Workshop 2

Yuxuan Han, Fleming Society

November 16, 2023

Contents

1 How to Code in C++ 1

1.1 Basic C++ Syntax in "Blink" Example . . . 1
1.2 Variables . . . 3
1.3 Operators . . . 3
1.4 Conditions and If Statements . . . 5
1.5 Further about C++ . . . 5

2 Important Arduino Built-in Functions for Workshop 2 6

2.1 pinMode() . . . 6
2.2 digitalWrite() . . . 6
2.3 analogWrite() . . . 7
2.4 digitalRead() . . . 7
2.5 analogRead() . . . 8
2.6 delay() . . . 8
2.7 delayMicroseconds() . . . 8
2.8 millis() . . . 9
2.9 micros() . . . 9
2.10 Serial Communication . . . 9

3 References 10

1 How to Code in C++

C++ is one of the world's most popular programming languages. You will use C++ to program your Arduino UNO boards in the following workshops. This document will teach you something basic about C++ to help you code the Arduino easier in the following workshop.

1.1 Basic C++ Syntax in "Blink" Example

To introduce the basic syntax of C++, let's start with the example program "Blink" [1], which makes the built in LED on your Arduino board to blink.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

```

[1]Source: https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink

```
 7  // the loop function runs over and over again forever
 8  void loop() {
 9    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
10    delay(1000);                       // wait for a second
11    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
12    delay(1000);                       // wait for a second
13  }
```

Example Explained:

Line 1: This line is a comment, which is used to explain the code. Any text between `//` and the end of the line is ignored by the arduino board and will not be executed.

Line 2: `void setup()` is called a **function**. This is an important function in Arduino Programming. Any code inside its curly brackets `{}` will be executed *ONLY ONCE* after each powerup or reset of the Arduino board.

Line 3: Comment, will not be executed.

Line 4: `pinMode(LED_BUILTIN, OUTPUT)` calls the Arduino built-in function `pinMode()` with two *parameters*: `LED_BUILTIN` and `OUTPUT`. It initialised the digital pin LED_BUILTIN as an output.

Line 5: Remember to add the closing curly bracket `}` to end the function

Line 6: A blank line. C++ will ignore the blank line. However, using suitable blank lines can make your code more readable.

Note: C++ also ignores blank spaces. However, suitable blank spaces, for example, indenting, will also increase the readability of the code.

Line 7: Comment.

Line 8: Definition of another function called `void loop()`. This the other important function in Arduino programming. Any code inside its curly brackets will *LOOP CONSECUTIVELY* after finish executing `void setup()`.

Line 9: `digitalWrite(LED_BUILTIN, HIGH)` calls another Arduino built-in function called `digitalWrite()`. This sets the output of the of the digital pin LED_BUILTIN to be HIGH, and thus turns on the built-in LED.

Line 10: `delay(1000)` calls another built-in function called `delay()`. This function pauses the whole program for certain milliseconds. Here, it pauses the code for 1000ms, which is one second.

Line 11: Similar to line 9. However, here, the output of LED_BUILTIN is turned to LOW. This turns off the built-in LED.

Line 12: Same as line 10.

Line 13: Curly bracket `}` used to end the function.

Important: Every C++ statement ends with a semicolon `;`. But there is no need to add a semicolon after a curly bracket `}` used to end the function definition or if statements, etc.

Now, hope that you completely understand how to write a simple C++ program to blink the built-in led on the Arduino board. You can try to build a driving circuit for an external LED and modify the code to blink it in your own way.

1.2 Variables

Variables are used to store data values. In C++, there are many different **types** of variables. The most important five will be introduced here.

- `int` - stores integers (whole numbers), without decimals, like 114 or -514.
- `double` - stores floating point numbers, with decimals, like 19.19 or -8.1.
- `char` - stores single characters, such as ’a’ or ’b’. Char values must be surrounded by single quotes.
- `string` - stores text, such as ”Deep Dark”. String values are surrounded by double quotes.
- `bool` - stores values with two states: true (1) or false (0).

For other data types supported by arduino, see Arduino’s Language Reference.

You need to (create) a variable and specify its type first before using it. This is very different from some other programming languages, for example, *Python*. The following code example shows you how to create a int variable called *myNum*:

```
    int myNum;
```

You can also assign a value to the variable when you create it using the assignment operator `=`:

```
    int myNum = 1145;
```

As a good practice, if you want the variable to be Read Only and do not want other one to change it, you can add a `const` in the front when declearing:

```
    const int myNum = 1145;
```

Another one very IMPORTANT thing that you need to know is at which position in the code to declare the variables.

If you declare the variables INSIDE a function, e.g. in `setup()` or `loop()`, the variable will become **local variables**, which can only be accessed INSIDE that function and will be initialised every time when the function is being called.

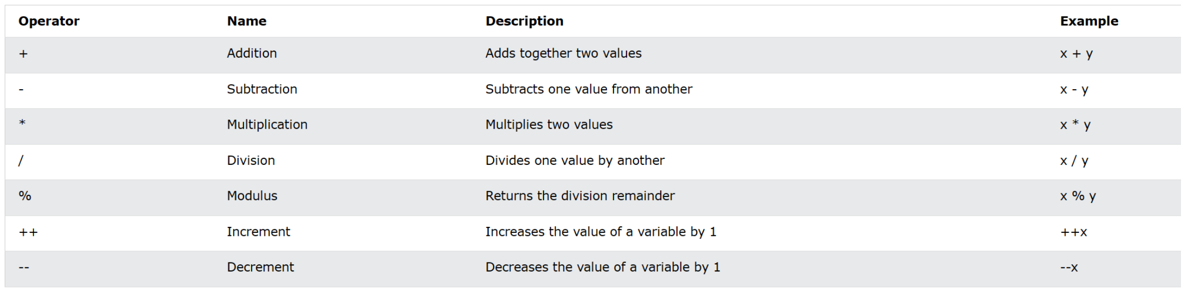
If this is not the case that you want, you can try to declare the variable outside the functions. For example, outside `setup()` and `loop()`. This will make these variables **Global Variables**, which can be accessed from anyplace in the program.

1.3 Operators

C++ contains 4 big types of operators: *Arithmetic, Assignment, Comparison* and *Logical operators.*

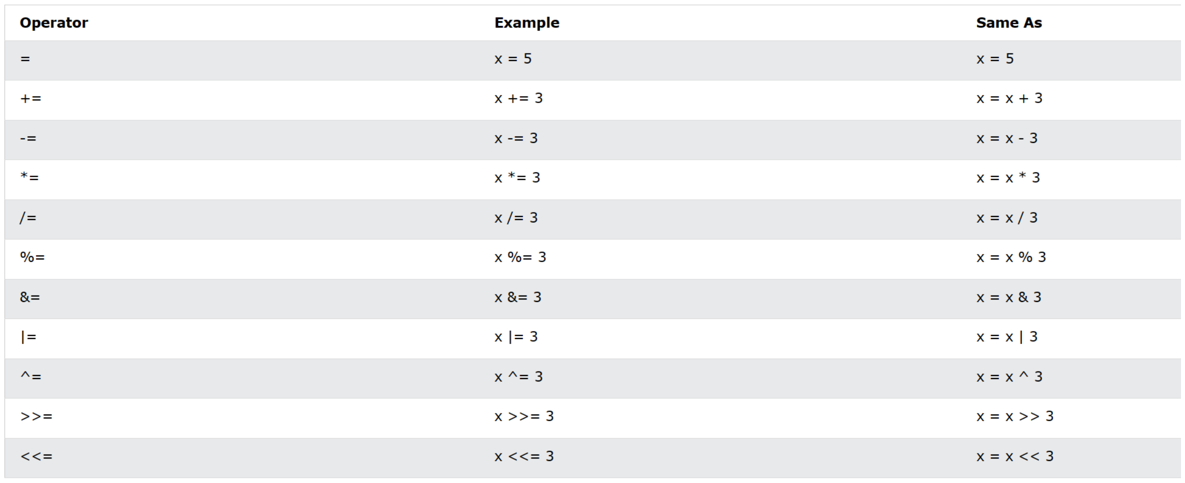
Arithmetic Operators:[2]

[2]Source: https://www.w3schools.com/cpp/cpp_operators.asp



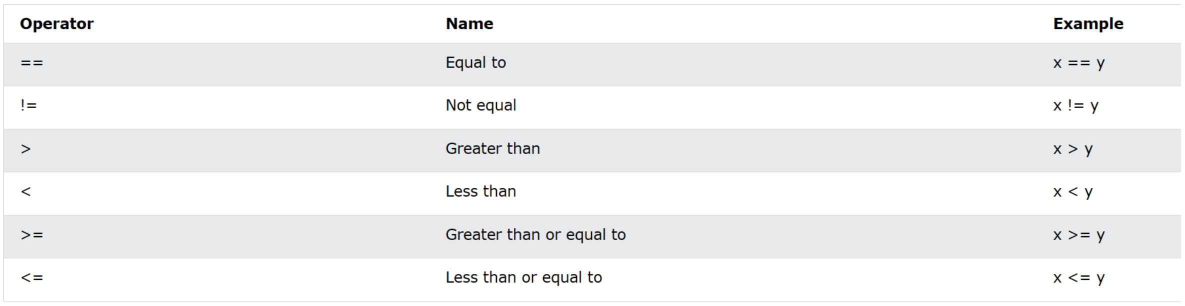
Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators:[3] In most cases, using `=` is enough. However, other assignment operators will make the expression shorter. *If you are lazy :)*



Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators:[4] Comparison Operators compare values and return **true** (1) of **false** (0)

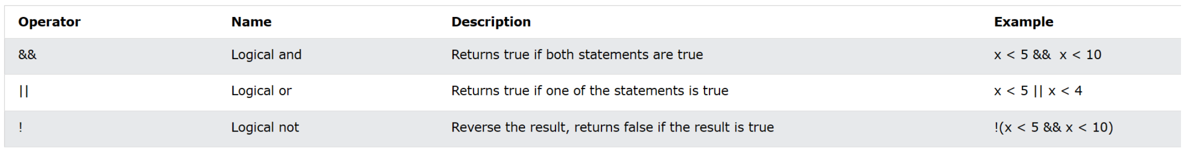


Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Note: Do not confuse the use of `=` *and* `==`

Logical Operators:[5] Sometimes, you may want all, some, or non of the conditions is satisfied. Then you need to use logical operators.

[3]Source: https://www.w3schools.com/cpp/cpp_operators_assignment.asp
[4]Source: https://www.w3schools.com/cpp/cpp_operators_comparison.asp
[5]Source: https://www.w3schools.com/cpp/cpp_operators_logical.asp



Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

1.4 Conditions and If Statements

C++ has four different conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true.
- Use `else` to specify a block of code to be executed, if the same condition is false.
- Use `else if` to specify a to specify a new condition to test, if the first condition is false.
- When there are many alternative block of code to be executed, `switch` can be a good alternative since it can significantly simplify the code.

Syntax of a conditional statement including `if`, `else` and `else if` can be:

```cpp
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

`switch` is a little bit different. The syntax of a switch statement can be:

```cpp
switch(expression) {
  case x:
    // block of code to be executed if the expression value is equal to x
    break;
  case y:
    // block of code to be executed if the expression value is equal to y
    break;
  default:
    // block of code to be executed if no case is matched
}
```

The *expression* is first executed ONCE. The value of the expression is then compared with the value of each case. When they match, the corresponding code block will be executed. The `default:` is optional.

1.5 Further about C++

C++ knowledge in previous subsections should be enough for you to handle most of the problems. If you want to learn further or have queries about some parts, you can check W3Schools C++ Tutorial, which have more detailed explanations with plenty example codes, or feel free to contact one of the Fleming Society members before or during the workshops.

2 Important Arduino Built-in Functions for Workshop 2

This section will explain some important built-in functions that you will probably use during workshop 2. See Arduino's Language Reference for further details.

2.1 pinMode()

Description:

Configures the specified pin to behave either as an input or an output.

Syntax:

```
pinMode(pin, mode);
```

Parameters:

`pin` : the Arduino pin number to set the mode of.
`mode` : `INPUT` , `INPUT_PULLUP` or `OUTPUT` .

Returns:

N/A

Explainations of the modes:

When pins being configured as `INPUT` , it is said to be in a **high-impedance state**, with input impedance of approximately $100M\Omega$.

When pins being configured as `INPUT_PULLUP` , a built in pull-up resistor is used (about $20k\Omega$, value depend on board type). Avoid using pin 13 because pin 13 controls the built-in LED.

When pins being configured as `OUTPUT` , it is said to be in a low-impedance state. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40mA of current to other devices/circuits. If more current is needed, consider using an external power supply controlled via transistors or op-amps to avoid damage to the Arduino board.

2.2 digitalWrite()

Description:

Write `HIGH` or `LOW` value to a digital pin.
For Arduino UNO, `HIGH` corresponds to 5V, and `LOW` coressponds to 0V.

If pin is configured as `INPUT` , this will enable(`HIGH`) or disable(`LOW`) the internal pull-up resistor. However, it is advised to use the `INPUT_PULLUP` instead.

If `pinMode` is not previously configured, writing `HIGH` will activated the pull-up resistor and limit the current.

Note: For Arduino UNO, analog pins can also act as digital pins.

Syntax:

```
digitalWrite(pin, value)
```

Parameters:

`pin` : Arduino pin number.
`value` : `HIGH` or `LOW`.

Returns:

N/A

2.3 analogWrite()

Description:

Write an analog value to a PWM pin using pulse width modulation (PWM). For Arduino UNO, PWM frequency is 490Hz (980Hz for pin 5 and 6).

Note: no need to call `pinMode()` before `analogWrite()`

Syntax:

```
analogWrite(pin, value)
```

Parameters:

`pin` : the Arduino PWM pin to write to.
`value` : duty cycle: between 0 (always out, output = 0V) and 255 (always on, output = 5V for UNO). Only accept `int` type.

Returns:

N/A

2.4 digitalRead()

Description:

Read value from a specific digital pin, either `HIGH` or `LOW`.

Note: For Arduino UNO, analog pins can be used as digital pins.

Syntax:

```
digitalRead(pin)
```

Parameters:

`pin` : the Arduino pin number you want to read.

Returns:

`HIGH` or `LOW`.

2.5 analogRead()

Description:

Read value from a specific analog pin. For UNO, the 10-bit ADC will divide 0V - 5V into integer values from 0 to 1023. Resolution is $5/1024 = 4.9$mv/unit.

For Arduino UNO, maximum sample rate is 10kHz.

Syntax:

```
analogRead(pin)
```

Parameters:

`pin` : the name of the analog input pin to read from.

Returns:

The analog reading on the pin. Ranging from 0 - 1023 for Arduino UNO.

2.6 delay()

Description:

Pauses the program for the amount of time (in milliseconds, integer) specified as parameter.

Warning: `delay()` will pause all the things in the program except those using *interrupt*. Serial transmission received via RX pin, PWM values and pin states are maintained. Interrupts will still work as they should. Delays longer than 10's of milliseconds should be avoided and use the `millis()` method instead.

Syntax:

```
delay(ms)
```

Parameters:

`ms` : the number of milliseconds to pause. Allowed data types: `unsigned long`

Returns:

N/A

2.7 delayMicroseconds()

Description:

Pauses the program for the amount of time (in microseconds, integer) specified by the parameter.

For best accurancy, delay time should within $3\mu s$ to $16383\mu s$.

Syntax:

```
delayMicroseconds(us)
```

Parameters:

`us` : the number of microseconds to pause. Allowed data types: `unsigned int`

Returns:

N/A

2.8 millis()

Description:

Returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Note: Use `unsigned long` variables to avoid errors in data type conversion. For accurate timing over short intervals, consider using `micros()` .

Syntax:

```
millis()
```

Parameters:

N/A

Returns:

Number of milliseconds passed since the program started. Data type: `unsigned long` .

2.9 micros()

Description:

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes.

Syntax:

```
micros()
```

Parameters:

N/A

Returns:

Returns the number of microseconds since the Arduino board began running the current program. Data type: `unsigned long` .

2.10 Serial Communication

See Serial Communication for more details.

3 References

W3School, *C++ Tutorial.* https://www.w3schools.com/cpp/default.asp (Accessed Nov. 16th, 2023)
Arduino, *Language Reference.* https://www.arduino.cc/reference/en/ (Accessed Nov. 16th, 2023)