

0.1 VARIABLES AND DATA TYPES

In earlier examples we have stored some values into *variables* (e.g. counting **for-next** loop repetition). Variables are the containers for storing data values usually located in RAM (also in EPROM, FLASH ...). In order to store different data (e.g. numbers, words ...) we have to use different type of variables. The declaration of the variable (=creation) has next syntax:

```
1  type variable_name = value;
```

With next example we will solve the problem how to make light blinking while the robot is driving in reverse.

0.1.1 Task: USING VARIABLES

1. Start with this example of driving the robot for 3s forward and then for 3s backward. Test program example in prog. 1. Then try to add some code to blink the light while the robot is driving backward.

Program 1: Variables and Data Types.

```
1  #include "RobotMovingFunctions.h"
2  void setup()
3  {
4      setIOpins();
5
6      moveForward();
7      delay(3000);
8      moveBack();
9      deay(3000);
10     stopTheRobot();
11 }
12 void loop()
13 {
14 }
```

2. As you probably find out you have to divide the duration of 3000 ms into smaller durations and meanwhile controlling the light output. This can be done with **for-next** loop which repeats 10 times.

Change the 9th line `delay(3000)` in previous example into **for-next** loop with 10 repetition, but with the same overall duration of 3000 ms.

```

1  ...
2  moveBack();
3  for (int i = 0; i < 10; i++)
4  {
5      delay(150);
6      delay(150);
7  }
8  stopTheRobot();
9  ...

```

3. Add some code for blinking the LED in the **for-next** loop during the robot is driving backward.

Don't forget to set the REVERSE_LIGHT_PIN value and its `pinMode(...)`.

```

1  ...
2  moveBack();
3  for (int i = 0; i < 10; i++)
4  {
5      digitalWrite(REVERSE_LIGHT_PIN, HIGH);
6      delay(150);
7      digitalWrite(REVERSE_LIGHT_PIN, LOW);
8      delay(150);
9  }
10 stopTheRobot();
11 ...

```

4. More advanced way to do a time conditioned loop is shown in next example:

```

1  ...
2  robotBack();
3  unsigned long start_time = millis();
4  int time_diff = 0;
5  while (time_diff < 3000)
6  {
7      digitalWrite(REVERSE_LIGHT_PIN, HIGH);
8      delay(150);
9      digitalWrite(REVERSE_LIGHT_PIN, LOW);
10     delay(150);
11     unsigned long now = millis();
12     time_diff = now - start_time;
13 }
14 stopTheRobot();

```

0.1.2 Questions:

1. Show some examples of programming assignment statement!
2. What is the operator for assign the value to the variable?

0.1.3 Summary:

0.1.3.1 What is variable? A variable in a program is a specific piece of memory that consists of one or more contiguous bytes, typically 1, 2, 4, 8, or 16 bytes. Every variable in a program has a name, which will correspond to the memory address for the variable. You use the variable name to store a data value in memory or retrieve the data that the memory contains.

Variables are used in C++ where you will need to store any type of values within a program and whose value can be changed during the program execution. These variables can be declared in various ways each having different memory requirements and storing capability. Variables are the name of memory locations that are allocated by compilers, and the allocation is done based on the data type used for declaring the variable.

0.1.3.2 Variable definition and initialization in C++ A variable definition means that the programmer writes some instructions to tell the compiler to create the storage in a memory location. The syntax for defining variables is:

```
1 data_type variable_name;
```

Here `data_type` means the valid C++ data type which includes `int`, `float`, `double`, `char`, `wchar_t`, `bool` and variable list is the lists of variable names to be declared which is separated by commas. Variables are declared in the above example, but none of them has been assigned any value. Variables can be initialized, and the initial value can be assigned along with their declaration.

```
1 data_type variable_name = value;
```

Examples:

```
1 int value = 1234;           // whole numbers from -32768 .. 32767
2 char smalVal = 123;        // whole numbers from 0 .. 255
3 char letterA = 'A';        // character value like !"#$%&'()*+,-./:;<=>?@A-Z[a-z]_
4 bool logicVal = true;      // 0 and 1 or false and true
5 float pi_value = 3.14;     // from -3.4E+38 .. +3.4E+38
6 char text[32] = "Some text.;"
```

In next fig. 1 we can find previous variables stored in controllers' RAM memory (upper window of fig. 1). In the lower left corner of the fig. 1 we can find printed memory addresses of these variables. In the memory table we can first notice `text` variable from the address `0x0100` within next 32 bytes (2 rows of the memory table). Next 4 bytes are occupied by `pi_value` variable, at the memory address `0x0124` `logicVal` is stored (1 byte), following with character letter A stored in variable named `letterA` at the address `0x0125` with the HEX value of `0x41`. At the memory address `0x0126` we can find `smalVal`

variable which storing the value 123 (DEC) or 0x7B in HEX. The last 2 bytes are occupied by the integer variable named `value` where the number 1234 is stored or in HEX 0x04 0xD2.

Address	Value
0x00	0x00
0x01	0x00
0x02	0x00
0x03	0x00
0x04	0x00
0x05	0x00
0x06	0x00
0x07	0x00
0x08	0x00
0x09	0x00
0x0A	0x00
0x0B	0x00
0x0C	0x00
0x0D	0x00
0x0E	0x00
0x0F	0x00
0x10	0x00
0x11	0x00
0x12	0x00
0x13	0x00
0x14	0x00
0x15	0x00
0x16	0x00
0x17	0x00
0x18	0x00
0x19	0x00
0x1A	0x00
0x1B	0x00
0x1C	0x00
0x1D	0x00
0x1E	0x00
0x1F	0x00
0x20	0xC3
0x21	0xF5
0x22	0x48
0x23	0x40
0x24	0x01
0x25	0x41
0x26	0x7B
0x27	0xD2
0x28	0x04
0x29	0x00
0x2A	0x00
0x2B	0x00
0x2C	0x00
0x2D	0x00
0x2E	0x00
0x2F	0x00
0x30	0x00
0x31	0xB1
0x32	0x00
0x33	0x71
0x34	0x01
0x35	0xE2
0x36	0x00
0x37	0xC0
0x38	0x00
0x39	0xD4
0x3A	0x00
0x3B	0x00
0x3C	0x0A
0x3D	0x00
0x3E	0x76
0x3F	0x61
0x40	0x72
0x41	0x2E
0x42	0x6E
0x43	0x61
0x44	0x6D
0x45	0x65
0x46	0x09
0x47	0x6D
0x48	0x65
0x49	0x00
0x4A	0x20
0x4B	0x20
0x4C	0x76
0x4D	0x61
0x4E	0x6C
0x4F	0x75
0x50	0x73
0x51	0x09
0x52	0x30
0x53	0x78
0x54	0x30
0x55	0x00
0x56	0x20
0x57	0x73
0x58	0x6D
0x59	0x61
0x5A	0x56
0x5B	0x61
0x5C	0x6C
0x5D	0x09
0x5E	0x30
0x5F	0x00
0x60	0x00
0x61	0x00
0x62	0x00
0x63	0x00
0x64	0x00
0x65	0x00
0x66	0x00
0x67	0x00
0x68	0x00
0x69	0x00
0x6A	0x00
0x6B	0x00
0x6C	0x00
0x6D	0x00
0x6E	0x00
0x6F	0x00
0x70	0x00
0x71	0x00
0x72	0x00
0x73	0x00
0x74	0x00
0x75	0x00
0x76	0x00
0x77	0x00
0x78	0x00
0x79	0x00
0x7A	0x00
0x7B	0x00
0x7C	0x00
0x7D	0x00
0x7E	0x00
0x7F	0x00

Figure 1: Table of values stored in RAM memory of Arduino UNO controller.

0.1.3.3 Measuring Time with programming loops The easiest way to measure time is to simply count the number of loop's executions. And if we know how long is one execution of the loop - we can easily determine the time lapsed for the whole process.

Example:

```

1  int t = 0;
2  while (t<10){
3    t++;
4    delay(100);
5  }

```

In the previous example the **while** loop is executed 10 times ($t = [0 .. 9]$), since each execution of the loop last 100 ms (determined by `delay(100);`) the whole **while** loop last 1 s.

0.1.3.4 Time measuring with Timers More proper way of measuring the time is by using the timer's values. More on that can be read [here](#).

Example:

```
1  unsigned long start_time;  
2  unsigned long stop_time;  
3  start_time = millis();  
4  // time measured process goes here  
5  // ...  
6  stop_time = millis();  
7  unsigned long duration = stop_time - start_time;
```

Where the `duration` is time measured in milliseconds.

0.1.4 Issues:

0.1.4.1 <++> <++>