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 1 — Programação[1]

1 Introdução

Apesar do nome "Introdução à Informática para Automação", esta é na verdade uma disciplina sobre *solução de problemas*. O que se deseja nesta disciplina é trabalhar nossas habilidades para formular um problema, pensar criativamente em sua solução e expressar essa solução com clareza e precisão. Claro que nos interessam os problemas de controle e automação, os quais surgirão na disciplina gradualmente. Para resolver esses problemas faremos o uso de programas de computador.

Um *programa* de computador é uma sequência de instruções que especifica como realizar uma computação. Essa *computação* pode ser necessária para resolver o cálculo da trajetória de um robô, para o processamento de um dado de um sensor ou para o processamento de imagem de um sistema de visão. Um programa é escrito com instruções disponibilizadas por uma *linguagem de programação*. A maioria das instruções nas diferentes linguagens são de um dos seguintes tipos: entrada, saída, matemática, execução condicional ou de repetição.

Precisaremos, portanto, programar. E programar nada mais é do que um processo de dividir tarefas grandes e complicadas em subtarefas cada vez menores e simples, até que essas subtarefas possam ser realizadas por meio de uma das instruções disponibilizadas pela linguagem escolhida. E qual é a linguagem escolhida?

2 Linguagem formal e linguagem natural

Antes de revelarmos a linguagem que usaremos para resolver computacionalmente os nosso problemas, é importante esclarecermos a diferença entre linguagem formal e linguagem natural. Não iremos muito a fundo, o interessado pode verificar mais detalhes na entrada sobre Linguagem na Wikipédia. Uma linguagem natural corresponde a uma linguagem falada que evoluiu socialmente como o português ou o inglês. Já uma linguagem formal foi projetada por pessoas para aplicações específicas, por exemplo, a notação matemática. Linguagens de programação são linguagens formais projetadas para expressar computações.

Linguagens formais têm regras rígidas de *sintaxe*. Por exemplo, em matemática "$3 + 3$" é uma sintaxe válida, mas "$2 + -5?7$" não é. O segundo exemplo não é válido porque, ainda que "+" e "−" sejam símbolos válidos, a combinação dos símbolos dessa maneira não é válida. Já o símbolo "?" não é um símbolo válido em matemática. Conclui-se disso, que as regras de sintaxe de uma linguagem formal dividem-se em dois tipos, um conjunto de símbolos válidos, como palavras ou números, e a maneira como os símbolos são combinados, ou seja a estrutura. Uma linguagem formal possui, além de sua sintaxe, uma semântica precisa que determina como as frases nesta linguagem devem ser interpretadas.

Apesar das semelhanças entre linguagens naturais e linguagens formais, pelo menos no que diz respeito à existência de uma sintaxe — símbolos e estrutura — e a análise sintática, há diferenças cruciais entre elas que determinam nossa capacidade de usar linguagens formais ou, mais especificamente, linguagens de programação para resolver problemas. Linguagens naturais são cheias de ambiguidades que dependem do contexto ou de outras informações disponíveis para a correta interpretação. Em linguagens formais não há ambiguidade e o significado de alguma coisa é o mesmo independente do contexto, do local ou do tempo. Por conta da ambiguidade, linguagens naturais são redundantes, o que as torna extensas, enquanto que linguagens formais são pouco redundantes e portanto concisas. Por fim, as linguagens naturais permitem expressões idiomáticas ou metafóricas. Em outras palavras, é possível dizer uma coisa para expressar outra completamente diferente. Por exemplo, alguém que ficou intrigado ou desconfiado provavelmente irá dizer que "está com a pulga atrás da orelha", mesmo que não haja de fato pulga alguma atrás da orelha

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

dele. Isto não é possível em liguagens formais, as quais são literais, ou seja, aquilo que é escrito em uma linguagem formal possui uma única interpretação.

A relevância dessas características nas linguagens formais, permite que o significado de um programa de computador possa ser compreendido simplesmente pela análise sintática dos símbolos e da estrutura que definem a linguagem.

3 Arduino/Genuino

Na primeira parte desta disciplina nós usaremos uma plataforma de desenvolvimento chamada Arduino e sua linguagem de programação de mesmo nome que é muito parecida com a linguagem C/C++. O Arduino é uma plataforma eletrônica aberta equipada com um microcontrolador e baseada em *hardware* e *software* fáceis de usar. Com placas Arduino é possível ler dados de um sensor, identificar a ativação de um botão ou até mesmo receber mensagens via internet. Acionar um motor, ligar uma lâmpada ou publicar em um *blog* na internet são algumas das atividades possíveis. E se isso tudo é possível, é claro que também podemos controlar processos ou automatizar plantas.

Notar que por questões comerciais, atualmente o nome Arduino é utilizado apenas para placas comercializadas nos EUA. No resto do mundo a marca utilizada é Genuino. Ou seja, Arduino e Genuino (Figura 1) são a mesma coisa, mas com um nome diferente.

Figura 1: Logotipos do Arduino e do Genuino.

4 Tinkercad Circuits

Na disciplina de "Introdução à Engenharia de Controle e Automação" vocês terão a oportunidade de usar uma placa Arduino, componentes eletrônicos e o ambiente integrado de desenvolvimento próprio do Arduino (Arduino *Integrated Development Environment* ou apenas IDE). Nesta disciplina usaremos um simulador chamado Tinkercad Circuits. Mais que um simulador, o Tinkercad Circuits é um ambiente *online* para o desenvolvimento e teste de circuitos eletrônicos, incluindo placas Arduino, e também para o projeto de placas de circuito impresso. Além disso, o Tinkercad Circuits faz parte da plataforma Tinkercad que permite o projeto e modelagem em 3D.

4.1 Conta

Para utilização do Tinkercad Circuits, é necessário criar uma conta gratuita. Para criar a conta siga as instruções da página na internet do Tinkercad Circuits ao clicar no botão `INSCREVER-SE` no canto superior direito (Figura 2). É possível usar uma de suas contas em redes sociais. Em caso de dúvidas ou dificuldades consulte o professor ou algum colega que tenha tido sucesso na criação da conta.

4.2 Visão geral

A Figura 3 mostra a tela do Tinkercad Circuits após o *login*. Nesta aula iremos trabalhar a partir de um projeto já iniciado. Utilize o link Circuito Display para ter acesso ao projeto inicial. Esse *link* dá acesso à tela do projeto (Figura 4) que mostra o circuito e permite copiá-lo para que cada um trabalhe em seu próprio projeto. Para isso, clique no botão `Copiar e Tinker` para fazer uma cópia em sua própria conta. Após o clique, a cópia para sua conta é realizada e é dado acesso ao ambiente de edição do circuito. Não faça nenhum alteração neste momento, queremos apenas ser apresentados ao ambiente de edição.

O editor é bastante simples. No canto superior esquerdo, há um logotipo do Tinkercad que quando clicado que leva de volta para a tela inicial com todos os circuitos em sua conta. É

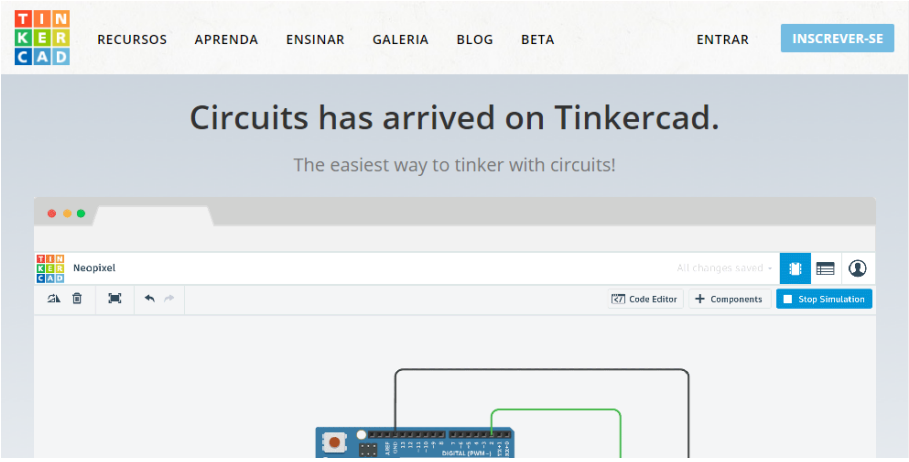

Figura 2: Tela inicial da página na internet do Tinkercad Circuits.

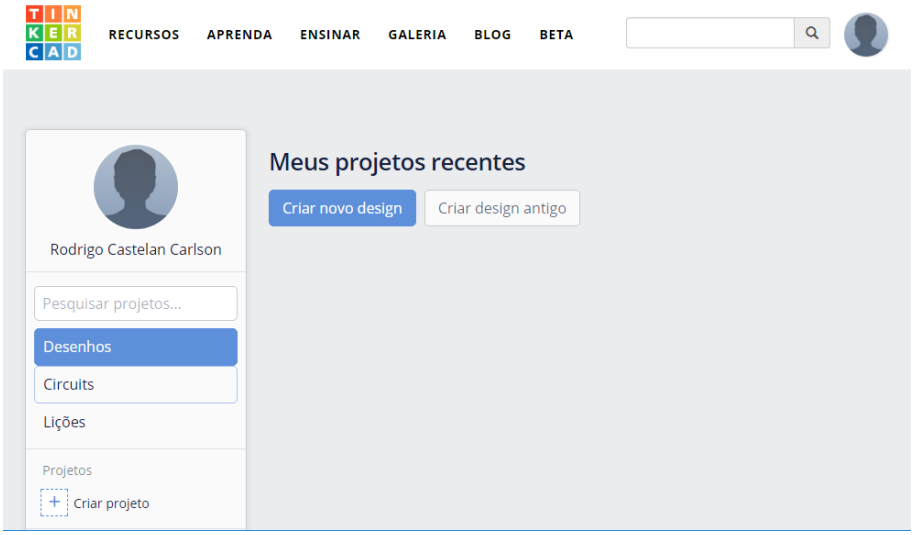

Figura 3: Tela inicial após o *login* no Tinkercad Circuits.

equivalente a clicar em `Circuits` no menu da esquerda da Figura 3. Logo ao lado está o nome do projeto (neste caso, Circuito Display) e abaixo botões para girar ou apagar um componente selecionado, ajustar o desenho do circuito ao tamanho da tela, e refazer ou desfazer operações. No canto superior direito podemos escolher entre visualizar o circuito (desenho de um circuito integrado) ou ver a lista de componentes (desenho de uma tabela). Logo abaixo é possível ativar o editor de código (`Code Editor`), o editor de componentes (`Components`) (não usaremos), e dar início a uma simulação (`Start Simulation`). Na parte central da figura há um circuito implementado e logo abaixo a janela do editor de código. Na topo da janela do editor de código podemos escolher o dispositivo que queremos programar (nesse circuito é o `1 (Arduino Micro)`), carregar e simular o programa (`Upload & Run`), ver as bibliotecas disponíveis (`Libraries` — mais tarde aprenderemos sobre bibliotecas), baixar o código para o computador (`Download Code`), acionar um recurso de auxílio na busca de erros (`Debugger`), e habilitar o monitor da serial (`Serial Monitor`) que permite testar a comunicação serial sem a necessidade imediata de outro dispositivo.

5 Programas em Arduino

Um programa para Arduino é chamado de *sketch*. Tipicamente um *sketch*, que passaremos a chamadar simplesmente de 'programa' é elaborado na IDE do Arduino e carregado no microcontrolador para então ser executado após a energização da placa. Este aspecto será trabalhado nas aulas de introdução à engenharia. Nesta disciplina usaremos a área de codificação do Tinkercad Circuits como visto na Figura 5. A listagem a seguir apresenta a estrutura básica de um programa:

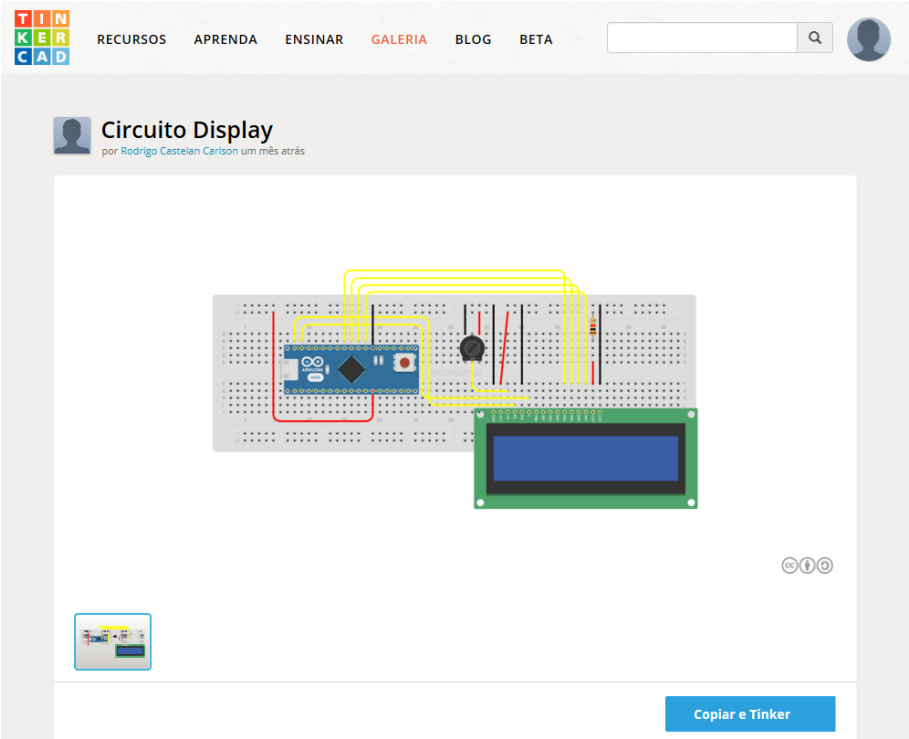

Figura 4: Tela do projeto Circuito Display no Tinkercad Circuits.

```
1  void setup() {
2
3  }
4
5  void loop() {
6
7  }
```

O programa da listagem cria duas funções, a função `setup()` nas linhas 1 a 3, e a função `loop()` nas linhas 5 a 7. Neste momento não precisamos nos preocupar com os detalhes dessas duas funções, teremos bastante tempo para tratar disso mais à frente. Porém, precisamos saber como usá-las. Atenção para o fato que os números das linhas são apenas para referência e não fazem parte do programa.

Inicialmente, digitaremos nosso código (instruções) entre as chaves (`{}`), isto é, entre a chave ao final da linha 1 e a chave no início da linha 3. Essas instruções dentro da função `setup()` serão executadas pelo Arduino toda vez que for ligado, ou toda vez que o botão *reset* for pressionado. Também digitaremos nosso código entre a chave ao final da linha 5 e a chave no início da linha 7. Essas instruções serão executadas após as instruções da função `setup()` e serão repetidas até que o Arduino seja desligado ou que o botão de *reset* seja pressionado. O código dentro da função `loop()` será o responsável pela interação com os processos e sistemas que iremos controlar/automatizar.

Atenção para o fato que a linguagem Arduino é *case sensitive*, ou seja, letras minúsculas e maiúsculas são símbolos diferentes. Essas diferenças devem ser respeitadas e usadas de modo consistente em nossos programas.

Para testar nossos programas usaremos, no simulador, a placa Arduino Micro (Figura 6). A placa Arduino Micro é a menor placa disponível na família de placas Arduino, mas ainda assim permitirá resolvermos vários problemas de controle e automação. Trata-se de uma versão um pouco mais recente, mas muito parecida com a placa Arduino Nano que será usada nas aulas de introdução à engenharia.

A esta altura, você deve estar se perguntando como é que a placa do Arduino compreenderá os programas que escreveremos. O que acontece é que ao clicarmos no botão `Upload & Run` ou no botão `Start Simulation` do Tinkercad Circuits, o código que escrevemos na linguagem do Arduino é processado por um programa de computador chamado compilador antes de ser gravado na memória do Arduino. O compilador traduz o nosso programa para uma linguagem de máquina que pode ser executada pelo microcontrolador da placa. O mesmo acontece no caso da IDE do

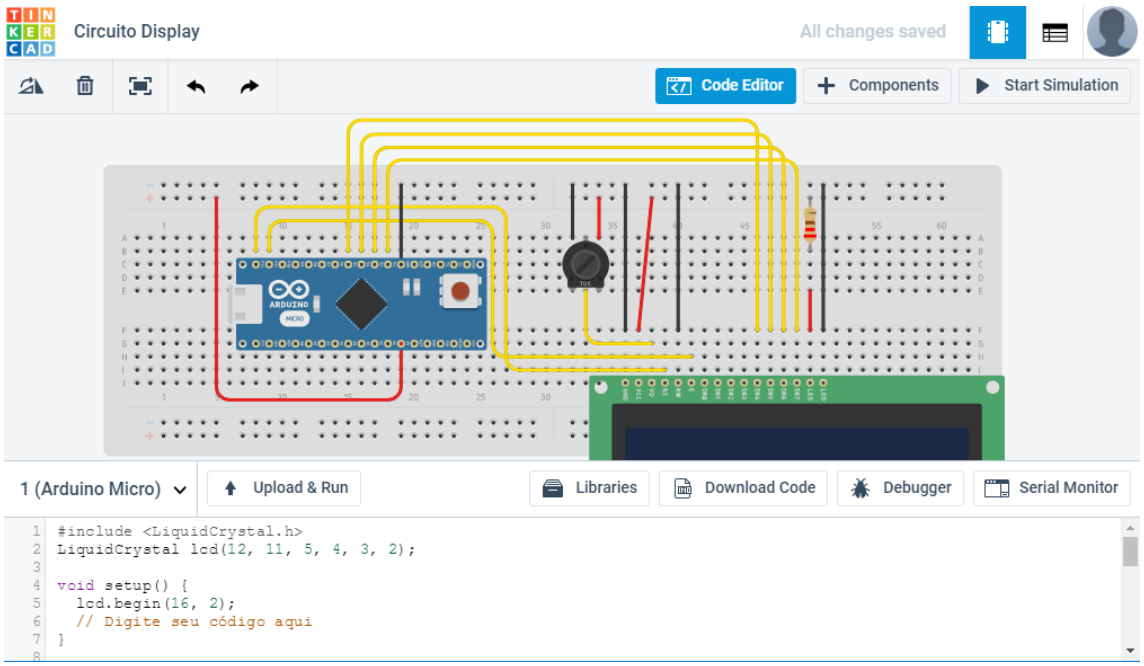

Figura 5: Editor de "Eletrônica" do Tinkercad Circuits.

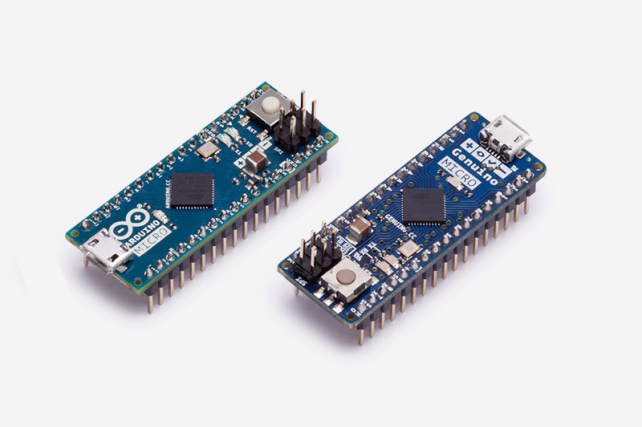

Figura 6: Vista superior e inferior da placa Arduino Micro.

Arduino, que já tem embutida também um compilador. Quando for resolver os problemas, observe que após clicar o botão `Upload & Run` ou o botão `Start Simulation`, se não houver nenhum erro no programa, aparecerá, temporariamente, a mensagem `Successfully compiled`.

6 Operadores aritméticos

Como mencionado na introdução, entre os tipos de instruções disponíveis em linguagens de programação temos as instruções matemáticas. O Arduino oferece operadores aritméticos, entre eles o operador de adição (`+`), o operador de subtração (`-`), o operador de multiplicação (`*`), e o operador de divisão (`/`). Todos esses operadores requerem dois operandos. Assim, se quisermos somar o número 3 com o número 5, devemos escrever `3 + 5`.

7 Valores e tipos

Valores são "coisas" básicas com que um programa trabalha, aos quais poderíamos chamar de dados primitivos. Exemplos são um número ou uma letra. Os valores de mesma natureza são agrupados em conjuntos, chamados *tipos*. Por exemplo, o tipo `int` representa o conjunto de todos os valores que são números inteiros. Fique atento durante a resolução do Problema 2 para ver o que acontece com os resultados das seguintes operações aritméticas: `83/2` e `83.0/2`. Veja se são

iguais ou diferentes. Os números 83 e 2 são números inteiros e portanto o resultado da divisão deve ser inteiro. Por outro lado, 83.0 é um número real que mesmo dividido por um número inteiro deve resultar um número real. Veja se é isso que acontece no Problema 2. Atenção: em Arduino, o separador decimal é o ponto. Veremos nas próximas aulas exatamente como definir e usar os diferentes tipos disponíveis na linguagem Arduino.

8 *Debugging*

A escrita de programas está sujeita a erros, geralmente chamados de *bugs*. O processo de procura por um erro é chamado de *debugging* ou depuração. A tarefa de *debugging* pode ser demorada e laboriosa, mas é uma habilidade indispensável para qualquer programador. Ao longo da disciplina vocês terão a oportunidade de provocar e resolver vários *bugs*. Aprenderemos mais sobre erros de programação e *debugging* nas próximas aulas.

9 Problemas

A seguir resolveremos o nossos primeiros problemas. Para a resolução desses problemas, acesse a sua cópia do Circuito Display.

Problema 1

Sistemas de automação geralmente funcionam sem a intervenção do operador. Mas isto não significa que ele não deva estar atento ao que acontece na planta. Para facilitar as atividades do operador, esses sistemas costumam estar equipados com módulos de monitoração que permitem acompanhar variáveis do processo, o seu estado ou receber mensagens de emergência. Escreva um programa em Arduino para o circuito da Aula 1 que mostre para o operador uma mensagem de que a operação foi iniciada.
Por sorte o circuito da Aula 1 conta com um *display* LCD e já funciona perfeitamente. Assim, não precisamos nos preocupar com a parte eletrônica. Além disso, a estrutura básica de um *sketch* e instruções auxiliares para o uso do *display* já estão implementadas:

```
1  #include <LiquidCrystal.h>
2  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
3
4  void setup() {
5    lcd.begin(16, 2);
6    // Digite o seu código aqui
7  }
8
9  void loop() {
10
11 }
```

Os comandos das linhas 1, 2 e 5 são necessários para o uso do *display* LCD. Aprenderemos sobre eles mais tarde. O conteúdo da linha 6 deve ser substituído pelo comando que mostra a mensagem para o operador. Usaremos uma instrução de saída, o comando `lcd.print("Inicio operacao!");` que mostra no *display* LCD o texto entre aspas (`""`). Um comando sempre é encerrado com o símbolo de ponto e vírgula (`;`). Veja que nos limitamos a 16 caracteres e não usamos acentos. Para simular clique no botão `Upload & Run` ou no botão `Start Simulation`.
Depois que o programa estiver funcionando, experimente outras mensagens, inclusive com mais caracteres ou com acentos e veja o que acontece. Para mudar a mensagem é necessário interromper a simulação com um clique no botão `Stop Simulation`.
Obs.: Como resolveremos os problemas seguintes com o mesmo circuito, ao final de cada problema você pode salvar o código em seu computador para posterior estudo com um clique no botão `Download Code` (ver Figura 5).

Problema 2

Você deve ter percebido que o nosso sistema de monitoração do Problema 1 mostra uma mensagem de início da operação, mas mantém essa mensagem até que o Arduino seja desligado ou que o botão *reset* seja pressionado. Dessa maneira, o operador não sabe se o sistema travou ou se de fato continua em operação. Modifique o código do Problema 1 para que sejam apresentados no *display* LCD em sequência e de maneira repetitiva os resultados das seguintes operações: adição de 37 e 3, a subtração de 1 de 26, o produto de 4 e 8, a divisão de 83 por 2, a divisão de 83.0 por 2, e a adição de 3 com 2 e com 1. Para resolver o problema siga os seguintes passos:

1. Antes de começar a programar, reflita sobre qual é parte do código existente em que serão inseridas as novas instruções. Ou seja, se a resolução desse problema será programada na função `setup()` ou na função `loop()`.

2. Use a função `lcd.print()` para mostrar o resultado de cada uma das operações, por exemplo `lcd.print(1 + 1);`.

3. Você deve ter percebido após o item 2 que o conteúdo do *display* LCD é preenchido e o resultado é uma tremenda bagunça. Isso acontece, pois o cursor, isto é, o ponto de escrita, continua a partir do último ponto onde parou. Para resolver esse problema use antes de cada chamada à função `lcd.print()` a função `lcd.clear()` que limpa o *display* LCD e reposiciona o cursor no início.

4. Oops! Com o uso de `lcd.clear()` resolvemos a bagunça, mas os valores mudam rápido demais. Que tal colocar um tempo de espera após cada chamada à função `lcd.print()` com o uso da função `delay()`? Essa função recebe entre parênteses um número que corresponde a um tempo dado em milisegundos para que o programa fique parado. Assim, `delay(1000);` interrompe o programa por 1 s.

5. Experimente clicar no botão de *reset* (botão vermelho na placa do Arduino). Descreva o que acontece.

Problema 3

O seguinte programa para o circuito da Aula 1 mostra no *display* LCD os valores 1 e 2 alternadamente para indicar que o sistema está em operação. Este tipo de comportamento é muito comum em alarmes, mas com o uso de luzes. Porém o programador cometeu um erro. Tente executar este programa, identifique o erro e corrija-o. Preste atenção na mensagem de erro gerada pelo simulador e veja se é de alguma ajuda para identificar e resolver o problema.

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

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

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

Problema 4

Repita o Problema 3, mas com o código que segue. A mensagem de erro gerada pelo simulador ainda foi útil?

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

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

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

Problema 5

Observe o código abaixo:

```
#include <LiquidCrystal.h>
                  LiquidCrystal lcd( 12      ,
11,
    5,            4,3,                        2
  )
  ; void
                                      setup(
) {  lcd.     begin(          16          ,

      2)
      ;
      lcd     .    print   (
      "Inicio operacao!")
;  // Digite o seu código aqui
                       }   void loop()            {
                }
```

Conseguiu identificar o que ele faz? Não? Então observe este outro código abaixo:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);void setup(){lcd.begin(16,2);lcd.print("Inicio
operacao!");
// Digite o seu código aqui
}void loop(){}
```

Depois de algum esforço talvez você tenha reconhecido que ambos são iguais!!! E são exatamente a solução do Problema 1. Durante o aprendizado nesta disciplina você será alertado todo o tempo da necessidade de:

- Uso correto de indentação do código;
- Uso adequado de espaçamentos e saltos de linhas;
- Uso de comentários em trechos de códigos mais complexos (veremos mais à frente o que são comentários).

Dica: Pesquise na Internet o que é *indentação*.

Lembre-se que um código feito por um programador muitas vezes é parte de um sistema muito maior programado por uma equipe de programadores. Códigos são feitos para serem legíveis! Por enquanto, fica aqui uma dica fundamental para ser seguida: **Todo código que você desenvolver poderá e deverá ser compreendido (o mais facilmente possível) pelos seus colegas!**.