ME 120: Counting Time with Arduino Variables

Gerald Recktenwald
gerry@pdx.edu

October 16, 2019

1 Introduction

Suppose you had a program that needed to keep track of time for weeks or months. You would not be able to directly measure those large time intervals with the built-in `millis()` function in Arduino UNO[1]. Recall that the Arduino `millis()` function returns the current value of the system clock in milliseconds.

The problem is that the number of milliseconds grows quickly. For example, 1 minute is equivalent to 60000 milliseconds, which exceeds the largest value that can be stored in an `int` variable.

$$1 \text{ minute} \times \frac{60 \text{ seconds}}{1 \text{ minute}} \times \frac{1000 \text{ milliseconds}}{1 \text{ second}} = 60000 \text{ milliseconds}$$

Recall that an Arduino `int` can take values between 32768 and +32767, as indicated in Table 1.

2 Range Limits for Integers

Table 1 shows the ranges of values that can be stored in the four types of integer variables used in an Arduino program. The range of each variable type puts limits on the maximum number of milliseconds that can be stored.

Table 2 provides one way of visualizing how the number of milliseconds grows with the size of time intervals you might want to measure. The second through fourth columns of Table 2 show the number of seconds, milliseconds and microseconds[2] for the time intervals in the first column.

For example, suppose you are monitoring the temperature and other environmental variables at a remote site. The experiment runs unattended for weeks. If you were using the built-in `millis` function to track the time for each measurement, and if you wanted to have your program run unattended for 30 days, you would need a variable that could store (or at least compute with) time values up to 2.59×10^9 milliseconds. Checking the ranges for integer variables in Table 1, you conclude that you would need to use an `unsigned int` to hold the values of milliseconds returned by the `millis()` function.

Exercise: Shade the cells in Table 2 to indicate the smallest variable type that can hold each of the numerical values in the seconds, milliseconds and microseconds columns.

3 Real Time Clock

Given the problems of counting milliseconds, the recommended way to keep track of large time intervals is to use a device called a real time clock (RTC). There are several RTC products for Arduino and other microcontrollers. When provided with a battery power source, RTC devices can track large time intervals even when the microcontroller has lost power. The software libraries that accompany RTC devices also have features (usually supported in hardware) for tracking calendar dates and compensating for leap years.

[1] With some careful computation and integer math, you could count the seconds, minutes, hours and days, but to do that you would have to keep track of when the value returned by `millis()` exceeded the storage ability of a single variable

[2] The Arduino `micros()` function returns the value of the system clock in microseconds.

Table 1: Ranges for integer variables

Variable type	Minimum value	Maximum value
`int`	32768	+32767
`unsigned int`	0	65535
`long`	$2,147,483,648$	$2,147,483,648$
`unsigned long`	0	$4,294,967,295$

Table 2: Time values expressed in seconds, milliseconds and microseconds.

Time interval	Seconds	Milliseconds	Microseconds
1 minute	60	60000	6.00×10^7
5 minute	300	300000	3.00×10^8
30 minutes	1800	1.80×10^6	1.80×10^9
1 hour	3600	3.60×10^6	3.60×10^9
5 hours	18000	1.80×10^7	1.80×10^{10}
12 hours	43200	4.32×10^7	4.32×10^{10}
1 day	86400	8.64×10^7	8.64×10^{10}
7 days	604800	6.05×10^8	6.05×10^{11}
30 days	2.59×10^6	2.59×10^9	2.59×10^{12}
90 days	7.78×10^6	7.78×10^9	7.78×10^{12}
6 months	1.56×10^7	1.56×10^{10}	1.56×10^{13}
1 year	3.15×10^7	3.15×10^{10}	3.15×10^{13}
5 years	1.58×10^8	1.58×10^{11}	1.58×10^{14}

4 Solution to the exercise on page 1

Table 3 is a variation on Table 2, using shading to indicate which variable type is suitable for storing the value of the time intervals measured in seconds, milliseconds and microseconds. The Key at the bottom of Table 2 shows the relation between the cell shading and the variable type. For example, the value of 60000 in the first row under the *Milliseconds* column is shaded yellow because 1 minute of milliseconds could be stored in a `unsigned int`, which has an upper limit of 65535.

Note that there are many time intervals not represented in Table 3. The purpose of the exercise is to give a qualitative sense of the limits of using integer values to count time in seconds, milliseconds and microseconds.

Table 3: Time values expressed in seconds, milliseconds and microseconds, with cells shaded according to the smallest integer type that can represent the value of the time interval in the first column.

Time interval	Seconds	Milliseconds	Microseconds
1 minute	60	60000	6.00×10^7
5 minute	300	300000	3.00×10^8
30 minutes	1800	1.80×10^6	1.80×10^9
1 hour	3600	3.60×10^6	3.60×10^9
5 hours	18000	1.80×10^7	1.80×10^{10}
12 hours	43200	4.32×10^7	4.32×10^{10}
1 day	86400	8.64×10^7	8.64×10^{10}
7 days	604800	6.05×10^8	6.05×10^{11}
30 days	2.59×10^6	2.59×10^9	2.59×10^{12}
90 days	7.78×10^6	7.78×10^9	7.78×10^{12}
6 months	1.56×10^7	1.56×10^{10}	1.56×10^{13}
1 year	3.15×10^7	3.15×10^{10}	3.15×10^{13}
5 years	1.58×10^8	1.58×10^{11}	1.58×10^{14}

Key: Cell color	Smallest suitable variable type
	`int`
	`unsigned int`
	`long`
	`unsigned long`
	Too large for `unsigned long`