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 4 — Variáveis[1]

Na Aula 1 programamos um sistema de monitoração bastante simples, na Aula 2 exploramos uma diversidade de pequenos problemas e, já na Aula 3, fizemos um sistema para estacionamentos. A despeito de qualquer dificuldade que você possa ter apresentado nessas duas aulas, deve ter percebido que ainda estamos lidando com problemas relativamente simples quando comparados a uma usina nuclear ou um avião, não é mesmo? Muito embora simples, as mesmas ideias e linhas de código parecidas serão usadas para resolver diversos outros problemas pelo resto de suas carreiras. Enquanto isso, você deve estar se perguntando como fazer para guardar valores obtidos a partir de expressões e comandos para posterior uso em outras expressões e comandos. Faremos isso por meio de variáveis.

1 Variáveis

Uma variável é um *nome* que se refere a um valor. Variáveis permitem que armazenemos na memória do computador valores para posterior uso. Assim, uma variável pode ser usada para guardar o valor obtido a partir da leitura de um sensor ou resultados intermediários de cálculos matemáticos. Mais do que isso, os valores das variáveis podem ser alterados ao longo do programa conforme a necessidade.

Os nomes das variáveis devem ser escolhidos adequadamente, de maneira que descrevam o seu papel no programa, de forma a ajudar na leitura do código e na própria documentação. Em Arduino, qualquer palavra que não seja uma palavra reservada da linguagem pode ser usada como nome de variável. Todas as palavras que aparecem na página de referência da linguagem Arduino são palavras reservadas. Tipicamente são usadas letras minúsculas, por exemplo, `led` como nome para uma variável relacionada a um LED. Nomes compostos são escritos com o auxílio de maiúsculas (`ledVerde`) ou de sublinhado (`led_verde`).

As variáveis precisam ser *declaradas* em um programa. A declaração de uma variável corresponde à definição do tipo de dado a que ela se refere e à definição do seu nome. Opcionalmente uma variável pode também ser inicializada na declaração, isto é, pode ter atribuído a ela um valor. A atribuição de um valor inicial durante a declaração, porém, não é obrigatória e pode ser feita em outra parte do programa, por exemplo, por ocasião do primeiro uso. O código a seguir apresenta a declaração e inicialização de duas variáveis:

```
1 int ledVerde = 13;
2
3 int ledVermelho;
4 ledVermelho = 12;
```

Na linha 1 foi declarada uma variável com o nome `ledVerde`, do tipo inteiro (`int`), à qual foi atribuído o valor inicial de 13. Já na linha 3, foi declarada uma variável com o nome `ledVermelho`, também do tipo inteiro, mas nenhum valor foi atribuído a ela na declaração. Só na linha 4 o valor 12 é atribuído à variável `ledVermelho`. A inicialização não precisa ser feita imediatamente após a declaração, pode ser feita em outra parte do programa, como acontece na linha 4.

Por enquanto declararemos as variáveis antes da função `setup()`. Isso implica que essas variáveis serão *globais*, isto é, podem ser usadas em qualquer lugar do programa.

As duas variáveis do exemplo acima são do tipo inteiro, chamado em Arduino de `int`, e permitem armazenar números inteiros. Para armazenar números reais é usado o tipo de dado `float`. Há outros tipos de dados (*Data Types*) em Arduino, listados na página de referência da linguagem. Mais detalhes sobre os tipos de dados serão vistos ao longo do curso quando forem necessários.

2 Atribuição

Uma caraterística fundamental de uma variável é que o valor associado a ela pode mudar durante a execução do programa. A mudança de valor é realizada pelo comando de atribuição, representado pelo '='. A sintaxe é bem simples: à esquerda do símbolo '=' se escreve o nome da variável e à direita uma expressão. A execução

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

do comando de atribuição consiste em calcular o valor da expressão e alterar o valor da variável para tal valor. Exemplos:

```
int v;        // declaração da variável v com tipo inteiro
v = 13 + 4;   // v recebe 17
v = v + 1;    // v tem seu valor incrementado em 1
v = 2 * digitalRead(A2); // v recebe o dobro do que é lido em A2
```

É importante não confundir variável com incógnitas normalmente usadas na matemática. Incógnitas são nomes que representam um valor, como as variáveis, porém representam um *único* valor. As variáveis *mudam* de valor por meio da atribuição.

Igualmente importante é não confundir o comando de atribuição '=' com o operador de igualdade '=='. Enquanto o primeiro é um comando que muda o valor da variável, o segundo é um operador relacional e que pode ser usado em expressões lógicas.

3 Constantes

Constantes são valores pré-determinados no programa e que não se alteram nem podem ser alterados. Os valores que correspondem a tipos básicos da linguagem, como números inteiros e números reais, são constantes e já os usamos em nossos programas. Por exemplo, quando fizemos `lcd.print(83.0/2);`, o valor `83.0` é uma constante de ponto flutuante (real) e o valor `2` é uma constante inteira.

Há também as constante simbólicas que possuem um nome que se refere a um valor, da mesma forma que uma variável, mas não podem ser alteradas. Também já usamos diversas dessas constantes, como `INPUT` (vale 0), `OUTPUT` (vale 1), `true` (vale 1), `false` (vale 0), `HIGH` (vale 1), e `LOW` (vale 0). Uma constante simbólica que não vimos ainda é a `LED_BUILTIN` que nos programas para Arduino Micro tem o valor 13, que é o número do pino ligado ao LED integrado à placa!

Podemos criar nossas próprias constantes, da mesma forma que criamos variáveis, mas com o uso adicional da palavra chave `const`. Por exemplo:

```
const int ledVerde = 13;
```

No exemplo, criamos uma constante `ledVerde` com o valor `13`. Constantes devem sempre ser inicializadas durante a declaração, já que não podem ser modificadas posteriormente. Veja que a palavra chave `const` antecede o tipo de valor.

Todas as constantes têm seus nomes substituídos pelos respectivos valores já em tempo de compilação, economizando espaço de memória. Além disso, expressões formadas apenas por constantes já são computadas em tempo de compilação.

A criação de constantes também pode ser realizada por meio de `#define`. Porém, o uso de `const` é preferível e não abordaremos `#define`.

4 Ordem das operações (ou precedência de operadores)

Quando uma expressão possui mais de um operador, o resultado depende da ordem ou precedência das operações, isto é, da ordem com que cada um dos operadores é avaliado. Por exemplo, qual o resultado da expressão `3 + 5 * (8 / 4) - 2`? Ainda que a página de referência da linguagem Arduino não especifique a precedência de operadores, pode-se tomar como referência a precedência de operadores em C/C++. Por exemplo, o operador de multiplicação tem maior precedência do que o operador de adição. Os operadores de multiplicação e de divisão possuem a mesma precedência. Operações entre parênteses têm maior precedência. Quando dois operadores têm a mesma precedência, considera-se a associatividade de operadores, que para os operadores aritméticos é da esquerda para a direita.

Assim, respondendo à pergunta do parágrafo anterior, o resultado seria 11. Como a operação entre parênteses tem precedência, a divisão de oito por quatro seria a primeira operação a ser avaliada, e a expressão intermediária seria `3 + 5 * 2 - 2`. A multiplicação tem maior precedência que a subtração e a adição, assim o produto de 5 por 2 é a próxima operação avaliada, o que resulta em `3 + 10 - 2`. A associatividade da adição e da multiplicação é da esquerda para a direita, assim a operação de adição é realizada primeiro, seguida da operação de subtração.

Apesar de termos discorrido dois parágrafos sobre ordem das operações, a melhor prática é a mesma empregada na matemática: usar parênteses para evitar a dúvida. Portanto, a melhor forma de escrever a expressão do primeiro parágrafo dessa seção é `3 + (5 * (8 / 4)) - 2`.

5 Expressões (de novo!)

Um definição mais completa para expressões do que aquela vista nas aulas anteriores é que expressões são a combinação de valores, **variáveis**, **constantes**, funções e operadores para produzir novos valores.

Segue alguns exemplos de expressões aritméticas:

```
10                    // uso da constante 10
10+5*2                // uso do operador binário +
(10+5)*2              // uso de parenteses
10 * -(3.0/2)         // uso do operador unário -
10+digitalRead(A2)    // uso de funções
10+a                  // uso de variáveis
```

6 Uso de variáveis

Variáveis podem ser usadas pra muitas finalidades na programação. Um primeiro uso para variáveis é evitar calcular a mesma coisa mais de uma vez. Considere, por exemplo, os seguintes programa:

```
digitalWrite(1, digitalRead(A3) + digitalRead(A2) + digitalRead(A1) + digitalRead(A0));
lcd.print(      digitalRead(A3) + digitalRead(A2) + digitalRead(A1) + digitalRead(A0));
```

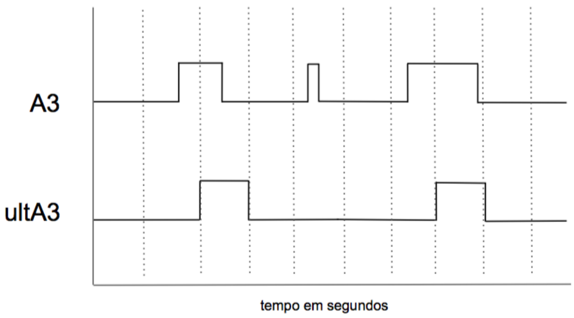
Notamos que o mesmo trecho de programa aparece duas vezes. Isso é ruim por dois motivos: correções na parte comum precisam ser feitas duas vezes e o computador tem que calcular a soma duas vezes. O seguinte programa resolve esse problema com variáveis:

```
int somaChaves = digitalRead(A3) + digitalRead(A2) + digitalRead(A1) + digitalRead(A0);
digitalWrite(1, somaChaves);
lcd.print(      somaChaves);
```

Uma outra utilização usual para variáveis é guardar na memória "valores do passado". Por exemplo, temos o `digitalRead()` para saber o valor *atual* de uma chave, mas qual era o seu valor na última vez que o `loop()` foi executado? O seguinte programa detecta uma mudança no valor da chave comparando seu valor atual com o valor que tinha na última execução do `loop()`:

```
1  ....
2  int ultimoValorA3 = LOW;
3
4  void loop() {
5    if (ultimoValorA3 != digitalRead(A3)) { // mudou o valor?
6      lcd.print("mudou");
7    } else {
8      lcd.clear();
9    }
10   ultimoValorA3 = digitalRead(A3); // o último valor é o atual
11   delay(1000);
12 }
```

No caso, a variável `ultimoValorA3` é uma "memória" do valor da chave A3 na última execução do `loop()` (ver linha 11). A figura abaixo ilustra as mudanças no valor desta variável no decorrer do tempo. Por causa do *delay*, a variável está sempre "atrasada" com relação ao valor da chave.



Variáveis também podem ser usadas para *acumular* valores. Por exemplo, podemos guardar em uma variável quantas vezes a chave mudou de valor da seguinte forma:

```
....
int ultimoValorA3 = LOW;
int nroMudancas   = 0;

void loop() {
  if (ultimoValorA3 != digitalRead(A3)) { // mudou o valor?
    lcd.print("mudou");
    nroMudancas = nroMudancas + 1;
  } else {
    lcd.clear();
  }
  ultimoValorA3 = digitalRead(A3); // o último valor é o atual
  delay(1000);
}
```

Cada vez que o valor da chave muda, a variável acumula 1, dito de outra forma, ela recebe o valor que já tinha mais um. Lembre que o comando de atribuição calcula a expressão da direita primeiro e depois associa o resultado à variável.

Note que variáveis acumuladoras precisam de um valor inicial, caso contrário, haverá um problema na primeira vez que `nroMudancas + 1` for calculado pois o valor de `nroMudancas` é indeterminado.

7 *Debugging*

Encontrar erros de programação, os *bugs*, não é tarefa fácil, mesmo se nós tivermos escrito ou que estejamos escrevendo o código. Há quem diga que muitas vezes debugar se assemelha à atividade de detetive, que precisa familiarizar-se com o problema, analisá-lo minuciosamente, e encontrar provas ou evidências. Na literatura ou nos filmes o estereótipo dos detetives é moldado em torno de impressões digitais e uma lupa para supostamente enxergá-las! Em programação muitas vezes as impressões digitais são os valores contidos em variáveis e a lupa é o *debugger*.

O *debugger* é um programa de computador, muitas vezes associado a uma IDE, que permite testar um programa e encontrar erros em seu código. O nível de sofisticação dos *debuggers* disponíveis é variado. Porém, a grande maioria permite pelo menos três ações: i) parar o programa em pontos previamente definidos e depois continuar sua execução; ii) executar o programa passo a passo (por exemplo, uma linha de código por vez); e iii) ver em tempo de execução os valores das variáveis.

O Tinkercad Circuits oferece um *debugger* bastante simples, mas que permite as três ações descritas acima. A Figura 1 mostra o circuito da Aula 4 no laboratório de eletrônica do Tinkercad Circuits com a janela de edição de código (`Code Editor`) ativada. Na janela de edição de código está ativado o botão `Debugger` que abre uma subjanela com dois botões e instruções de uso (*How it works*). O uso de *debugger* depende da escolha de *breakpoints* ou pontos de parada. Um ponto de parada é adicionado clicando-se no número da linha no editor de código que ficará destacada (ver linha 11). Mais de um *breakpoint* pode ser inserido em linhas diferentes. Um clique no botão `Upload & Run` ou no botão `Start Simulation` inicia a execução do programa que irá parar quando atingir uma linha com um *breakpoint* (ver linha 11). Nesse momento é possível passar o mouse sobre o nome de variáveis e ver seu valor. Na linha 11 o valor da variável `vaga` é `false`. Para continuar a execução do programa basta clicar no botão ▶. O programa irá parar no próximo breakpoint. O botão ↷ permite executar o código passo a passo, isto é, uma linha por vez.

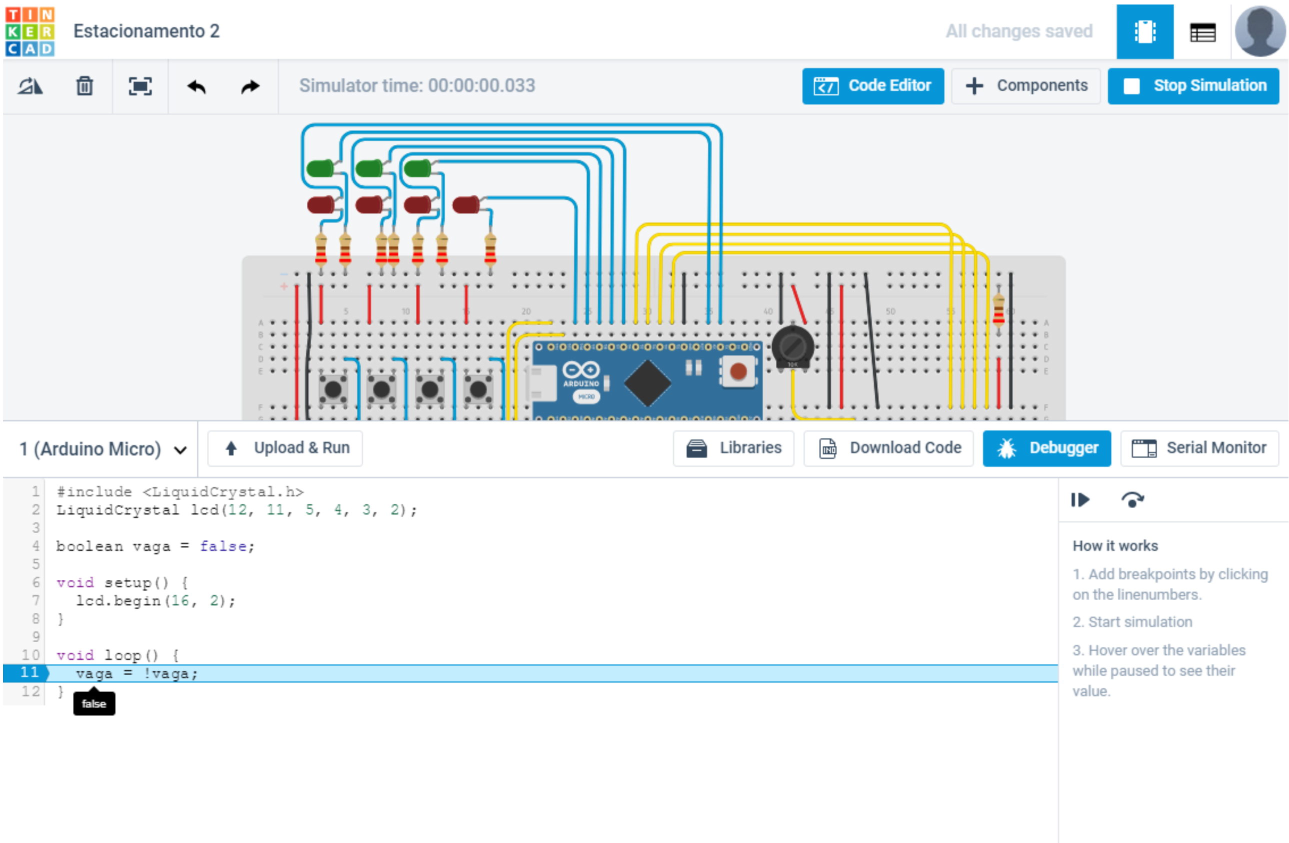


Figura 1: Uso do *debugger*.

8 Problemas

A seguir resolveremos os problemas desta aula. Acesse o circuito Estacionamento 2 e faça uma cópia em sua própria conta com um clique no botão `Copiar e Tinker`. Esse circuito (Figura 2) é praticamente o mesmo que o Estacionamento 1, mas as chaves foram trocadas por botões *pushbutton* ou botão tátil. Enquanto o botão estiver pressionado (usar o mouse), o nível no respectivo pino será `HIGH`, caso contrário será `LOW`.

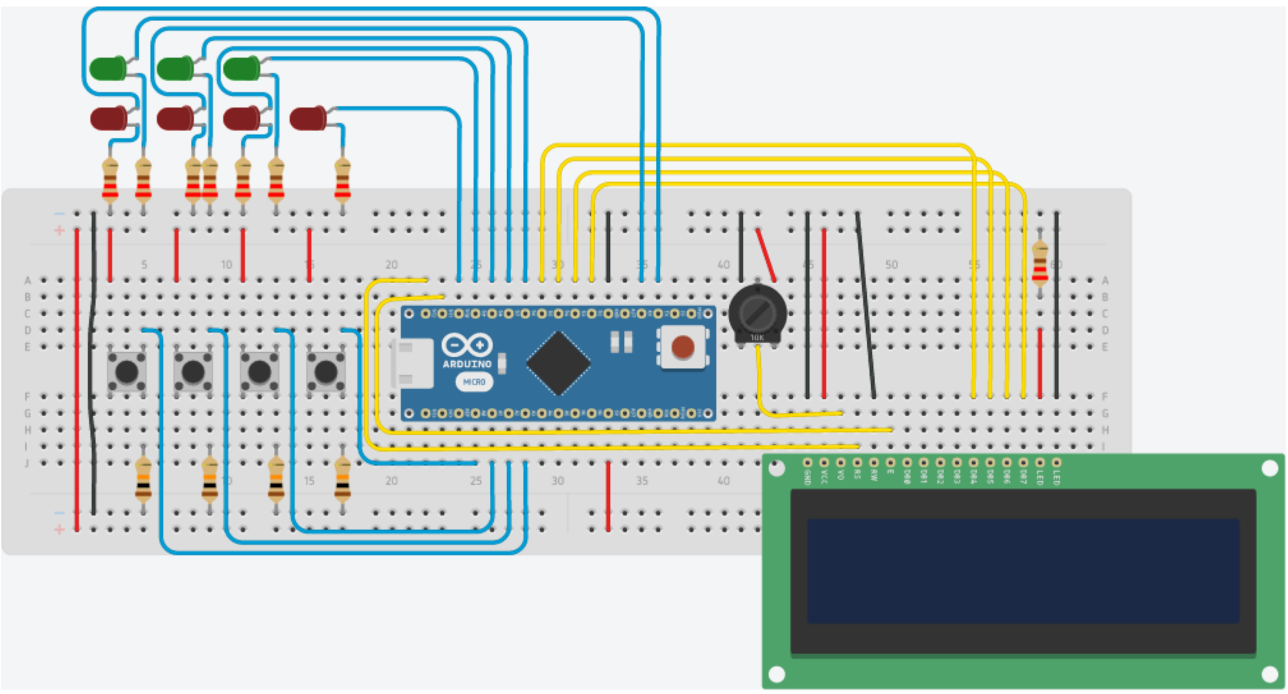


Figura 2: Circuito Estacionamento 2.

Problema 1

Reimplemente o programa do Problema 4 da Aula 2 com o novo circuito. Considere que apenas os *pushbuttons* conectados aos pinos `A0` e `A1` estão disponíveis, isto é, para entrar na sala segura, a pessoa deve apertar um botão e para sair deve apertar o outro. O programa deve atender os seguintes requisitos:

a) usar o circuito acima, que usa *pushbuttons*;

b) usar variáveis para, por exemplo, saber o número de pessoas na sala;

c) usar constantes para identificar os botões e os LEDs.

Por fim, compare a legibilidade do programa feito nesta aula com a versão anterior.

Problema 2

Um sistema de estacionamento como o que foi visto nos problemas das últimas aulas foi instalado em um estabelecimento comercial. Porém, o proprietário não contratou bons engenheiros formados pela UFSC e o sistema não funciona. O mal funcionamento ocorre pois os sensores foram instalados no lugar errado. O proprietário não está disposto a fazer mudanças na infraestrutura em função dos custos elevados. Mas alguém deu a dica para ele de que os alunos da disciplina de Introdução à Informática para Controle e Automação poderiam corrigir o problema apenas reprogramando o sistema.
A Figura 3 mostra a configuração das vagas. Como pode ser visto na figura, os sensores foram posicionados na entrada/saída das vagas junto com as lâmpadas. Assim, os sensores são ativados apenas enquanto o carro está saindo ou entrando na vaga.

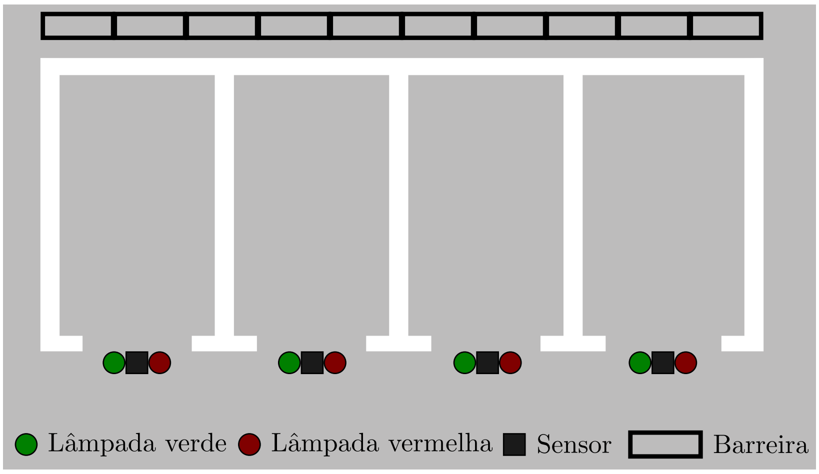


Figura 3: Estacionamento do Problema 1.

a) Modifique o programa do Problema 3 da Aula 3 para funcionar com a nova configuração. Primeiro tente resolver o problema sem o uso de variáveis e depois tente resolver com o uso de variáveis. Pode ser de interesse usar variáveis do tipo booleano, `boolean`, que recebem os valores `true` ou `false`.

b) Modifique o item a) considerando que a vaga é ocupada assim que o sensor for ativado (durante a entrada) e é desocupada assim que o sensor for desativado (durante a saída).

c) O que acontece se um motorista iniciar o ingresso na vaga e depois desistir?

d) Suponha agora que a barreira não existe e que é conveniente para os motoristas sairem de maneira irregular. Discuta o que aconteceria nesse caso. Seria possível resolver o problema apenas com o uso de programação?