Universidade Federal de Roraima
Departamento de Ciência da Computação
Introdução a Sistemas Embarcados

LISTA DE EXERCÍCIO 02

Aluno: Paulo Ricardo Dutra Ribeiro da Silva

1. **Implemente os seguintes esquemas abaixo que deverá controlará um display de 7 segmentos que irá conectado diretamente ao Arduino e fará um contador hexadecimal configurável através de duas teclas onde você pode usá-lo de forma crescente (0-9) e decrescente (9-0).**

Descreva o resultado usando o simulador.

O Simulador utilizado foi o TinkerCad, disponibilizado online, para visualização da simulação, basta acessar o link abaixo:

https://www.tinkercad.com/things/i54E3ac4UvP-questao-1-display-de-7-segmentos/editel?sharecode=3mp63P5bKq8fJjHgRC9Lt3halChS0pujlY9fW_wih0c=

Figura 1 – Simulação do TinkerCad.

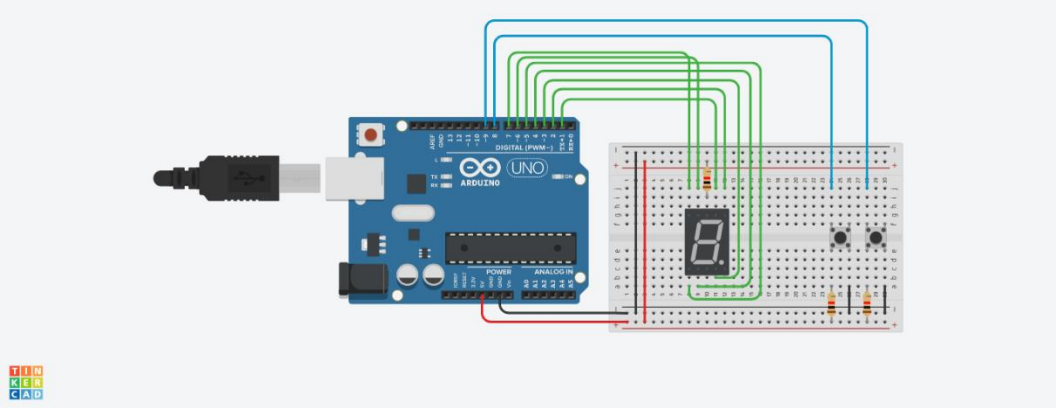
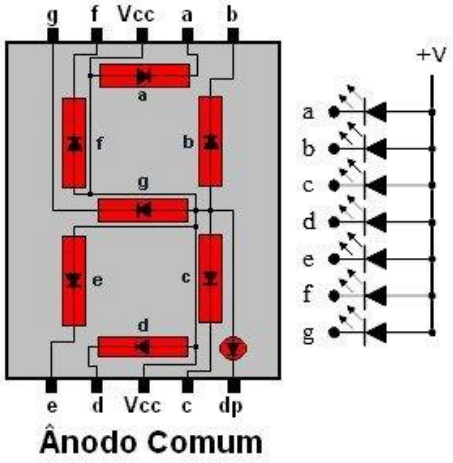


Figura 2 – Disposição dos led's no Display de 7 segmentos.



Ânodo Comum

O Código utilizado foi:

```
byte displayDig[10][7] =
{
  { 0, 0, 0, 0, 0, 0, 1 }, //DIGITO 0
  { 1, 0, 0, 1, 1, 1, 1 }, //DIGITO 1
  { 0, 0, 1, 0, 0, 1, 0 }, //DIGITO 2
  { 0, 0, 0, 0, 1, 1, 0 }, //DIGITO 3
  { 1, 0, 0, 1, 1, 0, 0 }, //DIGITO 4
  { 0, 1, 0, 0, 1, 0, 0 }, //DIGITO 5
  { 0, 1, 0, 0, 0, 0, 0 }, //DIGITO 6
  { 0, 0, 0, 1, 1, 1, 1 }, //DIGITO 7
  { 0, 0, 0, 0, 0, 0, 0 }, //DIGITO 8
  { 0, 0, 0, 1, 1, 0, 0 }, //DIGITO 9
};
```

Inicialmente criou-se uma variável do tipo Byte em matriz, onde cada linha contém o código em binário dos dígitos, o Display foi ligado em Ânodo Comum, logo onde está com o valor 0, tem se o pino correspondente acesso, na linha a ordem é o MSB (Most Significant Bit) é o led A da estrutura da Figura 2, enquanto que o LSB (Low Significant Bit) é o led G.

```
int auxC;
int auxD;
int contador = 0;
```

Em seguida é criado as variáveis auxiliares responsáveis pela coleta do valor do pushbutton e o contador para identificar em que número a contagem está.

```
void atualizaDisplay(byte digit) { //FUNÇÃO QUE MODIFICA O DISPLAY
  byte pino = 1;
  for (byte contadorSegmentos = 0; contadorSegmentos < 7; ++contadorSegmentos) {
    digitalWrite(pino, displayDig[digit][contadorSegmentos]);
    ++pino;
  }
  delay(100); //INTERVALO DE 100 MILISSEGUNDOS
}
```

Acima está descrito a função responsável por implementar o numero do contador no display. A função recebe como parâmetro o valor a ser mostrado no display, logo depois cria uma variável local chamada “pino”, para definir os pinos que serão ativados ou desativados, ela inicia em 1, pois o pino do led A, é o pino digital 1. Posteriormente a função inicializa um “for”, usando como parâmetro o valor do contador de segmentos, que irá contabilizar as

colunas da linha setada da matriz, o valor da linha é definido através do parâmetro de entrada chamado “digit”, o “for” será executado 7 vezes, e o valor do pino e do contador de segmentos será incrementado a cada execução, coletando o valor da matriz e ligando ou desligando o pino correspondente, atualizando assim o valor a ser mostrado no display.

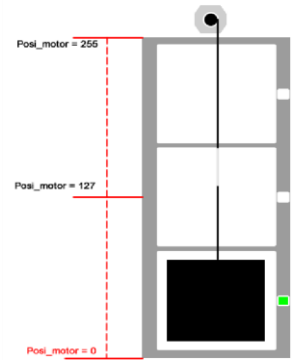
```
28 void setup()
29 {
30   pinMode(1, OUTPUT); //PINO 2 -> SEGMENTO A
31   pinMode(2, OUTPUT); //PINO 3 -> SEGMENTO B
32   pinMode(3, OUTPUT); //PINO 4 -> SEGMENTO C
33   pinMode(4, OUTPUT); //PINO 5 -> SEGMENTO D
34   pinMode(5, OUTPUT); //PINO 6 -> SEGMENTO E
35   pinMode(6, OUTPUT); //PINO 7 -> SEGMENTO F
36   pinMode(7, OUTPUT); //PINO 8 -> SEGMENTO G
37
38   pinMode(8, INPUT); //PINO 8 -> PUSH CRESCENTE
39   pinMode(9, INPUT); //PINO 8 -> PUSH DESCRECENTE
40
41 }
```

Na função “Setup”, foi configurando os pinos de conexão do display como “OUTPUT”, enquanto que os pinos dos pushbuttons como “INTPUT” para que possa ser coletado os momentos em que os pinos são pressionados.

```
43 void loop() {
44   auxC = digitalRead(8);
45   auxD = digitalRead(9);
46   if (auxC == LOW) {
47     if (contador == 9) {
48       contador = contador;
49     } else {
50       contador++;
51     }
52   }
53   if (auxD == LOW) {
54     if (contador == 0) {
55       contador = contador;
56     } else {
57       contador--;
58     }
59   }
60   atualizaDisplay(contador); //ATUALIZA O VALOR DE CONTADOR NO DISPLAY
61   delay(50);
62 }
```

Por fim, tem-se a função “Loop”, onde inicialmente se verifica os pinos dos pushbuttons e salva nas variáveis auxiliares, posteriormente é verificado através de “if” se algum pushbutton foi pressionado, e modifica a variável contador, caso a condição seja verdadeira, mas antes da modificação, existe um outro “if”, que verifica o valor do contador, para que ele não ultrapasse 9 nem fique abaixo de 0, e para finalizar, é chamada a função de atualização do display, colocando como parâmetro o valor da variável auxiliar “contador”.

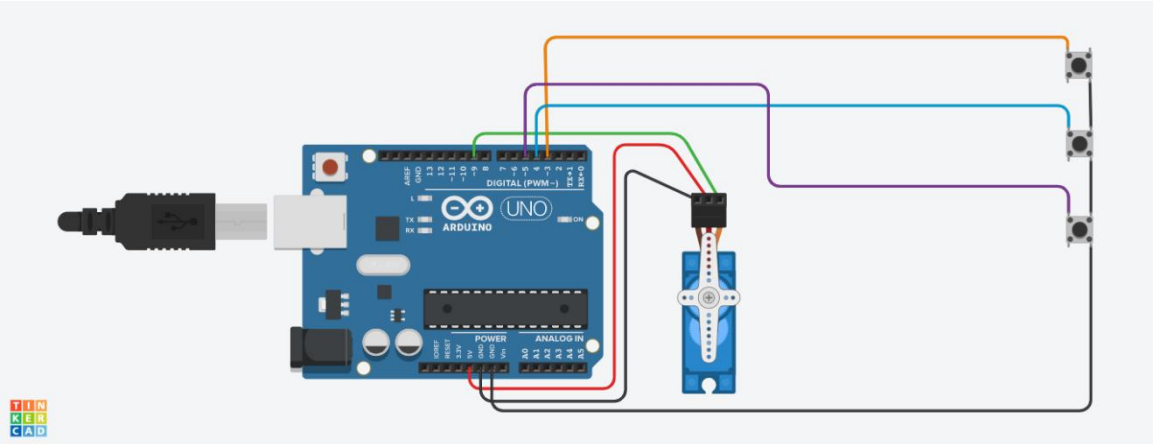
2. **Crie um programa para controlar um elevador que atenda 3 andares (1 botão para cada andar), onde cada andar e correspondente as seguintes posições em graus de um servo motor: andar 1 = 0º; andar 2 = 90º; e andar 3 = 180º. Apresente um esquema da ligação dos componentes necessários. Descreva o resultado usando o simulador.**



O Simulador utilizado foi o TinkerCad, disponibilizado online, para visualização da simulação, basta acessar o link abaixo:

https://www.tinkercad.com/things/bukfBmXzaXf-questao-2-elevador/editel?sharecode=NeFrh_niAtWt0gsqkgHPPkUKnZEZsz1Cf-o8LH0U3Lk=

Figura 3 - Simulação do TinkerCad.



O Código utilizado foi:

```
1 #include <Servo.h>
```

O código inicia incluindo a biblioteca necessária para o controle do servo motor, disponibilizada no site do Arduino, ou até mesmo no github.

```
3  const int pinTerreo = 3;
4  const int pin1Andar = 4;
5  const int pin2Andar = 5;
6  const int Terreo = 0;
7  const int Andar1 = 90;
8  const int Andar2 = 180;
9  int auxTerreo;
10 int aux1Andar;
11 int aux2Andar;
12 Servo servo_9;
```

Logo em seguida é declarado as variáveis necessárias para o funcionamento, inicialmente são declarada as constantes das portas onde os pushbuttons de cada andar está conectado, logo em seguida as constantes dos valores de ângulo dos andares (Foi necessário alterar os andares 2 e 3, pois o ângulo máximo de giro do servo motor é 180º), posteriormente a criação das variáveis auxiliares, que receberá o valor dos pinos digitais, por fim a criação do objeto do tipo Servo, para o controle do motor.

```
14 void setup()
15 {
16   servo_9.attach(9);
17   pinMode(pinTerreo, INPUT_PULLUP);
18   pinMode(pin1Andar, INPUT_PULLUP);
19   pinMode(pin2Andar, INPUT_PULLUP);
20   servo_9.write(Terreo);
21 }
```

Na função “Setup” inicia-se definindo o pino de conexão do servo motor, que está conectado no pino digital 9, posteriormente a configuração dos pushbuttons como sinal de entrada, a escrita “INPUT_PULLUP” auxilia na ativação do circuito interno ao Arduino para

que não seja necessário a inserção de um resistor físico ao pushbutton, ao mesmo tempo que o pino sempre inicia em valor alto, e quando pressionado, torna o valor da porta baixo. Ao final da função é enviado ao servo motor que ele inicialize sempre no Terreo, ou seja, em caso de desligamento, ele retornará ao térreo quando religar.

```
23 void loop()
24 {
25   auxTerreo = digitalRead(pinTerreo);
26   aux1Andar = digitalRead(pin1Andar);
27   aux2Andar = digitalRead(pin2Andar);
28   if (auxTerreo == LOW) {
29     servo_9.write(Terreo);
30   }
31   if (aux1Andar == LOW) {
32     servo_9.write(Andar1);
33   }
34   if (aux2Andar == LOW) {
35     servo_9.write(Andar2);
36   }
37 }
```

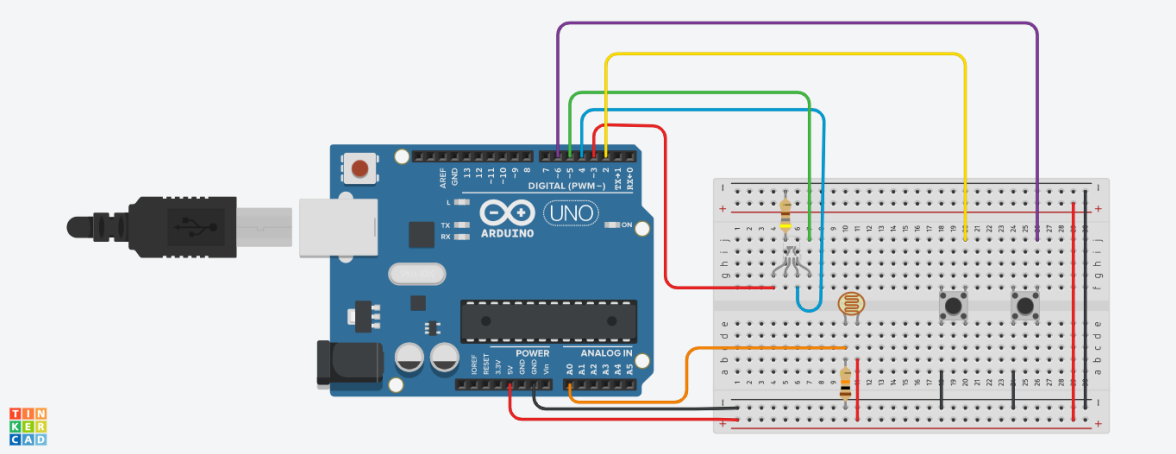
Por fim, a função “Loop” inicia salvando os valores das portas dos pushbuttons nas variáveis auxiliares, posteriormente é verificado através de “if” se algum pushbutton foi pressionado, em caso de verdadeiro, ele envia o comando do andar correspondente para servo motor executar.

3. **Implemente os seguintes esquemas abaixo que deverá identificar cores com o Arduino utilizando um LDR e um LED RGB. O programa deve gerar como saída as cores como o nome e o número em RGB. Descreva o resultado usando o simulador.**

O Simulador utilizado foi o TinkerCad, disponibilizado online, para visualização da simulação, basta acessar o link abaixo:

https://www.tinkercad.com/things/85hkDcQkSZ5-questao-3-led-rgb/editel?sharecode=t4_EdxPxKdwpTFjkcUufxQn_C7MmYaaFqNF26LhiYLc=

Figura 4 – Simulação do TinkerCad.



Ouve uma alteração no circuito padrão, para um melhor funcionamento das funções, foi adicionado 2 pushbuttons, onde um fica responsável pela calibração, quando ele for pressionado, entra na interrupção e executa a função de calibração, enquanto que o outro quando é pressionado, executa a função de leitura e demonstração no monitor serial da cor em código RGB.

O Código utilizado foi:

```
int matrizPinodosLEDS[] = {3, 4, 5};
float matrizCores[] = {0, 0, 0};
float matrizBranco[] = {0, 0, 0};
float matrizPreto[] = {0, 0, 0};

int mediaLeituras;
```

O Código inicia com a declaração das variáveis, declarando 1 matriz do tipo “int”, responsável por armazenar os pinos digitais conectado ao LED RGB, e 3 matrizes do tipo “float”, responsável por armazenar os valores lidos, a calibração do branco e calibração do preto respectivamente, e a criação da variável auxiliar do tipo “int” que armazenará a média das leituras realizadas.

```
8  void mediaSensor(int vezes) {
9    int leituras;
10   int somatorio = 0;
11   for (int i = 0; i < vezes; i++) {
12     leituras = analogRead(0);
13     somatorio = leituras + somatorio;
14     delay(10);
15   }
16   mediaLeituras = (somatorio) / vezes;
17 }
```

Logo em seguida inicia a declaração das funções, de inicio é declarada a função responsável por realizar a leitura e tirar uma média dos valores lidos, a função tem como parâmetro o numero de vezes que será realizada a leitura para verificar a média, posteriormente é declarado 2 variáveis locais e inicia-se o "for" que será executado o número de vezes que foi definido pelo parâmetro de entrada, onde será efetuada a leitura do pino analógico e salva na variável local, ao fim do loop "for", é salva a media das leituras na variável auxiliar.

```
19 void Calibracao()
20 {
21   Serial.println("Calibrando o branco");
22   delay(5000);
23   for (int i = 0; i <= 2; i++) {
24     digitalWrite(matrizPinodosLEDS[i], HIGH);
25     delay(100);
26     mediaSensor(5);
27     matrizBranco[i] = mediaLeituras;
28     digitalWrite(matrizPinodosLEDS[i], LOW);
29     delay(100);
30   }
```

A função de calibração declarada acima, inicia realizando a verificação do branco, o sensor deve ser posicionado em direção a uma superfície branca, 5 segundos depois é realizado a ativação do LED RGB e posteriormente é chamada a função de media das medições descrita anteriormente, por fim é salvo os valores na matriz de branco e desligado o RGB.

```
31    Serial.println("Calibrando o preto");
32    delay(5000);
33    for (int i = 0; i <= 2; i++) {
34      digitalWrite(matrizPinodosLEDS[i], HIGH);
35      delay(100);
36      mediaSensor(5);
37      matrizPreto[i] = mediaLeituras;
38      digitalWrite(matrizPinodosLEDS[i], LOW);
39      delay(100);
40    }
41    Serial.println("Sensor Calibrado");
42    delay(3000);
43  }
```

Dando continuação a função de “Calibracao”, é feita a calibração do preto, seguindo o mesmo padrão, é ligado o RGB, posteriormente feita a leitura das medias, salva em sua matriz correspondente e desligado o RGB, por fim é posto uma mensagem no monitor Serial informando que o sensor foi calibrado.

```
46  void checaCores() {
47    for (int i = 0; i <= 2; i++) {
48      digitalWrite(matrizPinodosLEDS[i], HIGH);
49      delay(100);
50      mediaSensor(5);
51      matrizCores[i] = mediaLeituras;
52      float cinzaDif = matrizBranco[i] - matrizPreto[i];
53      matrizCores[i] = (matrizCores[i] - matrizPreto[i]) / (cinzaDif) * 255;
54      digitalWrite(matrizPinodosLEDS[i], LOW);
55      delay(100);
56    }
57  }
```

A função “checaCores” é responsável por realizar a leitura do sensor e salva-la na matriz Cores, por questões de precisão da cor obtida, antes do salvamento é realizado o calculo do cinza, que consiste na diferença entre as matrizes branco e preto e antes do salvamento na matriz cores, é realizado a multiplicação do cinza por 255 e divido o resultado da diferença da matriz cores com a matriz preto. O Valor de 255 é devido a variação do valor de PWM do Arduino, que vai de 0 a 255, assim os pinos do RGB mandam o valor da intensidade de cada cor.

```
61 void mostraCores() {
62   Serial.print("R = ");
63   Serial.println(int(matrizCores[0]));
64   Serial.print("G = ");
65   Serial.println(int(matrizCores[1]));
66   Serial.print("B = ");
67   Serial.println(int(matrizCores[2]));
68 }
```

A outra função criada foi a responsável por apresentar no monitor serial os valores obtidos de R, G e B, que estão salvos na matriz de Cores, mostrando os valores individuais em ordem.

```
70 void setup()
71 {
72   Serial.begin(9600);
73   pinMode(3, OUTPUT); //LED VERMELHO
74   pinMode(4, OUTPUT); //LED AZUL
75   pinMode(5, OUTPUT); //LED VERDE
76   pinMode(6, INPUT_PULLUP); //PUSH BUTTON
77   pinMode(2, INPUT_PULLUP); //PUSH BUTTON INTERRUPÇÃO
78   attachInterrupt(digitalPinToInterrupt(2), Calibracao, FALLING);
79 }
```

Na função “Setup”, inicialmente é inicializado o monitor Serial com uma velocidade de comunicação de 9600, posteriormente é declarado os pinos de conexão dos LED como saída e os pushbuttons como entrada, logo em seguida é inicializado a função de interrupção, onde o pino 2 é responsável pela execução da função “Calibração”, configurada no tipo “Falling”, configurada como “FALLING”, então quando ocorrer a transição de alto para baixo a interrupção é ativada.

```
81 void loop()
82 {
83   if (digitalRead(6) == LOW)
84   {
85     checaCores();
86     mostraCores();
87   }
88 }
```

Por fim, na função “Loop” existe um “if” que só permite a execução da leitura e mostrar os dados, caso o pushbutton ligado ao pino digital 6 esteja pressionado, se não ele fica ligado, mas sem executar nenhuma função.

4. Defina o que é teste de software e descreva o microciclo do TDD.

O teste de software é a etapa, na construção de um programa, responsável por checar o nível de qualidade. Os defeitos que um teste busca identificar incluem erro de compatibilidade, de algum algoritmo, de requisitos que não podem ser complementados, limitação de hardware entre outros.

Existem diferentes tipos de testes que podem ser aplicados num software para identificar suas falhas, sendo as principais:

– Teste da caixa branca;
– Teste da caixa preta;
– Teste da caixa cinza;
– Teste de regressão;
– Teste de unidade;
– Teste de integração;
– Teste de carga;
– Teste de usabilidade;
– Teste de stress;

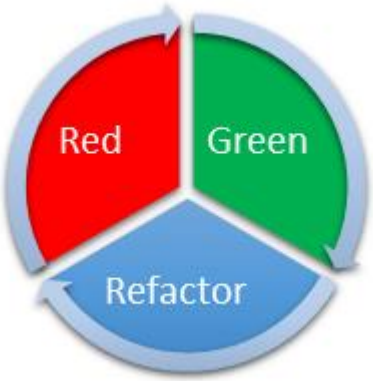
Para que esses testes possam ser realizados de modo mais rápido e com maior abrangência, existem ferramentas que automatizam alguns deles ou auxiliam na execução de outros. Essas são as ferramentas de teste de software.

O Test Driven Development é uma metodologia de desenvolvimento avançado com foco na qualidade do software orientado a objetos, amplamente utilizado dentro da

comunidade Agile. O que torna o TDD único é o seu foco na especificação do projeto, ao invés de ser na sua validação.

A abordagem TDD leva a uma melhor arquitetura de software de qualidade, melhorando a qualidade, simplicidade e capacidade de teste, enquanto aumenta a confiança no código e a produtividade da equipe. Ao implementarmos essa metodologia, os testes utilizados são compostos de afirmações verdadeiras e falsas que indicam o comportamento correto com relação ao teste que está sendo usado no momento.

Figura 5 – Cliclo TDD.



O TDD é conduzido por uma filosofia básica, conhecida como "Red-Green-Refactor" (Figura 5), que é um microciclo de desenvolvimento iterativo no qual os testes e comportamentos são implementados.

Neste processo, o que ocorre primeiramente é que os casos de teste recém-escritos testam o código ainda não escrito e, devido a isso, falham. Esta é a primeira etapa, definida como Red.

A partir deste momento o desenvolvedor implementa o recurso da maneira mais simples, fazendo com que os testes correspondentes sejam bem-sucedidos, o que representa o segundo passo do ciclo, o Green.

E por fim, antes de começarmos com um novo teste, podemos realizar o processo de refatoração do código de forma a deixá-lo o mais claro e eficiente possível, de forma a proporcionar confiança pela capacidade de executar o teste novamente que foi implementado.

Este é o terceiro item do ciclo, o Refactor. Ao adotarmos o ciclo Red-Green-Refactor, temos que o nosso código pode evoluir para um nível superior de comunicação simplificada, confiança e produtividade global.