Co-funded by the
Erasmus+ Programme
of the European Union

UNIT 4: DECSION MAKING AND CONTROL FUNCTIONS

AIMS

Study the functions that control the flow and execution of a program. They're also called "control functions" and they're available in all programming languages; Arduino is no exception.

The examples you've done up until now were programs comprising a series of functions that were executed sequentially from first to last. From now on the programs are going to have a certain amount of "intelligence" and decision making ability; there'll functions that are only executed under certain circumstances.

- EXAMPLE 5: The Traffic Signal V4
- EXAMPLE 6: Bursts
- EXAMPLE 7: the Electric Bell V3
- EXAMPLE 8: The Meter
- EXAMPLE 9: Time

PRACTICE MATERIALS

-Lap top or desk top computer

-Arduino IDE work environment; this should include the supplementary material already installed and configured.

-Arduino UNO controller board

-A USB cable

TABLE OF CONTENTS

THEORY SECTION

We human beings spend our days making decisions - some of us more than others, of course. We're able to analyse our surroundings and circumstances and make decisions based on our interests, feelings, abilities, intuition, commitments etc. These decisions cause us to act in different ways depending on what suits us.

The programs you've been working on until now have been very sequential ones. All the instructions are executed one after the other from first to last without any other consideration.

Nevertheless, Arduino, just like any other controller, has the ability to make decisions and execute programs or carry out tasks appropriate for each particular case. As I'm sure you're aware, Arduino hasn't got any feelings or intuition; it's not even intelligent. It's a machine and all it knows how to do is how to work with numbers. It makes decisions based on them: logical and arithmetic calculations, comparisons, states of input signals, analogical values read off a sensor etc…

1. COMPARISON OPERATORS

As you know, Arduino can perform arithmetic operations like addition and subtraction. You also know that the results can be expressed as constants or variables or a combination of both.

But Arduino also knows how to make comparisons between numbers or the results of certain functions. Here are the various comparison operators as well as the signs that represent them:

OPERATOR	SIGN	OPERATOR	SIGN
Equal to	= =	Different to	!=
Less than	<	Greater than	>
Less than or equal to	< =	Greater than or equal to	>=

Whatever comparison it is, there are only two possible results: "true" or "false". Have a close look at the following examples:

Suppose that...

```
char Letter = 'J';                int B = 12345;

byte A = 13;                      float PI = 3.1416
```

Then…

```
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. BOOLEAN OPERATORS

It's even possible to relate some of preceding comparisons to each other. Arduino's got three logical, or "boolean", operators for this purpose; they're sure to remind you of some of the examples you did in Unit 1. Here they are with their signs:

OPERATOR	SIGN
NOT	!
AND	&&
OR	||

In the same way, there are also only two possible results when two or more expressions are related using these logical operators: "true" o "false". I suggest you use curly braces to group each relation: it'll make reading them easier and avoid errors. Have a look at the following examples:

```
(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 if pins, 4 and 8, are on level 1

3. COMPOUND OPERATORS

Lots of times you'll be doing very simple operations with a variable and the result will end up in the same variable. Remember that you can use the so-called "compound operators" to simplify these expressions. Here's a summary of them in the table below:

OPERATION	OPERATOR	EXAMPLE	EQUAL TO
++	increases a unit	X++	X= X + 1
- -	decreases a number	Y- -	Y = Y - 1
+ =	addition	X+=Y	X = X + Y
-=	subtraction	X-= 3	X = X - 3
***=**	multiplication	X *= Y	X = X * Y
/=	division	X /= 5	X = X / 5

4. IF(...) FUNCTION

This is the most basic and important control function. It makes it possible to evaluate expressions and then make decisions. In computing, the sign in the figure on the left is used to represent the taking of a decision in diagrams known as flow charts. The controller assesses an expression as complicated as or more complicated than the ones you've already seen. If the result is "true" it executes all the functions inside the curly braces "{…}". If the result is "false" the controller doesn't execute them and the program keeps going.

Remember this: any expression can be formed by arithmetic operations between constants and/or variables related to each other by comparison operators which can in turn be related using logical or Boolean operators. Take your time to think about this and have another look at the previous examples. This is important...there's no rush!

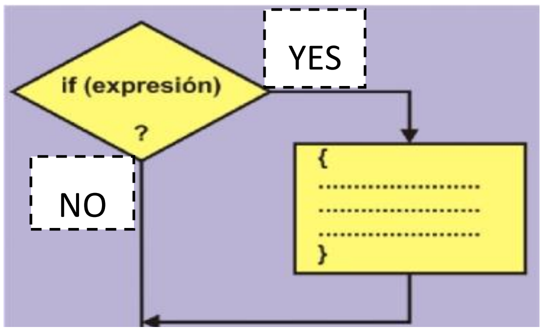


Figure 1

Syntax:

```
if(expression)
{
        ….
….
}
```

expression: establishes the condition that the Arduino controller has to assess. It might be the relationship between constants and/or variables. There might be one or more comparisons related to each other by logical operators. The results of arithmetic operations or other functions of Arduino language can also be compared.

curly braces: they might look like the two slices of bread in a sandwich – the curly brackets enclose the functions the controller has to execute if the "true" condition is met. You don't have to use them if there's only one function to execute.

Example:

```
 void loop()
{
if((A>B) || (C < 25))                    //If the condition is true…
{
        digitalWrite(6,HIGH);            //Switches on pin 6
C=25;                                    //A value of 25 is stored in the C variable
}
....                                     //Continues the execution....
}
```

Tip: Because the use of the curly brace is so varied, it is good programming practice to type the closing brace immediately after typing the opening brace when inserting a construct which requires curly braces. Then insert some carriage returns between your braces and begin inserting statements. In the example the braces that open and close the **loop()** function are easy to distinguish from the braces that enclose the functions to be executed if the if() condition is true.

5. IF(…) ELSE FUNCTION

This is another control function; it's a derivative of the previous **if(…)** function. It tells you exactly what to do if the **(else)** condition is false. Have a look at the Figure 2. The function assesses the expression or condition. If it's "true", all the functions in the curly braces "{…}" are executed just the same as with the **if(…)** function.

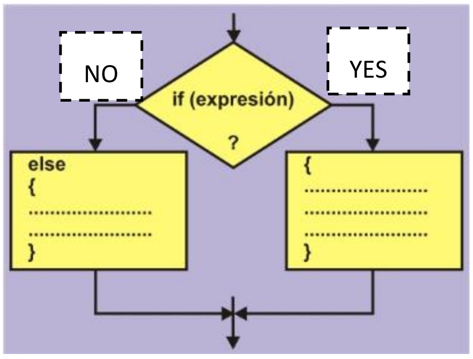


Figure 2

If the condition is "false" all the functions enclosed by the else {…} braces are executed. Once the functions have been executed, whether or not the condition is true or false, the program keeps going.

Syntax:

```
if(expression)
{
   ….
   ….
}
else
{
….
….
}
```

expression: establishes the condition the Arduino controller has to assess. This could be a comparison of constants and/or variables. There might be one or more comparisons related to each other by logical operators. It can also compare the results of arithmetic operations or other Arduino language functions.

curly braces: contain all the functions the controller has to execute if the **(if)** condition is true or false (**else**). You don't have to include them if there's only one function to execute in either case.

Example

```
if(digitalRead(4) == 1)          //If pin 4 is on "1" …
  {
digitalWrite(6,HIGH);            //Enables pin 6
tone(2,2000,400);                //Generates a 2 KHz tone on pin 2
}
else                             //…and if not…
digitalWrite(6,LOW);             //Disables output 6
….                               //Continues executing the program
…
```

6. FOR() FUNCTION:

This function enables us to create controlled loops. A loop repeats a block of functions enclosed in curly braces.

A picture's worth a thousand words. Have a look at the flow chart on Figure 3. We declare an initial value of 1, for example, for the "N" variable. The value is tested and if it's less than 5 all the functions enclosed in the curly braces are executed. The value of the "N" variable is changed automatically. It's raised by one unit in the example (**N++**). It's tested again. If it's true, the statement block and the increment are executed and then the condition is tested again.

When the condition becomes false, the loop ends. The functions enclosed in the curly brackets in the example in the figure are executed four times: that's how long "N" remains less than 5.

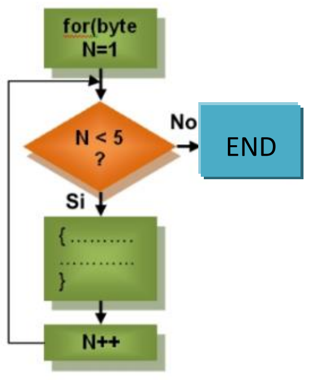


Figure 3

Syntax:

for(initialization,condition,increment)

{

….

….

}

initialization: this is an expression that makes it possible to establish the value of a variable. It only happens once at the beginning of the loop.

condition: it's the condition that's tested. If it's "true", the statement block and the increment is executed. When the condition becomes "false", the loop terminates and the program keeps going. The condition is tested every time the loop is repeated.

increment: this expression makes it possible to change the value of the **i** variable. This expression is executed each time the loop is repeated.

curly braces: they "sandwich" all the functions making up the loop. The functions are executed a specified number of times.

Example:

```
for (int N = 1; N < 5; N=N+1)       //Establishes the loop conditions
{
digitalWrite(6,HIGH);               //Enables pin 6
delay (150);                        //Pauses the program for 0.15"
digitalWrite(6,LOW);                //Disenables pin 6
delay (1000);                       //Pauses the program for 1"
}
….                                  //Continues executing the program
```

The sequence of “enable, pause, disenable, pause” on pin 6 is repeated four times.

You can use all the kinds of expressions we studied at the beginning of this unit to set the initialization, the condition and the increment of any for() loop. They could be arithmetic, logic or Boolean or comparisons between variables and/or constants. Just suppose the following:

```
int A = 5;

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

{

….

….

}
```

Now it's your turn:

What's the initial value? ___________________________________

And the final one? ____

How much does it increase each time? _________________________

How many times is the loop repeated? __________

7. WHILE() FUNCIÓN:

while loops are a variation on **for()** functions so they're also used for loops where a group of functions are executed a certain number of times. Loops and more loops...

Syntax:

```
while(condition)

{

….

….

}
```

condition: this is the conditional expression. It will loop continuously, and infinitely, until the expression inside the curly braces, {}, becomes false. When it does, the loop finishes and the program keeps going.

Example:

```
int N = 6 while (N > 0)          //While N is greater than 0
…
{
digitalWrite(6,HIGH);            //Enables pin 6
delay (150);                     //Pauses for 0.15"
digitalWrite(6,LOW);             //Disenables pin 6
delay (1000);                    //Pauses for 1"
n--;                             //The next value for N ( N = N – 1)
}
….
….                               //Continues executing the program
```

A. OTHER FORMS OF WHILE()

It's very common to use the **while()** function as a loop within itself. There are no curly braces and the loop executes the function until the condition is false. Have a close look at the following examples and note where the ';' is positioned.

while(digitalRead(4)==1);

The **while()** function is executed constantly as long as pin 4 is on level "1". To put it another way, the example waits until pin 4 goes to level "0" to continue executing the program.

while(! digitaRead(4));

This is just the like the previous example but back to front. The **while()** function is executed constantly as long as pin 4 ISN'T on level "1", or other words, as long as it's on level "0". It therefore waits until pin 4 goes to level "1" to continue executing the program.

while(digitalRead(4)==0);

This is the same as the previous example: it waits until pin 4 goes to level "1" to continue with executing the program.

while(1);

An infinite loop. The condition is always true (1), so the **while()** function is executed constantly and indefinitely. It doesn't execute any subsequent function.

8. SWITCH() / CASE FUNCTION

This function will enable you to choose between several “ways” of executing various groups of functions. A switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The *break* keyword exits the **switch()** statement, and is typically used at the end of each case. It’s something like this:

“If the value of the variable is X do these functions. If the value of the variable is Y do these other ones. If it’s Z do these other ones etc…”

Syntax:

```
switch(variable)

{

case X:

….;

break;

case n:

….;

….;

break;

default:

….;

}
```

variable: this is the value of the variable that’s going to be compared with the ones in each “case” mentioned.

case: this fixes all the values that are going to be compared to the contents of the variable. When the contents of the variable coincide with one of the values all the functions between this case and the break expression are executed.

default: this is optional. If none of the values coincides all the functions are executed.

Example:

```
switch(A)                                    //Variable to be compared
{
case 1:                                      //If the value of A is 1...
digitalWrite(6,HIGH);                        //Enables pin 6
tone(2,200,200);                             //Tone on pin 2
break;                                       //Exit
case 3:                                      // If the value of A is 3…
B=digitalRead(7);                            //Reads pin 7
break;                                       //Exit
case 124:                                    //If the value of A is 124…
B=12*4;                                      //The value of B is 48
digitalWrite(11,HIGH)                        //Enables pin 11
break;                                       //Exit
default:                                     //If nothing else matches…
 digitalWrite(6.LOW);                        //Disables pin 6
 digitalWrite(11,LOW);                       //Disables pin 11
}
```

9. OTHER CONTROL FUNCTIONS

The control functions I've introduced you to are the most important ones and the ones you're going to use. The Arduino programming language does however dispose of other ones. I don't want you to get snowed under though so I'll leave it up to you: if you want to use tem you can. Maybe as you learn more and improve your technique, you'll see their usefulness.

A. DO…WHILE() FUNCTION

The **do** loop works in the same way as the **while()** loop, with the exception that the condition is tested at the end of the loop, so the **do** loop will *always* run at least once.

Syntax:

```
do
{
….
} while(condition)
```

B. BREAK FUNCTION

break is used to exit from a **for()**,**while()** or **do()** loop, bypassing the normal loop condition. It is also used to exit from a **switch**() / **case** statement.

Syntax:

```
break;
```

C. RETURN FUNCTION

This one terminates a function and returns a value from any function created by the user to the calling function, if desired. You'll learn how to create your own functions a little later on.

Syntax:

```
return;
return value;
```

value: this is the value the function returns when it goes back to the calling program. It's optional and it may be a constant or form part of a variable.

D. THE GOTO FUNCTION

Transfers program flow to a labelled point in the program.

Syntax:

```
test:
….
goto test:          //Automatically jumps to wherever the label indicated ("test") is in the program.
```

PRACTICE SECTION

10. EXAMPLE 1: Electric bell V2

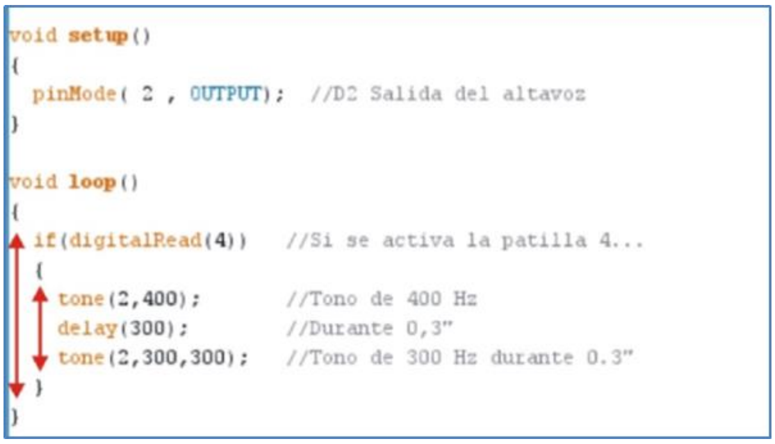
This is an improved version of the example we did in Unit 5 where we had to reboot the entire system every time we wanted to ring the bell.

This improved version also provides us with a good opportunity to use the if() function. The program checks to see if pin 4 is enabled;

When this happens two consecutive tones of 400 and 300 Hz are generated.

If pin 4 isn't enabled the program doesn't do anything.

Have a close look at the red arrows in the Figure 4. I've already suggested that it's advisable to align the curly braces so that the relationship between the two is quite clear: one of the braces opens and the other closes.



```
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"
  }
}
```

Figure 4

11. EXAMPLE 2: The fairy lights V2

Here's another old friend: the fairy lights. The movement is from right to left or vice versa depending on whether pin 4 is on level "0" or level "1". This is a very good opportunity to use the function **if(..) else** function.

12. EXAMPLE 3: The traffic signal V3

Let’s keep improving on some of the examples from the last unit. Now it’s time for the traffic signal. In this version the sequence begins when the pedestrian presses a pushbutton connected to pin 4. You don’t have to press the RESET button to reboot the system like you did in the previous versions.

If nobody presses the button the traffic signal stays on red. This is another good example of how to use the **if(…) else** function.

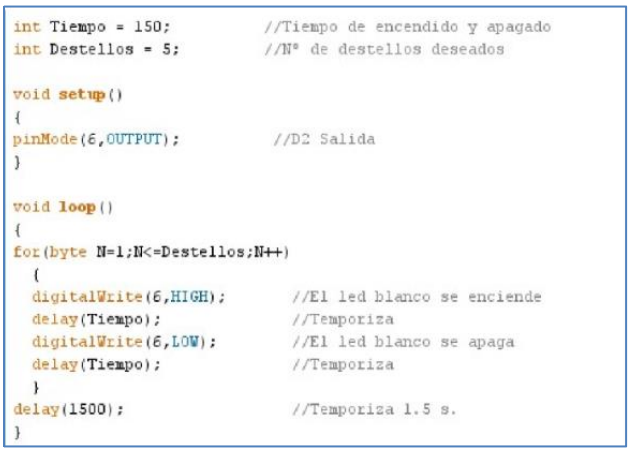
I suggest you compare this example to the previous versions. The differences are small but significant; we’ll be improving our traffic signal bit by bit.

13. EXAMPLE 4: Electric beacon

This is a good, practical example which also serves an introduction to the for(…) function. It attempts to simulate a beacon or electric lighthouse that gives out a certain number of flashes of light during a given period of time.

The loop() function in the figure on the right contains the main program. for(…) fixes the initial value of the variable (N=1) for the number of the times the loop has to be executed until N reaches the final value (N< Flashes); it also fixes the increment of N each time the loop (N++) is executed.

If you have a close look at it you’ll see that the loop has to be repeated five times. Each time the loop is repeated the white LED goes on for 0.15” and then goes off for another 0.15”. Once the loop is finished there’s a pause of 1.5” and the process is repeated.



```
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.
}
```

Figure 5

- **Now it’s your turn**

This program is very simple. Why don’t you try changing the number of flashes, the time the light stays on and off and the time between one burst of flashes and the next. Experiment with different durations.

14. EXAMPLE 5: The Traffic signal V4

Since you now know how the **for...()** function works, this is the ideal time to design the final version (V4) of the traffic signal project. It doesn't provide any functional improvements to the previous version, but technically speaking it does provide the best solution of all. Have a look.

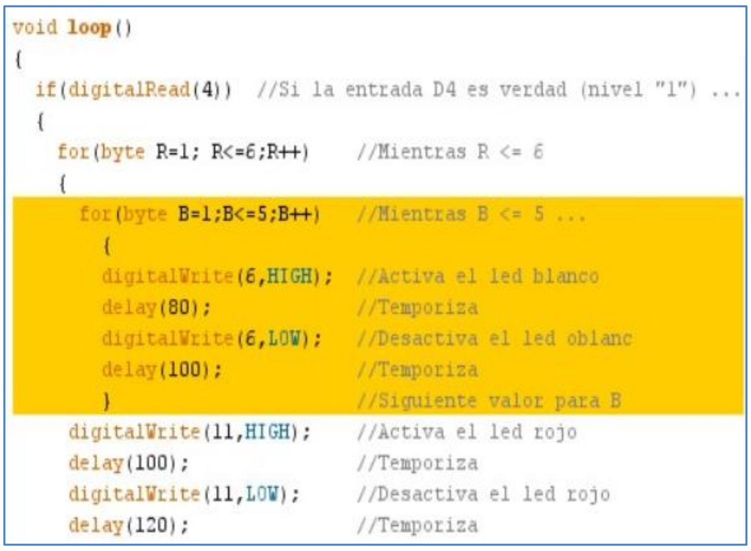
If you have a close look at the program you'll realize why it's the best solution. That's right: the two **for()** loops generate the sequence of tones emitted every time the green and amber lights go on. The program therefore has fewer functions, uses less flash memory and is consequently more efficient.

15. EXAMPLE 6: BURSTS

for(…)loops can be nested. This means that we can place **for(…)** loops within other **for(…)** loops, and other **for()** loops within those ones and so on and so forth.

This program is a simple example of two nested **for(…)** loops. They make two lights flash on and off each time you press a pushbutton connected to pin 4.

The red LED flashes once for every five flashes of the white LED; the sequence terminates with the sixth 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
```

Figure 6

There are two **for(…)** loops inside the **if(…)** function. The loops are executed every time someone presses the pushbutton connected to pin 4. The "*B*" variable controls the inner loop. The variable goes from 1 to 5. It generates a burst of five flashes from the white LED.

The "*A*" variable goes from 1 to 6 and controls the outer **for(…)** loop. As it encloses the inner loop it goes around six times. Each time it goes round the red LED flashes.

When two or more for(…) loops are nested, the inner loop is resolved first and then the next one and so on until arriving at the outer ring.

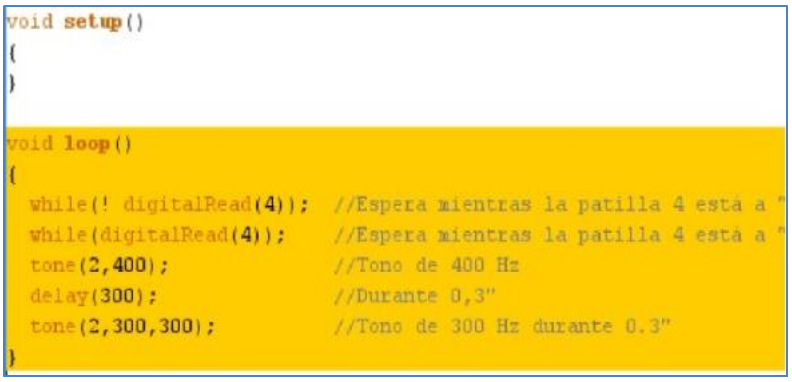
16. EXAMPLE 7: Electric bell V3

Here's version V3 of our old friend, the electric bell; it's the final and definitive one. The important thing about this example is the way we treat the input signal on pin 4. It's no longer enough to press the pushbutton and set a level "1". No, we have to take our finger off it and set the rest level "0".

This task is known as "pulse detection". It's used a lot in control systems where the input signal isn't made up of a static logic level like "1" o el "0". There are lots of input peripherals that give off pulses. A pulse is a complete transition from level "0" to level "1" and back again to "0" (0-1-0) or the other way round (1-0-1).

The while() function is ideal for detecting these situations and lots of others. Have a look at the program in the figure on the right.

The first while() function waits as long as the number 4 digital input is on level "0". The second while() function waits as long as the number 4 digital input is on level "1". Conclusion: until you press the pushbutton (level "1") and then take your finger off it (level "0"), it doesn't make a sound. Try it and pay attention to this detail; this is what makes it different to the electric bell in the V2 version.



```
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"
}
```

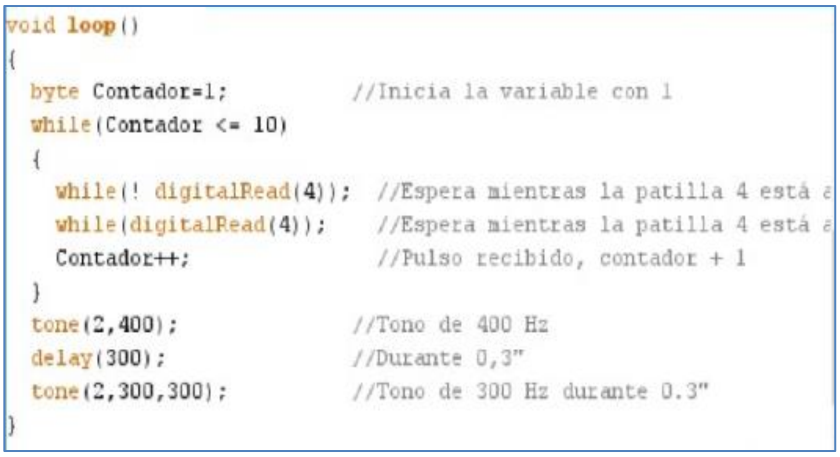
Figure 7

17. EXAMPLE 8: METER

This is a practical application. Imagine an access control system for an auditorium with a capacity of ten people. A sensor connected to pin 4 detects when a person goes by and generates a "pulse" for each person that enters. When the auditorium is full, the device gives a beep to tell the audience that the performance is about to begin. Have a look at the **loop()** function in the figure on the left.

The functions contained in the **while()** function are executed as long as the "counter" variable is less than or equivalent to 10. This variable increases each time a 0-1-0 pulse is detected on pin 4. Two separate **while()** functions are also used for detection.

When the sensor has generated ten pulses, which means there are ten people in the auditorium, it gives a beep.



```
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"
}
```

Figure 8

- **Now it's your turn**

Once you've recorded the program make sure it works properly. All you have to do is press the pushbutton on pin 4. This simulates the people counter sensor. Is it working properly? Is it? No, I don't think so...

If you have a good look you'll notice that it seems to count fewer people than there really are. Or in other words, when five or six people have gone past the beeper goes off. This defect is due to a problem known as the "rebound effect" and it occurs a lot with pushbuttons and switches.

Even though you only press the button once, the input pin on the controller doesn't just receive a single 0-1-0 impulse, it receives several. This is due to the fact that the metal contacts on the pushbutton take a while to re-establish themselves each time you push the button. Have a look at the figure on the left.

Even though this time is just a few milliseconds, our Arduino is a great deal faster and detects all those pulses. What this means is that although you only press the pushbutton once, Arduino may detect this as one, two or more pulses.

One way of avoiding this is by inserting a short pause. When a change of state in the pushbutton is detected we insert a 20 mS pause before waiting for the next change. This way we avoid any rebounds during that lapse of time. Make the following changes in the EXAMPLE 8 program, record it and then check to see if it counts properly.

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. EXAMPLE 9: TIME

Do you remember the **millis()** function? That was the one that enabled you to know how much time goes by between the moment you connect Arduino and/or the moment you reboot it. Okay then, we're going to use it in conjunction with the **switch(…) / case** function to measure different time lapses.

That's right: after the Arduino reboot sequence we're going to measure how much time elapses. After one second the white LED goes on. After two, the green LED goes on, after three, the amber LED and after four, the red LED. Last of all, when 8 seconds have elapsed since the system was rebooted, all the LEDs go out and a tone sounds. Have a close look at the **loop()** function in the program in the figure.

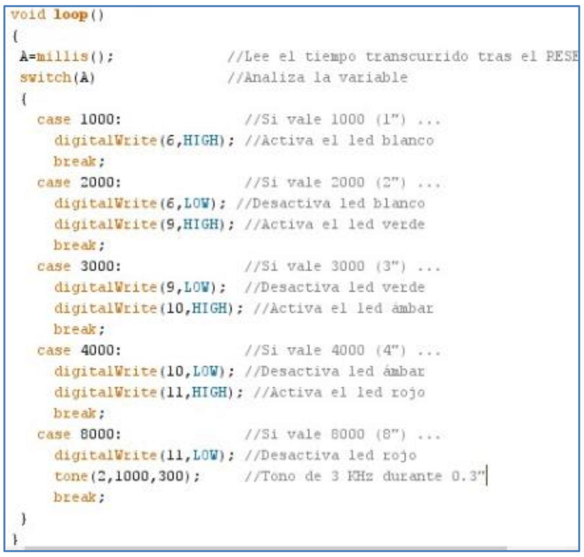
The **A=milis()** function stores the milliseconds that have elapsed since the system reboot in the "A" variable. The **switch(A)** function analyses its contents. If the value of the contents is 1000 (**case 1000**:), the white LED goes on. A second has elapsed (1000 mS). If the value of the contents is 2000 (**case 2000**:), the green LED goes on; if the value is 3000, the amber light goes on and if the value is 4000, the red LED goes on. Finally, if the value of the contents is 8000 (**case 8000**:), 8 seconds will have elapsed. All the LEDs go off and a 1 KHz tone is emitted.

The program keeps running but the time elapsed now exceeds 8000 ms (8 seconds) so the "A" variable no longer meets any of the five conditions.

Do you know how long you'd have to wait for the variable to meet the five conditions again?

Let's work it out. The **millis()** function returns an unsigned long integer of 32 bits. In other words, it returns a value of between 0 and 4,294,967,295 thousandths of a second which equals 4,294,967 seconds. Bearing in mind that a day has 86,400 seconds (24 * 60 * 60), it'd take 50 days for the **millis()** function to overflow and go back to 0.

If you want to go back and see the LED light-up sequence, you can wait all that time or just reboot the system. Whatever you reckon…



```
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;
 }
}
```

Figure 9

REFERENCES

BOOKS

[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

WEB SITES

[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