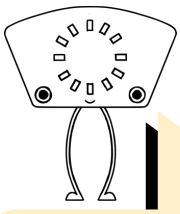
# LED CLOCK PROGRAMMING GUIDE

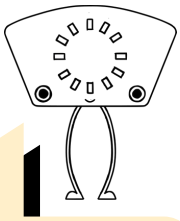


## VARIABLE TYPES

In Arduino, a number variable is usually one of two types: an integer number which has no decimals such as 1, 2, 3, -5, -100 and real numbers which have decimals such as 1.001, 2343.234, -2.2999. The most common integer type is int and the most common real type is float.

An int variable can be assigned with a number in the range of -2,147,483,648 to 2,147,483,647. When int variables are made to exceed their maximum or minimum capacity they overflow. The result of an overflow is unpredictable so this should be avoided. A typical symptom of an overflow is the variable "rolling over" from its maximum capacity to its minimum or vice versa. Check out www.arduino.cc

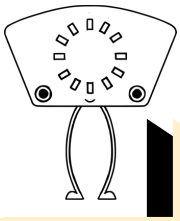
To define a variable type the variable type, a blank space, and the variable name.
int a;
float b;



## MATH OPERATIONS

**ADDITION (+)**
Add two or more numbers. The result should be assigned to a variable or compared to another number or variable.
Examples:
Z = 2 + 3;
Z = X + Y + 4;
Z > 2 + 4;
A = A + 1;
To increment a variable by 1, you can use ++ operator.
A++; is the same as A = A + 1;

# LED CLOCK PROGRAMMING GUIDE



## MATH OPERATIONS

### SUBTRACTION (-)

Subtracts a number from another number. The result should be assigned to a variable or compared to another number or variable.

Examples:

Z = 10 - 2;

Z = A - 10 - C;

Z > 20 - 30;

A = A - 1;

To decrement a variable by 1, you can use -- operator.

A--; is the same as A = A - 1;

### MULTIPLICATION (*)

Multiplies two or more numbers. The result should be assigned to a variable or compared to another number or variable.

Examples:

Z = 10 * 2;

Z = X * Y * -10.5;

Z > 20 - 30;

A = A * 5;

### DIVISION (/)

Divides a number by another number. The result should be assigned to a variable or compared to another number or variable.

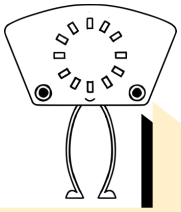
Z = 10 / 2;

Z > 20 / 30;

A = A / 2;

If the result of any math operation is a real number, you should assign it to a real (floating-point) variable to keep the decimals.

# LED CLOCK PROGRAMMING GUIDE



## MATH OPERATIONS

### MODULO (%)

The modulo leaves a remainder which is the amount "left over" after performing a series of subtractions or additions by n on some number x such that the result r is between 0 and n.

```
x % n = r, x = a*n + r
if n > 0, 0 < r < n
if n < 0, 0 > r > n
```

For example 17 modulo 7 is 3. See the calculations: 17 % 7, 17 - 7 = 10, 10 - 7 = 3, 0 < 3 < 7; 17 % 7 = 3
Another Example: -5 % 3, -5 + 3 = -2, -2 + 3 = 1, 0 < 1 < 3; -5 % 3 = 1

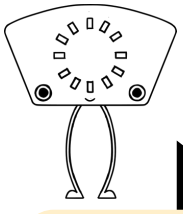
In the Arduino programming language, we can use % operator to find the remainder of a division. So the result of 17 % 7 will be 3. The remainder is always between zero and our number n.

## ORDER OF OPERATIONS

If there are multiple operations in one line or statement, the priority of operations is as below.

| PRIORITY | OPERATOR | DESCRIPTION | ASSOCIATIVITY |
|---|---|---|---|
| 1 | ++ , --<br>() | Suffix/Posfix Increment and Decrement<br>Grouping | Left-to-Right |
| 2 | *<br>/<br>% | Multiplication<br>Division<br>Modulo / Remainder | Left-to-Right |
| 3 | +<br>- | Addition<br>Subtraction | Left-to-Right |

# LED CLOCK PROGRAMMING GUIDE



## EXAMPLE

The result of the following statement is calculated as below.

```
A = 10;
B = 4 + 5 * 9 * (4 - 5) - A++;
```

**STEP 1:**
++ and () are the same priority but the () are left of ++. So the statement inside the () is calculated first.

**STEP 1–1:**
Calculation: 4-5 = -1
New statement after calculation: B = 4 + 5 * 9 * -1 - A++;

**STEP 1–2:**
Calculation: A++ = 10++ = 10 + 1 = 11
New statement after calculation: B = 4 + 5 * 9 * -1 - 11;

**STEP 2:**
* has a higher priority than - and + and would be done first. As there are two *s in the statement, the one on the left will be calculated first

**STEP 2–1:**
Calculation: 5 * 9 = 45
New statement after calculation: B = 4 + 45 * -1 - 11;

**STEP 2–2:**
Calculation:45 * -1 = -45;
New statement after calculation: B = 4 + -45 - 11;

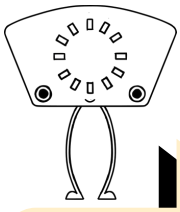
**STEP 3:**
+ and - have the same order; therefore, the + would be calculated first as it is on the left.

**STEP 3–1:**
Calculation: 4 + -45 = -41;
New statement after calculation: B = -41 - 11;

**STEP 3–2:**
Calculation: -41 - 11 = -52;
Final result: B = -52;

| CHALLENGE ONE | CHALLENGE TWO | CHALLENGE THREE |
|---|---|---|
| A = 8;<br>B = 5 * 6 / 3 + 7 - A; | A = 2;<br>B = A + 5 % 2; | A = -7;<br>B = A % 4 + 7 * (2 + 1); |

# LED CLOCK PROGRAMMING GUIDE

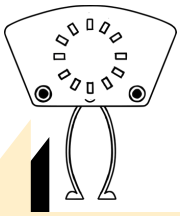


## DOWNLOADING...

Open the Git Hub repository and download the code as a zip file.

https://github.com/ZenMakerLabInc/LED_CLOCK

Extract the zip file and copy the “FastLED-master” folder into your Arduino libraries folder.



## WRITE SOME CODE!

Scroll down to the “updateTime” function:

```
void updateTime() {
   /*Activity Starts Here*/

   /*Activity Ends Here*/
}
```

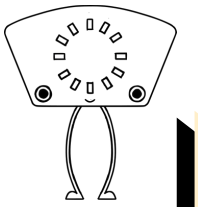
This code provides you with an int variable that increases by 1 every half a second. The name of the variable is:

`half_seconds_irq`

In other words, the value of this variable shows the number of seconds passed from the time that the clock is powered up. The value of this variable should be converted to seconds, minutes, and hours and then assigned to variables names:

`seconds, minutes, hours`

# LED CLOCK PROGRAMMING GUIDE



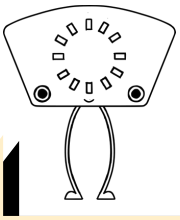
## CALCULATING SECONDS

To calculate the seconds passed so far, we should divide half_seconds_irq variable by 2. If half_seconds_irq is 120, it means that only 60 seconds is passed, as half_seconds_irq increases every half second.

But we know that a clock only shows a number between 0 to 59 as seconds. Therefore, we should show the remainder of the seconds passed so far modulo 60.

The result is:

```
seconds =
```



## CALCULATING MINUTES

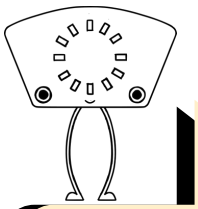
We know that each minute is 60 seconds. To calculate the minutes passed so far, we should divide the half_seconds_irq variable by 120. If half_seconds_irq is 120, it means that only 1 minute has passed, as half_seconds_irq increases every half second.

But we also know that a clock only shows a number between 0 to 59 for the minutes. Therefore, we should show the remainder of the minutes passed so far modulo 60.

The result is:

```
minutes =
```

# LED CLOCK PROGRAMMING GUIDE



## CALCULATING HOURS

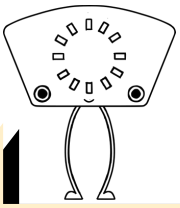
We know that each hour is 60 minutes and each minute is 60 seconds, Therefore, each hour is 3600 seconds which is 60 times 60 (60 x 60 = 3600).

To calculate the hours passed so far, we should divide the half_seconds_irq variable by 7200. If half_seconds_irq is 7200, it means that only 1 hour is passed, as half_seconds_irq increases every half second (3600 * 2 = 7200).

But we know that a clock only shows a number between 0 to 11 as the hour. Therefore, we should show the remainder of the hours passed so far modulo 12.

The result is:

```
hours =
```



## SETTING THE TIME

Every time that an adjustment button is pressed, a variable is incremented. For the hour adjustment, the variable is called hours_offset and for minute adjustment, it is called minutes_offset.

The hour adjustment variable is always between 0 and 11, and the minute adjustment is within the range of 0 to 59. To apply the adjustment, the hours and minutes variable should be defined as the following lines. It is the original time offset by a specific value.

```
minutes =
```

```
hours =
```