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 10 — Bibliotecas e Objetos[1]

1 Bibliotecas

A linguagem Arduino se resume praticamente àquilo que consta na página de referência da linguagem. Sim, com as informações dessa página é possível fazer praticamente tudo em Arduino. Além disso, já vimos as principais estruturas de programação da linguagem que são, de certa forma, comuns a diversas linguagens de alto nível e que, com um breve ajuste de sintaxe, nos permitirão programar nessas outras linguagens com pouco esforço de aprendizado.

O aluno dedicado que já navegou pelas páginas de referência da linguagem deveria se perguntar nesse instante: "E os comandos ou funções usados com o *display LCD* ou com os *NeoPixels*? O que são?" De fato, eles não constam na página de referência da linguagem e mesmo assim os usamos como se fizessem parte da linguagem Arduino.

A linguagem Arduino, como diversas outras linguagens, oferece a comodidade de estender o seu conteúdo por meio de bibliotecas. Bibliotecas, sem muito rigor na definição, nada mais são que coleções de funções, variáveis, constantes, e etc., geralmente com um propósito bem claro, programadas com a própria linguagem. As bibliotecas promovem a reutilização do código e sua organização.

Temos usado bibliotecas desde a primeira aula. O uso de bibliotecas é feito por meio da diretiva `include` seguida do nome da biblioteca entre os símbolos de menor (<) e maior (>). A diretiva `include` é usada sempre no início do programa e essencialmente corresponde a copiar o conteúdo do arquivo da biblioteca para dentro do nosso programa. Por exemplo:

```
1  #include <LiquidCrystal.h>
2  #include <Adafruit_NeoPixel.h>
```

A listagem acima inclui duas bibliotecas, a primeira, `LiquidCrystal`, permite que usemos o *displayLCD*, e a segunda, `Adafruit_NeoPixel`, permite que usemos os *NeoPixels*. Notar que cada uma delas atende uma necessidade bem específica.

Há diversas bibliotecas disponíveis para uso com o Arduino, várias delas listadas na página de bibliotecas do Arduino, outras disponibilizadas por fabricantes de dispositivos ou hobbystas. Vocês descobrirão novas bibliotecas conforme a necessidade, por exemplo, como nós descobrimos a biblioteca `Adafruit_NeoPixel` em função do nosso interesse por *NeoPixels*. Podemos também criar novas bibliotecas. Porém, a criação de bibliotecas foge dos objetivos desta disciplina.

2 Objetos

Muitas bibliotecas usam a noção do *objetos*. Assim como valores de um tipo (e.g. 10 é um valor do tipo pré-definido na linguagem, o tipo `int`), objetos são valores de tipos definidos por programadores (chamados de *classes*). Porém, além do valor, objetos têm operações associadas (chamadas de *métodos*). Classes, objetos e métodos são conceitos de programação orientada a objetos, um assunto que não é visto nesta disciplina. Nos limitaremos aqui ao *uso* de classes definidas em bibliotecas.

Por exemplo, considerando o seguinte programa

```
1  Adafruit_NeoPixel strip1 = Adafruit_NeoPixel(7, PIN1, NEO_GRB + NEO_KHZ800);
2  Adafruit_NeoPixel strip2(16, PIN2, NEO_GRB + NEO_KHZ800);
3  ...
4  strip1.begin();
5  strip1.show();
6  uint32_t magenta = strip1.Color(255, 0, 255);
```

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

```
7  ...
8  strip2.begin();
9  strip2.show();
10 ...
```

na linha 1, um objeto da classe `Adafruit_NeoPixel` é criado. Esse objeto tem várias informações sobre o estado do *NeoPixel*, como o pino usado para ligá-lo ao Arduino e o número de *pixels*. A variável `strip1` é uma referência para o lugar de memória onde o objeto está localizado. Podemos notar que a sintaxe da linha 1 é a mesma da declaração de variáveis: um tipo seguido de uma variável seguido de '=' seguido de um valor. Neste caso, o valor é mais complexo que os valores dos tipos primitivos como `int` e `float`. A linha 2 utiliza uma sintaxe abreviada para criar outro objeto e guardar uma referência a ele na variável `strip2`.[2]

Nas linhas 4 em diante, métodos destes objetos são chamados (de forma muito simular ao uso de funções – podemos pensar em métodos como "funções próprias de um objeto"). O ponto separando a variável do método indica que o método do objeto referenciado pela variável será chamado. Em `strip1.show()` o método `show()` do objeto representado por `strip1` é chamado. Já em `strip2.show()` o método `show()` do objeto representado por `strip2` é chamado.

3 Comunicação Serial e tipo Char

Além de botões e potenciômetros, o Arduino pode interagir com o "mundo externo" (i.e. fazer I/O) por meio de *comunicação serial*. No IDE do Circuits.io, o `Serial Monitor` na janela do `Code Editor`, que permite tanto ler quando enviar dados via porta Serial.

A comunicação serial pode ser usada tanto para a comunicação entre duas placas Arduino como entre um placa de computador. A conexão USB também pode ser usada para a comunicação serial com o computador. É importante que os pinos `RX` (`D0`) e `TX` (`D1`) não estejam sendo usados para outro propósito.

No caso do Arduino Micro, a comunicação serial é realizada por um objeto `Serial`. Os métodos disponíveis estão detalhados na página sobre Serial e os mais básicos são `available()`, `begin()`, `print()`, `println()`, e `read()`. A listagem a seguir exemplifica o uso desses métodos:

```
1  void setup()
2  {
3    Serial.begin(9600);
4  }
5
6  void loop()
7  {
8    if (Serial.available()){
9      char c = Serial.read();
10     Serial.print("Foi digitado: ");
11     Serial.println(c);
12   }
13 }
```

Esse programa escreve na serial tudo o que é lido nela. A linha 3 configura a serial para comunicação a uma taxa de 9600 bps. Não precisamos nos preocupar com os detalhes da velocidade de comunicação. A serial contém um *buffer*, isto é uma área e armazenamento capaz e guardar alguns caracteres. Na linha 8, o método `available()` devolve verdadeiro se houver algum dado disponível no *buffer*. O método `read()` na linha 9 lê um caractere do *buffer* e atribuiu a uma variável do tipo `char`. Na linha 10, usamos o método `print()` para escrever um texto na serial e na linha seguinte, usamos o método `println()` para escrever o valor do caractere armazenado na variável `c` seguido de uma quebra de linha.

O tipo `char` (da linguagem C) utiliza um byte de memória e representa um carácter, uma "letra". Os valores literais para esse tipo são escritos entre aspas simples, por exemplo `'a'`. Internamente a linguagem converte essas letras em números conforme a tabela ASCII. Veja mais detalhes na documentação do Arduino.

[2] Essa sintaxe abreviada foi utilizada nas aulas anteriores para criar o objeto `lcd`.

4 *Debugging*

Nas aulas passadas vimos as vantagens de usar LEDs ou *display* como recursos para *debugging* de programas, além do próprio `Debugger`. Outro recurso interessante para *debugging* no *Arduino* é o `Serial Monitor`.

5 Problemas

Problema 1

Estude o programa usado em https://tinkercad.com/things/d0n34kXi8O8 para adotarmos uma solução similar no relógio feito nas aulas anteriores.
Nessa nova implementação do relógio, o usuário deve poder digitar comandos no serial monitor para atualizar a hora e controlar cronometros conforme segue.

a) Quanto o usuário digitar `s1`, o programa deve iniciar um novo cronometro, identificado por `1`. No lugar do número 1, outro valor pode ser digitado.

b) Quando o usuário digita `g1` aparece no serial monitor o tempo decorrido desde o comando `s1`.

c) Quando o usuário digita `p1` o cronometro 1 para de contar o tempo. O comando `s1` é utilizado para reiniciar a contagem do tempo no cronometro 1.

d) Quando o usuário digita `hHH:MM` a hora atual do relógio muda conforme indicado. No caso, HH e MM são números com dois digitos indicando as horas e minutos, respectivamente.

Problema 2

Estude a biblioteca Date Time e identifique onde ela poderia simplificar a implementação do relógio. Considere também as várias bibliotecas de "time" disponíveis em aqui.