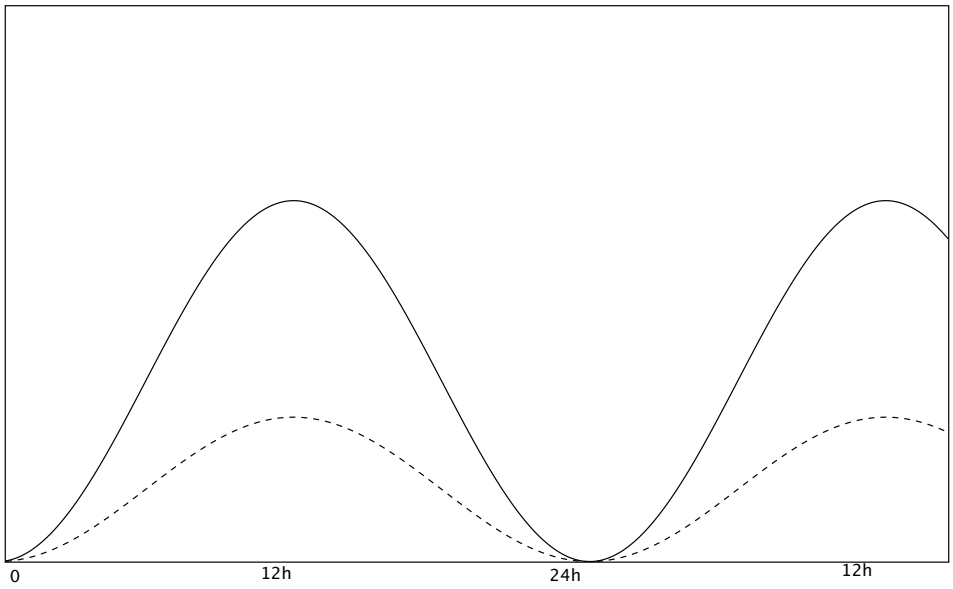


Figura 3: Comportamento da luminosidade no tempo

d) Calcular a potência a ser atribuída ao LED que corresponde à lâmpada, que deve ser igual à diferença entre a luminosidade atual e a ideal, e acionar o LED correspondente.

e) Altere o programa para atuar na luminosidade a cada minuto (no lugar de a cada hora).

hora	potência do LED (em %)
0	0.0
1	1.703 708 685
2	6.698 729 810
3	14.644 660 940
4	25.0
5	37.059 047 744
6	50.0
7	62.940 952 255
8	75.0
9	85.355 339 059
10	93.301 270 189
11	98.296 291 314
12	100.0
13	98.296 291 314
14	93.301 270 189
15	85.355 339 059
16	75.0
17	62.940 952 255
18	50.0
19	37.059 047 744
20	25.0
21	14.644 660 940
22	6.698 729 810
23	1.703 708 685

Tabela 1: Valores para o LED “ideal” no decorrer do dia.