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 2 — Entradas, Saídas e Operadores[1]

1 Entradas e Saídas

Em sistemas de automação e controle, pouco poderemos fazer se não recebermos informações da planta que queremos controlar ou automatizar. Mais especificamente, precisamos de meios para obter e fornecer dados para dispositivos da planta para que esta opere adequadamente. Essencialmente estamos interessados em poder operar com dados de sensores e intervir no sistema por meio de atuadores, além, é claro, do sistema de supervisão e monitoração.

Nos computadores estamos tão habituados com os dispositivos de entrada e saída que muitas vezes nem nos damos conta que as trocas de informações acontecem por meio de portas de entrada e saída no computador. Isso fica ainda mais mascarado pelo fato de que muitos dispositivos funcionam *wireless* via *Wi-Fi*, como impressoras, ou via Bluetooth, como teclados e mouses. Todos esses são dispositivos de saída ou entrada que ainda podem ser encontrados no mercado, por exemplo, com cabos USB para conectá-los ao computador. O monitor é um dispositivo de saída que em *notebooks* e nos, cada vez mais comuns, *All-in-one Desktops* já vem integrado ao computador, mas em *Desktops* convencionais ainda é conectado ao computador por meio de um cabo e a respectiva porta HDMI ou VGA.

No Arduino a maneira mais direta de receber ou enviar dados para outros dispositivos é por meio dos seus pinos de entrada e saída. A placa Arduino Micro possui ao todo 20 pinos que podem ser usados como entrada ou saída digitais, dos quais 12 podem ser usados como entrada ou saída analógicas. Você certamente já ouviu falar sobre eles nas aulas de introdução à engenharia.

Os pinos de entrada e saída da placa Arduino Micro são identificados de `D0` a `D13` para os digitais e de `A0` a `A11` para os analógicos, com alguma superposição entre eles, lembrando que os analógicos também podem ser usados como pinos digitais. Para reconhecer os pinos na placa no Tinkercad Circuits, basta passar o mouse sobre os contatos (ver Figura 1). Ao passar o mouse você verá que em alguns pinos aparece mais de uma identificação. Por exemplo, os pinos `D0` e `D1` também possuem a identificação `RX` e `TX`, respectivamente, pois podem ser usados para comunicação serial.

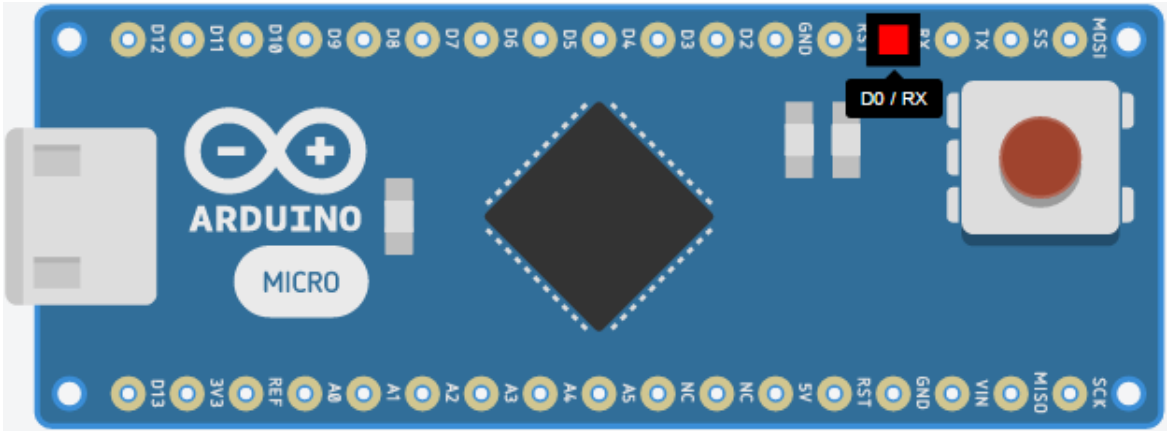

Figura 1: Placa Arduino Micro no Tinkercad Circuits com o mouse sobre o pino `D0 / RX`.

Já usamos os pinos de entrada e saída da placa logo na primeira aula e para a resolução do nosso primeiro problema. Quando fizemos `LiquidCrystal lcd(12, 11, 5, 4, 3, 2);` na Aula 1, estávamos configurando os pinos 12, 11, 5, 4, 3 e 2 para uso com o *display* LCD. Não nos interessa ainda entender como esses pinos foram usados para configurar o *display*, pois na aula de hoje usaremos os pinos individualmente. Nos ateremos, por enquanto aos pinos como entradas ou saídas digitais.

O primeiro passo para usar um pino da placa do Arduino como entrada ou saída digital é configurá-lo como tal. Para isso usaremos a função `pinMode()`. Essa função recebe entre parênteses dois valores separados por

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

vírgula. O primeiro valor deve ser um número de 0 a 13, correspondendo a um dos 14 pinos digitais (D0 a D13), ou um código de A0 a A5, correspondendo a um dos 6 pinos analógicos. O segundo valor deve ser a palavra INPUT, se quisermos usar o pino como entrada, ou OUTPUT, se quiseremos usar o pino como saída. Lembre-se que as letras maiúsculas devem ser respeitadas. Por exemplo:

```
1 pinMode(7, OUTPUT);
2 pinMode(A3, INPUT);
```

Na linha 1 do exemplo o pino D7 foi configurado como saída e na linha 2 o pino A3 foi configurado como entrada.

Os pinos digitais aceitam apenas dois valores, também chamados de níveis: alto ou baixo. O valor alto na linguagem Arduino é escrito como HIGH e o valor baixo como LOW. Para colocarmos um valor alto ou baixo num pino configurado como saída usaremos a função digitalWrite(). Essa função recebe entre parênteses dois valores separados por vírgula. O primeiro valor é o número/código do pino que queremos alterar o nível. O segundo valor é o nível desejado para a saída, HIGH ou LOW. Por exemplo, digitalWrite(7, HIGH); coloca o pino D7 em nível alto.

Para obtermos o valor de um pino de entrada usaremos a *função* digitalRead(). Essa função recebe entre parênteses o número/código do pino que queremos saber o nível e fornece o nível no pino. Por exemplo, digitalRead(A3); fornece o nível no pino A3. Ou seja, quando executada, essa função corresponde ao valor HIGH ou LOW, conforme o nível vigente no pino A3. Por ser uma função, é possível usar diretamente o valor obtido numa saída. Por exemplo, se quisermos que o nível no pino de saída D7 seja o mesmo que o nível obtido do pino de entrada A3, basta escrever digitalWrite(7, digitalRead(A3));.

As palavras HIGH e LOW são, na verdade, representações para os números 1 e 0, respectivamente. Para fins de computação, os valores 1 e 0 são usados e não as palavras HIGH e LOW, que existem apenas para facilitar a leitura do programa.

2 Comandos

Comandos foram mencionados superficialmente no Problema 1 da Aula 1. Comandos são unidades de código que têm um efeito. O uso da função lcd.print() pode ser visto como um comando de apresentação, em que o efeito é o valor (ou o resultado da expressão) entre parênteses ser mostrado no *display* LCD. Comandos são sempre encerrados com ponto e vírgula ";".

Caso prefira, o **problema 1** pode ser resolvido com o conteúdo apresentado até aqui.

3 Expressões booleanas

Expressões são a combinação de valores e operadores para produzir novos valores. Assim, toda a parte de código que admite um valor, também admite uma expressão. Por exemplo, a função lcd.print() usada na Aula 1, recebe um valor entre parênteses e lcd.print(5);, mostra o número 5 no *display* LCD. No Problema 2 da Aula 1 fizemos o uso de *expressões aritméticas* ao invés de valores para mostrar o resultado das operações aritméticas, por exemplo, lcd.print(37 + 3);.

Enquanto expressões aritméticas produzem um valor numérico quando avaliadas, expressões booleanas produzem o valor verdadeiro ou o valor falso como resultado. Em Arduino a palavra true é usada para verdadeiro e a palavra false é usada para falso. A palavra true também corresponde ao valor 1 e a palavra false ao valor 0. Ou seja, o valor 0 pode ser usado no lugar de false e o valor 1 pode ser usado como true. Além disso, qualquer valor diferente de zero, mesmo que negativo, também é avaliado como true. Não há motivos para assustar-se com a palavra "booleana" que não é nada mais do que uma homenagem ao matemático George Boole que introduziu o sistema algébrico que hoje leva seu nome, a Álgebra do Boole.

Lembre-se que muitos dispositivos usados em controle e automação operam também com apenas dois valores. Por exemplo, sensores de fim de curso, sensores de presença, alguns tipos de válvulas, alguns tipos de pistões pneumáticos, etc.

3.1 Operadores de comparação

Os operadores de comparação permitem comparações entre expressões, isto é, permitem estabelecer relações numéricas. Por isso são também chamados de *operadores relacionais*. Lembrar que um valor também é uma expressão. A utilização desses operadores é muito parecida com o que é feito na matemática, mas os símbolos não são exatamente iguais. Os operadores de comparação são: igual (==), diferente (!=), maior (>), menor (<),

maior ou igual (>=), e menor ou igual (<=). Como no caso dos operadores matemáticos, cada operador exige dois operandos. O resultado de uma operação relacional é sempre `true` ou `false`. Por exemplo:

```
1  5 == 7
2  3 != 2
3  5 > 3
4  5 < 3
5  4 >= 2
6  3 <= 3
```

O resultado das expressões nas linhas 1 a 6 são, respectivamente, `false`, `true`, `true`, `false`, `true`, e `true`.

Deve-se ter atenção especial no uso do operador de comparação igual. O correto é dois símbolos "=" (ver linha 1 na listagem acima), pois o uso de apenas um símbolo tem outro significado que veremos em outra aula.

3.2 Operadores booleanos

Os operadores booleanos, também chamados de *operadores lógicos*, permitem estabelecer relações lógicas entre valores verdadeiro ou falso. Os três operadores que usaremos são: E lógico (`&&`), OU lógico (`||`), e NEGAÇÃO (`!`). Os operadores "`&&`" e "`||`" exigem dois operandos e o operador "`!`" apenas um operando. Os operandos são os valores `true` ou `false`, ou valores numéricos como descrito acima, ou expressões. O resultado sempre é `true` ou `false`. A operação "`&&`" só resulta verdadeira se os dois operandos forem verdadeiros. A operação "`||`" resulta verdadeira se pelo menos um dos operandos for verdadeiro. A operação "`!`" devolve a negação do operando, isto é, inverte o valor (`true` vira `false` e `false` vira `true`). Por exemplo:

```
1  true && false
2  -1 && true
3  false || true
4  false || 0
5  ! true
```

O resultado das expressões nas linhas 1 a 5 são, respectivamente, `false`, `true` (já que `-1` é interpretado como `true`), `true`, `false` (já que `0` é interpretado como `false`), e `false`.

É comum a representação de expressões formadas por operadores lógicos por tabelas verdades. Em um tabela verdade, as combinações possíveis de valores para uma determinada expressão são organizados em colunas. E cada expressão tem sua própria coluna com o resultado da avaliação. A seguir são apresentadas as tabelas verdades das três operações descritas anteriormente:

Tabela 1: Tabela verdade do operador lógico `||` (OU)

Valor 1	Valor 2	Valor 1 `||` Valor 2
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`

Tabela 2: Tabela verdade do operador lógico `&&` (E)

Valor 1	Valor 2	Valor 1 `&&` Valor 2
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

Tabela 3: Tabela verdade do operador lógico `!` (NEGAÇÃO)

Valor	!(Valor)
`true`	`false`
`false`	`true`

Lembrar que como o valor de zero equivale a false e qualquer valor diferente de zero equivale a true número poderiam ser igualmente utilizados para compor as tabelas verdades, com a ressalva que em caso de resultado verdadeiro, o valor numérico é sempre 1. Por exemplo, no caso da negação:

Tabela 4: Tabela verdade do operador lógico ! (NEGAÇÃO), com números

Valor	!(Valor)
1	0
0	1

Tabelas verdade podem trabalhar com expressões mais complexas, que envolvam vários operadores e vários valores. Neste caso, pode ser interessante decompor a expressão em várias colunas antes de obter o valor final da expressão. Por exemplo:

Tabela 5: Tabela verdade da expressão lógica

Valor 1	Valor 2	Valor 3	Valor 1 && Valor 2	(Valor 1 && Valor 2) || Valor 3
true	true	true	true	true
true	true	false	true	true
true	false	true	false	true
true	false	false	false	false
false	true	true	false	true
false	true	false	false	false
false	false	true	false	true
false	false	false	false	false

Notar na tabela que a expressão da quarta coluna foi avaliada com base nos valores da primeira e da segunda coluna. Já a expressão da quinta coluna foi avaliada com base nos valores da terceira e da quarta coluna.

Caso prefira, os **problemas 2 e 3** podem ser resolvidos com o conteúdo apresentado até aqui.

4 Comentários

Mesmo programas simples podem ser difíceis de compreender, especialmente se foi escrito por outra pessoa ou se passou algum tempo desde que o escrevemos. Em função disso recomenda-se que um programa contenha comentários com informações úteis para a sua compreensão. Comentários são textos colocados no programa que são ignorados pelo compilador e não afetam o programa final. Deve-se evitar o uso exagerado de comentários para não "poluir" o código, em particular deve-se evitar comentários óbvios, como descrever um comando conhecido.

Em Arduino os comentários podem ser incluídos no código de duas maneiras. A primeira é com o uso de "*//*" que transforma em comentário tudo o que vier na sequência numa mesma linha. A segunda maneira é com o uso de "*/**/*" que transforma em comentário tudo que estiver entre os asteriscos, mesmo que isso envolva múltiplas linhas. Por exemplo:

```
1  /* Já usamos o este código para resolver
2  um problema da Aula 1, mas aqui acrescentamos
3  comentários.*/
4
5  #include <LiquidCrystal.h>
6  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
7
8  void setup() {
9    lcd.begin(16, 2); // Configura o número de linhas e colunas do LCD
10   lcd.print("Inicio operacao!");
11 }
12
13 void loop() {
14   lcd.clear();
15   lcd.print(37 + 3);
16   delay(1000);
17 }
```

Nas linhas 1 a 3 do exemplo, temos um comentário geral sobre o exemplo. Na linha 9, colocamos uma breve explicação sobre o que o comando faz, talvez desnecessária.

5 *Debugging*

Depreende-se da seção anterior que comentários são muito importantes para o *debugging* já que as informações neles contidas têm o papel de facilitar essa árdua tarefa. Mas a utilidade de comentários para *debugging* vai além disso. Comentários também podem ser usados para suprimir temporariamente partes de código usadas para teste, sem a necessidade de apagar o código e eventualmente ter que voltar a escrevê-lo mais tarde. Veja a seguir uma versão do exemplo da seção anterior:

```
1  #include <LiquidCrystal.h>
2  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
3
4  void setup() {
5    lcd.begin(16, 2); // Configura o número de linhas e colunas do LCD
6    lcd.print("Inicio operacao!");
7  }
8
9  void loop() {
10   //lcd.clear();
11   lcd.home();
12   lcd.print(37 + 3);
13   delay(1000);
14 }
```

Nas linhas 10 usamos um comentário de uma única linha para fazer com que o código naquela linha não tenha efeito. Na linha 11 introduzimos `lcd.home()`, por exemplo, para ver se o efeito no *display* é melhor do que usando `lcd.clear()`. Se o resultado for melhor, podemos apagar o conteúdo da linha 10, caso contrário apagamos o conteúdo da linha 11 e removemos o símbolo de comentário da linha 10. Comentários desse tipo podem ser mantidos por longos períodos num programa, até que não sejam mais necessários.

6 Problemas

A seguir resolveremos os problemas desta aula. Acesse o cicuito Estacionamento 1 e faça uma cópia em sua própria conta com um clique no botão `Copiar e Tinker`. Esse circuito (Figura 2) é um pouco mais sofisticado do que o Circuito Display, pois além do *display* LCD, conta com LEDs e chaves (*DIP switch*). São quatro LEDs vermelhos, três LEDs verdes, quatro chaves (caixinhas vermelhas com o centro branco). Os LEDs estão conectados aos pinos `D0`, `D1`, `D6`, `D7`, `D8`, `D9`, e `D10`, que devem ser configurados como `OUTPUT`. As chaves estão conectadas aos pinos `A0`, `A1`, `A2`, e `A3`, que devem ser configurados como `INPUT`. Há também um LED verde integrado à placa e conectado internamente ao pino 13, o qual deve ser configurado como `OUTPUT`. Os componentes eletrônicos do circuito estão conectados de tal maneira que um valor `LOW` num pino de saída apaga o respectivo LED e um valor `HIGH` acende o LED. As chaves como mostradas na Figura 2, mantêm nível `LOW` no respectivo pino de entrada, enquanto se estiverem na posição 1 (para cima) mantêm nível `HIGH`. Ao final do último problema teremos usado todos esses recursos por meio de nossos programas.

Problema 1

No Problema 3 da Aula 1 exibimos os números 1 e 2 alternadamente no *display* LCD para indicar que o sistema estava em funcionamento. Você deve estar lembrado que o enunciado dizia que isto era muito comum em sistemas de alarme, mas com luzes. Então que tal fazermos o mesmo com luzes desta vez? Usaremos dois LEDs, um verde e um vermelho, conectados às saídas digitais `D0` e `D1`, respectivamente. Esses pinos estão conectados aos dois LEDs mais à esquerda no circuito. Usaremos como ponto de partida a solução do Problema 3 da Aula 1:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
}

void loop() {
```

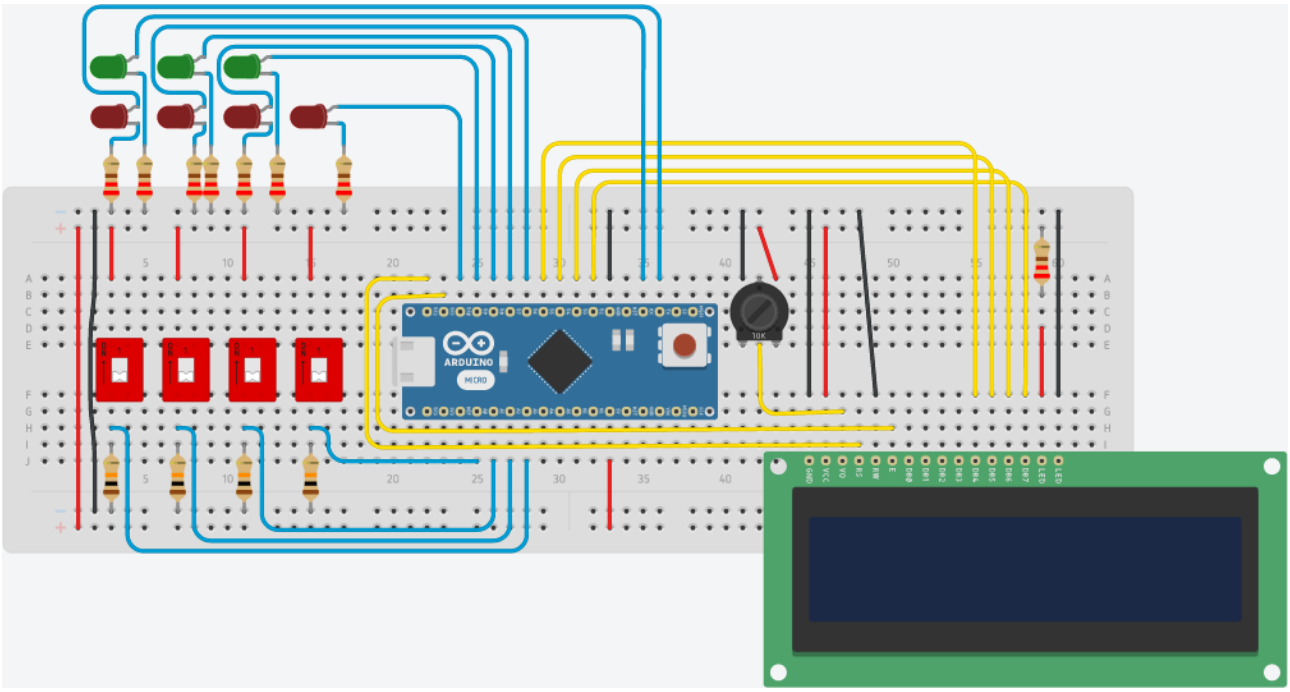

Figura 2: Circuito da Estacionamento 1.

```
    lcd.print("1");
    delay(1000);
    lcd.clear();
    lcd.print("2");
    delay(1000);
    lcd.clear();
}
```

Siga os seguintes passos para resolver o problema:

1. Configure os pinos `D0` e `D1` como saída. Antes de programar reflita sobre qual é parte do código em que serão inseridas essas instruções, na função `setup()` ou na função `loop()`?

2. Insira no código os comandos para acender e apagar os LEDS e remova os comandos relativos à escrita no *display* LCD.

Problema 2

A área de engenharia é recheada de normas de segurança. Por exemplo, não é permitido que robôs operem enquanto humanos estiverem em sua área de ação. Na indústria química, de alimentos ou de bioengenharia, máquinas não podem entrar em operação enquanto operadores humanos estiverem na sala para evitar danos ou contaminação.
Implemente um programa que usa uma chave para indicar a presença de um operador humano (indicada pela ativação da chave mais a esquerda). Se a chave estiver ativada, o LED vermelho deve acender. Se a chave não estiver ativada, o LED verde deve acender. Podemos usar os mesmos dois LEDs do Problema 1 e a chave conectada ao pino `A3`. Lembre-se de configurar o pino `A3` como entrada.
Dica: O operador booleano `!` pode ser útil para resolver esse problema: `!0 = 1` e `!1 = 0`.

Problema 3

Continuando o problema anterior, mostre no *display* o número de pessoas presentes considerando que cada chave ligada indica a presença de uma pessoa. As chaves estão conectadas aos pinos `A0`, `A1`, `A2`, e `A3`, que devem ser configurados como entrada. Notar que agora o LED vermelho deve acender se qualquer uma das chaves for ativada, enquanto o LED verde deve acender se nenhuma chave estiver ativada!
Para evitar que o *display* LCD fique piscando com as trocas de mensagem, ao invés de usar a função `lcd.clear()` use a função `lcd.home()`. Essa função reposiciona o cursor no alto à esquerda do *display* sem apagar o conteúdo.
Dica: Lembre que `digitalRead()` é uma função que retorna 1 para uma chave ligada e 0 caso contrário.

Problema 4

Ainda continuando o problema anterior, acenda todos os LEDs vermelhos caso o número de pessoas presentes seja maior ou igual a 2. Os LEDs vermelhos estão conectados aos pinos `D1`, `D6`, `D8`, e `D10`, que devem ser configurados como saída. Um operador relacional pode ser necessário.
Dica: Lembre que o resultado de operações relacionais é ou `true` ou `false`, que correspondem, respectivamente, aos valores 1 e 0.