

## LESSON 3

### Introduction to C Programming

#### The Basic Arduino Program

This lesson assumes that the Arduino IDE is correctly installed on your computer and tested. If not, refer to [Appendix A](#) before beginning this lesson.

1. Let's start with a simple program, as shown in figure 3.1. This is the same program referenced in [Appendix A](#).

```
void setup()
{
    // Your setup code here, to run once:
    Serial.begin(9600);
}

void loop()
{
    // Your main code here, to run repeatedly:
    Serial.println("Hello World");
    delay(1000);
}
```

*Figure 3.1: Program - First.pde*

2. The program consists of two separate functions, namely `setup` and `loop`. You will be adding additional functions to your code, but you must have a `setup` function and a `loop` function in every program that you write. Referring to the program, you should note:
  - The `setup` function is used to define certain parameters on the Arduino microprocessor. In this example, the `setup` function includes the statement `Serial.begin(9600);` which sets the communication speed of the serial port on the Arduino to 9600 baud. This will allow the Arduino to communicate with the serial monitor on the PC. The `()` after the name `setup` is the parameter list passed to the function. Since there are no parameters being passed to the function, the list is empty. However, the `()` after the name is still required. Following the function name is a `{` at the opening, and a matching `}` at the end of the function. Everything between the first `{` and its matching `}` belong to the function.
  - The `//` indicates that whatever follows is a Comment. Comments may begin in any column. For example, one could write:

```
Serial.println("Hello World");           // used to print to a serial port
```

- Use comments to help document the program.
- The word `void` in front of the functions names indicates that the functions are not returning a value. More on this later.

- In the case of the function `Serial.begin(9600)`, the name of the function is `Serial.begin` and when called, the program passes the value of 9600, which in this case is the baud rate (transfer rate) for the serial communication. For another example, when the function `Serial.println("Hello World");` is called, the string `Hello World` is a parameter being passed to the communication port.
- The semicolon `;` terminates the command. It is required. Note that the function name `void loop()` is not terminated by a `;`. That is because the function name contains the lines between the `{` and `}` and therefore cannot be terminated.
- The function `delay(1000);` calls a delay function where the parameter list contains the number of millisecond to wait. In this case, there is a one (1) second delay. Note that when the program executes this command, it is literally waiting the specified time. If something is happening, such as a sensor value has changed, the Arduino microprocessor will not see it during the wait time.
- The `void loop()` is the main function and it must be in every Arduino code that you write. It is the entry point for your code. The loop function is executed continuously, that is, when the program reaches the end of the loop function, denoted by the last `}`, the Arduino goes back to the top the of the loop function and begins execution again.

### Exercise 1

1. After you get the "Hello World" program to run, modify it to:
  1. Print your first name
  2. Wait ½ second
  3. Print surname on the next line
  4. Print a blank line
  5. Wait 2 second
  6. Repeat
2. If you have any difficulty with this exercise, ask your instructor for help.

## Constants and Variables

1. Constants can be thought of as predefined variables that cannot change during the execution of a program. There are two types of constants, the user defined constant and the predefined constants. For example, load the program shown in figure 3.2.
  2. Don't be overwhelmed by the length of the program. It is mostly print statements. We will see shortly how we can reduce the number of lines to do the same thing.
  3. Let's investigate the program:
    1. You again have the two required functions: `setup()` and `loop()`.
    2. The first line of the program we have added `#define var1 1`. The `var1` is now a user defined constant that cannot be changed. This also has a global definition, that is, `var1` is equal to 1 in all for the functions. If you try to redefine `var1`, you will get a compile error. Another method to define the `var1` as a constant global variable will be place the statement `int const var1` before the first function call.
    3. `var2` is also a constant defined to be 1. However, it is local to the function `loop()`. Therefore, it is not available in any functions. Note that if the variable name `var2` is used in another function, it will be a unique variable that is local to that function.
    4. `var3` is a global integer variable because it is typed before the first function call.
    5. `var4` and `var5` are integer variables and can be changed anywhere within the function. They are local variables and their scope is limited to the function that they are defined (typed) in. Run the programs and watch how the variable change as the programs are executed. The purpose of the program is simply to print the values of each variable, then increment the variables using the `++` command and print the values again. Repeat the loop every 1 second. Make sure that you open the serial monitor after the program is completely downloaded. As you can see, the constants `var1` and `var2` do not change. For fun, try to change their values in the code and see what happens.
    6. The value of `var3` increments continuously, as it is a global variable and is not reset upon successive loops. The values of `var4` and `var5` are local variables and get re-initialized at the beginning of each loop. The only difference between `var4` and `var5` is that `var5` is initialized when it is typed. `var4` is not, so it needs to be assigned a value prior to using it.
- Note: Non-constant variables are referred to as variables, and constant variables are referred to as constants.
7. `var6` is a Boolean variable, which is often used with the predefined constants: "true", defined as 1 and "false", defined as 0. Be sure to look at the code and its output to help understand the use of the Boolean variables.

```

#define var1 1
int const var3=1;
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    const int var2 = 1;
    int var4;
    int var5=1;
    boolean var6=false;

    var4=1; // The value of c4 is now defined
    Serial.print("var1 = ");
    Serial.print(var1, DEC);
    Serial.print("  var2 = ");
    Serial.print(var2, DEC);
    Serial.print("  var3 = ");
    Serial.print(var3, DEC);
    Serial.print("  var4 = ");
    Serial.print(var4, DEC);
    Serial.print("  var5 = ");
    Serial.print(var5, DEC);
    Serial.print("  var6 = ");
    Serial.println(var6, DEC);

    //var3++;
    var4++;
    var5++;
    var6=true;

    Serial.print("var1 = ");
    Serial.print(var1, DEC);
    Serial.print("  var2 = ");
    Serial.print(var2, DEC);
    Serial.print("  var3 = ");
    Serial.print(var3, DEC);
    Serial.print("  var4 = ");
    Serial.print(var4, DEC);
    Serial.print("  var5 = ");
    Serial.print(var5, DEC);
    Serial.print("  var6 = ");
    Serial.println(var6, DEC);
    Serial.println(" ");
    delay(1000);
}

```

*Figure 3.2: Program - Second.pde*

4. We will discuss integers, Booleans, and other variables in more detail later. For now, let's take a look at some control statements in the next section.

## Control Statements

1. The control statement is used to control the flow of the program. In this section we will be addressing three of the commonly used control statements: The `if` statement, the `for` statement, and the `do while` statement.

2. The form of the `if` statement:

```
if(logical test)
{
    statements are executed if logical true.

}
else if(logical)
{
    statements are executed if the first logical is false and the second logical
    is true.

}

else
{
    statements are executed if none of the above logical are true. This is the
    default condition.
}
```

3. Note that both the `else if` and `else` statements are optional.

4. Let's do the example shown in figure 3.3.

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

void loop()
{
    int var1, var2;
    var1 = 1;
    var2 = 1;

    if( var1 == 1)
    {
        var2 = 2;
    }

    Serial.print("var1 = ");
    Serial.print(var1, DEC);
    Serial.print("  var2 = ");
    Serial.println(var2, DEC);
    Serial.println(" ");
    delay(1000);
}
```

*Figure 3.3: Program - Third.pde*

5. Things to note about the `if` statement in the example shown in figure 3.3. The logical test is `==` not just `=`. A single `=` is an assignment statement. In the statement `var3=1`, value of 1 is assigned to the variable `var3`. In the statement `var1 == 1`, the value of `var1` is compared to the value of 1 and returns a “true” if the values are equal and “false” if the values are not equal.
6. Take a few minutes to change the value of `var1` in the example to see what happens to the value of `var2`. Does the output make sense?
7. The logical operators are:
  - `==` test if equal
  - `!=` test if not equal
  - `<` test if less than
  - `>` test if greater than
  - `<=` test if less than or equal to
  - `>=` test if greater than or equal to

7. Time for another example. Copy the program Fourth.pde shown in figure 3.4.

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

void loop()
{
  int var1;
  var1 = 3;

  Serial.print("var1 = ");
  Serial.print(var1, DEC);
  if( var1 == 1)
  {
    Serial.println(" is equal to 1");
  }
  else if( var1 < 1)
  {
    Serial.println(" is less than 1");
  }
  else if( var1 <= 4)
  {
    Serial.println(" is less than or equal to 4");
  }
  else
  {
    Serial.println(" unknown");
  }
  delay(1000);
}
```

*Figure 3.4: Program - Fourth.pde*

8. Look at the use of the `if`, `else if` and `else` statements. Does it make sense?

## Exercise 2

- Change the value of `var1 = 0`. What part of the `if` statement prints the results?
- Change the value of `var1 = 1`. Why does the program say it is less than 1, but does not say it is less than 4?
- Change the value of `var1 = 3`. Does the program correctly state the value of `var1` is less than 4?
- Change the value of `var1 = 7`. This is the default value.
- Try several numbers to make sure that you understand the program flow.
- Try the `!=` and `>=` logical operators.

1. Next we will look at the `for` statement, which is used for repetitive operations.

2. The form of the `for` statement is given by:

```
for(initialize the variable, exit condition, increment the variable)
{
    statements to be executed in the loop
}
```

3. Lets jump into the example shown in figure 3.5.

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

void loop()
{
    int var1;
    for(int count=0; count<10; count++)
    {
        var1 = var1 + count;
    }
    delay(1000);
}
```

*Figure 3.5: Program - Fifth.pde*



4. In the program Fifth.pde, the `for` statement is used to loop a set of statements ten times. You should notice several items:
- The variable `var1` is typed and initialized in the `int var1=0` statement.
  - The variable `count` is typed and initialized to 0 in the `for` statement, hence, its scope is limited to the `for` loop.
  - At the start of each of the `for` loops, `count` is tested to see if it is less than 10.
  - At the completion of each loop, the variable `count` is incremented by 1.
  - The last statement `while(1==1)` is another control statement which in this case is used to hold the program execution at that point. Since 1 always equals 1, the program never leaves the `while` statement so the function `loop()` is only executed once.
  - Experiment by changing the initial value of `count` and the logical test.

**CAUTION** – Make sure that you SAVE on a regular basis when you are testing your code. It is easy to get the program into an infinite loop and sometimes you just have to kill (stop) the program.

5. For our last example, before we start controlling the hardware, let's combine the use of the `if` statement and the `for` statement. The program shown in figure 3.6 counts from 1 to 10 and determines if the number is less than 5, equal to 5, or greater than 5.

```

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

void loop()
{
    int var1=0;
    for(int count=1; count<=10; count++)
    {
        if(count < 5)
        {
            Serial.print(count, DEC);
            Serial.println(" is less than 5 ");
        }
        else if(count > 5)
        {
            Serial.print(count, DEC);
            Serial.println(" is greater than 5 ");
        }
        else
        {
            Serial.print(count, DEC);
            Serial.println(" is equal to 5 ");
        }
    }
    while(1==1);
}

```

*Figure 3.6: Program - Sixth.pde*

### Exercise 3

1. Write a program based on Sixth.pde to computer and print the number from 0 to 10 together with the factorial of the number. The output of the program should look like this:

```

The factorial of 0 is 1
The factorial of 1 is 1
The factorial of 2 is 2
The factorial of 3 is 6

```

etc...

### Advanced Exercise

1. Modify the program to count from -5 to 5. For the negative numbers, the program should state the factorial could be computed.

In lesson 3, we will start interfacing to the hardware.