ELEC-C9820 ED Workshop, Exercise 3, 21.-25.1.2019

(Based on the Morse example in the Sähköpaja course by Sami Pukkila)

Current version 21st of January 2019 by Salu Ylirisku

A Morse code transmitter

Needed equipment: Arduino + USB cable

Required result

An Arduino system that blinks LED according to a Morse code sequence that is sent from an attached computer.

Step 1: Inspect the Blinking LED example again

The first part of the exercise is the blink the on-board LED, which was tried on the first exercise. Here we analyse the example in more detail before going further. You find the original tutorial at https://www.arduino.cc/en/tutorial/blink. The code, which is written in a C/C++ programming language, reads as follows:

```
01.     const int ledpin = LED_BUILTIN;
02.     void setup() {
03.         pinMode(ledpin, OUTPUT);
04.     }
05.     void loop() {
06.         digitalWrite(ledpin, HIGH);
07.         delay(1000);
08.         digitalWrite(ledpin, LOW);
09.         delay(1000);
10.     }
```

Explanation:

```
01.     const int ledpin = LED_BUILTIN;
```

Row 01 begins with the type qualifier ‘const,’ that defines the type of the variable as a ‘constant’. This means that the value of the variable cannot be changed dynamically while the program is running.

The term ‘int’ after it defines the data type of the variable to be an ‘integer’. Other common data types are *char*, *float* and *double*. Char is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers. Float is used to store decimal numbers (numbers with floating point value) with single precision, while double is It is used to store decimal numbers (numbers with floating point value) with double precision.

The text ‘ledpin’ stands for a variable name. It is assigned the value of LED_BUILTIN with the ‘=’ sign.

LED_BUILTIN is a constant that is defined for the chosen board. For Arduino UNO the value is 13. The text LED_BUILTIN could be basically replaced with 13, but it is a good convention to use the named constants, as they will work properly still if the board is changed.

```
02.     void setup() {
03.         pinMode(ledpin, OUTPUT);
```

```
04.     }
```

Row 02 defines the start of the setup()-function, which is written between the curly brackets {}. setup() is always called when Arduino starts, and it is utilised to load the initial values into the memory.

Row 03 calls a function named ‘pinMode’ with two parameters ‘ledpin’ and ‘OUTPUT’. When Arduino is started, it initializes all pins as a high-impedance INPUT to avoid short circuits. Whenever you need to use a pin for digital output, you need to call this function with the OUTPUT constant. ‘ledpin’ has the value defined in LED_BUILTIN.

Row 04 ends the setup().

Next we shall look at the loop.

```
05. void loop() {
06.     digitalWrite(ledpin, HIGH);
07.     delay(1000);
08.     digitalWrite(ledpin, LOW);
09.     delay(1000);
10. }
```

Row 05 defines the loop()-function, which will run until the device is powered off. It simply goes over and over again through the rows defined inside the curly brackets {}.

Row 06 calls the function ‘digitalWrite()’. It has parameters ledpin and ‘HIGH’. ‘HIGH’ is a constant (typically number 1) that tells Arduino to put the voltage high (5V on 5V boards and 3.3V on 3.3V boards) on that pin. As the on-board LED is on the pin, the LED should light up.

Row 07 (and 09) calls delay()-function. It is a built-in function that delays the progress of the code for the number of milliseconds specified in the parameter, here 1000ms, which is a full second.

Row 08 calls ‘digitalWrite()’ again with the ‘LOW’ parameter. ‘LOW’ is typically a number 0 that tells Arduino to reduce voltage in the given pin down to 0 Volts.

Row 10 ends the loop.

Upload the code to Arduino and test that the LED works properly.

Step 2: Morse algorithm

Our LED works and we have specified the Morse sequence. Now we get the LED blinking accordingly. Let us think through the algorithm before jumping in. What does the algorithm have to do in order to get the blinking to happen according to our sequence?

```
1. Store the Morse sequence in memory
2. Read the first character in the sequence
3. If the character is S, blink the LED briefly
4. If the character is L, blink the LED long
5. If the character is P, keep the LED dark for some time
6. Repeat 2-6 until the sequence if finished
```

Step 3: Storing a Simple Morse Sequence in Memory

In this simple version of the exercise we shall only cover a small subset of the whole Morse Code, challenge and do a code that can blink the LED according to a Morse sequence, i.e. short blinks, long blinks and pauses.

```
S = Short
```

```
L = Long
P = Pause
```

Our Morse code will consist of sequences of these simplified codes. We shall use a variable to store the sequence. The type of the variable is String, which is used for dynamic character sequences.

```
String sequence = "LSSSP";
```

Note: Variable Scope

In this example we are using a *global* variable, which is typically not advised. A global variable is visible to all parts of the program, whereby it may be accidentally changed in an unexpected manner. A variable becomes global, when it is defined outside any function, typically near the top of the code.

It is recommended to use *local* variables whenever possible. A local variable is defined within a function and it will be only visible inside the function.

Step 4: Simple Implementation

```
01.   const int ledpin = LED_BUILTIN;
02.   const int dot_duration = 300;

03.   String sequence = "LSSSP";
04.   unsigned int sequence_pos = 0;

05.   void setup() {
06.      pinMode(ledpin, OUTPUT);
07.   }

08.   void loop() {
09.      if (sequence[sequence_pos] == 'S')
10.      {
11.             digitalWrite(ledpin, HIGH);
12.             delay(dot_duration);
13.             digitalWrite(ledpin, LOW);
14.             delay(dot_duration);
15.      }
16.      else if (sequence[sequence_pos] == 'L')
17.      {
18.             digitalWrite(ledpin, HIGH);
19.             delay(dot_duration*3);
20.             digitalWrite(ledpin, LOW);
21.             delay(dot_duration);
22.      }
23.      else if (sequence[sequence_pos] == 'P')
24.      {
25.             digitalWrite(ledpin, LOW);
26.             delay(dot_duration*3);
27.      }

28.      sequence_pos++;
29.      if (sequence_pos >= sequence.length())
30.      {
31.             sequence_pos = 0;
32.      }
33.   }
```

We added a constant called “dot_duration”. This tells how long a short blink should last, i.e. 300 milliseconds. A longer blink is this value multiplied. Instead of writing 300 milliseconds into the code, it is clearer to use a named constant that is re-used in different locations.

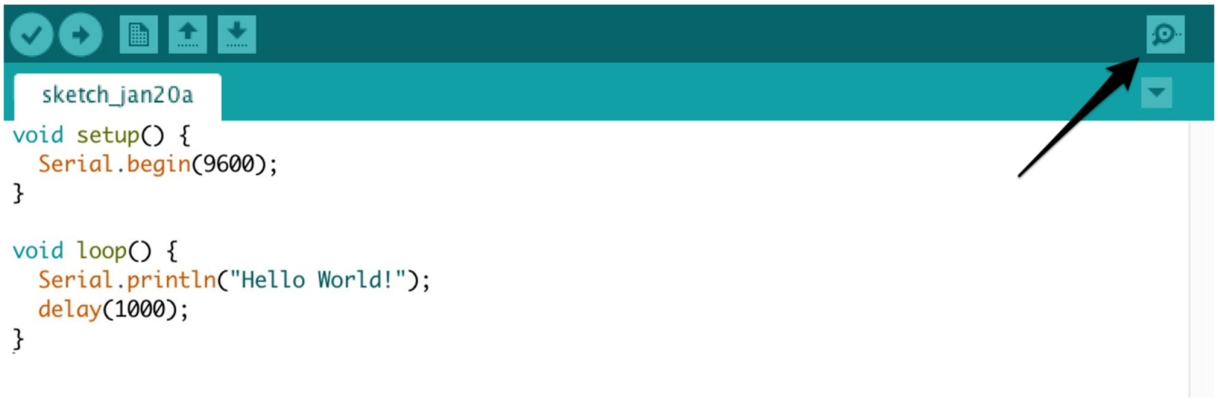
We use the global variable called “sequence_pos” to store the location of where we are in the Morse sequence. A global variable is needed, because we use the value repeated in the loop() function. If it was defined inside loop(), then its value would be lost every time loop() was run.

Tip: Debugging with Serial.println()

You can now try to run the code. If you have written it correctly, it should blink the LED as planned. However, if there are mistakes that were not caught by the compiler, it is very difficult to know where problems are. A typical way of debugging code with Arduino is the use of text printing through the virtual serial port.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Hello World!");
  delay(1000);
}
```

This example prints out the text “Hello World!” once every second to the port. In order to read the messages from Arduino, you can open the Serial Monitor from the Arduino IDE.



It opens a new window that displays messages from the Arduino, if the communication speed is set according to the settings in Arduino. Here it is 9600 bps. Whenever you suspect potential errors in your code, you can send simple texts through

```
Serial.println("message")        // prints a line with a new line feed
Serial.print("message")          // prints only the characters between brackets
```

Step 5: Morse Code with Computer Input

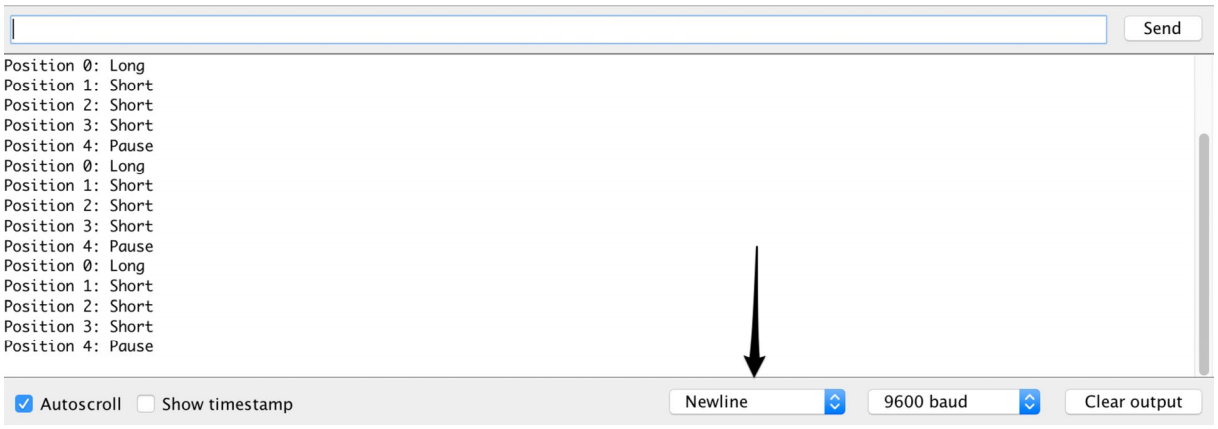
Let us now modify to code so that it accepts input from a computer. For this we will need the Serial.readStringUntil()-function. This is a specific function to accept serial communication until a particular character is met.

```
01.  const int ledpin = LED_BUILTIN;
02.  const int dot_duration = 300;
03.  String sequence = "LSSSP";
04.  unsigned int sequence_pos = 0;
```

```
05.  void setup() {
06.       pinMode(ledpin, OUTPUT);
07.       Serial.begin(9600);
08.  }

09.  void check_update() {
10.       if (Serial.available() > 0) {
11.            sequence = Serial.readStringUntil('\n');
12.            Serial.println("Sequence updated!");
13.       }
14.  }
15.
16.  void loop() {
17.       check_update();
18.       Serial.print("Position ");
19.       Serial.print(sequence_pos);
20.       Serial.print(": ");

21.       if (sequence[sequence_pos] == 'S') {
22.            Serial.println("Short");
23.            digitalWrite(ledpin, HIGH);
24.            delay(dot_duration);
25.            digitalWrite(ledpin, LOW);
26.            delay(dot_duration);
27.       } else if (sequence[sequence_pos] == 'L') {
28.            Serial.println("Long");
29.            digitalWrite(ledpin, HIGH);
30.            delay(dot_duration*3);
31.            digitalWrite(ledpin, LOW);
32.            delay(dot_duration);
33.       } else if (sequence[sequence_pos] == 'P') {
34.            Serial.println("Pause");
35.            digitalWrite(ledpin, LOW);
36.            delay(dot_duration*3);
37.       }

38.       sequence_pos++;
39.       if (sequence_pos >= sequence.length()) {
40.            sequence_pos = 0;
41.       }
48.  }
```

We created a new function called check_update(). We call that every time in the start of the loop()-function. The call looks like “check_update()” on the row 17.

The function checks the serial port for input. The function “Serial.readStringUntil('\n')” reads the string until the magical ‘\n’ character is met. This is the newline, which is added as the last character into to sequence, when you send the message, if the “Newline” is selected.



Tip: Parallel Computing

The current code relies heavily on the use of the delay() function. It is problematic in situations, where the processor should be used for other tasks than just waiting. No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function, so in effect, it brings most other activity to a halt.

Another way to create delays of specified length is to calculate milliseconds using the millis() function.

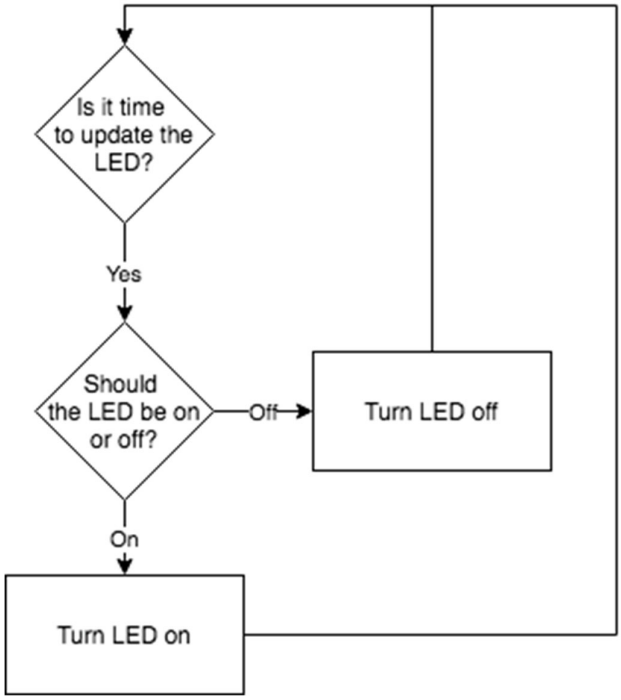
LED blinking does not require us to simply wait for the appropriate moment, but we can do many things while time passes. We are just interested for checking the clock for time. For achieving this, we need to store the time of the event that we are interested in. We can then calculate the time-difference to the current moment by subtracting the earlier milliseconds from the current one. Here is an example:

```
01.  const int blinkInterval = 1000;
02.  const int ledpin =  LED_BUILTIN;
03.  int ledstate = LOW;
04.  unsigned long previoustime = millis();

05.  void setup() {
06.       pinMode(ledpin, OUTPUT);
07.  }

08.  void loop() {
09.       long time_since_blink = millis() - previoustime;
10.       if (time_since_blink >= blinkInterval) {
11.            previoustime = millis();
12.            if (ledstate == LOW) {
13.                 ledstate = HIGH;
14.            } else {
15.                 ledstate = LOW;
16.            }
17.       digitalWrite(ledpin, ledstate);
18.       }
29.  }
```

We store the moment that the LED is manipulated to the variable called “previoustime”. This is compared to the current time, which comes from the millis() function.

Step 6: Morse Code without Delay()-function
(Note that the solution is different from the one in Sähköpaja in Finnish)

Let us remove the Delay()-function from our code altogether. In Morse code the duration of on long blink is three times the duration of a short blink. On the next page you find a flow diagram that shows the algorithm logic visually. (The diagram is made withe the free draw.io).



This idea can be translated into code into the following way:

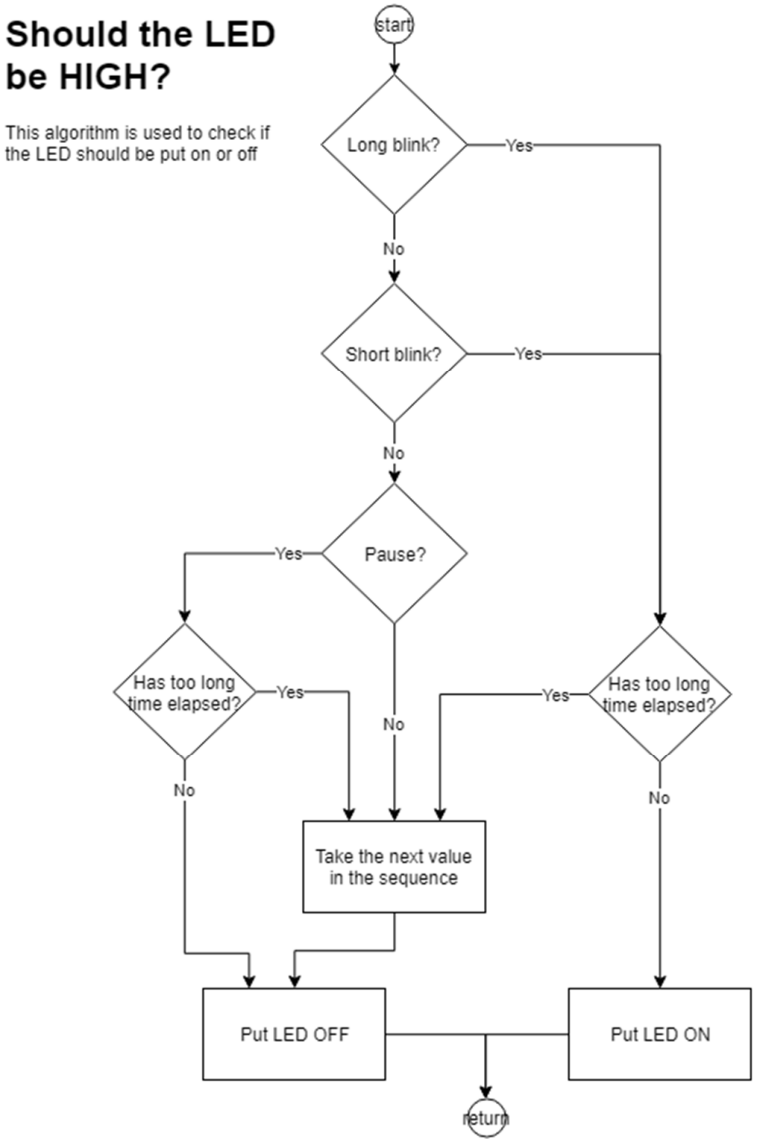
```
void loop() {
  if (timeToUpdateLED()
    updateLED();
}

void updateLED() {
   if (LEDShouldBeHigh())
     digitalWrite(ledpin, HIGH);
   else
     digitalWrite(ledpin, LOW);
}
```

This example assumes that we create three functions:

1. timeToUpdateLED()
2. updateLED()
3. LEDShouldBeHigh()

The first one will need to return a truth value (i.e. true/false) so that the if-test can work. The updateLED()-function puts the LED on or off depending on the return value of the LEDShouldBeHigh()-function.

The most difficult function in the example is the last one, i.e. LEDShouldBeHigh(). Let us first outline the algorithm visually.



The function will ultimately result in putting the LED off after successful operation, regardless of it being short or long blink / pause, once too long time has elapsed. The next character value in the sequence is picked and LED is put off.

It is important to note that the LED then will always stay off until the function "timeToUpdateLED()" gives a true value. This will happen after one dot_duration. Thus, in the final implementation we need to add one more dot durations to our checks of proper durations. Otherwise, the whole will not work properly.

Example code is found on the next page as well as on Codepad (https://codepad.co/snippet/0TA4seQI):

Step 7: Example Code

```
01.  const int ledpin = LED_BUILTIN;
02.  const int dot_duration = 300;
03.  String sequence = "LSSSP";
04.  unsigned int sequence_pos = 0;
05.  unsigned long sequence_start_time = millis();

06.  void setup() {
07.       pinMode(ledpin, OUTPUT);
08.       Serial.begin(9600);
09.  }

10.  void loop() {
11.       if (Serial.available()) { // if new message from computer
12.            sequence = Serial.readStringUntil('\n');
13.            sequence_pos = 0;
14.       }
15.       if (timeToUpdateLED())
16.       updateLED();
17.  }

18.  bool timeToUpdateLED() {
19.       if ((millis() - sequence_start_time ) > dot_duration)
20.            return true;
21.       return false;
22.  }

23.  void updateLED() {
24.       if (LEDShouldBeHigh())
25.            digitalWrite(ledpin, HIGH);
26.       else
27.            digitalWrite(ledpin, LOW);
28.  }

29.  bool LEDShouldBeHigh() {
30.       long time_elapsed = millis() - sequence_start_time;
31.       if (sequence[sequence_pos] == 'L') {
32.            if (time_elapsed < dot_duration*4) // Why not 3? Think!
33.                 return true;
34.            else advance_in_sequence();
35.       } else if (sequence[sequence_pos] == 'S') {
36.            if (time_elapsed < dot_duration*2) // Why not 1?
37.                 return true;
38.            else advance_in_sequence();
39.       } else if (sequence[sequence_pos] == 'P') {
40.            if (time_elapsed < dot_duration*4) // And, again here...?
41.                 return false;
42.            else advance_in_sequence();
43.       }
44.       advance_in_sequence(); // The input was something else than L,S or P
45.       return false;
46.  }

47.  void advance_in_sequence() {
48.       sequence_pos++;
49.       if (sequence_pos >= sequence.length())
50.       sequence_pos = 0;
51.       sequence_start_time = millis();
52.  }
```

Note variable types in the function definitions. On line 18, timeToUpdateLED() returns Boolean, as it is only used to give a simple true of false depending on the time. On line 23, updateLED() does not return any value, as it only puts a LED on or off.