5.5 Variables and data types

In earlier examples we have stored some values into `variables` (e.g counting **`for`** `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:

```
type varialble_name = value;
```

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

5.5.1 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.

```
#include "RobotMovingFunctions.h"
void setup()
{
  setIOpins();

  moveForward();
  delay(3000);
  moveBack();
  deay(3000);
  stopTheRobot();
}
void loop()
{
}
```

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

```
...
moveBack();
for (int i = 0; i < 10; i++)
{
  delay(150);
  delay(150);
}
stopTheRobot();
...
```

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

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

```
...
moveBack();
for (int i = 0; i < 10; i++)
{
  digitalWrite(REVERSE_LIGHT_PIN, HIGH);
  delay(150);
  digitalWrite(REVERSE_LIGHT_PIN, LOW);
  delay(150);
}
stopTheRobot();
...
```

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

```
...
robotBack();
unsigned long start_time = millis();
int time_diff = 0;
while (time_diff < 3000)
{
  digitalWrite(REVERSE_LIGHT_PIN,HIGH);
  delay(150);
  digitalWrite(REVERSE_LIGHT_PIN,LOW);
  delay(150);
  unsigned long now = millis();
  time_diff = now - start_time;
}
stopTheRobot();
```

5.5.2 Questions:

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

5.5.3 Summary:

5.5.3.1 What is variable? In computer programming, a variable is a storage location in memory that is used to hold a value. The value of a variable can be changed during the execution of a program.

Each variable has a name, which is used to refer to the variable in the code, and a data type, which determines the kind of value that the variable can hold.

There are several different data types in C++, including:

Integers: Integers are whole numbers that can be positive, negative, or zero. In C++, there are several different integer data types, including char, short, int, and long.

Floating-point numbers: Floating-point numbers are numbers with decimal points. In C++, the float and double data types are used to represent floating-point numbers.

Characters: Characters are single letters, digits, or symbols. In C++, the char data type is used to represent characters.

Booleans: Booleans are values that can either be true or false. In C++, the bool data type is used to represent booleans.

To use variables in C++, you will need to declare them and assign them values. Here is an example:

```
int x;        // Declare an integer variable called x
x = 10;       // Assign the value 10 to x

char c;       // Declare a character variable called c
c = 'A';      // Assign the value 'A' to c

double d;     // Declare a double variable called d
d = 3.14;     // Assign the value 3.14 to d
```

5.5.3.2 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:

```
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.

```
data_type variable_name = value;
```

Examples:

```
    int value = 1234;         // whole numbers from -32768 .. 32767
    char smalVal = 123;       // whole numbers from 0 .. 255
    char letterA = 'A';       // character value like !"#0123..ABC..xyz
    bool logicVal = true;     // 0 and 1 or false and true
    float pi_value = 3.14;    // from -3.4E+38 .. +3.4E+38
    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 lover 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 nuber 1234 is stored or in HEX 0x04 0xD2.

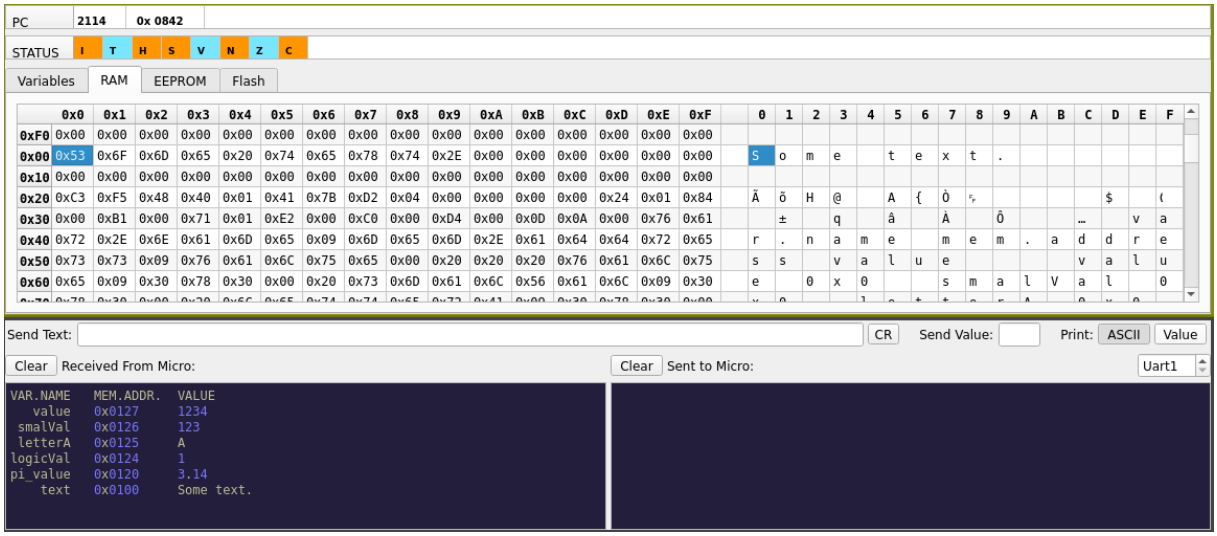

Figure 1: Table of values stroed in RAM memory of Arduino UNO controller.

5.5.3.3 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:

```
int t = 0;
while (t<10){
t++;
delay(100);
}
```

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.

5.5.3.4 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:

```
unsigned long start_time;
unsigned long stop_time;
start_time = millis();
// time measured process goes here
// ...
stop_time = millis();
unsigned long duration = stop_time - start_time;
```

Where the `duration` is time measured in milliseconds.

5.5.3.5 Structures n C++, a struct is a user-defined data type that groups together a collection of variables. It is similar to a class in that it can contain variables and functions, but there are a few key differences between the two.

One of the main differences between a struct and a class in C++ is that structs have public members by default, while classes have private members by default. This means that, by default, all members of a struct can be accessed directly from outside the struct, while members of a class can only be accessed through its member functions.

Another difference is that structs are often used for small, simple data structures that do not require the encapsulation and data hiding features provided by classes. Structs are commonly used for situations where you simply want to group together related data, such as representing a point in two-dimensional space, a date, or a color.

Here is an example of a simple struct in C++:

```
struct Point {
    int x;
    int y;
};
```

This struct defines a new type called Point, which contains two variables of type int, x and y, representing the coordinates of a point in a two-dimensional space.

```
Point p1;
p1.x = 3;
p1.y = 4;
```

In this example, we create a variable p1 of type Point and assign values to its members x and y.

It's also worth noting that C++ has also a keyword class which is semantically equivalent to struct except for the default access level of its members.

5.5.3.6 Enumeration In C++, an **enum** (short for "enumeration") is a user-defined data type that consists of a set of named values. It is used to create a new type with a fixed set of possible values, which can make your code more readable and maintainable.

Here's an example of an enumeration that could be used in a mobile robot program to represent the different states of the robot:

```
enum class RobotMoves{
    FORWARD,
    BACKWARD,
    MOVE_LEFT,
    MOVE_RIGHT,
    STOP
};
```

You can use this enumeration in the robot's control loop to check and update the current state of the robot:

```
RobotMoves currentRobotState = RobotMoves::STOP;

while (true) {
    // Some other logic here
    // ...

    // Sampling the sensors based on the state of the robot
    switch (currentRobotState){
      case RobotMoves::FORWARD    : checkFrontSensors(); break;
      case RobotMoves::BACKWARD   : checkBackSensors(); break;
      case RobotMoves::MOVE_LEFT  : checkLeftSensors(); break;
      case RobotMoves::MOVE_RIGHT : checkRightSensors(); break;
      default: //nothing to do...
    }
}
```

This way, it's clear and easy to understand the current state of the robot, and it can also help to

implement logic and different behaviors for each state. It’s also easy to add or remove states in the future if needed, without having to modify the code in many different places.

5.5.4 Issues:

5.5.4.1 <++> <++>