ΕΝΟΤΗΤΑ 4 ΛΗΨΗ ΑΠΟΦΑΣΕΩΝ ΚΑΙ ΣΥΝΑΡΤΗΣΕΙΣ ΕΛΕΓΧΟΥ

ΣΤΟΧΟΙ

Να μελετηθούν οι συναρτήσεις που ελέγχουν την ροή και την εκτέλεση ενός προγράμματος. Λέγονται «συναρτήσεις ελέγχου» και είναι διαθέσιμες σε όλες τις γλώσσες προγραμματισμού. Το Arduino δεν αποτελεί εξαίρεση.

Τα παραδείγματα με τα οποία ασχοληθήκατε μέχρι τώρα αφορούσαν προγράμματα που περιλάμβαναν συναρτήσεις που εκτελέστηκαν διαδοχικά από την πρώτη στην τελευταία. Από εδώ και στο εξής τα προγράμματα πρόκειται να μπορούν να λαμβάνουν αποφάσεις και να είναι κατά κάποιο τρόπο «ευφυή». Θα υπάρχουν συναρτήσεις που θα εκτελούνται κάτω από συγκεκριμένες συνθήκες.

ΕΝΟΤΗΤΑ ΘΕΩΡΙΑΣ

- ΤΕΛΕΣΤΕΣ ΣΥΓΚΡΙΣΗΣ
- ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ
- ΣΥΝΘΕΤΟΙ ΤΕΛΕΣΤΕΣ
- Η ΣΥΝΑΡΤΗΣΗ IF(…)
- Η ΣΥΝΑΡΤΗΣΗ IF(…) ELSE
- Η ΣΥΝΑΡΤΗΣΗ FOR(…)
- Η ΣΥΝΑΡΤΗΣΗ WHILE(…)
    - ΠΑΡΑΛΛΑΓΕΣ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ WHILE
- Η ΣΥΝΑΡΤΗΣΗ SWITCH(…) / CASE
- ΆΛΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΕΛΕΓΧΟΥ
    - Η ΣΥΝΑΡΤΗΣΗ DO … WHILE(…)
    - Η ΣΥΝΑΡΤΗΣΗ BREAK
    - Η ΣΥΝΑΡΤΗΣΗ RETURN
    - Η ΣΥΝΑΡΤΗΣΗ GOTO

ΕΝΟΤΗΤΑ ΕΞΑΣΚΗΣΗΣ

- ΠΑΡΑΔΕΙΓΜΑ 1: Ηλεκτρικό Κουδούνι V2
- ΠΑΡΑΔΕΙΓΜΑ 2: The Fairy Lights V2
- ΠΑΡΑΔΕΙΓΜΑ 3: Σήματα Κυκλοφορίας V3
- ΠΑΡΑΔΕΙΓΜΑ 4: Ο Ηλεκτρικός Φάρος
- ΠΑΡΑΔΕΙΓΜΑ 5: Σήματα Κυκλοφορίας V4
- ΠΑΡΑΔΕΙΓΜΑ 6: Ριπές
- ΠΑΡΑΔΕΙΓΜΑ 7: Ηλεκτρικό Κουδούνι V3
- ΠΑΡΑΔΕΙΓΜΑ 8: Μετρητής
- ΠΑΡΑΔΕΙΓΜΑ 9: Χρόνος

ΥΛΙΚΟ ΕΞΑΣΚΗΣΗΣ

-- Η/Υ φορητός ή επιτραπέζιος

-Περιβάλλον εργασίας Arduino IDE. Αυτό θα πρέπει να περιλαμβάνει το συμπληρωματικό υλικό που έχει ήδη εγκατασταθεί και διαμορφωθεί.

-Ελεγκτής Arduino UNO

-Καλώδιο USB

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΝΟΤΗΤΑ ΘΕΩΡΙΑΣ

Οι άνθρωποι καθημερινά παίρνουν αποφάσεις και κάποιοι από εμάς παίρνουμε περισσότερες από τους υπόλοιπους.

Μπορούμε να αναλύσουμε το περιβάλλον και τις περιστάσεις και να λάβουμε αποφάσεις με βάση τα ενδιαφέροντά μας, τα συναισθήματα, τις ικανότητές μας, τη διαίσθηση, τις δεσμεύσεις κλπ. Αυτές οι αποφάσεις μας αναγκάζουν να ενεργούμε με διαφορετικούς τρόπους ανάλογα με το τι μας ταιριάζει.

Τα προγράμματα στα οποία δουλέψατε μέχρι τώρα, εκτελούνταν ακολουθιακά. Αυτό σημαίνει ότι οι εντολές εκτελούνταν η μία μετά την άλλη από την πρώτη προς την τελευταία.

Παρ 'όλα αυτά, το Arduino, όπως και οποιοσδήποτε άλλος ελεγκτής έχει την ικανότητα να παίρνει αποφάσεις και να εκτελεί προγράμματα ή να διεκπεραιώνει τις κατάλληλες διεργασίες για κάθε περίσταση. Όπως σίγουρα γνωρίζετε, το Arduino δεν έχει κανένα συναίσθημα ή διαίσθηση, δεν είναι καν ευφυές. Είναι ένα μηχάνημα και το μόνο που ξέρει πώς να κάνει, είναι να δουλέψει με αριθμούς. Λαμβάνει αποφάσεις βασισμένες σε: λογικούς και αριθμητικούς υπολογισμούς, συγκρίσεις, καταστάσεις σημάτων εισόδου, αναλογικές τιμές από ένα αισθητήρα κλπ ...

1 ΤΕΛΕΣΤΕΣ ΣΥΓΚΡΙΣΗΣ

Όπως γνωρίζετε το Arduino μπορεί να εκτελέσει αριθμητικές πράξεις όπως πρόσθεση και αφαίρεση. Γνωρίζετε επίσης ότι τα αποτελέσματα μπορούν να εκφράζονται ως σταθερές ή μεταβλητές ή συνδυασμός και των δύο.

Το Arduino επίσης γνωρίζει πώς να κάνει συγκρίσεις ανάμεσα σε αριθμούς ή αποτελέσματα συγκεκριμένων συναρτήσεων. Παρακάτω βρίσκονται διάφοροι τελεστές σύγκρισης καθώς και τα αντίστοιχα σύμβολα αναπαράστασης τους:

ΤΕΛΕΣΤΗΣ	ΣΥΜΒΟΛΟ	ΤΕΛΕΣΤΗΣ	ΣΥΜΒΟΛΟ
Ίσο	= =	Διάφορο	!=
Μικρότερο	<	Μεγαλύτερο	>
Μικρότερο ή ίσο με	< =	Μεγαλύτερο ή ίσο	>=

Ανεξάρτητα από την σύγκριση υπάρχουν πάντα δύο αποτελέσματα: “true” («Αληθές») ή “false” («Ψευδές»). Δείτε τα παρακάτω παραδείγματα:

Υποθέστε ότι...

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

Τότε...

```
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 αν το pin 4 είναι στο επίπεδο "1"
```

2 ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ

Είναι ακόμη δυνατό να συσχετιστούν μερικές από τις προηγούμενες συγκρίσεις μεταξύ τους. Το Arduino διαθέτει τρείς λογικούς ("boolean") τελεστές για αυτή την περίπτωση, όπως τους βλέπετε παρακάτω μαζί με τα αντίστοιχα σύμβολα αναπαράστασης τους:

ΤΕΛΕΣΤΗΣ	ΣΥΜΒΟΛΟ
NOT	!
AND	&&
OR	||

Με τον ίδιο τρόπο, όταν δύο ή περισσότερες εκφράσεις σχετίζονται με αυτούς τους λογικούς τελεστές υπάρχουν δύο μόνο πιθανά αποτελέσματα: "true" ή "false". Σας προτείνουμε να χρησιμοποιείτε παρενθέσεις για να ομαδοποιήσετε κάθε σχέση: θα διευκολύνει την ανάγνωσή τους και θα αποφύγει λάθη. Ρίξτε μια ματιά στα ακόλουθα παραδείγματα:

(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 αν και τα δύο pins 4 και 8,**

//είναι στο επίπεδο “1”

3 ΣΥΝΘΕΤΟΙ ΤΕΛΕΣΤΕΣ

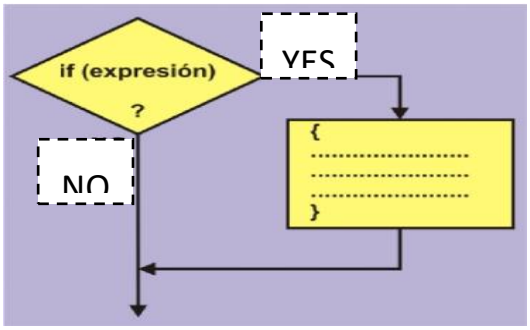
Πολλές φορές θα κάνετε πολύ απλές πράξεις με μια μεταβλητή και το αποτέλεσμα θα καταλήξει στην ίδια μεταβλητή. Θυμηθείτε ότι μπορείτε να χρησιμοποιήσετε τους λεγόμενους "σύνθετους τελεστές" για να απλοποιήσετε αυτές τις εκφράσεις. Ακολουθεί μια περίληψη αυτών στον παρακάτω πίνακα:

ΛΕΙΤΟΥΡΓΙΑ	ΤΕΛΕΣΤΗΣ	ΠΑΡΑΔΕΙΓΜΑ	ΙΣΟ ΜΕ
++	αυξάνει κατά μία μονάδα	X++	X= X + 1
- -	μειώνει κατά μια μονάδα	Y- -	Y = Y - 1
+ =	πρόσθεση	X+=Y	X = X + Y
-=	αφαίρεση	X-= 3	X = X - 3
***=**	πολλαπλασιασμός	X *= Y	X = X * Y
/=	διαίρεση	X /= 5	X = X / 5

4 Η ΣΥΝΑΡΤΗΣΗ IF(...)

Είναι η πιο βασική και σημαντική συνάρτηση ελέγχου. Επιτρέπει την αξιολόγηση των εκφράσεων και στη συνέχεια τη λήψη αποφάσεων. Στον υπολογισμό, το σύμβολο στην Εικόνα 1 χρησιμοποιείται για να αντιπροσωπεύει τη λήψη απόφασης σε διαγράμματα γνωστά ως διαγράμματα ροής. Ο ελεγκτής αξιολογεί μια έκφραση το ίδιο ή και πιο πολύπλοκη από εκείνες που έχετε ήδη δει. Αν το αποτέλεσμα είναι «True” ("αληθές"), εκτελεί όλες τις λειτουργίες μέσα στα άγκιστρα "{...}". Εάν το αποτέλεσμα είναι “False” ("ψευδές"), ο ελεγκτής δεν τις εκτελεί και το πρόγραμμα συνεχίζεται.

Θυμηθείτε αυτό: οποιαδήποτε έκφραση μπορεί να σχηματιστεί με αριθμητικές πράξεις μεταξύ σταθερών και / ή μεταβλητών που σχετίζονται μεταξύ τους, με τη βοήθεια τελεστών σύγκρισης οι οποίοι με τη σειρά τους μπορούν να συσχετιστούν με λογικούς ή Boolean τελεστές. Σκεφτείτε το και ρίξετε μια ακόμα ματιά στα προηγούμενα παραδείγματα. Αυτό είναι σημαντικό ... δεν υπάρχει βιασύνη!



Εικόνα 1

Σύνταξη:

if(έκφραση)

{

….

….

}

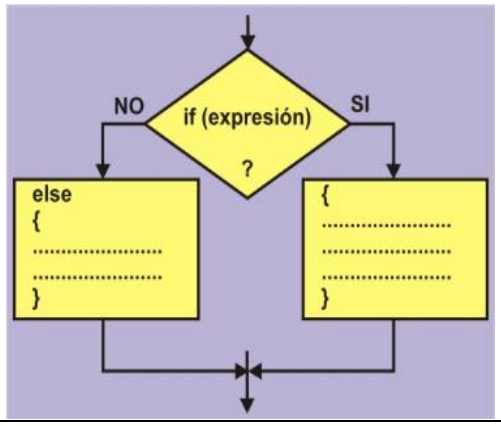
έκφραση: καθορίζει την συνθήκη που πρέπει να αξιολογήσει ο ελεγκτής Arduino. Μπορεί να είναι η σχέση μεταξύ σταθερών και / ή μεταβλητών. Μπορεί να υπάρχουν μία ή περισσότερες συγκρίσεις που σχετίζονται μεταξύ τους από λογικούς τελεστές. Μπορούν επίσης να συγκριθούν τα αποτελέσματα των αριθμητικών πράξεων ή άλλων συναρτήσεων της γλώσσας Arduino.

άγκιστρα: τα άγκιστρα περικλείουν τις συναρτήσεις που ο ελεγκτής πρέπει να εκτελέσει αν η συνθήκη είναι "true" («αληθής»). Μπορείτε να μην τα χρησιμοποιήσετε αν υπάρχει μόνο μια συνάρτηση προς εκτέλεση.

Παράδειγμα:

```
 void loop()
{
    if((A>B) || (C < 25))          //Αν η συνθήκη είναι αληθής…
    {
    digitalWrite(6,HIGH);          //θέσε το pin 6 σε HIGH
```

```
    C=25;                    //Η τιμή 25 αποθηκεύεται στην
                               μεταβλητή C
    }
    ....                     //Συνεχίζεται η εκτέλεση....
}
```



Συμβουλή: Επειδή η χρήση των άγκιστρων διαφέρει, είναι καλή πρακτική προγραμματισμού κατά την εισαγωγή μιας δομής που απαιτεί άγκιστρα, να πληκτρολογείτε το άγκιστρο κλεισίματος αμέσως μετά την πληκτρολόγηση του άγκιστρου ανοίγματος. Στη συνέχεια μεταξύ των άγκιστρων αρχίστε να εισάγετε δηλώσεις. Στο παράδειγμα, τα άγκιστρα που ανοίγουν και κλείνουν τη συνάρτηση loop() είναι εύκολο να διακριθούν από τα άγκιστρα που περικλείουν τις συναρτήσεις που πρέπει να εκτελεστούν αν η συνθήκη if () είναι αληθής.

5 Η ΣΥΝΑΡΤΗΣΗ IF(…) ELSE

Αυτή είναι ακόμα μια συνάρτηση ελέγχου. Προέρχεται από την προηγούμενη συνάρτηση **if(…).** Μας λέει τι να κάνουμε αν η συνθήκη είναι ψευδής (**else**). Δείτε την Εικόνα 2 στα αριστερά. Η συνάρτηση αξιολογεί την έκφραση ή την συνθήκη. Αν είναι «αληθής» όλες οι συναρτήσεις μέσα στα άγκιστρα "{…}" εκτελούνται ακριβώς με το ίδιο τρόπο όπως η συνάρτηση **if(…).**

Εικόνα 2

Αν η συνθήκη είναι «ψευδής» εκτελούνται όλες οι συναρτήσεις που περικλείονται στα άγκιστρα μετά το **else** {…}. Όταν οι συναρτήσεις εκτελεστούν, ανεξάρτητα από το αν η συνθήκη είναι αληθής ή ψευδής η εκτέλεση του προγράμματος συνεχίζεται.

Σύνταξη:

if(έκφραση)

{

….

….

}

else

{

….

….

}

έκφραση: καθορίζει την συνθήκη που πρέπει να αξιολογήσει ο ελεγκτής Arduino. Μπορεί να είναι η σχέση μεταξύ σταθερών και / ή μεταβλητών. Μπορεί να υπάρχουν μία ή περισσότερες συγκρίσεις που σχετίζονται μεταξύ τους από λογικούς τελεστές. Μπορούν επίσης να συγκριθούν τα αποτελέσματα των αριθμητικών πράξεων ή άλλων συναρτήσεων της γλώσσας Arduino.

άγκιστρα: τα άγκιστρα περικλείουν τις συναρτήσεις που ο ελεγκτής πρέπει να εκτελέσει αν η συνθήκη είναι “true” ή «false» (**else**). Μπορείτε να μην τα χρησιμοποιήσετε αν υπάρχει μόνο μια συνάρτηση προς εκτέλεση, σε κάθε περίπτωση.

Παράδειγμα:

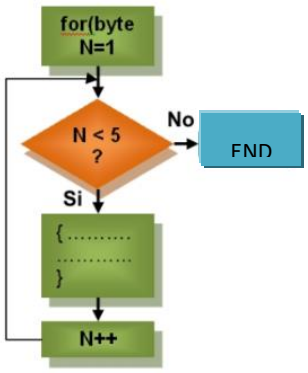
```
if(digitalRead(4) == 1)                    //Αν το pin 4 είναι “1” …
  {
        digitalWrite(6,HIGH);              //Ενεργοποίησε το pin 6
        tone(2,2000,400);                  //Παράγει ένα τόνο2 KHz στο pin 2
        }
```

```
else                    //…και αν είναι ψευδής…
digitalWrite(6,LOW);    //Απενεργοποίησε την έξοδο 6
….                      //Συνεχίζεται η εκτέλεση του προγράμματος
```

6 Η ΣΥΝΑΡΤΗΣΗ FOR():

Η συνάρτηση μας επιτρέπει να δημιουργούμε ελεγχόμενους βρόχους. Ένας βρόχος επαναλαμβάνει ένα μπλοκ εντολών που περικλείονται από άγκιστρα. Δείτε το διάγραμμα ροής στην Εικόνα 3.

Θέτουμε μια αρχική τιμή 1 στην μεταβλητή N. Η τιμή της μεταβλητής ελέγχεται και αν μικρότερη του 5, εκτελούνται οι συναρτήσεις που βρίσκονται ανάμεσα σα άγκιστρα. Η τιμή της μεταβλητής N αυξάνει αυτόματα κατά 1 (N++) και ελέγχεται ξανά. Αν η συνθήκη είναι αληθής το μπλοκ των συναρτήσεων και η αύξηση της τιμής της μεταβλητής εκτελούνται και έπειτα ελέγχεται ξανά η συνθήκη. Όταν η συνθήκη γίνει ψευδής, η εκτέλεση του βρόχου σταματά. Οι συναρτήσεις που περικλείονται στα άγκιστρα του παραδείγματος στην Εικόνα 3 εκτελούνται τέσσερις φορές, όσο δηλαδή η τιμή της μεταβλητής N είναι μικρότερη του 5.



Εικόνα 3

Σύνταξη:

*for(αρχικοποίηση,**συνθήκη,αύξηση**)*

{

….

….

}

αρχικοποίηση: είναι η έκφραση που αποδίδει αρχική τιμή σε μια μεταβλητή. Αυτό συμβαίνει μόνο στην αρχή του βρόγχου.

συνθήκη: είναι η συνθήκη που ελέγχεται. Αν είναι “αληθής”, τότε εκτελούνται το μπλοκ εντολών και η αύξηση της τιμής της. Όταν η συνθήκη γίνει “ψευδής”, ο βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζεται. Η συνθήκη ελέγχεται κάθε φορά που επαναλαμβάνεται ο βρόχος.

αύξηση: είναι η έκφραση που αλλάζει την τιμή της μεταβλητής. Η έκφραση εκτελείται κάθε φορά που επαναλαμβάνεται ο βρόχος.

curly braces: περικλείουν όλες τις συναρτήσεις που αποτελούν τον βρόχο. Οι συναρτήσεις εκτελούνται για συγκεκριμένο αριθμό επαναλήψεων.

Παράδειγμα:

```
for (int N = 1; N < 5; N=N+1)          //Θέτει τις συνθήκες για τον βρόγχο
{
digitalWrite(6,HIGH);                  //Ενεργοποιεί το pin 6
delay (150);                           //Προκαλεί παύση του προγράμματος για 0.15”
digitalWrite(6,LOW);                   //Απενεργοποιεί το pin 6
delay (1000);                          // Προκαλεί παύση του προγράμματος για 1”
}
….                                     // Συνεχίζεται η εκτέλεση του προγράμματος
```

Η ακολουθία “ενεργοποίηση, παύση, απενεργοποίηση, παύση” στο pin 6 επαναλαμβάνεται τέσσερις φορές.

Μπορείτε να χρησιμοποιήσετε όλα τα είδη των εκφράσεων που μελετήσαμε στην αρχή αυτής της ενότητας για να ρυθμίσετε τις παραμέτρους (αρχικοποίηση, συνθήκη, αύξηση) για κάθε βρόγχο for().

Θα μπορούσαν να είναι αριθμητικές, λογικές ή Boolean τιμές ή συγκρίσεις μεταβλητών ή / και σταθερών. Απλώς υποθέστε τα εξής:

```
int A = 5;

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

{

….

….

}
```

- **Άσκηση**

Ποια είναι η αρχική τιμή; _____

Ποια είναι η τελική τιμή; _____

Πόσο αυξάνεται η τιμή της μεταβλητής κάθε φορά; _____

Πόσες φορές επαναλαμβάνεται ο βρόχος; _____

7 Η ΣΥΝΑΡΤΗΣΗ WHILE():

Η συνάρτηση **while** είναι παραλλαγή της συνάρτησης for() και χρησιμοποιείται επίσης για βρόχους όπου πολλές συναρτήσεις εκτελούνται πολλές φορές.

Σύνταξη:

while(συνθήκη)

{

….

….

}

συνθήκη: εδώ βρίσκεται η υποθετική έκφραση. Θα κάνει συνεχώς επαναλήψεις μέχρι η έκφραση που βρίσκεται μέσα στις παρενθέσεις γίνει ψευδής. Όταν αυτό συμβεί, ο βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζεται.

Παράδειγμα:

```
int N = 6 while (N > 0)                    //Όσο το N είναι μεγαλύτερο από 0
…
{
digitalWrite(6,HIGH);                      //Ενεργοποίησε το pin 6
delay (150);                               //Κάνε παύση για 0.15”
digitalWrite(6,LOW);                       //Απενεργοποίησε το pin 6
Page 11 (cont.)
delay (1000);                              // Κάνε παύση για 1”
n--;                                       //Η επόμενη τιμή του N (N = N – 1) }
….
….                                         //Το πρόγραμμα συνεχίζει να εκτελείται
```

Α. ΑΛΛΕΣ ΜΟΡΦΕΣ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ WHILE()

Είναι πολύ συνηθισμένο να χρησιμοποιούμε την συνάρτηση while() όπως παρακάτω. Δεν υπάρχουν άγκιστρα και ο βρόγχος εκτελείται μέχρι η συνθήκη να γίνει ψευδής. Δείτε τα παρακάτω παραδείγματα και προσέξτε που βρίσκεται το ‘;’.

```
while(digitalRead(4)==1);
```

Η συνάρτηση while() εκτελείται συνεχώς για όσο διάστημα το pin 4 είναι στο επίπεδο “1”. Για να το θέσουμε με διαφορετικό τρόπο, το πρόγραμμα θα συνεχίσει να εκτελείται όταν το pin 4 πάει στο επίπεδο “0”.

```
while(! digitaRead(4));
```

Αυτό είναι όπως το προηγούμενο παράδειγμα αλλά ανάποδα. Η συνάρτηση while() εκτελείται συνεχώς όσο το pin 4 ΔΕΝ είναι στο επίπεδο “1”, ή με άλλα λόγια όσο είναι στο επίπεδο “0”. Περιμένει επομένως μέχρι το pin 4 να πάει στο επίπεδο “1” για να συνεχίσει να εκτελεί το πρόγραμμα.

```
while(digitalRead(4)==0);
```

Αυτό είναι το ίδιο με το προηγούμενο παράδειγμα, δηλαδή περιμένει μέχρι το pin 4 να πάει στο επίπεδο “1” για να συνεχίσει την εκτέλεση του προγράμματος.

```
while(1);
```

Ένας ατέρμονας βρόγχος. Η συνθήκη είναι πάντοτε αληθής (1), και η συνάρτηση while() εκτελείται συνεχώς.

8 Η ΣΥΝΑΡΤΗΣΗ SWITCH() / CASE

Αυτή η συνάρτηση μας επιτρέπει να επιλέξουμε ανάμεσα σε διαφορετικούς τρόπους εκτέλεσης διάφορων γκρουπ εντολών. Μια έκφραση switch συγκρίνει την τιμή μιας μεταβλητής με τις τιμές που καθορίζονται στις δηλώσεις case. Όταν εντοπιστεί μια δήλωση case της οποίας η τιμή ταιριάζει με εκείνη της μεταβλητής, εκτελείται ο κώδικας σε αυτή την περίπτωση. Η λέξη-κλειδί break χρησιμοποιείται για την έξοδο από την συνάρτηση switch () και βρίσκεται συνήθως στο τέλος κάθε περίπτωσης. Είναι κάτι σαν αυτό:

‘’Εάν η τιμή της μεταβλητής είναι X εκτέλεσε αυτές τις συναρτήσεις. Εάν η τιμή της μεταβλητής είναι Y, εκτέλεσε κάποιες άλλες συναρτήσεις αυτές οι άλλες. Αν είναι Z εκτέλεσε κάποιες άλλες κλπ... "

Σύνταξη:

```
switch(μεταβλητή)
{
    case X:
        ….;
        ….;
        break;
    case n:
        ….;
        ….;
        break;
    default:
        ….;
}
```

μεταβλητή: είναι η μεταβλητή της οποίας η τιμή πρόκειται να συγκριθεί με τις τιμές που αναφέρονται σε κάθε περίπτωση (“case”).

case: είναι η τιμή που θα συγκρίνεται με αυτήν της μεταβλητής. Όταν τα περιεχόμενα της μεταβλητής συμπίπτουν με μία από τις τιμές, εκτελούνται όλες οι λειτουργίες μεταξύ αυτής της περίπτωσης και της έκφρασης διακοπής.

default: είναι προαιρετικό. Αν καμία από τις τιμές σε κάθε περίπτωση («case») δεν συμπίπτει με αυτήν της μεταβλητής, τότε εκτελούνται οι συναρτήσεις που βρίσκονται μετά το default.

Παράδειγμα:

```
switch(A)                    //Η μεταβλητή που θα συγκριθεί
```

```
{
        case 1:                         //Αν η τιμή της μεταβλητής A είναι 1...
                digitalWrite(6,HIGH);   //Ενεργοποίησε το pin 6
                tone(2,200,200);        //Στείλε ένα τόνο στο pin 2
                break;                  //Βγες από την εκτέλεση της switch
        case 3:                         //Αν η τιμή της μεταβλητής A είναι 3…
                B=digitalRead(7);       //Διάβασε την κατάσταση του pin 7
                break;                  //Βγες από την εκτέλεση της switch
        case 124:                       //Αν η τιμή της μεταβλητής A είναι 124…
        B=12*4;                         //Η τιμή της μεταβλητής B γίνεται 48
        digitalWrite(11,HIGH)           //Ενεργοποίησε το pin 11
        break;                          //Βγες από την εκτέλεση της switch
default:                                // Αν δεν ταιριάζει τίποτα άλλο …
        digitalWrite(6.LOW);            //Απενεργοποίησε το pin 6
        digitalWrite(11,LOW);           //Απενεργοποίησε το pin 11
}
```

9 ΑΛΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΕΛΕΓΧΟΥ

Οι συναρτήσεις ελέγχου που είδατε είναι οι πιο σημαντικές και αυτές που θα χρησιμοποιείτε. Ωστόσο, η γλώσσα προγραμματισμού Arduino διαθέτει και άλλες συναρτήσεις. άλλα. Είναι στην ευχέρεια σας αν τις χρησιμοποιήσετε. Ίσως όσο μαθαίνετε περισσότερα και βελτιώνετε την τεχνική σας, να διακρίνετε την χρησιμότητα τους.

A. Η ΣΥΝΑΡΤΗΣΗ DO...WHILE()

Ο βρόχος **do** λειτουργεί με τον ίδιο τρόπο όπως ο βρόχος **while()**, με την διαφορά ότι η συνθήκη ελέγχεται στο τέλος του βρόγχου. Έτσι ο βρόγχος **do** πάντοτε θα εκτελείται τουλάχιστον μια φορά.

Σύνταξη:

do

{

….

….

} while(συνθήκη)

Β. Η ΣΥΝΑΡΤΗΣΗ BREAK

Η συνάρτηση **break** χρησιμοποιείται για την έξοδο από ένα βρόχο **for(), while()** ή **do()**, παρακάμπτοντας την συνθήκη του βρόγχου. Χρησιμοποιείται επίσης για την έξοδο από την συνάρτηση **switch() / case**.

Σύνταξη:

break;

Γ. Η ΣΥΝΑΡΤΗΣΗ RETURN

Αυτή τερματίζει την εκτέλεση μιας συνάρτησης και επιστρέφει μια τιμή από οποιαδήποτε συνάρτηση δημιουργήθηκε από τον χρήστη, στο πρόγραμμα που την κάλεσε . Θα μάθετε πώς να δημιουργείτε δικές σας συναρτήσεις λίγο αργότερα.

Σύνταξη:

return; return τιμή;

τιμή: είναι η τιμή που επιστρέφει η συνάρτηση όταν επιστρέφει στο πρόγραμμα που την κάλεσε. Είναι προαιρετική και μπορεί να είναι μια σταθερά ή να αποτελεί μέρος μιας μεταβλητής.

Δ. Η ΣΥΝΑΡΤΗΣΗ GOTO

Μεταφέρει τη ροή του προγράμματος σε ένα συγκεκριμένο σημείο στο πρόγραμμα.

Σύνταξη:

test:

….

….

….

goto test: // Πραγματοποιείται αυτόματη μετακίνηση σε όποιο σημείο εμφανίζεται η ετικέτα ("test") μέσα στο πρόγραμμα.

ΕΝΟΤΗΤΑ ΕΞΑΣΚΗΣΗΣ

10. ΠΑΡΑΔΕΙΓΜΑ 1: ΗΛΕΚΤΡΙΚΟ ΚΟΥΔΟΥΝΙ V2

Αυτή είναι μια βελτιωμένη έκδοση του παραδείγματος που είδαμε στην Ενότητα 5 που έπρεπε να κάνουμε επανεκκίνηση στο σύστημα κάθε φορά που θέλαμε να χτυπήσουμε το κουδούνι.

Η βελτιωμένη έκδοση μας δίνει επίσης την ευκαιρία να χρησιμοποιήσουμε την συνάρτηση if(). Το πρόγραμμα ελέγχει αν το pin 4 είναι ενεργοποιημένο. Όταν αυτό συμβεί, παράγονται δύο συνεχόμενοι τόνοι 400 and 300 Hz. Αν το pin 4 δεν είναι ενεργοποιημένο, το πρόγραμμα δεν κάνει τίποτα. Δείτε προσεκτικά τα κόκκινα βέλη στην Εικόνα 4.

Όπως σας έχουμε ήδη αναφέρει είναι σκόπιμο να ευθυγραμμίζονται τα άγκιστρα έτσι ώστε η σχέση μεταξύ των δύο να είναι αρκετά ξεκάθαρη, δηλαδή ένα από τα άγκιστρα να ανοίγει και το άλλο να κλείνει.

```
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"
  }
}
```

Εικόνα 4

11. ΠΑΡΑΔΕΙΓΜΑ 2: THE FAIRY LIGHTS V2

Η κίνηση γίνεται από δεξιά προς τα αριστερά ή αντίστροφα ανάλογα με το αν το pin 4 είναι στο επίπεδο “0” ή στο επίπεδο “1”. Αυτή είναι μια πολύ καλή ευκαιρία για να χρησιμοποιήσετε την συνάρτηση if(..) else.

12. ΠΑΡΑΔΕΙΓΜΑ 3: ΣΗΜΑΤΑ ΚΥΚΛΟΦΟΡΙΑΣ V3

Ας συνεχίσουμε να βελτιώνουμε κάποια από τα παραδείγματα που είδαμε στην τελευταία ενότητα. Τώρα θα δούμε το παράδειγμα με τα σήματα κυκλοφορίας. Σε αυτή την έκδοση η ακολουθία

ξεκινά όταν ο πεζός πατά ένα κουμπί που είναι συνδεδεμένο με το pin 4. Δεν χρειάζεται να πατήσετε το κουμπί RESET για να επανεκκινήσετε το σύστημα όπως κάνατε στις προηγούμενες εκδόσεις.

Αν κανένας δεν πιέσει το κουμπί, το σήμα κυκλοφορίας παραμένει κόκκινο. Αυτό είναι άλλο ένα καλό παράδειγμα για το πώς να χρησιμοποιείτε την συνάρτηση if (...) else. Σας προτείνουμε να συγκρίνετε αυτό το παράδειγμα με τις προηγούμενες εκδόσεις. Οι διαφορές είναι μικρές αλλά σημαντικές. Θα βελτιώνουμε το πρόγραμμα βήμα-βήμα.

13. ΠΑΡΑΔΕΙΓΜΑ 4: Ο ΗΛΕΚΤΡΙΚΟΣ ΦΑΡΟΣ

Αυτό είναι ένα καλό παράδειγμα που χρησιμεύει επίσης για μια εισαγωγή στην συνάρτηση for(…). Προσπαθεί να προσομοιώσει έναν φάρο ή έναν ηλεκτρικό φάρο που δίνει ένα συγκεκριμένο αριθμό φωτεινών αναλαμπών κατά τη διάρκεια μιας δεδομένης χρονικής περιόδου.

Η συνάρτηση loop() στην Εικόνα 5 περιέχει το κυρίως πρόγραμμα. Η συνάρτηση for(…) θέτει ως αρχική τιμή στην μεταβλητή την τιμή 1 (N=1), ο βρόγχος εκτελείται μέχρι η μεταβλητή N να φτάσει την τελική της τιμή (N< Flashes) και επίσης θέτει το βήμα που αυξάνεται η μεταβλητή N (N++) κάθε φορά που εκτελείται ο βρόχος.

Εάν ρίξετε μια προσεκτική ματιά θα δείτε ότι ο βρόχος επαναλαμβάνεται πέντε φορές. Κάθε φορά που επαναλαμβάνεται ο βρόχος, το λευκό LED ανάβει για 0.15" και στη συνέχεια σβήνει για άλλα 0.15". Μόλις τελειώσει ο βρόχος υπάρχει μια παύση 1,5" και η διαδικασία επαναλαμβάνεται.

```
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.
}
```

Εικόνα 5

- **Άσκηση**

Αυτό το πρόγραμμα είναι πολύ απλό. Γιατί δεν προσπαθείτε να αλλάξετε τον αριθμό των αναλαμπών, τον χρόνο που ανάβει και σβήνει το LED και τον χρόνο μεταξύ μιας ριπής αναλαμπών και της επόμενης; Πειραματιστείτε με διαφορετικές χρονικές διάρκειες.

14. ΠΑΡΑΔΕΙΓΜΑ 5: ΣΗΜΑΤΑ ΚΥΚΛΟΦΟΡΙΑΣ V4

Καθώς τώρα ξέρετε πως λειτουργεί η συνάρτηση for...(), είναι ο ιδανικός χρόνος να σχεδιάσουμε την τελική έκδοση (V4) του προγράμματος για τα σήματα κυκλοφορίας. Δεν παρέχει λειτουργικές βελτιώσεις στην προηγούμενη έκδοση, αλλά από τεχνικής απόψεως παρέχει την καλύτερη λύση από όλες.

Αν δείτε προσεκτικά το πρόγραμμα θα καταλάβετε γιατί είναι η καλύτερη λύση. Σωστά καταλάβατε: οι δύο βρόγχοι **for()** παράγουν την ακολουθία των τόνων που εκπέμπονται κάθε φορά που ανάβουν τα πράσινα και πορτοκαλί φώτα. Επομένως, το πρόγραμμα έχει λιγότερες συναρτήσεις, χρησιμοποιεί λιγότερη μνήμη flash και ως εκ τούτου είναι πιο αποτελεσματικό.

15. ΠΑΡΑΔΕΙΓΜΑ 6: ΡΙΠΕΣ

Οι βρόχοι **for(…)** μπορεί να είναι εμφωλευμένοι. Αυτό σημαίνει ότι μπορούμε να βάλουμε ένα βρόχο **for(…)** μέσα σε ένα άλλο και άλλους βρόχους **for(…)** μέσα στους προηγούμενους κοκ.

Αυτό το πρόγραμμα είναι ένα απλό παράδειγμα δυο εμφωλευμένων βρόχων **for(…).** Κάνουν δύο φώτα να αναβοσβήνουν κάθε φορά που πιέζεται ένα κουμπί που είναι συνδεδεμένο στο pin 4.

Το κόκκινο LED αναβοσβήνει μία φορά για κάθε πέντε αναλαμπές του λευκού LED. Η ακολουθία τερματίζεται με την έκτη αναλαμπή.

```
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
```

Εικόνα 6

Υπάρχουν άλλοι δύο βρόχοι **for(…)** εντός της συνάρτησης **if(…).** Οι βρόχοι εκτελούνται κάθε φορά που κάποιος πιέζει το κουμπί που είναι συνδεδεμένο στο pin 4. Η μεταβλητή "B" ελέγχει τον εσωτερικό βρόχο. Η μεταβλητή πηγαίνει από 1 έως 5 και δημιουργεί μια ριπή πέντε αναλαμπών από το λευκό LED.

Η μεταβλητή “A” πηγαίνει από το 1 στο 6 και ελέγχει τον εξωτερικό βρόγχο **for(…).** Καθώς περικλείει τον εσωτερικό βρόγχο εκτελείται για 6 φορές. Κάθε φορά που εκτελείται το κόκκινο LED αναβοσβήνει.

Όταν δύο ή περισσότεροι βρόγχοι for(…) είναι εμφωλευμένοι, εκτελείται πρώτα ο εσωτερικός βρόγχος και έπειτα ο επόμενος κοκ. μέχρι εκτελεστεί ο εξωτερικός

16. ΠΑΡΑΔΕΙΓΜΑ 7: ΗΛΕΚΤΡΙΚΟ ΚΟΥΔΟΥΝΙ V3

Εδώ βρίσκεται η έκδοση V3 του γνωστού σας προγράμματος, που θα είναι και η τελική του. Το πιο σημαντικό σε αυτό το παράδειγμα είναι ο τρόπος που χειριζόμαστε το σήμα εισόδου στο pin 4. Δεν είναι πλέον αρκετό να πατήσετε το κουμπί και να ρυθμίσετε το επίπεδο "1".

Η συγκεκριμένη διεργασία είναι γνωστή ως «ανίχνευση παλμού» (“pulse detection”). Χρησιμοποιείται πολύ στα συστήματα ελέγχου όπου το σήμα εισόδου δεν αποτελείται από ένα στατικό λογικό επίπεδο όπως "1" ή "0". Υπάρχουν πολλά περιφερειακά εισόδου που παράγουν παλμούς. Ένας παλμός είναι μια πλήρης μετάβαση από το επίπεδο "0" στο επίπεδο "1" και πάλι πίσω στο "0" (0-1-0) ή το αντίστροφο (1-0-1).

Η συνάρτηση while() είναι ιδανική για την ανίχνευση αυτών των καταστάσεων και πολλών άλλων. Ρίξτε μια ματιά στο πρόγραμμα στην Εικόνα 7. Η πρώτη συνάρτηση while() περιμένει όσο η ψηφιακή είσοδος 4 βρίσκεται στο επίπεδο "0". Η δεύτερη συνάρτηση while() περιμένει όσο η ψηφιακή είσοδος 4 βρίσκεται στο επίπεδο "1". Συμπέρασμα: το να πατήσετε το κουμπί (επίπεδο "1") και στη συνέχεια να αφαιρέσετε το δάχτυλό σας (επίπεδο "0"), δεν έχει καμία σημασία. Δοκιμάστε το και δώστε προσοχή σε αυτή τη λεπτομέρεια. Αυτό είναι που κάνει το πρόγραμμα διαφορετικό από αυτό της έκδοσης 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"
}
```

Εικόνα 7

17. ΠΑΡΑΔΕΙΓΜΑ 8: ΜΕΤΡΗΤΗΣ

Αυτή είναι μια πρακτική εφαρμογή. Φανταστείτε ένα σύστημα ελέγχου πρόσβασης για μια αίθουσα χωρητικότητας δέκα ατόμων. Ένας αισθητήρας που είναι συνδεδεμένος με τον ακροδέκτη 4 ανιχνεύει πότε ένα άτομο περνάει και δημιουργεί ένα "παλμό" για κάθε άτομο που εισέρχεται. Όταν η αίθουσα γεμίσει, η συσκευή δίνει ένα προειδοποιητικό ήχο (beep) για να πει στο κοινό ότι η παράσταση πρόκειται να αρχίσει. Ρίξτε μια ματιά στην συνάρτηση loop() στην Εικόνα 8.

Οι συναρτήσεις που περιέχονται στην συνάρτηση while() εκτελούνται όσο η μεταβλητή «μετρητής» είναι μικρότερη ή ίση του 10. Αυτή η μεταβλητή αυξάνεται κάθε φορά που ανιχνεύεται ένας παλμός 0-1-0 στο pin 4. Δύο ξεχωριστές συναρτήσεις while() χρησιμοποιούνται επίσης για τον έλεγχο.

Όταν ο αισθητήρας έχει δημιουργήσει δέκα παλμούς – που σημαίνει ότι υπάρχουν δέκα άνθρωποι στην αίθουσα-, δίνει ένα προειδοποιητικό ήχο.

```
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"
}
```

Εικόνα 8

- **Άσκηση**

Μόλις καταγράψετε το πρόγραμμα βεβαιωθείτε ότι λειτουργεί σωστά. Το μόνο που έχετε να κάνετε είναι να πατήσετε το κουμπί στο pin 4. Αυτό προσομοιώνει τον αισθητήρα που μετράει τα άτομα. Λειτουργεί σωστά; Μάλλον όχι …

Αν παρατηρήσετε προσεκτικά, θα παρατηρήσετε ότι φαίνεται να μετράει λιγότερους ανθρώπους από ό, τι πραγματικά υπάρχουν. Ή με άλλα λόγια, όταν έχουν περάσει πέντε ή έξι άνθρωποι ο βομβητής σβήνει. Αυτό το ελάττωμα οφείλεται σε ένα πρόβλημα που είναι γνωστό ως "εφέ ανάκαμψης" (“rebound effect”) και παρουσιάζεται συχνά με τα κουμπιά και τους διακόπτες

Παρόλο που πατάτε μόνο το πλήκτρο μία φορά, ο ακροδέκτης εισόδου του ελεγκτή δε λαμβάνει απλώς ένα μόνο παλμό 0-1-0, αλλά αντίθετα λαμβάνει πολλούς. Αυτό οφείλεται στο γεγονός ότι οι μεταλλικές επαφές του κουμπιού χρειάζονται λίγο χρόνο για να αποκατασταθούν κάθε φορά που πιέζετε το κουμπί. Ρίξτε μια ματιά στην Εικόνα 8.

Παρόλο που αυτός ο χρόνος είναι μόνο μερικά χιλιοστά του δευτερολέπτου, το Arduino είναι πολύ ταχύτερο και ανιχνεύει όλους αυτούς τους παλμούς. Αυτό σημαίνει ότι παρόλο που πιέζετε το πλήκτρο μόνο μία φορά, το Arduino μπορεί να το ανιχνεύσει δύο ή περισσότερους παλμούς, ως ένα.

Ένας τρόπος για να αποφευχθεί αυτό είναι με την εισαγωγή μιας σύντομης παύσης. Όταν εντοπιστεί μια αλλαγή της κατάστασης στο κουμπί, εισάγουμε μια παύση 20 mS πριν αναμείνουμε για την επόμενη αλλαγή. Έτσι αποφεύγουμε οποιαδήποτε «rebound” κατά τη διάρκεια αυτής της περιόδου. Πραγματοποιήστε τις ακόλουθες αλλαγές στο πρόγραμμα του παραδείγματος 8, καταγράψτε το και στη συνέχεια ελέγξτε το για να δείτε αν μετράει σωστά.

```
while(Counter <= 10)
{
        while(! digitalRead(4));          //Περίμενε όσο το pin 4 είναι στο επίπεδο“0”
        delay(20);                        //Παύση για την αποφυγή του rebound
        while(digitalRead(4));            // Περίμενε όσο το pin 4 είναι στο επίπεδο "1"
        delay(20);                        // Παύση για την αποφυγή του rebound
        Counter ++;                       //Λήψη παλμού, ο μετρητής αυξάνει κατά 1
}
```

18. ΠΑΡΑΔΕΙΓΜΑ 9: ΧΡΟΝΟΣ

Θυμάστε την συνάρτηση millis(); Αυτή είναι που σας επιτρέπει να μάθετε πόσος χρόνος περνάει από τη στιγμή που συνδέετε το Arduino ή / και τη στιγμή που το επανεκκινείτε. Θα το χρησιμοποιήσουμε σε συνδυασμό με τη συνάρτηση switch (...) / case για να μετρήσουμε διαφορετικές χρονικές στιγμές.

Μετά την ακολουθία επανεκκίνησης του Arduino, θα μετρήσουμε πόσο χρονικό διάστημα παρήλθε. Μετά από ένα δευτερόλεπτο το λευκό LED ανάβει. Μετά από δύο, ανάβει το πράσινο LED, μετά από τρία το πορτοκαλί LED και μετά από τέσσερα το κόκκινο LED. Τέλος, όταν έχουν παρέλθει 8 δευτερόλεπτα από την επανεκκίνηση του συστήματος, όλα τα LED σβήνουν και ακούγεται ένας ήχος. Δείτε προσεκτικά την συνάρτηση loop() στο πρόγραμμα στην Εικόνα 9.

Η συνάρτηση A=milis() αποθηκεύει στην μεταβλητή A τον χρόνο που πέρασε από την επανεκκίνηση του συστήματος. Η συνάρτηση switch(A) εξετάζει το περιεχόμενο της μεταβλητής A. Αν η τιμή της A είναι 1000 (case 1000:), ανάβει το άσπρο LED. Ένα δευτερόλεπτο έχει περάσει (1000 mS). Αν η τιμή της A είναι 2000 (case 2000:), ανάβει το πράσινο LED. Αν η τιμή της A είναι 3000, ανάβει το πορτοκαλί LED και αν η τιμή της A είναι 4000, ανάβει το κόκκινο LED. Τέλος, αν η τιμή της A είναι 8000 (case 8000:), έχουν παρέλθει 8 δευτερόλεπτα. Όλα τα LEDs σβήνουν και εκπέμπεται τόνος 1 KHz.

Το πρόγραμμα συνεχίζει να εκτελείται αλλά ο χρόνος τώρα υπερβαίνει τα 8000 ms (8 seconds) και έτσι η μεταβλητή “A” δεν δεν πληροί πλέον καμία από τις πέντε προϋποθέσεις.

Ξέρετε πόσο χρόνο θα πρέπει να περιμένετε για να πληροί η μεταβλητή κάποια από τις πέντε προϋποθέσεις;

Ας το δούμε. Η συνάρτηση millis() επιστρέφει ένα μη προσημασμένο μεγάλο ακέραιο (unsigned long integer) μήκους 32 bits. Με άλλα λόγια, επιστρέφει μια τιμή μεταξύ 0 και 4,294,967,295 χιλιοστά ενός δευτερολέπτου, που ισούται με 4,294,967 δευτερόλεπτα. Γνωρίζοντας ότι μία ημέρα έχει 86,400 δευτερόλεπτα (24 * 60 * 60), θα έπαιρνε 50 ημέρες για να γίνει overflow η συνάρτηση millis() και να επέστρεφε ξανά στο 0.

Εάν θέλετε να δείτε ξανά την ακολουθία ανάμματος των LED, μπορείτε είτε να περιμένετε όλη αυτή την ώρα ή απλώς να επανεκκινήσετε το σύστημα. Ό, τι νομίζετε ...

```
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;
 }
}
```

Εικόνα 9

ΑΝΑΦΟΡΕΣ

ΒΙΒΛΙΑ

[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

ΙΣΤΟΣΕΛΙΔΕΣ

[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