UNIDADE 4: TOMADA DE DECISÃO E FUNÇÕES DE CONTROLO

OBJETIVOS

Estudar as funções que controlam o fluxo e a execução de um programa. Também são denominadas de “funções de controlo” e estão disponíveis em todas as linguagens de programação; Arduino não é exceção.

Os exemplos apresentados até ao momento são programas que incluem uma série de funções executadas sequencialmente deste o primeiro ao último passo. A partir de agora, os programas terão uma certa capacidade de “inteligência” e de tomada de decisão. Existem funções que só são executadas sob determinadas circunstâncias.

SECÇÃO TEÓRICA

- OPERADORES DE COMPARAÇÃO
- OPERADORES BOOLEANOS
- OPERADORES COMPOSTOS
- A FUNÇÃO IF(…)
- A FUNÇÃO IF(…) ELSE
- A FUNÇÃO FOR(…)
- A FUNÇÃO WHILE(…)
    - OUTRAS FORMAS DE WHILE
- A FUNÇÃO SWITCH(…) / CASE
- OUTRAS FUNÇÕES DE CONTROLO
    - A FUNÇÃO DO … WHILE(…)
    - A FUNÇÃO BREAK
    - A FUNÇÃO RETURN
    - A FUNÇÃO GOTO

SECÇÃO PRÁTICA

- EXEMPLO 1: A Campainha Elétrica V2
- EXEMPLO 2: Os *Fairy Lights* V2

- EXEMPLO 3: O Semáforo V3
- EXEMPLO 4: O *Beacon* Elétrico
- EXEMPLO 5: O Semáforo V4
- EXEMPLO 6: Explosões
- EXEMPLO 7: A Campainha Elétrica V3
- EXEMPLO 8: O Metro
- EXEMPLO 9: O Tempo

MATERIAL

-Laptop ou computador de secretária.

-Ambiente de trabalho Arduino IDE; deve incluir o material suplementar já instalado e configurado.

-Placa de controlo Arduino UNO.

-Um cabo USB.

ÍNDICE

SECÇÃO TEÓRICA

Nós, seres humanos, passamos grande parte dos nossos dias a tomar decisões – alguns de nós mais do que outros, claro. Podemos analisar o ambiente e as circunstâncias à nossa volta e tomar decisões com base nos nossos interesses, sentimentos, habilidades, intuição, compromissos, etc.. Essas decisões levam-nos a agir de maneiras diferentes, dependendo do que considerarmos mais adequado nesse momento.

Os programas que temos vindo a trabalhar até ao momento são muito sequenciais. Todas as instruções são executadas uma após a outra, desde a primeira à última, sem ter em conta qualquer outro aspeto.

No entanto, o Arduino, assim como qualquer outro controlador, tem a capacidade de tomar decisões e de executar programas ou realizar tarefas adequadas a cada caso particular. Como tenho certeza de que tem consciência, o Arduino não tem nenhum sentimento ou intuição; nem sequer é inteligente. É uma máquina e tudo o que sabe fazer é trabalhar com números. Ele toma decisões baseadas neles: cálculos lógicos e aritméticos, comparações, estados de sinais de *input*, valores analógicos que leem um sensor, etc.…

1. OPERADORES DE COMPARAÇÃO

O Arduino pode executar operações aritméticas, como adição e subtração. Os resultados podem ser expressos como constantes ou variáveis, ou como uma combinação de ambos.

Mas o Arduino também sabe como fazer comparações entre os números ou os resultados de certas funções. Na tabela que se segue apresentam-se os vários operadores de comparação, bem como os sinais que os representam:

OPERADOR	SINAL	OPERADOR	SINAL
Igual a	= =	Diferente de	!=
Menor do que	<	Maior do que	>
Menor do que ou igual a	< =	Maior do que ou igual a	>=

Seja qual for a comparação, existem apenas dois resultados possíveis: “true” (verdadeiro) ou “false” (falso). Observe atentamente os seguintes exemplos:

Supondo que...

```
char Letter = 'J';        int B = 12345;
byte A = 13;              float PI = 3.1416
```

Depois...

Letter == 'J'	**//True**
Letter != 'Q'	**//True**
18 < A	**//False**
A == 8 + 5	**//True**
B >= A	**//True**
PI * 2 > 8.16	**//False**
B – 1000 <= A * 12	**//False**
digitalRead(4)==1	**//True if pin 4 is on level "1"**

2. OPERADORES BOOLEANOS

É ainda possível relacionar algumas das comparações anteriores entre si. O Arduino tem três operadores lógicos, ou "booleanos", para este efeito. De certeza que se irá recordar de alguns dos exemplos que resolveu na Unidade 3. Aqui estão, abaixo, com os sinais correspondentes:

OPERADOR	SINAL
NOT	!
AND	&&
OR	||

Da mesma forma, existem apenas dois resultados possíveis quando duas ou mais expressões estão relacionadas, usando esses operadores lógicos: "true" (verdadeiro) ou "false" (falso). Sugerimos que utilize chavetas para agrupar cada relação: vai facilitar a leitura e evitar erros. Observe os seguintes exemplos:

(Letter == 'X') && (A > 10) **//False**

(A == 10+3) && (B >= 12345) && (Letter != 'Q') **//True**

(B > 12300) || (PI = 3.1412) **//True**

(A == B) || (A > 10 + 4) **//False**

!(A == B) **//True**

(digitalRead(4) ==1) && (digitalRead(8)==1) **//True se os pins, 4 e 8, estiverem no nível 1**

3. OPERADORES COMPOSTOS

Muitas vezes pode surgir a situação de estar a fazer operações muito simples com uma variável e o resultado acabar na mesma variável. Lembre-se de que pode usar os chamados "operadores compostos" para simplificar essas expressões. A tabela seguinte apresenta um resumo destes aspetos:

OPERAÇÃO	OPERADOR	EXEMPLO	IGUAL A
++	aumenta a unidade	X++	X= X + 1
- -	diminui um número	Y- -	Y = Y - 1
+ =	adição	X+=Y	X = X + Y
-=	subtração	X-= 3	X = X - 3
***=**	multiplicação	X *= Y	X = X * Y
/=	divisão	X /= 5	X = X / 5

4. A FUNÇÃO IF(...)

Esta é a função de controlo mais básica e a mais importante. Permite avaliar expressões e depois tomar decisões. Na computação, o sinal na figura à esquerda é usado para representar a tomada de uma decisão em diagramas conhecidos como fluxogramas. O controlador avalia uma expressão tão complicada ou ainda mais complicada do que as que já viu. Se o resultado for "true" (verdadeiro), executa todas as funções dentro das chavetas "{...}". Se o resultado for "false" (falso), não as executa e o programa continua a correr.

Lembre-se disto: qualquer expressão pode ser formada por operações aritméticas entre constantes e/ou variáveis relacionadas entre si por operadores de comparação que, por sua vez, podem estar relacionados usando operadores lógicos ou booleanos, como representado na Figura 1. Tome o tempo necessário para pensar sobre este aspeto e observe de novo os exemplos anteriores. Isto é importante ... não há pressa!

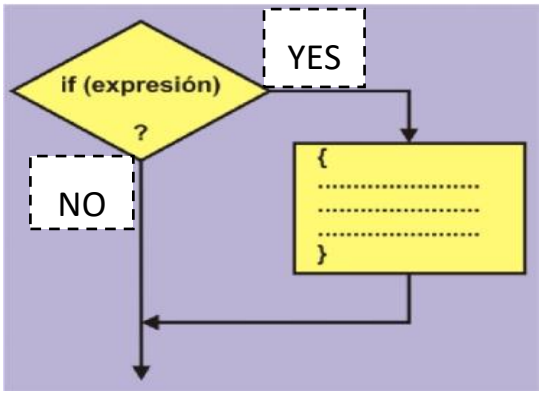

Figura 1

Sintaxe:

if(expression)

{

….

….

}

expression: estabelece a condição que o controlador Arduino tem de avaliar. Pode ser a relação entre constantes e/ou variáveis. Pode haver uma ou mais comparações entre si pelos operadores lógicos. Os resultados das operações aritméticas ou outras funções da linguagem Arduino também podem ser comparados.

curly braces: podem parecer as duas fatias de pão numa sandes – as chavetas encerram as funções que o controlador deve executar se a condição "true" existir. Não é necessário utilizá-los se houver apenas uma função para ser executada.

Exemplo:

```
 void loop()
{
if((A>B) || (C < 25))                    //Se a condição for true…
{
      digitalWrite(6,HIGH);              //Aciona no pin 6
C=25;                                    //Um valor de 25 é armazenado na variável C
}
....                                     //Continua a executar....
}
```

Dica: Como a utilização de chavetas é tão variada, é uma boa prática de programação digitar a chaveta de fecho imediatamente após a digitação da chaveta de abertura ao inserir uma construção que requer chavetas. De seguida, insira algumas barras entre as chavetas e comece a inserir instruções. No exemplo, as chavetas que abrem e fecham a função **loop()** são fáceis de distinguir das chaves que encerram as funções a serem executadas se a condição if() for verdadeira.

5. A FUNÇÃO IF(…)ELSE

Esta é outra função de controlo; é uma derivada da função **if(...)** apresentada na secção anterior (4). Refere exatamente o que fazer se a condição **(else)** for falsa. Observe a Figura 2. A função avalia a expressão ou condição. Se for "true", são executadas todas as funções dentro das chavetas "{...}" exatamente como com a função **if(…)**.

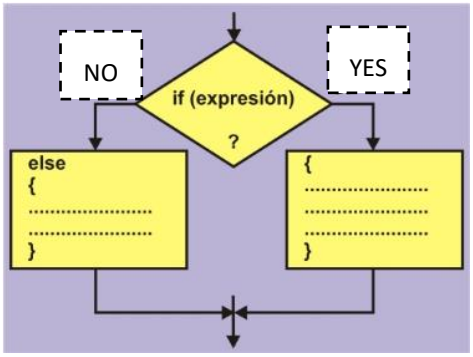

Figura 1

Se a condição for "false", todas as funções incluídas nas chavetas **else{...}** são executadas. Assim que as funções tenham sido executadas, independentemente de a condição ser verdadeira ou falsa, o programa continua a correr.

Sintaxe:

if(expression)

{

….

….

}

else

{

….

….

}

expression: estabelece a condição que o controlador Arduino tem de avaliar. Isto pode ser uma comparação de constantes e/ou variáveis. Pode haver uma ou mais comparações relacionadas entre si pelos operadores lógicos. Também pode comparar os resultados de operações aritméticas ou outras funções da linguagem Arduino.

curly braces: contém todas as funções que o controlador deve executar se a condição **(if)** for verdadeira ou falsa **(else)**. Não precisa incluí-las se houver apenas uma função para executar em qualquer um dos casos.

Exemplo:

```
if(digitalRead(4) == 1)          //Se o pin 4 estiver em "1" …
  {
digitalWrite(6,HIGH);            //Aciona o pin 6
tone(2,2000,400);                //Gera um nível de 2 KHz no pin 2
}
else                             //…e se não…
digitalWrite(6,LOW);             //Disativa o output 6
….                               //Continua a executar o programa
…
```

6. A FUNÇÃO FOR():

Esta função permite-nos criar *loops* controlados. Um *loop* repete um bloco de funções fechado em chavetas.

Uma imagem vale mais do que mil palavras. Observe o fluxograma representado na Figura 3. Declaramos um valor inicial de 1, por exemplo, para a variável "N". O valor é testado e se for inferior a 5 todas as funções incluídas nas chavetas são executadas. O valor da variável "N" é alterado automaticamente. É aumentado em uma unidade no exemplo (N ++). É testado novamente. Se for true, o bloco de informação e o *increment* são executados e, de seguida, a condição é testada novamente.

Quando a condição se torna *false*, o loop termina. As funções entre as chavetas no exemplo da figura são executadas quatro vezes: é esse o tempo que o "N" permanece inferior a 5.

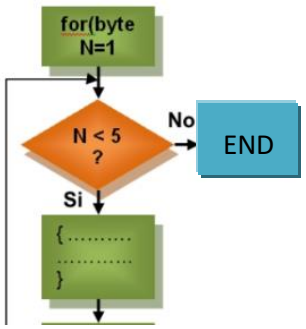

Figura 2

Sintaxe:

for(initialization,condition,increment)

{

….

….

}

initialization: esta é uma expressão que permite estabelecer o valor de uma variável. Acontece apenas uma vez no início do *loop*.

condition: é a condição que é testada. Se “true”, o bloco de instrução e o incremento são executados. Quando a condição se torna "false", o *loop* termina e o programa continua. A condição é testada sempre que o *loop* se repete.

increment: esta expressão torna possível alterar o valor da variável i. Executa-se cada vez que o *loop* se repete.

curly braces: aglomeram todas as funções que constituem o *loop*. As funções são executadas um número específico de vezes.

Exemplo:

```
for (int N = 1; N < 5; N=N+1)        //Estabelece as condições do loop
{
digitalWrite(6,HIGH);                //Aciona o pin 6
delay (150);                         // Coloca o programa em pausa durante 0.15”
digitalWrite(6,LOW);                 //Desativa o pin 6
delay (1000);                        //Coloca o programa em pausa durante 1”
}
….                                   //Continua a executar o programa
```

A sequência “aciona, pausa, desliga, pausa” no pin 6 repete-se quatro vezes.

Pode usar todos os tipos de expressões que estudamos no início desta unidade para definir a inicialização, a condição e o incremento de qualquer loop for(). Podem ser aritmética, lógica ou booleana ou comparações entre variáveis e/ ou constantes. Suponha apenas o seguinte:

```
int A = 5;

for(byte N=A+3; N <= A * 2 + 8; N = N + 3)

{

….

….

}
```

Agora é a sua vez!

Qual é o valor inicial? _____

E o valor final? _____

Quanto aumenta de cada vez? _____

Quantas vezes se repete o *loop*? _____

7. A FUNÇÃO WHILE():

while loops são uma uma variação nas variações **for()** por isso também são utilizadas para os loops quando o grupo de funções é executado um determinado número de vezes. Loops e mais loops...

Sintaxe:

```
while(condition)

{

….

….

}
```

condition: esta é a expressão condicional. Ele fará um loop continuamente e infinitamente até que a expressão dentro das chavetas, { }, se torne *false*. Quando isso acontece, o loop termina e o programa continua.

Exemplo:

```
int N = 6 while (N > 0)          //Enquanto N é maior do que 0
…
{
digitalWrite(6,HIGH);            //Aciona o pin 6
delay (150);                     //Coloca em pausa durante 0.15”
digitalWrite(6,LOW);             //Desativa o pin 6
delay (1000);                    // Coloca em pausa durante 1”
n--;                             //O valor seguinte para N ( N = N – 1)
}
….
….                               //Continua a executar o programa
```

A. OUTRAS FORMAS DE WHILE()

É muito comum usar a função **while()** como um loop em si. Não há chavetas e o loop executa a função até que a condição seja *false*. Observe atentamente os exemplos a seguir e observe onde o ";" se posiciona.

while(digitalRead(4)==1);

A função **while()** é executada constantemente desde que o pin 4 esteja no nível “1”. Por outras palavras, o exemplo espera que the example waits until pin 4 goes to level “0” to continue executing the program.

while(! digitaRead(4));

Esta função é exatamente como o exemplo anterior, mas de trás para a frente. A função **while()** é executada constantemente, desde que o pin 4 NÃO ESTEJA no nível “1”. Por outras palavras, desde que esteja no nível “0”. Portanto, espera até que o pin 4 atinja o nível “1” para continuar a executar o programa.

while(digitalRead(4)==0);

Este exemplo é igual ao anterior: aguarda que a função atinja o pin 4 e que vá para o nível “1” para continuar a executar o programa.

while(1);

Um loop. A condição é sempre *true* (1), por isso, a função **while()** é constantemente e indefinidamente executada. Não executa nenhuma função subsequente.

8. A FUNÇÃO SWITCH() / CASE

Esta função permite-lhe escolher entre várias “formas” de executar vários grupos de funções. Uma instrução *switch* compara o valor de uma variável com os valores especificados em instruções *case*. Quando uma declaração de caso (*case*), com valor correspondente ao da variável, é encontrada, é executado o código nessa instrução *case*.

A palavra-chave *break* faz sair a afirmação **switch()** e utiliza-se habitualmente no final de cada caso. É algo deste género:

“Se o valor da variável X efetuar estas funções. Se o valor da variável Y efetuar outras. Se for Z realiza estas outras funções, etc…”

Sintaxe:

switch(variable)

{

case X:

….;

break;

case n:

….;

….;

break;

default:

….;

}

variable: este é o valor da variável que será comparado com os valores de cada “case” mencionado.

case: corrige todos os valores que serão comparados ao conteúdo da variável. Quando o conteúdo da variável coincide com um dos valores, todas as funções entre este *case* e a expressão de quebra são executadas.

default: opcional. Se nenhum valor for coincidente, todas as funções serão executadas.

Exemplo:

```
switch(A)                          //Variável a ser comparada
{
case 1:                            //If the value of A is 1...
digitalWrite(6,HIGH);              //Ativa o pin 6
tone(2,200,200);                   //Tom no pin 2
break;                             //Sair
case 3:                            // Se o valor de A for 3…
B=digitalRead(7);                  //Lê o pin 7
break;                             //Sair
case 124:                          //Se o valor de A for 124…
B=12*4;                            //O valor de B é 48
digitalWrite(11,HIGH)              //Ativa o pin 11
break;                             //Sair
default:                           //Se nada mais coincidir…
 digitalWrite(6.LOW);              //Desativa o pin 6
 digitalWrite(11,LOW);             //Desativa o pin 11
}
```

9. OUTRAS FUNÇÕES DE CONTROLO

As funções de controlo introduzidas são as mais importantes e as que vai utilizar. A linguagem de programação do Arduino, no entanto, dispõe de outras funções de controlo. Fica ao seu critério fazer uso delas ou não! Talvez, à medida que vai aprendendo mais e que aperfeiçoe a sua técnica, poderá aperceber-se da sua utilidade.

A. A FUNÇÃO DO…WHILE()

O loop **do** funciona da mesma forma que o loop **while()**, com a exceção de que a condição é testada no final do loop, portanto o loop **do** sempre será executado pelo menos uma vez.

Sintaxe:

do

{

….

} while(condition)

B. A FUNÇÃO BREAK

break é utilizado para sair de um loop **for()**,**while()** ou **do()** loop, ignorando a condição de loop normal. Também é usado para sair de uma afirmação **switch**() / **case**.

Sintaxe:

break;

C. A FUNÇÃO RETURN

A função Return termina uma função e devolve um valor de qualquer função criada pelo utilizador para a função de chamada, se pretendido. Vai aprender como criar as suas funções um pouco mais à frente.

Sintaxe:

return;

return value;

value: este é o valor que a função retorna quando volta ao programa de chamada. É opcional e pode ser uma constante ou fazer parte de uma variável.

D. A FUNÇÃO GOTO

Transfere o flow do programa para um ponto etiquetado do programa.

Sintaxe:

test:

….

goto test: *//Automatically jumps to wherever the label indicated (“test”) is in the program.*

SECÇÃO PRÁTICA

10. EXEMPLO 1: A Campainha Elétrica V2

Esta é uma versão melhorada do exemplo apresentado na Unidade 5, onde tivemos que reinicializar todo o sistema sempre que queríamos tocar à campainha.

Essa versão melhorada também nos fornece uma boa oportunidade para usar a função if(). O programa verifica se o pin 4 está ativado;

Quando isso acontece, geram-se dois tons de 400 e 300 Hz consecutivos.

Se o pin 4 não estiver ativado, o programa não faz nada.

Observe as setas vermelhas na Figura 4. Foi sugerido anteriormente que é aconselhável que se alinhem as chavetas para que a relação entre as duas seja bem clara: uma das chavetas abre-se e a outra está fechada.

```
void setup()
{
  pinMode( 2 , OUTPUT);  //D2 Salida del altavoz
}

void loop()
{
if(digitalRead(4))    //Si se activa la patilla 4...
  {
  tone(2,400);        //Tono de 400 Hz
  delay(300);         //Durante 0,3"
  tone(2,300,300);    //Tono de 300 Hz durante 0.3"
  }
}
```

Figura 3

11. EXEMPLO 2: Os *fairy lights* V2

Aqui está outro velho amigo: as luzes intermitentes (*fairy lights*). O movimento é da direita para a esquerda ou vice-versa, dependendo se o pin 4 está no nível “0” ou no nível “1”. Esta é uma ótima oportunidade para usar a função **if( )else**.

12. EXEMPLO 3: O Semáforo V3

Vamos continuar a desenvolver e a melhorar alguns dos exemplos considerados na unidade anterior. Agora é o momento do semáforo. Nesta versão, a sequência começa quando o pedestre pressiona um botão conectado ao pin 4. Não precisa de pressionar o botão RESET para reiniciar o sistema como fez nas versões anteriores.

Se ninguém apertar o botão, o sinal de trânsito ficará vermelho. Este é outro bom exemplo de como usar a função **if(...) else**.

Sugerimos que compare este exemplo com as versões anteriores. As diferenças são muito pequenas, mas são significativas; vamos aperfeiçoar o semáforo pouco a pouco.

13. EXEMPLO 4: O *Beacon* Elétrico

Este é um bom exemplo prático que também serve como uma introdução à função for(…). Tenta simular um farol ou farol elétrico que fornece um certo número de flashes de luz durante um determinado período de tempo.

A função loop() na Figura 4 apresenta o programa o programa principal. A função for(…) fixa o valor inicial da variável (N=1) para o número de vezes que o loop tem de ser executado até o N atingir o valor final (N< Flashes); também fixa a melhoria de N cada vez que o loop (N++) é executado.

Se observar de perto, verá que o loop deve ser repetido cinco vezes. Cada vez que o loop é repetido, o LED branco acende-se durante 0,15” e, em seguida, apaga-se por outro 0,15”. Quando o ciclo estiver concluído, haverá uma pausa de 1,5” e o processo repetir-se-á.

```
int Tiempo = 150;          //Tiempo de encendido y apagado
int Destellos = 5;         //Nº de destellos deseados

void setup()
{
pinMode(6,OUTPUT);          //D2 Salida
}

void loop()
{
for(byte N=1;N<=Destellos;N++)
  {
  digitalWrite(6,HIGH);       //El led blanco se enciende
  delay(Tiempo);              //Temporiza
  digitalWrite(6,LOW);        //El led blanco se apaga
  delay(Tiempo);              //Temporiza
  }
delay(1500);                  //Temporiza 1.5 s.
}
```

Figura 4

- **Agora é a sua vez!**

Este programa é muito simples. Porque não tenta mudar o número de *flashes*, o tempo em que a luz permanece ligada e desligada, e o tempo entre os *flashes*. Experimente com durações distintas.

14. EXEMPLO 5: O Semáforo V4

Uma vez que já sabe como funciona a função for(...), este é o momento ideal para projetar a versão final (V4) do projeto do semáforo. Não apresenta melhorias funcionais relativamente à versão anterior, mas, tecnicamente falando, fornece a melhor solução para todos. Observe.

Se observar atentamente o programa, vai aperceber-se porque é que é a melhor solução. É verdade: os dois loops **for()** geram a sequência de tons emitida cada vez que as luzes verde e âmbar se acendem. O programa, por isso, tem poucas funções, utiliza menos memória flash e é, consequentemente, mais eficiente.

15. EXEMPLO 6: Intermitências

Os loops **for(…)** podem ser integrados. Isto significa que podemos colocar loops **for(…)** dentro de outros loops **for(…)**, e outros loops **for()** dentro desses e assim por diante.

Este programa é um exemplo simples de dois loops integrados **for(…)**. Fazem com que as luzes intermitentes pisquem de cada vez que se pressiona um botão que esteja ligado ao pin 4.

O LED vermelho uma vez por cada cinco flashes do LED branco; a sequência termina com o sexto flash.

```
void loop()
{
  if(digitalRead(4))  //Si la entrada D4 es verdad (nivel "1") ...
  {
    for(byte R=1; R<=6;R++)     //Mientras R <= 6
    {
      for(byte B=1;B<=5;B++)    //Mientras B <= 5 ...
        {
        digitalWrite(6,HIGH);   //Activa el led blanco
        delay(80);              //Temporiza
        digitalWrite(6,LOW);    //Desactiva el led oblanc
        delay(100);             //Temporiza
        }                       //Siguiente valor para B
     digitalWrite(11,HIGH);     //Activa el led rojo
     delay(100);                //Temporiza
     digitalWrite(11,LOW);      //Desactiva el led rojo
     delay(120);                //Temporiza
```

Figura 5

Existem dois loops for(…) dentro da função if(…). Os loops são executados sempre que alguém pressiona o botão conectado ao pin 4. A variável “B” controla o loop interno. A variável vai de 1 a 5. Ela gera um piscar de cinco flashes do LED branco.

A variável “*A*” vai de 1 a 6 e controla o loop **for(…)** exterior. À medida que envolve o loop interior, dá a volta seis vezes. Cada vez que dá a volta o LED vermelho pisca.

Quando dois ou mais loops for(…) são integrados, o loop interior é resolvido em primeiro lugar, depois o que se segue e assim por diante.

16. EXEMPLO 7: A Campainha Elétrica V3

Aqui está a versão V3 da nossa já conhecida campainha elétrica; é a finalíssima. O importante neste exemplo é o modo como tratamos o sinal de imput no pin 4. Não é suficiente pressionar o botão e definir um nível “1”. Temos que tirar o dedo dele e definir o nível de descanso “0”.

Essa tarefa é conhecida como "deteção de impulso". É muito usada em sistemas de controlo em que o sinal de entrada não é composto por um nível de lógica estática como “1” ou “0”. Existem muitos periféricos de entrada que emitem impulsos. Um impulso é uma transição completa do nível “0” para o nível “1” e vice-versa para o “0” (0-1-0) ou o contrário.

A função while() é ideal para detetar estas situações e muitas outras. Observe o programa representado na Figura 6.

A primeira função while() aguarda enquanto o input digital número 4 está no nível “0”. A segunda função while() aguarda enquanto o input digital número 4 está no nível “1”. Conclusão: até pressionar o botão (nível “1”) e depois retirar o seu dedo (nível “0”), não faz nenhum som. Experimente e preste atenção a este detalhe; isto é o que o torna diferente da campainha elétrica apresentada na versão V2.

```
void setup()
{
}

void loop()
{
  while(! digitalRead(4));  //Espera mientras la patilla 4 está a "
  while(digitalRead(4));    //Espera mientras la patilla 4 está a "
  tone(2,400);              //Tono de 400 Hz
  delay(300);               //Durante 0,3"
  tone(2,300,300);          //Tono de 300 Hz durante 0.3"
}
```

Figura 6

17. EXEMPLO 8: O Metro

Esta é uma aplicação prática. Imagine um sistema de controlo de acesso para um auditório com capacidade para dez pessoas. Um sensor ligado ao pin 4 deteta quando uma pessoa passa e gera um “impulso” para cada pessoa que entra. Quando o auditório está cheio, o dispositivo emite um sinal sonoro para dizer ao público que a peça está prestes a começar. Observe a função loop () na Figura 7.

As funções contidas na função while() são executadas no caso de a variável “contador” ser menor que ou equivalente a 10. Esta variável aumenta cada vez que um impulso 0-1-0 é detetado no pin 4. Duas funções while() separadas também são usadas para deteção.

Quando o sensor tiver gerado 10 impulsos, o que significa que há dez pessoas no auditório, emite um bip.

```
void loop()
{
  byte Contador=1;          //Inicia la variable con 1
  while(Contador <= 10)
  {
    while(! digitalRead(4)); //Espera mientras la patilla 4 está a
    while(digitalRead(4));   //Espera mientras la patilla 4 está a
    Contador++;              //Pulso recibido, contador + 1
  }
  tone(2,400);              //Tono de 400 Hz
  delay(300);               //Durante 0,3"
  tone(2,300,300);          //Tono de 300 Hz durante 0.3"
}
```

Figura 7

- **Agora é a sua vez!**

Depois de ter gravado o programa, verifique se funciona corretamente. Tudo o que precisa é pressionar o botão no pin 4. Isso simula o sensor do contador de pessoas. Está a funcionar corretamente? É isso? Não, não me parece…

Se observar atentamente, vai aperceber-se que parece considerar menos pessoas do que realmente são. Por outras palavras, quando cinco ou seis pessoas passam o beeper dispara. Este defeito deve-se ao "efeito de redundância" que ocorre muitas vezes na utilização de botões e de comutadores.

Mesmo que pressione o botão apenas uma vez, o pin de entrada no controlador não recebe apenas um único impulso 0-1-0, recebe vários. Isso deve-se ao fato de os contactos de metal no botão demorarem um pouco a se restabelecerem cada vez que aperta o botão.

Embora esse tempo seja de apenas alguns milissegundos, o nosso Arduino é muito mais rápido e deteta todos esses impulsos. O que isto significa é que, embora só pressione o botão uma vez, o Arduino pode detetar esse ato como um, dois ou mais impulsos.

Uma maneira de evitar isso é inserindo uma breve pausa. Quando uma mudança de estado no botão é detetada, inserimos uma pausa de 20 mS antes de aguardar a próxima mudança. Desta forma, evitamos quaisquer ressaltos durante esse lapso de tempo. Efetue as alterações seguintes no programa do EXEMPLO 8, grave e verifique se conta adequadamente.

while(! digitalRead(4)); *//Waits as long as pin 4 is on "0"*

delay(20); *//Anti-rebound pause*

while(digitalRead(4)); *//Waits as long as pin 4 is on "1"*

delay(20); *//Anti-rebound pause*

Counter ++; *//Pulse received, counter + 1*

18. EXEMPLO 9: O Tempo

Recorda-se da função millis()? É esta função que permiti saber quanto tempo se passa entre o momento em que liga o Arduino e/ou o momento em que o reinicia. Vamos usá-la em conjunto com a função switch(...)/case para medir lapsos de tempo diferentes.

É isso mesmo: depois da sequência para reiniciar o Arduino, vamos medir quanto tempo passa. Após um segundo, o LED branco acende-se. Depois de dois, o LED verde acende-se, depois de três, acende-se o LED âmbar e, após quatro, o LED vermelho. Por último, após 8 segundos do reinicio do sistema, apagam-se todos os LEDs ouve-se um som. Observe atentamente a função loop() no programa representado na Figura 8.

```
void loop()
{
 A=millis();             //Lee el tiempo transcurrido tras el RESE
 switch(A)               //Analiza la variable
 {
   case 1000:              //Si vale 1000 (1") ...
     digitalWrite(6,HIGH); //Activa el led blanco
     break;
   case 2000:              //Si vale 2000 (2") ...
     digitalWrite(6,LOW); //Desactiva led blanco
     digitalWrite(9,HIGH); //Activa el led verde
     break;
   case 3000:              //Si vale 3000 (3") ...
     digitalWrite(9,LOW);  //Desactiva led verde
     digitalWrite(10,HIGH); //Activa el led ámbar
     break;
   case 4000:              //Si vale 4000 (4") ...
     digitalWrite(10,LOW); //Desactiva led ámbar
     digitalWrite(11,HIGH); //Activa el led rojo
     break;
   case 8000:              //Si vale 8000 (8") ...
     digitalWrite(11,LOW); //Desactiva led rojo
     tone(2,1000,300);     //Tono de 3 KHz durante 0.3"
     break;
 }
}
```

Figura 8

A função **A=milis()** function armazena os milisegundos que passaram desde o reinício do sistema na variável "A". A função **switch(A)** analisa o seu conteúdo. Se o valor do conteúdo for 1000 (**case 1000**:), o LED branco acende-se. Passou um segundo (1000 mS). Se o valor do conteúdo for 2000 (**case 2000**:), acende-se o LED verde; se o valor for 3000, acende-se o LED âmbar e se o valor for 4000, é o LED vermelho que se acende. Finalmente, se o valor do conteúdo for 8000 (**case 8000**:), significa que decorreram 8 segundos. Apagam-se todos os LEDs, emitindo-se um tom de 1 KHz.

O programa continua a correr, mas o tempo decorrido exceed os 8000 ms (8 segundos), por isso, a variável "A" já não corresponde a nenhuma das cinco condições.

Sabe quanto tempo teria de esperar para que a variável corresponda novamente às cinco condições?

Vamos ver! A função millis() retorna um *unsigned long integer* de 32 bits. Por outras palavras, devolve um valor entre 0 e 4,294,967,295 milésimos de segundo, o que equivale a 4,294,967 segundos. Tendo em conta que um dia tem 86,400 segundos (24 * 60 * 60), seriam necessários 50 dias para que a função millis() sature e volte a 0.

Se quiser voltar atrás e recordar a sequência com que se acendem os LEDs, pode esperar todo esse tempo, ou simplesmente reiniciar o sistema. O que lhe parecer melhor…

REFERÊNCIAS

LIVROS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

SÍTIOS WEB

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com