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 6 — Funções definidas pelo Programador[1]

1 Novas funções

Na Aula 5 vimos que funções são uma sequência de comandos que realizam uma computação à qual se dá um nome. E quando queremos usar uma função, a chamamos pelo nome. Todas as funções que vimos até agora já estavam prontas, definidas, e estão incluídas de alguma forma na linguagem Arduino. Mesmo as funções `setup()` e `loop()` que tomamos a liberdade de editar em todas as aulas até aqui já estão de certa forma definidas, ainda que possamos mudar um pouco a sequência de comandos pela qual são formadas.

Uma nova função é criada por meio de sua definição que, como vimos, especifica um nome e a sequência de comandos executados quando a função é chamada. Envolve também a definição dos parâmetros e do valor devolvido, quando existirem, isto é, seus tipos.

Funções permitem:

- Agrupar comandos e dar um nome, o que resulta em facilidade de leitura do código e facilidade de realizar o *debugging*;
- Reduzir o tamanho dos programas por meio da reutilização do código. Uma função é definida apenas uma vez, mas pode ser utilizada quantas vezes for necessária;
- Melhorar a qualidade do código que pode ser testado separadamente para depois ser agregado ao restante, o que também facilita o *debugging*.

A sintaxe para a definição de uma função é a seguinte:

```
1  <tipo devolvido> <nome da funcao>([<tipo 1> <parametro 1>],  ...) {
2    <comandos>
3  }
```

O primeiro elemento na linha é 1 é o tipo devolvido. Quando a função devolve algum valor, o tipo do valor a ser devolvido deve ser indicado, por exemplo, `int` ou `float`. Se a função não devolver valor deve ser usada a palavra reservada `void`. O tipo devolvido é seguido por um espaço em branco e pelo nome da função. O nome da função, por sua vez, é seguido dos tipos e nomes dos parâmetros separados por vírgulas, entre parênteses, `()`. Se a função não tiver nenhum parâmetro, são colocados apenas os parênteses. A chamada da função deve conter como argumentos exatamente o número de parâmetros definidos e dos tipos correspondentes, caso contrário o compilador acusará um erro de sintaxe. O conteúdo da linha 1 é chamado de assinatura ou cabeçalho da função. O cabeçalho é seguido por um bloco de código entre chaves, `{}`, chamado de corpo da função, que corresponde à computação realizada pela função. Tantos comandos quantos forem necessários são possíveis, inclusive a chamada a outras funções. Quando a função devolve algum valor, o bloco de código deve conter a instrução `return` seguida do valor a ser devolvido. O valor a ser devolvido deve ser necessariamente compatível com aquele declarado no cabeçalho da função. Mais de uma instrução `return` poderá ser incluída no corpo da função caso condicionais sejam utilizados.

2 Exemplos

2.1 Funções matemáticas

Função matemáticas podem ser facilmente definidas na linguagem de programação: os parâmetros correspondem ao domínio da função e o retorno à imagem. Por exemplo, a função que mapeia dois números inteiros para a média entre eles é programada como segue:

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

```
float media(int a, int b) {
   return (a+b)/2.0;
}
...
// exemplos de uso da função:
x = media( analogRead(A0), 5);
...
lcd.print( media(x, analogRead(A1)));
```

2.2 Reutilização de código

O programa do Problema 4 da Aula 3, copiado a seguir já corrigido, é horrível em termos de qualidade de software. Um dos aspectos ruins é a repetição do código. Repare as linhas 20–54 e 60–94.

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

void setup() {
  pinMode(A3, INPUT);
  pinMode(1, OUTPUT);
  pinMode(0, OUTPUT);

  pinMode(A2, INPUT);
  pinMode(A1, INPUT);
  pinMode(A0, INPUT);

  lcd.begin(16, 2);
  lcd.print("Inicio operacao!");
}

void loop() {
  digitalWrite(1, HIGH);
  digitalWrite(0, LOW);
  if ((digitalRead(A3) == LOW) && (digitalRead(A2) == LOW) &&
      (digitalRead(A1) == LOW) && (digitalRead(A0) == LOW)) {
    lcd.home();
    lcd.print("Em operacao!    ");
    delay(1000);
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == LOW) &&
           (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
    lcd.home();
    lcd.print("Em operacao!    ");
    delay(750);
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
           (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
    lcd.home();
    lcd.print("Em operacao!    ");
    delay(500);
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
           (digitalRead(A1) == HIGH) && (digitalRead(A0) == LOW)) {
    lcd.home();
    lcd.print("Em operacao!    ");
    delay(250);
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
           (digitalRead(A1) == HIGH) && (digitalRead(A0) == HIGH)) {
    lcd.home();
    lcd.print("Em operacao!    ");
```

```
48      delay(125);
49    }
50    else {
51      lcd.home();
52      lcd.print("ERRO!           ");
53      delay(50);
54    }
55
56    digitalWrite(1, LOW);
57    digitalWrite(0, HIGH);
58    if ((digitalRead(A3) == LOW) && (digitalRead(A2) == LOW) &&
59        (digitalRead(A1) == LOW) && (digitalRead(A0) == LOW)) {
60      lcd.home();
61      lcd.print("Em operacao!    ");
62      delay(1000);
63    }
64    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == LOW) &&
65             (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
66      lcd.home();
67      lcd.print("Em operacao!    ");
68      delay(750);
69    }
70    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
71             (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
72      lcd.home();
73      lcd.print("Em operacao!    ");
74      delay(500);
75    }
76    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
77             (digitalRead(A1) == HIGH) && (digitalRead(A0) == LOW)) {
78      lcd.home();
79      lcd.print("Em operacao!    ");
80      delay(250);
81    }
82    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
83             (digitalRead(A1) == HIGH) && (digitalRead(A0) == HIGH)) {
84      lcd.home();
85      lcd.print("Em operacao!    ");
86      delay(125);
87    }
88    else {
89      lcd.home();
90      lcd.print("ERRO!           ");
91      delay(50);
92    }
93  }
```

É o mesmo código! Um código que mostra uma mensagem e faz a execução esperar um certo tempo (via `delay`). Isso pode ser facilmente resolvido criando uma função com esse trecho de código e chamando a função duas vezes:

```
1   #include <LiquidCrystal.h>
2   LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
3
4   void setup() {
5     pinMode(A3, INPUT);
6     pinMode(1, OUTPUT);
7     pinMode(0, OUTPUT);
8
9     pinMode(A2, INPUT);
10    pinMode(A1, INPUT);
```

```
11    pinMode(A0, INPUT);
12
13    lcd.begin(16, 2);
14    lcd.print("Inicio operacao!");
15  }
16
17  // nova rotina para mostrar a mensagem e executar o delay
18  void espera() {
19    if ((digitalRead(A3) == LOW) && (digitalRead(A2) == LOW) &&
20        (digitalRead(A1) == LOW) && (digitalRead(A0) == LOW)) {
21      lcd.home();
22      lcd.print("Em operacao!    ");
23      delay(1000);
24    }
25    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == LOW) &&
26             (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
27      lcd.home();
28      lcd.print("Em operacao!    ");
29      delay(750);
30    }
31    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
32             (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
33      lcd.home();
34      lcd.print("Em operacao!    ");
35      delay(500);
36    }
37    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
38             (digitalRead(A1) == HIGH) && (digitalRead(A0) == LOW)) {
39      lcd.home();
40      lcd.print("Em operacao!    ");
41      delay(250);
42    }
43    else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
44             (digitalRead(A1) == HIGH) && (digitalRead(A0) == HIGH)) {
45      lcd.home();
46      lcd.print("Em operacao!    ");
47      delay(125);
48    }
49    else {
50      lcd.home();
51      lcd.print("ERRO!           ");
52      delay(50);
53    }
54  }
55
56  void loop() {
57    digitalWrite(1, HIGH);
58    digitalWrite(0, LOW);
59    espera();         // ** chamada da nova rotina
60
61    digitalWrite(1, LOW);
62    digitalWrite(0, HIGH);
63    espera();         // ** chamada da nova rotina
64  }
```

Podemos perceber que a rotina `loop()` está muito mais fácil de ler e o tamanho do programa diminuiu de 93 para 64 linhas![2] Contudo, dentro da rotina `espera()` ainda vemos muitos trechos de código repetido. Uma das razões é que essa rotina está fazendo três atividades: ela calcula o *delay*, mostra uma mensagem e executa `delay`! Idealmente uma função/rotina deve fazer uma só coisa para poder ser melhor reaproveitada. A rotina acima só pode ser reaproveitada por programas que precisam fazer as três atividades!

[2]Notem que apesar do programa ser menor, a computação é a mesma! Ou seja, o tempo de execução é o mesmo.

Vamos então "tirar" a parte de cálculo do tempo de *delay* da rotina `espera()` e colocar esse cálculo em uma função. A seguinte função retorna o valor de *delay* conforme os valores dos sensores:

```
int calculaDelay() {
  if ((digitalRead(A3) == LOW) && (digitalRead(A2) == LOW) &&
      (digitalRead(A1) == LOW) && (digitalRead(A0) == LOW)) {
    return 1000;
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == LOW) &&
           (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
    return 750;
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
           (digitalRead(A1) == LOW)  && (digitalRead(A0) == LOW)) {
    return 500;
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
           (digitalRead(A1) == HIGH) && (digitalRead(A0) == LOW)) {
    return 250;
  }
  else if ((digitalRead(A3) == HIGH) && (digitalRead(A2) == HIGH) &&
           (digitalRead(A1) == HIGH) && (digitalRead(A0) == HIGH)) {
    return 125;
  }
  return 50;
}
```

A rotina `espera()` pode ser então simplificada para fazer somente duas atividades: mostrar a mensagem e executar o *delay*. Ainda é possível deixar a rotina `caculaDelay()` um pouco mais legível se criarmos uma função que faz a verificação dos valores dos pinos digitais:

```
boolean comparaPinos(int p1, int v1, int p2, int v2, int p3, int v3, int p4, int v4) {
   return (digitalRead(p1) == v1) && (digitalRead(p2) == v2) &&
          (digitalRead(p3) == v3) && (digitalRead(p4) == v4)
}
```

Note que essa função devolve um valor booleano. Assim, `caculaDelay()` pode ser reescrita como segue:

```
int calculaDelay() {
  if (comparaPinos(A3, LOW, A2, LOW, A1, LOW, A0, LOW)) {
    return 1000;
  }
  else if (comparaPinos(A3, HIGH, A2, LOW, A1, LOW, A0, LOW)) {
    return 750;
  }
  else if (comparaPinos(A3, HIGH, A2, HIGH, A1, LOW, A0, LOW)) {
    return 500;
  }
  else if (comparaPinos(A3, HIGH, A2, HIGH, A1, HIGH, A0, LOW)) {
    return 250;
  }
  else if (comparaPinos(A3, HIGH, A2, HIGH, A1, HIGH, A0, HIGH)) {
    return 125;
  }
  return 50;
}
```

Dessa forma, o programa final fica com apenas 61 linhas, bem melhor organizado, fácil de ler e de alterar.

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

```

```
4  void setup() {
5    pinMode(A3, INPUT);
6    pinMode(1, OUTPUT);
7    pinMode(0, OUTPUT);
8
9    pinMode(A2, INPUT);
10   pinMode(A1, INPUT);
11   pinMode(A0, INPUT);
12
13   lcd.begin(16, 2);
14   lcd.print("Inicio operacao!");
15 }
16
17 boolean comparaPinos(int p1, int v1, int p2, int v2, int p3, int v3, int p4, int v4) {
18    return (digitalRead(p1) == v1) && (digitalRead(p2) == v2) &&
19           (digitalRead(p3) == v3) && (digitalRead(p4) == v4)
20 }
21
22 int calculaDelay() {
23   if (comparaPinos(A3, LOW, A2, LOW, A1, LOW, A0, LOW)) {
24     return 1000;
25   }
26   else if (comparaPinos(A3, HIGH, A2, LOW, A1, LOW, A0, LOW)) {
27     return 750;
28   }
29   else if (comparaPinos(A3, HIGH, A2, HIGH, A1, LOW, A0, LOW)) {
30     return 500;
31   }
32   else if (comparaPinos(A3, HIGH, A2, HIGH, A1, HIGH, A0, LOW)) {
33     return 250;
34   }
35   else if (comparaPinos(A3, HIGH, A2, HIGH, A1, HIGH, A0, HIGH)) {
36     return 125;
37   }
38   return 50;
39 }
40
41 void espera() {
42   int d = calculaDelay(); //  guarda o valor do delay em uma variável para
43   delay(d);               //  usar no comando desta linha e no teste abaixo
44
45   lcd.home();
46   if (d > 50) {
47     lcd.print("Em operacao!    ");
48   } else {
49     lcd.print("ERRO!           ");
50   }
51 }
52
53 void loop() {
54   digitalWrite(1, HIGH);
55   digitalWrite(0, LOW);
56   espera();
57
58   digitalWrite(1, LOW);
59   digitalWrite(0, HIGH);
60   espera();
61 }
```

3 Variáveis locais e globais

A maioria das variáveis que declaramos até agora foram variáveis globais. Isto é, variáveis que foram declaradas fora das funções e que, portanto, podem ser usadas em todo o programa. Corresponde dizer que em qualquer lugar do programa é possível manipular seus valores. Variáveis locais são declaradas *dentro* de funções e podem ser usadas apenas dentro da função. Por exemplo, a variável `d` na linha 43 do programa acima só pode ser usada dentro da rotina `espera()`, se diz que o *escopo* da variável é a rotina `espera()`.

Podemos imaginar as variáveis como tendo um "ciclo de vida". Elas nascem, são usadas e morrem. As variáveis globais nascem quando o programa inicia a execução, existem durante toda sua execução e morrem quando o programa termina. As variáveis locais nascem quando a função inicia a execução, existem durante sua execução e morrem quando a função termina.

Não é a toa que a definição dos parâmetros de uma função assemelha-se muito com a declaração de variáveis. Parâmetros podem ser visto como variáveis locais. Por consequência, tal como variáveis locais, os nomes dados aos parâmetros não são compreendidos fora do corpo da função. A atribuição de valores aos parâmetros acontece por ocasião da chamada à função. Os valores passados como argumentos são atribuídos a cada um dos parâmetros, pare serem então usados dentro da função.

Sempre que possível, é preferível o uso de variáveis locais. Variáveis globais, por poderem ser alteradas em qualquer parte do programa, são difíceis de serem analisadas. Imagine uma variável que se chama `x` e é usada em um programa com 50000 linhas de código escrito por vários programadores. Se houver um erro associado ao seu valor, como achar o lugar do problema?

Um erro de compilação muito comum é: *not declared in this scope* - não declarada neste escopo. Escopo refere-se à área de validade de uma variável. Uma variável global é válida em todo o programa. Uma variável local, apenas no escopo onde ela foi definida. Assim, uma variável que foi definida dentro de uma função, tem essa função como escopo. Em outras palavras, a mensagem de erro acima indica que o programador está tentando usar uma variável que não foi declarada ou que foi declarada em outro lugar.

4 *Debugging*

A definição de função é como um contrato entre a função que executa um trabalho e quem a chama que fornece argumentos e espera um resultado. Os requisitos para a chamada da função, argumentos e seus tipos, são chamados de precondições. Nesse caso, a responsabilidade recai sobre quem chama a função, o qual deve passar argumentos dos tipos corretos. Quando há erros nas pré-condições o *bug* está em quem chama a função. Se as pré-condições estão corretas, a função cumpre sua tarefa e o que resulta dela são as pós-condições, que devem estar de acordo com o que foi estabelecido em sua definição. Quando há um erro nas pós-condições, o *bug* está na função.

5 Problemas

Nesta aula não iremos usar um novo problema, mas melhorar a qualidade de programas já feitos utilizando funções.

Problema 1

Considerando o programa abaixo como uma possível solução para o Problema 4 da Aula 1, reescreva-o usando funções para evitar repetição de código.

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

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

void loop() {
  lcd.print(17);
  delay(1000);
  lcd.clear();
  lcd.print(21);
  delay(1000);
```

```
  lcd.clear();
  lcd.print(3);
  delay(1000);
  lcd.clear();
}
```

Problema 2

Considerando o programa abaixo como uma possível solução para o Problema 4 da Aula 2, qual (ou quais) recurso(s) de programação poderia(m) ser usado para evitar repetição de código? Avalie possíveis soluções.

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

void setup() {
  lcd.begin(16, 2);
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(A3, INPUT);
  pinMode(A2, INPUT);
  pinMode(A1, INPUT);
  pinMode(A0, INPUT);
}

void loop() {
  digitalWrite(0, !(digitalRead(A3) + digitalRead(A2) +
                    digitalRead(A1) + digitalRead(A0)));
  digitalWrite(1,  (digitalRead(A3) + digitalRead(A2) +
                    digitalRead(A1) + digitalRead(A0)));

  digitalWrite(6,  (digitalRead(A3) + digitalRead(A2) +
                    digitalRead(A1) + digitalRead(A0)) >= 2);
  digitalWrite(8,  (digitalRead(A3) + digitalRead(A2) +
                    digitalRead(A1) + digitalRead(A0)) >= 2);
  digitalWrite(10, (digitalRead(A3) + digitalRead(A2) +
                    digitalRead(A1) + digitalRead(A0)) >= 2);
  delay(1000);
  lcd.print(digitalRead(A3) + digitalRead(A2) + digitalRead(A1) + digitalRead(A0));
  lcd.home();
}
```

Problema 3

Considerando o problema da Aula 4 (sobre vagas livres em um estacionamento) e a implementação abaixo, reescreva o programa usando funções procurando melhorar sua qualidade (evitando repetição de código, melhorando a legibilidade, etc.).

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

// variáveis para guardar o estado do estacionamento
```

```
boolean vagaLivreA0 = true;
int     ultA0       = LOW;

boolean vagaLivreA1 = true;
int     ultA1       = LOW;

boolean vagaLivreA2 = true;
int     ultA2       = LOW;

boolean vagaLivreA3 = true;
int     ultA3       = LOW;

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

  pinMode(A0, INPUT);
  pinMode(10, OUTPUT);
  pinMode(13, OUTPUT);

  pinMode(A1, INPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);

  pinMode(A2, INPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);

  pinMode(A3, INPUT);
  pinMode(1, OUTPUT);
  pinMode(0, OUTPUT);
}

void loop() {
  // controle da vaga A0
  if (ultA0 == LOW && ultA0 != digitalRead(A0)) {
    // mudou de LOW para HIGH, carro entrando ou saindo
    vagaLivreA0 = !vagaLivreA0; // muda o estado da variável
  }
  digitalWrite(10, vagaLivreA0);
  digitalWrite(13, !vagaLivreA0);
  ultA0 = digitalRead(A0);

  // vaga A1 (mesma idéia de programa que acima)
  if (ultA1 == LOW && ultA1 != digitalRead(A1)) {
    vagaLivreA1 = !vagaLivreA1;
  }
  digitalWrite(8, vagaLivreA1);
  digitalWrite(9, !vagaLivreA1);
  ultA1 = digitalRead(A1);

  // vaga A2
  if (ultA2 == LOW && ultA2 != digitalRead(A2)) {
    vagaLivreA2 = !vagaLivreA2;
  }
  digitalWrite(6, vagaLivreA2);
  digitalWrite(7, !vagaLivreA2);
  ultA2 = digitalRead(A2);
```

```
  // vaga A3
  if (ultA3 == LOW && ultA3 != digitalRead(A3)) {
    vagaLivreA3 = !vagaLivreA3;
  }
  digitalWrite(1, vagaLivreA3);
  digitalWrite(0, !vagaLivreA3);
  ultA3 = digitalRead(A3);

  lcd.home();
  lcd.print(vagaLivreA3+vagaLivreA2);
  lcd.print(" <- vagas -> ");
  lcd.print(vagaLivreA1+vagaLivreA0);
}
```